PARALLEL IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK INTEGRATED FEATURE AND ARCHITECTURE SELECTION ALGORITHM

THESIS

CRAIG WILLIAM RIZZO
CAPTAIN, USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/98M-20

PARALLEL IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK INTEGRATED FEATURE AND ARCHITECTURE SELECTION ALGORITHM

THESIS

CRAIG WILLIAM RIZZO
CAPTAIN, USAF

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

AFIT/GOR/ENS/98-20

PARALLEL IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK INTEGRATED FEATURE AND ARCHITECTURE SELECTION ALGORITHM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for

the Degree of Master of Science in Operations Research

Craig W. Rizzo, B.S., Operations Research

Captain, USAF

March, 1998

AFIT/GOR/ENS/98-20

PARALLEL IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK INTEGRATED FEATURE AND ARCHITECTURE SELECTION ALGORITHM

Craig W. Rizzo, B.S., Operations Research
Captain, USAF

Approved:

Advisor — 13 MAR 98 date

13 MAR 98 date

Acknowledgments

I would like to thank my friends and family for their support and humor throughout this affair. The golf was good.

Table of Contents

List of Figures

List of Tables

Table Page

Abstract

The selection of salient features and an appropriate hidden layer architecture contributes significantly to the performance of a neural network. A number of metrics and methodologies exist for estimating these parameters. This research builds on recent efforts to integrate feature and architecture selection for the multi-layer perceptron. In the first stage of work a current algorithm is developed in a parallel environment, significantly improving its efficiency and utility. In the second stage, improvements to the algorithm are proposed. With regards to feature selection, a common random number (CRN) addition is presented. Two new methods of architecture selection are examined, including an information criterion and a signal-to-noise based procedure. These methodologies are shown to improve algorithm performance.

PARALLEL IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK INTEGRATED FEATURE AND ARCHITECTURE SELECTION ALGORITHM

I. Introduction

This research contributes to the advancement of feature and architecture selection for feedforward neural networks. Feature selection involves determining a salient set of input features, while architecture selection involves determining an appropriate, or *efficient*, number of hidden layer nodes. This thesis is focused on a particular type of neural network, the multi-layered perceptron (MLP), as applied to classification problems.

The thesis effort has two phases. The first phase involves the development of an "Integrated Feature and Architecture Selection Algorithm" [1] in a parallel computing environment. This will be referred to as the Steppe algorithm for the remainder of this document. The goal of the Steppe algorithm is to determine an optimal feature set an number of hidden layer nodes for a given application. The current algorithm is very computationally demanding due to the number of networks that are examined and their relative training times. As such, a goal of this thesis is significantly reduce algorithm run-time by developing a parallel implementation. A FORTRAN 77 version of the Steppe algorithm was available as a starting point. While portions of this code were reused, a majority of the algorithm was re-coded to accommodate the parallel environment. In the second phase of research, architecture and feature selection decision methodologies are examined for potential improvement.

The remainder of this section addresses the background and objective of the research as well as the scope of this effort.

Background

Purpose. The goal of the integrated feature and architecture selection algorithm is to formalize the process of selecting an MLP structure and input feature set for a given application. The purpose of this work is to enhance the algorithm by utilizing multiple processors in a parallel computing environment, and to modify specific aspects of the Steppe algorithm with the goal of increasing performance and efficiency.

Objectives. The primary objective of this work is to produce a parallel implementation of an integrated feature and architecture selection algorithm suitable for handling large applications. A second objective is to improve the architecture and feature selection methodologies.

Problem Statement

Evolve the Steppe algorithm into an improved MLP feature and architecture selection algorithm capable of executing in a parallel computing environment. Examine architecture and feature selection methodologies for improvement.

Scope

This research focuses on the advancement of a feature and architecture selection algorithm. It is intended to provide a tool for pinpointing an appropriate set of input features and number of hidden nodes for MLP classification problems. Parallel computing is introduced to reduce algorithm run-time, providing the user with significant

time savings. Improvements are proposed with regards to specific decision methodologies within the algorithm. The final product is an analytic tool which was developed within the constraints of a local parallel computing facility.

Overview

The remainder of this document proceeds as follows. Chapter II reviews background information regarding the MLP, feature and architecture selection, the current feature and architecture selection algorithm and parallel programming. Chapter III contains the method of parallel implementation. Chapter IV first examines an application of the original algorithm developed in parallel, highlighting specific decision methodologies which could be improved. Proposed modifications are then presented and applied to the same data set as the original algorithm mentioned above. Finally, a modified algorithm is applied to a satellite imaging problem. Chapter V presents conclusions drawn from the research and recommendations for continued work.

II. Background

Introduction

This section provides background information on the major topics impacting this research. An introduction to the multi-layer perceptron is presented first, since it represents the foundation of the integrated feature and architecture selection algorithm. Next is a discussion of current techniques used for feature and architecture selection, followed by an overview of the Steppe algorithm. The final section is an introduction to the major concepts of parallel programming.

Multilayer Perceptron

The multi-layer perceptron is often applied to pattern recognition problems where the user is trying to discriminate between two or more classes of data. A typical MLP consists of an input layer, a hidden layer and an output layer (see Figure 1 below). Multiple hidden layers are possible, but this application will utilize only one. The input layer consists of feature input data, with the number of input nodes matching input dimensionality. Input data can be used in raw form or can be transformed. Typically, input data is normalized to eliminate the impact of relative magnitude differences *across* input features [2].

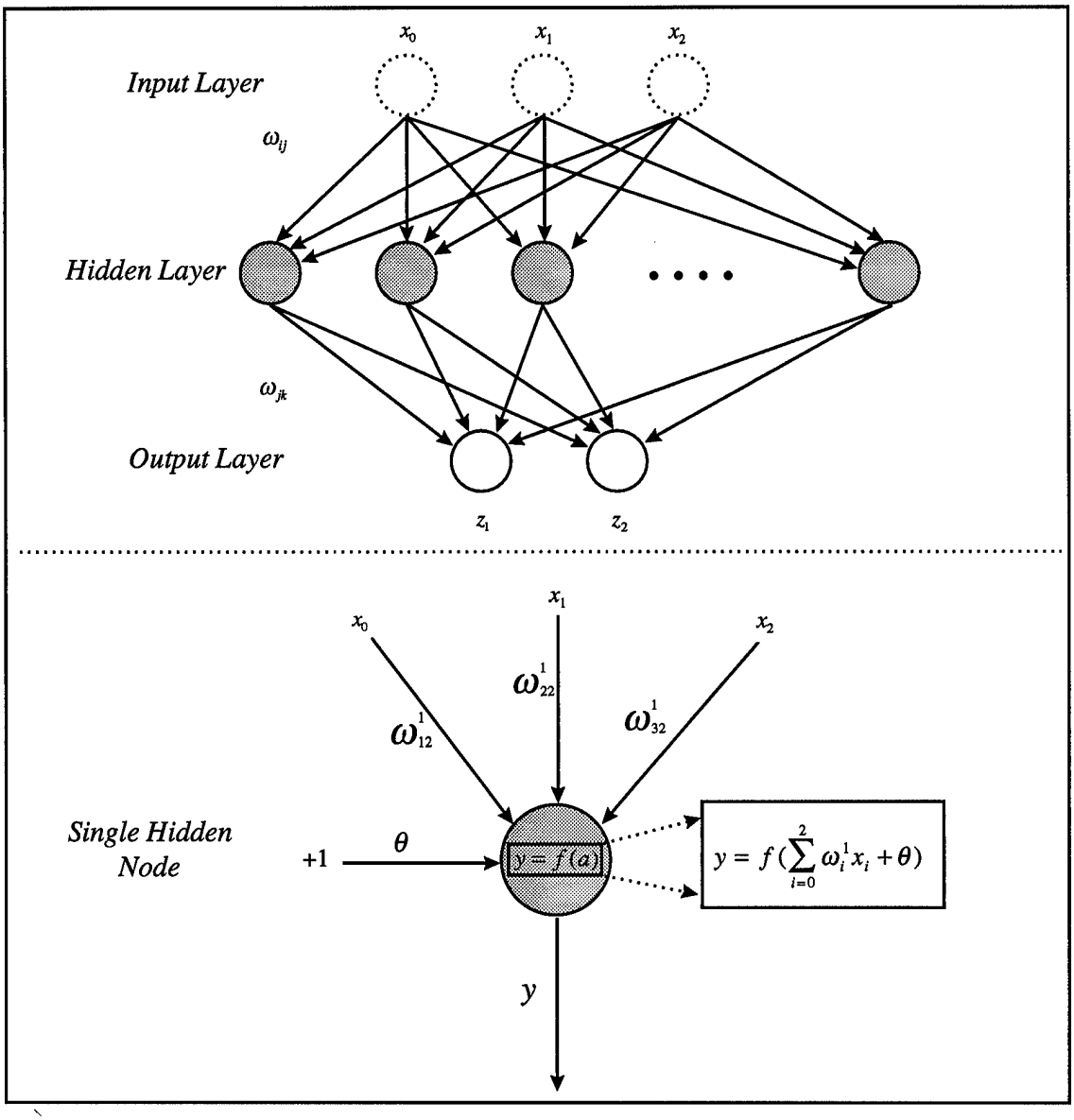


Figure1. Multi-layer Perceptron

The hidden layer nodes, combined with layer one (hidden layer) and layer two (output layer) weights, represent the learning mechanism of the network. As training data is introduced to the network, these weights are continually adjusted to minimize the error between output layer target values and actual output layer values. Typically the target values are binary in nature. For example, a two class problem could define the target

values as [0,1] for Class 1 data and [1,0] for Class 2 data. When defined in this manner, the actual output layer values estimate the class dependent posterior probabilities [3].

The amount of training required for an MLP depends on the complexity of the input space and the number of hidden layer nodes. While fewer hidden nodes minimizes training times, it may also correspond to poor classification accuracy. Too many nodes typically results in long training times and may cause a *memorization* of training data. When new data is subsequently introduced, the network has a poor ability to generalize. Therefore, an optimal number of hidden layer nodes is desired. The rule of thumb is to use the minimum number of nodes required to provide acceptable classification accuracy. This is highly dependent on the application and is typically determined through a trial and error process. More sophisticated methods have been proposed, such as an evolutionary programming [4] an iterative construction algorithm [5]. These methods are discussed later in this chapter.

Each hidden layer node applies a nonlinear transformation to the incoming weights. This transformation, referred to as an activation function, is "analogous to the sigmoid relationship between the excitation and frequency of firing observed in biological neurons" [6]. Commonly used activation functions are the hard limiter and sigmoid. This research will utilize the sigmoid transformation, defined as

$$f(a) = \frac{1}{1+e^{-a}},$$

where a is defined as

$$a = \sum_{i=0}^{N-1} \omega_i x_i + \theta,$$

N is number if input features, x_i is the value of input feature i, ω_i is the weight from the ith input node and θ is a bias term.

When a network trains, it adjusts the weights into and out of the hidden layer. Training can be accomplished with a number of different training algorithms, the most popular of which is *backpropagation.* This methodology minimizes the network sum of squared error (SSE). Output nodes are assigned target values, depending on known class membership of input exemplars. If there are two classes of data, one approach is to use two output nodes, assigning target values as follows:

Class Membership	Target Value, Output Node 1	Target Value, Output Node 2
I	1	0
II	0	1

In this case, output node values represent estimates of class conditional posterior probabilities. The class membership decision is based on a maximum pick of these probabilities. In backpropagation training, the output node value for each data exemplar is compared to its target value (either 0 or 1), and the difference used, combined with a gradient factor, to update hidden layer and output layer weights. The goal is to drive the value of each output node closer to its target value. A more thorough discussion of backpropagation can be found in [3].

Feature Selection

The goal of feature selection is to reduce the input dimensionality (number of features) without significantly reducing discriminative power. Many methods of feature

selection exist, most falling into two categories: feature saliency measures and feature screening. This section introduces both methods, while a more in-depth discussion can be found in [9].

Feature Saliency Measures. A salient feature is one that contributes useful information to the discrimination of multi-class data. Feature saliency measures do not eliminate features directly, but do provide a means of ranking them. The user decides the number features to retain. This section gives a brief description of some well known saliency metrics.

Le Cun and others propose a saliency metric which uses a Taylor series approximation of the network's error [7]. For a trained network, the saliency metric for feature i uses the second derivative of network output error with respect to the vector of associated feature input weights. The saliency metric is calculated for each input exemplar and averaged over the entire training set for a final value.

Mozer and Smolensky suggest that the relevance of a feature be measured as a function, ρ_i, of how well the network performs with the unit versus how well it performs without the unit [8]. In equation form,

$$\rho_i = \varepsilon_{\text{without unit } i} - \varepsilon_{\text{with unit } i},$$

where ε is a function of the error derivative with respect to a relevance factor coefficient of input features.

Ruck describes a saliency metric built from the partial derivatives of network outputs, z_k, with respect to feature inputs, x_i, using a trained network [9]. This metric captures the sum of the partial derivatives of the network outputs with respect to the

entire M-dimensional feature input space, R^M. The Ruck saliency metric for feature i is defined as

$$\bar{A}_i = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{r=1}^{R}\sum_{k=1}^{K}\left|\frac{\partial z_k}{\partial x_i}(x_{m(r)}^p, w)\right|$$

where P is the number of training vectors x, M is the number of features, R is the number of uniformly spaced points covering the range of each input feature, K is the number of output classes and $x_{m(r)}^p$ is the vector at which the partial derivative is evaluated. Since the partial derivative of the outputs is dependent on the weights and final weight values are affected by initial values, we typically calculate the Ruck saliency over a number of independent networks trained with different data orderings and initial weight values. Ruck Saliency for feature i is then calculated as an average,

$$\Lambda_i = \frac{1}{N}\sum_{n=1}^{N}\bar{A}_i^n$$

where N is the number of independent networks.

Tarr proposes a weight based saliency metric which does not require the calculation of derivatives [10]. There are three variates of the Tarr metric; the two norm variate is shown here:

$$\tau_i = \sum_{j=1}^{J}(\omega_{ij}^1)^2$$

where J is the number of hidden nodes and ω_{ij}^1 is the layer one weight between input node i and hidden node j. The weights must be taken from a trained network and input features normalized to unit length. The underlying premise is that weights emanating from important input nodes will assume greater magnitudes during training.

Feature Screening Measures. Screening measures are more direct methods of reducing the feature space. Two methods, saliency-based and weight-based screening are described here. Regression screening is also mentioned, since it incorporates the type of likelihood ratio testing applied in the integrated feature and architecture selection algorithm.

Saliency Screening. Steppe and Bauer propose a statistical approach for feature screening which compares the average saliency of candidate features to the average saliency of random noise [11]. The test uses a *known data approximation* to the Ruck saliency measure, calculated as

$$\bar{A}_i^{data} = \sum_{p=1}^{P} \sum_{k=1}^{K} \left| \frac{\partial z_k}{\partial x_i}(x^p, \vec{w}) \right|$$

where P is the number of training exemplars, K the number of output nodes and $\bar{A}_i^{data}$ represents the average network saliency over the P known data points. Random noise from a Uniform (0,1) distribution is injected into the network and the above saliency is calculated for all features including the noise. A Bonferroni procedure is then used to conduct the following test:

$$\text{Null Hypothesis } \mathbf{H_o}: \quad \mu_{D_i} = 0$$

$$\text{Alternate Hypothesis : } \mathbf{H_a}: \quad \mu_{D_i} > 0,$$

where μ_{D_i} is the difference between the *i*th feature's mean saliency and the noise feature's mean saliency. If we reject the null hypothesis for a given feature, the feature is retained. If we fail to reject, the feature is removed.

Weight-based Screening. White suggests a method of feature screening based on the assumption of multivariate normality of the network weights as $\hat{\mathbf{w}}$ approaches $\mathbf{w}^*$, the optimal weight vector [12]. The premise of White's screening procedure is that the vector of feature weights is not statistically different from zero for irrelevant features. In short, White's approach uses the weight vector from a trained network with no redundant feature inputs and no redundant nodes to test hypotheses that selected weights are not statistically different from zero, and are thus irrelevant. If the weights associated with a given feature are deemed irrelevant, the feature is removed. The problem with White's procedure is its sensitivity to other than optimal conditions. In other words, if there are more (or less) than the optimal number of hidden nodes, or if there are redundant features, the weights may not be multivariate normally distributed. Since multivariate normality is a requirement, the procedure becomes invalid. The procedure remains adequate for a network with optimal structure, but in that case feature screening is meaningless.

Regression Screening. The purpose of this section is to introduce a criterion used in univariate nonlinear regression model selection. This procedure is similar to a methodology used in the original integrated feature and architecture selection algorithm for removing input features. Univariate nonlinear regression is similar in form to a multi-layer perceptron [13]. There is a single response variable and a set of *s* input

variables (or features) which may possess redundant or irrelevant information. The criterion of interest is a likelihood test statistic, **L**, which is used to test the null hypothesis $\mathbf{H_0}$: $h(\theta^*) = 0$, where $h(\theta^*)$ is defined to be a subset of k variables from the full set of s variables. The likelihood test statistic is defined as,

$$\mathbf{L} = \frac{\frac{(SSE_{s-k} - SSE_s)}{((n-(s-k))-(n-s))}}{\frac{SSE_s}{(n-s)}}$$

where SSE_{s-k} corresponds to the sum of squared errors associated with fitting the model under the null hypothesis and SSE_s corresponds to the sum of squared errors associated with the full model. Under $\mathbf{H_0}$, **L** is F distributed with k numerator degrees of freedom and n-s denominator degrees of freedom. When **L** exceeds the specified critical point of the F distribution with equal degrees of freedom, $\mathbf{H_0}$ is rejected and no variables are removed from the model [9].

Architecture Selection

Architecture selection is important to maintain an efficient network. Choosing too few hidden nodes may hamper network convergence, while choosing too many (redundant) nodes may diminish the network's ability to generalize new data. For a given application there are typically numerous architectures that provide adequate classification, but some are more efficient than others (i.e. requiring less training time while achieving the same or better results). An ideal network has the minimum number of hidden nodes needed to achieve a specified classification accuracy.

Architecture selection is often viewed as an art form, since there exists no proven method for determining an appropriate number of hidden nodes for all MLP applications. A trial and error approach is typically the method of choice. This can be time consuming, and may not provide acceptable results. Some of the recent attempts to formalize architecture selection include evolutionary algorithms and similarity methods. Both are described here, as well as a method of regression selection.

Evolutionary Algorithms. Sarkar and Yegnanarayana [14] propose a method of evolutionary programming (EP) to determine an optimal number of hidden layer nodes. The EP algorithm minimizes a *mean square error* (MSE) function to obtain an optimal network structure and optimal set of weights. It employs a controlled stochastic search which explores many paths simultaneously. Multiple solutions are generated initially (*parent* networks), and subsequently *mutated* by adding or deleting hidden nodes or by small perturbations of the network weights. These altered networks are called *offspring*. A fitness function which incorporates MSE is then calculated for each parent and offspring network, and a "survival of the fittest" approach is used to retain the best networks. The process continues until the number of offspring generated is greater than some pre-specified constant. At that point, the network with the highest classification rate on a validation set is retained as optimal.

Iterative Construction Algorithm. Rathbun and others suggest a method of determining network structure which iteratively separates all inter-class data pairs [5]. Initially, two classes (out of N total) are arbitrarily chosen for separation. A euclidean distance is calculated for each inter-class data pair and the data pair with minimum

separation is chosen as a starting point. Hidden layer nodes are iteratively added until all data pairs are separated. Redundant nodes are eliminated through a *relaxation* phase. The process continues for all two-class combinations until all training data are classified with 100% accuracy. The danger in this algorithm is the tendency to memorize training patterns, resulting in a poor generalization ability for new data. The authors address this problem by making class neighborhoods as large as possible and by minimizing the number of hidden nodes, thus creating less complex decision boundaries.

Integrated Feature and Architecture Selection Algorithm

As mentioned earlier, the integrated feature and architecture selection algorithm was developed to formalize the process of selecting a set of salient features and an appropriate network structure. This section provides an overview of the algorithm. Further details can be found in [1] or [2].

The major components of the selection algorithm include an initial architecture selection module, a feature screening module, an architecture selection module and a feature selection module (see Figure 2 below). Once an initial architecture is found and noise-like features are removed in feature screening, the algorithm iteratively removes hidden layer nodes and input features with a conditional looping structure. A likelihood ratio statistic incorporating the networks' output sum-of-squared error (SSE) is used as the decision criterion for removing hidden nodes as well as features. If the feature selection module does not remove a feature, the algorithm terminates.

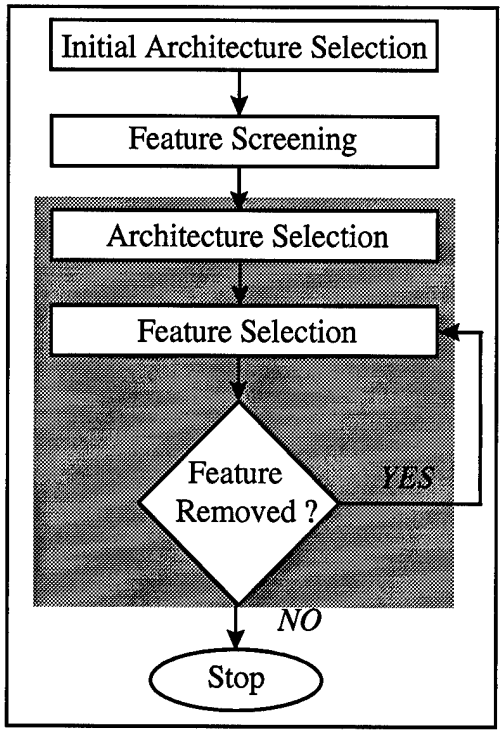


Figure 2. Integrated Feature and Architecture Selection Algorithm.

Further details of the algorithm are given next:

1) Initial Architecture Selection (see Figure 3 below)

   a) Select an upper bound, H, for the number of hidden layer nodes using Cover's approach [16], or a similar method. Call this the full model.
   b) Using all I features, train S replications of the full model with H hidden layer nodes and S replications of a reduced model with H-1 hidden layer nodes.
   c) Calculate the likelihood ratio statistic, L, as

$$L = \frac{\frac{SSE_R - SSE_F}{q}}{\frac{SSE_F}{P - v}}$$

   where SSE_R is the sum-of-squared error for the reduced model, SSE_F is the sum-of-squared error for the full model, q is the number of weight parameters which are constrained to be zero in the reduced model, P is the number of training exemplars, v is the number of weights and L is F-distributed.

d) Test the null hypothesis that the reduced model is equivalent to the full model.
   i) If $L > F_{crit}$ at the given α, reject the reduced model and continue with Step 2.
   ii) f $L \leq F_{crit}$ at the given α, accept the reduced model and eliminate a middle node; H=H-1. Return to step 1a.

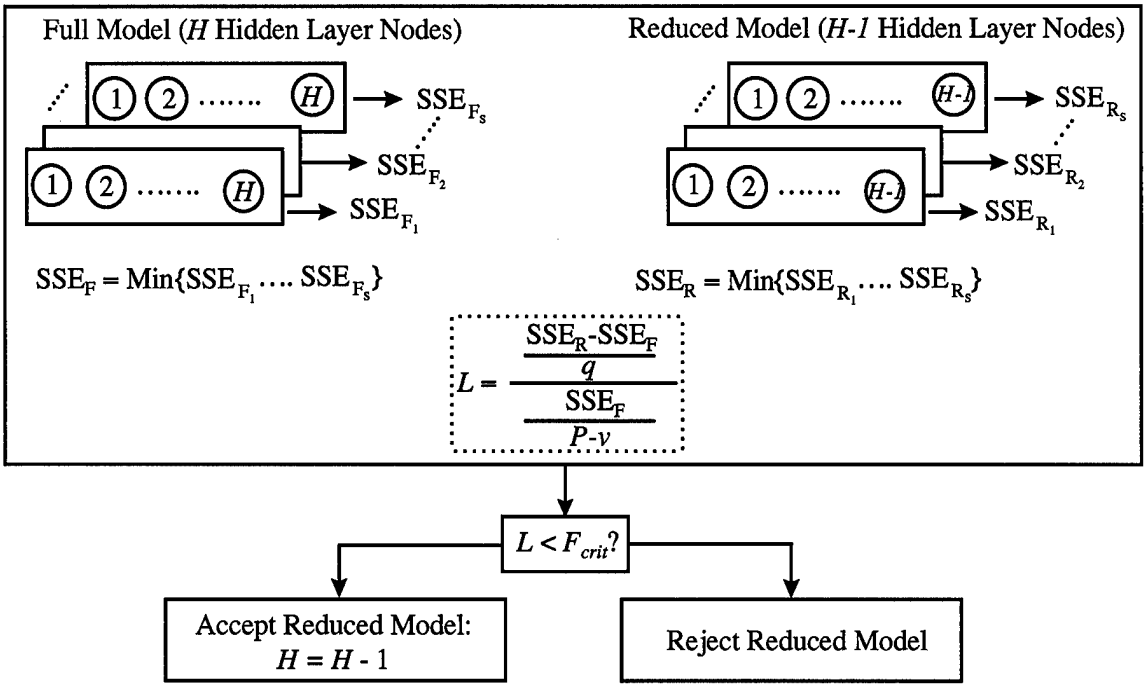


Figure 3. Architecture Selection.

2) Feature Screening

   a) Train S replications of a neural network using the current number of hidden nodes, H, and all I features plus an additional noise feature.
   b) Using the Ruck saliency measure and a Bonferroni test procedure, compare the I feature saliencies to the noise saliency.
   c) Retain all features that are significantly different from noise.

3) Architecture Selection (see Figure 3 above)

   a) Using the current number of hidden nodes, H, and current feature set, train S replications of the full model with H hidden nodes and S replications of a reduced model with H-1 hidden nodes.
   b) Calculate the likelihood ratio statistic, L, as

$$L = \frac{\frac{SSE_R - SSE_F}{q}}{\frac{SSE_F}{P - s}}$$

where SSE_R is the sum-of-squared error for the reduced model, SSE_F is the sum-of-squared error for the full model, q is the number of weight parameters which are constrained to be zero in the reduced model, P is the number of training exemplars, s is the number of weights and L is F-distributed.

d) Test the null hypothesis that the reduced model is equivalent to the full model.
   i) If $L > F_{crit}$ at the given α, reject the reduced model and continue with Step 4.
   ii) If $L \leq F_{crit}$ at the given α, accept the reduced model and eliminate a middle node; H=H-1. Return to step 3a.

4) Feature Selection (see Figure 4 below)

a) Using the number of hidden nodes chosen in Step 3, train S replications of a neural network with one feature removed from the current feature set.
b) Repeat step 4a I times, each time removing a different feature.
c) From these I models, select the model with minimum SSE and call this SSE_R.
d) Test the null hypothesis that the reduced model (I-1 features) is equivalent to the full model (I features) using the same test statistic as Step 3b.
   i) If $L \leq F_{crit}$, accept the reduced model and eliminate the feature that was used to select SSE_R; I=I-1. Return to Step 3.
   ii) If $L > F_{crit}$, reject the reduced model and continue to Step 5.

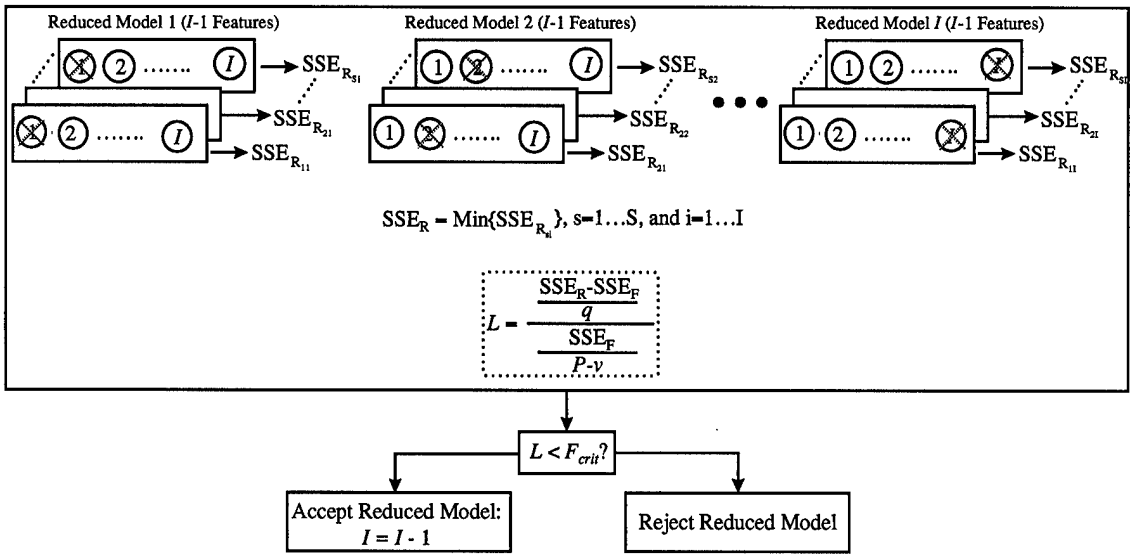


Figure 4. Feature Selection.

5) Stop. The reduced model has been rejected, therefore no additional features or hidden nodes are removed. Final number of features: *I*. Final number of hidden layer nodes: *H*.

Parallel Programming

Parallel computing is the utilization of multiple processors to solve a problem. It has evolved from the continually increasing need for greater computing power. The computational limits of computing in general have increased from hundreds of arithmetic operations per second at the dawn of the computer era to our present capability of billions of arithmetic operations per second. Parallel computing offers improvements into the trillions, and perhaps even quadrillions, of operations per second [15].

Hardware. There are many varieties of parallel computing hardware, but most fall within two main categories, SIMD and MIMD.

Single-instruction multiple data (SIMD). With this configuration, a single primary CPU maintains program control and continually broadcasts instructions to

subordinate arithmetic logic units (ALU's). ALU's are not capable of executing programs independently.

Multiple-instruction multiple data (MIMD). The key difference between SIMD and MIMD systems is the ability of subordinate units to act autonomously. In a MIMD system subordinates CPUs are capable of executing independently of the primary CPU, each possessing its own ALU. With this type of system, subordinate units may all receive different segments of a program and upon completion report their results to the primary CPU. MIMD systems are classified as either shared-memory or distributed-memory. As the name implies, with a shared memory design all CPU's have access to all memory units. This is illustrated below:

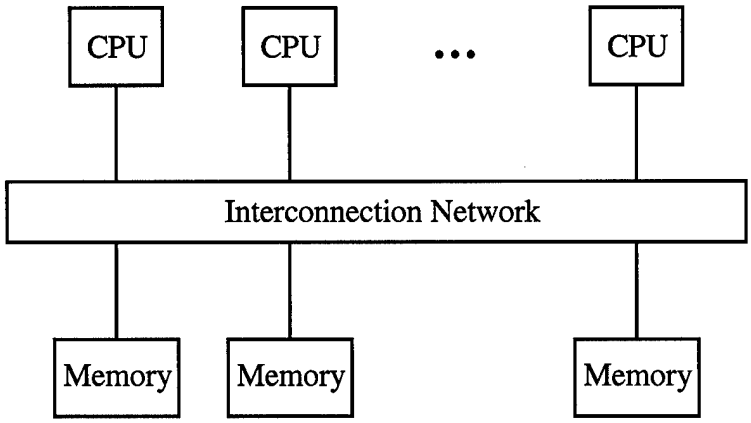


Figure 5 - Shared-memory system.

The primary limitation of a shared memory approach is the hardware necessary to allow all CPU's uniform access to the memory, which can be very costly. Distributed-memory systems, on the other hand, allocate private memory for each CPU. A generic distributed-memory system is shown below:

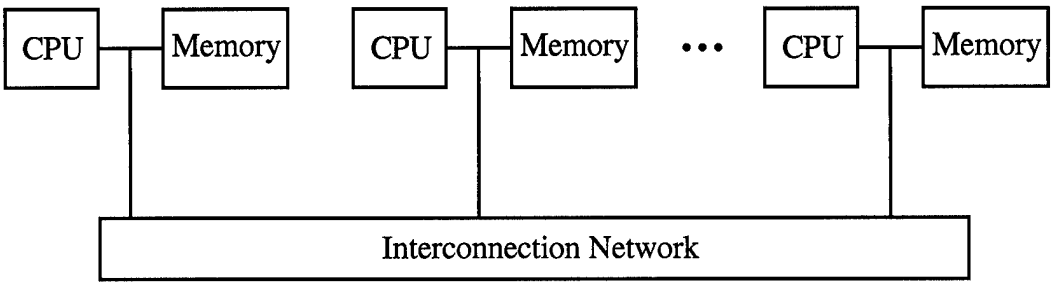


Figure 6 - Distributed-memory system.

The primary drawback in a distributed-memory approach is the relative difficulty of software programming.

Software. In parallel computing, a *process* is a fundamental building block. It is defined as an instance of a program or subprogram, operating independently on a processor. Shared memory programming and distributed memory programming coordinate inter-process interaction differently. With shared memory programming, a process can be created at the beginning of program execution or during program execution. Distributed memory programming, on the other hand, must initiate all processes at the beginning of program execution. The most common form of distributed memory programming for MIMD systems is *message passing*, where processes coordinate activities by sending and receiving messages.

Message Passing Interface (MPI). MPI is the most widely used method of parallel programming today. It is not a programming language, but rather a library of subprograms that can be implemented by C or FORTRAN routines. MPI is both portable and efficient, making it widely accepted throughout the parallel community. It was developed by an international forum consisting of representatives from industry,

academia and various government facilities. Further details of MPI are discussed in Chapter 3.

III. Methodology

Introduction

This section presents the parallel version of the Steppe algorithm. It details the parallel environment used for development as well as available options for parallel programming.

Parallel Environment

The Major Shared Resource Center (MSRC) is a supercomputing facility located on Wright-Patterson AFB and is available to Department of Defense agencies. Among its resources is a Cray Origin2000 (O2000) supercomputer with 224 Central Processing Units (CPUs). Each CPU is a 195 MHz MIPS R10000 with a 32 kilobyte primary data cache, a 32 kilobyte primary instruction cache and a 4 megabyte secondary cache. The O2000 is configured as seven 32-processor systems with 16 gigabytes of memory, 70 gigabytes of workspace and a High-Performance Parallel Interface (HiPPI). It has a symmetric multiprocessor (SMP) architecture, allowing all CPUs full access to the total system memory.

Programming Options

Script vs. MPI. There are two realistic options for implementing the integrated feature and architecture selection algorithm in parallel. The first option is to apply a UNIX shell script. The script would initiate multiple replications of the code and be responsible for directing program output. The primary limitation lies with the script's

inability to alter variables as the algorithm progressed. A shell script is *external* to the actual code and does not permit variable manipulation during program execution. A more suitable option is MPI, which is integrated directly into the code. MPI would allow the program logic to be handled within the code instead of through an external interface such as a shell script. For this reason, MPI is chosen as the development tool.

Algorithm Options. There were three realistic options for applying a parallel architecture to the algorithm.

1. Run individual neural networks in parallel - There is a significant amount of literature regarding the parallelisation of an individual neural network (specifically the multi-layer perceptron). An MLP lends itself to parallel implementation due to the similar calculations made for each of the networks hidden nodes. With this approach, significant resource savings would be realized with very large networks (i.e. many input features and many hidden nodes).
2. Use multiple processors to run *replications* in parallel - During every stage of the algorithm multiple replications are conducted for each potential network structure. For example, during architecture selection there may be an upper bound of 20 hidden nodes, thus 20 models are examined. For each model, 30 replications are run and the *best* (minimum test-set SSE) replication retained for hypothesis testing. This option distributes each of the 30 replications to separate processors.
3. Use multiple processors to run *models* in parallel - In the example above for architecture selection an upper bound of 20 hidden nodes (and thus 20 models)

was stated, with 30 replications required for each model. Instead of running replications in parallel, this option runs *models* in parallel. This example would require 20 processors for the 20 models.

From the above options number 3 is chosen for the following reason. Although in the example given here option 2 would seem optimal, it would be sub-optimal if the upper bound on hidden layer nodes exceeds 30. For many real-world problems this is indeed the case. Thus option 3 was chosen as the method of parallel implementation.

Parallel Implementation

The parallel version of the integrated feature and architecture selection algorithm is written in FORTRAN 77 and integrated with Message Passing Interface (MPI). The power of MPI lies in its ability to designate one processor as a "Master" and the rest as "Slaves" (see Figure 7 below). The master processor controls program logic, and slaves carry out specific calculations dictated by the master.

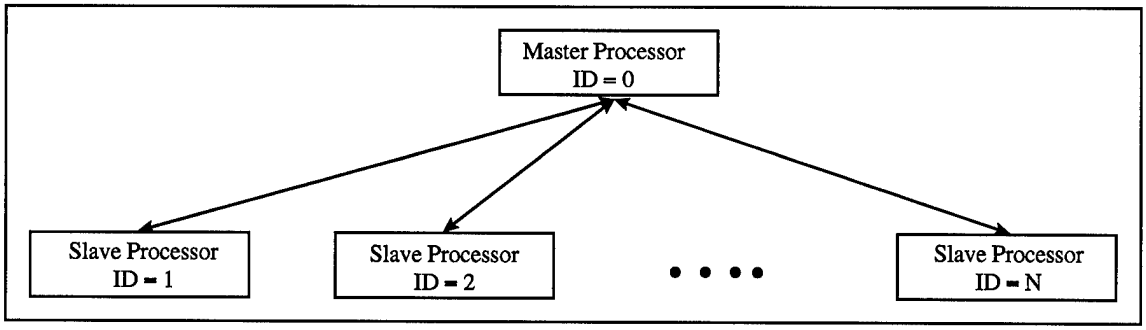


Figure 7. Master/Slave Relationship.

The Master is assigned an identification number of '0' and slaves are labeled from '1' to the number of slave processors *N*. All processors, master and slaves, receive the same code. During program execution each processor executes the appropriate code

dictated by a conditional *IF* statement. Typically, all slaves have the same operating instructions, as demonstrated below:

```
if ID = MASTER then
  do ...
elseif ID = SLAVE then
  do ...
end
```

When necessary, information is passed from the Master, using MPI_SEND or MPI_BCAST (for 'broadcast') statements, to one, some or all of the slaves, and computation results are passed back to the Master. Messages passed from slaves to the master are identified by *tag* numbers, equal to the slave ID's. Example code illustrating the master/slave concept and simple MPI_BCAST, MPI_SEND and MPI_RECV statements is given below:

```
...
if ID = 0 {MASTER} then
   call MPI_BCAST (varA, 1, MPI_INTEGER, MPI_COMM_WORLD, IERR)
   do I = 1,NUMSLAVES
      call MPI_RECV (ANS, 1, MPI_INTEGER, MPI_ANY_SOURCE,
                     MPI_ANY_TAG)
      SUM = SUM + ANS
   enddo
elseif ID ≠ 0 {SLAVE} then
   call MPI_BCAST (varA, 1, MPI_INTEGER)
   ANS = varA + ID
   call MPI_SEND (ANS, 1, MPI_INTEGER, MASTER, ID)
endif
...
```

In this example, the master broadcasts *varA* to all slaves, the slaves add their *ID* number to *varA*, and then return *ANS* to the master. Each time a slave returns a value for ANS, the master adds it to *SUM*. Note that the master must utilize a looping mechanism to account for all slaves that are returning a value of *ANS*. The master/slave paradigm is

used throughout the feature and architecture selection algorithm. An overview of the parallel implementation is given in Figure 8.

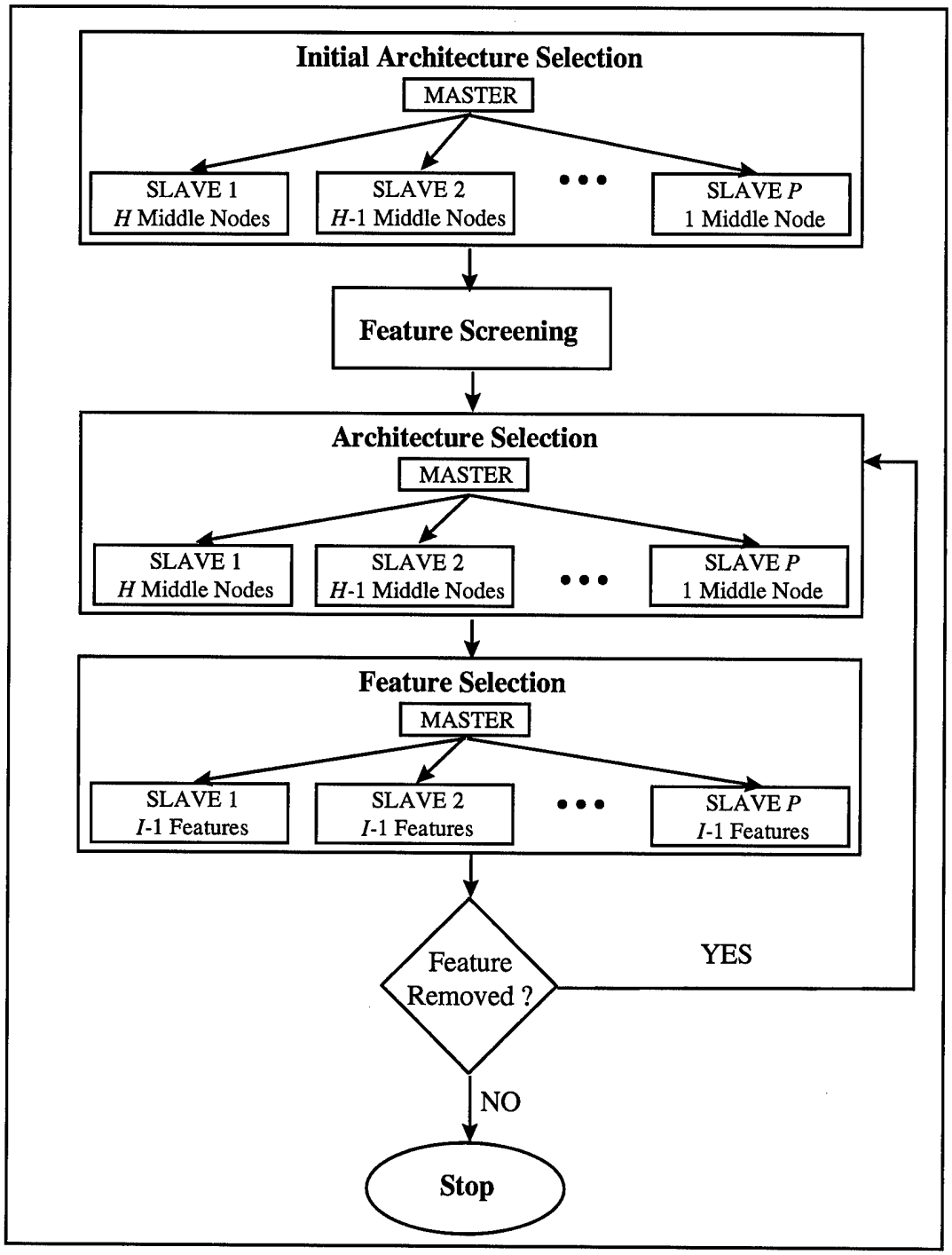


Figure 8. Parallel Implementation of feature and architecture selection Algorithm.

Details of the parallel implementation:

1) Initialize - this includes MPI statements to initialize Master and Slave processors as well as passing slaves required start-up information.

2) Initial Architecture Selection (see Figure 9 below):

   a) Select an upper bound, H, for the number of hidden layer nodes using Cover's method [16] or a similar approach. Use H hidden nodes for the full model, label as M_H.
   b) Using all I features, create $H\text{-}1$ reduced models by removing hidden layer nodes. Reduced model $M_{H\text{-}1}$, will have $H\text{-}1$ hidden nodes, $M_{H\text{-}2}$ will have $H\text{-}2$ hidden nodes, etc...down to M_1, having one hidden layer node.
   c) For each model, $M_{(\bullet)}$, use a slave processor to calculate S replications.
   d) Choose the replication with the minimum SSE; pass this value back to the Master.
   e) Label the SSE from the full model (H nodes) as SSE_H, and the reduced model ($H\text{-}1$ nodes) as $SSE_{H\text{-}1}$. Calculate a likelihood ratio statistic, L, as

$$L = \frac{\frac{SSE_R - SSE_F}{q}}{\frac{SSE_F}{P - v}}$$

   where SSE_R is the sum-of-squared error for the reduced model, SSE_F is the sum-of-squared error for the full model, q is the number of weight parameters which are constrained to be zero in the reduced model, P is the number of training exemplars, v is the number of weights and L is F-distributed.

   f) Test null hypothesis that the reduced model is equivalent to the full model.
      i) If $L \leq F_{crit}$, accept reduced model and eliminate a middle node; $H = H - 1$. Return to step 2d.
      ii) If $L > F_{crit}$, reject reduced model, continue with Step 3.

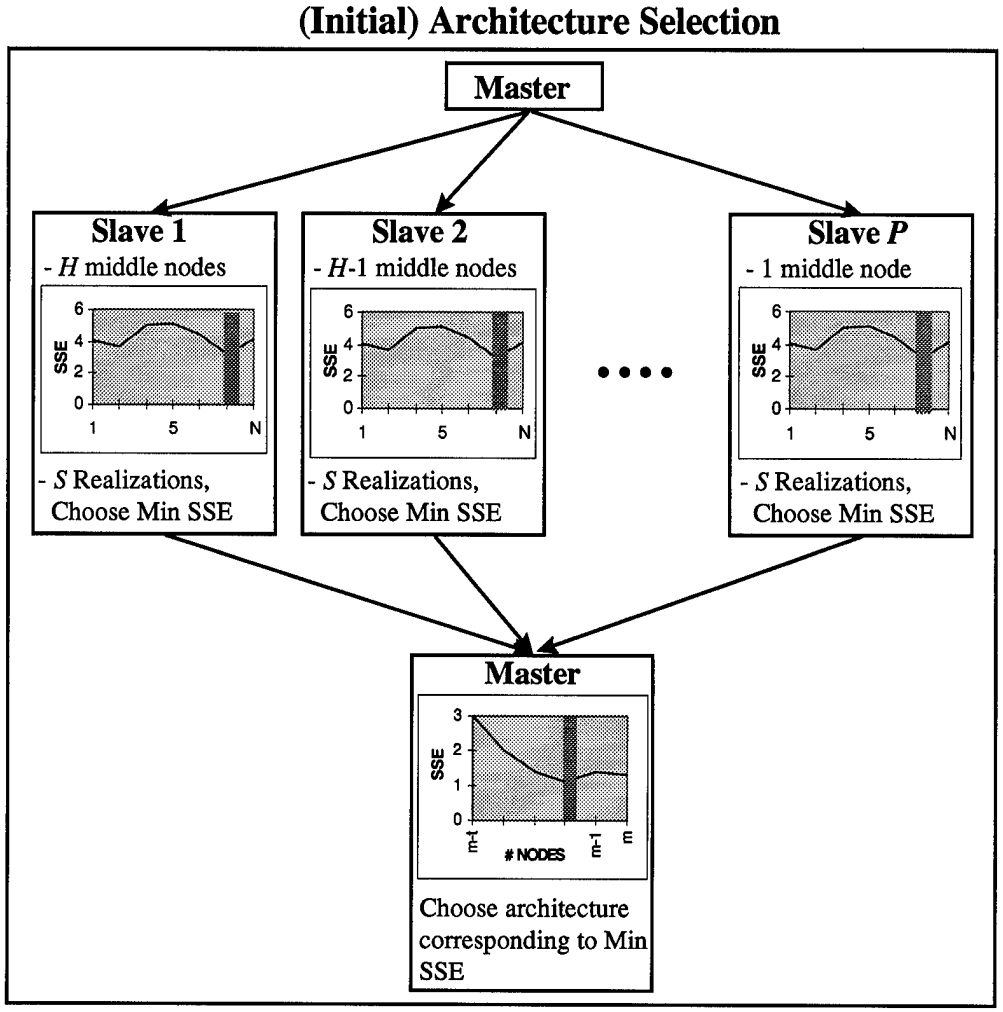


Figure 9. Architecture Selection for Parallel Implementation.

3) Feature Screening

a) Train S replications of a neural network using the number of hidden nodes from Step 1 and all I features plus an additional noise feature.
b) Using the Ruck saliency measure and a Bonferroni test procedure, compare the I feature saliencies to the noise saliency.
c) Retain all features that are significantly different from noise.

4) Architecture Selection (Refer to Figure 9)

a) Select an upper bound, H, for the number of hidden layer nodes using Cover's method [16] or a similar approach. Use H hidden nodes for the full model, label as M_H.

b) Using all *I* features, create *H-1* reduced models by removing hidden layer nodes. Reduced model M_{H-1}, will have *H-1* hidden nodes, M_{H-2} will have *H-2* hidden nodes, etc...down to M_1, having one hidden layer node.

c) For each model, $M_{(\bullet)}$, use a slave processor to calculate *S* replications.

d) Choose the replication with the minimum SSE; pass this value back to the Master.

e) Label the SSE from the full model (*H* nodes) as SSE_H, and the reduced model (*H-1* nodes) as SSE_{H-1}. Calculate a likelihood ratio statistic, *L*, as

$$L = \frac{\frac{SSE_R - SSE_F}{q}}{\frac{SSE_F}{P - v}}$$

where SSE_R is the sum-of-squared error for the reduced model, SSE_F is the sum-of-squared error for the full model, *q* is the number of weight parameters which are constrained to be zero in the reduced model, *P* is the number of training exemplars, *v* is the number of weights and *L* is F-distributed.

f) Test null hypothesis that the reduced model is equivalent to the full model.

   i) If $L \leq F_{crit}$, accept reduced model and eliminate a hidden node; $H = H - 1$. Return to step d).

   ii) If $L > F_{crit}$, reject reduced model, continue with Step 5.

5) Feature Selection (see Figure 10 below)

a) Using the number of hidden nodes chosen in Step 4, create *I* models, each with *I*-1 features. Each model will have a different feature removed.

b) Using a slave processor for each model from a), train *S* replications of each model, retaining the replication with minimum SSE; pass this value to the Master.

c) From these *k* models, select the model with the minimum SSE and call this SSE_R.

d) Test the null hypothesis that the reduced model (*k*-1 features) is equivalent to the full model (*k* features) using the same test statistic as Step 4.

   i) If $L \leq F_{crit}$, accept the reduced model and eliminate the feature that was used to select SSE_R ; $I = I - 1$. Return to Step 4.

ii) If $L > F_{crit}$, reject the reduced model and continue to Step 6.

6) Stop.

The reduced model has been rejected, therefore no additional features or hidden nodes are removed. Final number of features: *I*. Final number of hidden layer nodes: *H*.

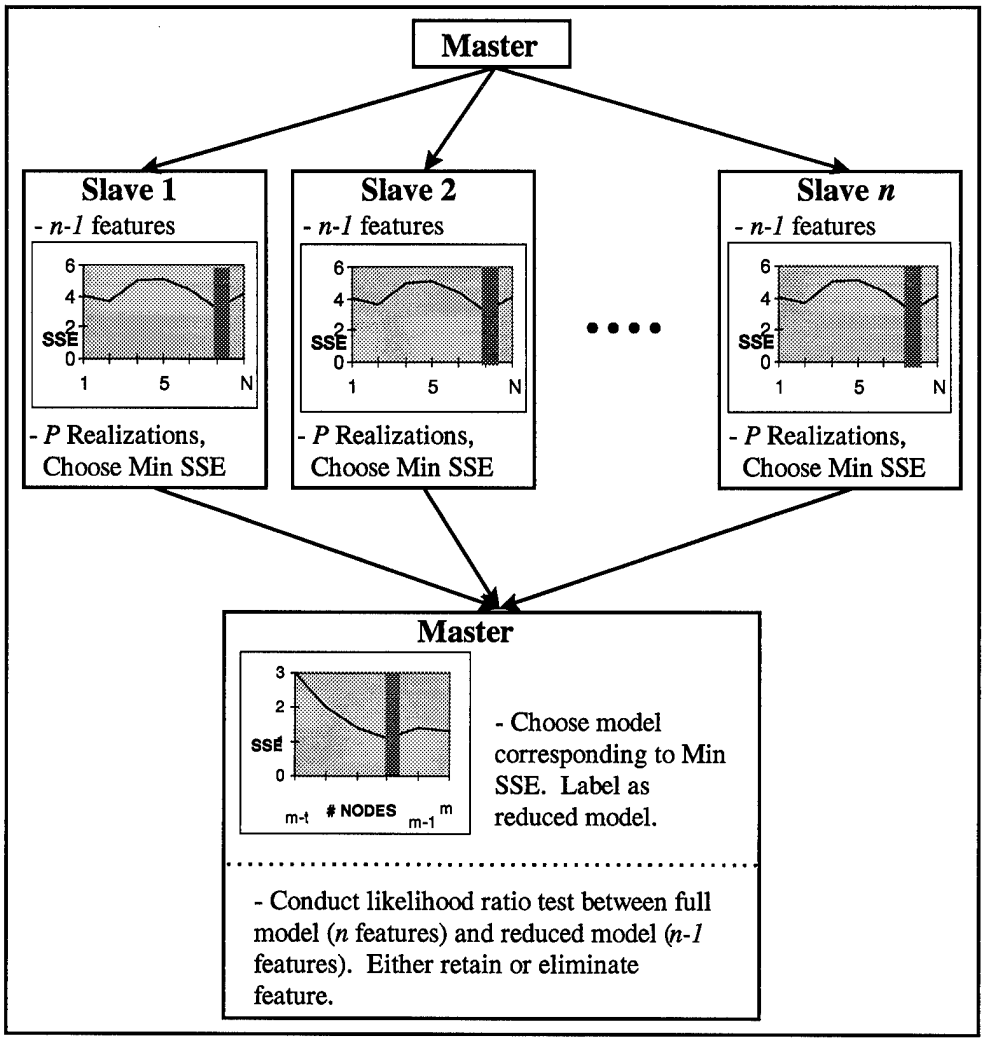


Figure 10. Feature Selection for Parallel Implementation.

Benefits of Parallel Implementation

The benefits of a paralleled integrated features and architecture selection algorithm are twofold. First, due to the sheer number of networks that can be calculated, the algorithm becomes an efficient tool for analyzing MLP properties with respect to weights, training times and different architecture and feature selection methodologies. Second, it provides significant time reductions when searching for an appropriate network structure (features and hidden nodes) for large problems. It is difficult to calculate exact time savings for all applications, but approximations and upper bounds are feasible, as discussed next.

As described in earlier in this chapter, the O2000 is segmented into seven 32-processor systems. Thus only 32 processors may be accessed by any given parallel application. In general, the amount of CPU time saved for a given application depends on the number of *models* which must be calculated during each phase of architecture and feature selection. A time-savings factor (TSF) can be used to approximate time-savings, and is defined as

$$TSF = \frac{\text{Run time on single CPU}}{\text{Run time in parallel}}.$$

An equation for the TSF can be stated as

$$TSF = \frac{N}{a}$$

where N is the number of models and

$$a = \frac{N}{31},$$

rounded *up* to the nearest integer. As an example, consider an application with 30 features and an upper bound of 50 hidden nodes. For initial architecture selection (IAS) there are 50 models that must be examined. With 31 processors (1 used for program logic), the first 31 are calculated in parallel and, upon completion, the remaining 19 are calculated in parallel. The *TSF* for IAS would be 25 (a=50/31 [rounded up] ➔ 2; *TSF* = 50/2 = 25). A plot of *TSF* for up to 124 models is shown below in Figure 11.

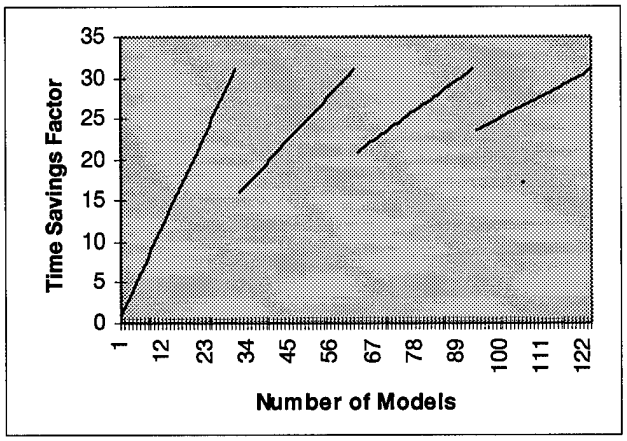


Figure 11. Time Savings Factor for varying numbers of models.

Note the consistent upper bound of 31, the number of parallel processors available currently accessible by a single application on the O2000.

IV. Findings and Analysis

Introduction

This section begins with an application of the Parallel Integrated Feature and Architecture Selection Algorithm. The purpose of this application is twofold. First, it demonstrates the parallel algorithm and steps through each phase using a standard XOR problem with added redundant and noisy features. Second, it highlights a portion of the algorithm where improvements seem appropriate. Sections 3 and.4 present two proposed methods for architecture selection and an improvement to the feature selection procedure. Section 5 includes XOR results of the proposed new methodologies and Section 6 presents results from an *anomalous satellite behavior* application.

Application of Parallel Integrated Feature and Architecture Selection Algorithm

This section applies the original algorithm in its' parallel form to a standard XOR problem

Data. The data for this example is includes 8 features with 500 training and test set exemplars. There are 2 primary features (see Figure 12), 2 redundant features (linear combinations of the primaries) and 4 noisy features. The first 2 noisy features are Unif [-1,1] random noise, while the remaining 2 are the primary features plus/minus some random error distributed as Norm[0, σ^2_ε], where σ^2_ε is small. Table 1 summarizes the data.

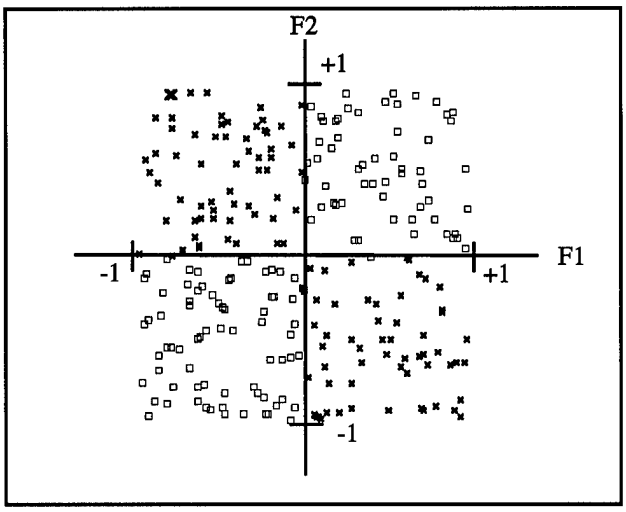


Figure 12. XOR data; Feature 1 (F1) and feature 2 (F2).

Feature	Description	Interval
1	X	[-1,1]
2	Y	[-1,1]
3	$aX + c_1$	[-1-max(c_1), 1+max(c_1)]
4	$bY + c_2$	[-1-max(c_2), 1+max(c_2)]
5	Unif Random Noise	[-1,1]
6	Unif Random Noise	[-1,1]
7	X +/- N[0, σ^2_ε]	[-1-max(σ^2_ε), 1+max(σ^2_ε)]
8	Y +/- N[0, σ^2_ε]	[-1-max(σ^2_ε), 1+max(σ^2_ε)]

Table 1. XOR Data summary.

Application. An upper bound on the number of hidden layer nodes was found using Cover's formula, which states:

$$H < \frac{0.5P - 1}{M + 1}$$

where H is the upper bound, P is the number of exemplars and M is the number of features. With 400 training exemplars and 8 features, the upper bound is 22 hidden nodes.

Recalling the general approach of the algorithm (Figure 13), the next section outlines the procedure and results of the original algorithm when applied to the XOR data described above. Figure 14 provides a general synopsis for each stage while Figures 15 and 16 are graphical representations of the algorithms decision methodologies.

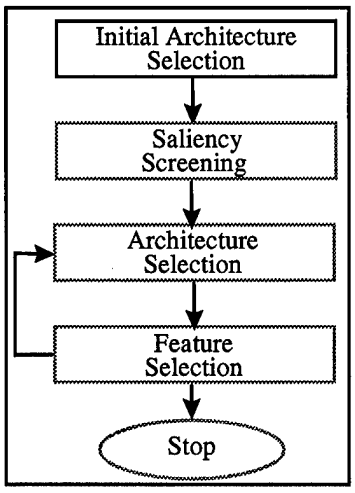


Figure 13. Integrated Feature and Architecture Selection Algorithm.

Initial Architecture Selection (IAS) - We are searching for an appropriate architecture given our complete feature set. Twenty-two networks are examined (referred to as *models*), the first model having one hidden node, the second model two hidden nodes, and so on up to the last model with 22 hidden nodes. The models are run in parallel, and the model with minimum test set sum-of-squared-error (SSE) retained from thirty replications. The purpose of thirty replications of each model is to avoid local minimums. Training data is randomly ordered prior to each replication. A likelihood ratio statistic incorporating training and test set SSE is used to test for model reduction. We eliminate one hidden layer node under the premise that reduced model performs equally as well or better than the full model.

Feature Screening - We conduct Feature Screening once at the beginning of the algorithm. One or more features may be eliminated, depending on their performance relative to an injected noise feature. Recall that the procedure utilizes a Ruck saliency measure as a decision criteria, calculated for all primary features plus the noise feature. Using a standard *t*-statistic and a Bonferroni test approach, we discard any of the 8 primary features not statistically different from noise. In this case 3 of the 8 are eliminated. The procedure typically identifies a majority of the noise features, but is not guaranteed to locate them all. This is likely attributed to the type of noise as well as the presence of other redundant features. Feature Selection is intended to identify the noisy features passed over in Feature Screening.

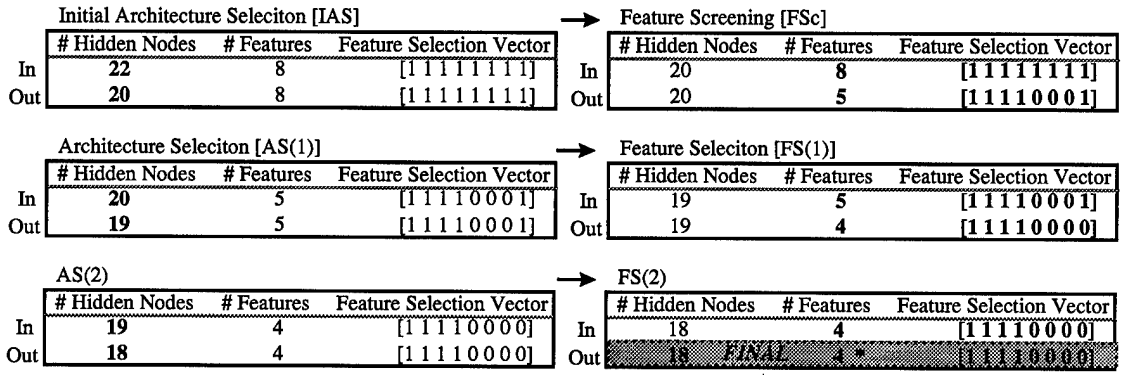


Figure 14. XOR application; original algorithm.

Architecture Selection [AS(1)] - We enter AS(1) with 5 features and 20 hidden nodes. As in IAS, 20 models are run in parallel with 30 replications of each, choosing the replication with minimum test set SSE. Using the likelihood ratio test we once again test to remove hidden nodes. If network performance is remains the same or improves, hidden nodes are removed. In this instance one node is eliminated.

Feature Selection [FS(1)] - Feature selection examines network performance with each of the remaining features removed. For instance, with 5 remaining features there are a total of 5 models, each with a different subset of 4 features. Once again the models are run in parallel and 30 replications of each are conducted, retaining the replication with minimum SSE. From the 5 models we retain the one with minimum SSE, calling this $SSE_{Reduced}$ (SSE_R). A likelihood ratio test is used to compare SSE_R to SSE_F, the full model SSE. If the reduced model performs equally as well *or better than* the full model, the feature corresponding to the chosen model is eliminated. The procedure compares an observed significance, or 'p' value to our given alpha of .05. In this case the p-value exceeds alpha, therefore we fail to reject our null hypothesis that the models are equal, allowing us to eliminate the feature corresponding to the reduced model and return to AS.

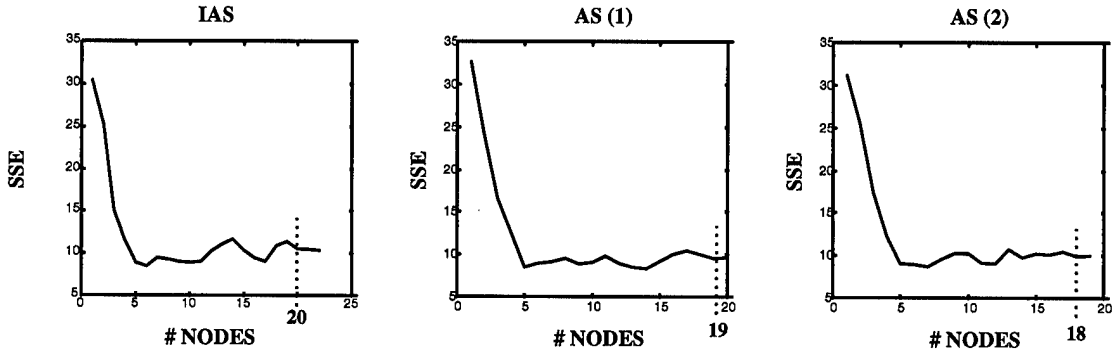


Figure 15. SSE vs. Number of hidden layer nodes for each stage of A.S.

AS(2) - Enter with 4 features and 18 hidden nodes, run18 models in parallel, conduct a likelihood ratio test and leave with 4 features and 17 hidden nodes.

FS(2) - Enter with 4 features and 17 hidden nodes, run 4 models in parallel and test for feature reduction. No features are removed, therefore the algorithm terminates with 17 hidden nodes and 4 features (2 primary, 1 redundant, 1 noise).

FS(1)

Model	SSE_R
1	10.12
2	10.69
3	11.04
4	10.56
5	**9.92**

SSE_F : 9.85
P-Value : .43 [> .05]
Reduce ? : **YES**

FS(2)

Model	SSE_R
1	10.54
2	10.60
3	**10.43**
4	10.86

SSE_F : 9.72
P-Value : .003 [< .05]
Reduce ? : **NO**

Figure 16. Feature Selection for original algorithm.

Three replications, or *experiments*, of this application were conducted. The results of all three are shown in Table 2 below and will be used for comparison purposes later in this chapter.

Initial # of Features: 8

Exp #	# Removed by Feature Screening	# Removed by Feature Selection	Final # of Features	Final # of Nodes
1	**4**	**0**	**4**	**19**
2	3	0	5	17
3	2	1	5	18

Table 2. Results of 3 experiments for original algorithm.

The objective of this example is two-fold. First, it highlights potential areas for improvement. This example began with 22 hidden layer nodes and finished with 18. From the plots in Figure 15 it is apparent that 4 hidden layer nodes are sufficient to handle the XOR problem. The algorithm in its current form has difficulty reaching the

optimal number of nodes due to local minima. For example, if 15 or fewer nodes is superior to 17, if the 16 node model produced a high enough SSE, the likelihood ratio test will reject the reduced model and retain 17 nodes. A methodology is desired which avoids local minima during architecture selection. Another issue with the XOR example is the final feature set. We would expect all noisy and redundant features to be removed, leaving only the two primary XOR features (1 & 2). This is not the case, as both redundant features were retained. One possibility is that the large number of redundant hidden nodes had a negative impact on the effectiveness of the Feature Selection procedure. This issue is addressed again in later in this chapter.

Second, it demonstrates the initial capability of the parallel algorithm. During each phase of the algorithm where multiple models are examined, a significant time savings is realized. To demonstrate the time savings on this problem, Table 3 lists the number of networks that are calculated with an approximate average training time for all. These times are based on approximations, since it is difficult to obtain an exact computation time when multiple users are utilizing the O2000 and CPU time is shared. Remember that *models* are run in parallel, thus the parallel computing time for this example depends on the number of replications per model.

Parallel Computing Time Approximations:

Algorithm Stage	**Models x Reps = Total # Nets**	**Parallel Computing Time (Reps x .5 min/rep)**
Initial Arch Sel	22 x **30** = 660 Nets	**30** x .2 = 6 min.
Saliency Screening	1 x **30** = 30	**30** x .2 = 6
Arch Sel (1)	20 x **30** = 600	**30** x .2 = 6
Feat Sel (1)	5 x **30** = 150	**30** x .2 = 6
Arch Sel (2)	19 x **30** = 570	**30** x .2 = 6
Feat Sel (2)	4 x **30** = 120	**30** x .2 = 6
TOTALS:	**2130 Nets**	**36 min**

Series Computing Time Approximation:
2130 Nets x .2 min/Net = **426 min ≈ 7.1 hrs.**

Table 3. Time savings for parallel algorithm applied to XOR problem.

These calculations demonstrate a time savings of approximately 7.1 hours, a *TSF* of about 11.8 for this problem.

Proposed Architecture Selection Methodologies

This section presents 2 proposed architecture selection methodologies. The first is an information criterion based on maximizing a networks information content, and the second is a signal-to-noise ratio based on an additional hidden layer noise node. The 2 methods are described here and their application to the XOR problem is presented in the next section.

Akaike Information Criterion (AIC). The typical objective of an information criterion is to minimize an error function, such as SSE, while penalizing for increased model size and complexity. The goal is to capture the maximum amount of 'information' with the most efficient model. With regards to regression modeling, the AIC is based on a weighted function of the fit of a maximum log-likelihood of a model and the number of independent parameters estimated to maximize likelihood [17]. It is defined as

$$\mathrm{AIC}(p) = N \ln(\frac{SSE}{N}) + 2(p+2),$$

where N is the number of samples and p the number of independent variables being estimated. Fogel [17] applied the AIC to multi-layer perceptrons in situations of several competing networks. It is also based on a maximum likelihood criterion which is a

function of network residuals. Let t_i be the target value of the ith training pattern where $t \in \{0,1\}$. If we construct a random variable

$$\begin{array}{rl} r_i = 1 - a_i & \text{if } t_i = 1 \\ a_i & \text{if } t_i = 0, \end{array}$$

where a_i is the actual network output for the ith training pattern, then $f(r_1)$ describes the density of the residual error r_i when $t_i = 1$ and $f(r_2)$ describes the density of the residual error r_i when $t_i = 0$. If we then denote ${}_i\beta_{1ki}$ to be the negative output of the final node at layer k for the ith training pattern having a target value of $t \in \{0,1\}$ *before* passing through the nonlinear sigmoid filter, then by the central limit theorem, if there are sufficient hidden layer nodes the asymptotic distributions of ${}_0\beta_{1ki}$ and ${}_1\beta_{1ki}$ (output of final layer node over all i training patterns) are Gaussian with means $\mu_{\beta 0}$ and $\mu_{\beta 1}$ and variances $\sigma^2_{\beta 0}$ and $\sigma^2_{\beta 1}$. The function $f(\bullet)$ can then be defined as

$$f(r_1) = \left[\left(\frac{r_1}{1-r_1}\right)\sigma_{\beta 1}(1-r_1)^2(2\pi)^{1/2}\right]^{-1} \cdot \exp\left(\frac{-\left(\ln\left(\frac{r_1}{1-r_1}\right) - \mu_{\beta 1}\right)^2}{2\sigma^2_{\beta 1}}\right), \text{ and}$$

$$f(r_0) = \left[\left(\frac{1-r_0}{r_0}\right)\sigma_{\beta 0} r_0^2 (2\pi)^{1/2}\right]^{-1} \cdot \exp\left(\frac{-\left(\ln\left(\frac{1-r_0}{r_0}\right) - \mu_{\beta 0}\right)^2}{2\sigma^2_{\beta 0}}\right).$$

If there are P total training patterns with p_1 from class 1 and p_2 from class 2, the likelihood function of the joint density is

$$L(\mu_{\beta 1}, \mu_{\beta 0}, \sigma^2_{\beta 1}, \sigma^2_{\beta 0}) = \prod_{p_1} f(r_1) \cdot \prod_{p_0} f(r_0).$$

The natural logarithm of the likelihood function is labeled as an *intermediate quantity (IQ)*, and is defined as

$$\ln\left(\prod_{p_1} f(r_1)\cdot\prod_{p_0} f(r_0)\right)=$$

$$-\frac{p_1}{2}\ln 2\pi\sigma_{\beta 1}^2-\frac{1}{2\sigma_{\beta 1}^2}\sum_{i=1}^{p_1}\left(\ln\frac{r_{1i}}{1-r_{1i}}-\mu_{\beta 1}\right)^2+\sum_{i=1}^{p_1}\ln\left(\frac{1}{r_{1i}(1-r_{1i})}\right)$$

$$+\frac{-p_0}{2}\ln 2\pi\sigma_{\beta 0}^2-\frac{1}{2\sigma_{\beta 0}^2}\sum_{j=1}^{p_0}\left(\ln\frac{1-r_{0j}}{r_{0j}}-\mu_{\beta 0}\right)^2+\sum_{j=1}^{p_0}\ln\left(\frac{1}{r_{0j}(1-r_{0j})}\right).$$

The maximum likelihood estimators of $\mu_{\beta 1}$, $\mu_{\beta 0}$, $\sigma^2_{\beta 1}$ and $\sigma^2_{\beta 0}$ are given as

$$\hat{\mu}_{\beta 1}=\frac{1}{p_1}\sum_{i=1}^{p_1}{}_1\beta_{1ki}$$

$$\hat{\mu}_{\beta 0}=\frac{1}{p_1}\sum_{i=1}^{p_0}{}_0\beta_{0ki}$$

$$\sigma^2_{\beta 1}=\frac{1}{p_1}\sum_{i=1}^{p_1}({}_1\beta_{1ki}-\hat{\mu}_{\beta 1})^2$$

$$\sigma^2_{\beta 0}=\frac{1}{p_0}\sum_{i=1}^{p_0}({}_0\beta_{1ki}-\hat{\mu}_{\beta 0})^2\,.$$

The *final information statistic* (FIS) is the sum of the negative IQ plus the number of weights being estimated, w, or

$$\text{FIS} = -\text{IQ} + w\,.$$

A notional graph of the FIS is shown if Figure 17. To obtain the desired trade-off between error reduction and model complexity, the minimum FIS is chosen.

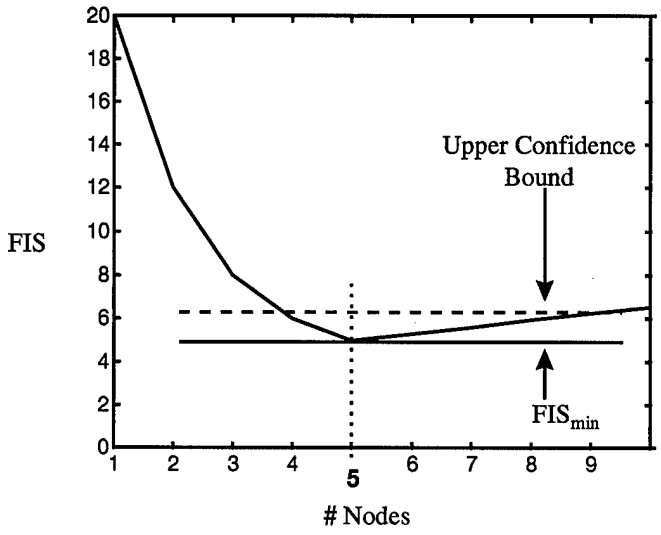


Figure 17. Notional plot of the final information statistic.

Due to the generally conservative nature of the information criterion, a *t*-test using a Bonferroni procedure is applied to examine models of lower complexity than the minimum. This test is equivalent to forming a confidence bound on the minimum value. If a reduced model falls within the upper confidence bound the reduced model is retained.

Assumptions. As applied to an MLP, the AIC assumes the distribution of each output node, prior to the nonlinear sigmoid filter, is gaussian distributed.

Signal-to-Noise Ratio (SNR). The genesis for a SNR approach to node elimination is a signal-to-noise based feature screening methodology developed by Greene and Bauer [18]. In this procedure, the input feature set is augmented with a Unif(0,1) random noise feature. An Elman recurrent neural network (RNN) is trained, and a SNR saliency measure calculated for each feature. The feature with the lowest saliency measure is then removed from training.

The idea of adding random noise is now extended to the hidden layer. Uniform (0,1) is injected as an additional hidden layer *noise node* (Figure 18).

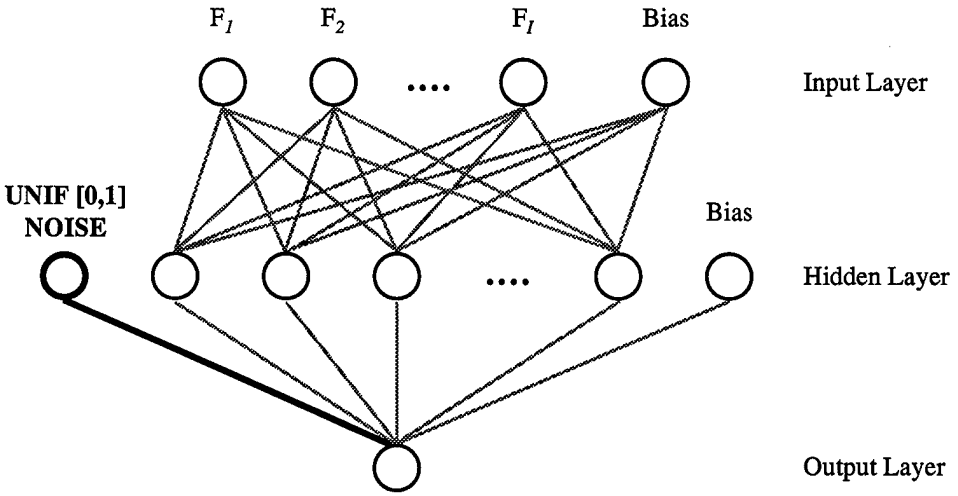


Figure 18. Multi-layer perceptron with added hidden layer noise.

An MLP is trained in the usual fashion (error backpropagation), updating the noise weight as any other weight. The Unif (0,1) output from the noise nodes is passed through a non-linear sigmoid filter, as all other hidden node activations. This is done for the following reason. We will be testing whether each hidden layer node contributes information above noise by examining the weight out of that node. If we use this as a null hypothesis (i.e. that each individual hidden layer node contributes nothing above noise), then we are assuming hypothesizing that all nodes are insignificant. This is obviously not the case, but when the hypothesis holds true, our test statistic (a function of the output weight) is dependent on the hidden node activation, which is passed through a non-linear sigmoid filter. In other words, if we postulate that a regular node is noisy and make comparisons based on the output of a noise node, the output of that noise node should undergo the same transformations as the regular node.

With a fully trained network, we now form signal-to-noise (S/N) ratios based on output weights of the regular nodes and noise nodes. On a decibel (dB) scale, this ratio is defined as

$$\mathrm{SNR}_h = 10\log\left(\frac{\sum_{k=1}^{K} w_{hk}^2}{\sum_{k=1}^{K} w_{Nk}^2}\right), \quad h = 1,\ldots H$$

where H is the number of hidden nodes (less a bias and noise node), K the number of output nodes (1 for the 2-class problem) and w_{Nk} is the noise node weight. A heuristic approach is then applied to determine which nodes are significant. The procedure is described below:

1. Conduct S replications of an MLP with the number of hidden layer nodes equal to the upper bound.
2. For each replication, calculate a SNR on each hidden layer node and based on the final weights of the trained network. Rank the ratios in descending order.
3. Calculate average SNR values *across* replications for each separate ranking. For example, the first average is calculated as the mean of the largest SNR for all replications.
4. Using the SNR averages, retain only those hidden nodes within 10 dB of the maximum.

Once again, this approach is heuristic in nature. The '10 dB' threshold was set based on XOR data, and may not be appropriate for other applications.

Proposed Improvement to Feature Selection Methodology

The *current* feature selection methodology was present in Chapter III. To recap, we construct I reduced models (for I features in the current set), each with a different subset of I-1 features. For each model, S replications are conducted and the replication

with minimum SSE retained *for that model.* Of the I models, the one with minimum SSE is again retained for statistical testing against the full model (with I features). A likelihood ratio statistic incorporating $SSE_{Reduced}$ and SSE_{Full} is used to test whether the reduced model performs as well as the full model. If so, the feature corresponding to the reduced model is eliminated.

As evident from the XOR example in section 2 of this chapter, this procedure often fails to identify a portion of the noisy/redundant features. One explanation is this. The standard procedure with MLPs is to randomly select training vectors from the full set of training and test data. The presentation order is also randomly ordered for each training epoch. The set of training data *and* its order of presentation can impact performance of the network, thus it seems appropriate that when comparing different network structures we should present all candidate models with the same set of training data as well as order of presentation. The Feature Selection procedure currently compares the minimum SSE replication of the full model to the minimum SSE replication of the reduced model, with all replication performed independently. This additional source of variation is perhaps limiting our ability to identify noisy and redundant features.

Another possible explanation is the number of hidden layer nodes used in the XOR example. The large number of redundant nodes is very likely detrimental to network performance, and is also contributing variation to the problem. The proposed solution to these issues utilizes a common-random-number (CRN) approach to training data selection and presentation order, and applies the AIC methodology described earlier

for selecting the number of hidden layer nodes. Results from this approach are presented in the following section. The CRN procedure is discussed next.

Common random numbers are often applied in situations of multiple comparisons where the objective is to compare structural variation of candidate models. The CRN technique is applied to reduce non-structural related variation. In the case of feature selection, we can apply CRNs to the selection and presentation order of training data *between* candidate models across all replications. The table below best describes this procedure.

Rep #	SSE_F Random # Seed	SSE_R Random # Seed
1	30822	30822
2	56753	56753
.	.	.
.	.	.
.	.	.
30	10384	10384

Table 4. Application of CRNs to feature selection.

With this approach the corresponding replications across models will receive the same set and presentation order of training data, thus eliminating a source of variation. The effectiveness of this approach is also discussed in the following section.

Proposed Methodologies Applied to XOR.

This sections presents XOR applications of the methodologies previously discussed. First, the impact of common random numbers on the feature selection methodology is examined. Two versions of the algorithm are applied to XOR data, the

first without CRNs and the second with CRNs. Both versions apply the AIC for architecture selection. Second, the SNR results are presented and observations made regarding this heuristic approach.

XOR Application without CRNs - Once again, this example uses the AIC for architecture selection and the original approach to feature selection (no CRNs). The data set has not changed and thus the upper bound for hidden nodes is 22. Thirty replications were used for both architecture and feature selection. Figures 19 through 21 illustrate the results of the first experiment performed, out of a total of three.

Initial Architecture Seleciton [IAS] → Feature Screening [FSc]

	# Hidden Nodes	# Features	Feature Selection Vector
In	22	8	[1 1 1 1 1 1 1 1]
Out	6	8	[1 1 1 1 1 1 1 1]

	# Hidden Nodes	# Features	Feature Selection Vector
In	6	8	[1 1 1 1 1 1 1 1]
Out	6	5	[1 1 1 1 0 0 1 0]

Architecture Seleciton [AS(1)] → Feature Seleciton [FS(1)]

	# Hidden Nodes	# Features	Feature Selection Vector
In	22	5	[1 1 1 1 0 0 1 0]
Out	5	5	[1 1 1 1 0 0 1 0]

	# Hidden Nodes	# Features	Feature Selection Vector
In	5	5	[1 1 1 1 0 0 1 0]
Out	5 *FINAL*	5	[1 1 1 1 0 0 1 0]

Figure 19. XOR application using original FS methodology.

As you can see from Figure 19, 3 features were eliminated in Feature Screening and none in Feature Selection. Figures 20 and 21 are plots of Architecture Selection and Feature Selection, respectively.

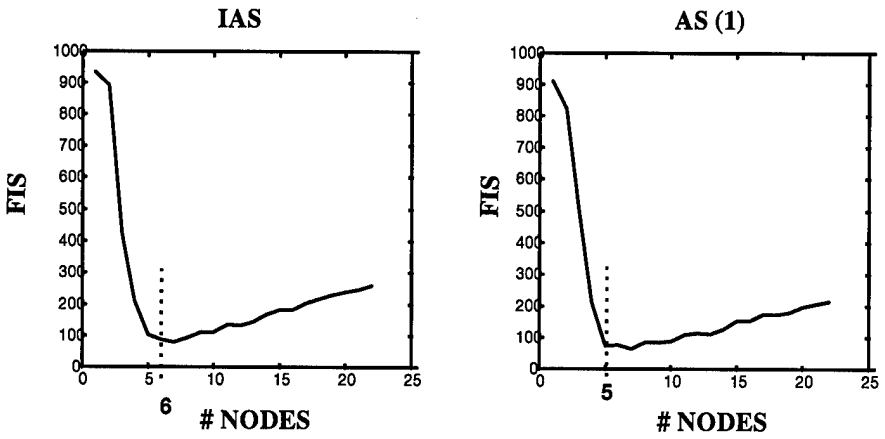


Figure 20. Architecture Selection using AIC.

FS(1)

Model	SSE_R
1	10.82
2	**9.90**
3	11.19
4	10.96
5	11.12

SSE_F : 9.58
P-Value : .009 [< .05]
Reduce ? : **NO**

Figure 21. Feature Selection w/out CRN.

Table 5 below illustrates the results of the 3 experiments, with the first experiment discussed above highlighted in bold.

Initial # of Features: 8

Exp #	# Removed by Feature Screening	# Removed by Feature Selection	Final # of Features	Final # of Nodes
1	**3**	**0**	**5**	**5**
2	3	2	3	5
3	2	3	3	6

Table 5. Results of 3 experiments w/out CRNs.

In the next section, these results are compared to three experiments conducted using CRNs with Feature Selection.

XOR Application with CRNs - This section presents the results when common random numbers are applied to the selection of training data and their presentation order. Three experiments were conducted, with detailed results from the first experiment presented below. Two noise features are removed in Feature Screening and 3 in Feature Selection (Figures 22 and 24). Plots of Architecture Selection are shown for completion (Figure 23).

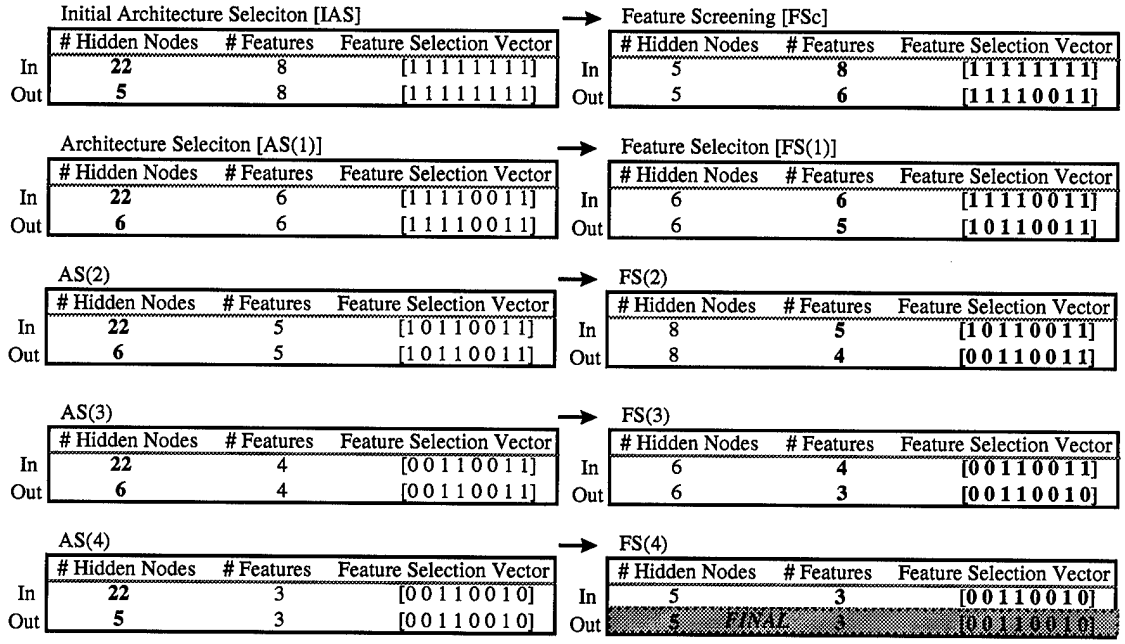


Figure 22. Algorithm Synopsis - AIC Methodology.

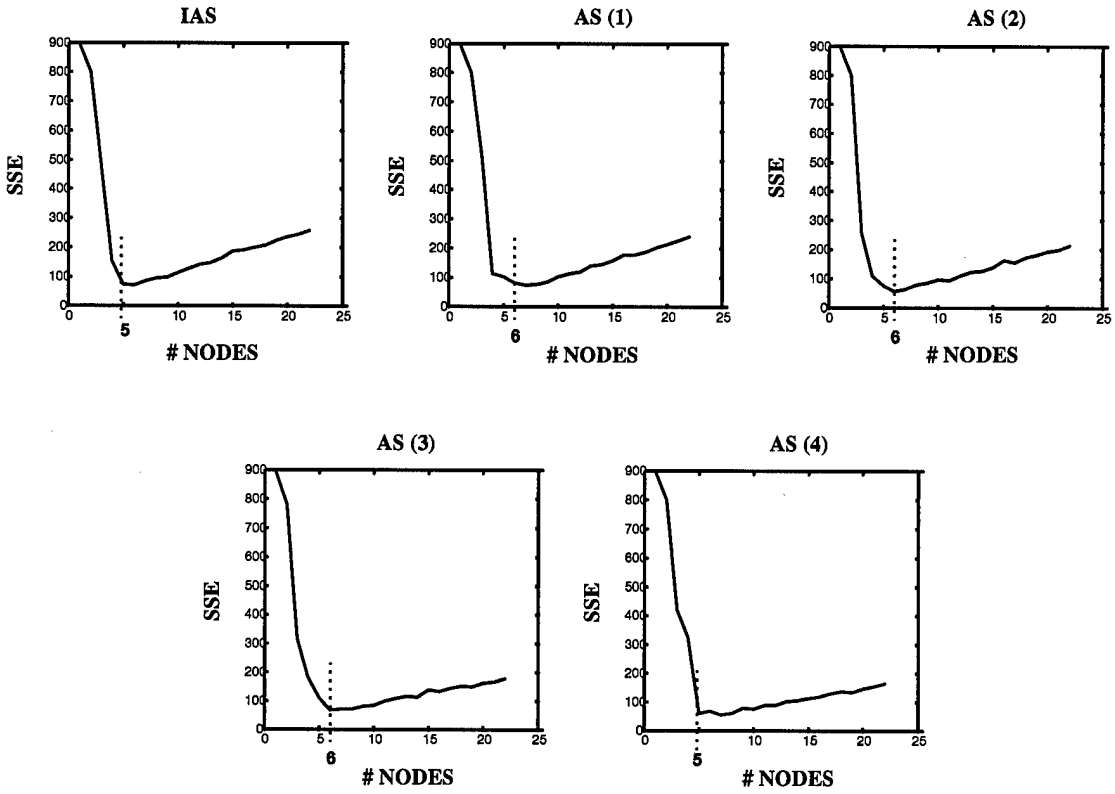


Figure 23. Architecture Selection using AIC methodology.

FS(1)

Model	SSE_R
1	10.22
2	**9.43**
3	10.68
4	10.44
5	9.86
6	10.02

SSE_F : 10.34
P-Value : .99 [> .05]
Reduce ? : **YES**

FS(2)

Model	SSE_R
1	**9.96**
2	10.18
3	103.85
4	10.70
5	10.25

SSE_F : 9.87
P-Value : .58 [> .05]
Reduce ? : **YES**

FS(3)

Model	SSE_R
1	106.03
2	107.08
3	10.75
4	**10.21**

SSE_F : 10.51
P-Value : .90 [> .05]
Reduce ? : **YES**

FS(4)

Model	SSE_R
1	106.88
2	124.38
3	**10.19**

SSE_F : 9.81
P-Value : .003 [< .05]
Reduce ? : **NO**

Figure 24. Feature Selection w/ CRNs.

Notice that the primary XOR features (1 & 2) were eliminated while *redundant* features remain. This is acceptable since the redundant features are linear combinations of the primary features, and should theoretically perform the same as features 1 & 2.

Once again, 3 experiment were conducted. The results are listed in Table 6 below.

Initial # of Features: 8

Exp #	# Removed by Feature Screening	# Removed by Feature Selection	Final # of Features	Final # of Nodes
1	**3**	**0**	**3**	**5**
2	3	3	2	5
3	2	1	5	6

Table 6. Results of 3 experiments w/ CRNs.

Summary of AIC and CRN effectiveness - The AIC used for the above example improves upon the original architecture selection methodology. Although it does not always conclude with the same number of nodes, it typically identifies an *efficient* number of nodes for a given application. This is preferable to the large number of redundant nodes characteristic of the original algorithm, but it is noted that the methodology is still conservative in nature.

The CRN approach did not effectively improve feature selection for the XOR application. This is likely attributed to the relatively low complexity of XOR as well as the large number of training examples used. In this case, the *set* of training data and their order of presentation just don't matter. It seems that CRNs would be more effective in problems of high complexity (complex decision boundaries) with relatively few training exemplars. In these situations, the set of training data could significantly impact network performance and thus any comparisons between different network structures should be made with the same set of training data and order of presentation for each.

XOR Application using SNR - This XOR example applies the SNR Architecture Selection methodology to the same set of XOR data as above. Common random numbers are also used in Feature Selection. From Figure 25 below you can see that the SNR heuristic performs about as well as the AIC for architecture selection. There are 5 hidden layer nodes at program completion.

Initial Architecture Seleciton [IAS] → Feature Screening [FSc]

	# Hidden Nodes	# Features	Feature Selection Vector		# Hidden Nodes	# Features	Feature Selection Vector
In	22	8	[1 1 1 1 1 1 1 1]	In	5	8	[1 1 1 1 1 1 1 1]
Out	5	8	[1 1 1 1 1 1 1 1]	Out	5	5	[1 1 1 1 0 1 0 0]

Architecture Seleciton [AS(1)] → Feature Seleciton [FS(1)]

	# Hidden Nodes	# Features	Feature Selection Vector		# Hidden Nodes	# Features	Feature Selection Vector
In	22	5	[1 1 1 1 0 1 0 0]	In	5	5	[1 1 1 1 0 1 0 0]
Out	5	5	[1 1 1 1 0 1 0 0]	Out	5	4	[0 1 1 1 0 1 0 0]

AS(2) → FS(2)

	# Hidden Nodes	# Features	Feature Selection Vector		# Hidden Nodes	# Features	Feature Selection Vector
In	22	4	[0 1 1 1 0 1 0 0]	In	5	4	[0 1 1 1 0 1 0 0]
Out	5	4	[0 1 1 1 0 1 0 0]	Out	5	3	[0 1 1 0 0 1 0 0]

AS(3) → FS(3)

	# Hidden Nodes	# Features	Feature Selection Vector		# Hidden Nodes	# Features	Feature Selection Vector
In	22	3	[0 1 1 0 0 1 0 0]	In	5	3	[0 1 1 0 0 1 0 0]
Out	5	3	[0 1 1 0 0 1 0 0]	Out	5	2	[0 1 1 0 0 0 0 0]

AS(4) → FS(4)

	# Hidden Nodes	# Features	Feature Selection Vector		# Hidden Nodes	# Features	Feature Selection Vector
In	22	2	[0 1 1 0 0 0 0 0]	In	5	2	[0 1 1 0 0 0 0 0]
Out	5	2	[0 1 1 0 0 0 0 0]	Out	5	FINAL	[0 1 1 0 0 0 0 0]

Figure 25. Algorithm synopsis - SNR Methodology.

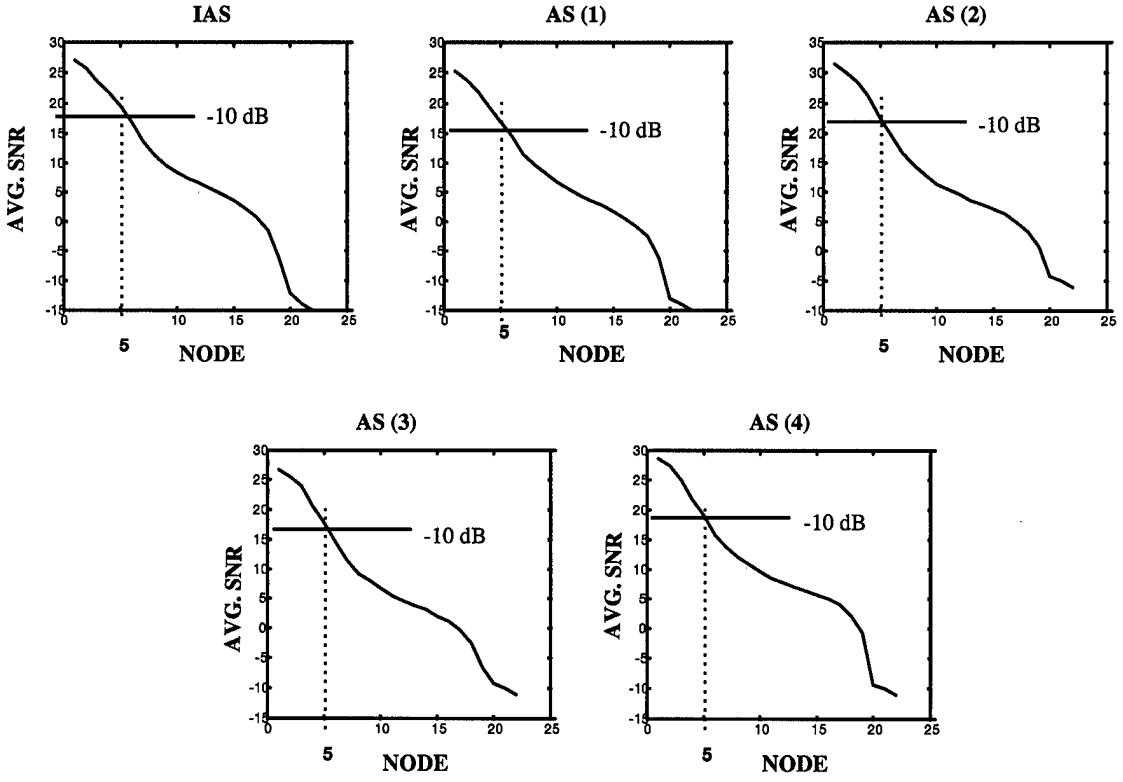


Figure 26. Architecture Selection - SNR Methodology.

FS(1)

Model	SSE_R
1	**9.92**
2	9.99
3	9.95
4	10.40
5	10.25

SSE_F : 11.09
P-Value : .99 [> .05]
Reduce ? : **YES**

FS(2)

Model	SSE_R
1	11.01
2	124.78
3	**9.73**
4	10.16

SSE_F : 10.43
P-Value : .99 [> .05]
Reduce ? : **YES**

FS(3)

Model	SSE_R
1	124.06
2	124.85
3	**9.72**

SSE_F : 10.54
P-Value : .99 [> .05]
Reduce ? : **YES**

FS(4)

Model	SSE_R
1	124.94
2	**124.38**

SSE_F : 10.56
P-Value : .000 [< .05]
Reduce ? : **NO**

Figure 27. Feature Selection - SNR Methodology.

Although the current heuristic performs well for the XOR problem, there may be limitations when applied to new problems. The issue lies with the 10 dB threshold used for eliminating nodes. This value was determined simply by observing SNR values and selecting the threshold which, on average, would implicate the appropriate number of nodes for the XOR problem. Had we not known that 4 hidden layer nodes are sufficient, this technique would have been quite impossible. Therefore, we desire a heuristic which is robust across many applications.

Recall that the current SNR procedure examines one model, that with the upper bound number of hidden nodes. The hope is that by including an excessive number of

nodes in the model, all redundant nodes will exhibit identifiably different SNR values and be eliminated by the heuristic. A problem with this approach was observed regarding the distribution of redundant node SNR values for models of increasing complexity. For models with only 1 or 2 redundant nodes, their is a noticeable difference between non-redundant and redundant node SNR values. As the model increases in size, the difference becomes less and less noticeable. This phenomenon is depicted in Figure 28, where Model 5 represents an MLP with 5 hidden layer nodes, and so on.

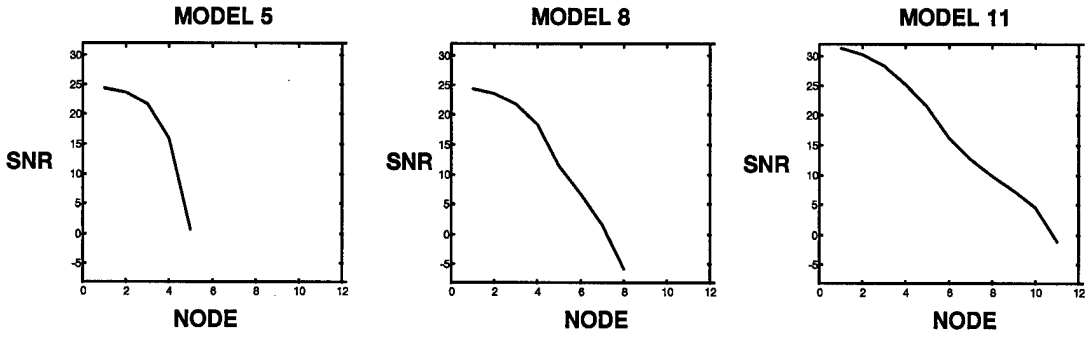


Figure 28. SNR values for models of increasing complexity.

From these plots we can see the sharp decrease in SNR for the 5th node of Model 5. For the Model 11, on the other hand, SNR values exhibit a steady decline from the maximum. Table 7 below displays numeric values for the above plots, with models 5, 8 and 11 in bold. The box in the middle of the table surrounds data for the 5th largest SNR value for Model 5 and beyond. These values reflect an increased inability do distinguish between non-redundant and redundant nodes.

MODEL										
1	2	3	4	**5**	6	7	**8**	9	10	**11**
42.9	32.3	28.3	23.8	**24.4**	22.9	24.3	**24.4**	27.1	27.0	**31.3**
	29.4	26.9	22.8	**23.7**	22.0	23.4	**23.5**	26.2	25.7	**30.2**
		19.4	20.8	**21.7**	20.9	22.0	**21.8**	24.6	23.9	**28.4**
			19.0	**16.9**	17.4	18.1	**18.4**	21.6	20.8	**25.2**
				0.7	5.3	9.6	**11.5**	15.0	15.9	**21.5**
					-6.1	1.8	**6.7**	10.0	10.9	**16.2**
						-6.3	**1.5**	5.9	6.0	**12.6**
							-5.8	2.7	3.2	**9.8**
								-5.1	-0.3	**7.4**
									-8.0	**4.6**
										-1.1

Table 7. SNR values for models of increasing complexity.

The observations made here imply a *bottom-up* heuristic approach to node selection. In other words, say we begin with an initial architecture of one hidden layer node. A heuristic procedure could be applied which augments the current model by one node and examines the resulting SNR values. At the point where a redundant node is added, there should be an identifiable difference in SNR values between the non-redundant nodes and the single redundant node. At this point the heuristic would terminate, retaining the current number of hidden layer nodes.

Satellite Behavior Application.

This section presents an application of the parallel algorithm to the identification of anomalous satellite behavior.

Data - Ten features with 1000 data exemplars are used for this problem. They include derivatives of block fourier 2nd order moment-based features extracted from 128 x 128 gray-scale telescope and simulated imagery of a TD-81 satellite.

Approach - The parallel algorithm with common random number and AIC additions was applied to the above data. As in earlier examples, 3 experiments were conducted. The results of these experiments are shown below.

Exp #	Final Number of Features	Final Feature Selection Vector	Final Number of Nodes
1	6	[1 1 1 0 0 0 1 1 1 0]	4
2	7	[1 1 1 0 1 1 1 0 1 0]	6
3	7	[1 0 1 1 1 1 1 0 1 0]	6

Table 8. Results of satellite behavior experiments.

Although the number of features retained across experiments is fairly constant, the specific features retained seems to vary slightly. Feature 10 was the only one eliminated in all experiments. Features 4 and 8 were eliminated in two of the three, and 2, 5 and 6 were all eliminated once. Features 1, 3, 5 and 7 were retained in all three experiments, thus they will be used in the MLP application. Regarding the number of hidden layer nodes, since we have retained only the features that were not eliminated in any of the three experiments, we will also use the lower end on hidden nodes, and use 4. Ten independent networks were run with these input parameters and average classification values calculated. These results are displayed in Figure 29.

MLP Approach

		Classified as: Normal	Anomalous
Truth	Normal	85.47 %	14.53
	Anomalous	12.62	89.38

Alternate Approach

		Classified as: Normal	Anomalous
Truth	Normal	84.23 %	15.77
	Anomalous	8.13	91.87

Figure 29. Confusion Matrices for MLP and Alternate Approach.

The confusion matrices above contrast the MLP based approach with a method of feature reduction and classification previously applied to the satellite problem. As you can see, the MLP approach is slightly better for the Normal/Normal case and slightly worse for the Anomalous/Anomalous case. We would have hoped that an MLP would outperform the statistical classifier instead of merely matching its performance. One possible explanation is this. After the experiments were conducted it was discovered that the input features had undergone multiple transformations. In essence, the data was tailored for a statistical classifier and not in raw form. Upon further investigation it was determined that a certain amount of discriminative information which an MLP would benefit from was lost in the transformations. Thus, if applied to the raw data it is entirely possible that the MLP approach would outperform the alternate approach.

V. Conclusions

This thesis effort has applied parallel programming techniques to an integrated and iterative procedure for selecting input features and hidden layer nodes in MLP applications. The parallel algorithm reduces run-times by a factor of up to 31, depending on the size of the problem. Proposed changes to the algorithm have been demonstrated to successfully improve architecture selection. The suggested improvements for feature selection, although ineffective for the XOR application, are likely to improve algorithm performance in problems with complex decision boundaries and relatively few training exemplars. In addition, a signal-to-noise heuristic was developed for architecture selection and shown to perform as well as the Akaike Information Criterion for XOR data. An improved SNR heuristic has also been recommended which may be more robust across other applications. The final parallel algorithm was applied to a satellite imaging application and shown to perform equally as well as the current procedure, given the data limitations described earlier.

The final product of this thesis is an analytic tool useful for finding an optimal MLP structure for a given application. In addition, this tool is useful for examining properties of the multi-layer perceptron, due to the large number of networks which can be run in a short amount of time. The final code is well documented and structured to allow for easy development and testing of algorithm improvements or additions.

VI. Recommendations for Continued Research

The Integrated Feature and Architecture Selection algorithm continues to be a powerful tool for determining an optimal MLP structure. However, it is not flawless. We should always be considering improvements to the algorithm as our knowledge and understanding of the neural network expands. Current limitations of the algorithm stem from its somewhat inconsistent results over multiple experiments, as demonstrated in the satellite imaging application. The method of feature selection, even with CRN applied, often fails to identify *all* of the noisy and redundant features. This is likely attributed to the strict nature of the likelihood ratio test applied in feature selection. The CRN modification should be examined across a variety of high and low complexity problems to gauge its impact on feature selection. In addition, the proposed improvement to the SNR heuristic should be investigated further to determine its robustness across different applications.

Bibliography

1. Steppe, Jean M., Kenneth W. Bauer, Jr., and Steven K. Rogers. "Integrated Feature and Architecture Selection," *IEEE Transactions on Neural Networks*, Vol. 7, No. 4, July 1996.

2. Steppe, Jean M. *Feature and Model Selection in Feedforward Neural Networks.* PhD Dissertation, Air Force Institute of Technology, June 1994.

3. Bishop, Christopher M. *Neural Networks for Pattern Recognition.* New York: Oxford University Press Inc., 1995.

4. Sarkar, Manish and B. Yegnanarayana. "Feedforward Neural Networks Configuration using Evolutionary Programming," *The 1997 IEEE Conference on Neural Networks, June 9 - June 12, 1997.* 438-443. New York: IEEE, 1997.

5. Rathbun, Thomas F., Steven K. Rogers, Martin P. DeSimio and Mark E. Oxley. "MLP iterative construction algorithm," *Neurocomputing* 17: 195-216 (1997).

6. Rogers, Steven K., and others. *An Introduction to Biological and Artificial Neural Networks.* 49-50. Air Force Institute of Technology, 1990.

7. Le Cun, Y., and others. Optimal Brain Damage. *Neural Information Processing Systems II,* 598-605, ed. D.S. Touretzky, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

8. Mozer, M. C. and P. Smolensky. "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," *Advances in Neural Information Systems I.* Ed. David S. Touretzky, Morgan Kaufmann Publishers, San Mateo, CA.

9. Ruck, D.W., and others. "Feature Selection Using a Multilayer Perceptron," *The Journal of Neural Network Computing,* 2(2):40-48, Fall 1990.

10. Tarr, G.L. *Multi-Layered Feedforward Neural Networks for Image Segmentation.* PhD Dissertation, Air Force Institute of Technology, December 1991.

11. Steppe, Jean M. and Kenneth W. Bauer, Jr. "Improved feature screening in feedforward neural networks," *Neurocomputing* 13: 47-58 (1996).

12. White, Halbert. "Learning in Artificial Neural Networks: A statistical Perspective," *Neural Computation* 1, 425-464, 1989.

13. White, Halbert. *Artificial Neural Networks Approximation & Learning Theory,* Blackwell Publishers, Cambridge, USA, 1993.

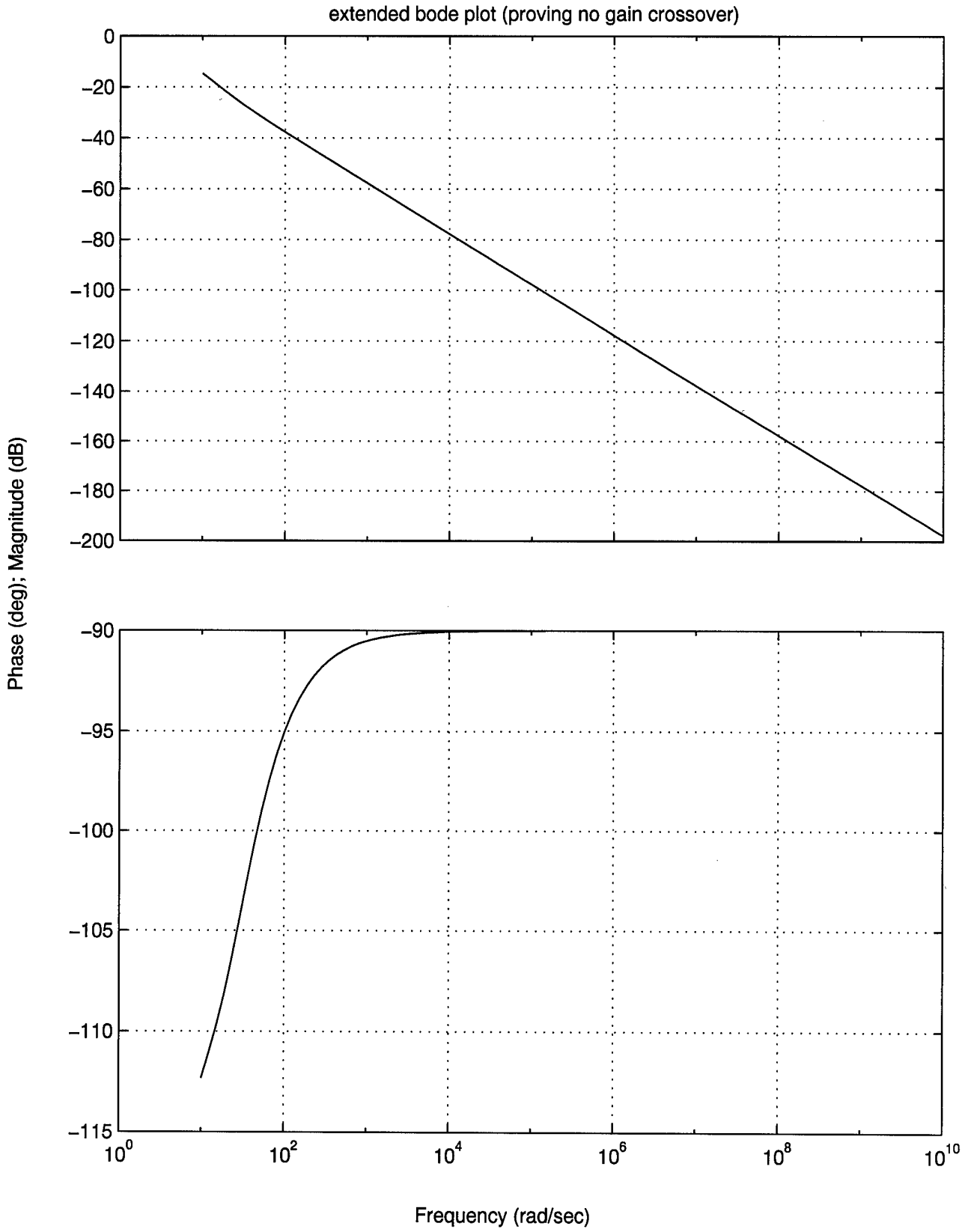
14. Sarkar, M. and B. Yegnanarayana. "Feedforward Neural Networks Configuration using Evolutionary Programming," *Proceedings of the International Conference on Neural Networks (ICNN '97),* IEEE New York, NY, 1997.

15. Pacheo, Peter S. *Parallel Programming with MPI,* Morgan Kaufmann Publishers, Inc. 1997.

16. Cover, T.M. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers,* EC-14:326-334, June 1965.

17. Fogel, David B. "An Information Criterion for Optimal Neural Network Selection," *IEEE Transactions on Neural Networks,* Vol 2, No 5, September 1991.

18. Greene, Kelly A., Kenneth W. Bauer, Jr. and others. "Estimating pilot workload using Elman recurrent neural networks: A preliminary investigation," *Intelligent Engineering Systems Through Artificial Neural Networks,* Vol 7, Nov 97.


Bode Diagrams
extended bode plot (proving no gain crossover)
Phase (deg); Magnitude (dB)
0
-20
-40
-60
-80
-100
-120
-140
-160
-180
-200
-90
-95
-100
-105
-110
-115
10^0
10^2
10^4
10^6
10^8
10^{10}
Frequency (rad/sec)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	13 March 1998	Masters Thesis

4. TITLE AND SUBTITLE
Parallel Implementation of an Integrated Feature and Architecture Selection Algorithm

5. FUNDING NUMBERS

6. AUTHOR(S)
Craig W. Rizzo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/98M-20

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Kenneth Bauer, Jr.
AFIT/ENS BLDG 640
2950 P STREET
WRIGHT PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The selection of salient features and an appropriate hidden layer architecture contributes significantly to the performance of a neural network. A number of metrics and methodologies exist for estimating these parameters. This research builds on recent efforts to integrate feature and architecture selection for the multi-layer perceptron. In the first stage of work a current algorithm is developed in a parallel environment, significantly improving its efficiency and utility. In the second stage, improvements to the algorithm are proposed. With regards to feature selection, a common random number (CRN) addition is proposed. Two new methods of architecture selection are examined, to include an information criterion and a signal-to-noise based procedure. These methodologies are shown to improve algorithm performance.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Multi-layer perceptron, architecture selection, feature selection, information criterion, common-random numbers	74
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL