12-1996

Adaptive Neural Network Controller for ATM Traffic

Jeffrey E. Larson

ADAPTIVE NEURAL NETWORK CONTROLLER FOR ATM TRAFFIC

THESIS

Jeffrey E. Larson
Captain, USAF

AFIT/GE/ENG/96D-09

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

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/96D-09

ADAPTIVE NEURAL NETWORK CONTROLLER FOR ATM TRAFFIC

THESIS

Jeffrey E. Larson
Captain, USAF

AFIT/GE/ENG/96D-09

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/GE/ENG/96D-09

ADAPTIVE NEURAL NETWORK CONTROLLER FOR ATM TRAFFIC

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 Electrical Engineering

Jeffrey E. Larson, BSB, BSEE
Captain, USAF

December, 1996

Acknowledgements

I would like to thank the people that assisted me in completing this project. This document is the result of a team effort and while there's only one name on the front, there were more than that responsible for its completion .

First I would like to thank my advisor, Dr Steve Rogers, for hanging in there with me and providing me with much needed direction when the objective occasionally blurred. The rest of my committee, Dr. Marty DeSimio and Capt Rick Raines, provided invaluable insight that kept me from areas that could have spelled disaster. Thank you.

The most important people in this process probably payed the highest price, my family. Thanks most of all to my wife Michelle for enduring the grueling hours and providing me the support and encouragement I needed. Thanks also to my girls Brittany, Kristin, Erika and Jeanette for constantly reminding me that school is not the most important thing.

Jeffrey E. Larson

Table of Contents

List of Figures

List of Tables

Abstract

Broadband-Integrated Services Digital Networks (B-ISDN), along with Asynchronous Transfer Mode (ATM), were designed to meet the requirements of modern communication networks to handle multiple users and a wide variety of diverse traffic including voice, data and video. ATM responds to requests for admission to the network by analyzing whether or not the grade of service (GOS) requirement, specified in the admission request, can be guaranteed without violating the GOS guaranteed to traffic already accepted into the network. The GOS is typically a parameter such as cell loss rate (CLR), average delay, or some other measurement associated with network performance. In order to develop a tractable mathematical algorithm for controlling admission, an accurate model of the communication network and traffic in question is necessary. The complex and dynamic nature of these communication networks make them very difficult to model. Even when such a model can be developed, often with unrealistic simplifications or unsupportable assumptions, the associated mathematical algorithm is frequently excessively cumbersome and timely processing of an admission request is lost. An alternative to conventional mathematical algorithms for cases like these is the use of neural networks (NN). NNs can learn complicated functions relating the inputs and outputs of a system without prior knowledge about the system itself. For ATM B-ISDN networks, NNs can learn the function relating input traffic parameters and resulting network performance by training on an appropriate set of traffic parameter inputs and resulting GOS outputs. In this work three neural network admission controller schemes are examined. The Bayes error rate, as bounded by the Parzen window technique, is also introduced as a benchmark for measuring the performance of these admission controllers. Results indicate that error rates approaching the Bayes error rate can be obtained by using a self organizing, or clustering, algorithm to segment the input space and then train separate MLPs on each cluster. This clustering algorithm can also be used to direct the traffic streams requiring classification to the appropriately trained NN admission controller.

ADAPTIVE NEURAL NETWORK CONTROLLER FOR ATM TRAFFIC

I. Introduction

1.1 Background

The Air Force's Global Reach - Global Power operational concept demands smaller, rapidly deployable forces operating in dynamic environments [7]. To support these forces, the Air Force has implemented new Command, Control, Communications and Intelligence (C3I) structures to provide command and control information to the warfighters, whenever and wherever it is needed. One example is Rome Laboratory's "Information For The Warrior" (IFTW) program [9] in support of Air Mobility Command (AMC). These communications structures will employ a variety of communications facilities, such as satellites, tactical radios, and terrestrial networks based on asynchronous transfer mode (ATM). These networks may be military, commercial or a combination of both. Efficient network management, including congestion control, is essential to optimize the resources available and allow the maximum amount of information possible to flow to the commanders who need it.

Modern communication networks must be capable of handling multiple users and a wide variety of diverse traffic including voice, data and video. Broadband-Integrated Services Digital Networks (B-ISDN) are designed to meet these requirements. In order to efficiently and flexibly allocate bandwidth in the dynamic environment these networks operate in, asynchronous transfer mode (ATM) was developed as the switching protocol. ATM combines high speed packet switching with statistical multiplexing. Packet switching involves dividing a message into finite length

segments, with each segment addressed and sent to its destination separately. The message is then reassembled at the destination. Each packet in ATM, referred to as a cell, is 53 bytes long. A message that is broken into cells is termed a call. With statistical multiplexing, packets from multiple traffic sources are combined in a single First-In-First-Out (FIFO) queue with a constant service rate. ATM is distinguished from synchronous transfer mode (STM) in that ATM allows each user to have an effective data rate that matches their instantaneous data rate requirement. This data rate requirement can be large or small and can change over time. In STM, the user is assigned a fixed data rate.

A distributed ATM network is represented in Figure 1. The individual calls, or messages, request admission to the network and identify the grade-of-service (GOS) they require. The GOS is typically a parameter such as cell loss rate (CLR), average delay, or some other measurement associated with network performance. The link controller must decide if the GOS of the incoming call can be guaranteed while maintaining the GOS of calls already admitted. The links to the network are bi-directional, that is they can transmit as well as receive. The intermediate nodes are routers and may also be sources or destinations of messages. The cells or packets of a message may all take different routes through the network before they are reassembled at their destination.

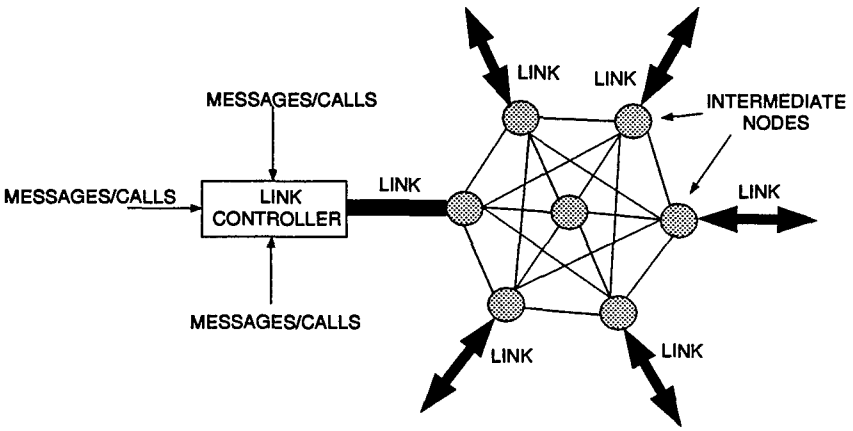


Figure 1 Distributed ATM Network

In order to develop a tractable mathematical algorithm for deciding which requests for admission are accepted, an accurate model of the communication network and traffic in question is necessary. The complex and dynamic nature of these communication networks make them very difficult to model. The traffic, as well as the B-ISDN components comprising the communication network, are non-linear with complex statistical characteristics. Even when such a model can be developed, often with unrealistic simplifications or unsupportable assumptions, the associated mathematical algorithm is frequently excessively cumbersome and timely processing of an admission request is lost. The high speed nature of these communications networks require high speed processing of each admission request. An alternative to conventional mathematical algorithms for cases like these is the use of neural networks (NN). Park and Lee [24] describe a NN as, "a parallel, distributed, information processing structure consisting of many processing elements interconnected via weighted connections." Another way to think of NNs is as a nonlinear circuit mapping multiple inputs to multiple outputs. NNs can learn complicated functions relating the inputs and outputs of a system without prior knowledge about the system itself. For ATM B-ISDN networks, NNs can learn the function relating input traffic parameters and resulting network performance by training on an appropriate set of input and output data. This function can then be used to decide the merits of network admission requests and make an appropriate decision.

The choice of an optimal neural network architecture, as well as the method of training, that results in an accurate prediction of network performance are design decisions that need to be made before an ATM B-ISDN network is deployed. The training technique must be able to react to widely diverse traffic patterns while maintaining the highest level of performance possible. An examination of the set of feature parameters describing the network traffic, and used as inputs to the NN, to arrive at an optimal minimum will further increase the efficiency of the admission control process.

1.2 Problem Statement

This thesis will develop several neural network ATM admission controllers and investigate their generalization capabilities and overall performance in recognizing communication traffic streams that cause congestion situations in a link to an ATM network.

1.3 Scope

Multi-layer perceptron (MLP) neural networks will be trained to classify ATM network traffic into acceptable and unacceptable streams using back propagation. The traffic streams will be generated by BONeS® DESIGNER™, a communication network simulator. A baseline parameter set representing the boundary between acceptable (does not cause congestion) and unacceptable streams (causes congestion) will be determined. The primary baseline parameters will then be varied resulting in 12 parameter sets at various distances from the boundary. These parameter sets represent communication streams that are the most difficult to classify because of their proximity to the boundary. The MLP neural network architecture will be determined, implemented in various link controller schemes, and the performance of these approaches will be compared to each other, as well as to the optimal Bayes error rate.

1.4 Thesis Organization

The following chapter reviews previous research conducted using NNs to control ATM networks. The types of neural networks used, features used for inputs, as well as various training techniques are examined. Chapter *III* describes the techniques employed in this effort to investigate the performance of various implementations of MLPs for ATM link controllers. The results and analysis of this investigation, as applied to simulated data is presented in Chapter *IV*. Chapter *V* contains conclusions drawn from the results and areas of further study.

1.5 Summary

The heterogeneous nature of ATM traffic makes it very difficult to model. Neural networks show promise for the admission control task in these networks because they do not require a predefined model. Without the effective control of high-speed networks, capable of handling voice, data and video traffic, the successful deployment of these networks, to provide timely and accurate C3I for commanders, is threatened.

II. Background

2.1 Introduction

This chapter begins with an examination of recent work in the area of algorithmic control of ATM networks. Current research using neural networks for the control of ATM networks is explored next, concentrating on network management functions. Finally, training techniques and feature parameter analysis are covered.

2.2 Non Neural Network ATM Congestion Control

Congestion control in an ATM network is defined as maintaining the network's, or link's, performance at a level sufficient to maintain the required grade of service (GOS) guaranteed to all calls at network admission, while also making efficient use of a large bandwidth [16]. Access, or admission, control is defined as keeping the traffic level in the network, or link, below the congestion level.

Many different techniques have been developed to control ATM network congestion [11,16,18,21,28] One approach involves employing different congestion control procedures for different traffic types. Sriram [28] divided the traffic into two types, data and voice. For voice traffic he proposed employing such techniques as cell dropping. Cell dropping involves eliminating some cells from a call in order to keep traffic below the congestion level and maintain the required GOS. It is argued that the loss of some individual cells in a voice traffic stream will not significantly degrade the sound quality. For data traffic he proposed a variety of techniques including window-based flow control, bandwidth usage monitoring, prioritization and rate based flow control. Window-based flow control involves dividing the traffic into finite lengths, or windows, and basing the congestion control criteria on the characteristics of the traffic contained in the window. Bandwidth usage monitoring tracks the traffic load and relates it to the network capacity. The use of a ranking system to assign some calls more importance than others and then using this ranking to decide which calls

are eliminated first when a congestion situation arises is the concept behind prioritization. Finally rate-based flow control uses the average and peak rate of calls to predict network performance and control congestion.

Jiang and Meditch [18] divided the traffic into constant bit rate (CBR) and variable bit rate (VBR) traffic. CBR traffic typically represents fixed rate video and other real time traffic while VBR traffic represents voice, data and other bursty traffic [14]. They used Synchronous Transfer Mode (STM), which allocates fixed time slots per call, as the switching protocol for CBR traffic and ATM for VBR data. An individual switch could then consist of all ATM, all STM, or a combination of both.

Most techniques involving an algorithmic approach to congestion and admission control make an a priori assumption of a finite number of traffic classes based on a model of the incoming traffic. The extremely heterogeneous and high speed nature of ATM networks, however, does not lend itself to any traffic modeling assumptions. Neural network admission control does not require any a priori traffic model assumptions, only that the data available for training is representative of the traffic needing classification, and so can handle a wide variety of traffic.

2.3 Neural Network Control of ATM Networks

2.3.1 Introduction. Hiramatsu [13] is widely cited as conducting some of the earliest work in controlling ATM networks with NNs. His initial work involved call admission control at a single entry point and established the feasibility of using NN controllers in ATM networks to achieve results comparable to algorithmic approaches. From this beginning, many researchers have expanded the scope of functions handled by NNs. Current work ranges from viewing the challenge as a network optimization problem [24] using feedback NNs to using distributed, interconnected NNs to control all network functions [1].

Neural networks have many advantages that make them good candidates for traffic controllers in ATM networks. Chen and Leslie [4] describe these as:

1. Adaptive learning process requires no system model.
2. High computation rate due to massive parallelism in the hardware implementation.
3. Ability to generalize, or adapt, to a situation it has not previously seen.
4. High degree of robustness.

There are also a few disadvantages [17] associated with neural networks:

1. Increased complexity.
2. Training phase.
3. Poor performance under low load.

The increased complexity and training phase are characteristics that will be minimized as part of the design process. Complexity can be reduced by designing the NN with the fewest number of nodes, or neurons, as possible. The number of nodes in a NN is affected by the dimension of the input space, the number of output classes, the number of hidden layers and the number of nodes in each hidden layer. The duration of the training phase is also related to the number of neurons in the NN, as well as the number of input, or feature, vectors available for training. Poor performance under low load conditions is not important given the objective of controlling the communication network under high load conditions. The benefit of adequately controlling the network under high load conditions outweighs any performance fall off under low load conditions.

2.3.2 Neural Network Architecture. A multi-layer perceptron (MLP) is a neural network consisting of an input layer, one or more hidden layers and an output layer. The input layer contains one neuron, or node, for each element of the input feature vector. The only observable output of an MLP is the result of the processing of the last layer, or output layer, with the middle, or "hidden" layer(s) outputs being fed to the next layer. Figure 2 shows a typical MLP neural network. The inputs are

weighted by matrix **W1** and fully interconnected to the single hidden layer. Each node in the hidden layer sums the weighted outputs of the input layer and applies a non-linear function. This sum of weighted outputs from the previous layer is termed the activation and the non-linearity is the activation function. The output of the hidden nodes are then weighted by matrix **W2** and supplied to the output node. Similar to the hidden node, the output node sums its inputs and applies its own activation function, which may or may not be the same activation function as the hidden layer.

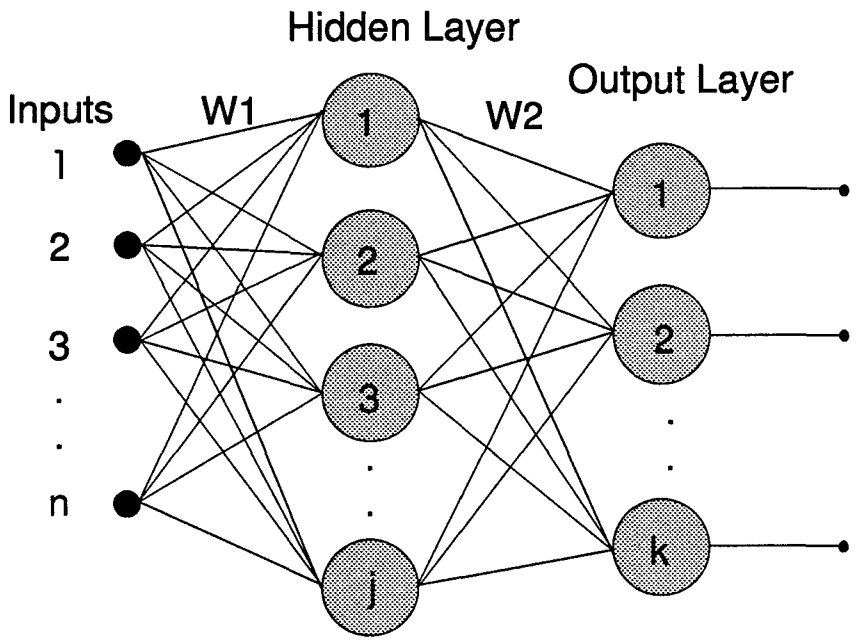


Figure 2 Typical Neural Network

The architecture of a MLP involves such things as the number of hidden layers, the number of nodes in each hidden layer and the activation function employed by each layer in a MLP. The following sections briefly describe these items.

2.3.2.1 Number of Hidden Layers. The number of hidden layers is usually chosen in anticipation of the complexity of the decision regions. The more complex the decision regions, the more hidden layers required to accurately represent it. It has been shown, however, that any non-linear function can be approximated

by a MLP with only one hidden layer, given enough nodes in that hidden layer [5]. This effort will, therefore, use a single hidden layer MLP.

2.3.2.2 Number of Hidden Layer Nodes. The number of nodes required in the single hidden layer, to accurately classify a given set of feature vectors, is a function of the complexity of the feature space. The more complex the feature space, the more nodes required. Since the feature space complexity is unknown, the number of hidden nodes will be varied to determine the minimum number required to achieve optimal performance. It is desirable to keep the number of hidden nodes to a minimum so the computational complexity of the MLP is minimized, resulting in faster processing and better generalization, and the number of feature vectors required to adequately train the NN is minimized.

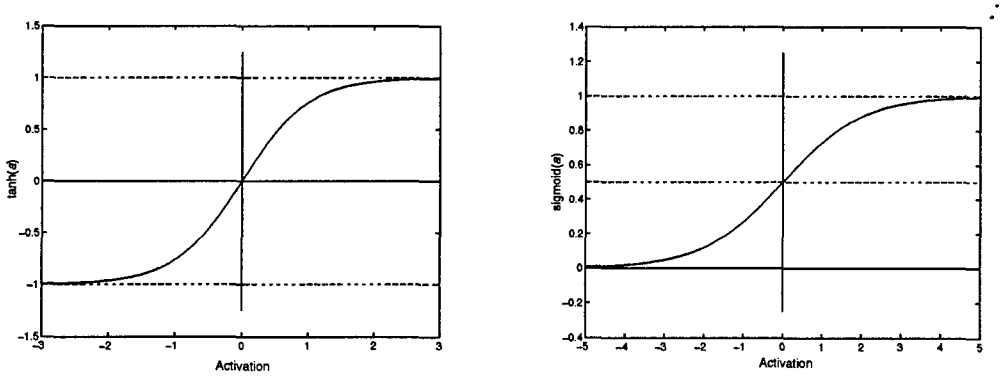
A rule of thumb for determining the number of training vectors required to adequately train a NN is given by $P = \frac{S}{\epsilon}$, where P is the number of training vectors required, S is the number of weights in the NN and ϵ is the acceptable uncertainty rate [2]. Using this rule, for an accuracy rate of 90% (corresponding to $\epsilon = 0.1$), $P = 10 * S$. For example, a NN with two input features, three hidden layer nodes and two classes (which can be implemented with one output node) has nine total weights associated with it (2 input nodes X 3 hidden nodes = 6 weights connecting the input and hidden layers, plus, 3 hidden nodes X 1 output node = 3 weights connecting the hidden and output layers for a total of nine weights). Now, using the rule of thumb at least 90 training vectors are required to adequately train this NN. Note that the value ϵ represents the level of confidence in the MLPs classification accuracy, not the actual classification accuracy.

2.3.2.3 Activation Functions. The last part of a NNs architecture is the activation function employed by the hidden and output layers. The activation function is a nonlinearity applied to the summed total of the weighted inputs to a particular node. The summation of the inputs to a particular node is the activation

and can be viewed as a measure of similarity between the input vector and the weight vector. The activation function typically approaches some upper limit for large positive activations and a lower limit for large negative activations. Two popular activation functions are the hyperbolic tangent (tanh) and the sigmoid (see Figure 3). It has been argued that the use of tanh as the activation function on hidden nodes is advantageous in the construction of efficient training methods [19] and so, based on this argument, it is used for all hidden layer nodes in this thesis. The problem being examined is a two class problem, so only a single output node is required. A sigmoid activation function is used on the output node, with a threshold of 0.5 separating the classes. The sigmoid is defined by Equation 1 and the tanh by Equation 2 where a is the activation.

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

$$f(a) = \tanh(a) \tag{2}$$



(a) (b)

Figure 3 Common activation functions (a) Hyperbolic Tangent (b) Sigmoid

2.3.3 Neural Network Training. Neural network training is an area of considerable interest and is widely discussed in the literature [3, 20, 30]. Thus, a full discussion is impossible here. Only training techniques specifically related to the neural network control of ATM networks will be covered.

The training of a neural network involves two important design decisions: first, the training method; and second, the update rule used to adjust the NNs interconnection weights.

2.3.3.1 Training Methods. Many researchers have explored the area of NN training methods and several innovative training methodologies have been proposed [3, 20, 30]. Hiramatsu, in 1991, [14] developed a two phased approach, combining off-line and on-line training. The NN is first trained off-line to obtain a near optimal set of interconnection weights. The weights are then adjusted, using on-line training, to the optimal values for the given situation. Training data for the on-line training is selected using a leaky pattern table approach. This involves the storing of data resulting in desirable output, as well as data resulting in undesirable output, in a pattern table. These outputs typically represent some GOS parameter such as throughput or cell loss rate. The NN then alternates between a control cycle, when the NN performs the function it is intended to, and a training cycle, when the NN is trained on the contents of the pattern table. This allows the NN to adapt to changing traffic patterns. The problem with this approach is the time required to retrain the NN. The NN is unavailable to perform it's required function while it is being retrained.

Hiramatsu continued to refine real-time training proposing a virtual output buffer method in 1995 [15]. This method routes the communication traffic through a number of virtual buffers, each with a smaller capacity than the actual buffer. By monitoring these virtual buffers, and using their loss and delay parameters as inputs to a NN connection admission controller, he was able to map the performance of the virtual buffers to the actual buffer.

Nordstrom, et al, proposed reinforcement training via construction of hypothetical targets [8]. This approach accumulates, and discounts, two sets of weight vectors. The first set assumes a series of actions will result in a favorable outcome

and the second assumes an unfavorable outcome. Once the result is determined to be favorable or unfavorable, the appropriate weight update vector (the one assuming a favorable outcome or the one assuming an unfavorable outcome) is used to adjust the NN weights. This approach assumes that if a series of actions results in a favorable/unfavorable outcome all actions leading to that outcome were favorable/unfavorable.

One of the most popular training methods, and the one used for this work, is back propagation. Back propagation is a gradient descent technique that seeks to set the interconnecting weights so as to minimize some objective function. The objective function is typically an operation performed on the difference between the desired and actual output. This difference, or error, is then propagated backwards through the net from the output layer to the hidden layer(s). The weights are typically updated immediately following the presentation of a feature vector used for training. Once all available feature vectors to be used for training are presented, an epoch of training has occurred. It typically takes many epochs of training before the MLP can accurately classify previously unseen feature vectors. One danger that must be considered is if the MLP is trained too long, it will memorize the training data and will not be able to generalize to data outside the training set.

2.3.3.2 Update Rule. The second area of interest, closely related to the training technique, is the method used to update the NN's interconnection weights. Traditional approaches use a mean-square-error function, with the error defined as:

$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - y_k)^2 \tag{3}$$

where K is the number of outputs, d_k is the desired output and y_k is the actual output. The learning law for each set of weights, $\mathbf{W}$, is written as:

$$\mathbf{W}^+ = \mathbf{W}^- + \eta\frac{\partial E}{\partial \mathbf{W}^-} \tag{4}$$

where $\mathbf{W}^+$ is the updated weight matrix, $\mathbf{W}^-$ is the old weight matrix and η is the learning rate. A learning rate is a parameter that controls step size and is used to control the rate of convergence. It is typically a constant and is chosen to be a small positive. Innovative variations of this traditional approach include Chen and Leslie's use of an adaptive learning rate [4]. They proposed reducing the learning rate each time the sign of the error changed. This scheme resulted in faster convergence of the weight matrix and a substantial reduction in training time. Ogier [23] proposed the use of an asymmetric error function. In the two class problem Ogier was concerned with, he used the following error function:

$$E = \underbrace{\frac{1}{2}\sum_{k=1}^{K}(d_y - y_k)^2}_{class1} + \underbrace{\frac{1}{2}\sum_{k=1}^{K}(d_y - y_k)^4}_{class2} \tag{5}$$

The asymmetric nature of this error function allows the controller to place more importance on one outcome versus another. In this case, Ogier's objective was to place more importance on rejecting a traffic stream that would result in unacceptable performance (class 2) than accepting a stream that would not adversely affect the network's performance (class 1).

The Levenburg-Marquardt method [6] uses an approximation of Newton's method versus the traditional gradient descent method. The update rule for this method is:

$$\mathbf{W}^+ = \mathbf{W}^- + (\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})^{-1}\mathbf{J}^T e \tag{6}$$

where $\mathbf{J}$ is the Jacobian matrix of derivatives of each error to each weight, μ is a scalar and e is the error vector. After each successful step (a decrease in the error rate), μ is decreased and is only increased when a step results in an increase in the error. As μ increases, this method approaches gradient descent and as μ decreases it approaches the Gauss-Newton method, making it faster and more accurate near an error minima. The problem with this approach is that while it usually converges

in fewer epochs than gradient descent it requires large amounts of memory and may take more actual time to train. This approach is used whenever computer resources permit, otherwise gradient descent will be used.

2.3.4 Feature Set Reduction. A new area of research into the NN approach to traffic control of ATM networks involves the analysis of NN input parameters, or features. The objective here is to reduce the set of features to some optimal minimum in order to reduce the complexity of the NN controller and increase processing speed. Another benefit of a reduced feature set is a fewer number of feature vectors are required to accurately train the MLP because there are fewer nodes and therefore fewer weights in the network (recall rule of thumb from Section 2.3.2.2). This will also reduce the training time. This is a new aspect of the NN design process as it relates to ATM networks and so there is little discussion in the literature. Ogier [23], however, was able to reduce the size of his input vector from 17 elements to 3, with no loss in performance, by preprocessing the feature vector using a linear compression technique. This technique essentially involved the additional of another hidden layer with a linear activation function used to pre-process the full feature vector. The three elements in the reduced feature set were not three of the original 17, but rather a combining of the 17 into three.

Ruck developed a parameter relating the sensitivity of the output to each input feature called saliency [26]. Using this metric it is possible to rank the feature set in order of importance. He showed the saliency metric, which involves the partial derivatives of the NN outputs with respect to the inputs, to be equivalent to criteria based on probability of error. The implementation of Ruck's method can become computationally intensive and was further improved by Steppe [29]. Steppe also provides an excellent overview of feature selection techniques as well as NN architecture design considerations.

A simpler feature selection technique involves the sequential search of the feature set. Forward search is a technique that ranks the features from the most to least important. This approach uses a k nearest neighbor (kNN) classifier and leave one out error estimation. All elements of the feature set are used to train a kNN classifier and then the feature elements are presented to the trained classifier one at a time. The feature element achieving the best classification rate is selected as the most important and the process is repeated using the remaining elements. This continues until there are no features left. Backward search is similar to forward search except all the feature elements except one are presented to a trained kNN classifier to arrive at a classification rate. The element left out of the feature set achieving the highest classification rate is then ranked last. Again this continues until there are no feature elements remaining. Backward search allows the interaction of multiple feature elements to be considered. Forward and backward search combines the previous two search methods. This technique starts with no features and proceeds as if it were a forward search. When two feature elements have been selected, it searches for one to take away, as in a backward search. It continues adding two and taking away one until all the feature elements have been ranked. The forward and backward search may find interdependencies amongst the feature elements that were not discovered by either the forward or the backward search. All three techniques are used for the feature set analysis accomplished in this thesis.

2.4 Feature Space

In order to gain insight into the feature space, it may be desirable to organize the feature vectors by some measure of similarity. One measure of similarity is the Euclidean distances between feature vectors. A self organizing network can learn to detect correlations among input vectors. A competitive learning self organizing network organizes the feature space into clusters by calculating the distance between an input vector and a user defined number of cluster centers. The closest cluster

center wins the "competition" and is the only cluster center to be updated, effectively moving it closer to the input vector. Using this approach the feature space can be segmented without making any a priori assumptions about its distribution.

2.5 Summary

The MLP architecture as well as the treatment of the feature set are important design decisions that impact the overall performance and computational complexity of the MLP link controller. This thesis will examine the MLP architecture, the feature set used to describe the communication traffic and the scheme used to employ the MLP(s) to control congestion in an attempt to determine the combination resulting in optimal performance.

III. Methodology

3.1 Introduction

The previous chapter discussed some of the research conducted in the area of neural network control of ATM communication networks. This chapter describes the network traffic model used to generate simulated communication streams that may be seen in an ATM network. The description of this network traffic model is followed by a presentation of the model used to represent the ATM communications network, or in this case, a single link into the network. The types of simulated traffic streams generated to depict ATM communications traffic flowing through a network are discussed next.

Section three presents the approach used to characterize the traffic streams and extract features that can be presented to a NN used to control link admission. The procedure used to arrive at an appropriate MLP architecture, including the number of hidden nodes, is the topic of section four. Section five describes the alternative architectures that will be used to implement the MLP as a link controller for an ATM network. Finally, the Bayes error rate, which is the minimum average error rate that can be achieved with the available data, is discussed in section six.

3.2 System Model

The ATM communications traffic assumed for this work is a superposition of multiple calls collected at a single source. A call refers to a series of cells containing information to be transmitted. A call may contain video, voice, or data information in an ATM network. The single source can be thought of as representing an entry point, or link, into a distributed ATM network (Figure 4). For any call, the admission criteria is the prevention of congestion on this particular link, which we define to be a cell loss rate (CLR) of less than 0.001. The challenge is to accurately predict whether the admission of the call will cause a congestion situation on the link.

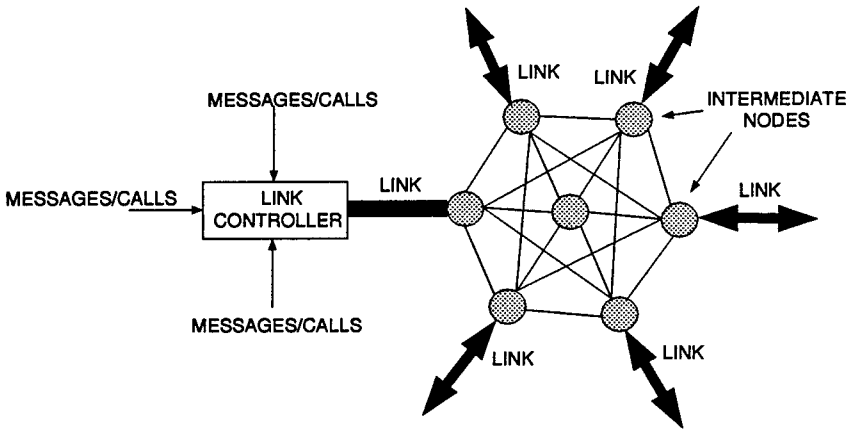


Figure 4 Distributed ATM Network

3.2.1 Traffic Generator. An ATM traffic source can be effectively modeled as an on-off Markov process [27]. This process is characterized by two states. When in the "on" state the source generates traffic at a constant rate and is idle in the "off" state. Traffic streams, representing a collection of calls present in an ATM communication link, were generated using the "Bursty Traffic Source" (BTS) in Designer™. The traffic source is controlled by three primary parameters: Inter-Pulse Time (IPT); Mean Delay Between Bursts (MDBB) and Mean Pulses per Burst (MPPB). Using this terminology the "on" time is a burst and the "off" time is the delay between bursts. IPT is a constant while MDBB is exponentially distributed and MPPB has a geometric distribution.

The BTS produces pulses at a constant rate (IPT) during the duration of the "on" time (MPPB) and is idle during the "off" time (MDBB). The pulses are treated as packets, or cells, for the purpose of running them through the ATM network. An example of the traffic generated is shown in Figure 5.

3.2.2 ATM Network. Only a portion of the ATM communication network is modeled; a single link into a distributed ATM network. Since the link has a finite capacity, it is represented by a simple First-In-First-Out (FIFO) queue with a constant transmission rate. An arbitrary capacity of 1000 cells was chosen as the

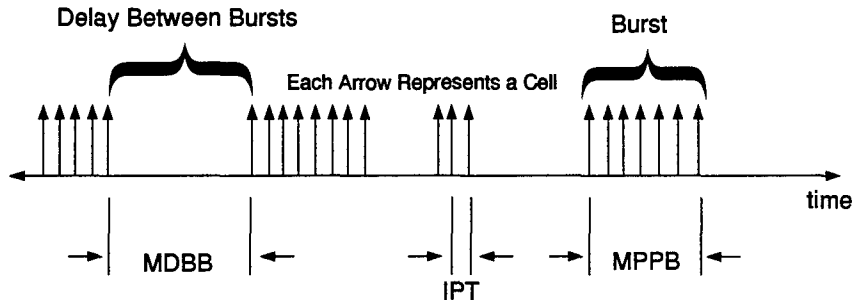


Figure 5 Traffic Stream

queue buffer size. A transmission delay of 6.817 μsec per cell (or packet) is used to represent a link transmission rate of 62.2 Mbps (62.2 Mbps/8 bits per byte = 7.775 e+06 bytes per sec/53 bytes per cell = 146.698 e+03 cells per second = 6.817 μsec to transmit one cell).

The objective of the admission controller is to keep traffic on the link below the congestion level, defined as a CLR of 0.001. To do this it must be able to recognize a traffic stream that causes congestion. The simulated traffic streams represent the traffic on the link resulting when a request for admission has been accepted and is added to previously accepted calls. The job of the NN admission controller is to analyze the stream and decide whether it is acceptable (does not result in a congestion situation) or unacceptable (results in a congestion situation).

3.2.3 Traffic Streams. The parameters used to describe the communication traffic, as discussed in 3.2.1, were varied and streams of one second in length were generated. One objective was to generate streams containing at least 100,000 cells so that an accurate CLR could be calculated. The streams were then fed through the ATM network described in Section 3.2.2 and a cell loss rate was determined by comparing the number of buffer rejects with the total cells generated. If the CLR was less than the congestion level of 0.001 the stream was classified as acceptable and unacceptable otherwise. The three primary parameters (IPT, MDBB, MPPB) were varied until they produced approximately 50% acceptable streams and 50%

unacceptable streams, while still generating at least 100,000 cells per stream. These parameters, along with the buffer size and service delay, were assumed to represent the boundary between the acceptable and unacceptable regions (Table 1).

Table 1 Baseline Parameter Set

Parameter	Baseline Value
Buffer Size	1000
Service Delay	6.817 μsec
Inter-Pulse Time (IPT)	5.863 μsec
Mean Delay Between Bursts	0.005 sec
Mean Pulses per Burst (MPPB)	1467

The three primary parameters were then adjusted away from the boundary, one parameter at a time while holding the others constant. This resulted in 12 parameter sets at various locations near the boundary. Each parameter set represents an adjustment in two directions, one towards the acceptable region and one towards the unacceptable region. For example, in parameter set IPT-1 the inter-pulse time was varied from its baseline of 5.863 μsec (86% of service delay (SD)) to 5.794 μsec (85% of SD) for a move towards the "unacceptable" region and 5.931 μsec (87% of SD) for a move towards the "acceptable" region (Table 2).

The choice of the parameters in Table 2 was determined by trial and error, with the goal of sequentially moving the streams farther into the two decision regions. As each parameter is moved away from the boundary into its respective region, it is expected that an ever increasing percentage of the streams produced will be classified as belonging to that region. This expectation is validated by Table 3.

These parameter sets represent the most difficult streams to classify because they are generated using parameters describing the "boundary" between acceptable and unacceptable streams. If the MLP can successfully classify these streams it should be able to generalize to streams that are deeper into the acceptable or unacceptable regions and therefore easier to classify.

Table 2 Parameter Set Values

Parameter Set	"Acceptable" Stream	"Unacceptable" Stream
IPT-1	5.931 μsec	5.794 μsec
IPT-2	5.999 μsec	5.726 μsec
IPT-3	6.067 μsec	5.658 μsec
IPT-4	6.135 μsec	5.590 μsec
MDBB-1	0.006 sec	0.004 sec
MDBB-2	0.007 sec	0.003 sec
MDBB-3	0.008 sec	0.002 sec
MDBB-4	0.009 sec	0.001 sec
MPPB-1	1367	1567
MPPB-2	1267	1667
MPPB-3	1167	1767
MPPB-4	1067	1867

3.3 MLP Inputs

This section describes the approach taken to characterize the traffic streams into a feature set used as inputs to the MLP link controller, specifically the variation of counts (VOC) method [12,23]. Once the traffic descriptors, or features, are determined the question of an optimal minimum set is explored, that is; is there a smaller number of features that can be used as MLP inputs without a loss in performance.

3.3.1 Feature Set. The traffic streams generated using the parameter sets described in Section 3.2 are divided into equal length segments. Using Ogier's [23] notation we let $N_S(i)$ represent the number of cells contained in segment i and $N_S(i,j)$ the number of cells in segments i through j. Each stream is then represented by a feature vector containing the mean number of cells in each segment (λ_S) and 15 VOCs of exponentially increasing length. The VOC of an interval of m segments

Table 3 Traffic Stream Classification

Parameter Set	% Acceptable in "Acceptable" Streams	% Unacceptable in "Unacceptable" Stream
IPT-1	65.4%	65.4%
IPT-2	75.0%	76.9%
IPT-3	80.8%	86.5%
IPT-4	90.4%	88.5%
MDBB-1	55.8%	65.4%
MDBB-2	59.6%	78.8%
MDBB-3	67.3%	96.2%
MDBB-4	75.0%	100%
MPPB-1	63.5%	63.5%
MPPB-2	71.2%	75.0%
MPPB-3	75.0%	80.8%
MPPB-4	88.5%	82.7%

is defined as:

$$VOC_S(m) = \text{var}(N_S(i+1, i+m)) \tag{7}$$

VOCs are used as the traffic descriptor because they characterize all second order statistics of the stream, they are additive, that is if S is the sum of statistically independent streams S_i then VOC_S is the sum of VOC_{S_i} over S_i, and because moments of interval counts have been shown to accurately predict queuing delay [12, 23].

Ogier [23] used intervals of lengths 2, 4, 8, ...,2^M with $M = 15$. This requires dividing the traffic stream into 32,768 equal length segments. In order to explore the reasonableness of this choice, the streams were also processed using 100 segments and 1000 segments. The interval lengths for these last two cases were determined by

dividing the space between $ln(2)$ and $ln(100)$ (or $ln(1000)$) into 15 equal segments. The interval lengths for each segmentation scheme are shown in Table 4.

Table 4 Interval Lengths

# of	Number of Segments in Interval:														
Segments	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
100	2	3	3	5	6	8	11	14	19	25	33	43	57	76	100
1000	2	3	5	8	12	18	29	45	70	109	169	264	412	642	1000
32,768	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768

Traffic streams were generated using each of the 12 parameter sets and each was processed with the three segmentation schemes. The resulting features were then used to train identical NNs. The minimum achievable training error over all parameter sets was used to determine which segmentation scheme provided the most information to the NN. The results are shown in Figure 6 for each parameter set sub-group. A sub-group is defined as the parameter sets associated with varying one of the three primary parameters. For example the IPT subgroup contains the four parameter sets associated with varying the inter-pulse time while holding the mean delay between bursts (MDBB) and mean pulse per burst (MPPB) constant. Based on these results the 32,768 segment scheme appears reasonable as it has the most consistently decreasing error rate, as would be expected, as the parameter sets are moved farther away from the boundary. The segmentation approach originally proposed by Ogier [23] will be used for the remainder of this thesis.

3.3.2 Feature Set Reduction. Following the choice of a segmentation scheme the question of reducing the size of the feature vector is considered. Recall from the previous section that the full feature vector contains the mean number of packets per segment and the VOC of 15 intervals of exponentially increasing length. The feature vectors of each parameter set were processed using all three feature selection

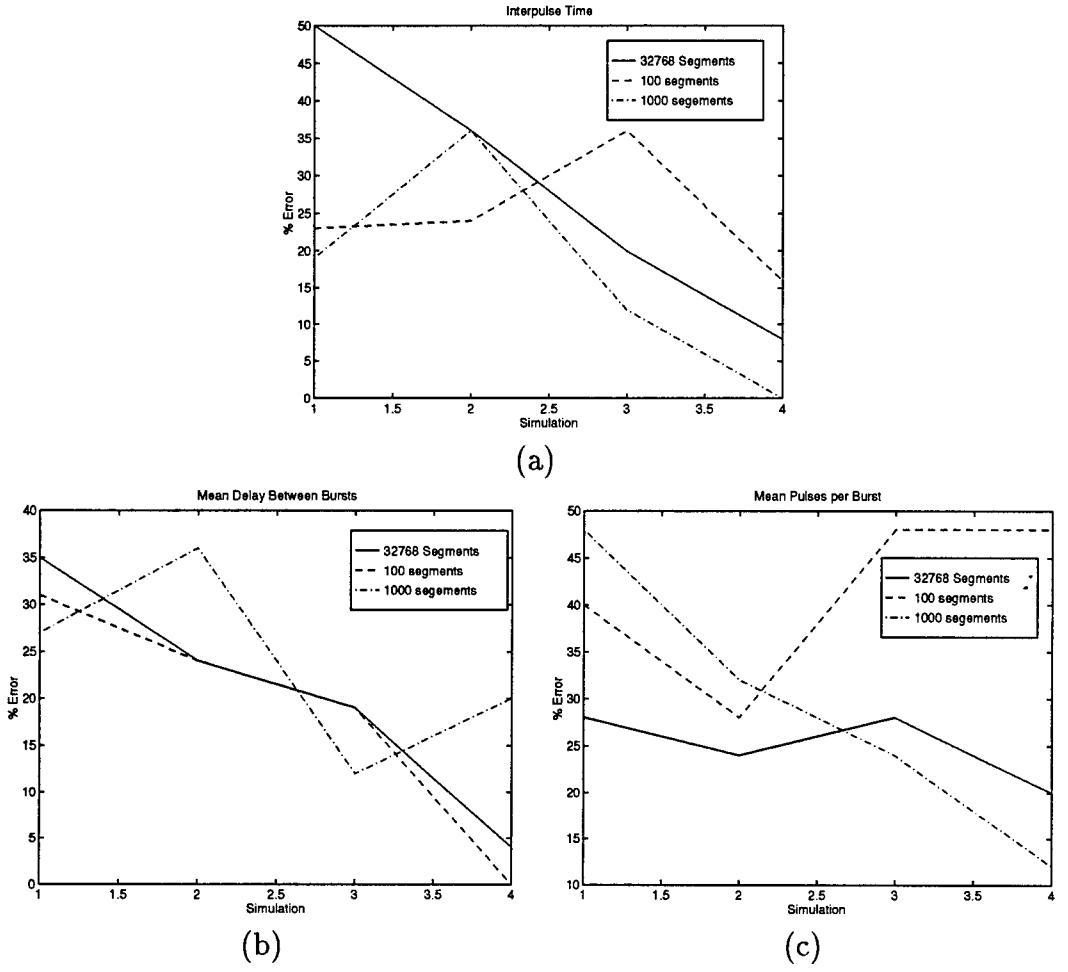


Figure 6 Segmentation scheme error for each of three parameter set subgroups (a) IPT (b) MDBB and (c) MPPB. Plots show the 32,768 segment scheme has the most consistently decreasing error rate as parameter sets are moved away from the boundary.

techniques discussed in section 2.3.4 and the number of times a parameter was ranked in a certain position tabulated. The detailed plot in figure 7 shows the results from feature 1 and feature 7. It can be seen that the distribution from feature 1 is skewed toward the higher ranks, indicating its importance across the parameter sets. The plot from feature seven shows a fairly uniform distribution indicating it does not consistently provide a significant amount of information for classification. The results for all feature vectors are shown in Figure 8. Based on this analysis, it appears the features in the middle of the feature vector provide little or no useful information.

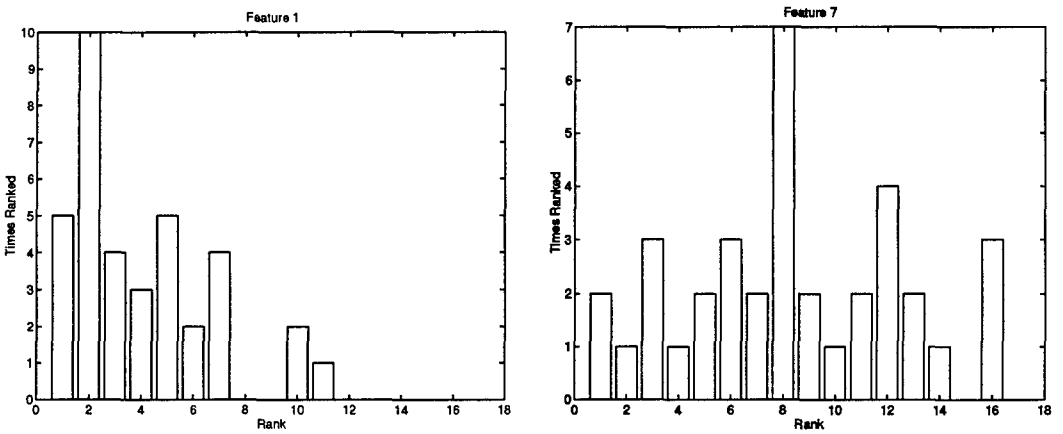


Figure 7 Detailed feature rank plots (a) Feature 1 (b) Feature 7.

In order to verify the notion that the center features in the feature vector provide no useful information, the feature vector is reduced to seven features by removing feature elements 5-13. These elements were choosen for removal after an empirical analysis of the ranking distributions (a more detailed analysis of the feature elements is left for future research). This reduced feature vector (along with the full full feature vector of 16 elements) is used to train and test the MLPs while varying the number of hidden nodes. The choice of feature vector used for continuing trials will be based on the results from the first few trials.

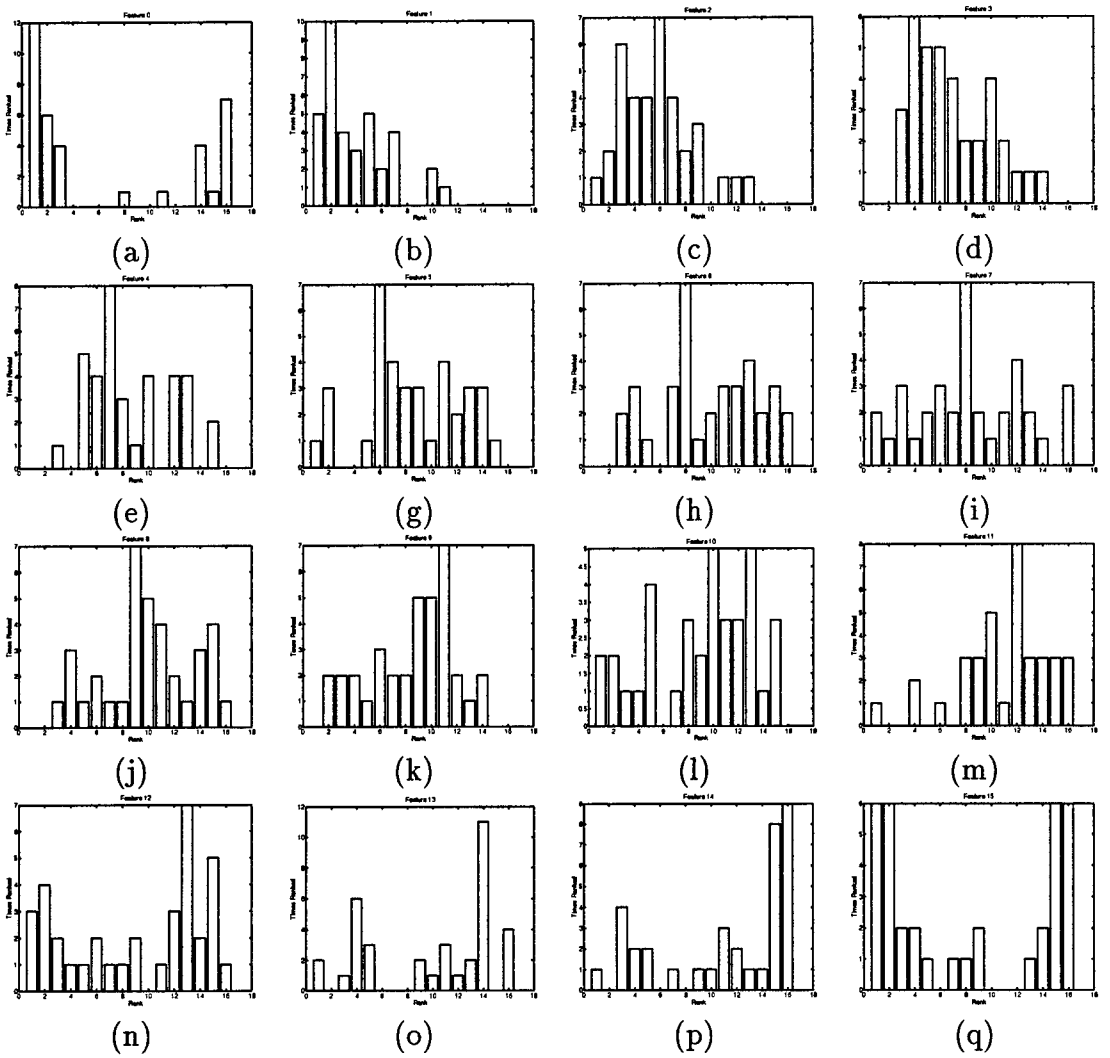


Figure 8 Feature selection combined results from forward search, backward search, and forward and backward search. (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q) Feature 0 through feature 16. Plots suggest little or no information contained in middle of feature vector.

3.4 Neural Network Architecture

The NN architecture design includes decisions involving the number of hidden layers used, the activation function used for the hidden and output layers and the number of hidden nodes used in each hidden layer.

3.4.1 Hidden Layers. As was discussed in Section 2.3.2.1 any non-linear function can be approximated by a MLP with a single hidden layer [5]. For this reason the MLPs used in this effort contain one hidden layer.

3.4.2 Activation Function. The activation function is a non-linear function applied to the weighted sum of inputs to each node (Section 2.3.2.3). Two popular activation functions are the hyperbolic tangent and sigmoid function. Each of these functions approach a lower limit for large negative activations and an upper limit for large positive activations (see Figure 3). The MLPs in this thesis will use the hyperbolic tangent activation function for the hidden layer nodes [19] and the sigmoid as the activation function for the output layer node.

3.4.3 Hidden Nodes. The more complex the function being approximated, the more hidden nodes required to achieve optimal accuracy. In order to investigate the complexity of the feature space, the number of hidden nodes is varied and the MLPs accuracy, as defined by the classification error rate, is used as the measure of performance. This is done in conjunction with the investigation of the feature set size. The optimal combination of feature vector size and number of hidden nodes will be determined from the results.

Once the number of hidden nodes has been decided, an estimate of the accuracy of the MLP can be made using the rule of thumb from Section 2.3.2.2. It should be pointed out that the accuracy discussed here is not an estimate of the classification accuracy expected but rather a level of confidence in the correctness of

the classification error rate. There are 900 feature vectors available for each of the 12 parameter sets described in Chapter *II* for a total of 10,800 feature vectors.

3.5 Link Controller

The methodology for generating the simulated communication traffic, as well as determining the MLP architecture, is complete and so the last piece to investigate is the best way to implement the MLP to achieve optimal results. There are three approaches that will be compared and analyzed: a single MLP, multiple MLPs based on parameter set and multiple MLPs based on clustering.

3.5.1 Single MLP. The first approach (Figure 9) makes no assumptions on the characteristics of the traffic, it only assumes that the training vectors available are representative of the traffic expected to be seen on the link. This scheme uses all 10,800 feature vectors available. The combined set of vectors are randomly mixed and divided into four groups to be used for training and testing. Three groups are used for training the MLP and the remaining group is used to test its performance. This is accomplished four times with each group used as the test group for one of the four cycles. The training vectors are used to train a single MLP for 1000 epochs using back propagation with gradient descent update rule, after which the testing vectors are presented and the classification error rate determined. This cycle is accomplished with the number of hidden nodes varied from one to twenty. It is expected that this approach will give an upper bound, or worst case error rate, for the link controllers considered because of the wide variety of traffic streams it must be able to generalize over.

3.5.2 Parameter Set Based MLPs. The next approach assumes it is possible to determine to which parameter set a stream belongs (Figure 10). The exact nature of this determination is not considered. Each of the 900 vectors associated with the 12 individual parameter sets is used to train a MLP. The MLPs are trained for 500

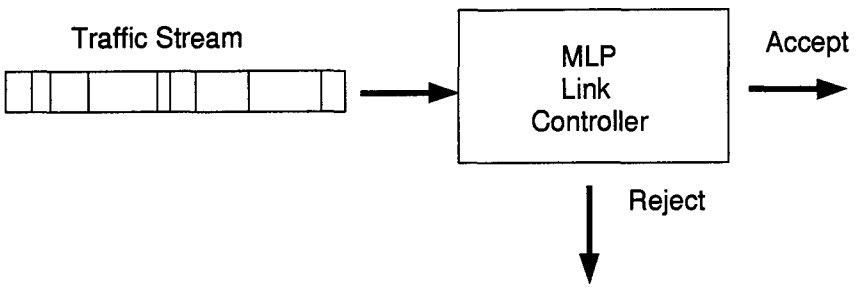


Figure 9 Single MLP trained with all available training data.

epochs, with the Levenberg-Marquardt update rule, using a hold out 25% technique. The 900 vectors are split into groups of 225, with three groups used for training and the remaining group used for testing. This is done four times with a different group of 225 vectors used as the test group each time. This allows all vectors to be used for testing and training and should also allow for a good averaging of error rates. This process is repeated for various numbers of hidden nodes from one to twelve. This approach should give us the best possible performance because it assumes a priori knowledge of which parameter set a particular stream belongs to and this knowledge is then used to switch the stream to an MLP trained only on examples from that parameter set.

In an attempt to reduce the complexity of the link controller as much as possible, an analysis of the cross-generalization abilities of the MLPs trained on each parameter set is explored. The question here is can we reduce the number of MLPs from 12 and maintain an similar level of performance. To investigate this possibility each MLPs is trained and tested on a particular parameter set as described in the previous paragraph. Following training each of the other 11 parameter sets are presented and the MLPs ability to correctly classify the streams is determined. It is anticipated that MLPs trained on feature vectors generated close to the boundary between the acceptable/unacceptable regions will be able to generalize well to feature sets generated farther away from the boundary.

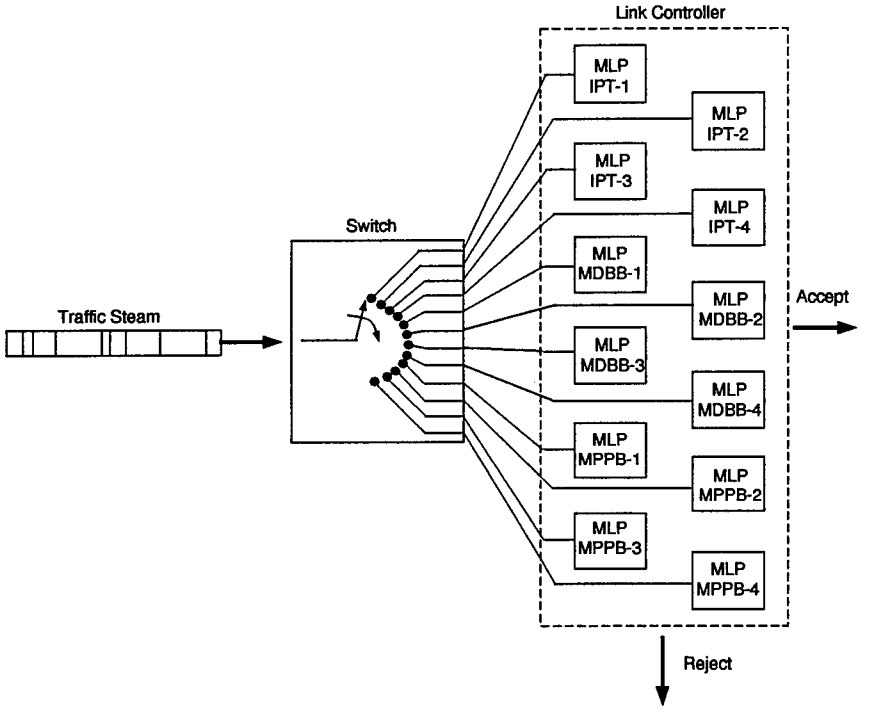


Figure 10 Parameter set based link controller. Multiple MLPs each trained on a separate parameter set. Correct switching to appropriate parameter set MLP is assumed.

3.5.3 Cluster Based MLPs. One final scheme will determine if the feature space clusters itself in a way different from the parameter set assumption of scheme two (Figure 11). In order to answer this question, the entire feature set was presented to a competitive learning algorithm using the Kohonen learning rule with the number of pre-defined cluster centers varied from two to twenty. Once the feature vectors were clustered, a separate MLP was trained and tested only on the vectors in each cluster. Again a hold out 25% technique is used for training and testing as described in previous sections. The total error rate used for comparison purposes is the average of the error rates for each cluster. The advantage of this technique is that the clustering algorithm provides a convenient switch that can be used to direct the communication streams to the MLP trained on similar vectors.

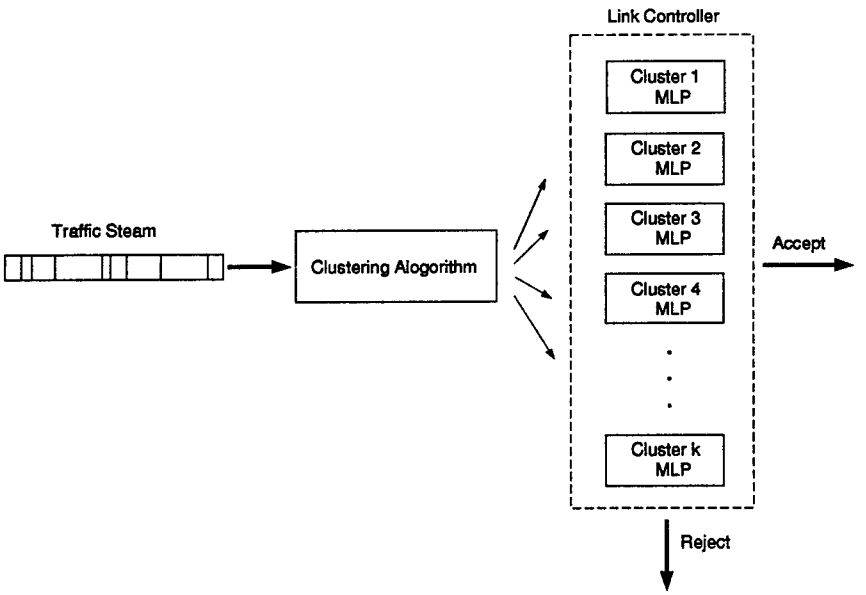


Figure 11 Cluster based link controller. Multiple MLPs are trained on vectors belonging to a given cluster. Clustering algorithm also used as a switch for new traffic streams.

3.6 Bayes Error Rate

The Bayes error rate is the minimum average probability of error. There are two general approaches to estimating the Bayes error rate. The first approach uses a

neural network to estimate the a posteriori probabilities directly from the available data. This is the approach employed above. The second approach entails the use of non-parametric techniques that estimate the density functions directly from the data, without assuming a functional form. The non-parametric technique used in this effort, to compare to the neural network estimate of the Bayes error rate, is the Parzen window technique [25]. The Parzen density estimation is a sum of window functions centered at each sample point [22]. These window functions are evaluated at a sample point of interest and the values are summed to obtain the estimate of the pdf at that point. The problem is how to use a finite set of data samples to test a classifier so that the error rate obtained by the test will accurately predict the error rate expected when future samples are classified. There are two approaches that will give an upper and lower bound on the actual Bayes error rate. The first approach is re-substitution. This method uses all available data to design the classifier and the same complete data set to test it. The error estimation is the percentage of vectors misclassified. Because the same vectors are used to design and to test the classifier, the resulting error estimation is overly optimistic, that is the classifier is not expected to achieve an error rate as low as this estimate when tested with new data. The second method is leave-one-out. This method uses all the data available, except one, to train the classifier and then it is tested with the held out vector. This process is repeated until all vectors have been held out and used for testing. The proportion of vectors misclassified is taken as the error rate estimate. Because each test vector is not used to train the classifier, this technique has significantly less bias. The re-substitution and leave-one-out methods for estimating the error rate can be used to bound the Bayes error rate [10,22] with the re-substitution estimate providing the lower bound and the leave-one-out estimate supplying the upper bound.

3.7 Summary

This chapter described the methodology used to generate the simulated traffic streams that represent communication traffic expected to be seen in an ATM network. Traffic streams were generated from 12 different parameter sets near the border between the acceptable and unacceptable regions. These traffic streams model the most difficult traffic streams to classify because of their proximity to the boundary. Each traffic stream was characterized into a feature vector by dividing it into fixed length segments and determining the mean number of packets per segment and 15 VOCs of intervals of exponentially increasing length. This 16 element feature vector was then analyzed using the forward search, backward search and forward and backward search algorithms.

Once the traffic parameters were determined, and the feature vector analyzed, the MLP architecture was explored. The result was a MLP with a single hidden layer, with an as yet undetermined number of hidden nodes, and an output layer containing one node. The activation function for the hidden layer nodes is the hyperbolic tangent while the sigmoid serves as the output layer node activation function.

The next area considered is the implementation scheme used by the link controller to most accurately classify the traffic streams. Three approaches are presented, training a single MLP to handle all traffic streams, training twelve MLPs corresponding to the twelve parameter sets and finally clustering the available training vectors and training an MLP for each cluster.

Finally a technique for bounding the actual Bayes error rate gives us an objective measure to compare the classification performance of the previous techniques against.

The following chapter implements the link controller schemes, analyzes and compares the results and compares their performance to the optimal Bayes error rate.

IV. Results

4.1 Introduction

The MLP architecture discussed in the previous chapter is now used in three link controller schemes. The ability of these link controllers to correctly classify sample communication streams is analyzed and compared to the Bayes error estimation as bounded by the Parzen window technique [25]. The Bayes error rate is the minimum average error rate any classifier can achieve.

The first link controller configuration is a single MLP used to classify all incoming streams. This configuration will show how well the MLP can classify the feature space with no prior knowledge of the parameters associated with the incoming traffic stream. The second configuration employs twelve separate MLPs, each trained on a separate parameter set, used to classify only communication streams identified as belonging to that particular parameter set. This configuration should result in the best performance because it assumes perfect prior knowledge of the feature vectors' origin. The final link controller configuration employs a promising new method that clusters the feature space with a competitive learning algorithm and trains MLPs based on this clustering. This approach will determine if the feature space naturally arranges itself in ways not associated with the parameter sets and also uses the clustering algorithm to direct incoming traffic to the proper MLP. These link controller schemes are then compared to the optimal Bayes error rate as bounded by the Parzen window technique [25].

4.2 Single MLP Link Controller

To explore the classification and generalization ability of a single MLP all the available data, generated using the twelve parameter sets, is used to train and test the MLP. The combined vectors are randomly mixed to ensure a random distribution from all parameter sets for training and testing. The vectors are then divided into

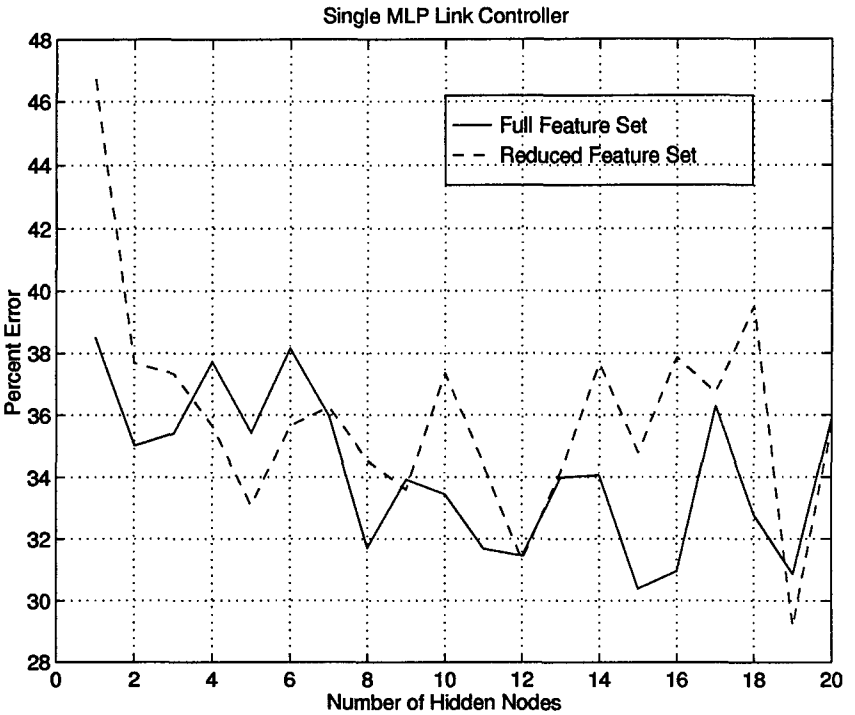


Figure 12 Results from training and testing a single MLP using all available data.

four groups, with three groups used for training the MLP and the remaining group used for testing the performance of the trained MLP. Following the training of the MLP the error rate is determined as the percentage of test vectors misclassified. The MLP was trained for 1000 epochs, using the back propagation method and gradient decent update rule, four times, with each group used for the test group for one of the four runs. This multiple presentation attempts to compensate for any variations resulting from the randomly chosen initial weights, vector presentation order and choice of training and testing groups. This training and testing cycle was conducted using two different feature sets (full set of 16 elements and a reduced set of seven elements) and the number of hidden nodes were varied from one to twenty. The average results for the two feature sets and various numbers of hidden nodes are shown in Figure 12.

Recall that the feature vector was reduced as a result of the search techniques described in Section 3.3.2, specifically vector elements five through thirteen were

removed. As indicated by the plots there is not a great difference between the full feature vector and the reduced feature vector. The error rate over the entire range of hidden nodes considered averaged 34.18% for the full feature vector and 35.95% for the reduced feature vector. The full feature vector achieved a minimum error rate, at fifteen hidden nodes, of 30.4% and and the reduced feature vector had a minimum at 19 nodes of 29.18%. The results are summarized in Table 4.2

Table 5 Single MLP Link Controller Results

	Full Feature Vector	Reduced Feature Vector
Average Error	34.18%	35.95%
Minimum Error	30.40%	29.18%
# of Nodes @ Min Error	15	19

4.3 Parameter Set Based MLP Link Controller

An alternative approach for classifying incoming communication streams assumes that the parameter set the stream was generated from can be determined and the stream can be switched to the appropriate classifier on that basis. For this scheme a separate MLP is trained and tested for each parameter set using only vectors generated using that particular parameter set. In order to reduce the complexity of the link controller the cross generalization ability of the individual MLPs is also investigated. The objective here is to reduce the number of MLPs required without a decrease in classification performance.

4.3.1 MLP for Each Parameter Set. Again each MLP is trained and tested on two different feature vectors as the number of hidden nodes is varied. There are 900 vectors from each parameter set available and the MLP is trained by holding out 225 feature vectors for testing after training on the other 675 vectors for 500

epochs, using back propagation and the Levenberg-Marquardt [6] update rule. This is done four times so that all vectors will be used for training as well as testing. This process was accomplished using the full feature vector as well as the seven element feature vector. The total average error rates, arrived at by averaging the errors of each parameter set MLP while varying the hidden nodes, are shown in Figure 13.

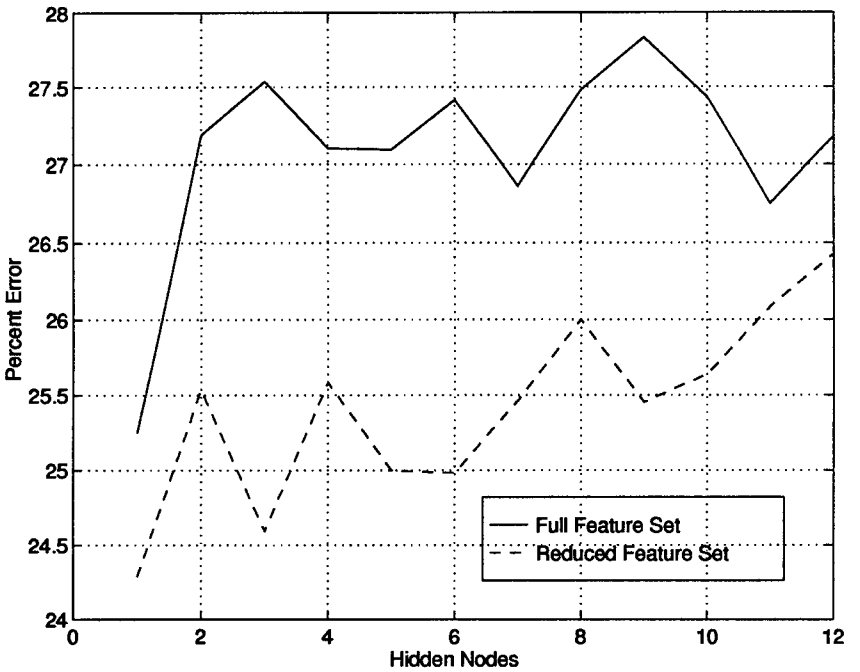


Figure 13 Total error rates for MLPs trained and tested on the individual parameters sets. Total average error rate has a minima at one hidden node, and a seven element feature vector, of 24.29%.

To employ this scheme the feature vector used must be constant for all the parameter set based MLP link controllers, as we must assume that for timely processing the communication stream is presented as a set of features and not as raw data that must be characterized before it can be classified. As can be seen from Figure 13, the seven element feature vector performs better than the full feature vector for total overall error. The same can be said for the individual parameter set based MLPs. An example of this is shown in Figure 14 for the IPT subset of parameter sets. Table 6 lists all the results for both feature sets. Based on these results the

seven element feature vector will be used for comparison purposes because of its superior performance and the added advantage of the reduced complexity associated with a smaller feature vector. The number of hidden nodes used for each MLP can be determined on a case by case basis because it involves a very small differential in processing time. The best results achieved from the previous examination are summarized in Table 7. Averaging these error rate gives us an overall average error rate of 23.70%.

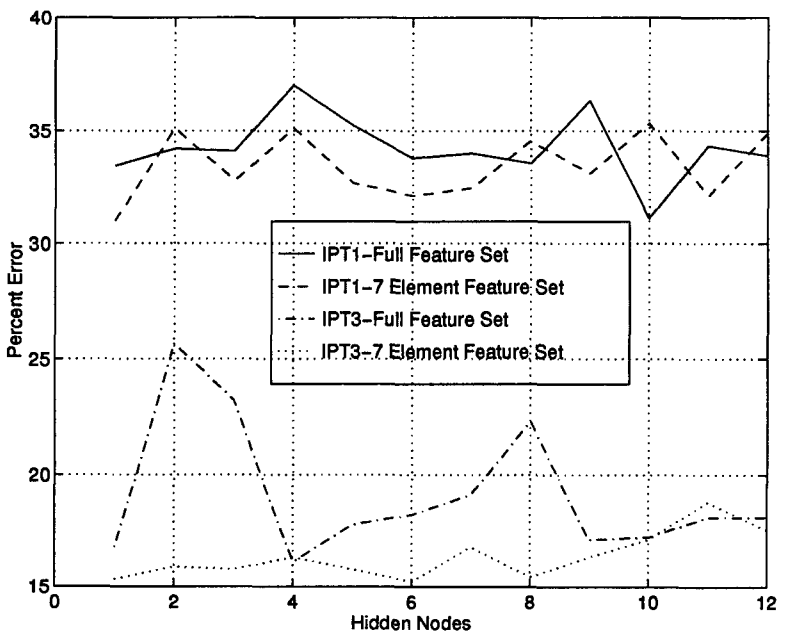


Figure 14 Comparison of feature vector performance in parameter set MLPs. Results here are typical across all parameter sets, the seven element feature vector consistently out-performs the full feature set.

4.3.2 Cross Generalization. The MLPs from the previous section are now used to classify the vectors from the other parameter sets to test their generalization abilities. To do this the MLPs trained on a particular parameter set are presented the feature vectors from the other parameter sets and the classification error rates are determined. As stated earlier the goal here is to eliminate some MLPs without significantly affecting the error rate. The seven element feature vector is used along with a four hidden node MLP. Four hidden nodes were choosen strictly for computa-

Table 6 Complete listing of error rates for parameter set based MLPs. For each parameter set the upper number is the error rate for the full feature vector and the lower number represents the error rate for the seven element feature vector. Bold face indicates best error rates achieved with seven element feature vector.

Para Set	Hidden Nodes											
	1	2	3	4	5	6	7	8	9	10	11	12
IPT1	33.44	34.22	34.11	37.00	35.22	33.78	34.00	33.56	36.33	31.11	34.33	33.89
	31.00	35.11	32.78	35.11	32.67	32.11	32.44	34.56	33.11	35.33	32.11	34.89
IPT2	27.11	28.33	27.67	27.22	30.00	30.67	27.67	27.67	26.89	29.33	27.22	27.56
	24.55	26.00	26.11	26.33	24.89	**24.44**	26.11	26.56	25.67	26.56	25.11	27.00
IPT3	16.78	25.67	23.22	16.11	17.78	18.22	19.11	22.33	17.11	17.22	18.11	18.11
	15.33	15.89	15.78	16.33	15.78	**15.22**	16.78	15.44	16.33	17.11	18.78	17.56
IPT4	11.44	10.89	11.44	12.22	11.67	12.11	10.67	12.78	13.11	13.00	10.44	11.00
	9.78	20.89	10.33	10.44	**9.22**	10.78	10.22	11.67	11.00	10.00	11.00	10.56
MDBB1	38.33	37.00	38.22	38.33	40.33	41.33	38.33	40.11	40.56	41.44	37.89	37.56
	35.33	35.44	35.67	37.89	35.33	36.11	37.78	37.78	39.56	36.67	38.33	38.44
MDBB2	31.67	33.67	35.89	33.00	35.78	35.89	35.22	36.67	37.89	36.33	35.44	36.11
	32.56	33.33	32.67	34.11	34.55	34.55	32.89	35.00	32.78	32.67	33.00	33.44
MDBB3	21.89	26.22	24.56	25.89	24.00	22.44	23.22	24.11	23.33	22.44	24.22	22.67
	20.56	22.00	21.22	23.56	22.00	23.56	23.11	24.67	20.89	23.67	24.44	23.00
MDBB4	18.33	19.33	18.56	19.11	18.33	19.89	18.22	17.11	19.67	19.22	17.56	18.78
	17.56	17.67	18.44	18.78	**17.00**	18.00	19.22	18.22	18.00	20.22	18.11	18.67
MPPB1	33.67	36.33	38.22	40.33	34.22	36.67	40.44	39.44	37.00	39.56	36.00	35.56
	37.33	**32.78**	34.11	35.44	34.67	34.89	34.78	34.22	34.78	33.33	37.44	36.89
MPPB2	31.11	33.11	35.89	33.67	34.00	31.67	31.11	33.11	32.22	32.89	34.22	36.44
	29.67	30.11	**29.11**	30.11	31.56	30.56	32.33	32.67	29.78	32.67	32.22	31.67
MPPB3	22.56	23.22	24.11	24.33	24.11	25.67	26.33	24.67	25.89	27.44	28.78	28.33
	21.78	**21.33**	23.00	22.00	23.89	22.67	23.33	22.67	24.89	22.22	23.00	26.11
MPPB4	16.67	18.22	18.56	19.00	19.67	20.67	18.00	18.22	24.00	19.22	16.78	20.11
	16.00	15.89	**15.89**	16.89	18.44	16.89	16.56	18.56	18.67	17.22	19.44	18.89

Table 7 Error rates for MLPs trained and tested on each parameter set. Error rates based on seven element feature vector. Total average error for this scheme is 23.70%.

Parameter Set	Hidden Nodes	Error Rate
IPT-1	1	31.00%
IPT-2	6	24.44%
IPT-3	6	15.22%
IPT-4	5	9.22%
MDBB-1	1	35.33%
MDBB-2	1	32.56%
MDBB-3	1	20.56%
MDBB-4	5	17.00%
MPPB-1	2	32.78%
MPPB-2	3	29.11%
MPPB-3	2	21.33%
MPPB-4	2	15.89%

tional efficiency. The MLPs are trained on one parameter set for 500 epochs, using back propagation and the Levenberg-Marquardt update rule, then tested against the others for five cycles. The results are shown in Table 8.

Using the error rates in Table 8, the average error over all MLPs, assuming all twelve are used, is 23.11%. The best error rates achieved are highlighted in bold and indicate the MLP trained on parameter set IPT1 classified the feature vectors from parameter sets IPT2 and IPT3 better than the MLPs trained on those parameter sets. The same effect can be seen with the MLP trained on parameter set MDBB2, it also classified the feature vectors from MDBB3 more accurately than the MLP trained on that parameter set. The optimal results are obtained by eliminating the MLPs associated with parameter sets IPT2, IPT3 and MDBB3. This reduces the number of MLPs by 25% while decreasing the error rate to 22.06%. We have therefore accomplished the objective of reducing the link controller complexity without decreasing its performance and in fact, the performance increased.

Table 8 Cross-generalization results with best error rate on given parameter set indicated in bold. Using strictly best error rates, the overall average error rate is 22.06%

Train Set	Testing Set											
	IPT1	IPT2	IPT3	IPT4	MDBB1	MDBB2	MDBB3	MDBB4	MPPB1	MPPB2	MPPB3	MPPB4
IPT1	**29.11**	**24.78**	**15.47**	9.36	35.04	32.80	38.38	48.38	32.24	29.16	23.40	18.44
IPT2	35.31	32.18	27.91	25.13	38.13	39.36	49.29	63.71	36.58	35.96	33.62	31.20
IPT3	32.91	28.58	20.58	17.04	36.58	36.42	39.93	51.11	35.20	33.04	29.00	25.27
IPT4	34.42	26.16	16.64	**7.40**	36.89	33.58	31.60	47.60	36.71	33.33	28.42	25.89
MDBB1	38.69	43.64	43.49	42.78	**32.47**	31.16	23.91	22.40	35.07	33.47	28.93	22.89
MDBB2	35.44	32.47	28.80	26.11	35.84	**28.47**	**20.78**	17.56	36.69	35.93	34.80	29.33
MDBB3	43.38	47.22	52.20	57.11	38.87	32.42	20.84	16.87	41.56	40.84	38.36	33.56
MDBB4	42.67	36.78	33.89	33.60	44.98	42.47	25.64	**15.18**	46.18	45.93	48.58	47.58
MPPB1	34.29	30.22	26.96	25.96	35.58	31.16	30.49	38.33	**30.16**	27.80	22.20	16.91
MPPB2	34.78	38.60	40.00	43.35	36.62	32.27	24.24	19.67	33.93	**26.56**	21.89	16.22
MPPB3	36.67	37.71	40.11	40.78	36.96	32.58	23.24	18.80	35.04	29.04	**20.04**	15.89
MPPB4	35.29	27.36	21.11	17.60	37.02	32.18	25.73	40.44	32.89	28.49	21.60	**14.27**

4.4 Cluster Based MLP Link Controller

The final scheme to be explored for the link controller is based on clustering the input space using a competitive learning algorithm with a pre-determined number of clusters. The advantage of this approach is that it allows the clustering algorithm to be used as a switch for communication streams requiring classification. For this approach all available seven element feature vectors are presented to a competitive learning clustering algorithm for 10,000 epochs. The feature vectors assigned to a particular cluster are then used to train and test a four hidden node MLP for 500 epochs using back propagation and the Levenberg -Marquardt update rule. 75% of the vectors are used for training and 25% used for testing. Again this cycle is repeated 4 times so all the vectors will be used for training as well as testing. The feature vectors are clustered for each pre-defined number of clusters from two through twenty and each time the feature space is clustered, MLPs are trained and tested as described above. The average total error rates (average error rate over all MLPs) versus number of clusters for this scheme are shown in Figure 15.

The results show that the best performance is achieved with the fewest number of clusters. Specifically an error rate of 23.17%, occurring with 4 clusters, is the lowest rate observed. While this error rate is slightly higher than the optimal rate achieved training MLPs on the individual parameter sets (22.06%)it has two distinct advantages over that approach. First there are less than half the number of MLPs required (four vs. nine) resulting in a simpler implementation. Second, and most importantly, the clustering algorithm provides a built in switch that can be used to direct the communication streams to the appropriate MLP classifier.

4.5 Bayes Error Rate

The bounds on the Bayes error rate are now calculated using the Parzen window technique [25] (see Section 3.6). Recall that the Bayes error rate can be bounded by the re-substitution (lower bound) and leave-one-out (upper bound) methods.

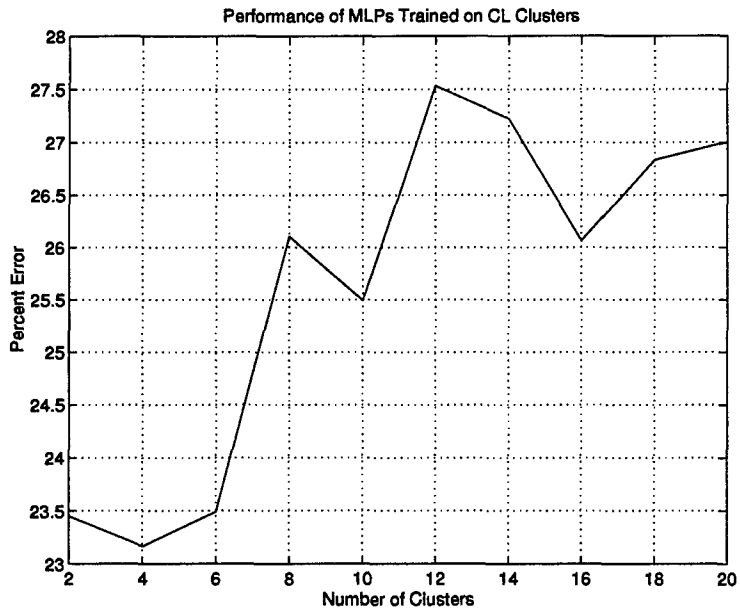


Figure 15 Average total error rates for MLPs trained on the basis of competitive learning clustering algorithm. Best performance is achieved with four clusters, 23.17% error rate.

The error rate was calculated for both these techniques in conjunction with the Parzen window method while varying the parameter h. The parameter h controls the "spread" of the window function [22]. The results of this examination are shown in Figure 16. Based on this figure, it appears that the upper and lower bounds are converging to a Bayes error rate of approximately 20-21%. This compares favorably with parameter set based MLPs error rate of 22.06% and the cluster based MLPs error rate of 23.17% while the single MLP error rate lags at 29.18%.

4.6 Summary

This chapter has presented three techniques for use of MLP(s) to classify communication streams in an attempt to control congestion in a link to an ATM network. The results indicate that making no assumptions about the distribution of the input space and training a single MLP to handle all incoming traffic resulted in a classification error rate of 29.18%. When perfect prior knowledge of a communication

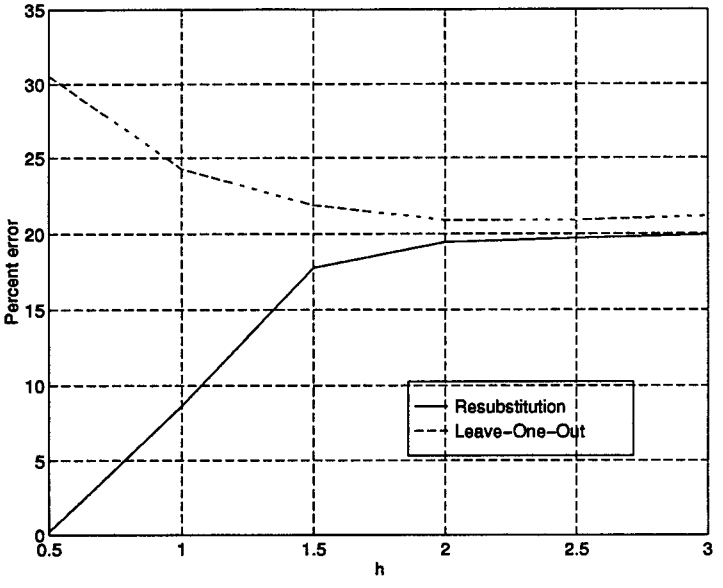


Figure 16 Bounded Bayes error rate estimation calculated using the Parzen window technique. Figure indicates the Bayes error rate to be within 20-21%.

streams distribution is assumed an error rate of 22.06% was achieved but the problem of how to determine which MLP to use remains. The final technique employed a clustering algorithm to gather similar feature vectors, and train MLPs based on this clustering. This approach has the advantage of using the clustering algorithm as a switch, directing incoming communication steams to the appropriate MLP, following the training period. A classification error rate of 23.17% was observed using this method. Finally the bounds on the Bayes error rate was calculated using the Parzen window technique. The result of this calculation indicates the Bayes error rate lies between 20 and 21 percent. This result shows that by dividing the feature space into groups based on a particular criteria, in this case parameter sets and Euclidean distance, the average error rate approaches the Bayes rate. This suggests that the link controller schemes employing these techniques are near optimal.

V. Conclusion

5.1 Introduction

The primary objective of this thesis was to investigate the best neural network approach to classifying simulated ATM traffic streams in order to control congestion on a link to an ATM network. A single hidden layer multi-layer perceptron was used to classify simulated communication streams in three different link controller schemes. The number of nodes in the hidden layer was examined, as well as the size of the feature set used. An objective way to measure the performance of the three link controller schemes, by comparing results to the Bayes error rate, was also introduced. This chapter summarizes the results of this effort and draws some conclusions based on these results.

5.2 Summary of Results

5.2.1 Single MLP. For this link controller approach all incoming communication streams requiring classification are handled by a single MLP. The only assumption about the distribution of the communication traffic made for this scheme is that the communication streams used for training the MLP are representative of the future traffic streams expected. The number of hidden nodes in the MLP were varied and the trial conducted using two feature sets, a full feature set as described in Section 3.3.1 and a reduced feature set containing seven elements using the techniques of Section 3.3.2. The results from this trial indicated that the seven element feature vector performed almost as well as the full 16 element feature vector. The average error over all hidden node variations was 34.18% for the full feature set and 35.95% for the reduced feature set. The seven element feature vector achieved a minimum average error rate of 29.18% compared to 30.40% for the full feature set. These averages were over four runs using a hold out 25% technique, with random presentation order and initial weights.

5.2.2 Parameter Set based MLPs. This link controller technique assumes it is possible to determine the origin of a communication stream, that is, which parameter set was used to generate the stream. This information can then be used to direct the communication stream to a MLP trained only on communication streams from that particular set. As part of this trial the generalization abilities of the MLPs were also examined. Here we looked at the ability of a MLP trained on one parameter set to classify streams from another parameter set. The objective was to reduce the number of MLPs without a significant decrease in overall performance. Again this trial used the full as well as the seven element feature vectors. This trial demonstrated, using all twelve MLPs, an average error rate of 23.11% was achievable with the seven element feature vector. The ability of the seven element feature vector to out-perform the full feature vector was evident in this trial. The best performance achievable by the full feature set was 24.72%. Only the seven element feature vector was used for the generalization experiment and the results showed that the the number of MLPs can be reduced to nine (a 25% reduction) with the overall error rate actually improving to 22.06%.

5.2.3 Cluster Based MLPs. The problem of not having a practical switch to direct the communication streams to the appropriate MLP is solved with this link controller scheme, which uses a competitive learning clustering algorithm to sort the feature space. This trial sorted the feature space into various numbers of clusters, trained MLPs based on the feature vectors belonging to those clusters and determined an overall error rate. It is assumed that the communication streams used for the original training, and determination of the cluster centers, are representative of future communication streams that will be processed by this scheme. Based on this, the same clustering algorithm can be used to determine which cluster center is closest to the communication stream requiring classification and an MLP, trained only on other communication streams assigned to the same cluster, is used to classify the stream. The results of this trial indicated that the feature space can be sorted

into a relatively few number of cluster centers to achieve optimal results. Specifically, the best performance, an error rate of 23.17%, was seen with four cluster centers. This error rate compares favorably with the error rate seen in the parameter set based MLPs of 22.06%

5.2.4 Bayes Error Rate. The last step in this research was to bound the best performance that can be expected, with the given data. This number is given by the Bayes error rate. Although the MLPs of the previous trials approximate the Bayes error rate, there was no way to tell how good that approximation was. Using the Parzen window technique, with re-substitution and leave-one-out, the Bayes error rate was bounded. The results from this analysis indicated that the Bayes error rate lies between 20 and 21 percent. The Bayes error rate represents the minimum average error rate any classifier can achieve. This indicates that by dividing the feature space, either by the parameter set assumption or by using a clustering algorithm, we can classify incoming streams at a near optimal rate.

5.3 Conclusion

The results from this research lead to a number of interesting conclusions:

1. The 16 element feature vector can be reduced to less than half with no significant reduction, and in some cases an increase, in performance.
2. The performance of a single MLP, trained to handle a wide variety of communication streams, can be significantly improved by dividing the feature space in some fashion.
3. A promising approach to dividing the feature space employs a clustering algorithm that serves the dual purpose of dividing the feature space and switching the incoming traffic to the appropriate classifier.
4. The performance of the cluster based MLP is a near optimal approximation of the true Bayes error rate.

These conclusions indicate that the objective of designing a link controller that can adequately control congestion on a single link to an ATM network has been met. An optimal MLP architecture with a minimum feature vector and a realistic link controller scheme was determined that achieved an error rate that closely approximates the Bayes error rate.

Appendix A. Traffic Stream Generation

A.1 Introduction

This appendix describes the simple model used to generate and classify the simulated traffic streams, using BONeS® DESIGNER™ block oriented network simulator. The model consists of two main parts, the traffic generator and the network model as shown in Figure 17. The two parts are considered separately in the following sections. All of the parameters shown are set when a simulation is run and are described in the section dealing with the part of the model where they originate.

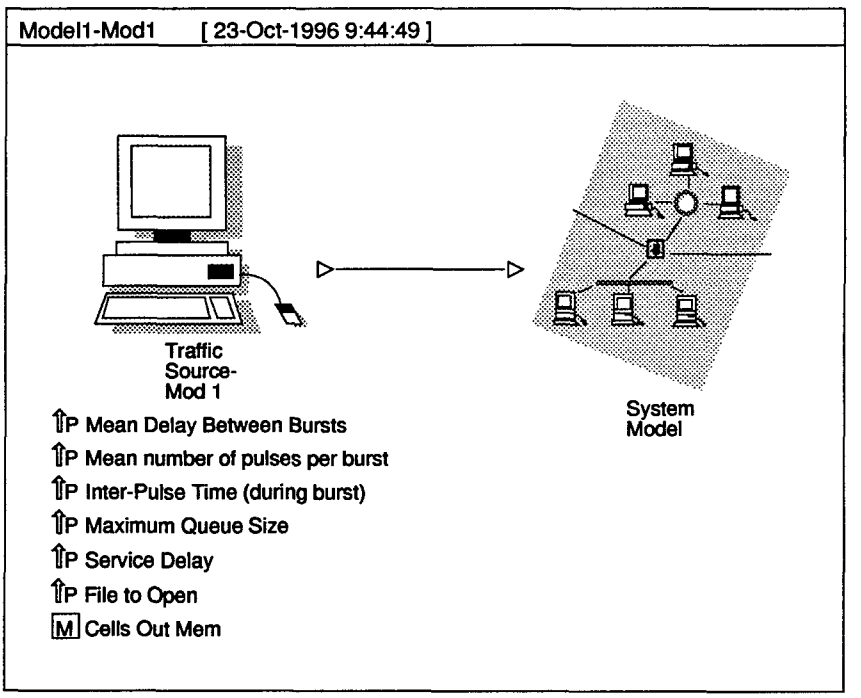


Figure 17 Top level model used for generating and classifying simulated traffic streams

A.2 Traffic Generator

The traffic generator, shown in Figure 18, generates pulses using the Bursty Pulse Train (BPT) block. This block generates pulses with constant Inter-Pulse

times (IPT), geometrically distributed pulses per burst and exponentially distributed delay between bursts. These parameters are adjusted by the user by specifying the IPT and the mean for the geometric and exponential distributions. The pulses coming from the BPT are converted into 53 byte cells and given a time stamp and sequence number.

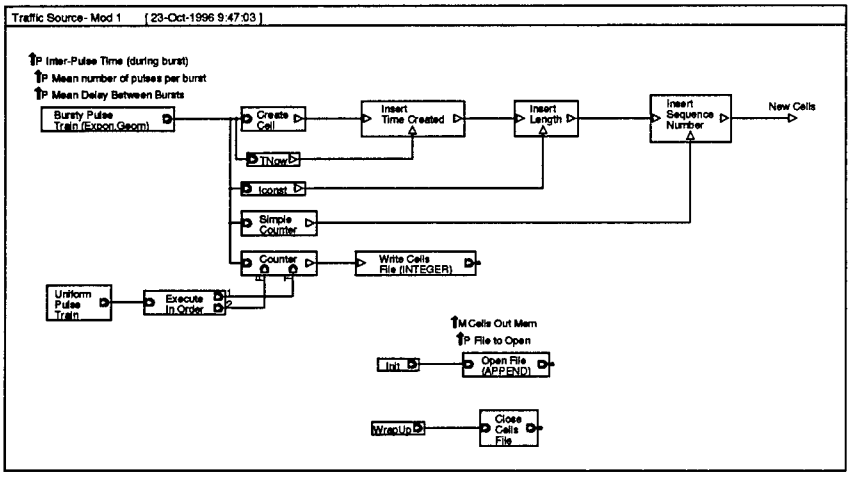


Figure 18 Block diagram of traffic source.

The uniform pulse train is used in conjunction with the counter to determine how many cells are generated during a given time period. The time period is determined by how long the data stream is and how many equal length segments are desired. Recall that for this thesis the data streams are one second long and 32,768 segments are desired. The uniform pulse train is therefore set to fire every 30.52 μsec $(1/32,768)$ triggering the counter to output the number of cells generated in that time period and then reset.

The final blocks in this part simply open a file to write the number of cells in a segment to, as well as the number of rejects resulting from the network model discussed in the next section. The "Cells Out Mem" is a pointer to the open file stream that can be used by other file access modules to write to this stream after it has been opened. In this case it will used to write the number of rejects from the system model.

A.3 Network Model

The network modeled in this thesis is an ATM network, but more specifically the entrance, or link, to an ATM network. Since the link has a finite capacity, it is represented by a simple First-In-First-Out (FIFO) queue with a constant transmission rate as represented by the two blocks in Figure 19.

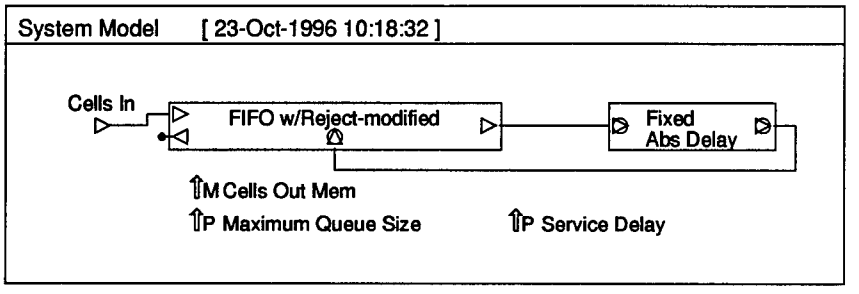


Figure 19 Block diagram of network model, or in this case a single link to an ATM network.

The FIFO queue is characterized by its maximum queue size. When the queue is full of cells awaiting service any newly arriving cells are rejected. It is these rejected cells, along with total cells generated, that is used to determine the cell loss rate (CLR). The queue cannot service, or transmit, more than one cell at a time so the time required to service a cell, represented by the fixed absolute delay, is directly related to the transmission rate of the channel.

Figure 20 shows the detailed breakdown of the FIFO queue to show how the number of rejects are output to the data file. The Final Queue Statistics block outputs various queue statistics at the completion of a simulation. The number of rejects occurring during the simulation is selected and written to the output data file. The last line in the data file is therefore the number of rejects.

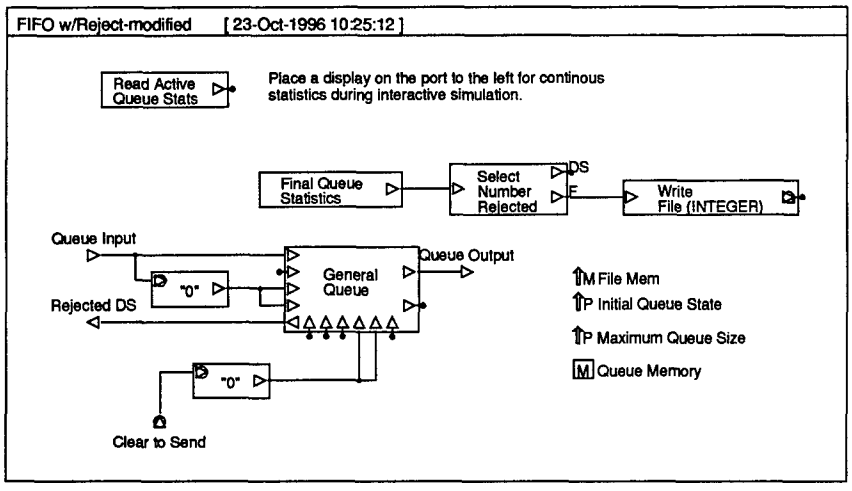


Figure 20 Detail of FIFO queue to show how number of rejects occurring during simulation are collected.

Appendix B. Link Controller and Bayes Error Code

This appendix contains the MATLAB® code used to implement the three link controller schemes. The code for the Bayers error bounding is also included with thanks to Curtis Martin. All functions and subfunctions are included.

B.1 Single MLP

This code implements the single MLP link controller. The MLP is trained and tested using a hold out 25% technique.

onenn.m

```
function [AveError, stdev]=onenn[train,target,minnodes,maxnodes]

% This function will train and test a single MLP using back propagation, 1000 epochs
%  using the hold out 25% technique
%
%      Pass:  train = feature vectors
%             target = classification vector
%             minnodes = minimum number of hidden layer nodes
%             maxnodes = maximum number of hidden layer nodes;
%                    Hidden nodes varied from minnodes to maxnodes
%      Return: AveError = average classification errors per # of hidden nodes
%             stdev = standard deviation of average classification error

%  Randomly mix feature and target vectors

mix = randperm(size(train,2));
train=train(:,mix);
target=target(:,mix);

% Train NN

for j=minnodes:maxnodes

             fprintf(1,'Processing  %d nodes\n',j)
             [error(j), stdev(j)]=holdoutnbp(train,target,2700,j);
end; %for j
```

holdoutnbp.m

```
function [errmean, errstd]=holdoutnbp(trainvect,targvect,numholdout,l1)
%       JEL 100896
%       Performs Hold-One-n training of MLP holds out n vectors for testing
%               after training on remaining vectors using BP Method where n is
%               an integer multiple of total number of vectors in p.  This
%               function holds out n vectors [(number of vectors in p)/n] times
%               and averages the error
%       Display Frequency, Maximum Epochs, SSE Goal and
%               # of Hidden Layers parameters set in function
%       Pass:   trainvect = Training vector
%               targvect = Target vector
%               numhold out = Number of vectors to hold out for testing
%               l1 = # of Neurons in Hidden Layer
%       Return: errmean = Mean of test errors
%               errstd = Standard Deviation of Training Errors

global trainvect targvect

%       Set NN parameters

df = 0;         % Display Frequency
me = 1000;      % maximum epochs
eg = .02;       % SSE Goal


numvect=size(trainvect,2);
numiterations=numvect/numholdout;

% Determine first training set
ptrain=trainvect(:,numholdout+1:numvect);
ttrain=targvect(numholdout+1:numvect);

% train and test
for i=1:numiterations
        ep=0;
        start=numholdout*(i-1)+1;
        while ep<25
                [w1,b1,w2,b2,ep,tr]=bprop(ptrain,ttrain,df,me,eg,l1);
        end; %while
        ptest=trainvect(:,start:numholdout*i);
        ttest=targvect(start:numholdout*i);
        testresult=simuff(ptest,w1,b1,'tansig',w2,b2,'logsig');
        error(i)=classerr(testresult,ttest);
        if i==numvect-1
                ptrain=trainvect(:,1:i*numholdout);
                ttrain=targvect(1:i*numholdout);
        elseif i==numvect
                xxx=0;  % if
        else
```

```
            ptrain=[trainvect(:,1:i*numholdout),trainvect(:,((i+2)*numholdout)+1:numvect)];
            ttrain=[targvect(1:i*numholdout),targvect(((i+2)*numholdout)+1:numvect)];
        end;  % if
end;  %for

errmean=mean(error);
errstd=std(error);
```

---

bprop.m

```
function [w1,b1,w2,b2,ep,tr] = bprop(p,t,df,me,eg,l1)
%       JEL 102496
%       Application of back propagation with learning rate and momentum
%         using a single tanh hidden layer MLP and sigmoid output layer
%               to Comm Stream problem.
%       Pass:  p = Training vector
%              t = Target vector
%              l1 = Number of neurons in hidden layer
%              df = Display frequency
%              me = Maximum # of epochs
%              eg = SSE goal
%       Returns: w1,b1,w2,b2
%              ep = Number of epochs
%              tr = error vector

% Initialize weights
[w1,b1,w2,b2]=initff(p,l1,'tansig',t,'logsig');  % Initiaize weights

% Set training parameters
tp=[df me eg]; % Training parameter vector

% Train the net
[w1,b1,w2,b2,ep,tr]=trainbpx(w1,b1,'tansig',w2,b2,'logsig',p,t,tp);
```

---

classerr.m

```
function error = classerr(test,target)
%       JEL 090396
%       Determines the average classification error of a NN 'logsig' output
%       Pass:  test = output vector from NN with a logsig output layer
%                     activation function
%              target = correct classification of input vectors
%       Return: error = Percent of input vectors mis-classified

numvect=length(test);

for k=1:numvect
        if test(k)<0.5
```

```
            test(k)=0;
        else
            test(k)=1;
        end; % if
end; % for k

error=nnz(test-target)/numvect;
```

B.2 Parameter Set Based MLPs

This code trains 12 separate MLPs based on the parameter set used for their generation. The hold out 25% technique is again used to get average error rate.

pbmlp.m

```
%       JEL 100896
%       This script trains 12 separate MLPs based on the parameter sets and two different subsets
%       of the feature vector (16 and 7 feature elements)

clear;
close all;

% Load all vectors
load ipts1train900.dat;
load ipts1target900.dat;
load ipts2train900.dat;
load ipts2target900.dat;
load ipts3train900.dat;
load ipts3target900.dat;
load ipts4train900.dat;
load ipts4target900.dat;
load mdbbs1train900.dat;
load mdbbs1target900.dat;
load mdbbs2train900.dat;
load mdbbs2target900.dat;
load mdbbs3train900.dat;
load mdbbs3target900.dat;
load mdbbs4train900.dat;
load mdbbs4target900.dat;
load mppbs1train900.dat;
load mppbs1target900.dat;
load mppbs2train900.dat;
load mppbs2target900.dat;
load mppbs3train900.dat;
load mppbs3target900.dat;
load mppbs4train900.dat;
load mppbs4target900.dat;
disp('Load Complete')
```

```
% mix feature vectors
mix=randperm(900);
ipts1train=ipts1train900(mix',:);
ipts2train=ipts2train900(mix',:);
ipts3train=ipts3train900(mix',:);
ipts4train=ipts4train900(mix',:);
ipts1target=ipts1target900(mix');
ipts2target=ipts2target900(mix');
ipts3target=ipts3target900(mix');
ipts4target=ipts4target900(mix');
mdbbs1train=mdbbs1train900(mix',:);
mdbbs2train=mdbbs2train900(mix',:);
mdbbs3train=mdbbs3train900(mix',:);
mdbbs4train=mdbbs4train900(mix',:);
mdbbs1target=mdbbs1target900(mix');
mdbbs2target=mdbbs2target900(mix');
mdbbs3target=mdbbs3target900(mix');
mdbbs4target=mdbbs4target900(mix');
mppbs1train=mppbs1train900(mix',:);
mppbs2train=mppbs2train900(mix',:);
mppbs3train=mppbs3train900(mix',:);
mppbs4train=mppbs4train900(mix',:);
mppbs1target=mppbs1target900(mix');
mppbs2target=mppbs2target900(mix');
mppbs3target=mppbs3target900(mix');
mppbs4target=mppbs4target900(mix');

for u=1:4
        num=num2str(u)
        eval(['clear ipts' num 'train900 ipts' num 'target900 mdbbs' num 'train900 mdbbs'
              num 'target900 mppbs' num 'train900 mppbs' num 'target900'])
end; % for u

% generate reduced feature sets
ipt1rs7trn=[ipts1train(:,1:4) ipts1train(:,14:16)]';
ipt2rs7trn=[ipts2train(:,1:4) ipts2train(:,14:16)]';
ipt3rs7trn=[ipts3train(:,1:4) ipts3train(:,14:16)]';
ipt4rs7trn=[ipts4train(:,1:4) ipts4train(:,14:16)]';
mdbb1rs7trn=[mdbbs1train(:,1:4) mdbbs1train(:,14:16)]';
mdbb2rs7trn=[mdbbs2train(:,1:4) mdbbs2train(:,14:16)]';
mdbb3rs7trn=[mdbbs3train(:,1:4) mdbbs3train(:,14:16)]';
mdbb4rs7trn=[mdbbs4train(:,1:4) mdbbs4train(:,14:16)]';
mppb1rs7trn=[mppbs1train(:,1:4) mppbs1train(:,14:16)]';
mppb2rs7trn=[mppbs2train(:,1:4) mppbs2train(:,14:16)]';
mppb3rs7trn=[mppbs3train(:,1:4) mppbs3train(:,14:16)]';
mppb4rs7trn=[mppbs4train(:,1:4) mppbs4train(:,14:16)]';

% peform hold-out-n training and testing
disp('Start training')
for i=1:12
        fprintf(1,'ipt1 %d nodes',i)
```

```
[ipt1err(i) ipt1std(i)]=holdoutn(ipt1train,ipt1target,225,i);
[ipt1err7(i) ipt1std7(i)]=holdoutn(ipt1rs7trn,ipt1target,100,i);

fprintf(1,'ipt2 %d nodes',i)
[ipt2err(i) ipt2std(i)]=holdoutn(ipt2train,ipt2target,225,i);
[ipt2err7(i) ipt2std7(i)]=holdoutn(ipt2rs7trn,ipt2target,225,i);

fprintf(1,'ipt3 %d nodes',i)
[ipt3err(i) ipt3std(i)]=holdoutn(ipt3train,ipt3target,225,i);
[ipt3err7(i) ipt3std7(i)]=holdoutn(ipt3rs7trn,ipt3target,225,i);

fprintf(1,'ipt4 %d nodes',i)
[ipt4err(i) ipt4std(i)]=holdoutn(ipt4train,ipt4target,225,i);
[ipt4err7(i) ipt4std7(i)]=holdoutn(ipt4rs7trn,ipt4target,225,i);

fprintf(1,'mdbb1 %d nodes',i)
[mdbb1err(i) mdbb1std(i)]=holdoutn(mdbb1train,mdbb1target,225,i);
[mdbb1err7(i) mdbb1std7(i)]=holdoutn(mdbb1rs7trn,mdbb1target,225,i);

fprintf(1,'mdbb2 %d nodes',i)
[mdbb2err(i) mdbb2std(i)]=holdoutn(mdbb2train,mdbb2target,225,i);
[mdbb2err7(i) mdbb2std7(i)]=holdoutn(mdbb2rs7trn,mdbb2target,225,i);

fprintf(1,'mdbb3 %d nodes',i)
[mdbb3err(i) mdbb3std(i)]=holdoutn(mdbb3train,mdbb3target,225,i);
[mdbb3err7(i) mdbb3std7(i)]=holdoutn(mdbb3rs7trn,mdbb3target,225,i);

fprintf(1,'mdbb4 %d nodes',i)
[mdbb4err(i) mdbb4std(i)]=holdoutn(mdbb4train,mdbb4target,225,i);
[mdbb4err7(i) mdbb4std7(i)]=holdoutn(mdbb4rs7trn,mdbb4target,225,i);

fprintf(1,'mppb1 %d nodes',i)
[mppb1err(i) mppb1std(i)]=holdoutn(mppb1train,mppb1target,225,i);
[mppb1err7(i) mppb1std7(i)]=holdoutn(mppb1rs7trn,mppb1target,225,i);

fprintf(1,'mppb2 %d nodes',i)
[mppb2err(i) mppb2std(i)]=holdoutn(mppb2train,mppb2target,225,i);
[mppb2err7(i) mppb2std7(i)]=holdoutn(mppb2rs7trn,mppb2target,225,i);

fprintf(1,'mppb3 %d nodes',i)
[mppb3err(i) mppb3std(i)]=holdoutn(mppb3train,mppb3target,225,i);
[mppb3err7(i) mppb3std7(i)]=holdoutn(mppb3rs7trn,mppb3target,225,i);

fprintf(1,'mppb4 %d nodes',i)
[mppb4err(i) mppb4std(i)]=holdoutn(mppb4train,mppb4target,225,i);
[mppb4err7(i) mppb4std7(i)]=holdoutn(mppb4rs7trn,mppb4target,225,i);

fserr=[ipt1err;ipt2err;ipt3err;ipt4err;mdbb1err;mdbb2err;mdbb3err;mdbb4err;
       mppb1err;mppb2err;mppb3err;mppb4err];
fsstd=[ipt1std;ipt2std;ipt3std;ipt4std;mdbb1std;mdbb2std;mdbb3std;mdbb4std;
       mppb1std;mppb2std;mppb3std;mppb4std];
```

```
        rs7err=[ipt1err7;ipt2err7;ipt3err7;ipt4err7;mdbb1err7;mdbb2err7;mdbb3err7;mdbb4err7;
                mppb1err7;mppb2err7;mppb3err7;mppb4err7];
        rs7std=[ipt1std7;ipt2std7;ipt3std7;ipt4std7;mdbb1std7;mdbb2std7;mdbb3std7;mdbb4std7;
                mppb1std7;mppb2std7;mppb3std7;mppb4std7];

        save pbmlperr.dat fserr -ascii
        save pbmlpstd.dat fsstd -ascii
        save pbmlperr7.dat rs7err -ascii
        save pbmlpstd7.dat rs7std -ascii
end;  % for i
```

---

holdoutnlm.m

```
function [errmean, errstd]=holdoutnlm(numholdout,l1)
%       JEL 100896
%       Performs Hold-One-n training of MLP holds out n vectors for testing
%               after training on remaining vectors using L-M Method where n is
%               an integer multiple of total number of vectors in p.  This
%               function holds out n vectors [(number of vectors in p)/n] times
%               and averages the error
%       Display Frequency, Maximum Epochs, SSE Goal and
%               # of Hidden Layers parameters set in function
%       Pass:   trainvect = Training vector
%               targvect = Target vector
%               numhold out = Number of vectors to hold out for testing
%               l1 = # of Neurons in Hidden Layer
%       Return: errmean = Mean of test errors
%               errstd = Standard Deviation of Training Errors

global trainvect targvect

%       Set NN parameters

df = 0;         % Display Frequency
me = 500;       % maximum epochs
eg = .02;       % SSE Goal


numvect=size(trainvect,2);
numiterations=numvect/numholdout;

% Determine first training set
ptrain=trainvect(:,numholdout+1:numvect);
ttrain=targvect(numholdout+1:numvect);

% train and test
for i=1:numiterations
        ep=0;
        start=numholdout*(i-1)+1;
        while ep<25
```

```
        [w1,b1,w2,b2,ep,tr]=levmar(ptrain,ttrain,df,me,eg,l1);
    end; %while
    ptest=trainvect(:,start:numholdout*i);
    ttest=targvect(start:numholdout*i);
    testresult=simuff(ptest,w1,b1,'tansig',w2,b2,'logsig');
    error(i)=classerr(testresult,ttest);

    if i==numvect-1
        ptrain=trainvect(:,1:i*numholdout);
        ttrain=targvect(1:i*numholdout);
    elseif i==numvect
        xxx=0;  % if
    else
        ptrain=[trainvect(:,1:i*numholdout),trainvect(:,((i+2)*numholdout)+1:numvect)];
        ttrain=[targvect(1:i*numholdout),targvect(((i+2)*numholdout)+1:numvect)];
    end;  % if

end;  %for

errmean=mean(error);
errstd=std(error);
```

---

levmar.m

```
function [w1,b1,w2,b2,ep,tr] = levmar(p,t,df,me,eg,l1)
%       JEL 081396
%       Application of Levenburg-Marquardt method to with tanh hidden activation
%               and sigmoid output activation.
%       Pass:   p = Training vector
%               t = Target vector
%               l1 = Number of neurons in hidden layer
%               df = Display frequency
%               me = Maximum # of epochs
%               eg = SSE goal
%       Returns: w1,b1,w2,b2
%               ep = Number of epochs
%               tr = error vector

% Initiaize weights
[w1,b1,w2,b2]=initff(p,l1,'tansig',t,'logsig');

%  Set training parameters
tp=[df me eg]; % Training parameter vector

% Train the net
[w1,b1,w2,b2,ep,tr]=trainlm(w1,b1,'tansig',w2,b2,'logsig',p,t,tp);
```

B.2.1 Parameter Set Cross-Generalization. This code trains MLPs on one parameter set and tests them on the other to test their generalization abilities

crossgen.m

```
%      JEL 081496
%      Train one single hidden layer MLP, with 4 hidden nodes, for each of 12 parameter sets
%       for 500 epochs, using L-M method.  Test against other parameter sets.
%        7 element feature vector.


clear;
close all;

% load feature vectors

for i=1:4

      eval(['load ipts',num2str(i),'train900.dat;'])
      eval(['load ipts',num2str(i),'target900.dat;'])
      eval(['load mdbbs',num2str(i),'train900.dat;'])
      eval(['load mdbbs',num2str(i),'target900.dat;'])
      eval(['load mppbs',num2str(i),'train900.dat;'])
      eval(['load mppbs',num2str(i),'target900.dat;'])
end % for i

disp('Load Complete')


% Reduce feature set to 7

ipts1train900=[ipts1train900(:,1:4) ipts1train900(:,14:16)]';
ipts2train900=[ipts2train900(:,1:4) ipts2train900(:,14:16)]';
ipts3train900=[ipts3train900(:,1:4) ipts3train900(:,14:16)]';
ipts4train900=[ipts4train900(:,1:4) ipts4train900(:,14:16)]';

mdbbs1train900=[mdbbs1train900(:,1:4) mdbbs1train900(:,14:16)]';
mdbbs2train900=[mdbbs2train900(:,1:4) mdbbs2train900(:,14:16)]';
mdbbs3train900=[mdbbs3train900(:,1:4) mdbbs3train900(:,14:16)]';
mdbbs4train900=[mdbbs4train900(:,1:4) mdbbs4train900(:,14:16)]';

mppbs1train900=[mppbs1train900(:,1:4) mppbs1train900(:,14:16)]';
mppbs2train900=[mppbs2train900(:,1:4) mppbs2train900(:,14:16)]';
mppbs3train900=[mppbs3train900(:,1:4) mppbs3train900(:,14:16)]';
mppbs4train900=[mppbs4train900(:,1:4) mppbs4train900(:,14:16)]';

j=0;
df=100;
me=500;
eg=.02;
ll=4;
```

```
hidact='tansig';
outact='logsig';

for i=1:5
        iteration = i

        % Train NN using L-M method
        fprintf(1,'Start MLP training, Iteration: %d\n',i)
        for j=1:4
                eval(['[ws',num2str(j),'1,bs',num2str(j),'1,ws',num2str(j),'2,bs',num2str(j),
                  '2,ep,tr]=levmar(ipts',num2str(j),'train900,ipts',num2str(j),'target900,
                    df,me,eg,l1);']);
                eval(['[wd',num2str(j),'1,bd',num2str(j),'1,wd',num2str(j),'2,bd',num2str(j),
                  '2,ep,tr]=levmar(mdbbs',num2str(j),'train900,mdbbs',num2str(j),'target900
                    ,df,me,eg,l1);']);

                eval(['[wp',num2str(j),'1,bp',num2str(j),'1,wp',num2str(j),'2,bp',num2str(j),
                  '2,ep,tr]=levmar(mppbs',num2str(j),'train900,mppbs',num2str(j),'target900,
                    df,me,eg,l1);']);
        end; % for j

        for j=1:4
                fprintf(1,'Starting Cross Generalization test, Iteration: %d\n',i)
                eval(['ts1s',num2str(j),'=simuff(ipts',num2str(j),'train900,
                        ws11,bs11,hidact,ws12,bs12,outact);']);
                eval(['errs1s',num2str(j),'(i)=classerr(ts1s',num2str(j),',ipts',
                        num2str(j),'target900);']);
                eval(['ts1d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
                        ws11,bs11,hidact,ws12,bs12,outact);']);
                eval(['errs1d',num2str(j),'(i)=classerr(ts1d',num2str(j),',mdbbs',
                        num2str(j),'target900);']);
                eval(['ts1p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
                        ws11,bs11,hidact,ws12,bs12,outact);']);
                eval(['errs1p',num2str(j),'(i)=classerr(ts1p',num2str(j),',mppbs'
                        ,num2str(j),'target900);']);
                eval(['ts2s',num2str(j),'=simuff(ipts',num2str(j),'train900,
                        ws21,bs21,hidact,ws22,bs22,outact);']);
                eval(['errs2s',num2str(j),'(i)=classerr(ts2s',num2str(j),',ipts'
                        ,num2str(j),'target900);']);
                eval(['ts2d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
                        ws21,bs21,hidact,ws22,bs22,outact);']);
                eval(['errs2d',num2str(j),'(i)=classerr(ts2d',num2str(j),',mdbbs',
                        num2str(j),'target900);']);
                eval(['ts2p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
                        ws21,bs21,hidact,ws22,bs22,outact);']);
                eval(['errs2p',num2str(j),'(i)=classerr(ts2p',num2str(j),',mppbs'
                        ,num2str(j),'target900);']);
                eval(['ts3s',num2str(j),'=simuff(ipts',num2str(j),'train900,
                        ws31,bs31,hidact,ws32,bs32,outact);']);
                eval(['errs3s',num2str(j),'(i)=classerr(ts3s',num2str(j),',ipts',
                        num2str(j),'target900);']);
```

```
eval(['ts3d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
      ws31,bs31,hidact,ws32,bs32,outact);']);
eval(['errs3d',num2str(j),'(i)=classerr(ts3d',num2str(j),',mdbbs'
      ,num2str(j),'target900);']);
eval(['ts3p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
      ws31,bs31,hidact,ws32,bs32,outact);']);
eval(['errs3p',num2str(j),'(i)=classerr(ts3p',num2str(j),',mppbs',
      num2str(j),'target900);']);
eval(['ts4s',num2str(j),'=simuff(ipts',num2str(j),'train900,
      ws41,bs41,hidact,ws42,bs42,outact);']);
eval(['errs4s',num2str(j),'(i)=classerr(ts4s',num2str(j),',ipts',
      num2str(j),'target900);']);
eval(['ts4d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
      ws41,bs41,hidact,ws42,bs42,outact);']);
eval(['errs4d',num2str(j),'(i)=classerr(ts4d',num2str(j),',mdbbs',
      num2str(j),'target900);']);
eval(['ts4p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
      ws41,bs41,hidact,ws42,bs42,outact);']);
eval(['errs4p',num2str(j),'(i)=classerr(ts4p',num2str(j),',mppbs',
      num2str(j),'target900);']);
eval(['td1s',num2str(j),'=simuff(ipts',num2str(j),'train900,
      wd11,bd11,hidact,wd12,bd12,outact);']);
eval(['errd1s',num2str(j),'(i)=classerr(td1s',num2str(j),',ipts',
      num2str(j),'target900);']);
eval(['td1d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
      wd11,bd11,hidact,wd12,bd12,outact);']);
eval(['errd1d',num2str(j),'(i)=classerr(td1d',num2str(j),',mdbbs',
      num2str(j),'target900);']);
eval(['td1p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
      wd11,bd11,hidact,wd12,bd12,outact);']);
eval(['errd1p',num2str(j),'(i)=classerr(td1p',num2str(j),',mppbs',
      num2str(j),'target900);']);
eval(['td2s',num2str(j),'=simuff(ipts',num2str(j),'train900,
      wd21,bd21,hidact,wd22,bd22,outact);']);
eval(['errd2s',num2str(j),'(i)=classerr(td2s',num2str(j),',ipts',
      num2str(j),'target900);']);
eval(['td2d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
      wd21,bd21,hidact,wd22,bd22,outact);']);
eval(['errd2d',num2str(j),'(i)=classerr(td2d',num2str(j),',mdbbs',
      num2str(j),'target900);']);
eval(['td2p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
      wd21,bd21,hidact,wd22,bd22,outact);']);
eval(['errd2p',num2str(j),'(i)=classerr(td2p',num2str(j),',mppbs',
      num2str(j),'target900);']);
eval(['td3s',num2str(j),'=simuff(ipts',num2str(j),'train900,
      wd31,bd31,hidact,wd32,bd32,outact);']);
eval(['errd3s',num2str(j),'(i)=classerr(td3s',num2str(j),',ipts',
      num2str(j),'target900);']);
eval(['td3d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
      wd31,bd31,hidact,wd32,bd32,outact);']);
eval(['errd3d',num2str(j),'(i)=classerr(td3d',num2str(j),',mdbbs',
```

```
        num2str(j),'target900);']);
eval(['td3p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
        wd31,bd31,hidact,wd32,bd32,outact);']);
eval(['errd3p',num2str(j),'(i)=classerr(td3p',num2str(j),',mppbs',
        num2str(j),'target900);']);
eval(['td4s',num2str(j),'=simuff(ipts',num2str(j),'train900,
        wd41,bd41,hidact,wd42,bd42,outact);']);
eval(['errd4s',num2str(j),'(i)=classerr(td4s',num2str(j),',ipts',
        num2str(j),'target900);']);
eval(['td4d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
        wd41,bd41,hidact,wd42,bd42,outact);']);
eval(['errd4d',num2str(j),'(i)=classerr(td4d',num2str(j),',mdbbs',
        num2str(j),'target900);']);
eval(['td4p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
        wd41,bd41,hidact,wd42,bd42,outact);']);
eval(['errd4p',num2str(j),'(i)=classerr(td4p',num2str(j),',mppbs',
        num2str(j),'target900);']);
eval(['tp1s',num2str(j),'=simuff(ipts',num2str(j),'train900,
        wp11,bp11,hidact,wp12,bp12,outact);']);
eval(['errp1s',num2str(j),'(i)=classerr(tp1s',num2str(j),',ipts',
        num2str(j),'target900);']);
eval(['tp1d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
        wp11,bp11,hidact,wp12,bp12,outact);']);
eval(['errp1d',num2str(j),'(i)=classerr(tp1d',num2str(j),',mdbbs',
        num2str(j),'target900);']);
eval(['tp1p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
        wp11,bp11,hidact,wp12,bp12,outact);']);
eval(['errp1p',num2str(j),'(i)=classerr(tp1p',num2str(j),',mppbs',
        num2str(j),'target900);']);
eval(['tp2s',num2str(j),'=simuff(ipts',num2str(j),'train900,
        wp21,bp21,hidact,wp22,bp22,outact);']);
eval(['errp2s',num2str(j),'(i)=classerr(tp2s',num2str(j),',ipts',
        num2str(j),'target900);']);
eval(['tp2d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
        wp21,bp21,hidact,wp22,bp22,outact);']);
eval(['errp2d',num2str(j),'(i)=classerr(tp2d',num2str(j),',mdbbs',
        num2str(j),'target900);']);
eval(['tp2p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
        wp21,bp21,hidact,wp22,bp22,outact);']);
eval(['errp2p',num2str(j),'(i)=classerr(tp2p',num2str(j),',mppbs',
        num2str(j),'target900);']);
eval(['tp3s',num2str(j),'=simuff(ipts',num2str(j),'train900,
        wp31,bp31,hidact,wp32,bp32,outact);']);
eval(['errp3s',num2str(j),'(i)=classerr(tp3s',num2str(j),',ipts',
        num2str(j),'target900);']);
eval(['tp3d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
        wp31,bp31,hidact,wp32,bp32,outact);']);
eval(['errp3d',num2str(j),'(i)=classerr(tp3d',num2str(j),',mdbbs',
        num2str(j),'target900);']);
eval(['tp3p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
        wp31,bp31,hidact,wp32,bp32,outact);']);
```

```
            eval(['errp3p',num2str(j),'(i)=classerr(tp3p',num2str(j),',mppbs',
                num2str(j),'target900);']);
            eval(['tp4s',num2str(j),'=simuff(ipts',num2str(j),'train900,
                wp41,bp41,hidact,wp42,bp42,outact);']);
            eval(['errp4s',num2str(j),'(i)=classerr(tp4s',num2str(j),',ipts',
                num2str(j),'target900);']);
            eval(['tp4d',num2str(j),'=simuff(mdbbs',num2str(j),'train900,
                wp41,bp41,hidact,wp42,bp42,outact);']);
            eval(['errp4d',num2str(j),'(i)=classerr(tp4d',num2str(j),',mdbbs',
                num2str(j),'target900);']);
            eval(['tp4p',num2str(j),'=simuff(mppbs',num2str(j),'train900,
                wp41,bp41,hidact,wp42,bp42,outact);']);
            eval(['errp4p',num2str(j),'(i)=classerr(tp4p',num2str(j),',mppbs',
                num2str(j),'target900);']);
        end; %for j
end;  %for i

% Generate error and standard deviation matrices
for k=1:4
        for m=1:4
            eval(['errs',num2str(k),'s',num2str(m),'=
                mean(errs',num2str(k),'s',num2str(m),');']);
            eval(['errs',num2str(k),'d',num2str(m),'=
                mean(errs',num2str(k),'d',num2str(m),');']);
            eval(['errs',num2str(k),'p',num2str(m),'=
                mean(errs',num2str(k),'p',num2str(m),');']);
            eval(['stds',num2str(k),'s',num2str(m),'=
                std(errs',num2str(k),'s',num2str(m),');']);
            eval(['stds',num2str(k),'d',num2str(m),'=
                std(errs',num2str(k),'d',num2str(m),');']);
            eval(['stds',num2str(k),'p',num2str(m),'=
                std(errs',num2str(k),'p',num2str(m),');']);
            eval(['errd',num2str(k),'s',num2str(m),'=
                mean(errd',num2str(k),'s',num2str(m),');']);
            eval(['errd',num2str(k),'d',num2str(m),'=
                mean(errd',num2str(k),'d',num2str(m),');']);
            eval(['errd',num2str(k),'p',num2str(m),'=
                mean(errd',num2str(k),'p',num2str(m),');']);
            eval(['stdd',num2str(k),'s',num2str(m),'=
                std(errd',num2str(k),'s',num2str(m),');']);
            eval(['stdd',num2str(k),'d',num2str(m),'=
                std(errd',num2str(k),'d',num2str(m),');']);
            eval(['stdd',num2str(k),'p',num2str(m),'=
                std(errd',num2str(k),'p',num2str(m),');']);
            eval(['errp',num2str(k),'s',num2str(m),'=
                mean(errp',num2str(k),'s',num2str(m),');']);
            eval(['errp',num2str(k),'d',num2str(m),'=
                mean(errp',num2str(k),'d',num2str(m),');']);
            eval(['errp',num2str(k),'p',num2str(m),'=
                mean(errp',num2str(k),'p',num2str(m),');']);
            eval(['stdp',num2str(k),'s',num2str(m),'=
```

```
                std(errp',num2str(k),'s',num2str(m),');']);
            eval(['stdp',num2str(k),'d',num2str(m),'=
                std(errp',num2str(k),'d',num2str(m),');']);
            eval(['stdp',num2str(k),'p',num2str(m),'=
                std(errp',num2str(k),'p',num2str(m),');']);
        end; %for m
end; % for k

s1err=[errs1s1 errs1s2 errs1s3 errs1s4 errs1d1 errs1d2 errs1d3 errs1d4
        errs1p1 errs1p2 errs1p3 errs1p4];
s2err=[errs2s1 errs2s2 errs2s3 errs2s4 errs2d1 errs2d2 errs2d3 errs2d4
        errs2p1 errs2p2 errs2p3 errs2p4];
s3err=[errs3s1 errs3s2 errs3s3 errs3s4 errs3d1 errs3d2 errs3d3 errs3d4
        errs3p1 errs3p2 errs3p3 errs3p4];
s4err=[errs4s1 errs4s2 errs4s3 errs4s4 errs4d1 errs4d2 errs4d3 errs4d4
        errs4p1 errs4p2 errs4p3 errs4p4];
d1err=[errd1s1 errd1s2 errd1s3 errd1s4 errd1d1 errd1d2 errd1d3 errd1d4
        errd1p1 errd1p2 errd1p3 errd1p4];
d2err=[errd2s1 errd2s2 errd2s3 errd2s4 errd2d1 errd2d2 errd2d3 errd2d4
        errd2p1 errd2p2 errd2p3 errd2p4];
d3err=[errd3s1 errd3s2 errd3s3 errd3s4 errd3d1 errd3d2 errd3d3 errd3d4
        errd3p1 errd3p2 errd3p3 errd3p4];
d4err=[errd4s1 errd4s2 errd4s3 errd4s4 errd4d1 errd4d2 errd4d3 errd4d4
        errd4p1 errd4p2 errd4p3 errd4p4];
p1err=[errp1s1 errp1s2 errp1s3 errp1s4 errp1d1 errp1d2 errp1d3 errp1d4
        errp1p1 errp1p2 errp1p3 errp1p4];
p2err=[errp2s1 errp2s2 errp2s3 errp2s4 errp2d1 errp2d2 errp2d3 errp2d4
        errp2p1 errp2p2 errp2p3 errp2p4];
p3err=[errp3s1 errp3s2 errp3s3 errp3s4 errp3d1 errp3d2 errp3d3 errp3d4
        errp3p1 errp3p2 errp3p3 errp3p4];
p4err=[errp4s1 errp4s2 errp4s3 errp4s4 errp4d1 errp4d2 errp4d3 errp4d4
        errp4p1 errp4p2 errp4p3 errp4p4];

s1std=[stds1s1 stds1s2 stds1s3 stds1s4 stds1d1 stds1d2 stds1d3 stds1d4
        tds1p1 stds1p2 stds1p3 stds1p4];
s2std=[stds2s1 stds2s2 stds2s3 stds2s4 stds2d1 stds2d2 stds2d3 stds2d4
        stds2p1 stds2p2 stds2p3 stds2p4];
s3std=[stds3s1 stds3s2 stds3s3 stds3s4 stds3d1 stds3d2 stds3d3 stds3d4
        stds3p1 stds3p2 stds3p3 stds3p4];
s4std=[stds4s1 stds4s2 stds4s3 stds4s4 stds4d1 stds4d2 stds4d3 stds4d4
        stds4p1 stds4p2 stds4p3 stds4p4];
d1std=[stdd1s1 stdd1s2 stdd1s3 stdd1s4 stdd1d1 stdd1d2 stdd1d3 stdd1d4
        stdd1p1 stdd1p2 stdd1p3 stdd1p4];
d2std=[stdd2s1 stdd2s2 stdd2s3 stdd2s4 stdd2d1 stdd2d2 stdd2d3 stdd2d4
        stdd2p1 stdd2p2 stdd2p3 stdd2p4];
d3std=[stdd3s1 stdd3s2 stdd3s3 stdd3s4 stdd3d1 stdd3d2 stdd3d3 stdd3d4
        stdd3p1 stdd3p2 stdd3p3 stdd3p4];
d4std=[stdd4s1 stdd4s2 stdd4s3 stdd4s4 stdd4d1 stdd4d2 stdd4d3 stdd4d4
        stdd4p1 stdd4p2 stdd4p3 stdd4p4];
p1std=[stdp1s1 stdp1s2 stdp1s3 stdp1s4 stdp1d1 stdp1d2 stdp1d3 stdp1d4
        stdp1p1 stdp1p2 stdp1p3 stdp1p4];
```

```
p2std=[stdp2s1 stdp2s2 stdp2s3 stdp2s4 stdp2d1 stdp2d2 stdp2d3 stdp2d4
       stdp2p1 stdp2p2 stdp2p3 stdp2p4];
p3std=[stdp3s1 stdp3s2 stdp3s3 stdp3s4 stdp3d1 stdp3d2 stdp3d3 stdp3d4
       stdp3p1 stdp3p2 stdp3p3 stdp3p4];
p4std=[stdp4s1 stdp4s2 stdp4s3 stdp4s4 stdp4d1 stdp4d2 stdp4d3 stdp4d4
       stdp4p1 stdp4p2 stdp4p3 stdp4p4];

cgerror=[s1err;s2err;s3err;s4err;d1err;d2err;d3err;d4err;p1err;p2err;p3err;p4err];
cgstd=[s1std;s2std;s3std;s4std;d1std;d2std;d3std;d4std;p1std;p2std;p3std;p4std];

save cgerror.dat cgerror -ascii
save cgstd.dat cgstd -ascii
```

B.3 Cluster Based MLPs

This code clusters all available data into a varying number of clusters, trains and tests MLPs based on those clusters using the hold out 25% techniques and Levenberg-Marquardt method and determines the average error.

clandtrain.m

```
%     JEL 091196
%     This script runs vectors through a competitive learning clustering algorithm,
%     with variable hidden nodes, trains NNs based on the clustering and
%     computes error rate.

clear;
close all;
load ipts1train900.dat;
load ipts1target900.dat;
load ipts2train900.dat;
load ipts2target900.dat;
load ipts3train900.dat;
load ipts3target900.dat;
load ipts4train900.dat;
load ipts4target900.dat;
load mdbbs1train900.dat;
load mdbbs1target900.dat;
load mdbbs2train900.dat;
load mdbbs2target900.dat;
load mdbbs3train900.dat;
load mdbbs3target900.dat;
load mdbbs4train900.dat;
load mdbbs4target900.dat;
load mppbs1train900.dat;
load mppbs1target900.dat;
load mppbs2train900.dat;
load mppbs2target900.dat;
```

```
load mppbs3train900.dat;
load mppbs3target900.dat;
load mppbs4train900.dat;
load mppbs4target900.dat;
disp('Load Complete')

% Create training and target matrices
train=[ipts1train900; ipts2train900; ipts3train900; ipts4train900;
       mdbbs1train900; mdbbs2train900; mdbbs3train900; mdbbs4train900;
       mppbs1train900; mppbs2train900; mppbs3train900; mppbs4train900];
target=[ipts1target900 ipts2target900 ipts3target900 ipts4target900
       mdbbs1target900 mdbbs2target900 mdbbs3target900 mdbbs4target900
       mppbs1target900 mppbs2target900 mppbs3target900 mppbs4target900];

% Reduce feature set to 7
train=[train(:,1:4) train(:,14:16)]';

%      Find minimum and maximum values for each feature
min1=min(train(1,:));
min2=min(train(2,:));
min3=min(train(3,:));
min4=min(train(4,:));
min5=min(train(5,:));
min6=min(train(6,:));
min7=min(train(7,:));
max1=max(train(1,:));
max2=max(train(2,:));
max3=max(train(3,:));
max4=max(train(4,:));
max5=max(train(5,:));
max6=max(train(6,:));
max7=max(train(7,:));
p=[min1 max1;min2 max2;min3 max3;min4 max4;min5 max5;min6 max6;min7 max7];

cldf=1000;
clme=10000;
lr=0.1;
tp=[cldf clme lr];

% Vary number of clusters from 2 to 20
for i=1:10;
       fprintf(1,'Number of Clusters: %d/n', 2*i)
       n=2*i;

       % initailize weights
       w1=initc(p,n);

       % Cluster with competitive learning
       w1=trainc(w1,train,tp);
       a1=simuc(train,w1);
```

```
% Classify vectors according to CL results train and test MLP
for j=1:n
        % find vectors belonging to cluster
        eval(['cltrn=train(:,find(a1(' num2str(j) ',:)));']);
        eval(['cltgt=target(:,find(a1(' num2str(j) ',:)));']);
        % Determine number in cluster
        numinclust=size(cltrn,2)
        % split vectors in cluster into four groups
        index1=round(numinclust*.25);
        index2=round(numinclust*.50);
        index3=round(numinclust*.75);
        train1=cltrn(:,1:index1);
        train2=cltrn(:,index1+1:index2);
        train3=cltrn(:,index2+1:index3);
        train4=cltrn(:,index3+1:numinclust);
        target1=cltgt(:,1:index1);
        target2=cltgt(:,index1+1:index2);
        target3=cltgt(:,index2+1:index3);
        target4=cltgt(:,index3+1:numinclust);

        % Train and test MLP using hold out 25%
        mlpdf=100;
        mlpme=500;
        eg=.02;
        l1=4;

        ep=0;
        while ep<50
        [w1,b1,w2,b2,ep,tr]=levmar([train1 train2 train3],
                [target1 target2 target3],mlpdf,mlpme,eg,l1);
        end; %while
        testdat=simuff(train4,w1,b1,'tansig',w2,b2,'logsig');
        err1=classerr(testdat,target4);
        ep=0;

        while ep<50
        [w1,b1,w2,b2,ep,tr]=levmar([train1 train2 train4],
                [target1 target2 target4],mlpdf,mlpme,eg,l1);
        end; %while
        testdat=simuff(train3,w1,b1,'tansig',w2,b2,'logsig');
        err2=classerr(testdat,target3);
        ep=0;

        while ep<50
        [w1,b1,w2,b2,ep,tr]=levmar([train1 train4 train3],
                [target1 target4 target3],mlpdf,mlpme,eg,l1);
        end; %while
        testdat=simuff(train2,w1,b1,'tansig',w2,b2,'logsig');
        err3=classerr(testdat,target2);
        ep=0;
```

```
            while ep<50
            [w1,b1,w2,b2,ep,tr]=levmar([train4 train2 train3],
                  [target4 target2 target3],mlpdf,mlpme,eg,l1);
            end; %while
            testdat=simuff(train1,w1,b1,'tansig',w2,b2,'logsig');
            err4=classerr(testdat,target1);

            err(j)=mean([err1 err2 err3 err4]);
      end; % for j ( go to next cluster)
      toterror(i)=mean(err);
      totstd(i)=std(err);
      save clusterr.dat toterror -ascii
      save cluststd.dat totstd -ascii
end; % for i ( re-cluster)
```

B.4 Bayes Error Rate

This code bounds the Bayes error rate using the Parzen window technique with re-substitution and leave-one-out. Thanks to Curtis Martin.

pknn.m

```
% PKNN:  Run Parzen and kNN procedure for one subset of X1 and X2
%
%     [Rp, Lp, Rk, Lk] = PKNN(X1, X2, S1, S2, h, k, opt)
%
% Inputs:  X1, X2:  data sets (n x N1 and n x N2)
%          S1, S2:  Covariances (true or estimated) of X1 and X2
%          h, k:    values to use for h and k
%          opt:     1 = threshold option 3
%                   2 = threshold option 4
%
%
% All h's must be greater than zero, and k must be between 2 and
% min(N1, N2)-1
%
% Outputs: Rp, Lp: Parzen R and L errors for each test (one test per row)
%          Rk, Lk: k-NN R and L errors for each test

function [Rp, Lp, Rk, Lk] = pknn(X1, X2, S1, S2, h, k, opt)

[n1, N1] = size(X1);
[n2, N2] = size(X2);
if n1 ~= n2
   fprintf(2, 'Data sets X1 and X2 must have same number of rows (features)\n');
   return;
end  % if

% Keep this value on hand
```

```
dim = n1;
ntests = 10;
Rp = zeros(size(h));
Lp = zeros(size(h));
Rk = zeros(size(k));
Lk = zeros(size(k));

n1 = size(X1, 2);
n2 = size(X2, 2);
fprintf(1, '  %d Class 1 samples, %d Class 2 samples\n', n1, n2);
fprintf(1, '  Inverting covariance matrices . . .\n');
detratio = -0.5 * log(det(S2)/det(S1));
iS1 = inv(S1);
iS2 = inv(S2);

fprintf(1, '  Computing distances . . .\n');

[d11, d12, d21, d22] = compute_distances(X1, X2, iS1, iS2);

clear X1 X2 iS1 iS2

fprintf(1, '  Classifying (Parzen) . . .\n');
Rerr = [];
Lerr = [];

for r = h,

    % Compute sums:
    temp = -0.5 / r^2;
    s11 = sum(exp(temp * d11));
    s12 = sum(exp(temp * d12));
    s21 = sum(exp(temp * d21));
    s22 = sum(exp(temp * d22));

z1 = find(s11==0 & s21==0);
s11(z1) = realmin * ones(size(z1));
s21(z1) = realmin * ones(size(z1));
z2 = find(s12==0 & s22==0);
s12(z2) = realmin * ones(size(z2));
s22(z2) = realmin * ones(size(z2));

    lr1 = detratio - log((n2 * s11) ./ (n1 * s21));
    lr2 = detratio - log((n2 * s12) ./ (n1 * s22));

s11(z1) = zeros(size(z1));
s21(z1) = zeros(size(z1));
s12(z2) = zeros(size(z2));
s22(z2) = zeros(size(z2));

z1 = find(s11==1 & s21==0);
s11(z1) = s11(z1) + realmin * ones(size(z1));
```

```
s21(z1) = realmin * ones(size(z1));
z2 = find(s12==0 & s22==1);
s12(z2) = realmin * ones(size(z2));
s22(z2) = s22(z2) + realmin * ones(size(z2));

    ll1 = detratio - log((n2 * (s11 - 1)) ./ ((n1 - 1) * s21));
    ll2 = detratio - log(((n2 - 1) * s12) ./ (n1 * (s22 - 1)));

    [rerr, lerr] = classify(lr1, lr2, ll1, ll2, opt);

    Rerr = [Rerr, rerr];
    Lerr = [Lerr, lerr];

end  % for r

Rp = 100 * Rerr / (n1+n2);
Lp = 100 * Lerr / (n1+n2);

fprintf(1,'  Classifying (k-NN) . . .\n');

% sort distances
d11 = sort(d11);
d12 = sort(d12);
d21 = sort(d21);
d22 = sort(d22);

tr = detratio + log(n1/n2);
tl1 = detratio + log((n1-1)/n2);
tl2 = detratio + log(n1/(n2-1));
Rerr = [];
Lerr = [];

for i = k,

    lr1 = tr + 0.5 * dim * log(d11(i,:) ./ d21(i,:));
    lr2 = tr + 0.5 * dim * log(d12(i,:) ./ d22(i,:));

    ll1 = tl1 + 0.5 * dim * log(d11(i+1,:) ./ d21(i,:));
    ll2 = tl2 + 0.5 * dim * log(d12(i,:) ./ d22(i+1,:));

    [rerr, lerr] = classify(lr1, lr2, ll1, ll2, opt);

    Rerr = [Rerr, rerr];
    Lerr = [Lerr, lerr];

end  % for i

Rk = 100 * Rerr / (n1+n2);
Lk = 100 * Lerr / (n1+n2);
```

cumpute_distances.m

```
% COMPUTE_DISTANCES:  Compute K distances between samples in X1 and X2.
%
%     [D11, D12, D21, D22] = COMPUTE_DISTANCES(X1, X2, invS1, invS2)
%
%   D11 = class 1 distances for samples of class 1
%   D12 = class 2 distances for samples of class 1
%   D21 = class 1 distances for samples of class 2
%   D22 = class 2 distances for samples of class 2
%
% invS1 and invS2 are the covariance matrix inverses (maybe estimated).
% X1 and X2 have one observation per column.

function [D11, D12, D21, D22] = compute_distances(X1, X2, invS1, invS2)

% Number of columns is the number of samples
N1 = size(X1,2);
N2 = size(X2,2);

% Allocate space
D11 = zeros(N1,N1);
D12 = zeros(N1,N2);
D21 = zeros(N2,N1);
D22 = zeros(N2,N2);

% Calculate intra-class distances:
  % Class 1:
for i = 1:N1-1
    for j = i+1:N1
        D11(i,j) = (X1(:,j) - X1(:,i))' * invS1 * (X1(:,j) - X1(:,i));
        D11(j,i) = D11(i,j);
    end
end

  % Class 2:
for i = 1:N2-1
    for j = i+1:N2
        D22(i,j) = (X2(:,j) - X2(:,i))' * invS2 * (X2(:,j) - X2(:,i));
        D22(j,i) = D22(i,j);
    end
end

% Calculate inter-class distances:
for i = 1:N1            % Rows for D12, Columns for D21
    for j = 1:N2        % Columns for D12, Rows for D21
        % Pull samples out of matrices only once
        v = X2(:,j) - X1(:,i);
        D12(i,j) = v' * invS1 * v;

        % These could be (-v), but there's no reason for it. (note (j,i))
```

```
        D21(j,i) = v' * invS2 * v;
    end
end
```

---

classify.m

```
% CLASSIFY: Select thresholds and classify resubstitution and
%           leave-one-out discriminant values.
%
%     [R, L] = CLASSIFY(Lr1, Lr2, Ll1, Ll2, option)
%
% Inputs:  Lr1, Lr2: resubstitution discriminant values
%          Ll1, Ll2: leave-one-out discriminant values
%          option:   1 = threshold option 3
%                    2 = threshold option 4
%
% Outputs: R: Resubstitution error
%          L: Leave-one-out error

function [R, L] = classify(Lr1, Lr2, Ll1, Ll2, option)

% First get the minimum resubstitution error and threshold
fprintf(1,' Resubstitution. . .');
[R, t] = min_error(Lr1, Lr2);
fprintf(1,'done.\n');

% Now, depending on the option, get the minimum leave-one-out error
fprintf(1,' Leave one out. . .');
if option == 1
    L = sum(Ll1 > t) + sum(Ll2 < t);
else
    n1 = size(Ll1, 2);
    n2 = size(Ll2, 2);
    L = 0;
fprintf(1,'Class 1. . .');
    for i = 1:n1,
        [err, t] = min_error(Ll1([1:i-1 i+1:n1]), Ll2);
        if Ll1(i) > t
            L = L + 1;
        end  % if Ll1(i) > t
    end  % for i
fprintf(1,'Class 2. . .');
    for i = 1:n2,
        [err, t] = min_error(Ll1, Ll2([1:i-1 i+1:n1]));
        if Ll2(i) < t
            L = L + 1;
        end  % if Ll2(i) < t
    end  % for i
end  % if option == 1
```

```
fprintf(1,'done.\n');
```

Bibliography

1. A. A. Tarraf, et al. "Intelligent Traffic Control for ATM Broadband Networks," *IEEE Communications Magazine* (October 1995).
2. Baum, Eric B. "What Size Net Gives Valid Generalization?," *Neural Computation, 1* (1989).
3. Chan, L.-W and F. Fallside. "An Adaptive Training Algorithm for Back Propagation Networks," *Computer Science and Language, 2*:205–218 (1987).
4. Chen, X. and I.M. Leslie. "Neural Network Approach Towards Adaptive Congestion Control in Broadband ATM Networks." *GlOBECOM '91*. 1990.
5. Cybenko, G. "Approximation by Superpositions of a Sigmoidal Function," *Math. Control, Signals Sys. 2*, 303–314 (1989).
6. Demuth, H. and M. Beale. *Neural Network Toolbox*. Natick, Massachusetts: The Mathworks, Inc., 1994.
7. Dennis M. Drew, Col. "Basic Aerospace Doctrine of the United States Air Force," *Air Force Manual 1-1* (March 1992).
8. E. Nordstrom, et al. "Neural Networks for Adaptive Control in ATM Networks," *IEEE Communications Magazine* (October 1995).
9. Evanowsky, John B. "Information for the Warrior," *IEEE Communications Magazine* (October 1995).
10. Fukunaga, Keinosuke. *Introduction to Statistical Pattern Recognition*. Boston: Acedemic Press, Inc, 1990.
11. Gudta, Barbosa and Georganas. "Switching Modules for ATM Switching Systems and Their Interconnection Networks," *Computer Networks and ISDN Systems*, *26*(1):433–445 (1993).
12. Heffes, H. and D. Lucantoni. "A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance," *IEEE Journal on Selected Areas in Communication* (September 1986).
13. Hiramatsu, A. "ATM Communications Network Control by Neural Networks," *IEEE Transactions on Neural Networks* (March 1990).
14. Hiramatsu, A. "Integration of ATM Call Admission Control and Link Capacity Control by Distributed Neural Networks," *IEEE Journal on Selected Areas in Communications* (September 1991).
15. Hiramatsu, A. "Training Techniques for Neural Network Applications in ATM," *IEEE Communications Magazine* (October 1995).

16. Imrich, et al. "A counter based congestion control for ATM networks," *Computer Networks and ISDN Systems*, *26*(1):1–162 (1993).

17. J.E. Neves, et al. "Neural Networks in B-ISDN Flow Control: ATM Traffic Predictor or Network Modeling," *IEEE Communiocations Magazine* (October 1995).

18. Jiang, X. and J. Meditch. "A high speed integrated services ATM/STM switch," *Computer Networks and ISDN Systems*, *1*:459–477 (1993).

19. Kalman, Barry L. and Stan C. Kwasny. "Why Tanh: Choosing a Sigmoidal Function," *IJCNN International Joint Conference on Neural Networks* (June 1992).

20. Karayiannis, N. B. "Accelerating the Training of Feedforward Neural Networks Using Generalized Hebbian Rules for Initializing the Internal Representations," *IEEE Transactions on Neural Networks* (March 1996).

21. Li and Weng. "B+ -Tree: A High Speed Switching Structure for ATM with Dual Input Buffering," *Computer Systems and ISDN Networks*, *28*(1):1499–1522 (1995).

22. Martin, 2Lt Curtis Eli. *Non-Parametric Bayes Error Estimation for UHRR Target Identification.* MS thesis, AFIT/GE/ENG/93D-26, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

23. Ogier, Roger and Nina T. Plotkin. "Neural Network Methods with Traffic Descriptor Compression for Call Admission Control." *IEEE Infocom Proceedings.* March 1996.

24. Park, Y.K. and G. Lee. "Applications of Neural Networks in High Speed Communications Networks," *IEEE Communications Magazine* (October 1995).

25. Parzen, Emanuel. "On Estimation of a Probability Density Function and Mode," *Ann. Math. Stat.*, *33* (1962).

26. Ruck, Capt Dennis W. *Characterization of Multilayer Perceptrons and Their Application to Multisensor Automatic Target Detection.* PhD dissertation, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1990.

27. Sole-Pareta, J. and J. Domingo-Pascual. "Burstiness characteristics of ATM cell streams," *Computer Networks and ISDN Systems*, *26*(2):1351–1363 (1994).

28. Sriram, K. "Methodologies for bandwidth allocation, transmission scheduling, and congestion avoidance in broadband ATM networks," *Computer Networks and ISDN Systems*, *26*(1):43–59 (1993).

29. Steppe, Jean. *Feature and Model Selection in Feedforward Neural Networks*. PhD dissertation, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1992.

30. Van Der Smagt, P. Patrick. "Minimisation Methods for Training Feedforward Neural Networks," *Neural Networks*, 7(1):1–11 (1994).

Vita

Captain Jeffrey E. Larson [redacted] He graduated from the University of Minnesota in 1982 with a Bachelor of Science in Business degree. In 1984 he enlisted in the United States Air Force and, following basic training and technical school, was stationed at Wright-Patterson AFB, OH. In 1987 he was accepted into the Airmen's Education and Commissioning Program and graduated *cum laude* from Wright State University with a Bachelor of Science in Electrical Engineering degree in 1990. Following Officer Training School, Captain Larson received his commission in the United States Air Force on October 1, 1990 and was assigned to Hanscom AFB, MA. In May 1995 he was assigned to the Air Force Institute of Technology, Wright-Patterson AFB, OH to pursue a Masters of Science in Electrical Engineering degree. Upon completion Captain Larson is assigned to Wright Laboratory, Wright Patterson AFB, OH.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December, 1996	final

4. TITLE AND SUBTITLE
ADAPTIVE NEURAL NETWORK CONTROLLER FOR ATM TRAFFIC

5. FUNDING NUMBERS

6. AUTHOR(S)
Jeffrey E. Larson

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

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/96D-09

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr John Evanowsky
RL/C3B
Griffis AFB, NY

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Broadband-Integrated Services Digital Networks (B-ISDN), along with Asynchronous Transfer Mode (ATM), were designed to meet the requirements of modern communication networks to handle multiple users and a wide variety of diverse traffic including voice, data and video. ATM responds to requests for admission to the network by analyzing whether or not the grade of service (GOS) requirement, specified in the admission request, can be guaranteed without violating the GOS guaranteed to traffic already accepted into the network. The GOS is typically a parameter such as cell loss rate (CLR), average delay, or some other measurement associated with network performance. In order to develop a tractable mathematical algorithm for controlling admission, an accurate model of the communication network and traffic in question is necessary. The complex and dynamic nature of these communication networks make them very difficult to model. Even when such a model can be developed, often with unrealistic simplifications or unsupportable assumptions, the associated mathematical algorithm is frequently excessively cumbersome and timely processing of an admission request is lost. An alternative to conventional mathematical algorithms for cases like these is the use of neural networks (NN). NNs can learn complicated functions relating the inputs and outputs of a system without prior knowledge about the system itself. For ATM B-ISDN networks, NNs can learn the function relating input traffic parameters and resulting network performance by training on an appropriate set of traffic parameter inputs and resulting GOS outputs. In this work three neural network admission controller schemes are examined. The Bayes error rate, as bounded by the Parzen window technique, is also introduced as a benchmark for measuring the performance of these admission controllers. Results indicate that error rates approaching the Bayes error rate can be obtained by using a self organizing, or clustering, algorithm to segment the input space and then train separate MLPs on each cluster. This clustering algorithm can also be used to direct the traffic streams requiring classification to the appropriately trained NN admission controller.

14. SUBJECT TERMS	15. NUMBER OF PAGES
neural networks, ATM, Bayes Error	91
	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