A NEURAL NETWORK APPROACH TO BORDER GATEWAY PROTOCOL PEER FAILURE DETECTION AND PREDICTION

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Cory B. White

December 2009

COMMITTEE MEMBERSHIP

TITLE:	A Neural Network Approach to Border Gateway Protocol Peer Failure Detection and Prediction
AUTHOR:	Cory B. White
DATE SUBMITTED:	December 2009
COMMITTEE CHAIR:	Franz Kurfess, Ph.D.
COMMITTEE MEMBER:	Hugh Smith, Ph.D.
COMMITTEE MEMBER:	Ignatios Vakalis, Ph.D.

Abstract

A Neural Network Approach to Border Gateway Protocol Peer Failure Detection and Prediction

Cory B. White

The size and speed of computer networks continue to expand at a rapid pace, as do the corresponding errors, failures, and faults inherent within such extensive networks. This thesis introduces a novel approach to interface Border Gateway Protocol (BGP) computer networks with neural networks to learn the precursor connectivity patterns that emerge prior to a node failure. Details of the design and construction of a framework that utilizes neural networks to learn and monitor BGP connection states as a means of detecting and predicting BGP peer node failure are presented. Moreover, this framework is used to monitor a BGP network and a suite of tests are conducted to establish that this neural network approach as a viable strategy for predicting BGP peer node failure. For all performed experiments both of the proposed neural network architectures succeed in memorizing and utilizing the network connectivity patterns. Lastly, a discussion of this framework's generic design is presented to acknowledge how other types of networks and alternate machine learning techniques can be accommodated with relative ease.

Acknowledgements

I am greatful for having a dedicated and reliable committee of professors who were always willing to discuss my progress and offer constructive suggestions. Thanks specifically to the chair of my committee–my mentor–Dr. Franz Kurfess. His guidance, interests, and eager enthusiasm provided me with a strong motivation to explore the field of artificial intelligence and machine learning. He has always maintained an optimistic sense of humor and has been understanding and more than willing to work with me through any difficulties. His devotion to the interests and paths of his students–myself being no exception–is exceptional. Thank you for being such a great teacher, mentor, and friend.

Also, a very special thanks to my parents and grandparents. Throughout my college experience, these members of my familiy have provided me with economic stability, confidence, understanding, guidance, and always a listening ear during the tougher times. Their self-sacrifice and selfless continual support have sculpted me into the person I am today and has allowed me to accomplish my college goals and earn this Master's degree. My accomplishments as a son reflect their successes as a family unit. Thank you for all that you have given.

Contents

List of Tables

List of Figures

List of Code Blocks

Chapter 1

Introduction

This chapter provides an exposition of the objectives of this thesis and the layout for the remainder of the document.

1.1 Overview of Problem Statement

The size and speed of computer networks continue to expand at a rapid pace, as do the corresponding errors, failures, and faults inherent within extensive networks. With this growth, large Internet-based companies such as Amazon, Google, and Yahoo! and even smaller companies with high reliance upon a computing infrastructure depend upon the reliability of such networks in order to earn revenue and remain dominant in competitive markets. Thus, the need grows for network tools and techniques to maintain and monitor such systems in order to quickly and efficiently detect problems and potentially even predict issues before they occur. These tools must therefore ascertain and store knowledge about the network topology and communication between nodes in order to draw inferences about any potential problems that may arise.

The specific protocol of interest is the Border Gateway Protocol (BGP). Acknowledged as the de facto interdomain routing protocol of the Internet, if a BGP router becomes jeopardized or goes offline,

> An autonomous system can have its traffic black-holed or otherwise misrouted, and packets to or from it can be grossly delayed or dropped altogether. Malfunctioning ASes harm their peers by forcing them to recalculate routes and alter their routing tables...these events can disrupt international backbone networks and have the potential to bring a large part of the Internet to a standstill.[16]

This thesis introduces a novel approach to interface BGP computer networks with neural networks to learn the precursor connectivity patterns that emerge prior to a router failure. Such patterns are collected and then memorized by neural networks which will then be able to detect or, potentially, even predict future node failures if similar connective patterns emerge in the future. The co-founder of the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology Marvin Minsky stated in 1991 that,

> To program today, we must describe things very carefully, because nowhere is there any margin for error. But once we have modules that know how to learn, we won't have to specify nearly so much–and we'll program on a grander scale, relying on learning to fill in the details.[56]

This thesis demonstrates how neural networks can fill in the details necessary for subset of problems that can emerge within a computer network. Thus, the work presented here takes another step in the direction that Minksy predicted eighteen years ago.

1.2 Outline

This thesis will first acknowledge the background of neural networks, their wide range of application, and their popular use as predictive tools. Next, some relevant details of BGP will be presented along with a survey of current network monitoring tools and techniques that utilize machine learning techniques will be discussed. After the survey, a more in-depth look at BGP and the particular neural network learning algorithm utilized in this thesis will be assessed. Following these details, the specific design applied in this thesis will be presented in both a high-level perspective as well as the details of the lower-level software framework implementation. Next, three representative network topologies will be considered and tested using the written software, which will be followed by corresponding results and conclusions. Lastly, a discussion of a wide range of potential future work stemming from this thesis will be conducted.

Chapter 2

Related Work and Background

This chapter will assess work that is related to the content of this thesis. The following sections will cover a brief history of neural networks, their use as predictive tools, some relevant details on BGP, and network monitoring tools.

2.1 Neural Networks

This section will introduce a brief history of neural networks as well as provide various examples of how neural networks have been used as predictive tools.

2.1.1 A Brief History

Artificial neural networks provide a new approach to solving ill-structured problems that are not easily solved using procedural solutions. Neural networks are composed of nodes, or neurons, that perform in a manner similar to that of the biological neuron along with interconnecting weights between neurons. These components allow neural networks to learn through training, generalize

from previous examples, and abstract characteristics from unclear data.

The epoch of neural network theory started in 1943 with the publication by Warren McCulloch and Walter Pitts, where they considered the case of a network made up of binary decision units (BDNs) and showed that such a network could perform any logical function on its inputs [73]. Then, in 1962, Frank Rosenblatt published his course book Principles of Neurodynamics [65] where he showed that it is possible to train a network of BDNs, which he coined a perceptron network, that could recognize a set of chosen patterns. However, in 1969 Marvin Minsky and Seymour Papert [57] mathematically proved that perceptrons were very limited, showing that they could not solve some very simple pattern classification tasks, such as the XOR function.

Neural networks were then left relatively overlooked for several years until Paul Werbos created the backpropagation algorithm in his 1974 PhD Thesis [79]. This algorithm allows the error of a neural network to be propagated back from the output neurons to earlier layers in the network to make the correct modification to all the hidden connections between neruons. An additional stimulus for continued research came from John Hopfield in 1982 where he related the training of BDNs to a gradient optimizations problem, followed by the introduction of the Boltzman learning algorithm by Hinton and Sejnowski in 1983 [73].

As research continued, numerous architectural methodologies emerged and two extremes arose: feedforward networks, where input flows from the input layer neurons through any inner layer neurons and then to the output, and recurrent networks, where the neurons of the network provide constant feedback to each other. Training strategies also developed into one of three categories: supervised, where the error output of a network is used to train the network, reinforcement, which rewards the network for good performance (thereby strength-

ening the corresponding weights), and unsupervised, which increases connection weights whenever two neurons are active together [73].

Along with the advances in the theory of neural networks is the continuous developments in their widespread application. Neural networks have been applied in the fields of vision, speech, signal analysis, robotics, expert systems, computers, and process planning/control, just to name a few [42]. One of the most influential and widely cited neural network papers is on the application of neural networks for face detection [66]. Additionally, the most recent publications in neural networks typically combine the newest training strategies to solve a very specific task, which is often heavily interdisciplinary in nature. This development and future of neural networks is exemplified in the work of Rui Xu and Donald Wunsch II and Ronald Frank [84] as well as Jinmiao Chen and Narendra Chaudhari [17], where uniquely structured and trained networks are used to aid in complex bioinformatics.

2.1.2 Neural Networks As Predictive Tools

Multi-layered feedforward neural networks that learn by the supervised training via the backpropagation algorithm will be utilized in this thesis and will be discussed in further detail in Chapter 3.2.2. This section simply assesses a wide range of recent work that utilizes the predictive power of neural networks.

The implementation and utilization of artificial neural networks dates back to the early 1940s with vast breadth in domain of application. An early survey of wide-spread uses of neural networks [27] is from 1992, which acknowledges research in the prediction of mortgage loan performance among many others. Moreover, a relevant survey of trends in neural network publications up until 1996 can be found in [77], which highlights the diversity of their usage. More

recently, in 2004, another survey on neural networks acknowledged several neural network techniques applied to the prediction of categorization tasks [14].

Within the past year alone, several papers have been published that apply neural networks to a domain that requires a predictive element. For example, both [11] and [62] use feedforward neural networks to predict the failure strength of composite tensile specimens. Additionally, [41] was able to predict defects in castings through the use of backpropagation neural networks. Moreover,[78] utilizes neural networks with temperature weather feature inputs for short-term electricity load forecasting. Further, in [45] neural networks are used to predict the short-term typhoon surge and surge deviation in Taichung Harbor, Taiwan. And, as a last example, [25] employs neural networks as a means for protein structural class prediction–a classic problem that has seen the utilization of neural networks for many years [5, 10, 17, 18, 34, 43, 50].

With the surplus of aforementioned examples exploring the wide-range of various neural network publications, neural networks are certainly a popular choice as a predictive tool. Having established this point, the domain in which neural networks will be employed can now be explored.

2.2 The Border Gateway Protocol

The Border Gateway Protocol (BGP) is a protocol that is used to exchange routing information among routers in different autonomous systems (ASs) [40]. An AS, as defined by Juniper Inc. is "a set of routers that are under a single technical administration and normally use a single interior gateway protocol and a common set of metrics to propagate routing information within the set of routers" [39]. To other ASs, an AS appears to have a single internal routing plan and

presents a consistent picture of what destinations are reachable through it.

The routing information transmitted in BGP is comprised of the complete route to a desired destination, rather than a simple one-hop step. This routing information is used by BGP to maintain a database of network reachability information, which is exchanged with other BGP systems via peer-to-peer communication through BGP Speakers (routers that implement BGP). A BGP system shares this reachability information with adjacent BGP systems, which are referred to as neighbors or peers. BGP uses the network reachability information to construct a graph of AS connectivity, thus allowing BGP to remove routing loops and enforce policy decisions at the AS level.

Two types of routing information exchange are allowed in BGP: information transferred between two different ASs (external BGP or eBGP) and information transferred within the same AS (internal BGP or iBGP). Thus, for eBGP, when two BGP routers connect, the two routers are located in different ASs, thereby performing inter-AS routing. As for iBGP, on the other hand, both BGP routers exchanging information would be located within the same AS and exchanges of information would be intra-AS routing.

When two BGP peers connect they can exchange four different types of messages to each other: open, update, keepalive, and notification. However, for a connection to first be established, a Transmission Control Protocol (TCP) connection must be made between the two BGP peers. With a TCP connection the two routers can then exchange BGP open messages to create a BGP connection between them. The complete connection-making process is described by a Finite State Machine (FSM) as defined in RFC 1771 [63], and is shown in Figure 2.1. Once the connection is established, the two systems can exchange other BGP messages and routing information.

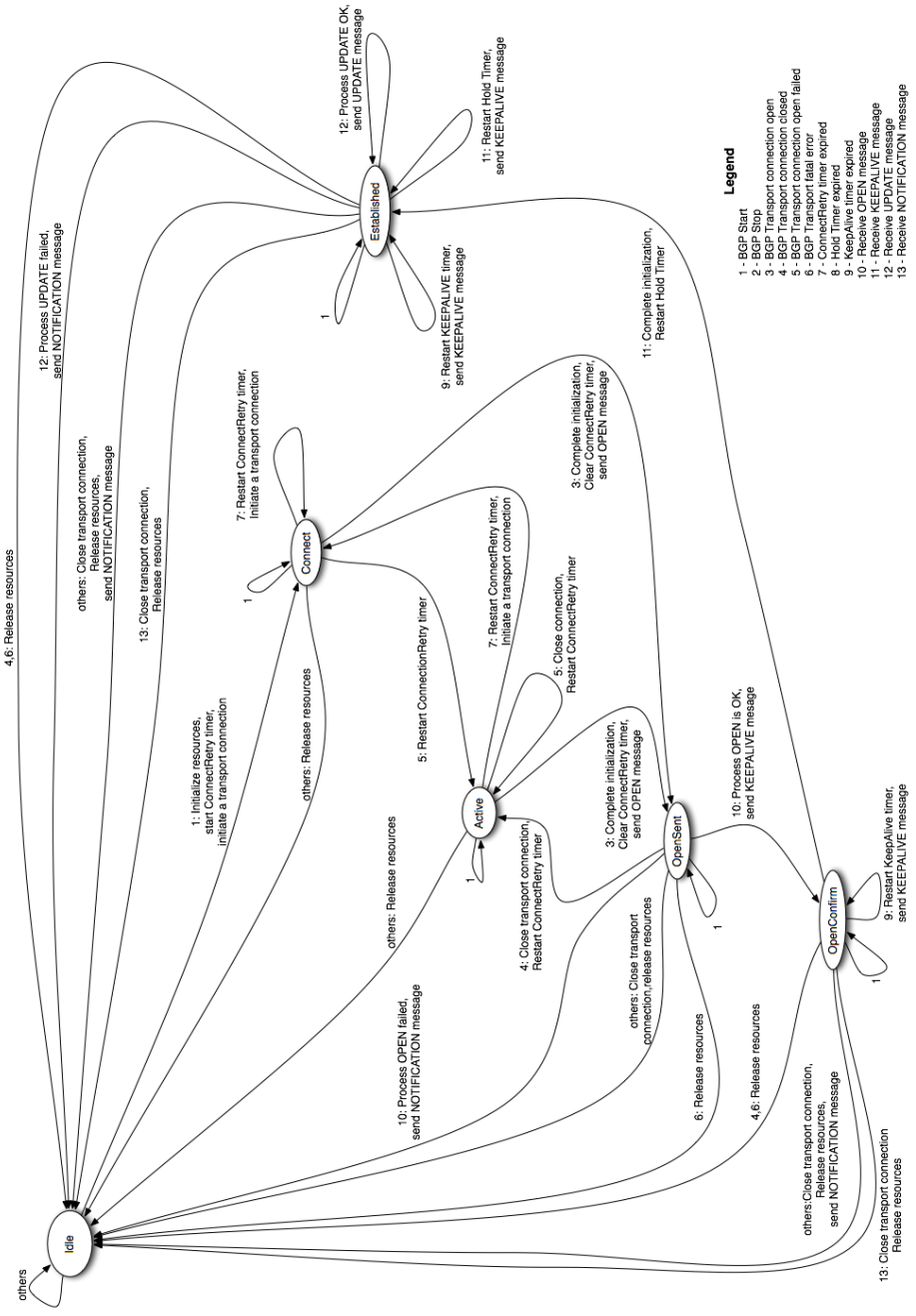


Figure 2.1: The Border Gateway Protocol connection Finite State Machine.

This thesis focuses primarily upon the connectionist perspective of BGP networks. However, for more information on the routing table and the various attributes associated with BGP paths, see Cisco's Internetworking Technology Handbook BGP Documentation [22].

2.2.1 BGP Monitoring Tools

This section will first assess the current BGP monitoring tools available, followed by various network monitoring tools that utilize some form of machine learning for the detection of failures or faults within a network.

In terms of BGP network monitoring tools, each of the most recent tools as listed in the incredibly verbose survey [24] will be briefly assessed. Of the four listed tools, the first is BGPlay [23], which is a Java application that displays animated graphs of routing activity within a specified time interval. Next is BGPMon [75], which can monitor routes and alert in case of an 'interesting' path change. Path changing became a high-interest topic when YouTube was taken offline due to false path broadcast [54]. The next tool of interest is iBGPlay [8], which is a free tool that, similar to BGPlay, graphically displays and animates BGP routing data and thereby enables a user the ability to timely identify and diagnose potential routing problems and anomalies. Lastly, LinkRank [44] offers a different approach in visualization by creating graphs that weigh the links between autonomous systems by the number of routing paths going through each link. However, out of all four tools, none contain a machine-learning mechanism.

2.2.2 Network Monitoring Tools Utilizing Machine Learning

The following is a brief survey of tools and techniques that monitor a network utilizing some form of machine learning.

The first technique is dynamic syslog mining for network failure monitoring, as presented in [85]. In this work, the authors pursue network failure correlation and detection through monitoring syslogs. A syslog is a sequence of events which are collected using the BSD syslog protocol and are used to address a wide range of important issues including network failure symptom detection and event correlation discovery. The authors propose a new methodology of dynamic syslog mining in order to detect failure symptoms with higher confidence, to discover sequential alarm patterns among computer devices, and to detect event correlations among syslogs for different devices. As stated by the authors, the key ideas of dynamic syslog mining are 1) to represent syslog behavior using a mixture of Hidden Markov Models, 2) to adaptively learn the model using an on-line discounting learning algorithm in combination with dynamic selection of the optimal number of mixture components, and 3) to give anomaly scores using universal test statistics with a dynamically optimized threshold. Strictly with respect to machine learning techniques, a mixture of a hybrid Baum-Welch algorithm for learning Hidden Markov Models and a Naive Bayes model to learn patterns were utilized within the syslog data. The validity of this technique has been demonstrated through the use of real syslog data in the scenarios of failure symptom detection, emerging pattern identification, and dynamic correlation discovery.

Another technique is predicting node availability in peer-to-peer networks, as presented in [55]. In this work the authors improve upon the accuracy of

previous peer-to-peer availability models, which are often too conservative to dynamically predict system availability at a fine-grained level. Three types of availability predictors are utilized: 1) A graph-based representation to represent likelihood of traversal, 2) linear prediction (a common statistical technique for predicting time series), and 3) hierarchical accuracy tournaments to dynamically select the most accurate sub-predictor for a particular lookahead interval. Here, the machine learning techniques used are saturating counter predictors and a linear predictor which are trained via availability traces. Though this technique does not consider fault detection, network administrators often need notification of availabilities just as much as for failures.

An additional technique is manifold learning visualization of network traffic data, as presented in [61]. In this work, the authors present a manifold learning-based tool for the visualization of large sets of data and allows for an easy comparison of data maps over time. This tool emphasizes the unusually small or large correlations that exist within a given data set along with an online Java-based GUI which allows interactive demonstration of the use of the visualization method. Moreover, data collection for visualization is made possible through the use of sensors which are located through a network that measure a chosen traffic statistic and divide traffic by source or destination IP address, port, autonomous system, time period, link, or router. Furthermore, this technique mainly considers monitoring a network for changes over time, across space (at various routers in the network), over source and destination ports, IP addresses, or AS numbers.

Another technique is detecting anomalies in network traffic using maximum entropy estimation, as proposed in [33]. In this technique, the authors develop a behavior-based anomaly detection method that detects network anomalies by comparing the current network traffic against a baseline distribution. The maxi-

mum entropy learning technique provides a flexible and fast approach to estimate the baseline distribution, which also gives the network administrator a multi-dimensional view of the network traffic by classifying packets according to a set of attributes carried by a packet. By computing a measure related to the relative entropy of the network traffic under observation with respect to the baseline distribution, this technique can distinguish anomalies that change the traffic either abruptly or slowly. Moreover, information is given regarding the type of anomaly detected and has a low false positive rate.

The next technique is an agent-based simulation of behavioral anticipation, specifically with regard to anticipatory fault managment in computer networks, as proposed in [67]. In this work, the authors explore the concept of anticipatory behavior to develop an intelligent agent-based network management model. An anticipatory agent is used to proactively detect occurrence of faults using a predictive model pertaining to network performance. Prediction is possible through the machine learning technique of a Bayesian classifier trained with past data. The agent is therefore an entity which uses the knowledge of predicted future states to decide what actions need to be taken at the present.

Another technique is anomaly detection by finding feature distribution outliers, as proposed in [71]. Here, the authors develop a means to detect traffic anomalies based on network flow behavior. First, baseline distributions for meaningful traffic features are estimated and measures of legitimate corresponding deviations are taken. Observed network behavior is then compared to the baseline behavior by means of a symmetrized version of the Kullback-Leibler divergence. The achieved dimension reduction enables effective outlier detection to flag deviations from the legitimate behavior with high precision. The actual machine learning mechanism is the application of probability mass functions in

conjunction with Kullback-Leibler divergence for learning the baseline network function. This technique supports online training and provides enough information to efficiently classify observed anomalies and allows in-depth analysis on demand.

The next technique involves exploring event correlation for failure prediction, as shown in [30]. In this work, the authors develop a spherical covariance model with an adjustable timescale parameter to quantify the temporal correlation and a stochastic model to describe spatial correlation. This is accomplished through the use of failure signatures, which extract the essential characteristics from a system state that are associated with a failure event and consider the hierarchical structure and interactions among components of the system. The authors further utilize the information of application allocation to discover more correlations among failure instances. Failure events are clustered based on their correlations and then used to predict similar future occurrences. The actual tool implemented is a failure prediction framework, called PREdictor of Failure Events Correlated Temporal-Spatially (hPREFECTs, where the 'h' stands for hierarchical), which explores correlations among failures and forecasts the time-between-failure of future instances, making use of both a neural network approach and a Bayesian network to learn and forecast failure dynamics based on the temporal and spatial data among the failure signatures.

An additional technique is an adaptive distributed mechanism used against flooding network attacks, as introduced in [9]. Here, the authors focus on early detection and the stop of distributed flooding attacks and network abuses. The framework cooperatively detects and reacts to abnormal behaviors before the target machine collapses and network performance degrades. In this framework, nodes in an intermediate network share information about their local traffic ob-

servations, improving their global traffic perspective. Also, the authors add to each node the ability of learning independently with a Naive Bayesian method to classify different types of traffic, therefore allowing each node to react differently according to its situation in the network and local traffic conditions. The learning component also allows the system to create, adjust, and renew the behavior models. This then frees the administrator from having to guess and manually set the parameters distinguishing attacks from non-attacks; now such thresholds are learned and set from experience or past data.

Another technique involves detecting attack signatures in the real network traffic with ANNIDA (Artificial Neural Network for Intrusion Detection Application) which is discussed in both [69] and [26]. In these works, a Hamming Net artificial neural network methodology was used with good results where strings in computer network packets are inserted in these neural networks for pattern classification. Test results highlight the high accuracy and efficiency of the application when submitted to real data from HTTP network traffic containing actual traces of attacks and legitimate data.

Another technique utilizes a cascading neural network (CNN) for traffic management of computer networks, as shown in [19]. In this work, the machine learning component consists of a two-level neural network model where one level (the back-propagation neural model), detects whether the tested network is overloaded and the second level, a counter-propagation neural model, classifies and excludes the status of congestion derived from the overload of tested network. In this way, if the effect gained from the first level neural model is positive, the second level neural model will be triggered to help reroute the traffic of computer networks. To validate this two-level neural network feasibility, the proposed CNN has been applied to a local area network environment. The experimental results

demonstrate that the developed CNN can efficiently and effectively provide substantial assistance for decision making in network traffic management.

Chapter 3

Domain Details

This chapter will discuss, in further detail, the two domains of interest for this thesis: BGP and neural networks.

3.1 BGP and SNMP

The primary work in this thesis is focused on the rigorously defined connection protocol for two BGP peers to start a session (as shown in the previous chapter), rather than the functionality of BGP in exchanging and storing routing information. Additionally, the focus of this work is oriented around network management of a single AS of interest, such as the domain that an administrator would have control over changing. Thus, the focus of this thesis is placed on iBGP rather then eBGP since only the network health of a single AS is of concern.

To provide some idea of the number of BGP Speakers in large Internet Service Provider (ISP) networks, see Table 3.1. These numbers will allow for designing representative experiments for scalability purposes in terms of the number of

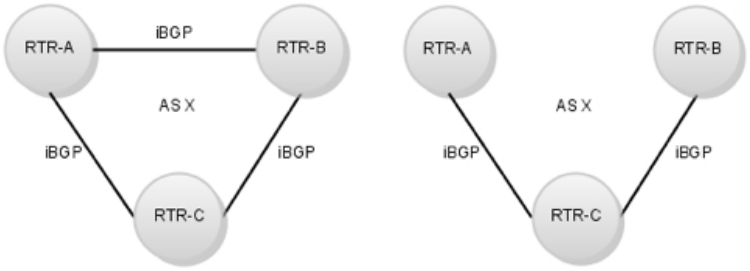
ASN	ISP Name	Total Routers	BGP Speakers
7018	AT&T	731	46
1239	Sprint	497	56
701	WorldCom/UUNet	4556	235
2914	Verio	865	80
3561	Cable & Wireless	2236	238
3356	Level3	483	61
6461	AboveNet	247	39
3967	Exodus	213	32

Table 3.1: BGP Speaker numbers for representative Internet Service Providers Backbones, as shown in [48].

BGP Speakers in Chapter 5.

One additional detail regarding iBGP is the moderately recent addition of route-reflection (RR), as shown in RFC 2796 [6] and RFC 4456 [7]. Route-reflection serves as an alternative to a fully-meshed iBGP network in that it can drastically reduce the number of required TCP sessions between BGP peers. A fully-meshed set of N BGP speakers must have $\frac{N(N-1)}{2}$ unique TCP Sessions, which presents obvious scalability issues when only 40 nodes yields 780 TCP sessions.

The best way to describe how route-reflection works is by example, as discussed in [6]. Given a simple three-node network as shown in Figure 3.1, both Figure 3.1a and Figure 3.1b represent the same three-node BGP network within the same AS and all links are iBGP sessions. In the case of Figure 3.1a, when RTR-A receives an external route that is selected as the best path, it must then advertise that path to both RTR-B and RTR-C. Once received, RTR-B and



(a) Full mesh iBGP (b) Route reflection iBGP

Figure 3.1: Full Mesh vs. Route Reflection iBGP.

RTR-C will not re-advertise this path. However, in the case of Figure 3.1b, when RTR-A receives the same route, it would then advertise only to RTR-C. In the case of route reflection, RTR-C is now allowed to re-advertise (or reflect) the path learned from RTR-A to RTR-B and vice versa. This shows that the need for the additional iBGP session between RTR-A and RTR-C is unnecessary. Thus, the option of route reflection seems appropriate for large-scale networks and a representative topology will be assessed during experimentation.

Lastly, the particular monitoring methodology will be via the Simple Network Management Protocol (SNMP). As stated in [72], "SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth". Moreover, SNMP is a fairly common methodology for network and BGP monitoring [37], and fits well into the scope of this thesis.

3.2 Neural Network Architecture

3.2.1 Biological Inspiration

The name "neural" network makes a direct connection to that of biological neural structures. Such a connection is very appropriate since the concept of mathematical and computation neural networks were inspired and designed, in part, thanks to the connectionist theories of the brain. For this reason, a brief overview of biological neurons will be considered and translated into the mathematical model that is used in this thesis.

First of all, the typical structure of a biological neuron is shown in Figure 3.2. For the purpose of computational neural networks, the key features of the neuron are the dendrites, the axon/axon terminal, and the cell body. In a biological neural network, the dendrites collect neurotransmitters from the axon terminals of adjacent neurons, as shown in Figure 3.3. These signals are accumulated within the cell body and, if a certain level have been ascertained, the neuron will also fire, sending its own signal out through its axon [58]. These high-level functions are translated directly into the mathematical model of a neuron.

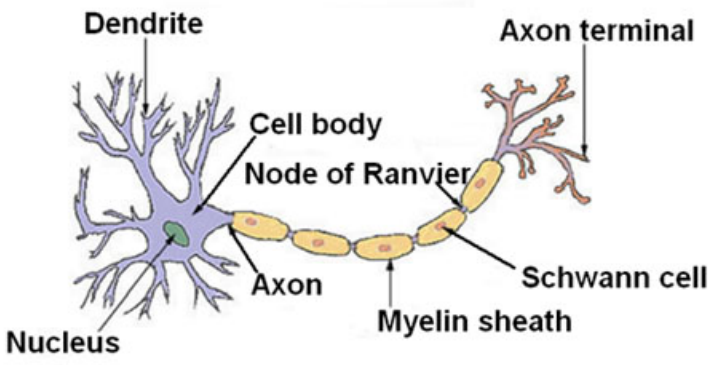


Figure 3.2: The structure of a typical biological neuron [29].

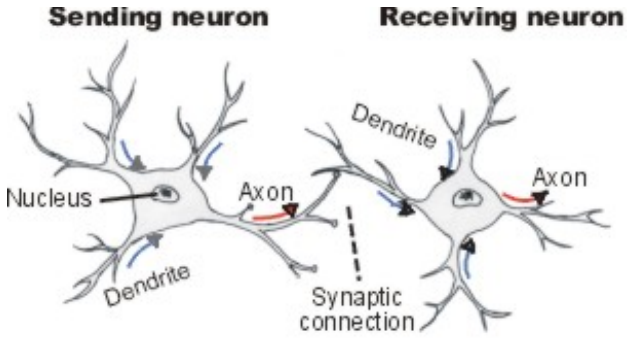
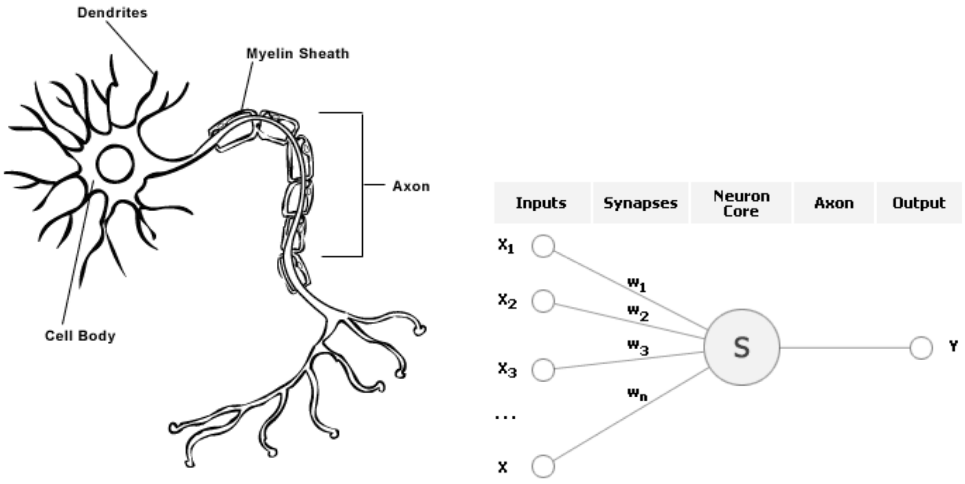


Figure 3.3: Two connected biological neurons [31].

To explain the translation from the biological neuron to a computational model, consider Figure 3.4. As shown in 3.4a and 3.4b, the dendrites on the biological neuron are represented by the inputs and synapses of the mathematical model, the cell body is the neuron core, and the axon retains the name axon but is simply the output of the mathematical neuron.



(a) Biological Neuron [60] (b) Mathematical Model [32]

Figure 3.4: The biological neuron and mathematical counterpart.

The internals of the computational neuron's core contains two things: a summation transfer function which feeds into an activation function. A closer look

at these features can be seen in Figure 3.5. With regard to activation functions, several types are permissible, ranging from a simple step function to a hyperbolic tangent with output typically constrained within [0, 1] or [-1, 1]. Some representative activation functions are shown in Figure 3.6. And finally, when many mathematical neurons are connected together in layers, a resulting artificial neural network forms, as shown in Figure 3.7.

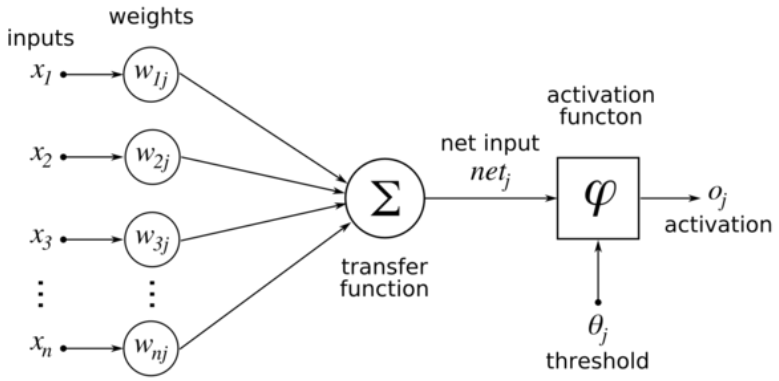


Figure 3.5: Details on the mathematical neuron core [83].

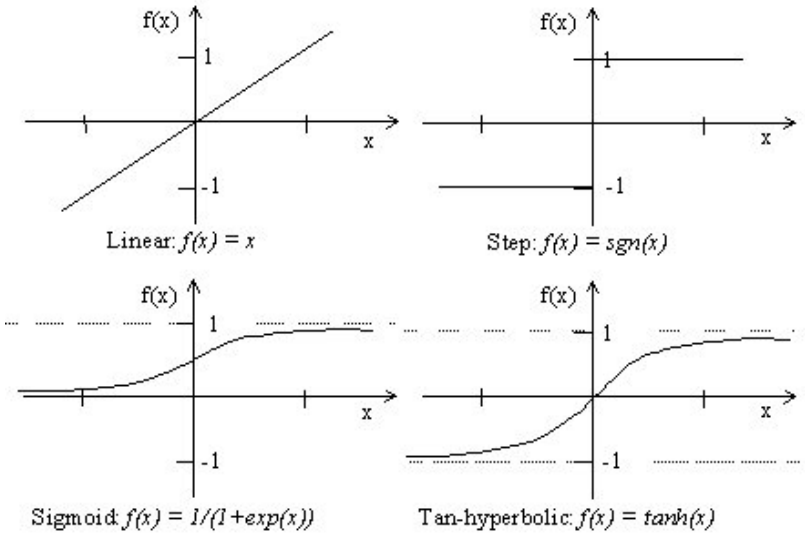


Figure 3.6: Common activation function used in the neuron core [4].

3.2.2 The Backpropagation Learning Algorithm

As noted in the previous chapter, the backpropagation algorithm was first introduced in 1974 [79] yet is still highly applicable today. All neural networks implemented in this thesis will be trained via this algorithm. The mathematical learning function associated with this algorithm will be described visually followed by a high-level perspective on how this learning technique can be utilized in a simple problem.

Since the purpose of this thesis is to propose a novel utilization of the well-established neural network backpropagation learning algorithm rather than to modify or enhance the learning algorithm itself, the algorithm's mathematical details will not be assessed in rigor. Rather, for an in-depth exposition on the backpropagation algorithm, see [64]. However, to summarize, the backpropagation learning algorithm is designed for a multi-layered feedforward neural network, as shown in Figure 3.7.

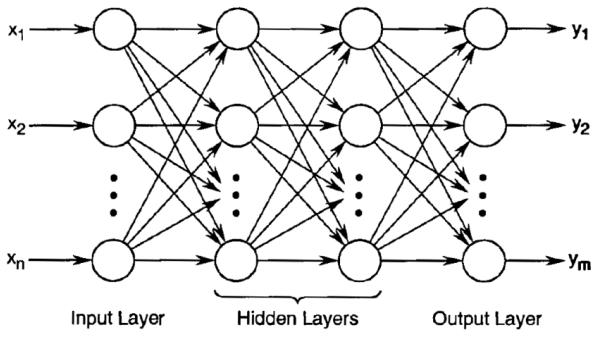


Figure 3.7: This is a standard multi-layered feedforward neural network, as shown in [21].

To train a neural network, the end-goal of training would be to have the network map a set of training inputs (input to the first layer of the neural network) where each individual input has a desired corresponding output value (outputs

on the last layer of the neural network). So, given a multi-layered feed-forward neural network such as the one shown in Figure 3.7, along with a set of inputs and corresponding set of outputs, the backpropagation algorithm trains the neural network on the input/output set so that it learns to map each of the inputs with the desired result. This is accomplished through modifying the weights on each of the synapses that inter-connect the neurons. These weights scale the input to a given neuron, which thereby modifies the output from a neuron's axon. Typically, the weights on a neural network start out randomized, meaning that for each input string the output will be some unmeaningful string since the network has not yet been trained. Therefore, before the backpropagation algorithm runs, an error term can be calculated for each input based upon the current output for that value as compared with the desired output. This (*desired result - actual result*) error term is precisely what the backpropagation algorithm minimizes–thereby reducing the error discrepancy between the real output and desired output of the current neural network. This feedback loop is shown visually in Figure 3.8.

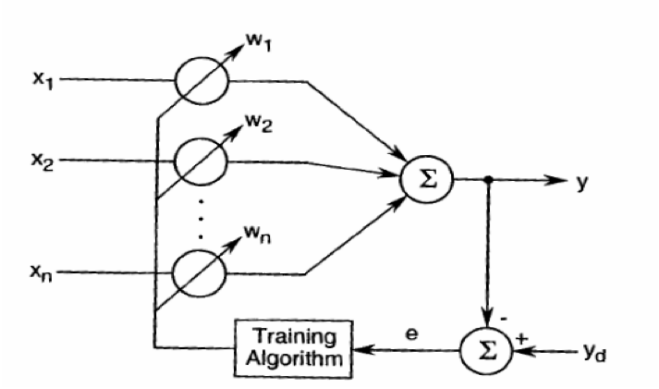


Figure 3.8: The training loop for a neural network, as shown in [21]. In this case the Training Algorithm is the backpropagation algorithm.

The backpropagation algorithm is also iterative–running a variable number of times, where each iteration is a step closer to the global minimum [21]. Also,

since the error term is slowly being minimized during each training iteration of the backpropagation algorithm, the training error can be visualized as a multi-dimensional error-surface, shown in Figure 3.9, where the current state of the neural network error is shown to be the dot on the error surface. Thus, as this figure shows, the end goal of training would be to reach the global minimum on the error surface, which corresponds to the point at which all the desired inputs map as closely as possible to the appropriate outputs. Furthermore, there are two specific parameters that can be fine-tuned on the backpropagation algorithm: learning rate and momentum. The learning rate parameter essentially represents the 'speed' of the virtual point on the error surface, basically specifying the step-size of the point for each iteration. Momentum, on the other hand, represents the 'inertia' of the point meaning that the degree of change to the weight of a given edge in the neural network during one training iteration impacts the change to that weight during the next iteration.

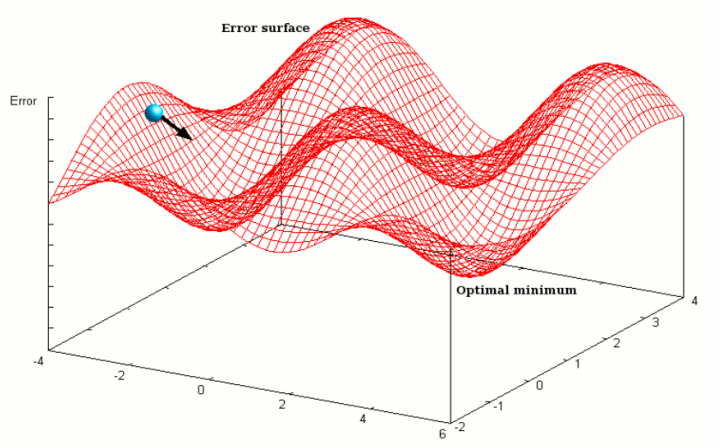
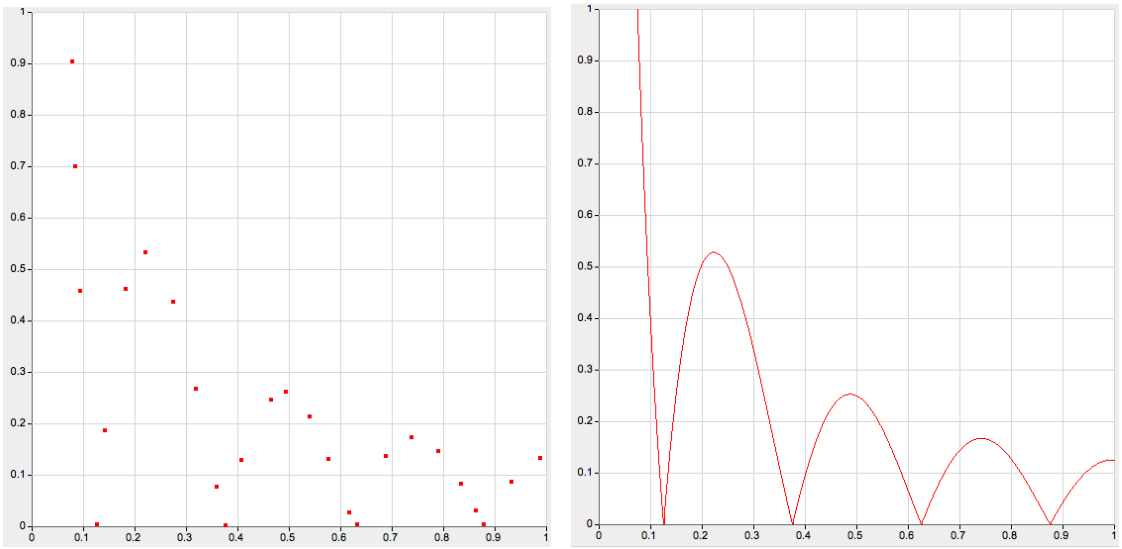


Figure 3.9: An example of a three-dimensional error surface for a training neural network, as shown in [52].

One potential problem while training neural networks is the potential for over-training or over-fitting. Over-training occurs when a network has learned

not only the basic mapping associated with input and output data, but also the subtle nuances and even the errors specific to the training set. If too much training occurs, the network essentially only memorizes the training set and loses its ability to generalize to new data input. The result is a network that performs well on the training set but performs poorly on out-of-sample test data.

One exemplary visual example for the utilization of neural networks would be for filling in the blanks on a mathematical function. Consider Figure 3.10. In this example, the graphed training data as shown in Figure 3.10a would be used by a simple neural network to generate the corresponding neural network output as shown in Figure 3.10b. Though a very simple example, this illustrates an inherently beneficial property of neural networks to "fill-in" the blanks between the training data points.



(a) Graphical training data (b) Trained neural network output

Figure 3.10: Exemplary neural network utilization.

3.2.3 Architectural Design Decisions

In deciding on which type of neural network architecture to implement, a traditional feedforward approach was favored rather than recurrent neural networks, since recurrent neural networks face inherent disadvantages as stated by Alessandro Sperduti:

> ...it is well known that training recurrent networks faces severe problems (Bengio, Simard, and Frasconi, 1994) and the generalization ability might be considerably worse compared to standard feedforward networks (Hammer, 2001). [70]

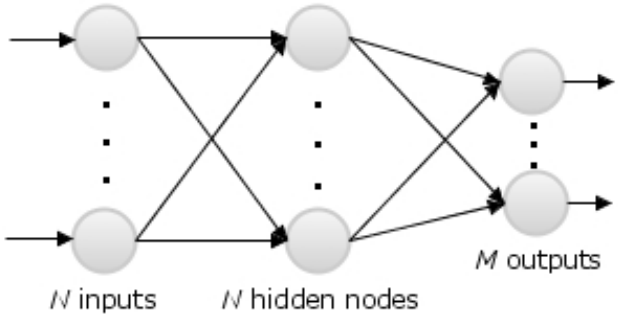
Moreover, several studies have evaluated the computing capacity of multi-layered feed-forward neural networks [51, 68, 76, 80, 81, 82]. One such study found that

> Feed-forward networks with a single hidden layer and trained by least-squares are statistically consistent estimators of arbitrary square-integrable regression functions under certain practically-satisfiable assumptions regarding sampling, target noise, number of hidden units, size of weights, and form of hidden-unit activation function. [80]

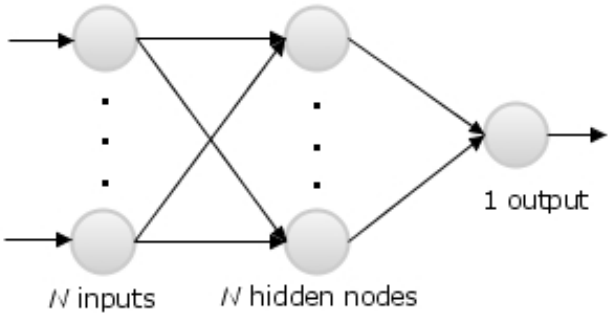
Essentially, such results have proven that multi-layered feed-forward neural networks with a single hidden layer are capable of approximating any real-valued function to any desired degree of precision.

The primary neural network architecture to be utilized in this thesis is a three-layered feed-forward neural network: one input layer, one hidden layer, and one output layer. However, two neural network versions will be utilized, both of which are shown in Figure 3.11. The neural network shown in Figure 3.11a is defined to be a General neural network, whereas the neural network shown in Figure 3.11b is defined as an Expert neural network. These two architectures

are nearly identical and differ only in the number of output nodes in the output layer. For the purposes of this thesis, the value of output nodes on the neural networks correspond to the predicted health state of a corresponding router. Thus, these two architectures differ only in the number of routers they are responsible for. Additional details on the specific utilization of these architectures will be discussed in the next chapter.



(a) A General neural network



(b) An Expert neural network

Figure 3.11: The two neural network architectures utilized.

Chapter 4

Design and Implementation

This chapter will go over the high-level conceptual design for constructing a BGP Failure detection and prediction tool as well as the specific details of the software implementation.

4.1 High-Level Concept Design

This section will discuss interfacing a iBGP network with a neural network, followed by the approach utilized in collecting training data.

4.1.1 Utilizing Neural Networks

The objectives of the design are to interface a generic iBGP network with a neural network, train the neural network with relevant past data, and then utilize the trained neural network along with the current iBGP connection statuses to make predictions and detections regarding the health of the BGP nodes. In order to assist the explanation, consider the visualization of a simple iBGP network as

shown in Figure 4.1. Shown in the figure is the iBGP network of interest in AS 4 and R1-R5 represent BGP routers 1 through 5 in a full mesh of connectivity.

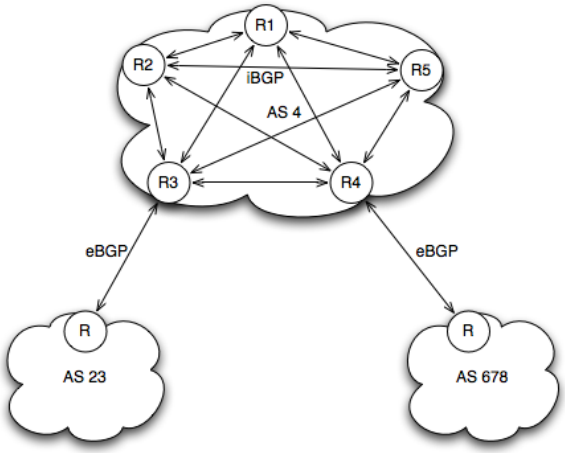


Figure 4.1: An example of a simple BGP network.

In preparation to interface the iBGP network to a neural network, an adjacency matrix is utilized to represent each of the edge connections in the full mesh, as shown in Figure 4.2. In the adjacency matrix, the connection from R2 to R5 (as an arbitrary example), would be found by locating the cell whose column is R2 and whose row is R5. Moreover, in the case of this particular matrix, each entry contains a BGP finite state machine that stores the current state of the corresponding connection from one router to another. In this way, each of the six states associated with representing a peer-to-peer connection can be assigned a numeric value between 0 and 1 representing the degree of connectedness for the given session. With numbers assigned to each potential state, these values can then be the inputs to a neural network, as shown in Figure 4.3a. In this case, the neural network is defined to be a General neural network, since it is responsible for outputing the belief states of all nodes in the iBGP network. An alternative to this approach would be to interface the adjacency matrix with five

unique Expert neural networks, as shown in Figure 4.3b.

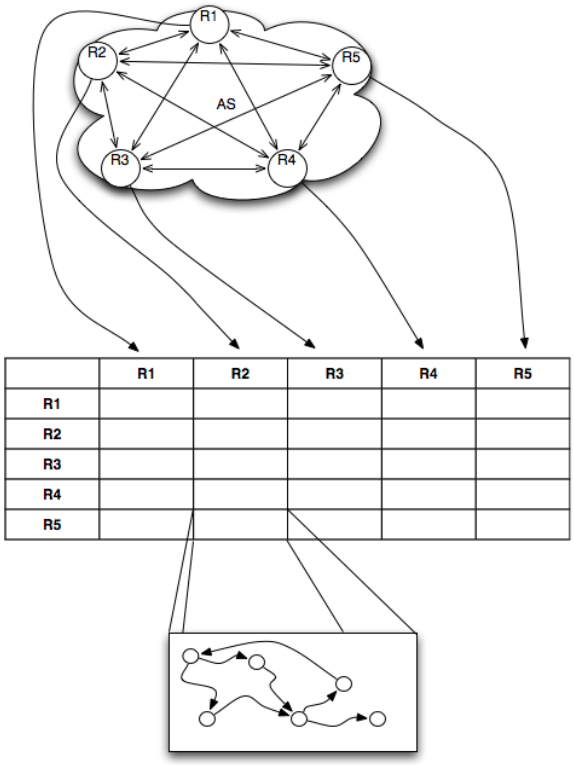
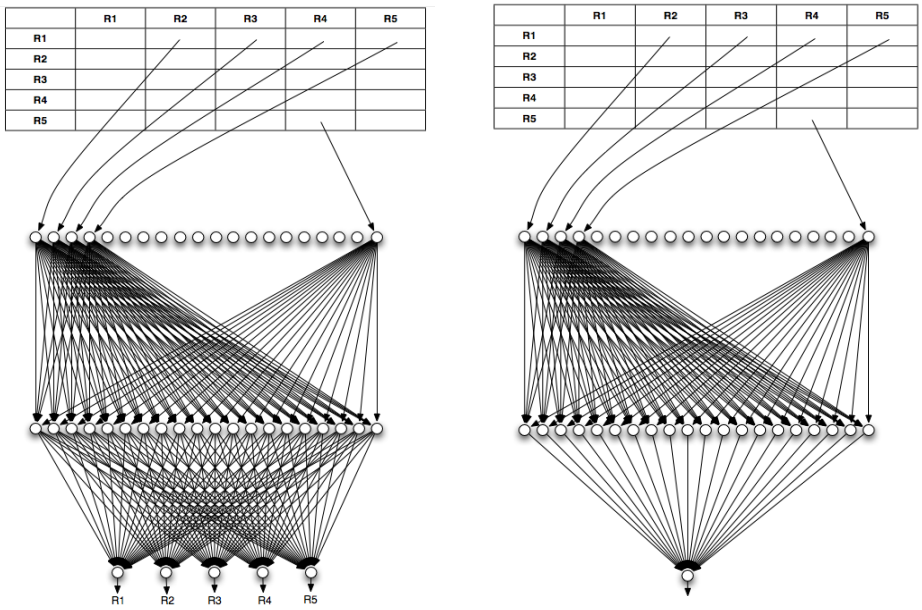


Figure 4.2: Interfacing the iBGP network with an adjacency matrix.

Now, consider the following simple example of how this high-level design should function end-to-end. For this example, the same iBGP network will be utilized with the modification that Router 1 loses connection with three of its neighbors, as shown in Figure 4.4. These lost connections can be followed down into the adjacency matrix and then into the neural network as shown in Figure 4.5. In this example, the neural network sees that even though Router 1 has not lost connection with all of its neighbors, a prediction is made that it is about to go down since the majority of connections has been lost while the other four routers are predicted to remain online. This example, of course, is under the assumption that a similar event has transpired in the past, training data had



(a) Interfacing the iBGP adjacency matrix with a General Neural Network

(b) Interfacing the iBGP adjacency matrix with an Expert Neural Network

Figure 4.3: Interfacing the adjacency matrix to a neural network.

been collected from the event, and the General neural network had been trained with the relevant data.

4.1.2 Training Data Collection Methodology

Though the methodology for collecting training data is not the focal point of this thesis, the design details are important nonetheless. The end goal in ascertaining representative training data would be to select a connectivity pattern from the network prior to a node being detected as going offline. Thus, the first requirement would be to detect whenever a node goes down.

For the purposes of this work, BGP peer failure is defined to be the case where a given BGP speaker loses connection with other speakers. This is considered

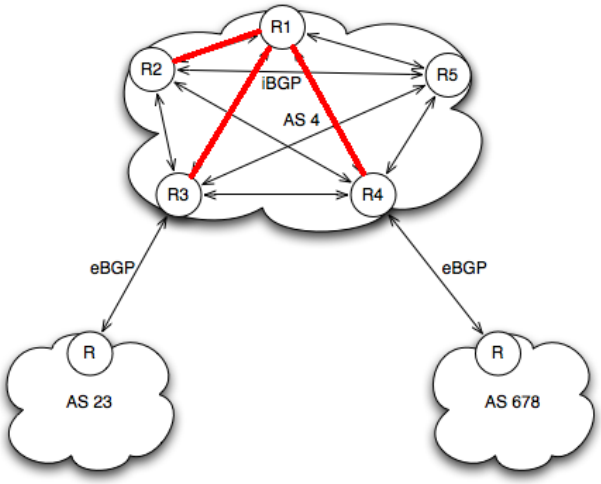


Figure 4.4: Router 1 loses connection to three of its neighbors.

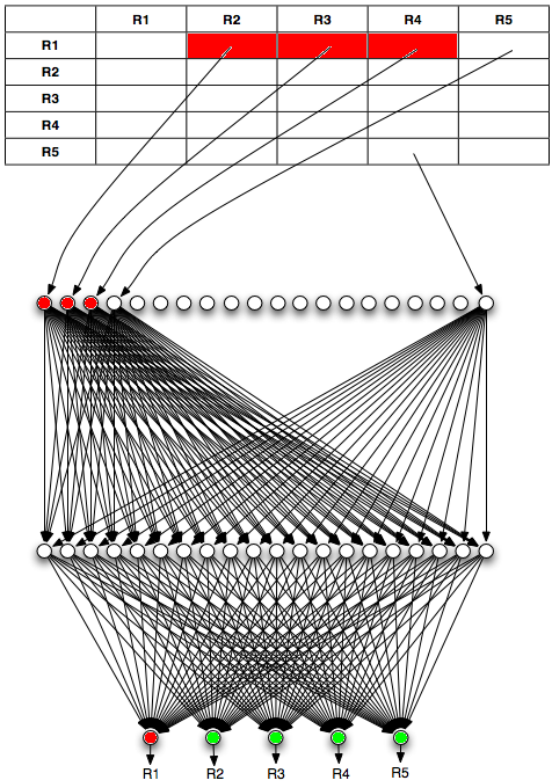


Figure 4.5: The resulting adjacency matrix and neural network output from Router 1 losing three connections.

a failure due to the fact that if a BGP speaker loses connection to its peers, then routing information cannot be transmitted–the BGP peer has failed its job and purpose. More specifically, two types of node-down failure patterns will be considered. First, an "externally-detected-node-down" (EDND) will refer to the case when, given node N, all nodes in the network have lost connection to N. In this case, all routers in the network would be incapable of updating the EDND router's routing table. On the other hand, a "self-detected-node-down" (SDND) will refer to the case when a given node N is found to have lost connection to all nodes it was previously connected to. In this case, the SDND router would be incapable of updating any other peers in the network. Therefore, some pre-processing must be executed to generate both of these possible node-down signatures and stored for reference when monitoring the corresponding network.

With the capability of detecting a node-down established, the next requirement would be to maintain a pattern history list for the network. An addition would be made to this list (or queue) every time there is some session state change within the network. In this way, all patterns leading up to a node failure would be stored for later access as training data. However, this brings to light the final requirement: a way to specify where excactly in the queue history to look for appropriate training data. Thus, some particular `lookback` variable must be defined such that when a node-down is detected, the pattern located within the history queue at a distance of `lookback` will be saved as a pre-cursor prediction pattern for the given node-failure. And lastly, this solution is appropriate for a single node going offline, but there may be multiple correlated nodes that go down as well, say in the case that two routers are on the same power grid during a black-out. In this special case, a different distance specifier may be desired, so a `correlatedLookback` variable should also exist and be independently

configurable from the standard `lookback` variable.

4.2 Software Implementation Details

The Java software written for this thesis is dubbed the BGP Neural Network Framework (BGPNNF). Moreover, note that this software is not designed with the intention of use as a polished tool for a network administrator. Rather, the framework software implementation presented in this section has been designed and written for experimental purposes to prove the hypothesis of this thesis, namely: Neural networks can interface with iBGP computer networks to learn and utilize the precursor connectivity patterns that emerge prior to a node failure. To present the BGPNNF, this section will present the relevant configuration options, the backend programmatic details, and some simple user interface details.

4.2.1 The User Interface

The BGPNNF user interface (UI) is designed solely to assist in the experiments for this thesis and provide a visual representation of the neural network inputs and outputs. Therefore, the UI is rather minimal and basic, as shown in Figure 4.6. The interface was implemented using the org.jdesktop.application-.Application and org.jdesktop.application.Action libraries and utilizes JGraph [3] and JGraphT [59] for the graph visualizations. Lastly, line charts are generated through the use of JFreeChart [49].

The major features of the interface are the row of buttons across the top of the main window, along with two network views. The network views are the most important feature of the user interface–the one on the left displays the

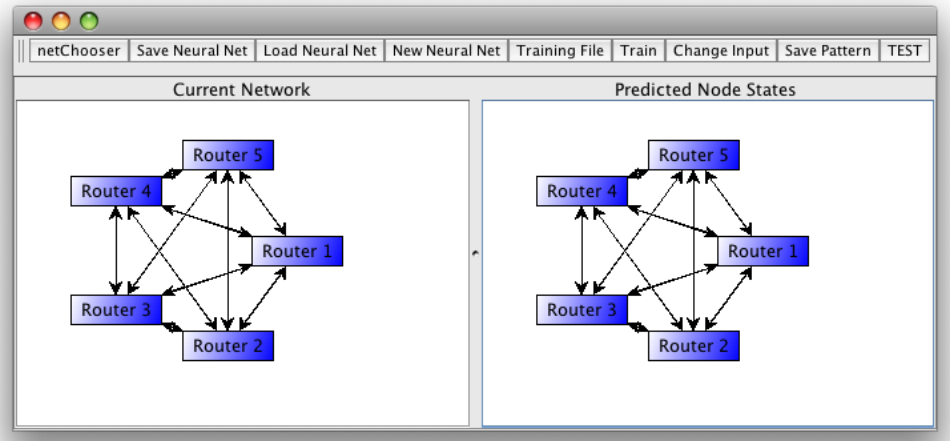


Figure 4.6: The BGPNNF User Interface.

current network connection states (which are the inputs to the internal neural network) whereas the view on the right side are the predicted node states (the output from the neural network(s)). An example of these views in action are shown in Figure 4.7. As shown, the red connections in the Current Network view show the links that are down or Idle and the corresponding Predicted Node States view shows which node is predicted to have issues. In terms of coloring, all inputs/outputs of the neural network are contained within the range [0.0, 1.0]. Thus, as for color displays ranging between green (good) and red (bad) for some given `state` and defining 1 to represent disconnected (0 represents established), the color displays are therefore simply `Color(state, 1 - state, 0)`, where the three parameters are red, green, and blue, respectively.

With regard to the row of buttons, each one has a specific function relevant to experimentation. Each button will be described in terms of functionality, moving from left to right.

Traversing sequentially through the buttons displayed on the UI, the first in

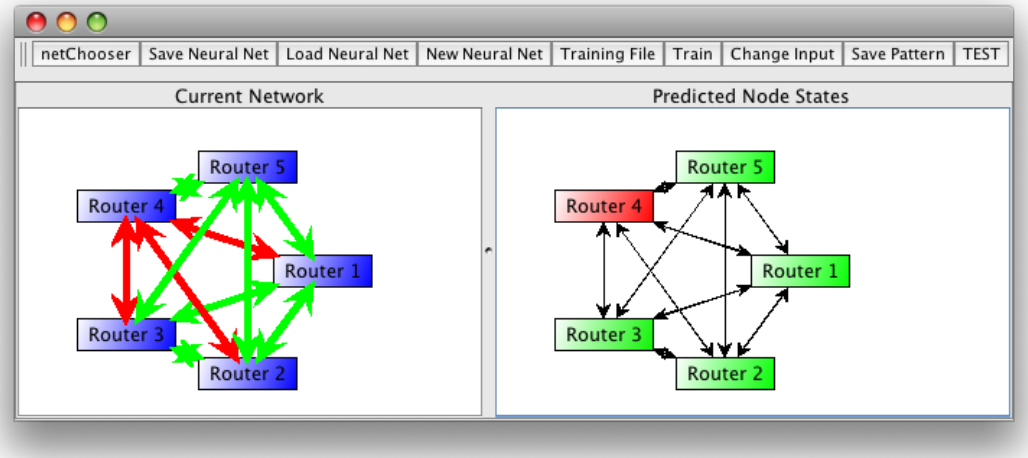


Figure 4.7: Example distinguishing the current view from predicted. In this case, Router 4 has lost the majority of its connections and is therefore predicted to go completely offline by the red coloring in the prediction panel.

question is the netChooser button. This button allows the user to select which Expert neural network will be affected by any of the other buttons shown. As such, this button only appears if a backend Expert neural network implementation is being run.

The next button is the "Save Neural Net" button. This button, as its name implies, serializes the current backend neural network to a file. Upon clicking this button, a standard JFileChooser dialog is displayed to choose the location and file name to save the neural network, as shown in Figure 4.8. This feature is of particular value when, after training a neural network for experimental purposes, it can be stored and returned to at another time. Thus, complementary to the Save button comes the next "Load Neural Net" button. This button functions much like the Save button in that, upon clicking, a JFileChooser dialog is opened, which allows the user to navigate to the previously saved neural network and

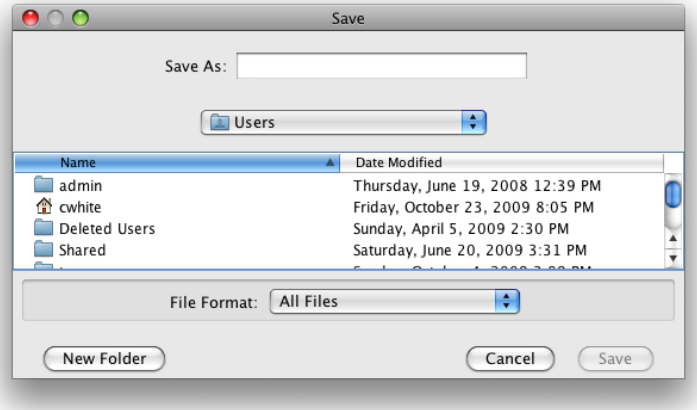


Figure 4.8: Save Dialog User Interface.

restore it, as shown in Figure 4.9.

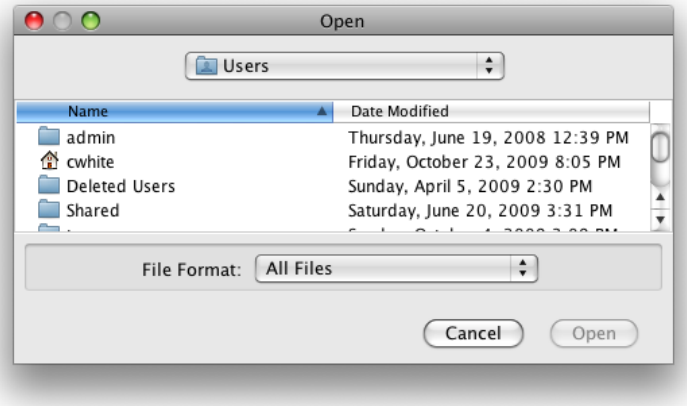


Figure 4.9: Load Dialog User Interface.

Continuing sequentially, the following button is the "New Neural Net" button. This button removes the internal neural network and replaces it with a newly specified neural network. When clicked, this button opens a dialog to specify how many nodes are to be located in the hidden layer (the input and output layers are fixed), as shown in Figure 4.10.

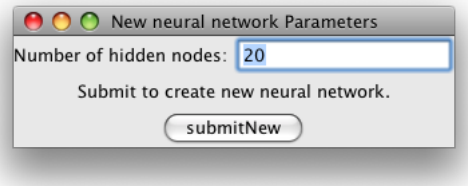


Figure 4.10: New Neural Net User Interface.

The next button on the user interface is titled "Training File". When clicked, this button opens a JFileChoose dialog that allows the user to navigate to the directory and file that contains the training data desired for training the internal neural network. The next button "Train" utilizes the training file to train the internal neural network. When clicked, the Train button opens a dialog as shown in Figure 4.11. As shown in the figure, the training parameters of Learning Rate, Momentum, and number of Iterations must be specified prior to training. Also, a lower-bound can be set for the resulting Root Mean Squared Error (RMSE) of the neural network through the "Desired RMSE" setting which will halt training if the RMSE of the network drops below the specified value. This feature helps to ensure that the neural networks do not become over-trained. Moreover, an additional check-box is available in this panel to optionally graph the RMSE versus iterations of the training.

The remaining three buttons are present strictly for testing and debugging purposes. The "Change Input" button stops and starts the internal SNMP polling mechanism. The "Save Pattern" button allows the current state of the neural network to be saved to the current training file. Lastly, the "TEST" button simply runs an internal test method for debugging purposes.

One final feature of the BGPNNF is a separate panel that launches on startup

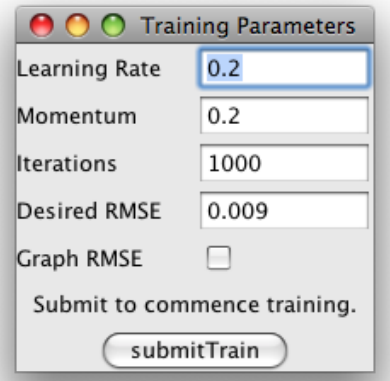


Figure 4.11: Train Button User Interface.

entitled the "Input Test Network Display", as shown in Figure 4.12. This UI allows a user to manipulate the individual session states between the displayed routers. As shown, a radio button can be selected as either "down" or "up", corresponding to the session state to be set as either up (established connection) or down (idle connection). Once the radio button has been set to the desired value, the user then clicks one router (session origin) followed by clicking a second router (session destination). The directed session between the two machines is then set to the value of the positioned radio button. This feature will allow for various combinations of testing to be performed separate from when the software is directly connected with and polling the hardware routers.

4.2.2 Relevant Configuration Options

The configuration utility of the BGPNNF parses an XML file that specifies the network topology, neural network type, number of nodes in the neural network hidden layer, and the size and lookback options for the `historyQueue`

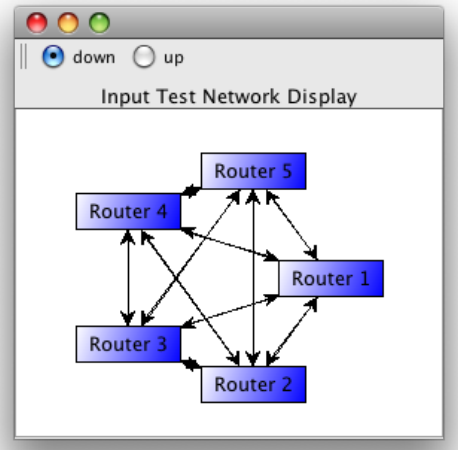


Figure 4.12: The input test network user interface.

of pattern changes in the network. The XML parser of the BPGNNF utilizes javax.xml.parsers.Document-Builder, javax.xml.parsers.DocumentBuilderFactory, along with org.w3c.dom.Document, org.w3c.dom.Element, org.w3c.dom.Node, and org.w3c.dom.NodeList. In order to elaborate upon the configuration options specifiable within the configuration file, example XML code in conjunction with visuals will be presented.

By far, the most important configuration option for the BGPNNF is specifying the network topology of interest. In order to specify a topology, a list of nodes (routers) must be defined in terms of name and interface IPs with corresponding lists of adjacent IPs. The resulting design to meet these requirements is shown in Code Block 4.1. In this case, the router name is simply Router 1 and contains two different interfaces with IPs 208.94.60.1 and 208.94.60.15. Within each interface tag are adjacent tags, which represent which peer IPs can be reached from each interface. Thus, 208.94.60.1 has one adjacent peer, namely 208.94.60.2, whereas interface 208.94.60.15 has two peers: 208.94.60.16 and 208.94.60.20. In order

to actually specify a complete network topology, however, many nodes must be defined. One example is shown in Code Block 4.2, which specifies the simple five-node fully-connected network as shown in Figure 4.13.

```
<node name="Router 1">
   <interface IP="208.94.60.1">
      <adjacent IP="208.94.60.2" />
   </interface>
   <interface IP="208.94.60.15">
       <adjacent IP="208.94.60.16" />
       <adjacent IP="208.94.60.20" />
   </interface>
</node>
```

Code Block 4.1: Example Node XML Element and Contents

With the network topology specified, the next most important configuration option is the type of neural network architecture to use and interface with the defined network topology. For the purposes of this thesis, the type of neural network must be specified as either a General neural network or an Expert neural network within the XML `neuralNet` element. An example of a General neural network configuration is shown in Code Block 4.3. As shown in the example, to choose a General neural network the `expert` attribute of the `neuralNet` element must be set to false. Moreover, since the neural network contains a hidden layer, the number of nodes in that layer must be specified with the `hiddenNodes` element

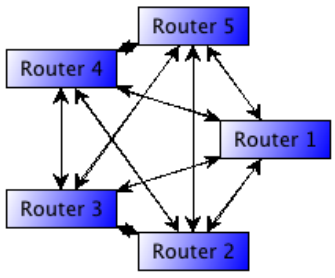


Figure 4.13: A simple fully-connected five-node network, as specified by Code Block 4.2.

and the number of nodes set to the `num` attribute. Lastly, the training data for the neural network is stored within an external text file, so the external file is specified within the `trainingData` element under the `file` attribute. In this example, the training file is simply set to "trainingData.txt".

On the other hand, an Expert neural network configuration is shown in Code Block 4.4. In the Expert case, the `expert` attribute is now set to true, but the biggest differentiating characteristic from the General neural network con-

```
<node name="Router 1">
   <interface IP="208.94.60.1">
      <adjacent IP="208.94.60.2" />
      <adjacent IP="208.94.60.3" />
      <adjacent IP="208.94.60.4" />
      <adjacent IP="208.94.60.5" />
   </interface>
</node>
<node name="Router 2">
   <interface IP="208.94.60.2">
      <adjacent IP="208.94.60.1" />
      <adjacent IP="208.94.60.3" />
      <adjacent IP="208.94.60.4" />
      <adjacent IP="208.94.60.5" />
   </interface>
</node>
<node name="Router 3">
   <interface IP="208.94.60.3">
      <adjacent IP="208.94.60.1" />
         <adjacent IP="208.94.60.2" />
      <adjacent IP="208.94.60.4" />
      <adjacent IP="208.94.60.5" />
   </interface>
</node>
<node name="Router 4">
   <interface IP="208.94.60.4">
      <adjacent IP="208.94.60.1" />
      <adjacent IP="208.94.60.2" />
      <adjacent IP="208.94.60.3" />
      <adjacent IP="208.94.60.5" />
   </interface>
</node>
<node name="Router 5">
   <interface IP="208.94.60.5">
      <adjacent IP="208.94.60.1" />
      <adjacent IP="208.94.60.2" />
      <adjacent IP="208.94.60.3" />
      <adjacent IP="208.94.60.4" />
   </interface>
</node>
```

Code Block 4.2: Example Router Configuration

figuration is the specification of training data files. In the General case, only one training file was specified since there is only a single neural network. In the case of the Experts, alternatively, since there is one expert per node in the network, there must be one training file per node. The `trainingData` element is utilized once again, but the `router` element is now utilized to specify which training file correponds to which router, therefore assigning the file to the neural network responsible for the given router. For example, in the case of the first `trainingData` element, the neural network responsible for Router 1 would use the expertData1.txt training file.

```
<neuralNet expert="false">
   <hiddenNodes num="12" />
   <trainingData file="trainingData.txt" />
</neuralNet>
```

Code Block 4.3: Example neuralNet Element for Configuring a General Neural Network

```
<neuralNet expert="true">
   <hiddenNodes num="20" />
   <trainingData router="Router 1" file="expertData1.txt" />
   <trainingData router="Router 2" file="expertData2.txt" />
   <trainingData router="Router 3" file="expertData3.txt" />
   <trainingData router="Router 4" file="expertData4.txt" />
   <trainingData router="Router 5" file="expertData5.txt" />
</neuralNet>
```

Code Block 4.4: Example neuralNet Element for Configuring an Expert Neural Network

The last configuration options of interest are the settings associated with the `historyQueue`. A sample configuration is shown in Code Block 4.5. The three configurable settings are as follows: 1) the size of the queue (essentially the length of the sliding-window of retained history), 2) the length of `lookback` for a single node down, and 3) the length of `correlatedLookback` for multiple

nodes down. As shown in the example, the size of the queue is set using the `size` attribute in the `historyQueue` element. The second configuration is the lookback number, which, when a node-down is detected, specifies how far back in the queue to look for the pre-cursor pattern to use as training data for the neural network. Again, as shown in the example, this is specified by setting the `lookback` attribute in the `historyQueue` element. Lastly, when multiple nodes go down, a different lookback history may be desired, which can be specified by setting the `correlatedLookback` attribute, which is set to 4 in the example.

```
1    <historyQueue size="10" lookback="2" correlatedLookback="4"/>
```

Code Block 4.5: Example Configuration of the historyQueue

4.2.3 Backend Programmatic Details

The primary points of interest for the backend implementation consist of initiating the SNMP polling mechanism, creating the signature strings for detecting a node-down, initializing the `historyQueue`, constructing the neural network from the configuration file, and instantiating the `CoreModel`. The descriptions will follow in line with the high-level flow diagram as shown in Figure 4.14.

Creating the SNMP Poller

The SNMP Poller modual utilizes the Java SNMP4j [28] library to sequentially poll each router, as specified in the configuration file. So, the only required information is a list of nodes to poll and this is accomplished through a list of custom `NodeInfo` objects. The primary components of a `NodeInfo` object are shown in Code Block 4.6. As shown in the code, the `String name` repre-

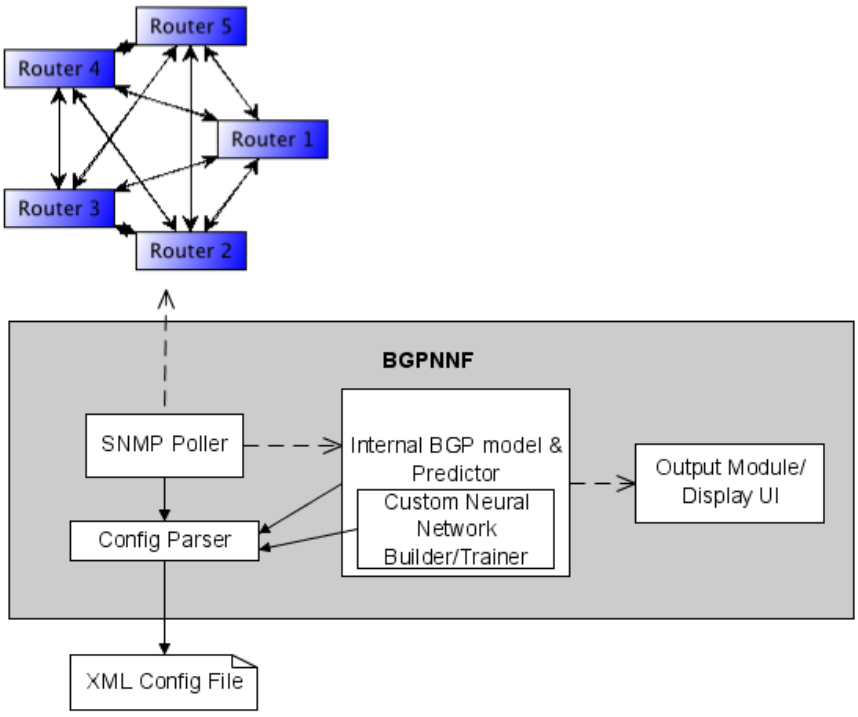


Figure 4.14: The BGPNNF Flow Diagram.

sents the name of the router. The `String[] interfaceIPs` represent the list of interfaces, ie ethernet ports, by IP on the named router. Lastly, each of the String IP addresses located within the interfaceIPs array serve as a key in the `HashMap<String,String[]> adjacent` map, and the corresponding value is another `String` array comprised of all the external IPs that this router is connected to.

```
1    private String name;
2    private String[] interfaceIPs;
3    private HashMap<String, String[]> adjacent;
```

Code Block 4.6: NodeInfo Private Variables.

Once the `NodeInfo` list is built, then polling can commence when the "ChangeInput" button on the UI is clicked. Polling is conducted as shown in Code Block

4.7.

```
1  private boolean poll; //boolean value for continuous polling
2  while(poll)
3  {
4     for (int i = 0; i < nodes.length; i++)
5     {
6        String[] interfaceIPs = nodes[i].getInterfaceIPs();
7        for (int j = 0; j < interfaceIPs.length; j++) {
8
9           String[] adjacentIPs = nodes[i].getInterfaces().get(interfaceIPs[j]);
10          for (int k = 0; k < adjacentIPs.length; k++) {
11
12             nodeState = pinger.snmpGet(interfaceIPs[j], bgpSNMP.READ_COMMUNITY,
13                                        bgpSNMP.OID_BGP_PEER_STATE + adjacentIPs[k]);
14
15             //update the CoreModel
16             //'6' corresponds to Established
17             if (nodeState.equals("6")) {
18                model.update(interfaceIPs[j], adjacentIPs[k], 0);
19             }
20              //If not established, then the session is considered down
21             else {
22                model.update(interfaceIPs[j], adjacentIPs[k], 1);
23             }
24          }
25       }
26    }
27 }
```

Code Block 4.7: SNMP Poller Main Loop.

Creating the Node-Down Detection Signature Strings

Creating the node-down detection signature strings is the last pre-processing that takes place prior to the start of SNMP polling. This requires only the `NodeInfo` array and every possible EDND and SDND pattern is recorded into the `nodeDownPatternList` private variable of the `CoreModel`, as shown in Code Block 4.8. These signatures are then utilized during each update if the `matrix` has changed. If there is a change, the `nodeDownCheck()` method is called, which is shown in Code Block 4.9. Note that the goal of `nodeDownCheck()` is to determine if the current `matrix` pattern is a *subset* of any of the strings in the `nodeDownPatternList`. This ensures that a node will be detected as down in

spite of other noise in the network.

History Queue

The `private OQueue historyQueue` of the CoreModel is a fairly standard queue that maintains a finite interval of history, as specified in the configuration file. In addition to the normal `enqueue(double[] matrix)` and `peak()` methods is the `lookback(int distance)` method, as shown in Code Block 4.10. This method provides the capability of procuring a matrix pattern from a specified `distance` in the queue's retained history. Also, one distinguishing characteristic of this particular queue is that no `dequeue()` method is ever called–it simply keeps a finite ordered list of network pattern history. Thus, whenever a pattern is enqueued that causes the queue to grow larger than the maximum size (as specified in the configuration file), the oldest pattern is removed and the new pattern is appended to the head of the queue.

Building the Neural Network

In order to build a neural network, first the network topology must be gathered out of the configuration file and stored in a `NodeInfo` data structure. The software used to build and train the neural networks in this thesis is the Java Object Oriented Neural Engine (JOONE) [53].

Once a full `NodeInfo` list is generated, the configuration utility can conclude the number of nodes within the network simply by the length of the `NodeInfo` list and also infers the number of total connections within the network based upon the total number of entries within the `String[]` of the `adjacent` variable. These values are used to construct the neural network since the num-

```
1  public void createNodeDownList()
2  {
3     nodeDownPatternList = new ArrayList<double[]>();
4     ArrayList<Integer> linkList = new ArrayList<Integer>();
5
6     //Loop for SDND patterns
7     for (int i = 0; i < nodes.length; i++) {
8        String[] interfaceIPs = nodes[i].getInterfaceIPs();
9        for (int j = 0; j < interfaceIPs.length; j++) {
10          String[] adjacentIPs = nodes[i].getInterfaces().get(interfaceIPs[j]);
11          for (int k = 0; k < adjacentIPs.length; k++) {
12              linkList.add(ipMap.get(interfaceIPs[j] + adjacentIPs[k]));
13          }
14       }
15
16       //transform the list of links
17       double[] pattern = new double[matrix.length];
18       for (int a = 0; a < pattern.length; a++) {
19          pattern[a] = 0.0;
20       }
21       for (Integer idx : linkList) {
22          pattern[idx] = 1.0;
23       }
24       nodeDownPatternList.add(pattern);
25       linkList.clear();
26    }
27
28    //Loop for EDND patterns
29    for (int i = 0; i < nodes.length; i++) {
30       String[] interfaceIPs1 = nodes[i].getInterfaceIPs();
31       for (int j = 0; j < interfaceIPs1.length; j++) {
32          for (int l = 0; l < nodes.length; l++) {
33             String[] interfaceIPs2 = nodes[l].getInterfaceIPs();
34             for (int m = 0; m < interfaceIPs2.length; m++) {
35                String[] adjacentIPs = nodes[l].getInterfaces().get(interfaceIPs2[m]);
36                for (int k = 0; k < adjacentIPs.length; k++) {
37                   if (adjacentIPs[k].equals(interfaceIPs1[j])) {
38                      linkList.add(nameMap.get(interfaceIPs2[j] + adjacentIPs[k]));
39                   }
40                }
41             }
42          }
43       }
44
45       //transform the list of links
46       double[] pattern = new double[matrix.length];
47       for (int a = 0; a < pattern.length; a++)
48       {
49          pattern[a] = 0.0;
50       }
51       for (Integer idx : linkList)
52       {
53          pattern[idx] = 1.0;
54       }
55       nodeDownPatternList.add(pattern);
56       linkList.clear();
57    }
58 }
```

Code Block 4.8: The createNodeDownList Method

```
1   private ArrayList<String> nodeDownCheck() {
2      //  boolean returnVal = false;
3      ArrayList<String> names = new ArrayList<String>();
4
5      for (double[] d : nodeDownPatternList)
6      {
7         for (int i = 0 ; i < matrix.length; i++)
8         {
9            if (d[i] == 1.0 && matrix[i] != 1.0) //not detecting a node down
10           {
11            break;
12           }
13           if (i == (matrix.length - 1)) //node down detected
14           {
15              //add name to the list
16              names.add(patternToName.get(arrayToString(d)));
17           }
18        }
19     }
20     return names;
21  }
```

Code Block 4.9: The nodeDownCheck Method

```
1   public double[] lookback(int distance)
2   {
3      Node temp = head;
4      for (int i = 0; i < lookback; i++)
5      {
6         temp = temp.next;
7      }
8
9      return temp.element;
10  }
```

Code Block 4.10: The OQueue's lookback Method

ber of total connections equates to the number of inputs to the neural network and the number of nodes in the network topology is equal to the number of outputs in the neural network. These values are used as the parameters for the `initNet(int inputNum, int hiddenNum, int outputNum)` method of a `CustomNeuralNetwork.java` class, as shown in Code Block 4.11.

```
1   public void initNet(int inputNum, int hiddenNum, int outputNum) {
2      nnet = new NeuralNet();      // create a new Joone Neural Network
3      unTrained = true;
4      this.inputNum = inputNum;
5      this.outputNum = outputNum;
6
7      input = new LinearLayer();
8      SigmoidLayer hidden = new SigmoidLayer();
9      output = new SigmoidLayer();
10
11     input.setRows(inputNum);
12     hidden.setRows(hiddenNum);
13     output.setRows(outputNum);
14
15      /* From input layer to the hidden layer */
16     FullSynapse synapse_IH = new FullSynapse();
17
18     /* From the hidden layer to the output layer */
19     FullSynapse synapse_HO = new FullSynapse();
20
21     input.addOutputSynapse(synapse_IH);
22     hidden.addInputSynapse(synapse_IH);
23     hidden.addOutputSynapse(synapse_HO);
24     output.addInputSynapse(synapse_HO);
25
26     /* Add all of the layers */
27     nnet.addLayer(input, NeuralNet.INPUT_LAYER);
28     nnet.addLayer(hidden, NeuralNet.HIDDEN_LAYER);
29     nnet.addLayer(output, NeuralNet.OUTPUT_LAYER);
30  }
```

Code Block 4.11: The initNeuralNet Method

In order to train a neural network with JOONE, the input and output data must be stored either in memory or within a simple text file. Due to the numerous experiments conducted for this thesis, training data is stored in text files for the purpose of simple retention between runs and experiments. To show the required formats and describe how Joone utilizes the external training data, consider the following simple XOR example, as shown in Figure 4.15. In order to train this network with the desired XOR functionality as shown in Table 4.1,

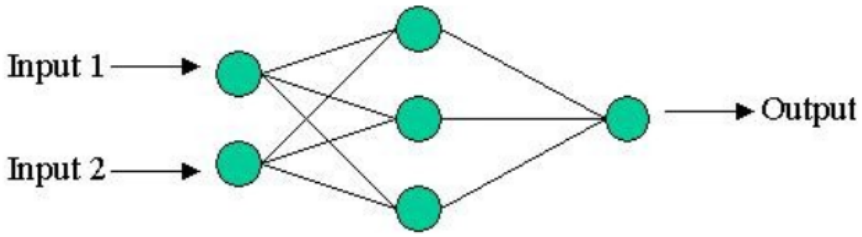


Figure 4.15: A standard XOR Neural Network architecture, as shown in [52].

the corresponding training data must be formatted as shown in Code Block 4.12. As shown in the Code Block, the training data format are semicolon-delimited double values where the inputs are listed sequentially and correspond to each input to the neural network, followed by the desired outputs. Also, one training pattern appears per line. Thus, when this neural network is constructed programatically, it can then be trained via the back-propagation algorithm built within Joone utilizing the training data file.

Input 1	Input 2	Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.1: XOR truth table

```
1 0.0;0.0;0.0
2 0.0;1.0;1.0
3 1.0;0.0;1.0
4 1.0;1.0;0.0
```

Code Block 4.12: The XOR Training Data

Initiating the Core Model

The CoreModel contains the neural network object and interfaces with the SNMP poller modual, as shown in Figure 4.14. The primary components of the CoreModel are the internal variables: `CustomNeuralNet[] nnet`, `double[] matrix`, `ArrayList<double[]> nodeDownPatternList`, `private OQueue historyQueue` and `HashMap<String, Integer> ipMap`. First off, the `nnet` array is simply a list of the custom neural networks, as initialized in the configuration. If the configuration is for a General neural network, then this array is only of length 1. However, in an Expert implementation the length of the array is then equal to the number of nodes in the specified BGP network topology. Next, the `matrix` represents the adjacency matrix of the high-level design–the interface from the BGP network to the neural network inputs. Thus, the length of this matrix array is equal to the total number of connections in the specified BGP network configuration and the value in each index represents a numeric equivalent to the session state between two peers. In the case of the extremes, an Idle (disconnected) state is represented by a 1 whereas an Established (connected) state is represented by a 0. Both the `historyQueue` and `nodeDownPatternList` have been described in prior subsections. The `ipMap` contains a mapping of all the connections (from one interface IP of a router to the connecting interface IP of another), and is instantiated as shown in Code Block 4.13.

The `ipMap` does come with one assumption for the hash function to work appropriately, as follows: Let the set of all IP interfaces in a configuration file be **IPs** and let '$\cdot$' represent concatenation, $\forall$ ip1, ip2 $\in$ **IPs** where ip1 $\neq$ ip2, then ip1$\cdot$ip2 $\neq$ ip2$\cdot$ip1. This assumption assures the uniqueness of each hash key and is considered an acceptable assumption for the purposes of this thesis.

```
1  private HashMap<String, Integer> ipMap;
2
3  public void createIPMap(NodeInfo[] nodes)
4  {
5     nodes;                 // A populated array of NodeInfo objects
6     int indexNumber = 0;  // Index in the model matrix array
7     for (int i = 0; i < nodes.length; i++)
8     {
9        String[] interfaceIPs = nodes[i].getInterfaceIPs();
10       for (int j = 0; j < interfaceIPs.length; j++)
11       {
12          String[] adjacentIPs = nodes[i].getInterfaces().get(interfaceIPs[j]);
13          for (int k = 0; k < adjacentIPs.length; k++)
14          {
15              // The IP from the router's interface to the adjacent peer is mapped
16              // to the index of indexNumber in the CoreModel's matrix array
17             ipMap.put(interfaceIPs[j] + adjacentIPs[k], indexNumber);
18             indexNumber++;
19          }
20       }
21    }
22 }
```

Code Block 4.13: Creating the IP Map for the CoreModel's Internal Matrix

With the `CoreModel` instantiated, it can now handle updates from the SNMP Poller module as shown in Code Block 4.14. This function is really the heart of the BGPNNF and will therefore be assessed with additional scrutiny. Traversing sequentially through the code, line 8 updates the internal matrix on the current state of the session between the two IP addresses from the parameters of the method. Line 11 checks if this update has changed the network connectivity and, if so, the new pattern is saved to the `historyQueue` on line 13. With a state change within the network, lines 16-27 determine whether a General or Expert neural network is being utilized and extracts the output pattern into the `netOutput` array. Next, on line 30, the `nodeDownCheck()` method is called, which returns a list of nodes that are considered down. Lines 33-41 decide whether one or more nodes are down–if more than one node is down, then the recorded pattern is taken from a distance of `CORRELATED_LOOKBACK` in the `historyQueue`, otherwise a distance of `LOOKBACK` is used. The `makeStringPattern(double[],`

`ArrayList<String>)` method can be seen Code Block 4.15 , and simply takes the `double[]` array and creates a formatted string to write to the current neural network's training file. Lastly, the UI is updated–line 44 updates the current network UI whereas line 47 utilizes the netOutput values to update the predicted network UI.

```
1  private networkView predictedNW; //Visual predicted network graph UI
2
3  public void update(String ipFrom, String ipTo, double val)
4  {
5     double[] netOutput; //output from the neural network(s)
6
7     //update the matrix
8     matrix[nameMap.get(ipFrom + ipTo)] = val;
9
10    //add the pattern to the queue history if there has been a state change
11    if (!equals(matrix, historyQueue.peak()))
12    {
13       historyQueue.enqueue(matrix.clone());
14
15       //check for experts here
16       if (nnet.length == 1) //not an expert..
17       {
18          netOutput = nnet[0].interrogate(matrix);
19       }
20       else //expert
21       {
22          netOutput = new double[nnet.length];
23          for (int i = 0; i < nnet.length; i++)
24          {
25             netOutput[i] = nnet[i].interrogate(getMatrixSig())[0];
26          }
27       }
28
29       //see if anything is down
30       names = nodeDownCheck();
31       if (!names.isEmpty())
32       {
33          //check for multiple nodes down
34          if (names.size() > 1)
35          {
36             makeStringPattern(historyQueue.lookback(CORRELATED_LOOKBACK), names);
37          }
38          else //single node down
39          {
40             makeStringPattern(historyQueue.lookback(LOOKBACK), names);
41          }
42       }
43       //change UI current network edge display
44       currentNW.setEdgeState(ipFrom, ipTo, (float)val);
45
46       //update the UI predicted nw display
47       predictedNW.setPredictions(netOutput);
48    }
49 }
```

Code Block 4.14: CoreModel Update Method.

```
1   public void makeStrPattern(double[] pattern, ArrayList<String> names)
2   {
3      String str = "";
4      int[] indices;  //indices to which outputs of the neural network should be 1
5
6      indices = new int[names.size()];
7      //save indicies
8      for (int i=0; i < indices.length; i++)
9      {
10        indices[i] = outputNodes.get(names.get(i));
11     }
12
13     //create the pattern to write to file
14     for (double d: pattern)
15     {
16        str += d + ";";
17     }
18
19     if (nnet.length == 1) //general neural network
20     {
21        for (int i = 0; i < outputNodes.size(); i++)
22        {
23           if (!arrayContains(indices, i))
24           {
25              str += "0.0;";
26           }
27           else
28           {
29              str +="1.0;";
30           }
31        }
32        //end the string with a newline
33        str = str.substring(0,str.length() - 1) + "\n";
34
35        //The general neural network is always index 0
36        nnet[0].updateTrainingFile(str);
37     }
38     else
39     {
40        for (int i = 0; i < nnet.length; i++)
41        {
42           if (!arrayContains(indices, i))
43           {
44              nnet[i].updateTrainingFile((str + "0.0\n"));
45           }
46           else
47           {
48              nnet[i].updateTrainingFile((str + "1.0\n"));
49           }
50        }
51     }
52  }
```

Code Block 4.15: CoreModel makeStringPattern Method.

Chapter 5

Experiments

This chapter will detail the exact experiments conducted utilizing the BG-PNNF.

5.1 Test Suite

The following set of tests will be performed for each network topology:

1. Node-down detection for each node in the network.
2. Numerous connections lost for a single node as a prediction of a node-down state.
3. Predictive correlated nodes-down. For example, if two nodes are always seen to go offline together, then detecting one going down should supply a higher weight for predicting the second.

These test cases are representative in the problem domain of a router going offline. From a network administrator's perspective, the bare-minimum desired

functionality of a network monitoring tool would be to at least detect when a node is down (test 1). However, even more beneficial would be a tool that can infer or predict when a node is about to encounter problems by assessing current conditions (test 2). Even further, if problems with one machine has been directly correlated with another in the past, then a tool that would remember such cases and alert on that repeated possibility would also be of high value (test 3).

Each of these cases will be conducted with first one General neural network and followed with an Expert neural network implementation. Additionally, the number of required patterns to train each neural network will be documented along with the learning rate and momentum backpropagation coefficients, the number of iterations of training, and the required training time. The hardware used for training is documented in Appendix A.

5.2 Experimental Network 1

The first experiment was conducted on a small fully-connected BGP network, as shown in Figure 5.1. This network was constructed in Cal Poly's Cisco Networks lab and the BGPNNF was connected to it via a hub. Once the network was constructed, BGP speakers were disconnected from the network, simulating either an EDND or an SDND, depending upon the location of the BGP SNMP poller. In this way, patterns were detected for nodes going down and saved to the neural network training file(s). The `lookback` and `correlatedLookback` variables were set to 1 and 2 respectively.

Before assessing the General neural network and Expert neural network implementations, Figure 5.2 shows exemplary patterns memorized by the neural network(s) for both cases of EDND and SDND.

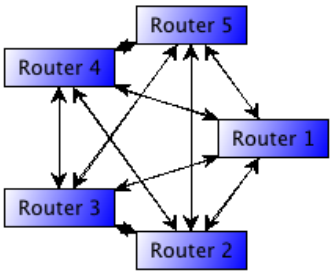
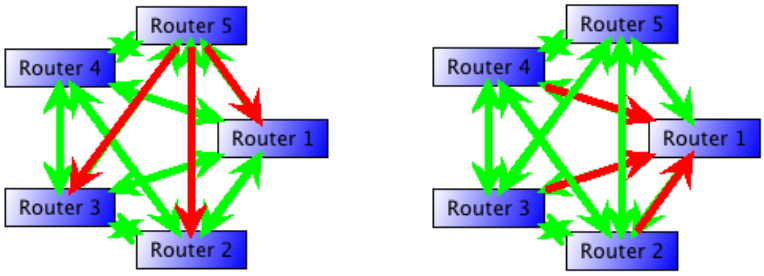


Figure 5.1: The first experimental network for this thesis.



(a) SDND Pattern for Router 5 (b) EDND Pattern for Router 1

Figure 5.2: Examples of the two types of patterns memorized for each Router in the network.

In addition to learning standard patterns for each individual machine, one additional pattern was memorized for a correlation when one machine going down is a precursor for another machine to encounter issues. The particular pattern memorized is shown in Figure 5.3. The reason this pattern has been memorized is due to Router 4 going offline shortly after (thereby triggering the distance of history to be recorded `correlatedLookback`).

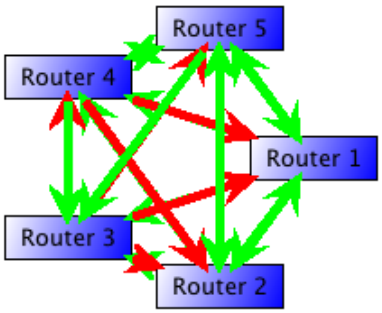


Figure 5.3: Pattern memorized for correlated routers down. Router 3 has SDND and Router 4 has lost connection to Router 1.

5.2.1 General Neural Network Implementation

In the case of the General neural network, a total of twelve different preconditional node-down patterns were collected for training data–an EDND and an SDND for each node in the network, one all-green pattern (all sessions established with all nodes up), and the correlated pattern. The neural network contains a hidden layer of 20 nodes and was then trained for 1000 iterations with a momentum coefficient of 0.2 with a resulting RMSE of 0.0369 or 3.69%, having taken 1.19 seconds. A graph of the RMSE as compared with the training iterations can be seen in Figure 5.4.

With training data acquired and training finished, the test suite experiments

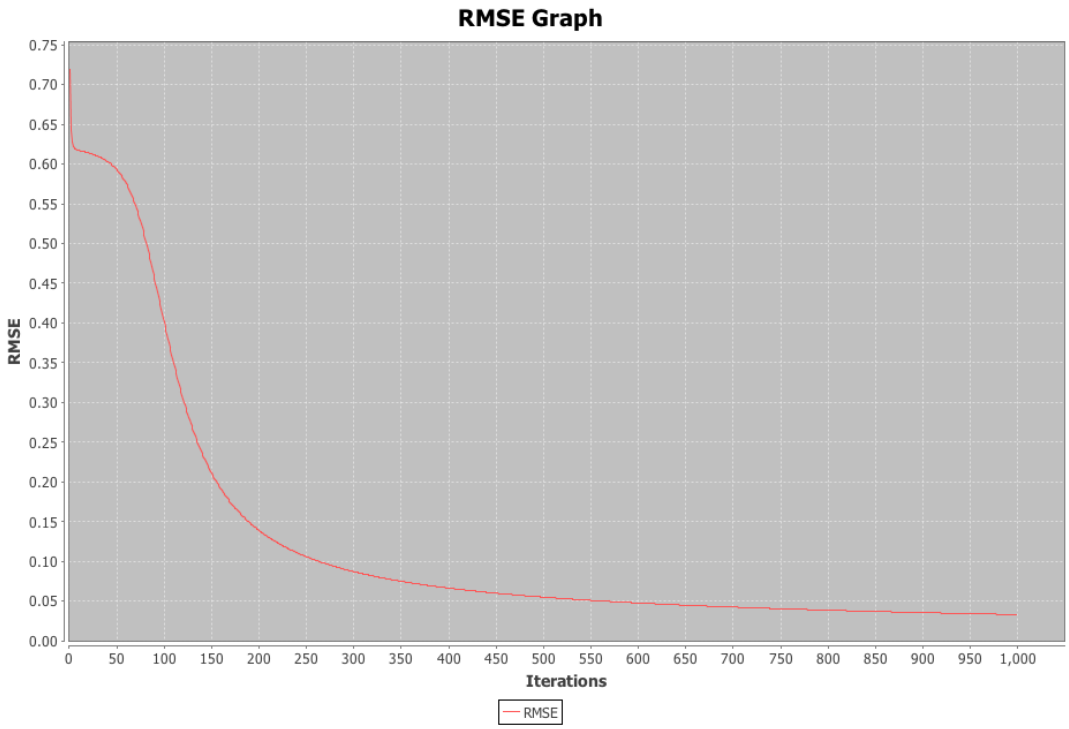


Figure 5.4: Training the General neural network for the first experimental network, as shown in Figure 5.1.

can now be conducted. The EDND and SDND test cases for each individual node are presented with an accompanying exemplary picture for clarity.

Ideally, when a router goes down the corresponding neural network output for that particular router should be as close to 1 as possible. All other routers experiencing no connection-loss issues (at least in addition to the connection lost to the router down), should be as close to 0 as possible to signify no problems. The first patterns to be assessed will be the SDND cases for each router–an exemplary visual of an SDND can be seen in Figure 5.5. In terms of the SDND cases shown in Table 5.1, the worst performing router case is with Router 5, whose neural network output is at 0.931%–6.9% from ideal. The best performer, on the other hand, is Router 3 whose neural network output is 0.994, only 0.6% error. The range is thus 0.6% to 6.9% in this test case. With regard to the routers that should not be affected, the hightest output was Router 4 with 0.0634 (6.34% error). As in the case of the EDND, this is also acceptable since Router 3 and Router 4 are the correlated case.

Router SDND	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.943	0.00382	0.00816	0.00371	0.0359
Router 2	0.0118	0.946	0.00143	0.00415	0.0789
Router 3	0.00564	0.00389	0.994	0.0634	0.00962
Router 4	0.0160	0.0131	0.0311	0.978	0.000609
Router 5	0.00348	0.00784	0.0181	0.00285	0.931

Table 5.1: Experiment 1 trained General neural network output for router SDND.

The next patterns to be assessed will be the EDND cases for each router–an

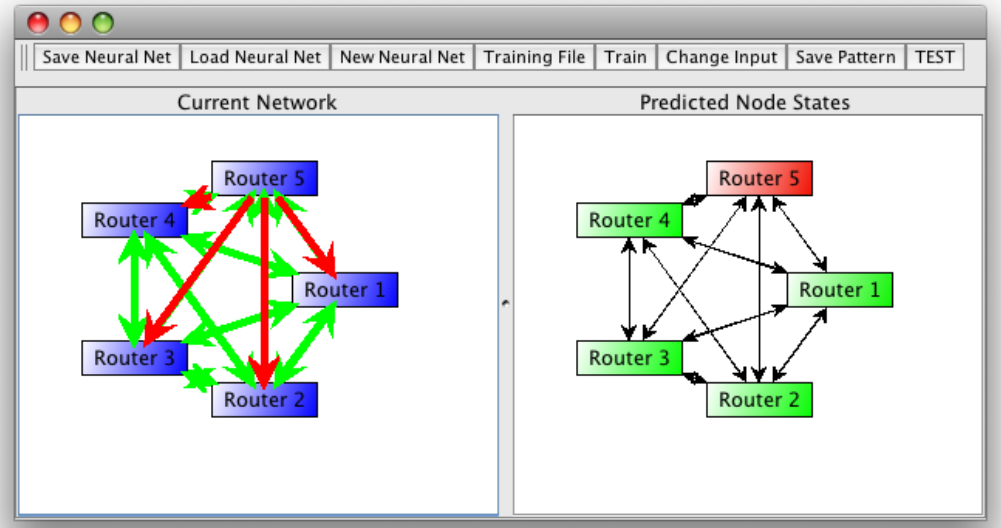


Figure 5.5: An example of SDND for Router 5.

exemplary visual of an EDND can be seen in Figure 5.6. In terms of the EDND cases shown in Table 5.2, the worst performing router case is with Router 1, whose neural network output is at 0.942–5.8% from the ideal. The best performer, on the other hand, is Router 4, whose neural network output is 0.987, only a 1.3% error. Thus, a range from 1.3% to 5.8% error is present within this test case. As for routers that should not be affected, the highest output was 0.249 (24.9% error) for Router 3 during the case of Router 4 going down. This is acceptable, however, since Router 3 and Router 4 are the correlated case.

The final patterns to be assessed are for routers going completely down–a visual of one such case is shown in Figure 5.7. For the case of routers completely down as shown in Table 5.3, the worst performing router case is Router 1 with neural network output of 0.994, 0.6% error. The best performing is Router 3 with 0.999, only 0.1% error. In terms of the routers that should not be affected, the highest output was for Router 3 at 0.173–17.3%. Mirroring both EDND and

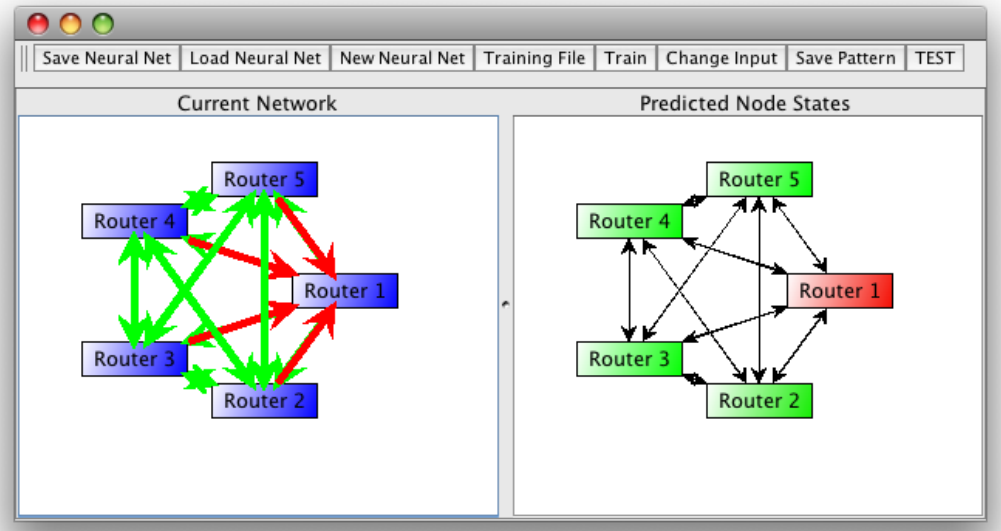


Figure 5.6: An example of EDND for Router 1.

Router EDND	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.942	0.00406	0.00723	0.00892	0.0411
Router 2	0.0273	0.954	0.00270	0.0106	0.0118
Router 3	0.00684	0.0147	0.953	0.00842	0.0159
Router 4	0.00490	0.00795	0.249	0.987	0.000865
Router 5	0.00643	0.00580	0.0142	0.00164	0.969

Table 5.2: Experiment 1 trained General neural network output for router EDND.

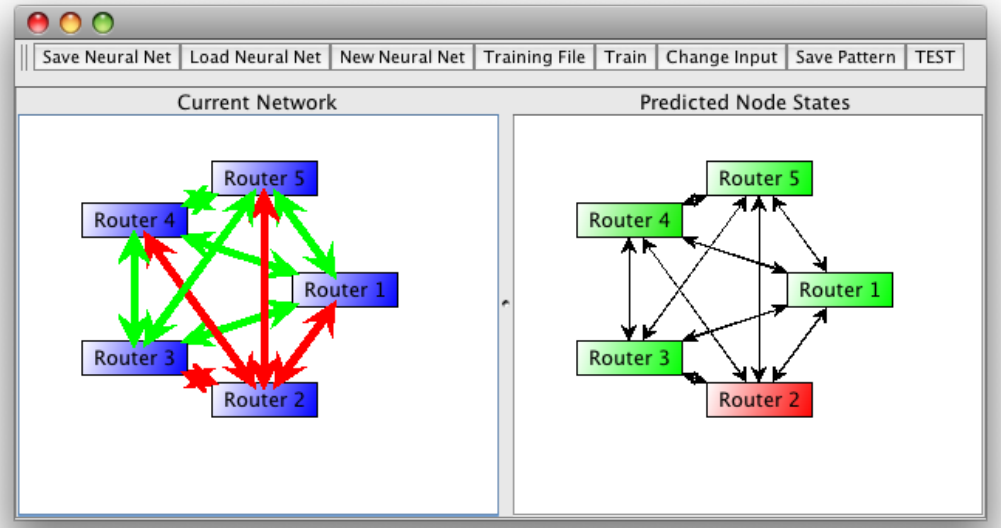


Figure 5.7: An example of Router 1 being completely down.

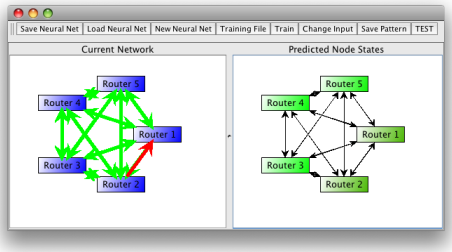
SDND, this is, once again, acceptable since it is the correlated case.

To show how the neural network smoothes out and dynamically determines which node is having issues based upon previously learned patterns, consider the sequence as shown in Figure 5.8 and Figure 5.9. And lastly, Figure 5.10 shows the capability of the neural network to make the correlated prediction.

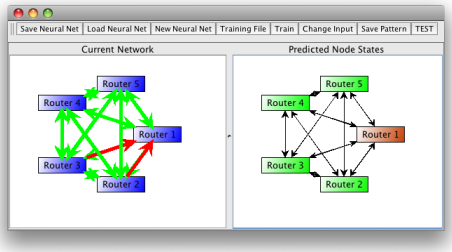
5.2.2 Expert Neural Network Implementation

The same training data has been utilized for the Expert neural networks. Each of the experts were trained for 1000 iterations with 0.2 momentum and 0.2 learning rate. The graphs of RMSE vs. iteration are shown in Figure 5.11.

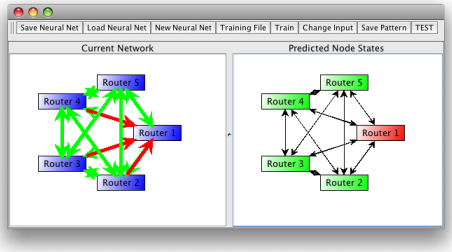
In terms of the SDND cases shown in Table 5.4, the worst performing router case is with Router 4, whose neural network output is at 0.981%–1.9% from ideal. The best performer, on the other hand, is a tie between Routers 1, 2, and



(a) First link down



(b) Second link down

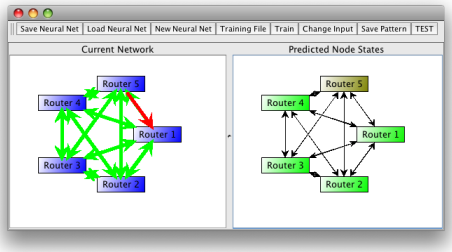


(c) Third link down

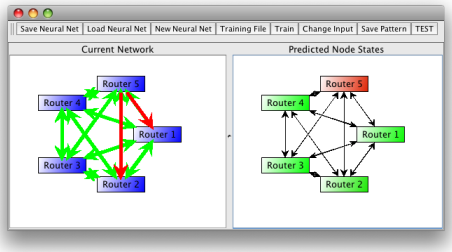
Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.285	0.288	0.00382	0.00186	0.0334
(b)	0.768	0.0363	0.0374	0.00126	0.0164
(c)	0.964	0.00692	0.0106	0.0192	0.00450

(d) Neural Network Outputs

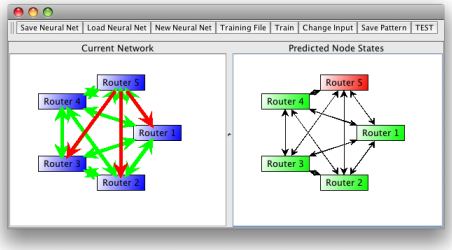
Figure 5.8: Experiment 1 example of EDND sequence and corresponding General neural network output starting with (a), then (b), and then (c).



(a) First link down



(b) Second link down

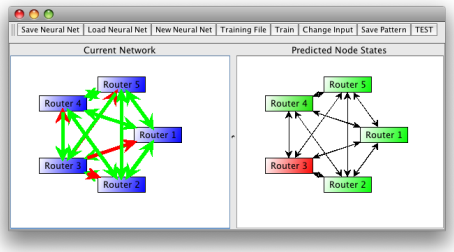


(c) Third link down

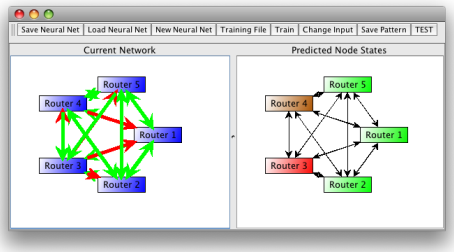
Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.0395	0.0109	0.00928	0.00576	0.489
(b)	0.0243	0.0921	0.00146	0.000635	0.876
(c)	0.00862	0.0219	0.0151	0.000322	0.925

(d) Neural Network Outputs

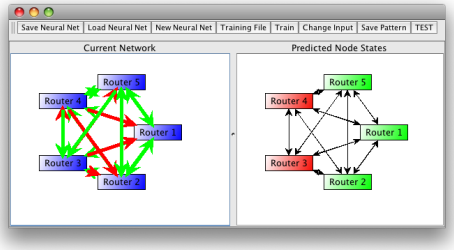
Figure 5.9: Experiment 1 example of SDND sequence and corresponding General neural network output starting with (a), then (b), and then (c).



(a) Router 3 SDND



(b) Router 4 lost one link

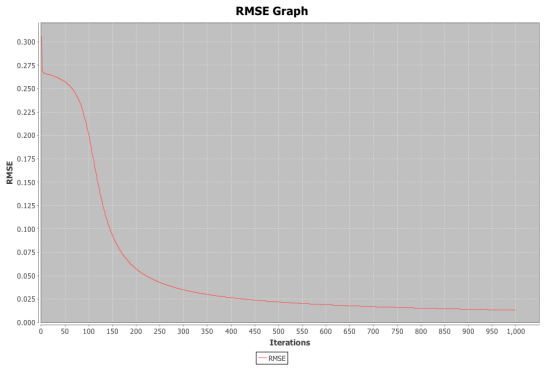


(c) Router 4 lost two links

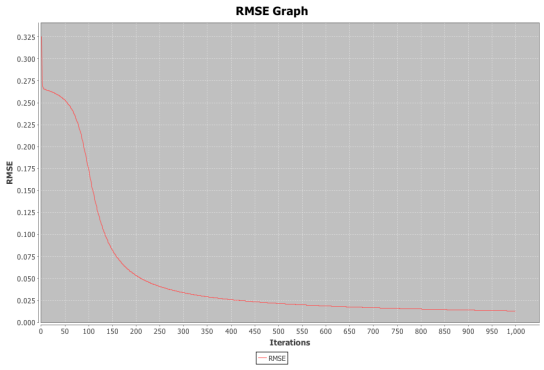
Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.00564	0.00389	0.994	0.0634	0.00962
(b)	0.0347	0.000559	0.984	0.674	0.00317
(c)	0.00836	0.00253	0.970	0.960	0.000641

(d) Neural Network Outputs

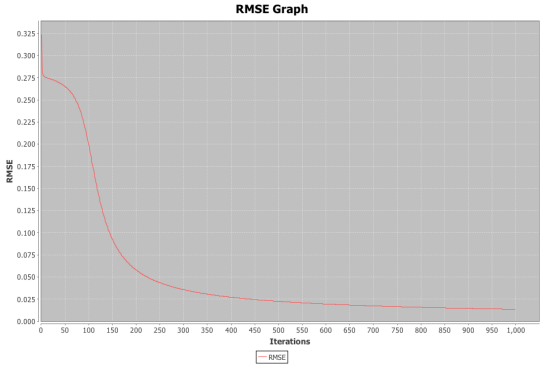
Figure 5.10: Experiment 1 example of correlated sequence and corresponding General neural network output starting with (a), then (b), and then (c).



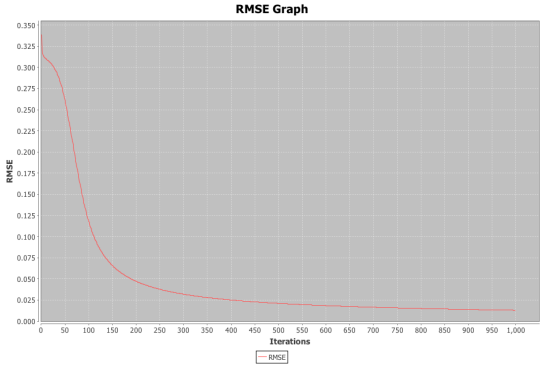
(a) Expert 1



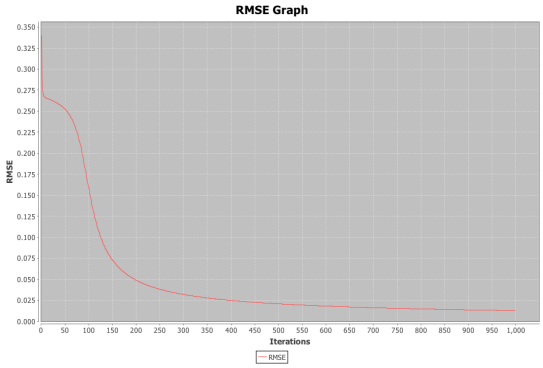
(b) Expert 2



(c) Expert 3



(d) Expert 4



(e) Expert 5

	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5
RMSE	0.0131	0.0130	0.0135	0.0139	0.0128
Time (seconds)	0.897	0.757	0.756	0.785	0.778

(f) Ending RMSE Values

Figure 5.11: Experiment 1 RMSE vs. iterations for each Expert neural network.

Router Completely Down	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.994	0.00259	0.00581	0.00477	0.0308
Router 2	0.0124	0.996	0.00108	0.00726	0.0218
Router 3	0.00580	0.00244	0.999	0.0209	0.00725
Router 4	0.00292	0.0117	0.173	0.998	0.000171
Router 5	0.00251	0.00337	0.0133	0.000495	0.997

Table 5.3: Experiment 1 trained General neural network output for router completely down.

5 whose neural network output are 0.997, only 0.3% error from ideal. The range is thus 0.3% to 1.9% in this test case. With regard to the routers that should not be affected, the hightest output was Router 5 with 0.115 (11.5% error). This particular error has no satisfactory explanation.

In the case of the EDND cases shown in Table 5.5, the worst performing router case is with Router 3, whose neural network output is at 0.955–4.5% from the ideal. The best performer, on the other hand, is Router 5, whose neural network output is 0.966, a 3.4% error. Thus, a range from 3.4% to 4.5% error is present within this test case. As for routers that should not be affected, the highest output was 0.067 (24.9% error) for Router 3 during the case of Router 4 going down. This is acceptable, however, since Router 3 and Router 4 are the correlated case.

Lastly, for the case of routers completely down as shown in Table 5.6, the worst performing router case is Router 1 with neural network output of 0.996, 0.4% error. The best performing is Router 4 with 0.999, only 0.1% error. Thus, the error range for this test is 0.1% to 0.4%. In terms of the routers that should

Router SDND	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.997	0.00230	0.00179	0.00295	0.0580
Router 2	0.00259	0.997	0.000539	0.00564	0.115
Router 3	0.0112	0.00636	0.992	0.0693	0.0917
Router 4	0.00940	0.0104	0.0129	0.981	0.00238
Router 5	0.000467	0.000458	0.00305	0.0190	0.997

Table 5.4: Experiment 1 trained Expert neural network output for router SDND.

Router EDND	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.963	0.00631	0.00379	0.00632	0.0606
Router 2	0.00630	0.956	0.00367	0.00816	0.0685
Router 3	0.00394	0.00599	0.955	0.0477	0.00888
Router 4	0.00520	0.00405	0.0637	0.965	0.00535
Router 5	0.00656	0.00616	0.00555	0.00440	0.966

Table 5.5: Experiment 1 trained Expert neural network output for router EDND.

Router Completely Down	Neural Network Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
Router 1	0.996	0.00301	0.00159	0.00413	0.0339
Router 2	0.00397	0.998	0.000314	0.00576	0.0755
Router 3	0.00213	0.00253	0.998	0.00243	0.0186
Router 4	0.00217	0.00238	0.0192	0.999	0.000346
Router 5	0.000545	0.000790	0.00319	0.0193	0.997

Table 5.6: Experiment 1 trained Expert neural network output for router completely down.

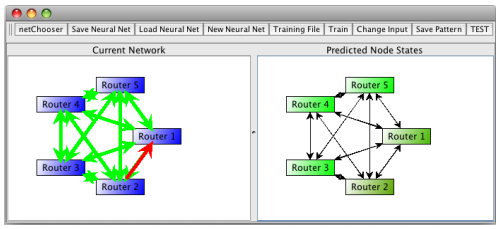
not be affected, the highest output was for Router 5 during the case of Router 2 being completely offline with neural network output of 0.0755 or 7.55%.

To show how the neural networks smooth out and dynamically predict and determine which node is having issues based upon previously learned patterns, consider the sequence as shown in Figure 5.12 and Figure 5.13. And lastly, Figure 5.14 shows the capability of the neural networks to make the correlated prediction.

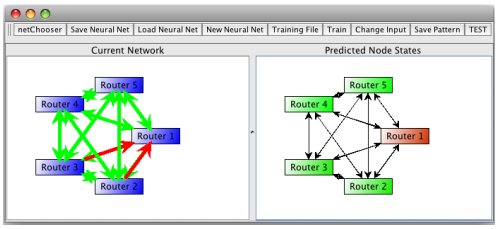
5.3 Experimental Network 2

This experiment will assess a large fully-connected BGP network, representative of the two smaller ISPs shown in Table 3.1. A visual of the fully-connected 40-node network topology can be seen in Figure 5.15.

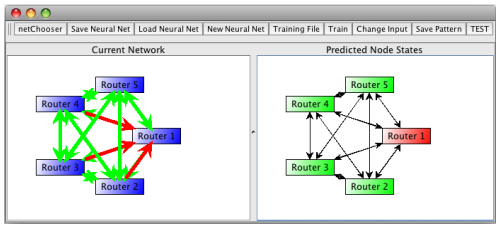
For this experiment, data was collected manually (offline from a physical BGP network) to simulate a 40-node network. So, even though this is contrived



(a) First link down



(b) Second link down

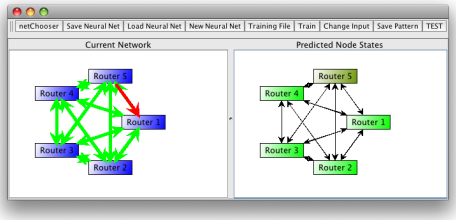


(c) Third link down

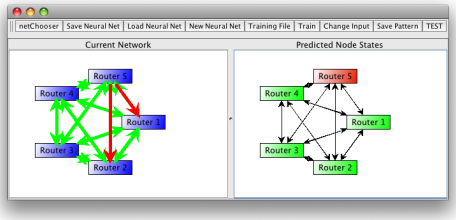
Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.278	0.351	0.00889	0.00148	0.0177
(b)	0.812	0.0730	0.0842	0.000620	0.00833
(c)	0.967	0.00952	0.0136	0.0124	0.00358

(d) Neural Network Outputs

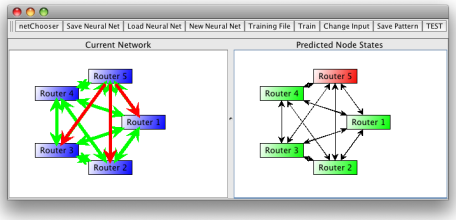
Figure 5.12: Experiment 1 example of EDND sequence and corresponding Expert neural network output starting with (a), then (b), and then (c).



(a) First link down



(b) Second link down

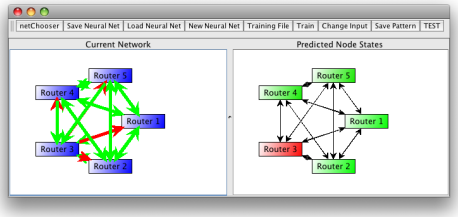


(c) Third link down

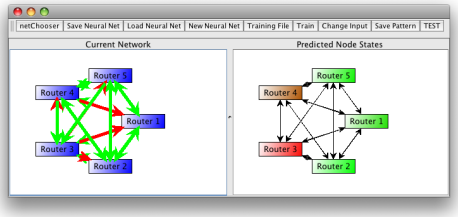
Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.0194	0.0159	0.00646	0.0100	0.417
(b)	0.0121	0.0128	0.00173	0.00547	0.902
(c)	0.00356	0.00440	0.0176	0.00354	0.967

(d) Neural Network Outputs

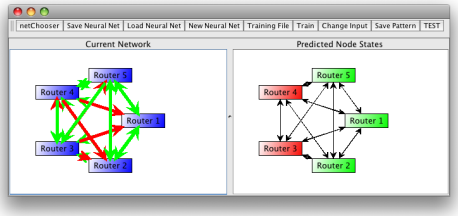
Figure 5.13: Experiment 1 example of SDND sequence and corresponding Expert neural network output starting with (a), then (b), and then (c).



(a) Router 3 SDND



(b) Router 4 lost one link



(c) Router 4 lost two links

Sequence	Neural Net Output by Router				
	Router 1	Router 2	Router 3	Router 4	Router 5
(a)	0.0112	0.00636	0.992	0.0693	0.0917
(b)	0.124	0.00100	0.986	0.677	0.0477
(c)	0.0187	0.00788	0.975	0.971	0.0205

(d) Neural Network Outputs

Figure 5.14: Experiment 1 example of correlated sequence and corresponding Expert neural network output starting with (a), then (b), and then (c).

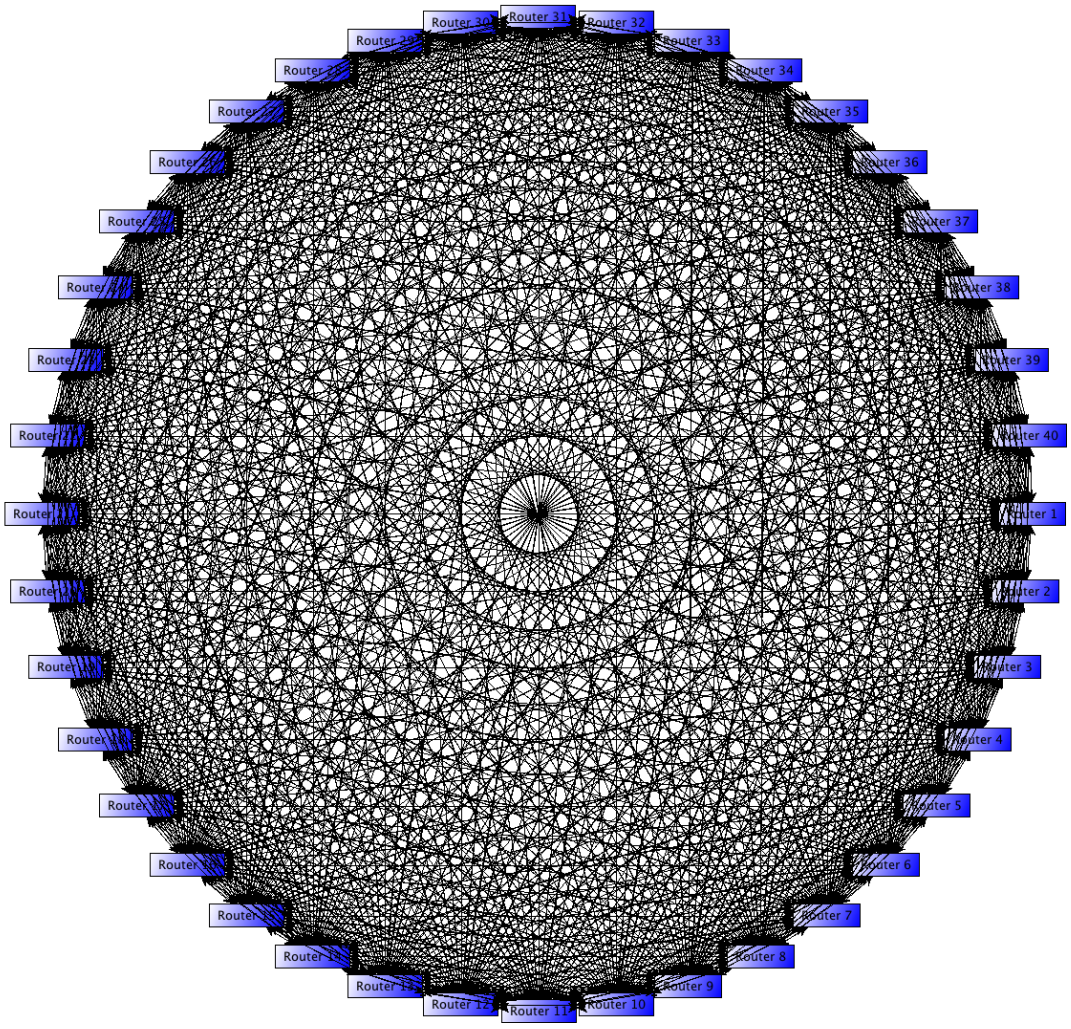


Figure 5.15: Fully connected 40-node BGP network for Experiment 2.

experimental data, care was taken to ensure that it is representative of patterns seen in lab within the smaller networks and previous experiment. Moreover, though this data may not be perfectly representative of that which would be collected in-lab, the point of this experiment is to prove the scalability of the neural network approach proposed in this thesis.

As in the first experiment, two patterns were saved per router–one representing an SDND and one of an EDND. Additionally, one correlated node-down pattern was saved, namely router 21 being a predictor for router 22. The `lookback` and `correlatedLookback` variables were set to 10 and 30 respectively.

5.3.1 General Neural Network

For the General neural network implementation, a total of 82 different patterns were collected for training purposes–an EDND and an SDND for each node in the network, one all-green pattern (all sessions established with all nodes up), and the correlated pattern. The neural network contains a hidden layer of 1560 nodes and an output layer of forty output nodes. The first training attempt was conducted for 1000 iterations with a training rate of 0.2 and momentum of 0.2, but the resulting RMSE was well over 0.364 or 36.4%, having taken 17,363,629 milliseconds (4.82 hours). A graph of the RMSE as compared with the training iterations can be seen in Figure 5.16. Clearly, this resulting error highlights the fact that the neural network was incapable of learning all of the desired patterns to an acceptable degree of accuracy.

Assessing the chaotic nature of the graph as shown in Figure 5.16, the learning rate and momentum coefficients seem to be set too high for such a massive error surface. Training was then retried on a new neural network with the same

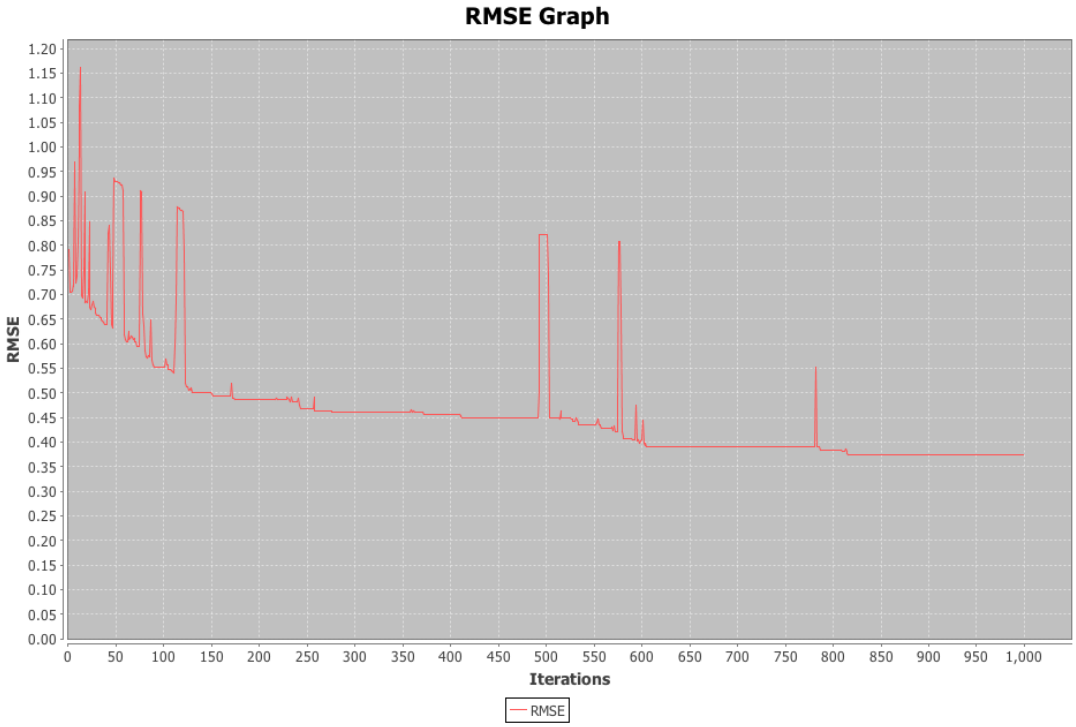


Figure 5.16: Failed training of the General neural network.

architecture for 1000 iterations but with a momentum coefficient and learning rate both set to 0.1, which had a resulting RMSE of 0.0112 or 1.12%, having taken 21,241,152 milliseconds (5.9 hours). A graph of the RMSE as compared with the training iterations can be seen in Figure 5.17. As shown by the resulting RMSE, this neural network was successful in learning the desired patterns and the resulting accuracy of this neural network can now be assessed.

In the case of the SDND cases shown in Table 5.7, the worst performing router case is with Router 30, whose neural network output is at 0.974–2.6% from the ideal. The best performer, on the other hand, is Router 21, whose neural network output is 0.996, only a 0.4% error. Thus, a range from 0.4% to 2.6% error is present within this test case for the routers of interest. As for routers that should not be affected, the highest output was 0.190 (19.0% error),

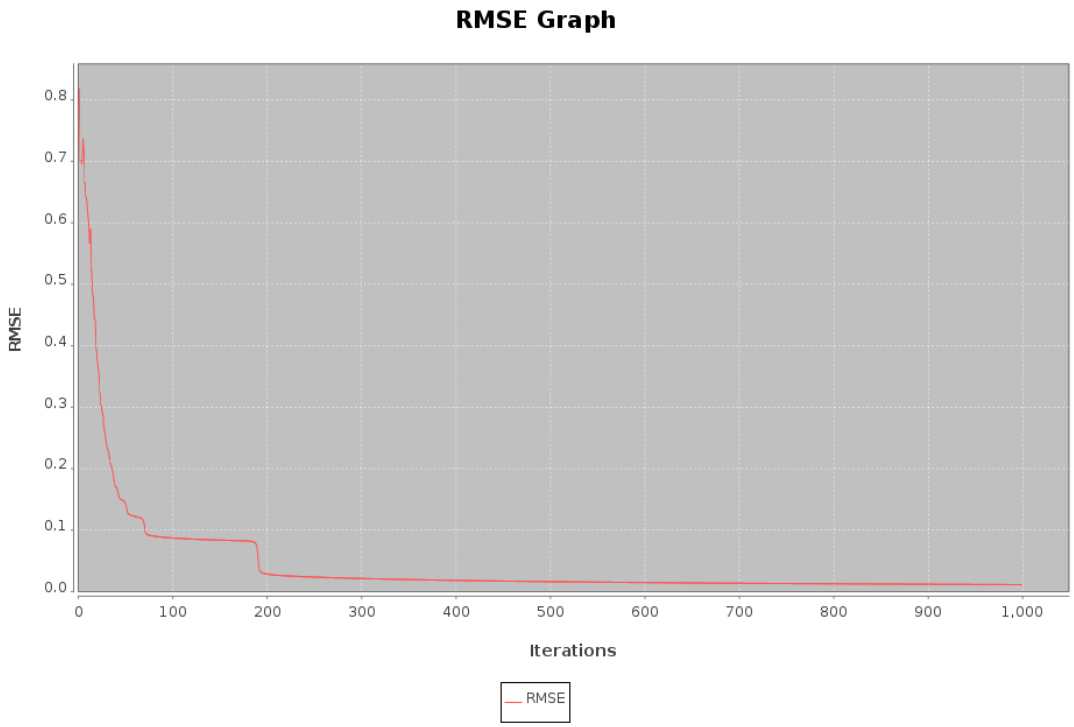


Figure 5.17: Succcessful training of the General neural network used for Experiment 2.

Router SDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.989	0.00317	0.000181
Router 2	0.977	0.00701	0.000232
Router 3	0.984	0.00478	0.000123
Router 4	0.990	0.00502	0.0000265
Router 5	0.977	0.00539	0.0000933
Router 6	0.987	0.00662	0.0000785
Router 7	0.982	0.00562	0.0000713
Router 8	0.987	0.00386	0.0000116
Router 9	0.990	0.00646	0.000145
Router 10	0.979	0.00599	0.0000233
Router 11	0.989	0.0104	0.0000268
Router 12	0.983	0.00828	0.0000453
Router 13	0.973	0.0105	0.0000905
Router 14	0.988	0.00395	0.0000216
Router 15	0.988	0.00470	0.0000994
Router 16	0.986	0.00553	0.000101
Router 17	0.990	0.00633	0.0000446
Router 18	0.978	0.00776	0.0000229
Router 19	0.983	0.00430	0.0000607
Router 20	0.988	0.00557	0.000154

Router SDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.996	0.190	0.0000117
Router 22	0.984	0.00474	0.000339
Router 23	0.977	0.00595	0.0000464
Router 24	0.987	0.00429	0.000177
Router 25	0.987	0.00651	0.000113
Router 26	0.987	0.00945	0.000147
Router 27	0.975	0.00586	0.000130
Router 28	0.976	0.00605	0.000100
Router 29	0.984	0.00482	0.000137
Router 30	0.974	0.00713	0.0000926
Router 31	0.988	0.00540	0.0000869
Router 32	0.985	0.00596	0.0000381
Router 33	0.983	0.00523	0.000142
Router 34	0.987	0.00468	0.000107
Router 35	0.985	0.00322	0.0000716
Router 36	0.986	0.00773	0.000200
Router 37	0.983	0.00586	0.000186
Router 38	0.984	0.00488	0.0000649
Router 39	0.976	0.00580	0.000100
Router 40	0.989	0.00426	0.0000696

Table 5.7: Experiment 2 trained General neural network output for router SDND.

which, incidentally, was the output for Router 22 during the case of Router 21 going down. This is acceptable, however, since Router 21 and Router 22 are the correlated case. The next highest, however, was 0.00945–only 0.945% from the ideal value.

In terms of the EDND cases shown in Table 5.8, the worst performing router case is with Router 32, whose neural network output is at 0.959–4.1% from the ideal. The best performer, on the other hand, is Router 21, whose neural network output is 0.992, only a 0.8% error. Thus, a range from 0.8% to 4.1% error is present within this test case for the routers of interest. As for routers that should not be affected, the highest output was 0.00884, only a 0.884% error.

Router EDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.981	0.00360	0.000300
Router 2	0.989	0.00396	0.000381
Router 3	0.979	0.00602	0.000151
Router 4	0.985	0.00498	0.0000746
Router 5	0.984	0.00380	0.0000304
Router 6	0.989	0.00884	0.000268
Router 7	0.983	0.00523	0.000231
Router 8	0.985	0.00500	0.0000235
Router 9	0.983	0.00605	0.000139
Router 10	0.985	0.00634	0.0000963
Router 11	0.979	0.00374	0.0000870
Router 12	0.974	0.00550	0.000441
Router 13	0.976	0.00451	0.000286
Router 14	0.989	0.00412	0.0000551
Router 15	0.968	0.00476	0.000269
Router 16	0.975	0.00446	0.000190
Router 17	0.984	0.00631	0.000188
Router 18	0.986	0.00624	0.0000868
Router 19	0.977	0.00453	0.000185
Router 20	0.967	0.00649	0.000209

Router EDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.992	0.00345	0.0000531
Router 22	0.990	0.00557	0.000485
Router 23	0.975	0.00361	0.0000910
Router 24	0.990	0.00602	0.000227
Router 25	0.987	0.00569	0.000191
Router 26	0.979	0.00599	0.000243
Router 27	0.986	0.00484	0.000169
Router 28	0.989	0.00383	0.000357
Router 29	0.987	0.00619	0.000215
Router 30	0.982	0.00407	0.000113
Router 31	0.984	0.00557	0.000104
Router 32	0.959	0.00447	0.0000601
Router 33	0.984	0.00485	0.000274
Router 34	0.989	0.00535	0.000167
Router 35	0.988	0.00596	0.000105
Router 36	0.978	0.00523	0.000162
Router 37	0.976	0.00363	0.000292
Router 38	0.973	0.00472	0.000273
Router 39	0.973	0.00504	0.000477
Router 40	0.986	0.00590	0.000114

Table 5.8: Experiment 2 trained General neural network output for router EDND.

Lastly, for the case of routers completely down as shown in Table 5.9, every single router of interest has a neural network output of 0.999–only 0.1% deviation from the ideal value. In terms of the routers that should not be affected, the highest output was for Router 22 during the case of Router 22 being completely offline with neural network output of 0.106 or 10.6%. Of course, this is acceptable because Routers 21 and 22 are the correlated case. The next highest error was during Router 26 which had a neural network output of 0.0186, 1.86% away from the ideal.

Router Completely Down	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.999	0.00486	0.0000672
Router 2	0.999	0.0146	0.000219
Router 3	0.999	0.00458	0.0000146
Router 4	0.999	0.0122	0.00000231
Router 5	0.999	0.00606	0.0000203
Router 6	0.999	0.0106	0.0000312
Router 7	0.999	0.00852	0.0000181
Router 8	0.999	0.00741	0.000000374
Router 9	0.999	0.00918	0.0000779
Router 10	0.999	0.00946	0.00000345
Router 11	0.999	0.00771	0.00000271
Router 12	0.999	0.0125	0.0000171
Router 13	0.999	0.00479	0.0000704
Router 14	0.999	0.00697	0.00000166
Router 15	0.999	0.00726	0.0000170
Router 16	0.999	0.00392	0.0000167
Router 17	0.999	0.0125	0.0000256
Router 18	0.999	0.00756	0.00000202
Router 19	0.999	0.00497	0.00000804
Router 20	0.999	0.00668	0.0000441

Router Completely Down	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.999	0.106	0.000000553
Router 22	0.999	0.00605	0.000160
Router 23	0.999	0.0108	0.0000185
Router 24	0.999	0.00545	0.000111
Router 25	0.999	0.00720	0.0000225
Router 26	0.999	0.0186	0.0000658
Router 27	0.999	0.00556	0.0000558
Router 28	0.999	0.00570	0.0000309
Router 29	0.999	0.00835	0.000110
Router 30	0.999	0.00995	0.00000901
Router 31	0.999	0.0110	0.0000105
Router 32	0.999	0.00854	0.00000213
Router 33	0.999	0.00718	0.0000593
Router 34	0.999	0.00741	0.0000177
Router 35	0.999	0.00468	0.00000762
Router 36	0.999	0.00919	0.0000225
Router 37	0.999	0.00847	0.000114
Router 38	0.999	0.00741	0.0000141
Router 39	0.999	0.00846	0.0000492
Router 40	0.999	0.00705	0.0000297

Table 5.9: Experiment 2 trained General neural network output for router completely down.

To show how the neural networks smooth out and dynamically predict and determine which node is having issues based upon previously learned patterns for this experiment, consider the sequence as shown in Table 5.10 and Table 5.11. And lastly, Table 5.12 shows the capability of the neural networks to make the

Links Down (Cumulative)	Neural Network Output by Router		
	Router 1	Next highest	Lowest
Routers 2, 3, 4, 5	0.00880	0.00297	0.000966
Routers 6, 7, 8, 9	0.0336	0.00479	0.000614
Router 10, 11, 12, 13	0.130	0.00397	0.000487
Router 14, 15, 16, 17	0.436	0.00365	0.000401
Router 18, 19, 20, 21	0.765	0.00441	0.000336
Router 22, 23, 24, 25	0.922	0.00432	0.000280
Router 26, 27, 28, 29	0.977	0.00358	0.000248

Table 5.10: Experiment 2 example sequential SDND for Router 1 using General neural network.

correlated prediction.

5.3.2 Expert Neural Network

The same training data from training the General neural network has been utilized for the Expert neural networks. Each of the experts were set to train for 1000 iterations with 0.1 momentum and 0.1 learning rate. However, each individual Expert learned tremendously faster than the single General neural network and, more interestingly, the RMSE value fell below the Desired RMSE of 0.009 under 100 iterations. A table comprised of all forty individual Expert training times and resulting RMSE values is shown in Table 5.13. Moreover, instead of showing all forty graphs, a couple exemplary graphs of training is shown in Figure 5.18.

In the case of the SDND cases shown in Table 5.14, the worst performing

Links Down (Cumulative)	Neural Network Output by Router		
	Router 2	Next highest	Lowest
Routers 3, 4, 5, 6	0.00991	0.00402	0.000876
Routers 7, 8, 9, 10	0.0407	0.00350	0.000768
Router 11, 12, 13, 14	0.176	0.00460	0.000635
Router 15, 16, 17, 18	0.453	0.00494	0.000735
Router 19, 20, 21, 22	0.727	0.00526	0.000613
Router 23, 24, 25, 26	0.883	0.00377	0.000564
Router 27, 28, 29, 30	0.973	0.00307	0.000457

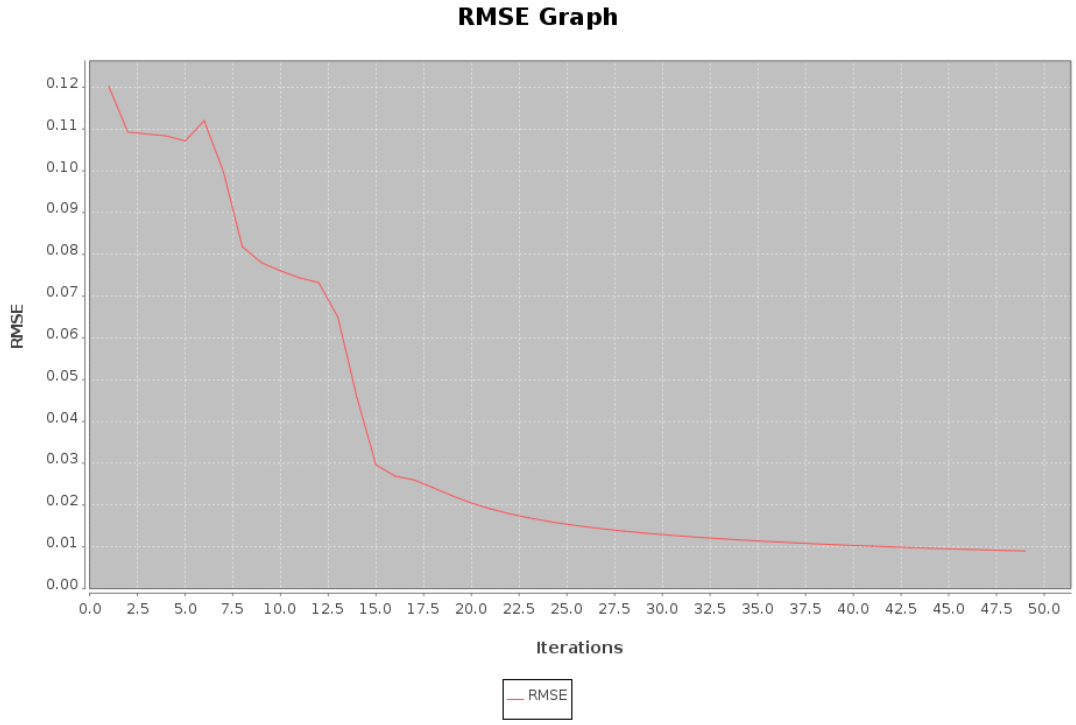
Table 5.11: Experiment 2 example Sequential EDND for Router 2 using General neural network.

Links Down (Cumulative)	Neural Network Output by Router			
	Router 21	Router 22	Next highest	Lowest
Router 21 SDND	0.996	0.190	0.00321	0.0000117
Router 22 loses Routers 1, 2	0.996	0.427	0.00346	0.0000115
Router 22 loses Routers 3, 4	0.996	0.705	0.00410	0.0000105
Router 22 loses Routers 5, 6	0.996	0.869	0.00375	0.00000869
Router 22 loses Routers 7, 8	0.996	0.957	0.00356	0.00000772
Router 22 loses Routers 9, 10	0.996	0.986	0.00375	0.00000778

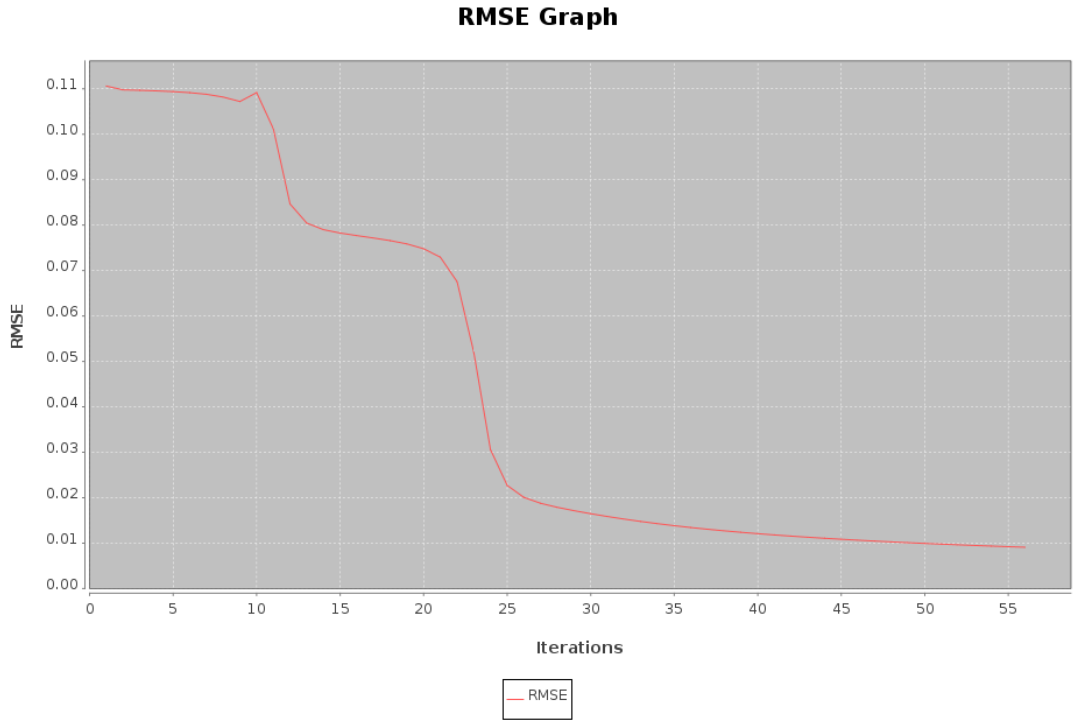
Table 5.12: Experiment 2 correlated example for Routers 21 and 22 using General neural network.

Expert	RMSE	Time (minutes)	Expert	RMSE	Time (minutes)
Expert 1	0.00887	17.03	Expert 21	0.00823	8.14
Expert 2	0.00878	16.48	Expert 22	0.00892	22.41
Expert 3	0.00885	18.58	Expert 23	0.00886	19.20
Expert 4	0.00886	23.7	Expert 24	0.00887	17.52
Expert 5	0.00888	17.09	Expert 25	0.00884	20.68
Expert 6	0.00884	19.86	Expert 26	0.00878	20.55
Expert 7	0.00880	21.31	Expert 27	0.00883	23.84
Expert 8	0.00886	18.76	Expert 28	0.00878	18.46
Expert 9	0.00888	15.80	Expert 29	0.00883	12.33
Expert 10	0.00877	21.69	Expert 30	0.00878	15.98
Expert 11	0.00886	16.45	Expert 31	0.00882	26.30
Expert 12	0.00884	13.15	Expert 32	0.00881	20.90
Expert 13	0.00885	15.95	Expert 33	0.00884	24.59
Expert 14	0.00888	23.12	Expert 34	0.00884	15.44
Expert 15	0.00877	14.23	Expert 35	0.00884	15.77
Expert 16	0.00885	17.76	Expert 36	0.00883	24.63
Expert 17	0.00878	16.36	Expert 37	0.00886	15.03
Expert 18	0.00877	17.63	Expert 38	0.00885	18.34
Expert 19	0.00883	21.83	Expert 39	0.00884	18.50
Expert 20	0.00884	20.60	Expert 40	0.0088	15.90

Table 5.13: Experiment 2 Expert training information.



(a) Expert 18 Training



(b) Expert 23 Training

Figure 5.18: Exemplary Graphs of Expert neural network training.

Router SDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.935	0.0265	0.00144
Router 2	0.936	0.0329	0.000138
Router 3	0.906	0.0251	0.000840
Router 4	0.948	0.0321	0.000167
Router 5	0.911	0.0253	0.000305
Router 6	0.902	0.0256	0.000591
Router 7	0.951	0.0240	0.000270
Router 8	0.932	0.0268	0.000491
Router 9	0.926	0.0237	0.000393
Router 10	0.964	0.0193	0.000139
Router 11	0.931	0.0301	0.000412
Router 12	0.954	0.0312	0.000134
Router 13	0.974	0.0213	0.000115
Router 14	0.905	0.0204	0.000486
Router 15	0.933	0.0359	0.000376
Router 16	0.957	0.0232	0.000332
Router 17	0.960	0.0639	0.000165
Router 18	0.944	0.0186	0.000342
Router 19	0.964	0.0200	0.000497
Router 20	0.926	0.0234	0.000281

Router SDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.997	0.270	0.000306
Router 22	0.943	0.0284	0.00002
Router 23	0.934	0.0191	0.000582
Router 24	0.919	0.0712	0.000352
Router 25	0.920	0.0198	0.000362
Router 26	0.951	0.0297	0.000368
Router 27	0.921	0.0256	0.000443
Router 28	0.963	0.0412	0.000338
Router 29	0.967	0.0254	0.000238
Router 30	0.965	0.0233	0.000547
Router 31	0.965	0.0224	0.000377
Router 32	0.960	0.0201	0.000650
Router 33	0.922	0.0252	0.000600
Router 34	0.964	0.0194	0.000431
Router 35	0.963	0.0182	0.000174
Router 36	0.936	0.0237	0.000575
Router 37	0.966	0.0158	0.000568
Router 38	0.928	0.0491	0.000568
Router 39	0.948	0.0419	0.000658
Router 40	0.940	0.0396	0.000410

Table 5.14: Experiment 2 Expert neural network output for router SDND.

router case is with Router 3, whose neural network output is at 0.906–9.4% from the ideal. The best performer, on the other hand, is Router 13, whose neural network output is 0.997, a 0.3% error. Thus, a range from 0.3% to 9.4% error is present within this test case for the routers of interest. As for routers that should not be affected, the highest output was 0.270 (27.0% error), which, mirroring the General neural network results, is the output for Router 22 during the case of Router 21 going down. This is acceptable, however, since Router 21 and Router 22 are the correlated case. The next highest, however, was 0.0639–6.39% from the ideal value.

As for the EDND cases shown in Table 5.15, the worst performing router case is with Router 12, whose neural network output is 0.868–13.3% from the ideal. The best performer, on the other hand, is Router 9, whose neural network output is 0.959, a 4.1% error. Thus, a range from 4.1% to 13.3% error is present within this test case for the routers of interest. As for routers that should not be affected, the highest output was 0.0515 (5.15% error), which occurs during Router 22 going offline.

Router EDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.879	0.0205	0.000566
Router 2	0.876	0.0248	0.000411
Router 3	0.930	0.0249	0.000789
Router 4	0.906	0.0252	0.000282
Router 5	0.869	0.0199	0.00202
Router 6	0.902	0.0309	0.00115
Router 7	0.924	0.0277	0.00187
Router 8	0.916	0.0214	0.00129
Router 9	0.959	0.0176	0.000790
Router 10	0.906	0.0293	0.000401
Router 11	0.928	0.0261	0.000452
Router 12	0.868	0.0169	0.000556
Router 13	0.953	0.0255	0.000199
Router 14	0.942	0.0317	0.000517
Router 15	0.902	0.0235	0.00141
Router 16	0.915	0.0217	0.00102
Router 17	0.906	0.0272	0.000706
Router 18	0.931	0.0311	0.000811
Router 19	0.890	0.0170	0.00155
Router 20	0.898	0.0494	0.000743

Router SDND	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.920	0.0384	0.00202
Router 22	0.953	0.0515	0.000470
Router 23	0.916	0.0255	0.00161
Router 24	0.930	0.0299	0.00108
Router 25	0.951	0.0182	0.000548
Router 26	0.929	0.0225	0.00204
Router 27	0.898	0.0428	0.00100
Router 28	0.937	0.0337	0.000901
Router 29	0.954	0.0354	0.000354
Router 30	0.930	0.0233	0.00110
Router 31	0.957	0.0186	0.00100
Router 32	0.929	0.0298	0.00130
Router 33	0.955	0.0138	0.00129
Router 34	0.953	0.0274	0.000896
Router 35	0.925	0.0186	0.000509
Router 36	0.901	0.0290	0.00152
Router 37	0.949	0.0286	0.00117
Router 38	0.917	0.0171	0.000664
Router 39	0.937	0.0134	0.000757
Router 40	0.917	0.0366	0.00112

Table 5.15: Experiment 2 Expert neural network output for router EDND.

As for the router completely down cases shown in Table 5.16, all routers have the exact same neural network output of 0.999–mirroring the General neural network. As for routers that should not be affected, the highest output was 0.181

Router Completely Down	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 1	0.999	0.0427	0.000576
Router 2	0.999	0.0204	0.0000123
Router 3	0.999	0.0268	0.0000477
Router 4	0.999	0.0499	0.0000146
Router 5	0.999	0.0366	0.000120
Router 6	0.999	0.0541	0.000321
Router 7	0.999	0.0307	0.000135
Router 8	0.999	0.0636	0.000305
Router 9	0.999	0.0409	0.000164
Router 10	0.999	0.0189	0.0000140
Router 11	0.999	0.0595	0.0000454
Router 12	0.999	0.0253	0.0000214
Router 13	0.999	0.0306	0.00000625
Router 14	0.999	0.0257	0.0000448
Router 15	0.999	0.0454	0.000145
Router 16	0.999	0.0425	0.0000655
Router 17	0.999	0.0798	0.0000276
Router 18	0.999	0.0223	0.0000557
Router 19	0.999	0.0342	0.000184
Router 20	0.999	0.0558	0.0000739

Router Completely Down	Neural Network Output by Router		
	Router of Interest	Next highest	Lowest
Router 21	0.999	0.181	0.000124
Router 22	0.999	0.0695	0.00000253
Router 23	0.999	0.0420	0.000293
Router 24	0.999	0.0566	0.000186
Router 25	0.999	0.0270	0.0000409
Router 26	0.999	0.0353	0.000111
Router 27	0.999	0.0350	0.000117
Router 28	0.999	0.0218	0.000103
Router 29	0.999	0.0344	0.0000184
Router 30	0.999	0.0313	0.000141
Router 31	0.999	0.0323	0.0000640
Router 32	0.999	0.0316	0.000175
Router 33	0.999	0.0442	0.000138
Router 34	0.999	0.0542	0.000192
Router 35	0.999	0.0205	0.0000186
Router 36	0.999	0.0323	0.000270
Router 37	0.999	0.0321	0.000101
Router 38	0.999	0.0486	0.000306
Router 39	0.999	0.0477	0.0000978
Router 40	0.999	0.0412	0.000224

Table 5.16: Experiment 2 Expert neural network output for router completely down.

(18.1% error), which occurs during Router 21 going offline. This output is once again from Router 22 and is therefore acceptable since this is the correlated case. The next highest, however, is 0.0798–7.98% error–during Router 17 going offline.

To show how the neural networks smooth out and dynamically predict and determine which node is having issues based upon previously learned patterns for this experiment, consider the sequence as shown in Table 5.17 and Table 5.18. And lastly, Table 5.19 shows the capability of the neural networks to make the correlated prediction.

Links Down (Cumulative)	Neural Network Output by Router		
	Router 1	Next highest	Lowest
Routers 2, 3, 4, 5	0.0246	0.0183	0.00397
Routers 6, 7, 8, 9	0.0719	0.0150	0.00381
Router 10, 11, 12, 13	0.212	0.0170	0.00304
Router 14, 15, 16, 17	0.389	0.0191	0.00297
Router 18, 19, 20, 21	0.713	0.0155	0.00240
Router 22, 23, 24, 25	0.870	0.0144	0.00135
Router 26, 27, 28, 29	0.923	0.0166	0.00122

Table 5.17: Experiment 2 example sequential SDND for Router 1 using Expert neural networks.

Links Down (Cumulative)	Neural Network Output by Router		
	Router 2	Next highest	Lowest
Routers 3, 4, 5, 6	0.0196	0.0126	0.00406
Routers 7, 8, 9, 10	0.0388	0.0142	0.00324
Router 11, 12, 13, 14	0.0979	0.0200	0.00279
Router 15, 16, 17, 18	0.208	0.0209	0.00219
Router 19, 20, 21, 22	0.411	0.0177	0.00189
Router 23, 24, 25, 26	0.669	0.0218	0.000944
Router 27, 28, 29, 30	0.878	0.0205	0.000620

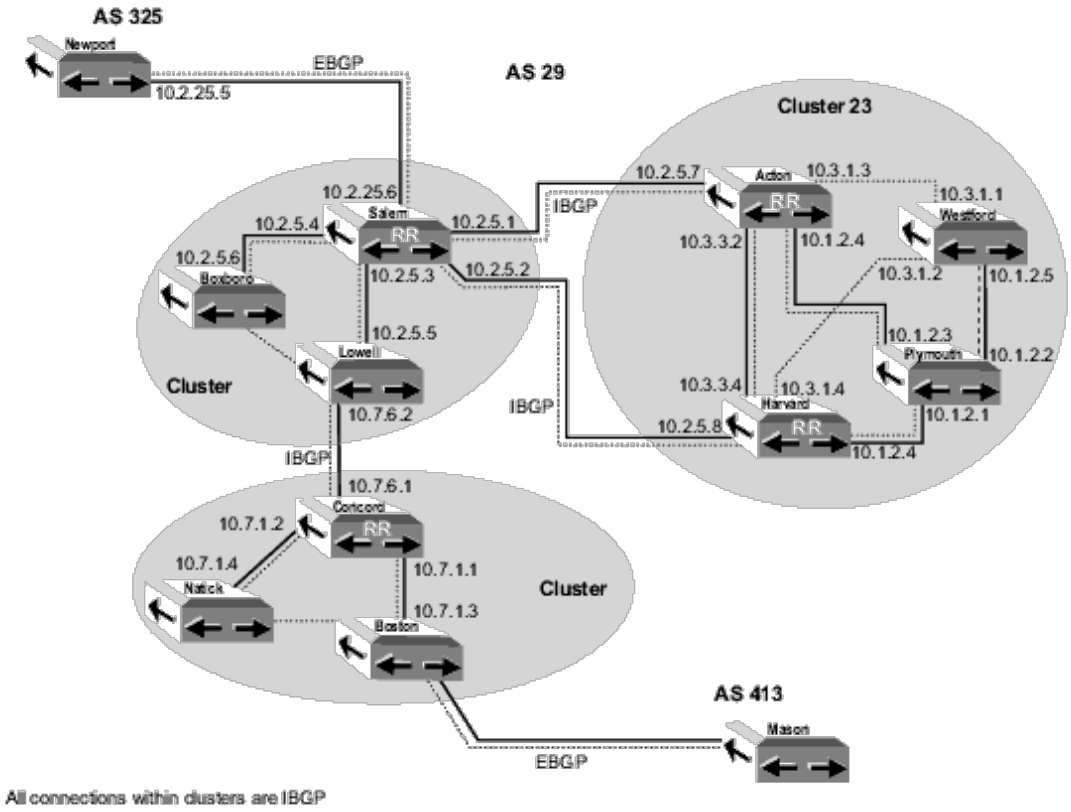
Table 5.18: Experiment 2 example sequential EDND for Router 2 using Expert neural networks.

Links Down (Cumulative)	Neural Network Output by Router			
	Router 21	Router 22	Next highest	Lowest
Router 21 SDND	0.997	0.270	0.0183	0.00832
Router 22 loses Routers 1, 2	0.966	0.408	0.0175	0.000285
Router 22 loses Routers 3, 4	0.996	0.587	0.0189	0.000235
Router 22 loses Routers 5, 6	0.993	0.826	0.0224	0.000155
Router 22 loses Routers 7, 8	0.993	0.904	0.0174	0.000143
Router 22 loses Routers 9, 10	0.992	0.955	0.0185	0.000160

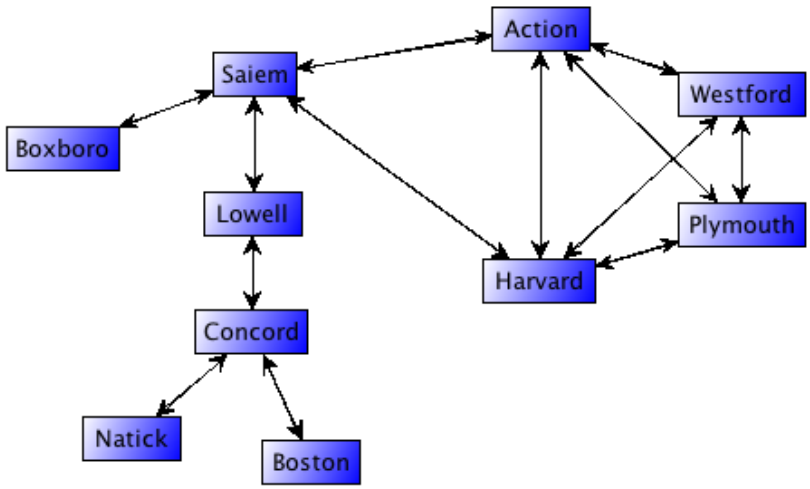
Table 5.19: Experiment 2 example sequential EDND for Routers 21 and 22.

5.4 Experimental Network 3

This experiment will assess a smaller route reflection BGP network. For this network, the topology was designed after an exemplary example from Juniper's website [38]. A visual of both the Juniper's topology as well as the comparable visual generated by the BGPNNF is shown in Figure 5.19. In the case of this network, the correlated case will be when Saiem goes offline, then Boxboro will go offline as well. Also, due to the fact that many routers in this topology only have one link of connection to the rest of the network, the `lookback` and `correlatedLookback` variables were both set to 1. Any other setting would allow for the case of an all-green pattern to be recorded as a pre-cursor pattern to router failures–for example, in the case of Boxboro, Natick, and Boston since only one link need go down for them to be considered down.



(a) Network as shown in [38]



(b) BGPNNF visual

Figure 5.19: Route reflection network topology used in experiment 3.

5.4.1 General Neural Network

For the General neural network implementation, a total of 22 different patterns were collected for training purposes–an EDND and an SDND for each node in the network, one all-green pattern (all sessions established with all nodes up), and the correlated pattern. The neural network contains a hidden layer of 26 nodes and was then trained for 1000 iterations with a momentum coefficient of 0.2 and learning rate 0.2 with a resulting RMSE of a whopping 0.122 or 12.2%, having taken 3,343 milliseconds. A graph of the RMSE as compared with the training iterations can be seen in Figure 5.20.

The reason for this high RMSE turns out to be a conflict of learning patterns in the neural network's training data. Specifically, when Saiem goes offline the corresponding pattern is memorized appropriately. However, when Boxboro goes offline immediately afterward and since Boxboro only requires a single link going down for it to be considered offline, the `correlatedLookback` variable that is equal to 1 triggers the same pattern to be memorized that had previously been memorized for Saiem. Thus, the training data now contains the same pattern twice, but with two different desired outputs: one desired output is to see Saiem offline whereas the other desired output is to see both Saiem and Boxboro offline.

There are two potential solutions to this problem. One solution would be to parse through the training data prior to training and remove any redundant training patterns–retaining only the pattern with the most expressive output. The second solution is to simply set the `correlatedLookback` variable to 0 as well. Since conflicting patterns are an intrinsic problem for machine learning, the former solution is utilized prior to training.

Now, with the training file cleansed of conflicting patterns the neural network

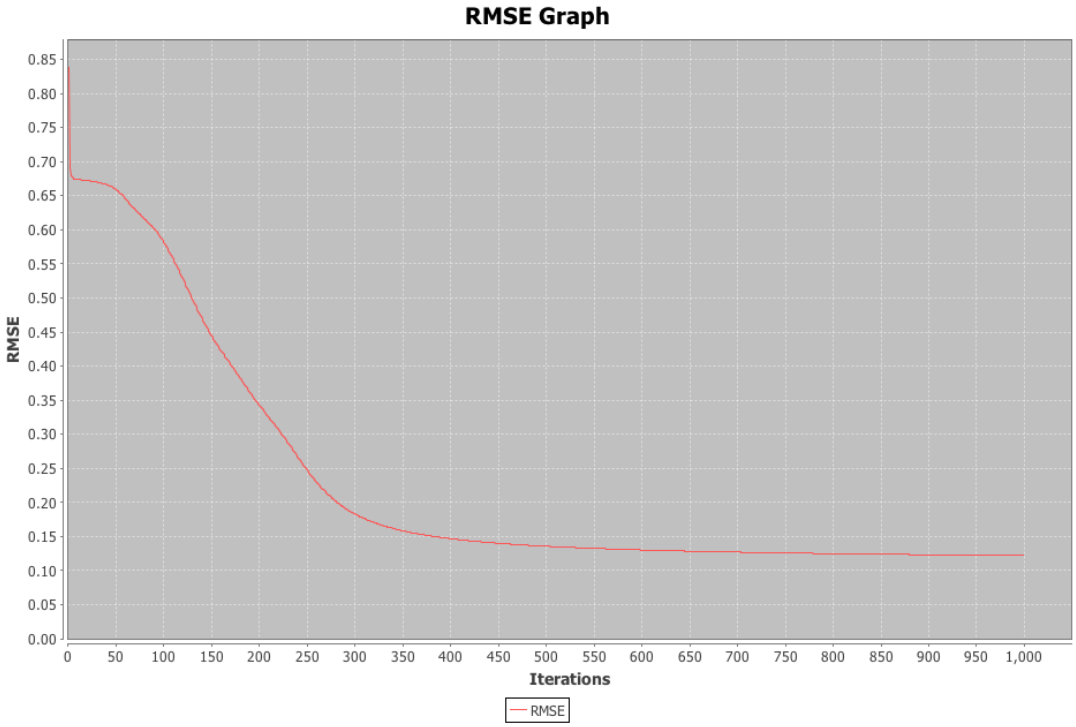


Figure 5.20: The General neural network RMSE training graph with conflicting training data present.

was trained once again. Training was performed again for 1000 iterations with a momentum coefficient of 0.2 and learning rate of 0.2 and a resulting RMSE of a much more appropriate 0.0396 or 3.96% in 3,656 milliseconds. The corresponding chart of training can be seen in Figure 5.21.

In the SDND cases as shown in Table 5.20, the best performers are Action and Plymouth, both with a corresponding neural network output of 0.970 or 3.0% from ideal. The worst routers are Natick and Boston both of which have a neural network output of 0.954 or 4.6% from ideal. Thus, the range of error for the case of SDND is between 3.0% and 4.6%. As for the routers that should not be affected, the highest output was 0.987–a whopping 98.7% error–which is the neural network output for Boxboro when Saiem goes offline. However, this is the correlated case and is therefore appropriate. The next highest is from

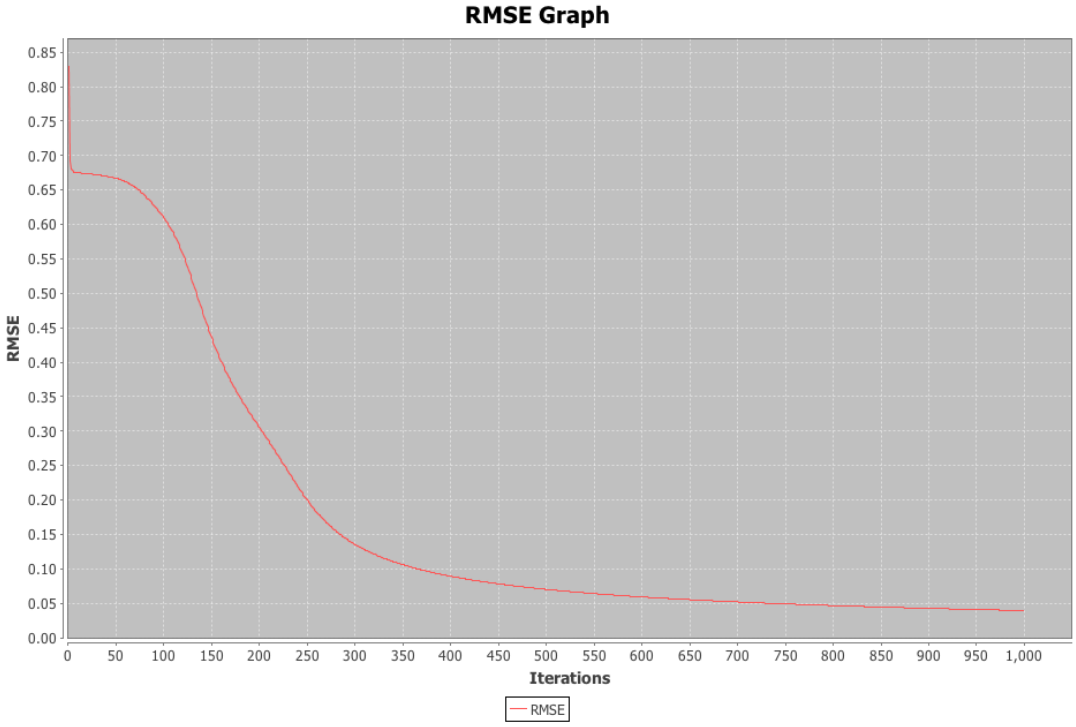


Figure 5.21: The General neural network RMSE training graph with conflicting training data removed.

Router SDND	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.970	0.0173	0.0125	0.0114	0.0104	0.000939	0.000396	0.0000601	0.0149	0.0108
Westford	0.0153	0.967	0.0170	0.0149	0.0000723	0.000736	0.0127	0.000649	0.00845	0.00920
Plymouth	0.0117	0.0175	0.970	0.0124	0.000284	0.000805	0.0108	0.00130	0.00813	0.00623
Harvard	0.0102	0.0135	0.00941	0.968	0.0126	0.000488	0.00154	0.000851	0.00351	0.00657
Saiem	0.0101	0.000727	0.00256	0.0174	0.963	0.987	0.0186	0.0000110	0.00529	0.00275
Boxboro	0.00539	0.00323	0.00252	0.00290	0.0329	0.949	0.00184	0.0000840	0.0233	0.0258
Lowell	0.000625	0.0113	0.00979	0.00300	0.0231	0.000655	0.959	0.0244	0.00178	0.000159
Concord	0.000736	0.00479	0.00809	0.00637	0.0000749	0.000226	0.0253	0.955	0.0313	0.0329
Natick	0.0130	0.00714	0.00752	0.00523	0.00105	0.0135	0.00255	0.0320	0.954	0.000407
Boston	0.00545	0.0118	0.00994	0.00798	0.000150	0.0184	0.000971	0.0309	0.000474	0.954

Table 5.20: Experiment 3 trained General neural network output for router SDND.

Boston, whose neural network output is 0.0329–3.29% from ideal–during the case of Concord going offline.

As for the EDND cases as shown in Table 5.21, the best performers are again Action and Plymouth, both with a corresponding neural network output of 0.969 or 3.1% from ideal. The worst routers are Concord and Boston, both of which have a neural network output of 0.953 or 4.7% from ideal. Thus the range of error for the case of SDND is between 3.1% and 4.7%. As for the routers that should not be affected, the highest output was 0.0343–a 3.43% error–which is the neural network output for Boxboro when Saiem goes offline. However, this again is the correlated case and is therefore appropriate. The next highest is from Lowell whose neural network output is 0.0332–3.32% from ideal–during the case of Saiem going offline.

Router EDND	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.969	0.0164	0.0139	0.0111	0.0136	0.00392	0.000371	0.0000374	0.0141	0.0146
Westford	0.0146	0.966	0.0163	0.0145	0.000117	0.00138	0.0119	0.000463	0.00842	0.0122
Plymouth	0.0123	0.0144	0.969	0.0129	0.000264	0.00209	0.0113	0.000936	0.00694	0.00794
Harvard	0.0100	0.0133	0.00919	0.968	0.0154	0.00433	0.00110	0.000535	0.00479	0.0104
Saiem	0.0210	0.000556	0.000611	0.0246	0.954	0.0343	0.0332	0.0000347	0.00363	0.000336
Boxboro	0.00586	0.00474	0.00474	0.00343	0.0291	0.961	0.00326	0.0000913	0.0230	0.0235
Lowell	0.000355	0.0111	0.0137	0.00327	0.0246	0.00338	0.962	0.0203	0.00175	0.000290
Concord	0.000810	0.00339	0.00824	0.00578	0.000116	0.000477	0.0262	0.953	0.0317	0.0327
Natick	0.0116	0.00847	0.00696	0.00518	0.00117	0.0152	0.00286	0.0291	0.954	0.000358
Boston	0.00520	0.0139	0.0102	0.00770	0.000118	0.0201	0.000873	0.0285	0.000535	0.953

Table 5.21: Experiment 3 trained General neural network output for router EDND.

As for the router completely down cases shown in Table 5.22, all routers have the exact same neural network output of 0.998, except for Plymouth which has an output of 0.999. Thus, the range of error is only 0.01 - 0.02%. As for routers that should not be affected, the highest output was 0.826 (82.6% error) from

Router Completely Down	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.998	0.0243	0.0105	0.0150	0.0248	0.000480	0.000197	0.0000251	0.0127	0.0147
Westford	0.0155	0.998	0.0154	0.0232	0.0000306	0.000186	0.0232	0.000379	0.00644	0.00536
Plymouth	0.00890	0.0189	0.999	0.0138	0.0000826	0.000180	0.0228	0.00279	0.00452	0.00535
Harvard	0.0102	0.0181	0.00733	0.998	0.0197	0.000527	0.00113	0.00102	0.00127	0.0118
Saiem	0.0151	0.000503	0.000346	0.0664	0.998	0.826	0.0596	0.00000789	0.00271	0.000435
Boxboro	0.00324	0.00182	0.00155	0.00325	0.159	0.998	0.00165	0.0000356	0.0202	0.0239
Lowell	0.000124	0.00683	0.00998	0.00345	0.0812	0.000557	0.998	0.0538	0.000659	0.0000245
Concord	0.000224	0.000908	0.00588	0.00647	0.0000624	0.0000814	0.0302	0.998	0.0282	0.0416
Natick	0.0130	0.00523	0.00565	0.00351	0.00130	0.00880	0.00256	0.199	0.998	0.0000267
Boston	0.00318	0.00813	0.0110	0.00858	0.0000328	0.0196	0.000254	0.108	0.0000949	0.998

Table 5.22: Experiment 3 trained General neural network output for router completely down.

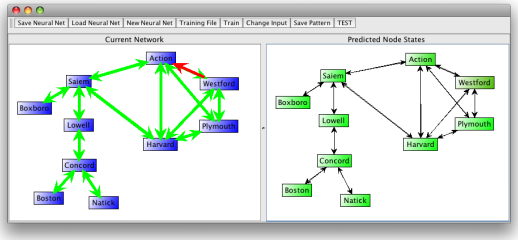
Boxboro and occurs during Saiem going offline, which is acceptable. The next highest, however, is 0.0812–8.12% error–from Saiem during Lowell going offline.

And finally, to show how the neural networks smooth out and dynamically predict and determine which node is having issues based upon previously learned patterns for this experiment, consider the sequence as shown in Figure 5.22 and Figure 5.23. And lastly, Figure 5.24 shows the capability of the neural networks to make the correlated prediction.

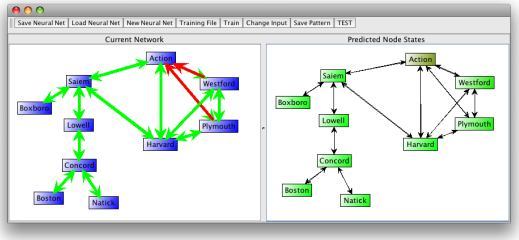
5.4.2 Expert Neural Network

For the Expert neural network implementation, each of the ten neural networks were trained for 1000 iterations with a 0.2 momentum coefficient and 0.2 learning rate. The final RMSE values, total training time, and a couple exemplary graphs can be seen in Figure 5.25.

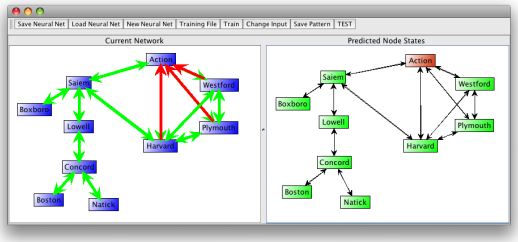
For the SDND cases shown in Table 5.23, the worst performing router is 0.952, whose neural network output is at 0.952–4.8% from the ideal. The best perform-



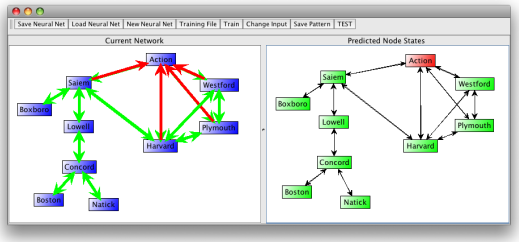
(a) First link down



(b) Second link down



(c) Third link down

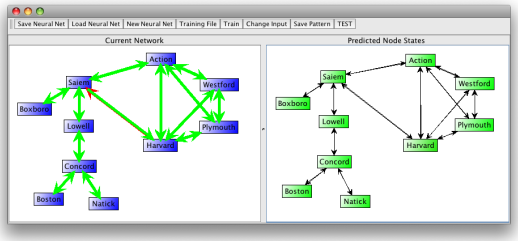


(d) Fourth link down

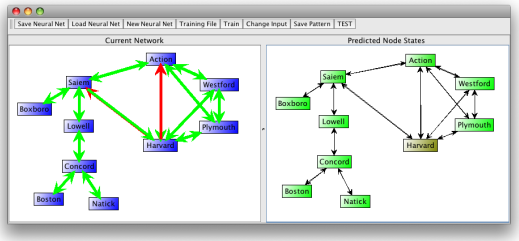
Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.103	0.268	0.00358	0.00273	0.000706	0.00863	0.00417	0.000379	0.0256	0.0180
(b)	0.427	0.0951	0.0502	0.00138	0.000472	0.00343	0.00276	0.000245	0.0281	0.0118
(c)	0.838	0.0473	0.0254	0.0126	0.00116	0.00134	0.000783	0.000133	0.0208	0.0117
(d)	0.969	0.0164	0.0139	0.0111	0.0136	0.00392	0.000371	0.0000374	0.0141	0.0146

(e) Neural Network Outputs

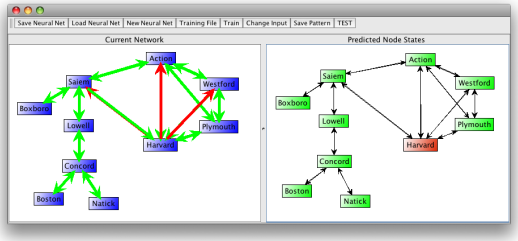
Figure 5.22: Example of EDND sequence and corresponding General neural network output starting with (a), then (b), then (c), and then (d).



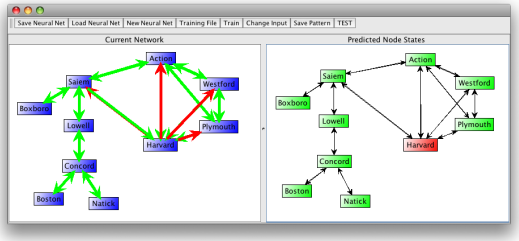
(a) First link down



(b) Second link down



(c) Third link down

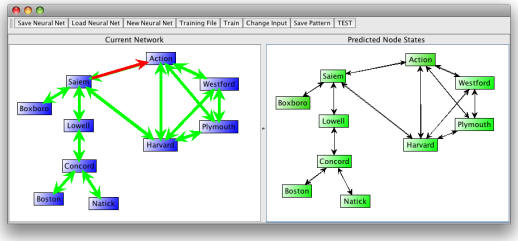


(d) Fourth link down

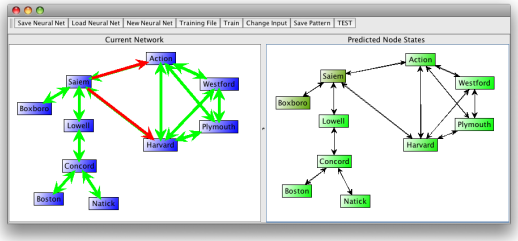
Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.00932	0.00477	0.00492	0.102	0.0266	0.00707	0.00294	0.000746	0.0217	0.00608
(b)	0.0665	0.00246	0.00232	0.481	0.0405	0.00305	0.000915	0.000459	0.0143	0.00697
(c)	0.0271	0.0287	0.00119	0.861	0.0157	0.00119	0.000849	0.000524	0.00783	0.00810
(d)	0.0102	0.0135	0.00941	0.968	0.0126	0.000488	0.00154	0.000851	0.00351	0.00657

(e) Neural Network Outputs

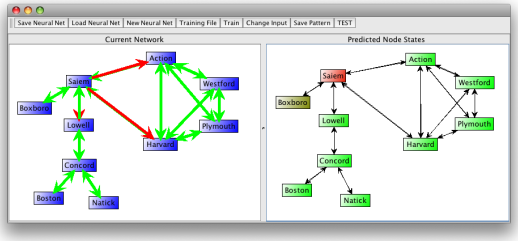
Figure 5.23: Example of SDND sequence and corresponding General neural network output starting with (a), then (b), then (c), and then (d).



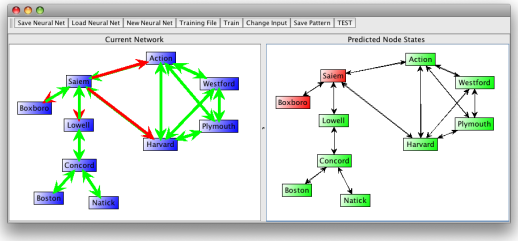
(a) First link down



(b) Second link down



(c) Third link down

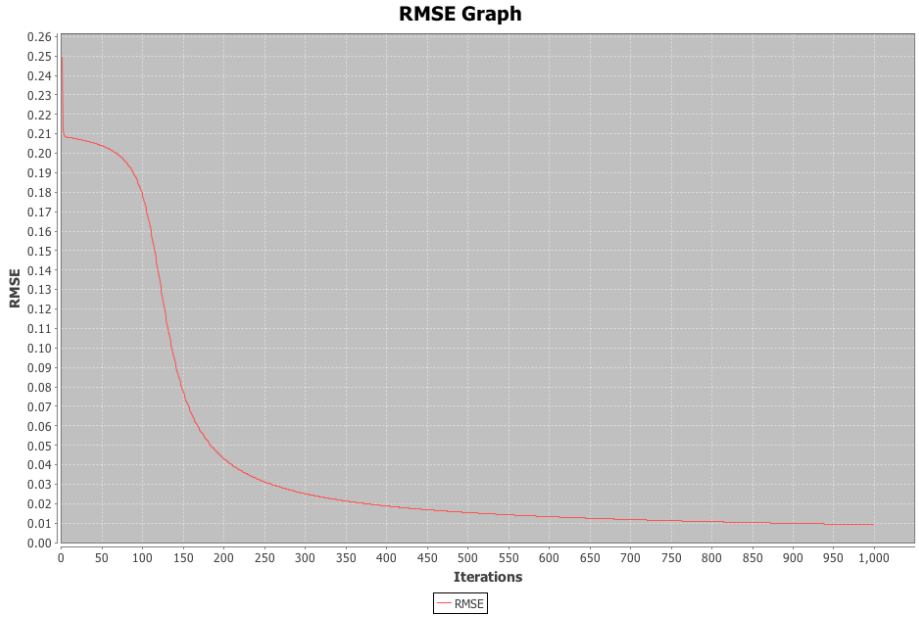


(d) Fourth link down

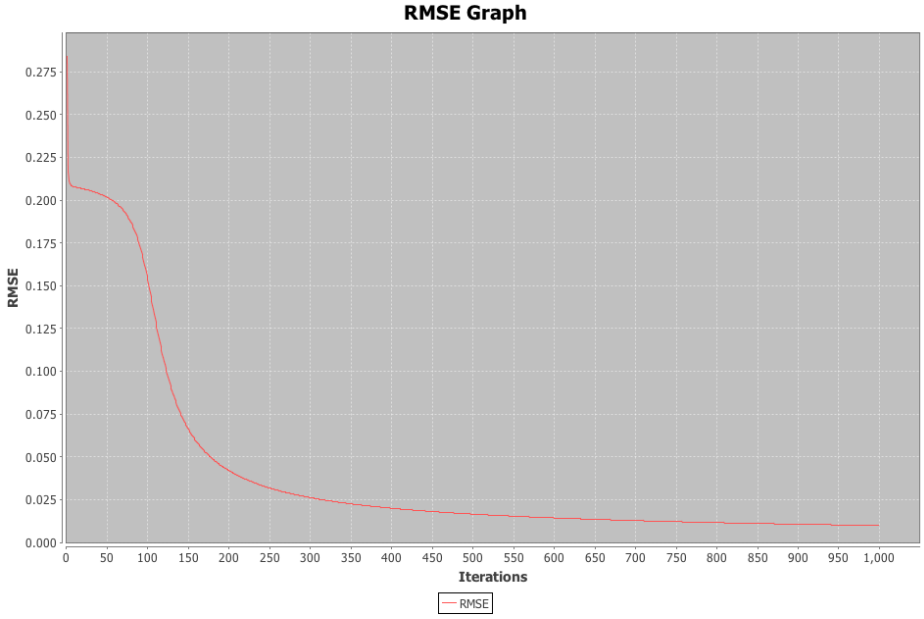
Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.131	0.00442	0.00607	0.00539	0.0324	0.158	0.00214	0.000157	0.0211	0.0247
(b)	0.0786	0.00131	0.00269	0.0637	0.373	0.352	0.00142	0.0000644	0.0188	0.00945
(c)	0.0216	0.00120	0.00280	0.0302	0.904	0.478	0.0518	0.0000345	0.00605	0.00154
(d)	0.0101	0.000727	0.00256	0.0174	0.963	0.987	0.0186	0.0000110	0.00529	0.00275

(e) Neural Network Outputs

Figure 5.24: Example of correlated sequence and corresponding General neural network output starting with (a), then (b), then (c), and then (d).



(a) Expert Westford



(b) Expert Saiem

	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
RMSE	0.00898	0.00920	0.00912	0.00899	0.00996	0.0120	0.0108	0.0122	0.0142	0.0145
Time (milliseconds)	2,441	2,110	2,508	1,573	1,800	1,758	1,739	1,724	1,800	1,810

(c) Ending RMSE Values

Figure 5.25: Experiment 3 RMSE vs. iterations for each Expert neural network.

Router SDND	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.970	0.0118	0.0106	0.0100	0.00886	0.000737	0.00223	0.000409	0.00214	0.00203
Westford	0.0111	0.969	0.0144	0.0133	0.000650	0.00216	0.00674	0.000482	0.00379	0.00362
Plymouth	0.0138	0.0138	0.969	0.0133	0.00133	0.00321	0.00588	0.000578	0.00413	0.00367
Harvard	0.00842	0.0108	0.00949	0.970	0.00941	0.000670	0.00122	0.000212	0.00314	0.00221
Saiem	0.0134	0.00159	0.00147	0.0126	0.967	0.994	0.0174	0.000176	0.00149	0.00131
Boxboro	0.00461	0.00886	0.00812	0.00422	0.0228	0.956	0.00534	0.000942	0.0112	0.0125
Lowell	0.00441	0.00452	0.00699	0.00348	0.0156	0.00150	0.964	0.0176	0.00116	0.000966
Concord	0.00365	0.00379	0.00329	0.00348	0.000947	0.000653	0.0201	0.959	0.0260	0.0253
Natick	0.00717	0.0107	0.00757	0.00693	0.00214	0.0118	0.00394	0.0253	0.952	0.00194
Boston	0.00913	0.00678	0.00862	0.00761	0.00239	0.0113	0.00344	0.0258	0.00112	0.952

Table 5.23: Experiment 3 trained Expert neural network output for router SDND.

ers, on the other hand, are routers Action and Harvard which both have a neural network output 0.970, a 3.0% error. Thus, a range from 3.0% to 4.8% error is present within this test case for the routers of interest. As for routers that should not be affected, the highest output was 0.994–a whopping 99.4% error–mirroring the General neural network results since this is, once again, the correlated case and is therefore acceptable. The next highest, however, was 0.0260–2.60% from the ideal value–which was the neural network output for Natick during the case of Concord going down.

As for the EDND cases as shown in Table 5.24, the best performers are again Action, Plymouth, and Harvard, all three with a corresponding neural network output of 0.970 or 3.0% from ideal. The worst router is Natick which has a neural network output of 0.951 or 4.9% from ideal. Thus the range of error for the case of SDND is between 3.0% and 4.9%. As for the routers that should not be affected, the highest output was 0.0258–a 2.58% error–which is the neural network output for Natick during Concord going offline.

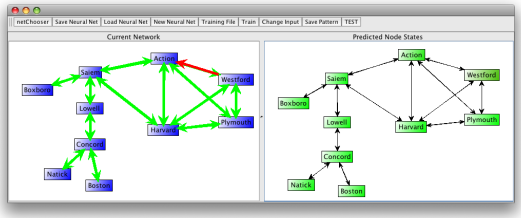
Router EDND	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.970	0.0122	0.0114	0.0101	0.00927	0.00642	0.00163	0.000373	0.00189	0.00177
Westford	0.0129	0.969	0.0159	0.0132	0.000826	0.00579	0.00502	0.000352	0.00386	0.00460
Plymouth	0.0124	0.0137	0.970	0.0134	0.000801	0.00516	0.00515	0.000513	0.00475	0.00382
Harvard	0.00865	0.0108	0.0117	0.970	0.00908	0.00663	0.00208	0.000384	0.00301	0.00183
Saiem	0.0124	0.00148	0.000936	0.0114	0.967	0.0202	0.0180	0.000172	0.00188	0.00138
Boxboro	0.00484	0.00778	0.00836	0.00502	0.0244	0.965	0.00586	0.000817	0.0121	0.0112
Lowell	0.00509	0.00696	0.00624	0.00324	0.0162	0.00985	0.964	0.0170	0.000806	0.00112
Concord	0.00350	0.00284	0.00379	0.00376	0.000702	0.00139	0.0203	0.958	0.0258	0.0253
Natick	0.00702	0.0104	0.00825	0.00835	0.00265	0.0107	0.00342	0.0249	0.951	0.00198
Boston	0.00837	0.00719	0.00796	0.00583	0.00292	0.0109	0.00393	0.0257	0.00123	0.952

Table 5.24: Experiment 3 trained Expert neural network output for router EDND.

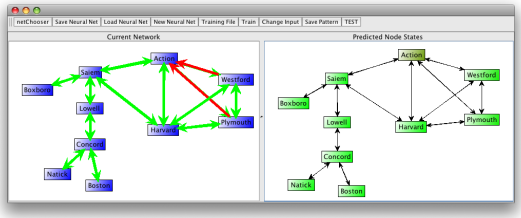
Router Completely Down	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
Action	0.998	0.0104	0.00870	0.0101	0.0224	0.000197	0.000230	0.000159	0.000178	0.000115
Westford	0.0133	0.998	0.0162	0.0178	0.000174	0.000409	0.00168	0.000166	0.000511	0.000421
Plymouth	0.0155	0.0137	0.999	0.0182	0.000350	0.000537	0.00134	0.000287	0.000660	0.000341
Harvard	0.00679	0.00854	0.00795	0.998	0.0242	0.000182	0.000168	0.0000933	0.000351	0.000127
Saiem	0.0151	0.000262	0.000152	0.0146	0.999	0.989	0.0134	0.0000409	0.000126	0.0000636
Boxboro	0.00209	0.00500	0.00490	0.00220	0.142	0.999	0.00143	0.000707	0.00390	0.00282
Lowell	0.00215	0.00233	0.00322	0.00123	0.0684	0.000449	0.999	0.230	0.0000538	0.0000399
Concord	0.00129	0.000890	0.00102	0.00139	0.000226	0.0000560	0.0172	0.999	0.0188	0.0126
Natick	0.00460	0.00802	0.00449	0.00589	0.00163	0.00315	0.000625	0.390	0.999	0.000112
Boston	0.00699	0.00357	0.00492	0.00453	0.00203	0.00315	0.000633	0.402	0.0000699	0.999

Table 5.25: Experiment 3 trained Expert neural network output for router completely down.

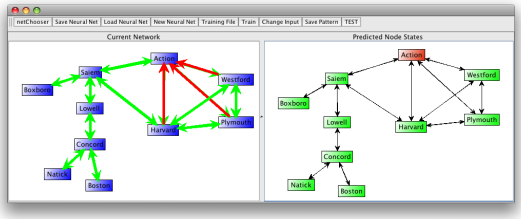
As for the router completely down cases shown in Table 5.25, all routers have the exact same neural network output of 0.999, except for Action, Westford, and Harvard which have an output of 0.998. Thus, the range of error is only 0.01 - 0.02%. As for routers that should not be affected, the highest output was 0.989 (98.9% error) from Boxboro and occurs during Saiem going offline, which is acceptable. The next highest, however, is 0.402–40.2% error–from Concord



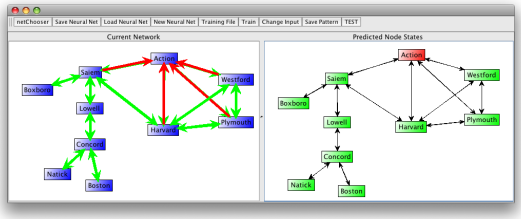
(a) First link down



(b) Second link down



(c) Third link down



(d) Fourth link down

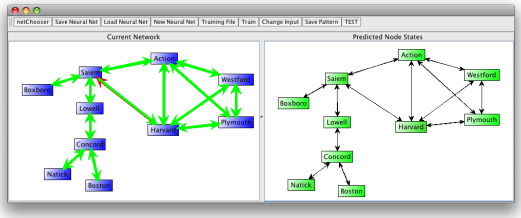
Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.0813	0.259	0.00294	0.00470	0.00190	0.0155	0.0159	0.000859	0.0156	0.0209
(b)	0.404	0.0715	0.0595	0.00205	0.00131	0.00587	0.0124	0.000576	0.00737	0.00820
(c)	0.864	0.0239	0.0189	0.0163	0.000732	0.00258	0.00649	0.000390	0.00380	0.00390
(d)	0.970	0.0122	0.0114	0.0101	0.00927	0.00642	0.00163	0.000373	0.00189	0.00177

(e) Neural Network Outputs

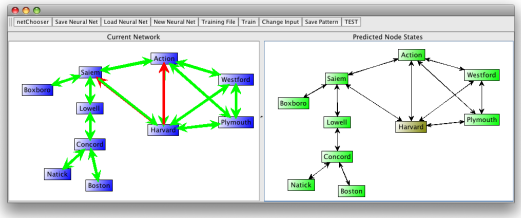
Figure 5.26: Example of EDND sequence and corresponding Expert neural network output starting with (a), then (b), then (c), and then (d).

when Boston goes offline. This large error highlights some inherent sensitivity of Expert neural networks when interfaced with sparse topologies.

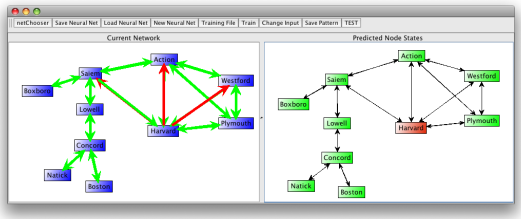
And finally, to show how the neural networks smooth out and dynamically predict and determine which node is having issues based upon previously learned patterns for this experiment, consider the sequence as shown in Figure 5.26 and Figure 5.27. And lastly, Figure 5.28 shows the capability of the neural networks to make the correlated prediction.



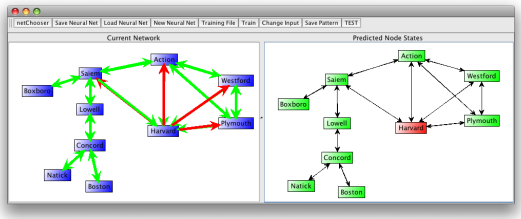
(a) First link down



(b) Second link down



(c) Third link down

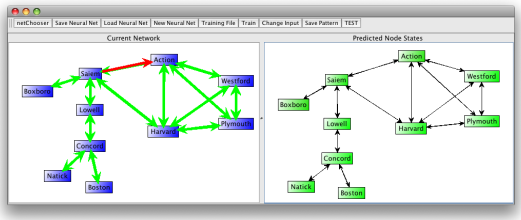


(d) Fourth link down

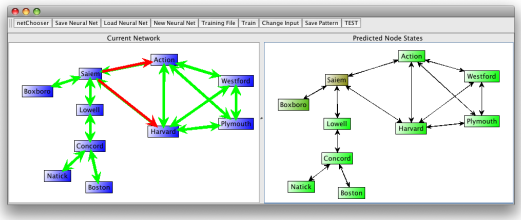
Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.00515	0.00688	0.00500	0.107	0.0698	0.00437	0.00530	0.000782	0.0175	0.0207
(b)	0.0601	0.00245	0.00168	0.487	0.0351	0.00191	0.00282	0.000540	0.00873	0.00968
(c)	0.0219	0.0399	0.000527	0.871	0.0171	0.00109	0.00178	0.000306	0.00470	0.00421
(d)	0.00842	0.0108	0.00949	0.970	0.00941	0.000670	0.00122	0.000212	0.00314	0.00221

(e) Neural Network Outputs

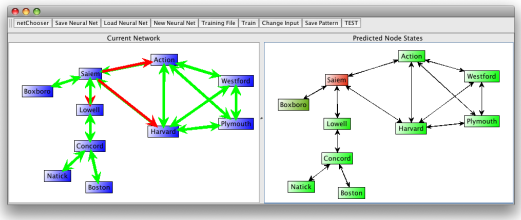
Figure 5.27: Example of SDND sequence and corresponding Expert neural network output starting with (a), then (b), then (c), and then (d).



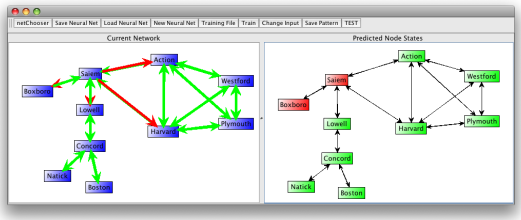
(a) First link down



(b) Second link down



(c) Third link down



(d) Fourth link down

Sequence	Neural Network Output by Router									
	Action	Westford	Plymouth	Harvard	Saiem	Boxboro	Lowell	Concord	Natick	Boston
(a)	0.102	0.00711	0.00849	0.00612	0.0534	0.111	0.00545	0.00104	0.0163	0.0217
(b)	0.0443	0.00360	0.00350	0.0533	0.491	0.287	0.00156	0.000748	0.00864	0.00870
(c)	0.0304	0.00268	0.00230	0.0246	0.885	0.349	0.0693	0.000227	0.00395	0.00561
(d)	0.0134	0.00159	0.00147	0.0126	0.967	0.994	0.0174	0.000176	0.00149	0.00131

(e) Neural Network Outputs

Figure 5.28: Example of correlated sequence and corresponding Expert neural network output starting with (a), then (b), then (c), and then (d).

Chapter 6

Results and Conclusions

Overall, for each experiment performed, both the General and Expert neural networks were capable of memorizing and utilizing the pre-cursor connectivity patterns within an iBGP network prior to a node failure. Moreover, the accuracy of the General neural network architecture surpassed expectations and resulted in comparable performance to the Experts in every experiment.

During experiment 1, for equal training parameters and iterations, the Expert neural networks out-performed the General within a few percent of accuracy for each testing case. Moreover, an individual Expert trained more quickly than the General–requiring approximately 67% of the time needed for training the General. However, since the Expert neural network strategy requires one neural network per router, the cumulative total results in about 300% more time than training the single General unless multiple Experts are trained simultaneously on a multi-core machine. The real tradeoff for slightly better accuracy comes at the cost of more memory–again resulting in the fact that the Expert strategy requires (N - 1) more neural networks than the General, where N is equal to the number of routers in a given network topology. This additional memory requirement is

fairly negligible with very small neural networks such as in experiment 1, but can become a serious issue with much larger neural networks.

Experiment 2 offers a different perspective from experiment 1. Here, a very well-trained General neural network out-performed the Expert neural networks in every test case in terms of accuracy. Admittedly, this is due to the Expert neural networks hitting the "Desired RMSE" value early on during training, and more training would have increased their accuracy. This highlights a key point: even though every single Expert neural network had a much lower RMSE value (all were below 0.009) than the single General (0.0112), the General still had greater overall accuracy. This is a result of the fact that the Experts have many patterns memorized that require their output of 0 and only a few patterns that require an output of 1. In the case of the General, on the other hand, nearly every pattern memorized required at least one of the output nodes to yield a 1. This difference explains the discrepancy between the RMSE and comparative performance of the two approaches.

Continuing with experiment 2, even though accuracy is comparable between the two neural network architectures, there are much greater utilization trade-offs. Overall, the scalability of the approach proposed in this thesis is definitely strained when interfaced with the large 40-node fully-connected iBGP network. This can be seen through the resulting required training time for the General neural network (5.9 hours), and even for the Experts (finishing on an average of approximately 20 minutes–a considerable amount of time when required for 40 individual networks). Of greater worry, however, is the memory requirements. The General neural network implementation can run with a 512MB maximum allocation of Java heap space. However, the Expert neural network strategy requires 3,000MB of memory. Moreover, a single serialized and saved Expert neural net-

work requires over 43MB of memory and the General requires over 44MB. These facts highlight the limitations of this approach for larger networks (containing greater than 1560 sessions of interest).

Experiment 3 mirrors many of the findings from experiment 1, but also proves the capability of this approach for sparse network topologies. Both detection and correlated prediction work very well for the comparatively sparse route-reflection network, but the prediction capability is greatly reduced, if not entirely impossible, for some nodes. Essentially the prediction mechanism of this approach fails for any router with only a single connection to the surrounding network when no correlated failure is in effect. Put simply, even though this approach works for a sparse network topology, the more links available directly influences the prediction ability of router failure. Along the lines of scalability, the network topology in experiment 3 only contained 26 unique sessions and experiment 2 proved that this approach can work with up to 1560 sessions, which implies that this approach can work with much larger networks–not fully-connected but preferably not extremely sparse either. Lastly, for additional details regarding the required training time of the two proposed neural network architectures in terms of a specific desired RMSE value, see Appendix A.

All in all, the hypothesis of this thesis–namely that neural networks can be utilized to memorize the preconditioned patterns that emerge prior to a node failure–has been proven through the experiments performed.

Chapter 7

Future Work

This section will discuss the various potential future works that can extend the findings of this thesis. Future work is appropriate for extending some of the BGPNNF framework itself, considering various other machine learning techniques, and considering networks other than iBGP.

7.1 Extending the BGPNNF

One unexplored neural network architectural strategy worthy of consideration would be a hybrid mixture of the disjoint Expert and General neural network strategies presented and implemented in this thesis. In such a scenario, a hybrid neural network implementation could use a combination of Expert and General neural networks for different subsets of a given network topology. For example, referring back to the network used in experiment 3 (Figure 5.19), a single neural network could be responsible for Boxboro, Natick, and Boston (all routers with only a single connection to the rest of the network), whereas Expert neural networks could be responsible for the remainder of nodes in the topology. Such an

approach would allow for additional flexibility in utilizing the expressiveness of nodes with greater connections versus nodes with very few connections, especially in terms of memory requirements for the neural networks.

Another interesting aspect worthy of additional research would be to find optimal momentum coefficients, learning rates, and desired RMSE values for this problem domain. Since the goal of this thesis is to prove the capability of neural networks to memorize network connectivity patterns and utilize them for detection and prediction purposes, no serious attempts were made in finding the best values for these variables–the numbers utilized in this work simply worked, but may not be universally optimal.

7.2 Other Machine Learning Techniques

This thesis focused entirely upon traditional feedforward backpropagation neural networks at the machine learning mechanism. However, a vast number of alternative machine learning techniques could also be employed and assessed for applicability in this problem domain. Just to name a few popular techniques, Logistic Regression [36], Classification and Regression Trees [13], Bayesian Additive Regression Trees [20], Support Vector Machines[15], and Random Forests [12] could be utilized as an alternative to neural networks. Also, one recent promising machine learning theory that is an implementation on a specific theory on how the neocortex functions is Hierarchical Temporal Memory, as introduced by Jeff Hawkins [35]. Furthermore, one great option would be to conduct a comparative survey of machine learning techniques for this domain, similar to the comparative analysis of machine learning techniques used for phishing detection as conducted in [1].

7.3 Other Networks

The approach presented in this thesis is designed to be generic to the extent that it is not necessarily tied to iBGP networks. Essentially any peer-to-peer network whose connection sessions can be represented in a finite state machine that describes the current connection status can be interfaced with this approach. With this fact in mind, this approach can be extended far beyond other computer network protocols to encapsulate entirely different types of networks.

To illuminate a few potential applications, a first would be various forms of wireless networks. Sparse power efficient topology for wireless networks as discussed in [47] has potentially fitting network topologies as shown in Figure 7.1. Moreover, potentially any of the topologies discussed in a survey of wireless mesh network [2] could be explored. The approach proposed in this thesis is not restricted to only router connectivity–so long as a client's connection state is available, client-server or wireless mesh clients could also be considered nodes in a network topology such as shown in Figure 7.2a. Furthermore, wireless mesh networks that reside within transportation systems Figure 7.2b, communities Figure 7.3a, metropolitan areas, Figure 7.3b, enterprises Figure 7.4a, and in automated buildings Figure 7.4b are domains for which this approach is applicable. The methods presented in this thesis could be applied to any of these domains for the purposes of node failure detection and prediction.

A fairly different network domain would be usage within power grids. There are various types of power networks as well such as on-chip power grid networks [46] with topologies as shown in Figure 7.5 where nodes could be considered as the individual resistors and the input to the neural network could be current, voltage, or power. Though more of a stretch from traditional computer networks, there

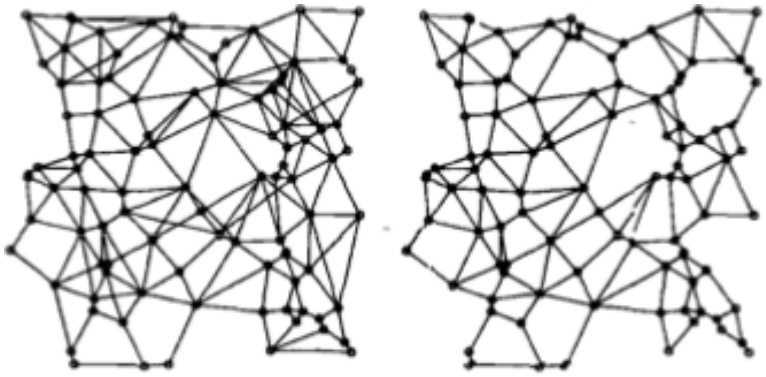
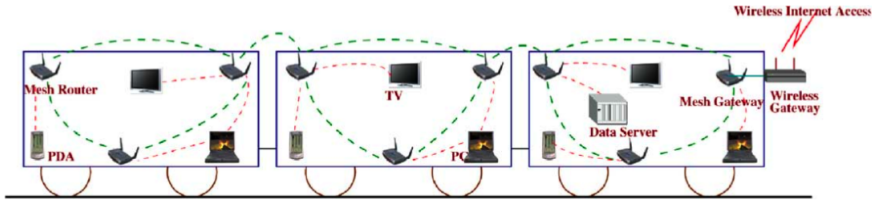


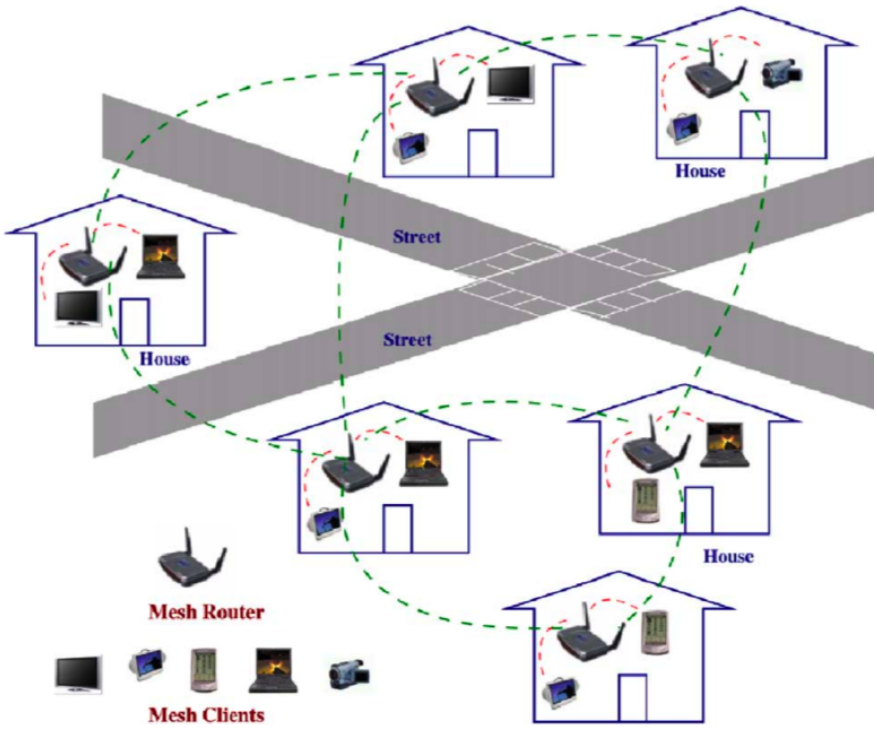
Figure 7.1: Two exemplary network topologies as shown in [47].

(a) Wireless Mesh Networks for Building Automation

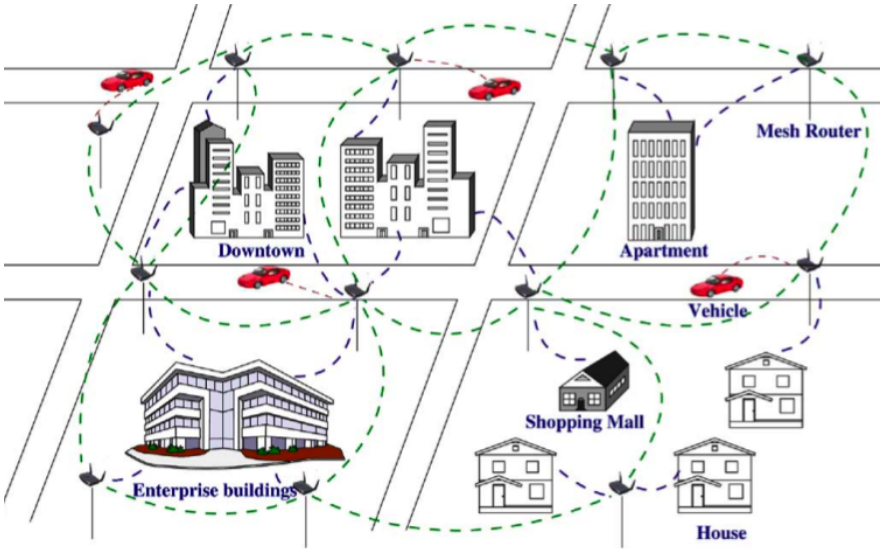


(b) Wireless Mesh Networks for transportation Networks

Figure 7.2: More exemplary network topologies as shown in [2].

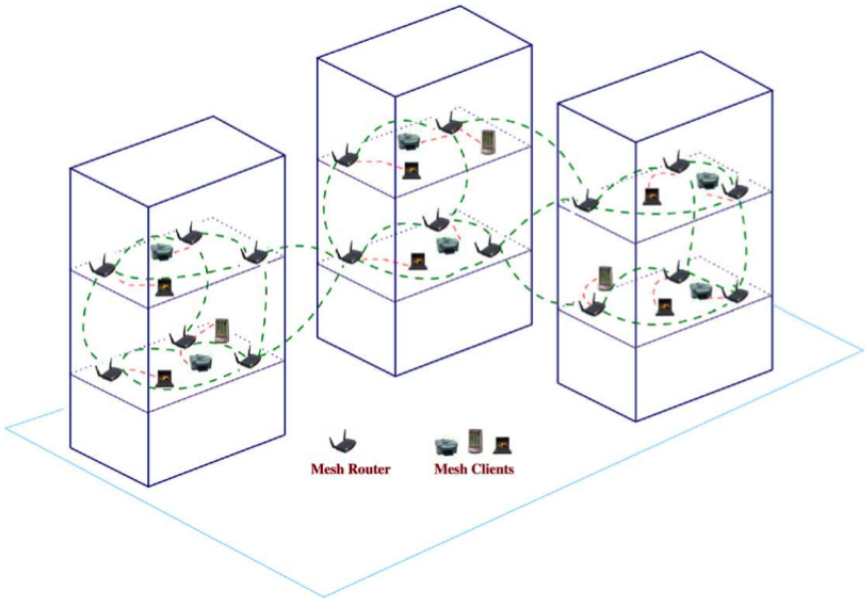


(a) Wireless Mesh Networks for communities

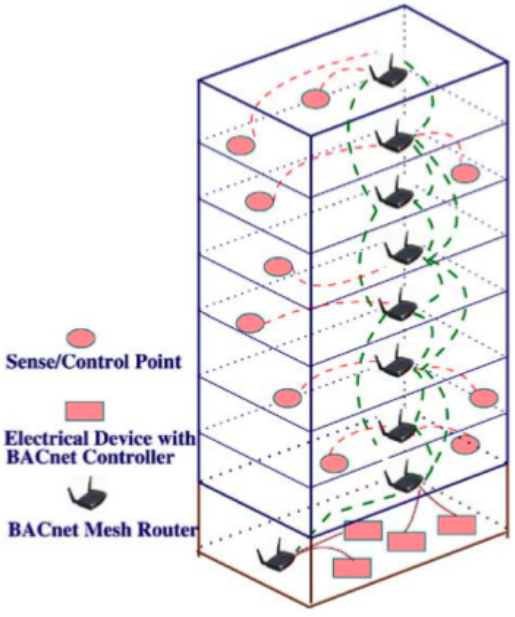


(b) Wireless Mesh Networks for metropolitan cities

Figure 7.3: Additional exemplary network topologies as shown in [2].



(a) Wireless Mesh Networks for enterprise networking



(b) Wireless Mesh Networks for Building Automation

Figure 7.4: Still more exemplary network topologies as shown in [2].

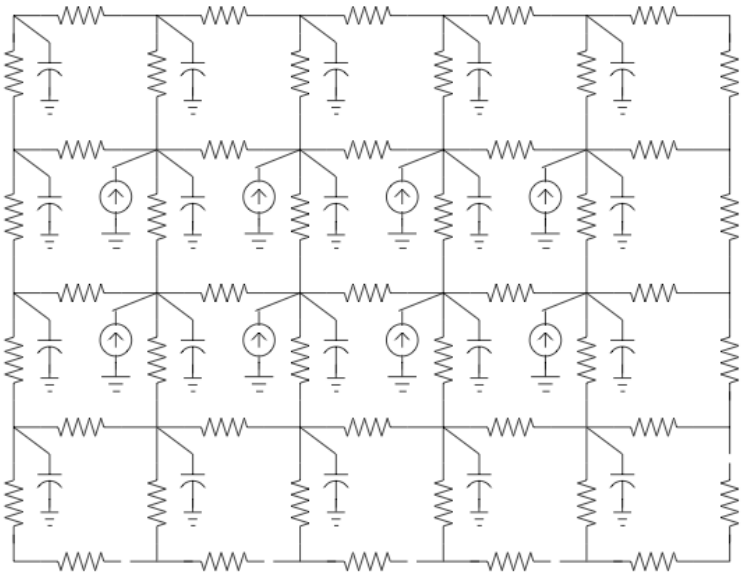


Figure 7.5: Exemplary on-chip power grid network model as shown in [46].

is still high potential for the approach proposed in this thesis to be applied in the power grid domain with some success. One exemplary topology is shown in Figure 7.6, which is currently being used by IEEE as a standard testing network for research purposes. In this case, the individual substations would be the nodes and the voltage or power running through the transmission lines could be used as the input to the neural network.

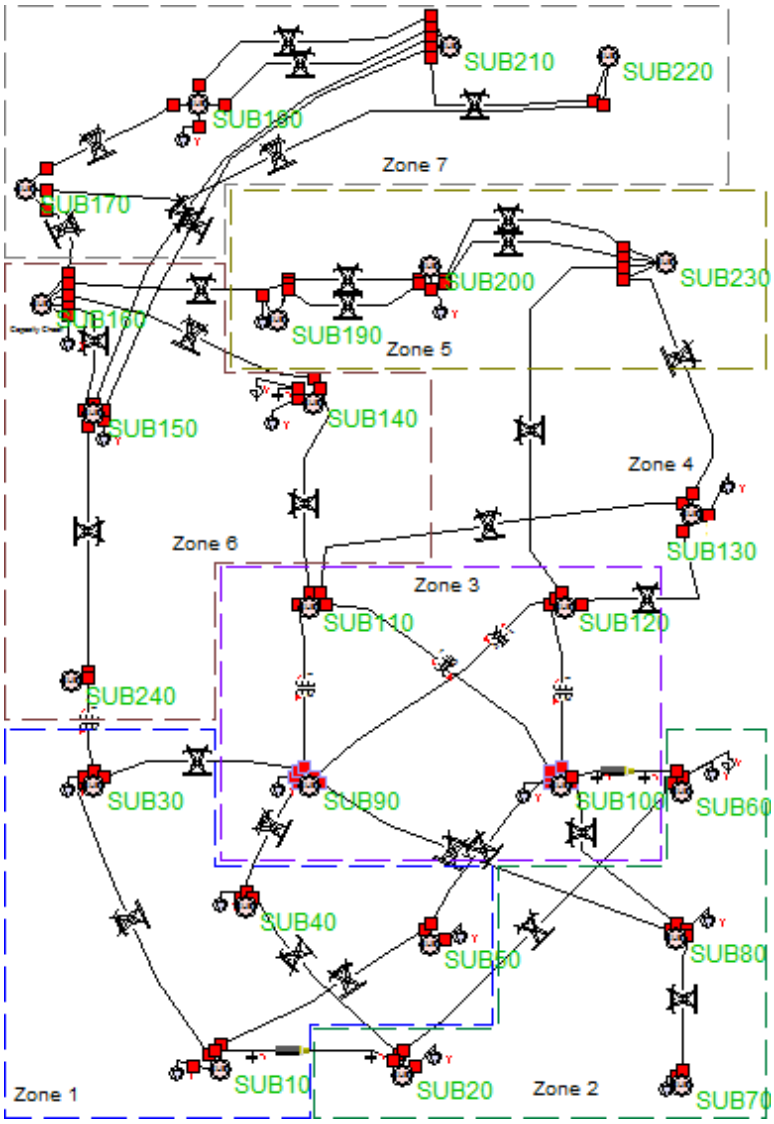


Figure 7.6: Three-phase, breaker-oriented IEEE 24-substation reliability test system [74].

Appendix A

Neural Network Comparisons

This appendix presents some findings that are separate from the core thesis, but contains interesting neural network research nonetheless. Here, additional details on the neural network training will be assessed for each of the three experimental BGP network topologies introduced in Chapter 5. Specifically, for each topology both neural network architectures are trained with the same parameters as used within Chapter 5, however the RMSE lower-bound to halt training is set to 0.05 (5%). This therefore presents more insight into the training time tradeoffs between the Expert and General neural network architectures. Moreover, all training in this thesis was conducted on a custom built desktop running Linux Ubuntu 8.10 (Intrepid Ibex). The computer hardware was comprised of an Asus Rampage Extreme motherboard equipped with an Intel Core 2 Quad Q9550 quad-core 2.83 GHz processor with a 12MB L2 cache, 4GB of 1333 MHz FSB Corsair PC3-10666 memory, and a 500GB Seagate SATA hard drive with a 32MB cache.

A.1 Neural Networks for Five Router Full Mesh

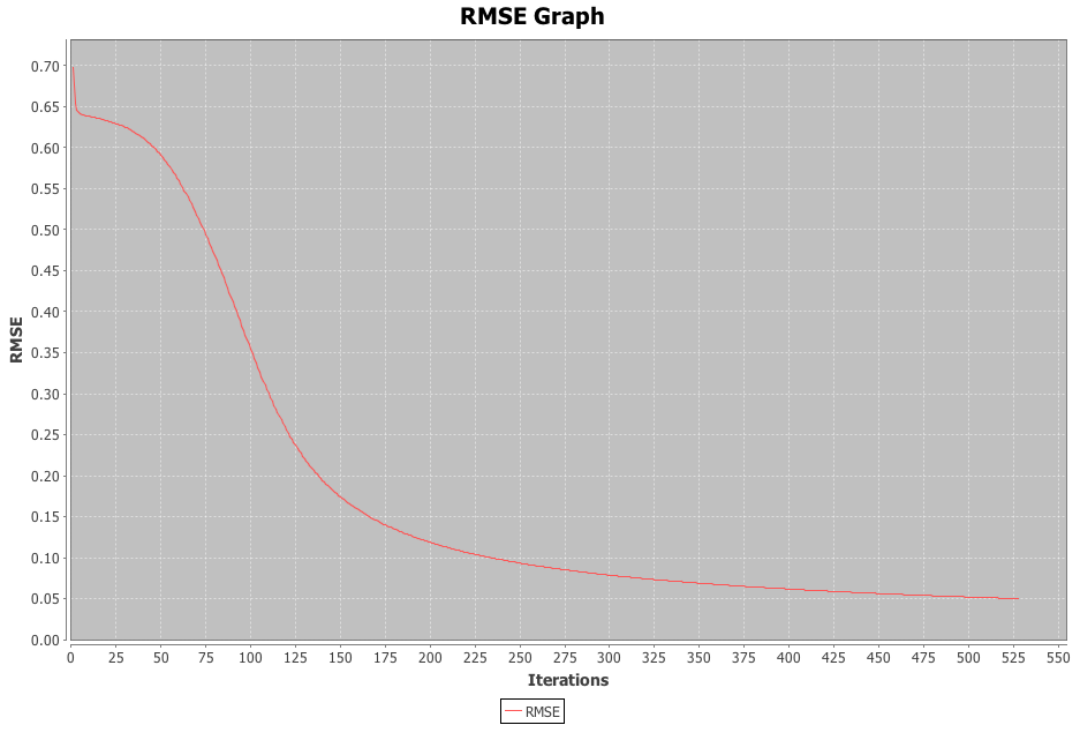
The General neural network trained for 529 iterations with both the learning rate and momentum coefficient set to 0.2. Training took 594 milliseconds and the neural network reached an RMSE of 0.0499 or 4.99%. A graph of the General neural network training is shown in Figure A.1a.

The Expert neural network trained for 177 iterations with both the learning rate and momentum coefficient set to 0.2. Training took 576 milliseconds and the neural network reached an RMSE of 0.0496 or 4.96%. A graph of the General neural network training is shown in Figure A.1b.

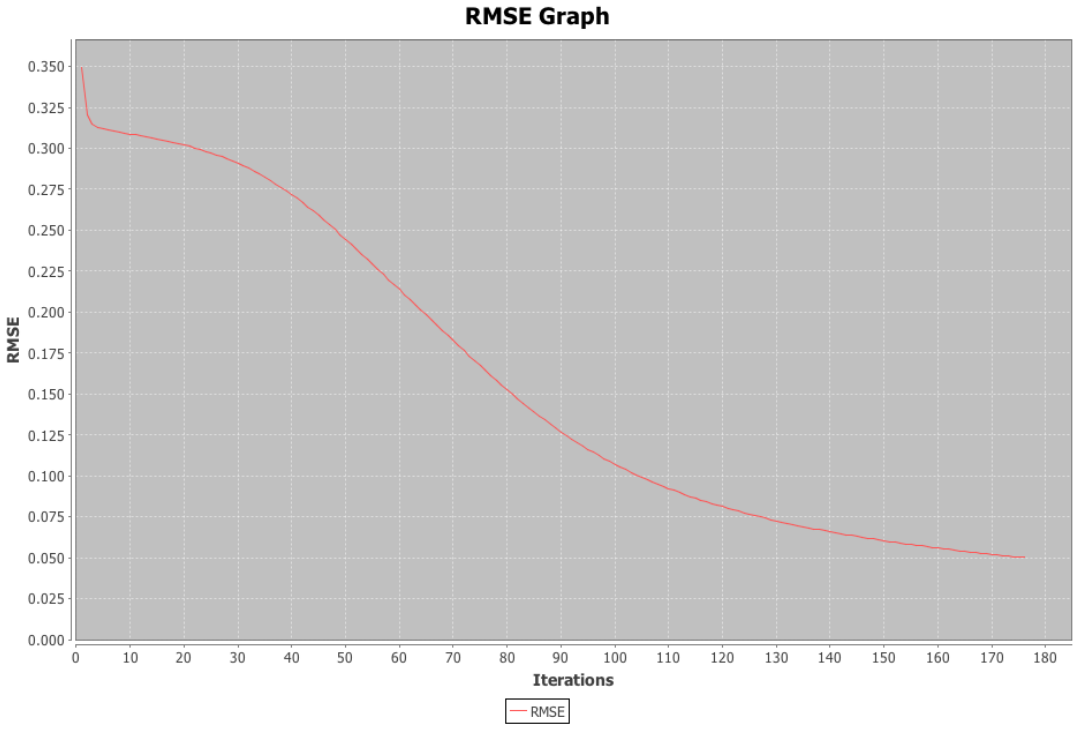
A.2 Neural Networks for Forty Router Full Mesh

The General neural network trained for 91 iterations with both the learning rate and momentum coefficient set to 0.1. Training took 1,623,178 milliseconds (27.05 minutes) and the neural network reached an RMSE of 0.0483 or 4.83%. A graph of the General neural network training is shown in Figure A.2a.

The Expert neural network trained for 17 iterations with both the learning rate and momentum coefficient set to 0.1. Training took 359,437 milliseconds (5.99 minutes) and the neural network reached an RMSE of 0.0289 or 2.89%. A graph of the General neural network training is shown in Figure A.2b.

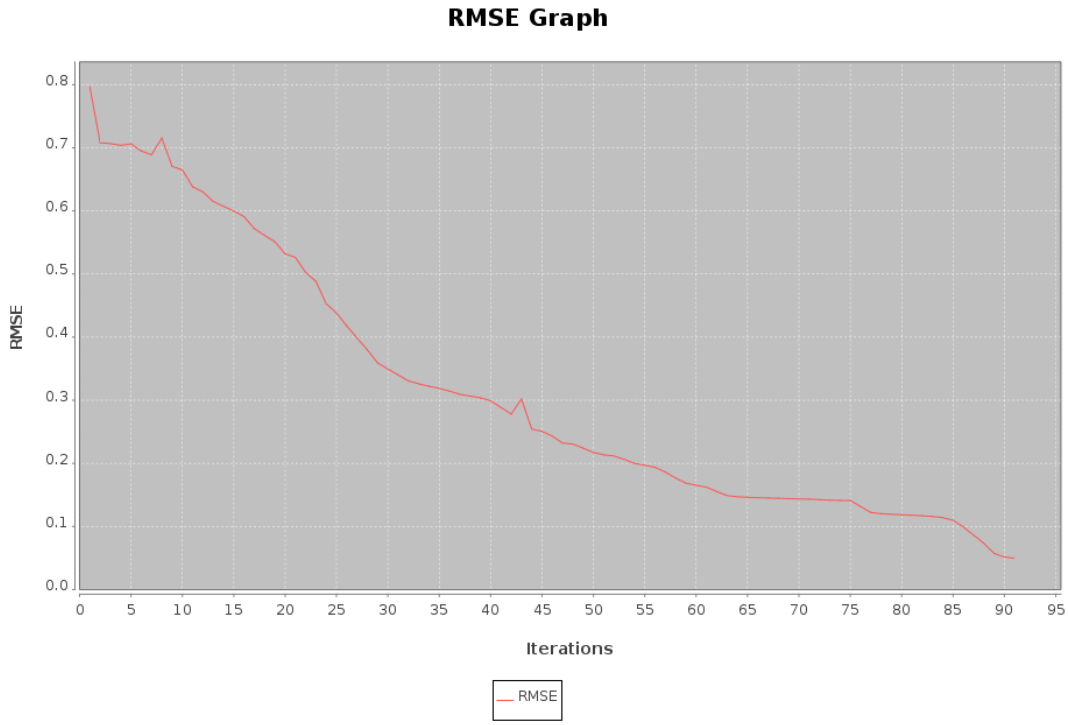


(a) General neural network training

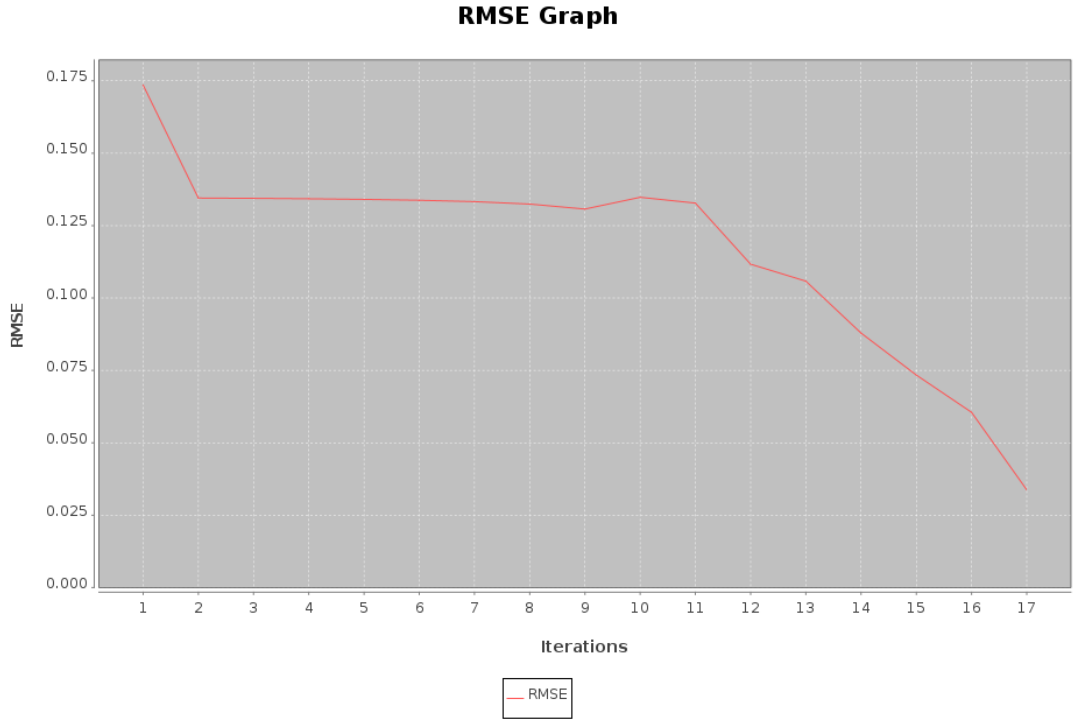


(b) Expert 3 neural network training

Figure A.1: Neural network training comparisons for the five router full mesh.



(a) General neural network training



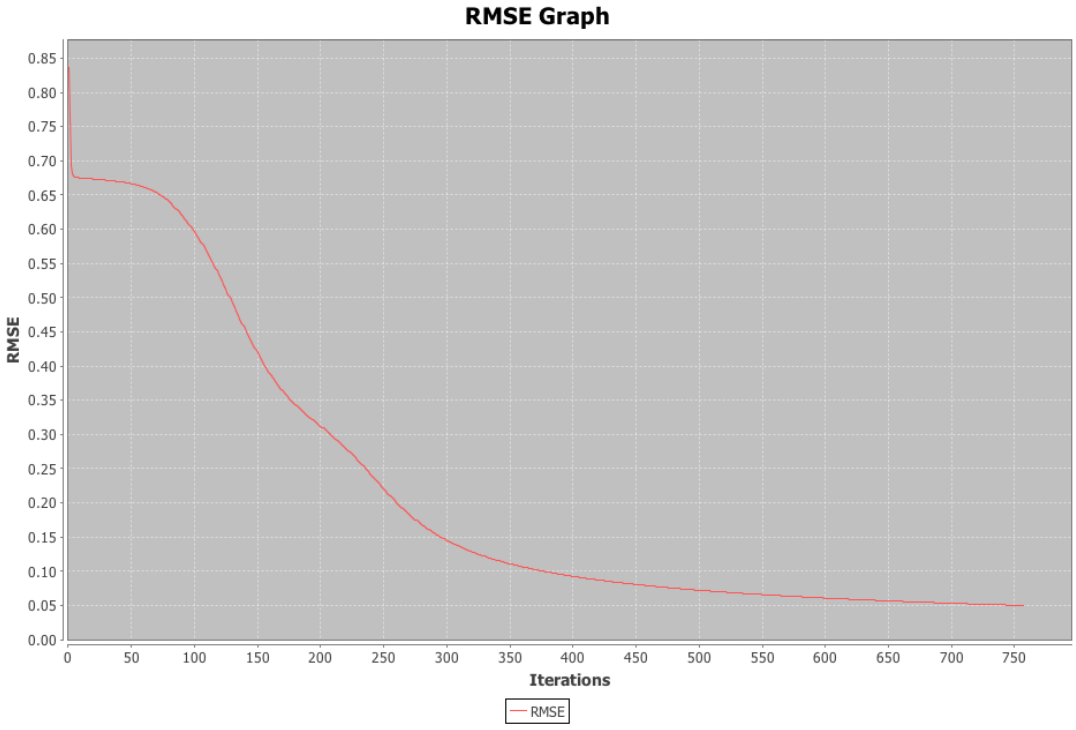
(b) Expert Saiem neural network training

Figure A.2: Neural network training comparisons for the forty router full mesh.

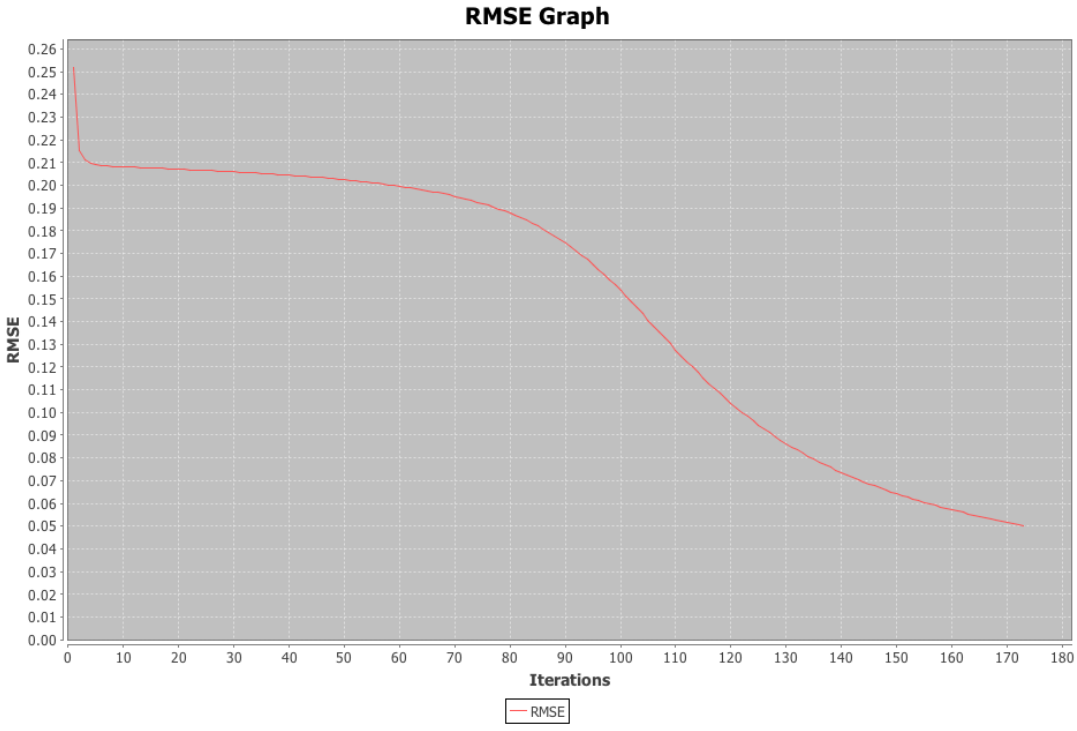
A.3 Neural Networks for Ten Router Sparsely Connected

The General neural network trained for 758 iterations with both the learning rate and momentum coefficient set to 0.2. Training took 2,825 milliseconds and the neural network reached an RMSE of 0.0499 or 4.99%. A graph of the General neural network training is shown in Figure A.3a.

The Expert neural network trained for 174 iterations with both the learning rate and momentum coefficient set to 0.2. Training took 761 milliseconds and the neural network reached an RMSE of 0.0494 or 4.94%. A graph of the General neural network training is shown in Figure A.3b.



(a) General neural network training



(b) Expert 22 neural network training

Figure A.3: Neural network training comparisons for the sparse network topology.

Bibliography

[1] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. A comparison of machine learning techniques for phishing detection. In *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69, New York, NY, USA, 2007. ACM.

[2] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Comput. Netw. ISDN Syst.*, 47(4):445–487, 2005.

[3] G. Alder and D. Benson. Jgraph - the leading java graph drawing component, June 2009. http://www.jgraph.com/jgraph.html.

[4] Artificial neural networks, 2009. http://www.stowa-nn.ihe.nl/pictures/transfer.jpg.

[5] P. Baldi and G. Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *J. Mach. Learn. Res.*, 4:575–602, 2003.

[6] T. Bates, R. Chandra, and E. Chen. BGP route reflection - an alternative to full mesh IBGP. RFC 2796 (Proposed Standard), Apr. 2000. Obsoleted by RFC 4456.

[7] T. Bates, E. Chen, and R. Chandra. BGP route reflection: an alternative to full mesh internal BGP (IBGP). RFC 4456 (Draft Standard), Apr. 2006.

[8] G. D. Battista, L. Cittadini, B. Palazzi, M. Patrignani, M. Pizzonia, T. Refice, M. Rimondini, F. Martorelli, A. Marzioni, S. Vissicchio, and L. Colitti. iBGPlay, visualizing interdomain routing, 2009. http://www.ibgplay.org/.

[9] J. L. Berral, N. Poggi, J. Alonso, R. Gavaldà, J. Torres, and M. Parashar. Adaptive distributed mechanism against flooding network attacks based on machine learning. In *AISec '08: Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 43–50, New York, NY, USA, 2008. ACM.

[10] H. G. Bohr. *Neural Network Prediction of Protein Structures*. Springer Verlag, 2004.

[11] S. R. Boopathy, T. Sasikumar, and E. S. Vasudev. Back propagation neural network prediction of failure strength of composite tensile specimens. *International Journal of Materials and Structural Integrity 2008*, 2(3):288–240, 2008.

[12] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[14] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: A survey and categorisation, 2004.

[15] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[16] K. Butler, T. Farley, P. McDaniel, and J. Rexford. A survey of BGP security . Technical report, AT&T, 2005.

[17] J. Chen and N. Chaudhari. Cascaded bidirectional recurrent neural networks for protein secondary structure prediction. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4(4):572–582, 2007.

[18] J. Chen and N. S. Chaudhari. Bidirectional segmented-memory recurrent neural network for protein secondary structure prediction. *Soft Comput.*, 10(4):315–324, 2006.

[19] J.-L. Chen. A cascading neural-net for traffic management of computer networks. In *CSC '93: Proceedings of the 1993 ACM conference on Computer science*, pages 272–277, New York, NY, USA, 1993. ACM.

[20] H. A. Chipman, E. I. George, and R. E. McCulloch. Bart: Bayesian additive regression trees. *Journal of the Royal Statistical Society*, 2006.

[21] E. K. P. Chong and S. H. Zak. *An Introduction to Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization)*. Wiley-Interscience, 3rd edition, February 2008.

[22] Cisco. Internetworking technology handbook: border gateway protocol (BGP), 2009. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/bgp.html.

[23] L. Colitti, G. D. Battista, I. D. Marinis, F. Mariani, M. Pizzonia, and M. Patrignani. Bgplay @ route views, 2008. http://bgplay.routeviews.org/bgplay/.

[24] L. Cottrell. Network monitoring tools, August 2009. http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html#bgp.

[25] A. Datta, V. Talukdar, A. Konar, and L. C. Jain. A neural network based approach for protein structural class prediction. *J. Intell. Fuzzy Syst.*, 20(1,2):61–71, 2009.

[26] L. de Sá Silva, A. C. F. dos Santos, T. D. Mancilha, J. D. S. da Silva, and A. Montes. Detecting attack signatures in the real network traffic with ANNIDA. *Expert Syst. Appl.*, 34(4):2326–2333, 2008.

[27] D. K. Electronics, D. H. Kemsley, T. R. Martinez, and D. M. Campbell. A survey of neural network research and fielded applications. *International Journal of Neural Networks*, 2:123–133, 1992.

[28] F. Fock and J. Katz. The SNMP API for java, July 2009. http://www.snmp4j.org/.

[29] T. E. Foundation. Epilepsy and the brain: Functions and makeup, 2009. http://www.epilepsyfoundation.org/about/science/functions.cfm.

[30] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.

[31] J. Gaunt. Neural networks, 2009. http://www.jgaunt.com/masters/neuralnets.html.

[32] B. Group. The neuron model, 2009. http://www.basegroup.ru/images/neural/math/fig1.en.gif.

[33] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 32–32, Berkeley, CA, USA, 2005. USENIX Association.

[34] J. Guo, Y. Lin, and Z. Sun. A novel method for protein subcellular localization based on boosting and probabilistic neural network. In *APBC '04: Proceedings of the second conference on Asia-Pacific bioinformatics*, pages 21–27, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[35] J. Hawkins and S. Blakeslee. *On Intelligence*. Holt Paperbacks, August 2005.

[36] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression (Wiley Series in Probability and Statistics)*. Wiley-Interscience Publication, September 2000.

[37] S. M. A. Incorporated. *InCharge: Network Protocol Manager for BGP Users Guide*. System Management ARTS Incorporated, 2004.

[38] Juniper. Managing a large-scale AS, 2009. http://www.juniper.net/techpubs/software/erx/erx41x/swconfig-routing-vol2/html/BGP_route_reflection.gif.

[39] I. Juniper Networks. Autonomous systems, 2009. http://www.juniper.net/techpubs/software/junos/junos60/swconfig60-routing/html/bgp-overview3.html#1013982.

[40] I. Juniper Networks. BGP overview, 2009. http://www.juniper.net/techpubs/software/junos/junos60/swconfig60-routing/html/bgp-overview.html.

[41] D. B. Karunakar and G. Datta. Prediction of defects in castings using back propagation neural networks. *International Journal of Modelling, Identification and Control 2008*, 3(2):140–147, 2008.

[42] D. Kemsley, T. Martinez, and D. Campbell. A Survey of Neural Network Research and Fielded Applications. *International Journal of Neural Networks*, 2(2/3/4):123–133, 1992.

[43] R. D. King. Drug design, protein secondary structure prediction and functional genomics. *SIGBIO Newsl.*, 18(3):5–5, 1998.

[44] M. Lad, J. H. Park, J. Zhu, D. Massey, and L. Zhang. Linkrank visualization, 2009. http://linkrank.cs.ucla.edu/.

[45] T.-L. Lee. Back-propagation neural network for the prediction of the short-term storm surge in Taichung Harbor, Taiwan. *Eng. Appl. Artif. Intell.*, 21(1):63–72, 2008.

[46] D. Li, S. X.-D. Tan, G. Chen, and X. Zeng. Statistical analysis of on-chip power grid networks by variational extended truncated balanced realization method. In *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 272–277, Piscataway, NJ, USA, 2009. IEEE Press.

[47] X.-Y. Li, P.-J. Wan, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless networks. *Hawaii International Conference on System Sciences*, 9:296b, 2002.

[48] M. Liljenstam, J. Liu, and D. M. Nicol. Simulation of large scale networks ii: development of an internet backbone topology for large-scale network simulations. In S. E. Chick, P. J. Sanchez, D. M. Ferrin, and D. J. Morrice, editors, *Winter Simulation Conference*, pages 694–702. ACM, 2003.

[49] O. R. Limited. Jfreechart, October 2009. http://www.jfree.org/jfreechart/.

[50] K.-L. Lin, C. Y. Lin, C.-D. Huang, H.-M. Chang, C. Y. Yang, C.-T. Lin, C. Y. Tang, and D. F. Hsu. Improving prediction accuracy for protein structure classification by neural network using feature combination. In *AIC'05: Proceedings of the 5th WSEAS International Conference on Applied Informatics and Communications*, pages 313–318, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[51] V. Maiorov. Approximation by neural networks and learning theory. *J. Complex.*, 22(1):102–117, 2006.

[52] P. Marrone. *The Complete Guide: All You Need to Know About Joone.* January 2007. http://www.jooneworld.com.

[53] P. Marrone and Xharze. Java Object Oriented Neural Engine, June 2009. http://sourceforge.net/projects/joone/.

[54] D. McCullagh. How Pakistan knocked YouTube offline (and how to make sure it never happens again), February 2008. http://news.cnet.com/8301-10784_3-9878655-7.html.

[55] J. W. Mickens and B. D. Noble. Predicting node availability in peer-to-peer networks. *SIGMETRICS Perform. Eval. Rev.*, 33(1):378–379, 2005.

[56] M. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Mag.*, 12(2):34–51, 1991.

[57] M. Minsky and S. Papert. *Perceptrons.* Boston, MA: MIT Press, 1969.

[58] D. G. Myers. *Exploring Psychology.* Worth Publishers, 6 edition, April 2004.

[59] B. Naveh and J. V. Sichi. Jgrapht, June 2009. http://jgrapht.sourceforge.net/.

[60] T. N. I. on Drug Abuse (NIDA). Psychoactive drugs and the brain, 2009. http://www.drugabuse.gov/JSP/MOD3/images/NEURON2.gif.

[61] N. Patwari, A. O. Hero, III, and A. Pacholski. Manifold learning visualization of network traffic data. In *MineNet '05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 191–196, New York, NY, USA, 2005. ACM.

[62] S. Rajendraboopathy, T. Sasikumar, K. M. Usha, and E. S. Vasudev. Artificial neural network a tool for predicting failure strength of composite tensile coupons using acoustic emission technique. *The International Journal of Advanced Manufacturing Technology*, 44(3-4):399–404, September 2009.

[63] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). RFC 1771 (Draft Standard), Mar. 1995. Obsoleted by RFC 4271.

[64] R. Rojas. *Neural Networks: A Systematic Introduction*. Springer, July 1996.

[65] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, 1962.

[66] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[67] A. Saple and L. Yilmaz. Agent-based simulation study of behavioral anticipation: anticipatory fault management in computer networks. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 383–388, New York, NY, USA, 2006. ACM.

[68] H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80, 1991.

[69] L. d. S. Silva, A. C. F. d. Santos, A. Montes, and J. D. d. S. Simoes. Hamming net and LVQ neural networks for classification of computer Nntwork attacks: a comparative analysis. In *SBRN '06: Proceedings of the Ninth Brazilian Symposium on Neural Networks*, page 13, Washington, DC, USA, 2006. IEEE Computer Society.

[70] A. Sperduti. *A tutorial on neurocomputing of structures*. MIT Press, Cambridge, MA, USA, 2000.

[71] M. Stoecklin. Anomaly detection by finding feature distribution outliers. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–2, New York, NY, USA, 2006. ACM.

[72] C. Systems. Simple network management protocol (SNMP), july 2009. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNMP.html.

[73] J. G. Taylor. *Handbook of Neural Computation*. IOP Publishing Ltd and Oxfor University Press, 1997.

[74] G. Tech. Three-phase, breaker-oriented IEEE 24-substation reliability test system, 2009. http://pscal.ece.gatech.edu/testsys/index.html.

[75] A. Toonk. Welcome to bgpmon.net, a bgp monitoring and analyzer tool, 2008. http://bgpmon.net/.

[76] L. G. Valiant. Functionality in neural nets. In *COLT '88: Proceedings of the first annual workshop on Computational learning theory*, pages 28–39, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc.

[77] B. S. Vijayaraman and B. Osyk. A survey of neural network publications. In *Proceedings of the International Academy for Information Management Annual Conference*, 1997.

[78] Y. Wang, D. Gu, J. Xu, and J. Li. Back propagation neural network for short-term electricity load forecasting with weather features. *Computational Intelligence and Natural Computing, International Conference on*, 1:58–61, 2009.

[79] P. Werbos. *Beyond Regression*. PhD thesis, Harvard University, 1974.

[80] H. White. Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings. *Neural Netw.*, 3(5):535–549, 1990.

[81] H. White. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell Publishers, Inc., Cambridge, MA, USA, 1992.

[82] H. White, A. S. Cr, and I. Introducnon. Nonparametric estimation of conditional quantiles using neural networks. In *Proceedings of the Symposium on the Interface*, pages 190–199, 1992.

[83] Wikimedia. Artificial neural networks, 2009. http://upload.wikimedia.org/wikipedia/commons/thumb/6/60/ArtificialNeuronModel_english.png/600px-ArtificialNeuronModel_english.png.

[84] R. Xu, D. W. II, and R. Frank. Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4(4):681–692, 2007.

[85] K. Yamanishi and Y. Maruyama. Dynamic syslog mining for network failure monitoring. In *KDD '05: Proceedings of the eleventh ACM SIGKDD inter-*

national conference on Knowledge discovery in data mining, pages 499–508, New York, NY, USA, 2005. ACM.