

Forecasting and Mitigating Network Congestion using Neural Networks

J. Alan Bivens, Mark J. Embrechts, and Boleslaw K. Szymanski

Rensselaer Polytechnic Institute
Department of Computer Science and
Department of Decision Sciences & Engineering Systems
Troy, NY USA 12180-3590
Email: bivenj@cs.rpi.edu, embrem@rpi.edu, szymansk@cs.rpi.edu

Abstract

Due to the quickly increasing need for networking, both on the Internet and intranet levels, network management is becoming more important in today's world. We propose using learning techniques to predict and fix different network problems before they start. In this paper we focus on using a Perceptron-type Neural Network to predict problems such as severe congestion in a network. This scenario sets the stage for a new wave of network managers, those that prevent networking problems instead of repairing them.

1 Introduction

The growing emphasis is being placed today on reliability, maintenance, and dependability of networks. Network problems cause annoying blackouts of communication and may cost a company thousands, or even millions, of dollars per occurrence. To address this issue, a system is needed to insure network availability and efficiency by preventing such costly network problems. The first step towards this end is creating a system with the intelligence to recognize what signals lead to network problems. If the problem can be recognized in advance, changing a couple of network parameters can possibly circumvent its effects.

Some sophisticated network management suites of applications try to address the problem. However, they merely give the network administrator a great deal of information about the problem after it has happened, hoping this will help the administrator find the cause of the problem quickly. Most of these products do not try to detect the problem before it starts.

We propose a system that detects these network problems, using neural networks, before they shut down a network. This system determines not only if a problem is arising, but also finds the source of the problem. Once the source of the problem is determined, the problem can be fixed before it surfaces or shortly thereafter.

2 Related Work

Approaching networking issues with Neural Networks is not uncommon. A group at Rensselaer Polytechnic Institute, is doing very similar research. They first detected changes in traffic patterns through the use of a sequential Generalized Likelihood Ratio (GLR) test. That information was then correlated with a filter to provide various alarms. Their algorithm was "trained" or optimized using five of nine fault data sets, and proved general enough to detect three out of the remaining four [3] [4]. The structure used by this group is very similar to the structure of this research. However, this group used a statistical method for detecting patterns, while we proposed a neural network to learn the patterns leading to network faults. We also focus on correcting the problem in a timely fashion, whereas their work was just in detection.

Similar studies have been done in ATM networks with Connection Admission Control (CAC) schemes. In these works, the authors use fast converging neural networks to predict cell loss in a switch using data from a cell bouncing between two switches [2]. This effort involved a degree of prediction, but offers solution for determining the problem source or solution.

In the Intelligent Network (IN) community, work has been done to characterize and learn the "correct" behavior of network elements with neural networks in respects to run-time feature interactions of a live network. This work was done to monitor and handle interactions between services offered by different providers, many of which do not provide examination abilities for all service specifications. Because many of the service specifics are unknown, the normal behavior of the service is learned as if it was a black box. Once the behavior is learned, this knowledge could be used to determine when feature interactions have occurred [5]. Although, the goals of this research are very different, the same concepts of learning the behavior of network devices is applied here.

A group with closer research goals can be found at

the University of Michigan. There is work being done there in congestion control and mitigation strategies. However, this group uses a queuing theory approach rather than a neural network to guide the mitigation efforts. And their mitigation efforts consist of rerouting the traffic instead of implementing a control architecture [1].

3 Architecture

A high level view of our architecture reveals a network with a control agent existing somewhere on a node in that network. This control agent has both, the power to read from network nodes, and the power to influence those network nodes. Either the nodes involved would report the necessary statistics to the control agent or the control agent would poll these nodes.

3.1 Data Network

An artificial data communication network was created using NS, a network simulator. The networks used consisted of several nodes in a configuration where all of the network nodes were attempting to send to one node. Each node attempts to send data at a random bit rate. Different variances are assigned to these rates at different nodes to better represent traffic in a real network and possible traffic coming in from nodes outside of our simulation. The link capacity between sending nodes were given arbitrary values. Some links were able to handle much more traffic than others.

3.2 Control Agent

We create a control agent containing a neural network that is trained prior to being placed in production. In our simulation, the control agent is part of the simulation code. This enables the agent to easily monitor and influence traffic statistics from each node. It could also be implemented as a stand-alone agent in a real network. The control agent gathers information from each managed node and makes a decision about whether network problems will occur. With the predicted problem in our grasps, we can then take steps to stop or prevent it.

4 Implementation Details

4.1 The Simulation Network

The simulated network is arranged so that several sending nodes are connected to one receiving node through several links which carry the packets. To determine how fast the sender sends data, the packet size and a packet interval must be given. The sender will send a packet of the required size at the required interval. We gather statistics, such as packets received and an estimate of the current bit rate at the preset

polling interval, during the simulation. Files are created to keep track of the data for each node. During each run of the control agent, the files are extended with the new data from the latest interval.

The most important part of the Control Agent is the Neural Network prediction module. For our agent we used a single hidden layer, feed-forward neural network. This was a compiled application, so wrappers were needed to control the input and output dealing with the Neural Network.

When the control agent is executed, a C wrapper is first called. This is where the bulk of the calculations for the neural network program are done. The wrapper handles all of the file manipulation, statistical calculations and normalizations, and the starting of our neural network package. The neural network program is executed using the new input files and a verdict is rendered in an output file. This file is read by the C wrapper and converted into a readable format for the tcl program to continue.

4.1.1 Neural Network Specifics

The neural network is run by the C program every time the control agent is called. It has $3*N$ input nodes, one hidden layer containing N nodes, and N nodes in the output layer, where N denotes the number of traffic generating nodes in the network simulation. There are three input nodes for each node in the network simulation, each corresponding to the average number of packets, its variance and the third momentum at each monitored node. The N nodes in the output layer correspond to an orthogonal encoding of the node or nodes predicted to be the source of the forecasted problem.

An additional twist placed on the neural network involved pruning the weights to the point that the neural network reflected the connectivity of the actual network. An example is shown in Figure 1.

In the model of the neural network in Figure 1, the hidden layer is representative of the participating nodes in the data network. The statistical data regarding each node is provided to the hidden node representing the actual node as well as to the hidden nodes representing the actual node's neighbors. This is continued for all of the first layer nodes in the neural network. As a result, the statistical information from node 1 is given to both the hidden node representing node 1, and the hidden node representing node 5 (1's neighbor.) This claim is important in realizing the relationships between adjacent nodes in a data communications network.

The neural network was trained off-line, with eighty-eight patterns. Half of the patterns were samples containing no network problems and half had congestion problems at various locations. In training the neural network, early stopping was used, allowing the training to go for about 15000 iterations, when the least squares error was equal to or less than .04%.

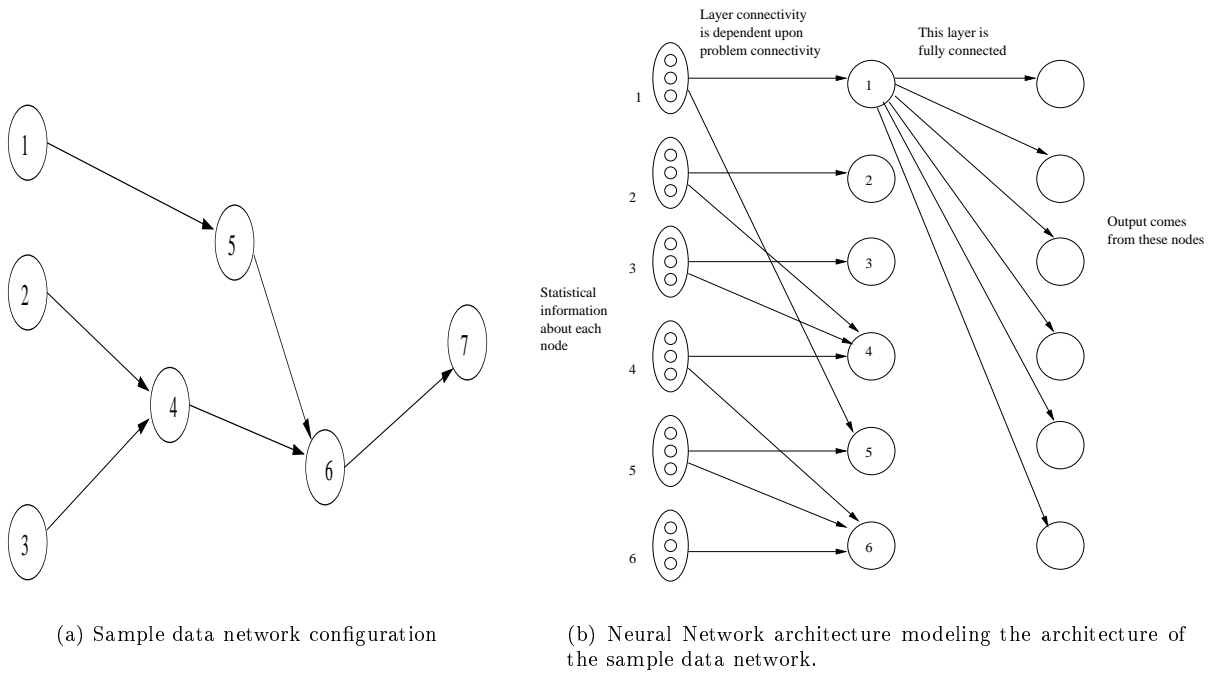


Figure 1: Relationship between data network configuration and a neural network.

4.1.2 Control from the Agent

As stated before, in NS, an interval and packet size are provided for agents sitting at the sending nodes to determine bit rate. The agent will send one packet at every interval. Therefore the smaller the interval, the higher the bit rate.

If our neural network predicts that a particular node will be responsible for congestion, we accuse the predicted problem source of using too many resources. To punish the predicted cause node, we add to its sending interval a small δt , thus reducing it's bit rate.

For example if Node 1 was predicted as the problem source, the equation to the right would explain how Node 1 will be corrected each

$$Interval_1 = Interval_1 + \delta t$$

Figure 2: Interval change formula

time it is predicted as the problem. This delta was chosen to be small because we don't want to take the chance of over-correcting or, even worse, in case our prediction is wrong, applying a large rate correction to the wrong node. To take into account the small δt , the interval in which our control agent executes is also relatively small. Therefore many of these small corrections can be applied and correct a problem slowly without drastically changing any one node's level of service.

5 Results

A general breakdown of the results can be found in the exploding pie chart of Figure 3. Figure 3 shows that our detector failed in about 10% of the cases. Failing can be interpreted by the detector either missing congestion or predicting congestion when there is no problem. However, as explained later, all 10% dealt

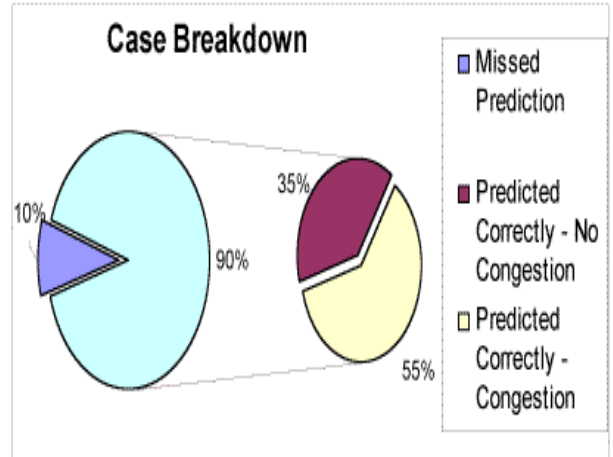


Figure 3: Breakdown of Results

with cases in which the detector missed congestion. In the remaining success cases, 90%, 35% of those were cases in which the detector applied no correctional procedures, but none were necessary. The last 55% of the

correct cases, were cases in which the detector accurately applied correctional procedures. A more detailed description of the test simulations is given below.

We ran thirty-one simulations of networks. Eleven of those thirty-one were simulations of a network without congestion problems. In these cases we want our control agent to realize it doesn't have to do anything. This happened in ten out of the eleven cases; however, there was one case that our agent unnecessarily applied a single small fix to a single node. The correction the control agent applied was with a single δt , so it was minimal.

Twenty of the thirty-one simulations had various levels of congestion in various locations in the network. Of these twenty cases, we were able to predict the cause and fix the problems of seventeen cases. Of these seventeen, there were eleven cases in which we were able to fix the problem before the problem even surfaced in the network. These were truly remarkable results, because the congestion was completely eliminated before ever occurring. The other six cases were fixed in less than 3.5 seconds after the congestion appears.

This leaves three cases in which the neural network did not notice a problem that was occurring. Even this news is encouraging, because these cases had some common characteristics. The neural network had trouble detecting when a single node in a particular part of the data network caused a problem. This can probably be easily fixed upon close examination of the training patterns and structure of the neural network.

6 Future Work

There are many ways this system can be expanded. The first problem that needs to be addressed is scalability. An extended architecture can be formed to accommodate a large communication network by dividing this network into domains small enough to be influenced easily by an agent (most large networks already are structured as a collection of domains). When this is in place, a separate protocol or addition to current protocols can be used to determine domain-to-domain agreements.

At present, our system only predicts congestion sources and implements a solution. It would be useful to create a system which can do the same for other problems. It would also be of great value to train the network over many different types of traffic. Introducing an optimization component to optimize the network resources instead of just safeguarding them from congestion would be a worthwhile goal to achieve. Lastly, it would be most interesting to implement this system on a real network to see if similar or better results can be obtained.

The relationship of the data feeding to physically adjacent nodes does a great deal to localize the gener-

alization power of the neural network. As is common with neural networks, a close analysis of the weights in these situations could help us understand a distribution model for data traffic, another open area of research.

7 Summary

In this work, we demonstrated that a Neural Network is a viable method of implementing a learning mechanism for data communication networks. We have illustrated, through the use of a network simulator, that a Neural Network can achieve great accuracy in predicting the particular network problem of congestion. We realize that many more problems exist, but predicting congestion is the first step. We have also shown how a carefully constructed neural network can achieve above average results when structural information about the actual data network is used to form the neural network. This is a feature that forces the neural network to only consider the relationships of nodes that we think are important.

A learning mechanism can be of great value as a network manager. The generalization power of a neural network is particular appropriate because of the unpredicted variance of parameters the manager will have to deal with. Neural networks are certainly an appropriate mechanism for decision making in pro-active network management, and should be the target of more research.

References

- [1] Wu chang Feng. *Improving Internet Congestion Control and Queue Management Algorithms*. PhD thesis, The University of Michigan, Ann Harbor, Michigan, 1999. Computer Science and Engineering.
- [2] C.K. Tham and W. S. Soh. Multi-service connection admission control using modular neural networks. In *Proceedings of the Conference on Computer Communications*, page 1022, San Francisco, California, March 1998. IEEE Infocom.
- [3] Marina Thottan and Chuanyi Ji. Adaptive thresholding for proactive network problem detection. In *IEEE Internation Workshop on Systems Management*, Newport, Rhode Island, April 1998. IEEE.
- [4] Marina Thottan and Chuanyi Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network, Special Issue on Network Management*, September 1998.
- [5] Simon Tsang and Evan H. Magill. Learning to detect and avoid run-time feature interactions in intelligent networks. *IEEE Transactions on Software Engineering*, 24(10), October 1998.