Simple Solutions to Network-Based Intrusion Detection

Razvan Surdulescu, Sukant Hajra

Department of Computer Sciences
The University of Texas at Austin
Austin, TX, 78705
{surdules, tnks}@cs.utexas.edu

Abstract

Current intrusion detection systems (IDS) do not easily adapt to detecting new attacks. Artificial neural networks (ANN) have been shown to successfully address this problem. In this paper, we show that a simple, feed-forward ANN performs surprisingly well when aggressively trained and tested on a realistic corpus of simulated attacks. We compared it to a similarly basic recurrent network (Elman) and we find that the feed-forward network detects attacks with fewer false positives and negatives. We explore a naive smoothing approach for improving ANN IDS detection rates.

Introduction

Security is becoming increasingly more important in our connected world. As the potential payoff for successful intrusions increases, so does the complexity and sophistication of attacks. Furthermore, the amount of network traffic noise makes it all that much easier to dissimulate the attack signal. We must fortify our current IDS solutions to keep up.

Current intrusion systems focus on detecting known attacks; the literature denotes this approach as *misuse detection*. They are generally implemented as rule-based signature-matching or statistical analysis systems. Human creation and maintenance of rules is slow and costly. The rules become obsolete and difficult to maintain over time, and they do not automatically adapt to discovering new attacks. The performance of such systems tends to suffer from high false negative counts (many attacks go unnoticed).

Systems that detect anomalies (new attacks) are very promising. Such systems "learn" the difference between normal and anomalous behavior, and report it. Aside from a nominal setup, they tend to operate with minimum human intervention. They are generally built using some form of ANN architecture. Existing research suggests that they suffer from high false positive counts (many reported attacks are benign).

We believe that a simple ANN architecture can be very effective at detecting anomalies. Previous research has focused on complex, recurrent architectures, but we feel that the strengths of a simple, feed-forward architecture have not been realistically explored. In this body of work, we have found that simple architectures do perform surprisingly well, specifically on the KDD Cup dataset [Hettich and Bay, 1999].

Background

There are many dimensions to the problem of detecting intrusions in a computer network [Axelsson, 2000]. We have already alluded to the dichotomies between misuse (signature) vs. anomaly, and supervised vs. unsupervised learning. Other important dimensions are domain (host vs. network based), detection window (periodic log evaluation vs. real-time), and implementation (local vs. distributed).

Attacks develop sequentially over time, so an ANN that learns attacks will benefit from having some form of memory [Ghosh et al., 1999]; memory is generally implemented by means of a recurrent architecture, such as Elman or Jordan [Elman, 1990]. To mitigate the false counts, a "leaky bucket" algorithm can be employed to smooth out erratic behavior [Ghosh et al., 1999].

Typical user activity can be assumed to cluster in neighborhoods, so a SOM can find the underlying natural organization of this data and detect anomalies [Hoglund et al., 2000]. Multiple SOMs can be grouped in a hierarchy where each level finds the salient features of the previous level [Lichodzijewski, 2002]. Unlike ANNs, SOMs can be used both quantitatively and qualitatively: the user can look at a physical map and provide another level of qualitative validation to the quantitative results from the SOM.

The strengths of ANN and SOM can be combined in hybrid systems where the SOM clusters the data according to similarity and the ANN classifies the data according to type [Jirapummin et al., 2002]. This research is still very much in its infancy.

Approach

We based our research largely on the groundwork in [Ghosh et al., 1999], which makes a compelling argument that the Elman recurrent network detects anomalies better than a simple feed-forward network. However, their methodology inherently frames intrusion detection as a regression problem by using ANNs to detect whether or not an attack follows the current event.

We believe that the problem of detecting anomalies is best framed as a classification problem, namely to classify a given current event as either anomalous or normal. Consequently, we trained our ANNs differently. In the line of [Ghosh et al., 1999], we compare the relative performance of a simple feed-forward network to a basic recurrent Elman network.

There is a staggering amount of available data [MIT, 1998] for training an IDS. For practical and theoretical reasons, we believe that it is unfeasible to use all this data. From the start, we outlined a methodology for reducing the size of the data without compromising the quality of the results.

[Ghosh et al., 1999] demonstrates the effectiveness of a "leaky bucket" smoothing algorithm. We believe this algorithm has equal merit in our classification approach and therefore we experimented with it.

Because we approached the problem from a different angle than [Ghosh et al., 1999], we methodically searched the parameter space in a tiered fashion to find the optimal configuration parameters.

Experiments

Network Architecture

To establish a comparison, we implemented a feed-forward and a recurrent Elman ANN. Figure 1 illustrates the two architectures; note that the feed-forward ANN does not have the Elman "context layer" nodes. For each hidden node, a context node (Figure 2) stores its previous output from event to event. The context nodes typically have a linear activation function.

We used the JOONE [Marrone et al., 2004] ANN library. This library's context node utilizes an atypical linear function combining the input and the *previous* output:

$$\mathbf{out} = beta * (\mathbf{input} + (k * \mathbf{out}))$$

where $beta$ and k are constants.

Because ANN-based detection systems are often prone to false positives, we additionally filtered the output of the network through the leaky bucket algorithm. The output from the network is classified as *normal* vs. *attack* (see Appendix A), and if deemed *attack*, stored in the leaky bucket. If the leaky bucket level ever goes over a certain threshold, the network is said to have detected an anomaly. This classification heuristic is a departure from [Ghosh et al., 1999] where the ANN outputs (predicts) the next connection vector, and the discrepancy is then classified as *attack*. We made this choice because we felt that the ANN, especially the feed-forward kind, would shine as a classifier as opposed to a predictor.

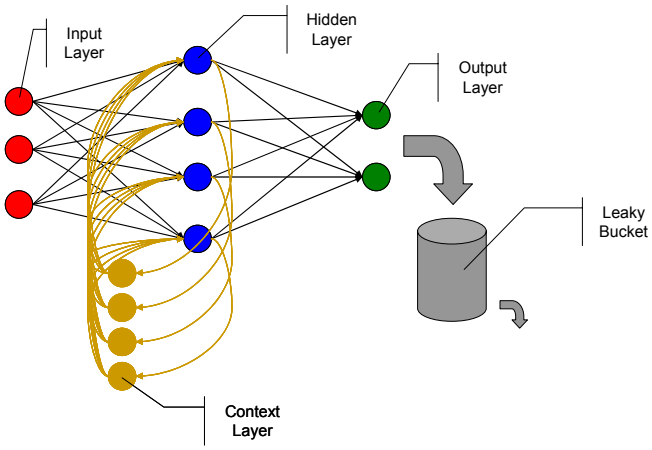

Figure 1: Recurrent Elman ANN Architecture

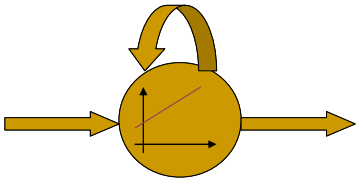

Figure 2: Context Node Detail

Dataset

We used the dataset from the KDD Cup competition [Hettich and Bay, 1999]. This dataset is also used by a number of research papers, so it is recognized in the research community as a standard corpus for this type of problem. This dataset originates from a very comprehensive joint DARPA-MIT experiment [MIT, 1998]. The original DARPA dataset is gigantic (9GB), while the KDD Cup dataset is smaller (1.2GB). Ideally, we would have liked to use the DARPA dataset, but we were forced to select the KDD Cup dataset due to time constraints. We ended up using a 10% subset of the full KDD Cup dataset (selected with even sampling by the KDD Cup committee).

In order to run ANN experiments on this dataset, we had to pre-process it further. At a high-level, this dataset contains vectors that roughly correspond to connections. A connection is "a sequence of TCP packets starting and ending at some well defined times, between which data

flows from a source IP address to a target IP address under some well defined protocol" [Hettich and Bay, 1999]. Each vector represents a set of connection features and is labeled either as *normal* or *attack* (there are 39 different attack types).

Because the KDD Cup data was designed for a competition, only the training data has labels, all the test data is unlabeled. In order for us to determine the accuracy of our experiments (i.e. obtain labeled data), we performed 5-fold cross-validation on this data from the start: we split the data into 5 subsets, and iteratively trained on four of the five subsets, each time using a different subset as a "hold out" test set. The relative proportions of the attack labels are in Table 1.

Table 1: Relative Proportions of Attack Labels

Sum of Count	Fold					
Type	1	2	3	4	5	Grand Total
Normal	11.38%	3.03%	0.00%	1.37%	3.90%	19.69%
DoS	8.23%	16.45%	20.00%	18.53%	16.03%	79.24%
Probe	0.37%	0.31%	0.00%	0.10%	0.06%	0.83%
R2L	0.02%	0.21%	0.00%	0.00%	0.00%	0.23%
U2R	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%

We restricted our analysis to the 6 basic of the 41 total available features. By definition, a basic feature is "derived from packet headers without inspecting the content of the packet" [Kayacik et al., 2003]; in our case this also means that they are derived before the connection has actually completed. By this definition, there are 9 basic features:

1. Duration of the connection [continuous]
2. Protocol type (e.g. TCP, UDP or ICMP) [discrete]
3. Service type (e.g. FTP, HTTP, Telnet) [discrete]
4. Status flag (summarizes the connection) [discrete]
5. Total bytes sent to destination host [continuous]
6. Total bytes sent to source host [continuous]
7. Are the destination and source addresses the same? [discrete]
8. Number of wrong fragments [continuous]
9. Number of urgent packets [continuous]

The last 3 features are related to specific attack types, so we exclude those [Kayacik et al., 2003]. The remaining 32 (extrapolated) features are content features (require domain knowledge, e.g. # of unsuccessful logins), time-based features (mature over a 2 second temporal window), and host-based features (mature over a 100 count connection window).

We converted all the discrete features to a bit vector. Some discrete features (such as *service* and *attack*) had over 40 distinct values, so we compressed them to a smaller set (of up to 7 distinct values) before converting to a bit vector. The *attack* type was converted by grouping all its individual values to their high-level type [Hettich and Bay, 1999]:

- DoS (Denial of Service): attacker tries to prevent legitimate users from using a service by flooding the network
- Probe: attacker tries to gather information on the target host by scanning its available services
- R2L (Remote to Local): attacker does not have an account on the victim machine, hence tries to gain local access
- U2R (User to Root): attacker has local access to the victim machine and tries to gain super-user privileges

SECHAP Scaling

When training our multi-layer perceptron (MLP), the standard practice of scaling continuous features to the [0, 1] range ensures faster training and reduction in the probability of encountering a local minimum. This practice prevents the scale of the inputs from influencing convergence.

A cursory analysis of our three continuous features reveals extremely long tails in their distributions. Linear normalization (constant scaling) of the data to the [0, 1] range threatens to compress meaningful information to very small values. Although a logarithmic normalization may sufficiently evade the problem of compressed data, we seek a method of data normalization that by nature ensures a balanced representation of the each feature's distribution; consequently, we have implemented a novel approach to input data normalization employing the SECHAP (Streamed Equi-Cost Histogram APproximation) algorithm [Brönnimann and Vermorel, 2004].

Research in distribution approximation algorithms has largely responded to data streams in communications, networking, and database applications that exceed our current technology's ability to maintain accurate representations of the streams. Many applications in these fields seek a representation with bounded approximation error requiring concise storage and limited overhead processing. In particular, on-line (one-pass) models have offered extremely fast, small-space algorithms.

On-line representations, in particular, benefit normalization of data for our IDS' continuous features by allowing us to evaluate every single one of data points (over 490,000). This ensures that our representation does not cast away abnormal data solely based on their infrequency. Although KDD has sufficiently loaded its dataset with attacks, we would like to address the possibility of training a network on data sparsely populated with attacks. On-line algorithms promise exactly this in bounded time and space.

Approximation algorithms span a variety of representation types based on histograms, wavelets, and other more

abstract data-sketching devices. Of these representations, though, variable bucket-width histogram representations conveniently help solve our IDS data normalization problem. The buckets determined to best characterize the dataset can be used to divide the semi-infinite range of values for each feature into discrete segments that can then be normalized independently and equally represented over the [0, 1] range.

We have chosen the SECHAP algorithm over other variable bucket-width histogram approximation models [Gilbert et al., 2001; Guha et al. 2001] for its simplicity (ease of implementation), *O(B)* storage requirement, and *O(log(B))* processing time, where *B* is the number of buckets used in the histogram; increasing *B* improves the histogram's accuracy. For our dataset, we have employed 200 buckets for the three histograms characterizing each feature. Figure 3 shows an example of the resulting normalization for one of our features, the number of bytes sent to the source host.

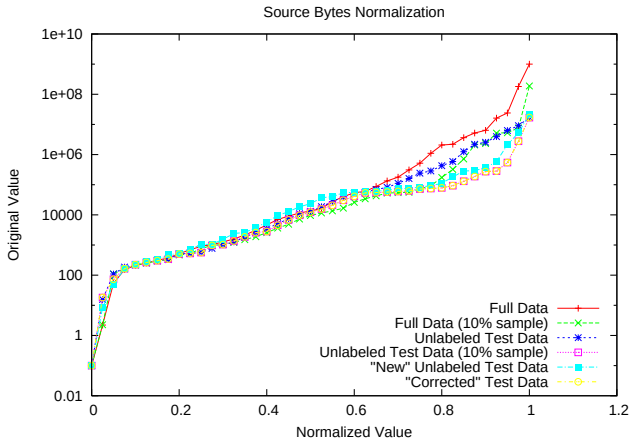

Figure 3: An Example of Continuous Data Normalizing

The SECHAP model systematically splits and merges buckets to ensure an approximate "sketch" of the distribution. Decisions to split and merge buckets seek to minimize the area of each bucket, which has shown to optimize the representation's ability to store information about the distribution.

This is precisely what we seek to accomplish in our normalization, an optimal representation of the distribution within the [0, 1] range. In Figure 3, we find that outlying data is well represented, but not at the expense of compressing more normal values.

Time Representation

Because attacks develop over time we would like to account for the chronology of event when training our ANN. We can represent time either explicitly (e.g. with a time stamp) or implicitly by presenting training examples to the network in their chronological order.

Existing research [Lichodzijewski, 2002], strongly suggests that implicit representation improves network performance over explicit representation, primarily due to high variance in the time dimension. Additionally, recurrent networks, such as the ones we used by nature capitalize particularly well on implicitly represented time-series data. Lastly, the KDD Cup dataset has been pre-sorted in chronological order.

Design of Experiments and Results

Experimentation with MLPs involves initially exploring and fixing certain rote parameters, such as:

- Number of Hidden Layers
- Number of Hidden Nodes
- Architecture (feed-forward vs. recurrent)
- Learning rate
- Momentum

Once we fixed these high-level parameters, we fine-tuned the remaining lower-level parameters:

- Leaky bucket rate
- Leaky bucket threshold

First Experiment

We ran an initial experiment to determine reasonable values for these parameters, based on a statistical significance test. We decided a-priori to use only a single hidden layer; many problem spaces are sufficiently addressed by a single hidden layer, as confirmed by current research papers.

Table 2: First Experiment Parameter Space

Parameter	Values
Topology	feed-forward, recurrent
Hidden Nodes	10, 20, 30
Learning Rate	0.1, 0.3, 0.5, 0.7
Momentum	0.3, 0.5, 0.7
Leak Rate	0.7
Leak Threshold	0.0

Because we suspect that the leak rate and threshold do not influence the *relative* accuracy when we vary the other parameters, we decided to fix the leak rate and the threshold to 0.7 and, respectively, 0.0, as seen in Table 2.

The experiment showed no statistically significant differences in detection rates across these parameters. Consequently, we chose to fix these parameters as follows, in order to explore in more detail other factors:

- Hidden Nodes = 20
- Learning Rate = 0.3

- Momentum = 0.7

The detailed results from this experiment are shown in Appendix B.

Second Experiment

In this second experiment we used the fixed values from the first experiment, and we focused on finding the leak rate and threshold that provide the best detection rate. We also looked for detection rate differences between the feed-forward and recurrent architectures.

Table 3: Second Experiment Parameter Space

Parameter	Values
Topology	feed-forward, recurrent
Hidden Nodes	20
Learning Rate	0.3
Momentum	0.7
Leak Rate	0.5, 0.75, 1.0
Leak Threshold	1.0, 1.5, 2.0

Averaging across all splits, both feed-forward and recurrent networks comparably detect normal activity (> 95%, Figure 4). However, the feed-forward network has *significantly* better attack detection rate compared to the recurrent network (75% vs., respectively, 8%, Figure 5). Note that the figures throughout this paper employ a code of "F" for feed forward networks and a code of "CH" for Elman implementations. As you can see in these figures, the standard deviation is relatively high, especially for the recurrent network architecture. This phenomenon can be explained two ways:

1. The leak rate and bucket threshold vary across splits, which as we will show, does affect performance.
2. Even if we fix the leak rate and bucket threshold, there is still inherent variance from run to run, all else being equal; this is especially prominent in the recurrent network.

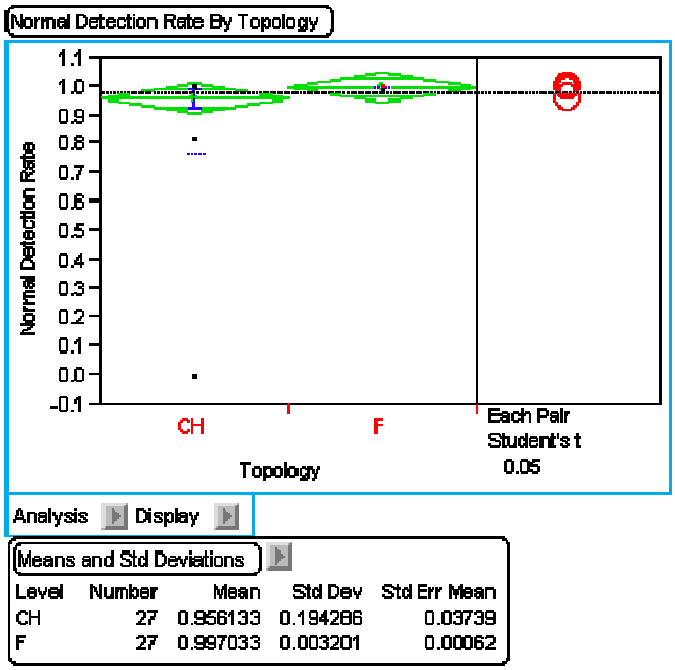

Figure 4: Normal Detection Rate

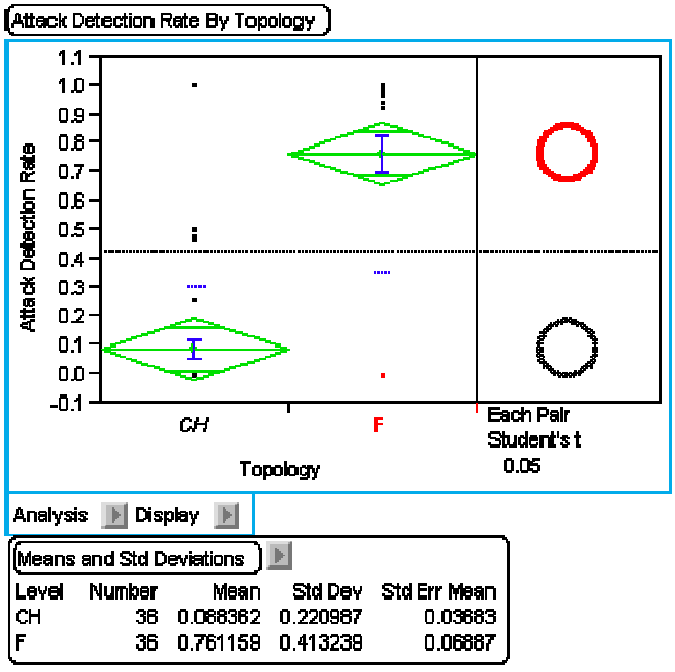

Figure 5: Attack Detection Rate

Our research concludes that the recurrent network performance is unstable and diverges as we train more (Figure 6). We believe this directly explains the poor performance of the recurrent network. Due to the high amount of training data, we were only able to run for 60 epochs; it is possible that if we were to run for more epochs, the recurrent network accuracy would converge, however we believe that there is significant redundancy in

the amount of data we used, and therefore the number of training epochs was appropriate. Given the feed-forward network's stellar performance, we felt that there is marginal value-add in continuing to pursue optimization of the recurrent topology.

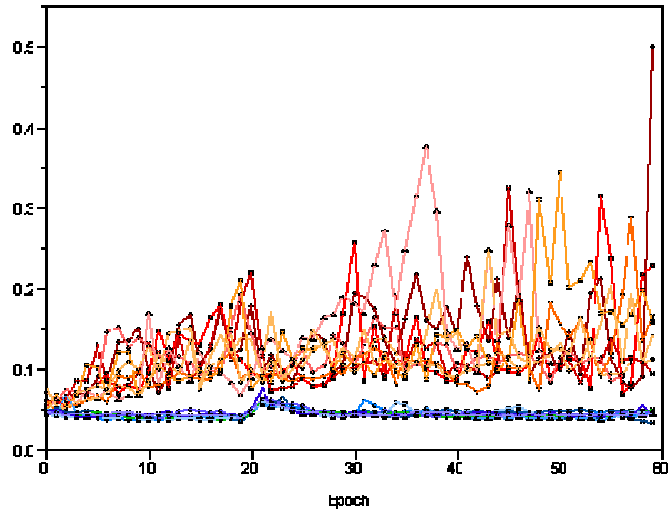

Figure 6: Root Mean Square Error across Epochs (blues are Feed-forward, reds are Recurrent)

Despite the failure of the recurrent network to train, we can still evaluate the effectiveness of the leaky bucket algorithm across both topologies.

In Figure 7, warm (red) colors indicate high accuracy, and cold (blue) colors indicate low accuracy. Note that the (1.0, 1.0) point on all these graphs effectively turns off the leaky bucket (the bucket has no "memory" across consecutive attacks). The recurrent network has a performance "sweet-spot" at (0.75, 1.5), for both normal and attack. The feed-forward network is not affected by the leaky bucket at all for the normal case, and for the attack case it must have a relatively low leak rate to prevent false negatives.

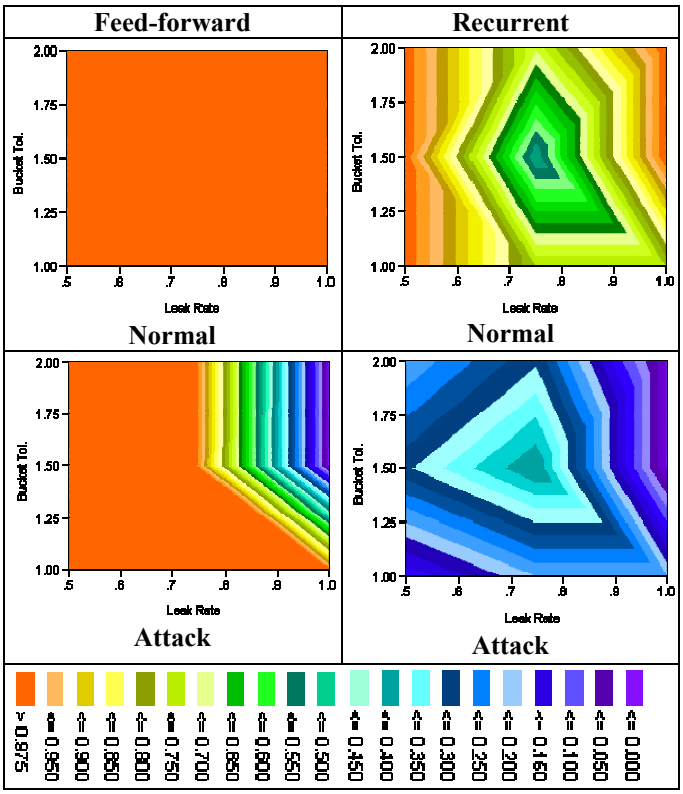

Figure 7: Performance Contour Maps

Discussion, Future Work

Our experimentation strongly asserts that Elman recurrent networks do not work well for the problem of classifying intrusion events, despite their theoretical ability to harness time-series information. In contrast to results in [Ghosh et al., 1999], the simple, feed-forward network achieves very good classification accuracy, both for *normal* and *attack* events (Figure 7), on par with the *recurrent* network from [Ghosh et al., 1999]. A corollary of this is that the leaky bucket algorithm works very well.

We suspect any of a number of reasons for the poor performance of our recurrent Elman network experiments:

1. JOONE's atypical context node implementation may have hindered convergence.
2. Due to the high volume of training data and time limitations, we were only able to train our networks for 60 epochs. It is possible that additional training may have helped the Elman recurrent network to converge.
3. The KDD Cup data may lack the chronology we assumed it to have. Specifically, a row in the KDD Cup data may represent a single connection (beginning to end), or may represent a fragment of a single connection. In the former interpretation, there is no chronology to this data, which means that training a recurrent network on it would likely overfit and

produce meaningless results. This might also be the reason why the feed-forward architecture performs so well.

For future work, we would like to migrate to a more stable, less buggy platform than JOONE and repeat some of the experiments. Additional experimentation would provide statistical significance to the results of this research.

In our training, we stopped iterating through additional epochs when the error delta between two consecutive epochs dropped below a certain threshold. This is a naive approach, and could stand to be improved by computing the error on the *test* set (instead of on the *train* set) and using it as a stopping criterion. This would likely prevent premature termination or overfitting.

Conclusion

Feed-forward networks coupled with a leaky bucket algorithm perform very well at classifying both normal and attack activity in the context of intrusion detection. These networks converge to a compellingly practical accuracy, and the additional work to setup, train, and use a recurrent network may not be justified.

Appendix A

This is the pseudo-code for the classification algorithm.

Description: classify the output from the ANN into one of the known attack meta-types (see the Dataset section), and update the statistics (true positive = T+, false positive = F+, false negative = F-, true negative = T-) based on the known output label

Input: *output* pattern, *known* pattern

Output: {T+, F+, F-, T-} counts

Pseudo-code:

1. Find the *closest* (L2 norm) pattern to *output*
2. If *closest* is *not* normal, add 1 to the bucket
3. If bucket level >= threshold, and *closest* is *not* normal, set *attack* flag
4. Decrement bucket by leaking rate
5. If *attack* flag
    a. If *known* is normal, increment F+
    b. Else increment T+
6. Else
    a. If *known* is normal, increment T-
    b. Else increment F-

Appendix B

These graphs show how we fixed the hidden nodes (**20**), learning rate (**0.3**), and momentum (**0.7**). There is no statistical significance (as measured by a paired t-test across the 5-fold run) between the accuracy of various topologies across these parameters. The statistical tool employed for graph generation groups statistically *insignificant* pairs by color around a bolded group, but only one at a time. Therefore, the graphs below feature this colored grouping for the feed-forward splits.

The *normal accuracy* is the percentage of normal events that were actually classified as normal (as opposed to attack); similarly the *attack accuracy* is the percentage of attack events that were actually classified as attack.

Note that the feed-forward attack detection rate in these graphs is artificially low because of a bug in our code. This bug did not affect the *relative* performance of these experiments.

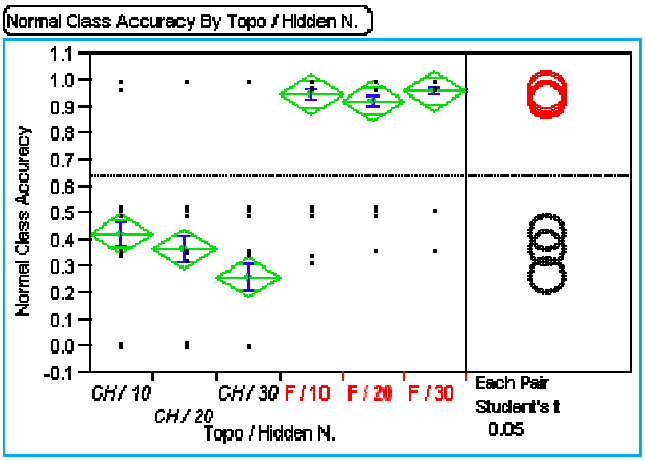

Figure 8: Normal Detection by Topology / Hidden Nodes

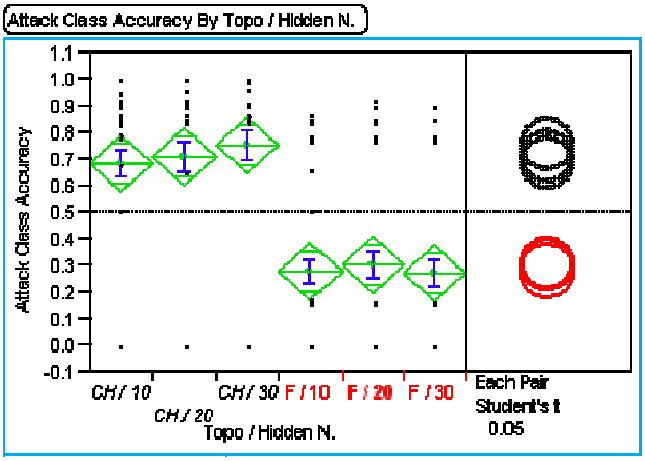

Figure 9: Attack Detection by Topology / Hidden Nodes

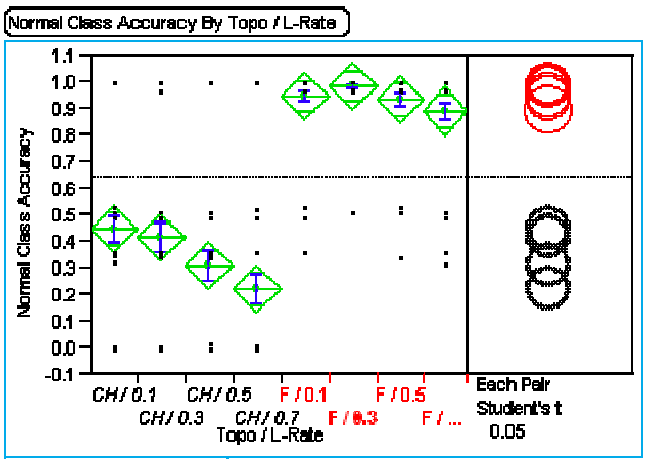

Figure 10: Normal Detection by Topology / Learn Rate

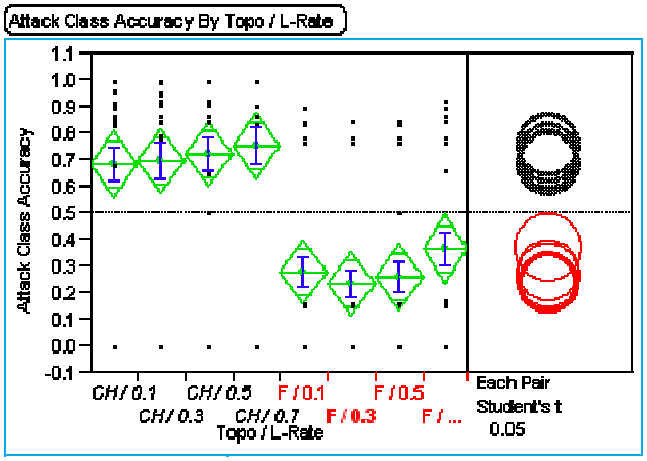

Figure 11: Attack Detection by Topology / Learn Rate

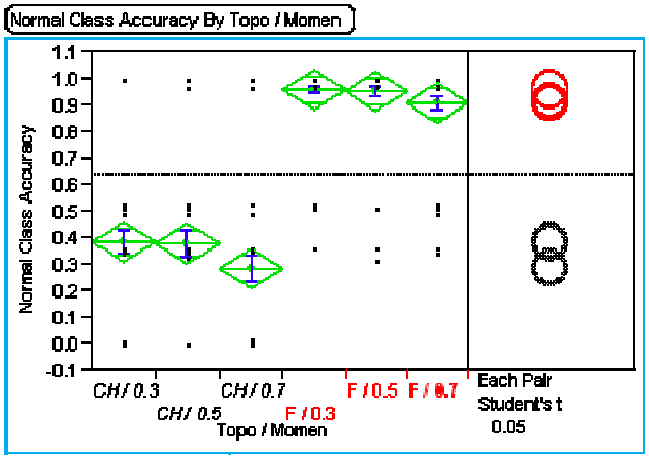

Figure 12: Normal Detection by Topology / Momentum

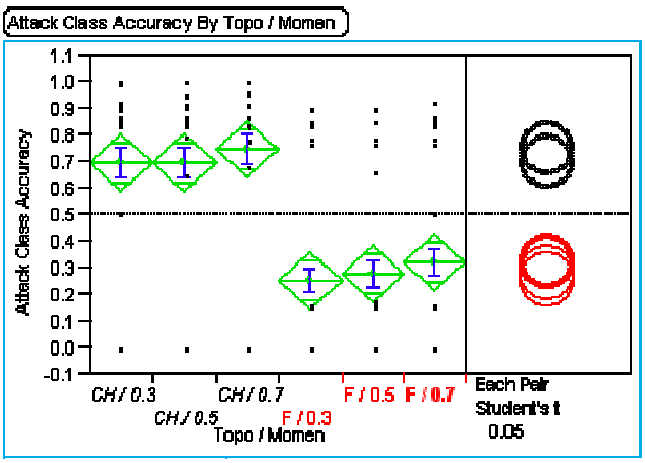

Figure 13: Attack Detection by Topology / Momentum

References

[Axelsson, 2000] Axelsson S., "Intrusion Detection Systems: A Survey and Taxonomy", Technical Report, Dept. of Computer Engineering, Chalmers University, March 2000

[Brönnimann and Vermorel, 2004] Brönnimann H., Vermorel J., "Streaming, Self-Scaling Histograms with Stability and Optimality Guarantees", New York 2004, to appear

[Elman, 1990] Elman J. L., "Finding Structure in Time", Cognitive Science, 1990

[Ghosh et al., 1999] Ghosh A. K., Schwartzbard A., Schatz M., "Learning Program Behavior Profiles for Intrusion Detection", Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 04, 1999

[Hettich and Bay, 1999] Hettich S., Bay S. D., "The UCI KDD Archive", Irvine, CA: University of California, Department of Information and Computer Science., 1999, http://kdd.ics.uci.edu

[Hoglund et al., 2000] Hoglund A. J., Hatonen K., Sorvari A. S. , "A Computer Host-Based User Anomaly Detection System Using The Self-Organizing Map", Proceedings of the International Conference on Neural Networks, IEEE IJCNN 2000, Vol. 5, pp. 411-416

[Jirapummin et al., 2002] Jirapummin C., Wattanapongsakorn N., Kanthamanon P., "Hybrid Neural Networks for Intrusion Detection System", 2002 International Technical Conference On Circuits/Systems, Computers and Communications

[Kayacik et al., 2003] Kayacik G., Zincir-Heywood N., Heywood M., "On the Capability of an SOM-based

Intrusion Detection System", Proceedings of International Joint Conference on Neural Networks, 2003

[Lichodzijewski, 2002] Lichodzijewski P., "Host Based Intrusion Detection Using Self-Organizing Maps, IEEE International Joint Conference on Neural Networks, May 2002

[Marrone et al., 2004] Marrone P., Joone Team, "Joone - Java Object Oriented Neural Engine", 2004, http://www.jooneworld.com/

[MIT, 1998] MIT Lincoln Laboratory Information Systems Technology Group, "The 1998 Intrusion Detection Off-line Evaluation Plan", March 1998, http://www.ll.mit.edu/IST/ideval/docs/1998/id98-eval-ll.txt