Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data

Conference Paper · October 2013

DOI: 10.1145/2513456.2513490

2 authors:

Ralf C. Staudemeyer
University of Applied Sciences Schmalkalden
34 PUBLICATIONS **414** CITATIONS
SEE PROFILE

Christian Omlin
University of South Africa
30 PUBLICATIONS **973** CITATIONS
SEE PROFILE

Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data

Ralf C. Staudemeyer
Department of Computer Science
Rhodes University
Grahamstown, SOUTH AFRICA
r.staudemeyer@ru.ac.za

Christian W. Omlin
School of Computer Science
University of the Witwatersrand
Johannesburg, SOUTH AFRICA
christian.omlin@wits.ac.za

ABSTRACT

This paper evaluates the performance of long short-term memory recurrent neural networks (LSTM-RNN) on classifying intrusion detection data. LSTM networks can learn memory and can therefore model data as a time series. LSTM is trained and tested on a processed version of the KDDCup99 dataset. A variety of suitable performance measures are discussed and applied. Our LSTM network structure and parameters are experimentally obtained within a series of experiments presented. Results finally show that LSTM is able to learn all attack classes hidden in the training data. Furthermore we learn that the receiver operating characteristic (ROC) curve and the corresponding area-under-the-curve (AUC) value are well suited for selecting well performing networks.

Keywords

long short-term memory, recurrent neural networks, KDDCup99, intrusion detection systems, machine learning, time series analysis, receiver operating characteristic.

1. INTRODUCTION

It is a challenge to build intrusion detection systems based on artificial intelligence. The purpose of this work is to apply the long short-term memory recurrent neural network (LSTM-RNN) classifier to intrusion detection data. We focus on the evaluation of different performance measures suitable to quickly filter well performing networks, when browsing through large numbers of potential candidates.

After outlining LSTM we observe a number of suitable performance measures and compare them to ROC-analysis. Then we run a number of experiments on a pre-processed version of the KDDCup99 dataset, which consists of connection records with 41 features. After finding a suitable neural network parameters and structure, we run a performance analysis. Confusion matrix, accuracy, ROC-curve and AUC-value are used as performance measures.

2. RELATED WORK

Long Short-Time Memory Recurrent Neural Networks (LSTM) developed by [7] and extended by [6] are a special class of Recurrent Neural Networks (RNN) with extended memory learning capabilities. RNNs are dynamic systems with an internal state at each time step of the classification. This is due to circular connections between higher- and lower-layer neurons and optional self-feedback connections. These feedback connections enable RNNs to propagate data from earlier events to current processing steps. Thus, RNNs build a memory of time series events.

This type of neural network ranges from partly to fully connected, and early RNNs were suggested by [8] and [4]. They need to be trained differently to feed-forward neural networks (FFNNs) for reflecting the recurrent connections. The most common and well-documented learning algorithms for training RNNs in temporal, supervised learning tasks are backpropagation through time (BPTT) (see [25], [21] and [20]) and real-time recurrent learning (RTRL) (see [26] and [20]).

An overview of common performance metrics for machine learning in intrusion detection is provided by [13]. A detailed introduction of ROC-analysis can be found in [5].

The choice of the available labelled intrusion detection data is very limited. The most comprehensive and well documented dataset is the KDDCup99-dataset extracted from the DARPA datasets (see [10] and [11]). The data was processed as described by [9]. The shortcomings of this dataset are well know and documented by [14], [12], [23] and [24].

Various static machine learning algorithms have previously been evaluated on the KDDCup99 dataset. The winning entries [3] of the challenge were all variants of the C5 decision tree algorithm [19]. After the challenge a comprehensive set of other algorithms was tested on the data, mostly with comparable results (see [22], [16], [1] and [17]).

3. LONG SHORT-TERM MEMORY

Here we outline Long Short-Term Memory (LSTM) recurrent neural networks (see [7] and [6]), a powerful dynamic classifier. Later, we compare the performance of this classifier on intrusion detection data.

LSTM uses special memory units, called memory cells, containing Constant Error Carousels (CEC), which enforce a constant error flow. Access to the cells is handled by multiplicative gate units, which learn when to grant access.

In a LSTM network all units in the hidden layer are replaced with so-called memory blocks. Each memory

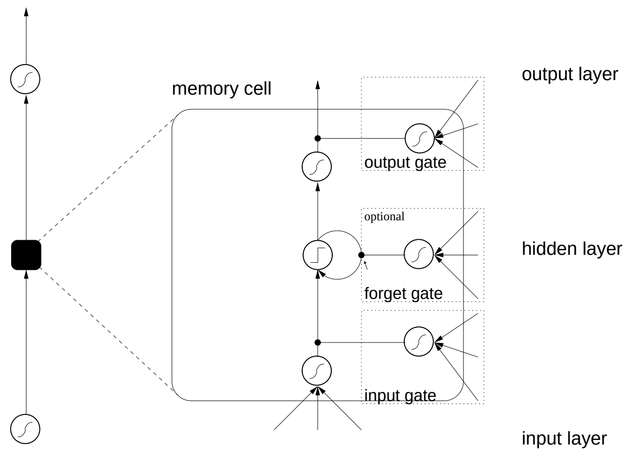

Figure 1: Simple LSTM network.

block contains at least one memory cell with regulating gates controlling incoming and outgoing information flow. LSTM neural networks can be equipped with several memory blocks. Figure 1 shows a simple LSTM network with one memory block.

The input of the memory cell passes from the previous layer through a non-linear squashing function being scaled to lie within $[-2, 2]$. It is then multiplied by the result of the input gate with ranges between $[0, 1]$. Thus the signal will only be able to pass if the signal from the input gate is sufficiently close to one. In a similar way the output of the memory cell is controlled by a non-linear squashing function that ranges between $[-1, 1]$ and multiplied by the result of output gate that ranges as well between $[0, 1]$.

The CEC is a simple linear unit with a self-recurrent connection. The self-connection has a fixed weight set to one to preserve the state of cell over time. Alternatively, a forget gate can be attached to the self-connection. Forget gates can learn to reset the state of the linear unit when the stored information is no longer needed.

The gates are responsible for deciding what information the central unit stores and when to apply that information. They are simple sigmoid threshold units with an activation function ranging over $[0, 1]$. The input for the gates comes from the network input layer and from other memory cells.

The output $y^{c_j}(t)$ of a memory cell is computed in the following way:

$$y^{c_j}(t) = y^{out_j}(t)(s_{c_j}(t)$$

where $y^{out_j}(t)$ is the activation and $s_{c_j}(t)$ is the state of the output gate. The state is given by the following recursive definition:

$$s_{c_j}(0) = 0$$

and

$$s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t)\ g(net_{c_j}(t))\ for\ t > 1$$

where $g(x)$ is an optional non-liniar squashing function for the cell input.

LSTM neural networks can be trained with a combination of the backpropagation through time (BPTT) and another RNN learning algorithm called real-time recurrent learning (RTRL).

Backpropagation through time (BPTT) and other RNN learning algorithms cannot look more than 10 to 12 time steps into the past. Errors tend to either not be considered at all or the learning algorithm places too much weight on them. LSTM is a RNN architecture designed for supervised time-series learning that overcomes this problem. A LSTM network can keep information over very long periods of time with at least 1,000 time steps. For these reasons LSTM seems to be well suited for learning from time series data.

pred. → **actual** ↓	**1** positive	**2** negative
1 positive	a	b
2 negative	c	d

Table 1: A confusion matrix for a binary problem.

4. MEASURES OF PERFORMANCE

For a meaningful performance comparison of different classifiers it is necessary to at least agree on the data, and what performance metrics are applied. Models build by the classifier from the data are an approximation of the true model. To evaluate these built models we need to divide the available data into training and testing data. The training data is used to build the model, and the test data to evaluate it. Given that the labels of the test data are known we can apply the following performance metrics.

4.1 Mean Squared Error

For numeric prediction tasks the 'mean squared error' (MSE) is a common method. MSE quantifies the amount by which an estimator differs from the targeted value. The MSE of a dataset is the average of the sum of all sqared errors of each pattern. Given n patterns of the dataset, for the ith example, let p_i be the predicted value and a_i the actual value. The MSE of the tested dataset is

$$MSE = \frac{\sum_{i=0}^{n}(a_i - p_i)^2}{n}$$

4.2 Confusion Matrix

For two-class problems, the result of a classification can either be predicted correctly or incorrectly. This yields four different conditions:

- (a) True Positive - model correctly predicts positive
- (b) False Negative (type II error) - model incorrectly predicts negative
- (c) False Positive (type I error) - model incorrectly predicts positive
- (d) True Negative - model correctly predicts negative

A confusion matrix shows the predicted and the actual classifications. The size of a confusion matrix is $n \times n$, where n is the number of different classes. The confusion matrix shown in Table 1 is for a two-class problem.

4.3 Performance Measures Derived

From the counts of these four conditions we can calculate the following simple performance measures:

- true positive rate (or detect rate) – portion of positive instances correctly predicted positive
  $a/(a + b)$

- false negative rate – portion of positive instances wrongly predicted negative
  $b/(a+b)$
- false positive rate – portion of negative instances wrongly predicted positive
  $c/(c+d)$
- true negative rate – portion of negative instances correctly predicted negative
  $d/(c+d)$
- precision – probability an instance gets correctly classified
  $a/(a+c)$
- accuracy – proportion of test results the model predicts correctly
  $(a+d)/(a+b+c+d)$

Accuracy and mean squared error are the most common performance measures; other performance metrics can include the time an agorithm needs to build a model from a dataset and/or the time to apply it on a dataset.

Simple performance measures like accuracy or error rate are problematic. Accuracy, for instance, does not provide information on the performance per class. Missing a positive or missing a negative is treated the same. If the majority of examples in a dataset are negative than a high accurancy might only be due to the exceptional performance on these negative examples. Observing the true positive rate may indicate that the performance on the positive examples is very poor.

It is necessary to satisfy certain conditions to apply these simple performance measures. These include an equal number of examples in each class. For highly skewed data where one class is much larger than the other these metrics are not very meaningfull.

4.4 ROC Analysis

The Receiver Operating Characteristic (ROC) analysis evaluates an algorithm over a range of possible operating scenarios. The ROC-graph is a two-dimensional plot of the false-positive rate (x-axis) of a model against its true-positive rate (y-axis). A true-positive rate of unity and false-positive rate of zero are indicators for perfect performance. The lower left point (0,0) in the graph represents a model with no false positive errors but also no true positives. This model would always classify negative but never positive. The opposite point at the upper right (1,1) represents a model that always classifies positive and never negative. The point at the upper left (0,1) on the ROC-graph represents a model that always classifies correctly.

In two-type classification discrete classifiers generate as a result a one class decision for every instance of a testset; and all classified instances yield to one confusion matrix. Each matrix has exactly one true positive rate and one false-positive rate. These two produce a single point on the ROC-graph.

This is in contrast to the result of probabilistic classifiers, like neural networks, which produce a numeric value. The value represents the probability that the observed instance is a member of a specific class; where a higher value indicates a higher probability. A decision threshold of e.g. 0.5 is used to produce the decision. If the value is above the threshold the instance belongs to a specific class. A value under the threshold is classified as noise. Every threshold applied produces its own confusion matrix and a different point on the ROC-graph.

To generate a ROC-curve the threshold is varied from $-\infty$ to $+\infty$, or in case of neural networks with target values in the range $[0,1]$ from 0 to 1. The resulting ROC-curve of a successfully learned classifier should look like an inverted 'L' with the corner pushing toward the upper left of the graph. Results similar to random guessing yield a diagonal line between $[0,0]$ and $[1,1]$.

One powerful strength of ROC-analysis is that it is independent from class distribution. The curve remains the same if proportion of positive and negative examples changes.

The Area Under the Curve (AUC) summarizes the ROC curve in a single value as a measure for expected performance. The AUC-value of a classifier is equal to the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance. It is a value between 0 and 1. No meaningful classifier should have a AUC-value below [0.5]. Depending on the shape of the ROC-curves, a high AUC value of one classifier can perform worse in a specific region of the curve than a low AUC value of another classifier. But in practise the AUC value performs very well.

5. THE DATA

These experiments use the well-known datasets ([24]) from the 1999 KDDCup contest at the fifth International Conference on Knowledge Discovery and Data Mining. The learning task of this competition was to classify preprocessed connection records to either normal traffic, or one out of four given attack categories:

- denial of service (DoS)
  Denial of service attacks try to exhaust network, computing or memory resources of a target system.
- network probes
  This kind of activity aims to gather information of a target network and/or computer (e.g. IP- and port-scanning activities).
- remote-to-local attacks (r2l)
  First class of exploits which target network services (like web-services) to gain user access to a target system.
- user-to-root (u2r)
  In the second class of exploits a user tries to execute commands with superuser privileges. Various buffer overflow attacks fall into this class.

Seven weeks of network traffic collected in four gigabytes of compressed raw tcpdump files were preprocessed into five million labelled and categorized connection records with approximately 100 bytes each; and two weeks of training data were processed into three million unlabelled connections records (see [9]).

The KDDCup99 competition provides the disjoint training and testing datasets in a full and a so called '10%' subset version. The '10%' subset was created due to the huge volume of connection records in the full set; in particular denial-of-service attacks have millions of records. For this reason not all of these connection records were selected.

To achieve approximately the same distribution of intrusions and normal traffic as the original DARPA dataset a selected set of sequences with 'normal' connections was left in the 10% dataset as well; although the training and test sets are of different probability distributions.

The connection records summarise the packets of one communication session between a connection initiator with a specified source IP address and a destination IP address over a pair of TCP/UDP ports. The labeled connection records in the training set are categorized normal or indicate one of 22 types of attacks. Although more then ten years old, the KDDCup99 dataset is the most well known and fully labelled intrusion detection dataset publically available today. For this reason it is well suited for classifier performance evaluation.

6. EXPERIMENTS

First experiments were run with different parameters and structures of an LSTM neural network; such as the number of memory blocks and the cells per memory block, the learning rate and the number of passes through data. Succeeding experiments were run with a layer of hidden neurons, peephole connections, forget gates and LSTM shortcuts; all being extensions of LSTM as documented in [6].

6.1 Network Parameters

Experiments were started with a basic LSTM network using 43 input neurons, two memory blocks with two cells each, peephole connections, and five target neurons. The input neurons were fully connected to the hidden layer with the two memory blocks. The number of iterations was fixed to 50 training cycles (epochs). The 10% training dataset was used for training and the 10% corrected dataset was used for testing. All values of the input features were preprocessed to the range [-1,1], including numeric and nominal features.

Each nominal target value was mapped to one of the five connection classes, each represented by a target neuron with an binary value. The target values were tested in the order 'normal', 'dos', 'probe', 'u2r' and 'r2l', to minimize the number of false positives. The traffic was classified according to the first value larger than the decision threshold. The decision threshold was set to [0.5]. Classified cases where no output was larger than the decision threshold were set as 'normal' per default.

The performance of the learned networks was evaluated by observation of the confusion matrix and by calculating the accuracy. To find a suitable learning rate for the datasets, it was varied in the interval $[0.01-0.5]$; no weight decay was used.

In a second step different 'pure' feed-forward networks and hybrid-networks including hidden neurons and LSTM memory blocks were compared. The number of hidden layers was fixed to one in all experiments. Improvements with more than one hidden layer were not expected. For both types of networks experiments were run with 5, 10, 15, 20, 32, 43 and 86 hidden neurons. Each experiment consisted of eight trials. The best performing result was selected.

Finally, consecutively forget gates (no manual reset), peephole connections and shortcuts to the basic LSTM network were added to assess their impact to learning performance.

Experiments with lower learning rates showed a slightly better classification performance. Naturally for a low learning rate the expected number of required iterations for low frequency attacks are very large ($> 10000 Epochs$). As a trade-off between training time and classification performance the learning rate was set to 0.1 for all following experiments.

All LSTM-hybrid-networks showed good performance in terms of accuracy. Learning of the hybrid-networks was faster as well. The best results were achieved using one feed-forward layer with 20 hidden neurons. All trained networks achieved 'good' results with an accuracy $> 90\%$, but they were less accurate than results possible with a standard LSTM. The generalization performance of the hybrid-network seems to be weakened in comparison to a LSTM network without a hidden layer. Furthermore the detection rate on rare and 'difficult-to-learn r2l and u2r' attacks decreased.

Experiments with forget gates, peephole connections and shortcuts yielded an average to better classification. In all futher experiments forget gates, peephole connections and shortcuts were used.

6.2 Network Structure

To find a suitable network structure for training the KDDCup99 data, experiments were run with LSTM networks using the following four topologies:

- Two memory blocks with two cells each
- Four memory blocks with two cells each
- Four memory blocks with four cells each
- Eight memory blocks with four cells each

All networks used forget gates, peephole connections and shortcuts. The learning rate was fixed to 0.1 and the decision threshold was set to 0.5. Traffic classification was according to the first value larger than the threshold in the order 'normal', 'dos', 'probe', 'r2l' and 'u2r'. Default classification was 'normal'. The preprocessed '10% training' and the '10% corrected' testset datasets with all features were used.

To find the minimum number of required iterations the training data was presented for five to up to one thousand epochs to each of the four observed network structures. We run eight trials of each network setup.

Results with good accuracy for attack detection (attack/normal two-class categorization) at reasonable cost was reached at 60-150 epochs. The lowest standard error was achieved after little more than 500 epochs for all four network topologies. More complex LSTM networks needed more iterations to get acceptable results but finally attained a higher accuracy as well.

For each of the five attack classes the LSTM network requires a different number of optimal iterations. After learning a specific traffic type the network starts overfitting. From that point the network improves in memorizing the training data and the generalization performance decreases for that specific traffic type.

After 25-90 epochs most networks learned DoS attacks and after 50-125 epochs probe attacks are learned. The rare attack categories r2l and u2r need many more presentations of the training data. Attacks of the class r2l need 200-1000 epochs and u2r attacks need 125-1000 epochs until they are learned as well as possible. Rare attacks might require more than 1000 presentations of the training data.

After 500-600 epochs approximately 50% of the trials get results with an low error rate. The performance of all networks decreases after further training. In comparison to standard neural networks with a hidden layer, LSTM is more prone to over-fitting.

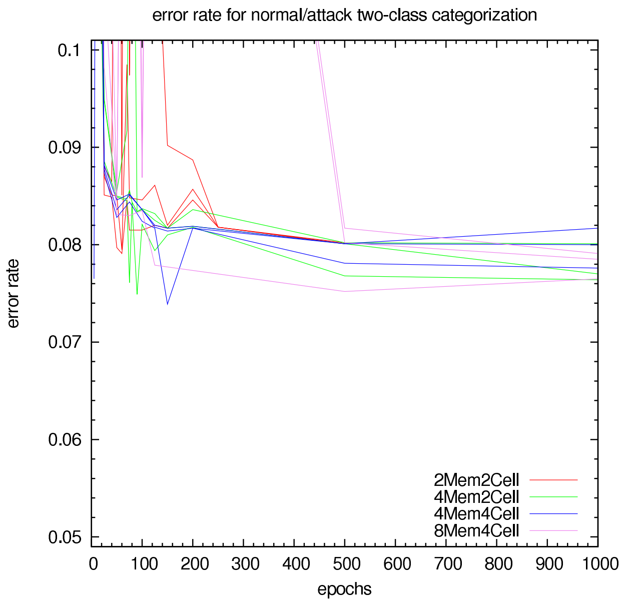

Figure 2: Standard error (1-accuracy) development for two-class categorization of best three trials of the four different network structures in dependency of epochs trained.

Out of the eight trials of each experiment three showed high accuracy for normal/attack two-class categorization. The development of the standard error rate is plotted in Figure 2.

With increasing size of the LSTM network learning requires more presentations of the training data. Rare 'r2l' attacks require more than 1000 epochs on the two larger networks. On the other hand the small LSTM network with two memory blocks has problems to learn the very rare and difficult to learn 'u2r' attacks.

We think that networks with four memory blocks each containing two cells offer a good compromise between calculation cost and detection performance. This type of network structure was used by us for our next experiments.

6.3 Performance Analysis

As learned from previous experiments, LSTM networks with four memory blocks containing two memory cells each were built. Forget gates, peephole connections, shortcuts and a fixed learning rate of 0.1 were used.

For these experiments ROC-curves were added as a performance measure. As required for ROC analysis no classification threshold was applied. All output values of all five target neurons for every tested pattern was recorded. At the end of every trial a full 5×5 confusion matrix was generated, accuracy for the resulting network was calculated, and for every target neuron detection rate, precision and the AUC-value were calculated. The target output with the highest numerical value was used for traffic classification.

For the ROC calculations we used *proproc* [18] to estimate the curve from the case ratings. For curve estimation we chose an non-parametic estimate [2]. The *proproc* software and support was kindly provided without cost on request by the developers [15].

For training the '10%' training dataset was used. The performance was tested on the training set and as well on the '10% corrected' testset. The networks were trained for up to 1000 epochs. The performance of the trained network was measured at 25, 50, 75, 90, 100, 125, 150, 175, 200, 250, 300, 400, 500, 600, 750 and 1000 epochs. Every experiment contained 30 trials.

Since multi-class ROC graphs are not plotable we examined the five target neurons of every trained LSTM network separately. Every target neuron represents one traffic class. We generated our own ROC graph and calculated the AUC value for each of the five traffic classes: 'normal', 'dos', 'probe', 'r2l' und 'u2r'.

Figure 3 shows the ROC-plots of the highest AUC value achieved for each attack traffic class. For reasons of comparison all 30 trials are plotted. We calculated detect rate and precision of each neuron as well. To support performance comparison the plots are flagged with different line types according to their detect rate.

The ROC curves show that the network actually learned at least parts of all five traffic classes. The bumpy ROC curves are expected since we present classes of traffic ('normal', 'dos', 'probe', 'r2l' and 'u2r') with every class contains subclasses (e.g. different attacks). During the learning process, distinguishable subclasses do appear as bumps in the ROC graph.

The Table 2 shows two corresponding confusion matrices for dos attacks and network probes with an exceptional good performance.

7. CONCLUSIONS

These experiments evaluate the performance of LSTM recurrent neural networks applied to intrusion detection data. Selected experimental parameters on a number of chosen network topologies are discussed. Performance is measured in terms of a confusion matrix, accuracy, ROC-curve and the AUC-value. Our results show that ROC-curves are well suited for selecting well performing networks.

Trained LSTM-networks learned all five traffic classes at least partially. The deviation of the ROC-curves from a straight diagonal between the coordinates [0,0] and [1,0] to a curve bowing into the corner [1,0] is recognisable in all graphs.

In most well trained networks the target neurons representing normal traffic and network probes showed an excellent performance. The target neuron representing DoS attacks even achieves close to perfect discrimination between attacks and other traffic. Even the ROC-Curves of the remaining two most difficult to learn attack classes 'r2l' and 'u2r' show that LSTM is in fact able to learn these, although it might require further training or different features to be added to the data.

In future work, we will apply LSTM to more recent intrusion detection datasets and evaluate the performance of complex LSTM neural network structures on network traffic metadata.

8. REFERENCES

[1] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptive neuro-fuzzy intrusion detection systems. In *International Conference on Information Technology: Coding and Computing*, volume 1, pages 70–74 Vol.1. IEEE, 2004.

[2] E. R. DeLong, D. M. DeLong, and D. L. Clarke-Pearson. Comparing the Areas under Two or More Correlated Receiver Operating Characteristic

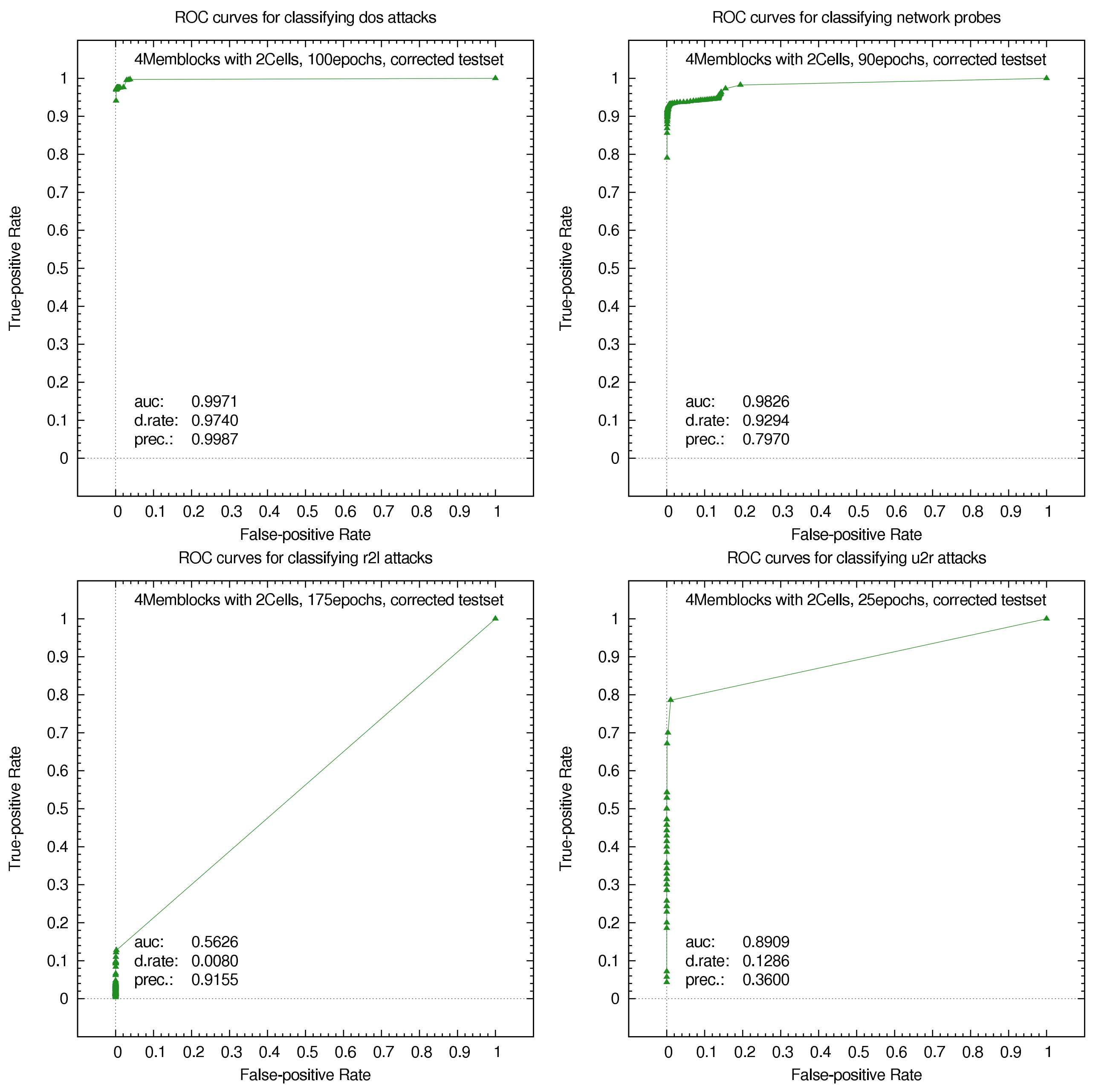

Figure 3: ROC curves for the target neurons representing the four atttack traffic classes

Curves: A Nonparametric Approach. *Biometrics*, 44(3):837, Sept. 1988.

[3] C. Elkan. Results of the KDD'99 classifier learning. *ACM SIGKDD Explorations Newsletter*, 1(2):63, Jan. 2000.

[4] J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, Mar. 1990.

[5] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006.

[6] F. A. F. A. Gers, J. J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. Technical Report IDSIA-01-99, IDSIA, Lugano, Lugano, CH, Oct. 1999.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. Technical Report 8, Technische Universität Muenchen, 1997.

[8] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequentialmachine. In *Proceedings of the Eigth Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

[9] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, Nov. 2000.

[10] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, Oct. 2000.

[11] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham,

prediction→	**1**	**2**	**3**	**4**	**5**	**%correct**
actual↓	u2r	r2l	pro	dos	nor	detect rate
u2r	2	0	0	1	67	2.86
r2l	0	10	124	6	16205	0.06
pro	3	0	3584	215	364	86.03
dos	0	0	113	223876	5864	97.40
nor	1	0	229	74	60282	99.50
%correct						**accuracy**
precision	33.33	1.00	88.49	99.87	72.82	92.67

prediction→	**1**	**2**	**3**	**4**	**5**	**%correct**
actual↓	u2r	r2l	pro	dos	nor	detect rate
u2r	19	2	5	5	39	27.14
r2l	4	1312	173	11	14845	8.03
pro	0	0	3872	251	43	92.94
dos	0	3	212	224155	5483	97.52
nor	88	69	596	916	58917	97.25
%correct						**accuracy**
precision	17.12	94.66	79.70	99.48	74.27	92.90

Table 2: confusion matrixes for two performing networks in terms of DoS attacks and network probes

M. Zissman, and Others. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26. IEEE, IEEE Comput. Soc, 2000.

[12] M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. *Recent Advances in Intrusion Detection*, 2820(Ll):220–237, 2003.

[13] M. A. Maloof. Some basic concepts of machine learning and data mining. In *Machine Learning and Data Mining for Computer Security*, Advanced Information and Knowledge Processing, pages 23–43. Springer London, 2006.

[14] J. McHugh. Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, Nov. 2000.

[15] C. E. Metz, Y. Jiang, H. MacMahon, R. M. Nishikawa, and X. Ran. ROC software. http://metz-roc.uchicago.edu/. Accessed: 2013-08-27.

[16] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, pages 1702–1707. IEEE, 2002.

[17] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114–132, Jan. 2007.

[18] L. L. Pesce and C. E. Metz. Reliable and computationally efficient maximum-likelihood estimation of "proper" binormal ROC curves. *Academic radiology*, 14(7):814–29, July 2007.

[19] J. R. J. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., Mar. 1993.

[20] D. Z. Ronald J. Williams, R. J. R. Williams, and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Back-propagation: Theory, Architectures and Applications*, pages 1–45. L. Erlbaum Associates Inc., Jan. 1995.

[21] D. E. Rumelhart, G. E. Hinton, R. J. Williams, D. Rummelhart, and W. R.J. Learning Internal Representations by Error Propagation. In J. L. McClelland and D. E. Rumelhart, editors, *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1, pages 318–362. MIT Press, Jan. 1986.

[22] M. Sabhnani and G. Serpen. Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context. In *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications*, pages 209–215. CSREA Press, 2003.

[23] M. Sabhnani and G. Serpen. Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set. *Intelligent Data Analysis*, 6(2002):1–13, Sept. 2004.

[24] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, number Cisda, pages 1–6. IEEE, July 2009.

[25] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[26] R. J. Williams and D. Zipser. Experimental Analysis of the Real-time Recurrent Learning Algorithm. *Connection Science*, 1(1):87–111, Jan. 1989.