Applying long short-term memory recurrent neural networks to intrusion detection

Ralf C. Staudemeyer

School of Computing, University of South Africa, Johannesburg, South Africa

ABSTRACT

We claim that modelling network traffic as a time series with a supervised learning approach, using known genuine and malicious behaviour, improves intrusion detection. To substantiate this, we trained long short-term memory (LSTM) recurrent neural networks with the training data provided by the DARPA / KDD Cup '99 challenge.
To identify suitable LSTM-RNN network parameters and structure we experimented with various network topologies. We found networks with four memory blocks containing two cells each offer a good compromise between computational cost and detection performance. We applied forget gates and shortcut connections respectively. A learning rate of 0.1 and up to 1,000 epochs showed good results.
We tested the performance on all features and on extracted minimal feature sets respectively. We evaluated different feature sets for the detection of all attacks within one network and also to train networks specialised on individual attack classes. Our results show that the LSTM classifier provides superior performance in comparison to results previously published results of strong static classifiers. With 93.82% accuracy and 22.13 cost, LSTM outperforms the winning entries of the KDD Cup '99 challenge by far. This is due to the fact that LSTM learns to look back in time and correlate consecutive connection records. For the first time ever, we have demonstrated the usefulness of LSTM networks to intrusion detection.

KEYWORDS: long short-term memory, recurrent neural network, KDD Cup '99, intrusion detection systems, machine learning, time series analysis, receiver operating characteristic.

CATEGORIES:
Computing methodologies → Neural networks
Computing methodologies → Machine learning
Computing methodologies → Supervised learning
Security and privacy → Intrusion detection systems
Security and privacy → Network security
Mathematics of computing → Time series analysis

ARTICLE HISTORY

Received 7 April 2014
Accepted 9 May 2015

1 INTRODUCTION

In modern society, increasingly powerful technologies have encouraged widespread dependency on *information and communication technology* (ICT), which, in turn, has created a strong requirement for dependable ICT functionalities. At the same time, however, there are increasingly sophisticated and diverse threats to modern ICT systems; this calls for novel security mechanisms. Intrusion detection aims at identifying various kinds of (malicious) activities, and is now a strategic task of the highest importance in safeguarding computer networks and systems. While traditional approaches to *intrusion detection systems* (IDSs) have proven to be efficient at detecting intrusions based on well-known parameters, they are completely ineffective in cases involving novel intrusions.

The analysis of collected data is either static or heuristic. When an IDS uses filters and signatures to describe attack patterns, the analysis is static; this is called *signature detection* (or misuse detection). Signature detection is limited to the detection of known attack patterns. For the detection of unknown attacks, heuristic methods must be used. Systems that use these methods offer the possibility of detecting patterns that are not 'normal'; these detection methods are termed *anomaly detection.* A third very common approach for extending static detection methods is called *stateful protocol analysis.* Here models of proto-

Email: Ralf C. Staudemeyer `staudrc@unisa.ac.za`

cols and applications are compared to their observed behaviour. Most systems combine static and heuristic methods in an hybrid approach.

Current commercial products offering anomaly detection are solely threshold-based, or make use of statistical measures [1]. These methods can model only relatively simple patterns, expressed in counts or distributions. Similar to signature-based approaches, this still limits their application to the detection of well-known and precisely defined attacks.

One potential solution to this limitation could be self-learning systems which are capable of detecting previously unknown threats. This is due to their ability to differentiate between 'normal' and 'anomalous' traffic by learning from monitored-network and host data. More precisely, machine learning methods can learn complex system behaviour. By learning whole classes of normal traffic and attacks, trained classifiers have the potential to detect irregularities and previously unseen attacks. In addition, machine learning methods promise to provide a solution that can detect possible attacks in real time, so that countermeasures can be taken in a timely manner.

At this time, since the implementation of machine learning methods in intrusion detection is in the very early stages of development, its practical applications are still quite limited [2]. For instance, trained classifiers still suffer from a high number of misclassifications because intrusive activity is too rare [3]. Furthermore, powerful classifiers require significant resources for training and optimisation [4], which is still unrealistic for commercial deployment. This is why feature selection is another key element in advancing the use of machine learning methods in intrusion detection. These and other significant issues are summarised in [5].

Here, we investigate the application to network intrusion detection of *long short-term memory* (LSTM) as introduced by [6], and enhanced by [7] and [8]. In investigating the possibilities of this powerful *dynamic* classifier, we were especially interested in the effects of a reduced feature set on the network intrusion detection performance.

We applied LSTM to the publicly available DARPA[1] / KDD Cup '99 dataset[2]. The KDD Cup '99 dataset consists of connection records with 41 features whose relevance for intrusion detection are not clear. This work documents experiments with different subsets of these features. The experiments are of a general nature and can be applied to any similar (e.g. more recent) dataset. The KDD Cup '99 dataset was mainly chosen for reasons of comparison with other machine learning algorithms.

This paper is structured in the following manner. After referencing a selection of related work, we will delve into standard recurrent neural networks. We point out one of their major limitations and how it is resolved by long short-time memory recurrent neural networks. Then we roughly cover the performance measures used by us to compare the tested classifiers; with a focus on the receiver operating characteristic favoured by us. It follows a discussion on the available training data. Therein we roughly explain how we extracted various feature subsets for the detection ([9, 10]). Finally we present a detailed description of our experiments and the results achieved.

2 RELATED WORK

Machine learning techniques have been used for network intrusion detection for some time, but the choice of the available training data is very limited. One of the few widely used datasets is derived from the DARPA datasets [11], which also happens to be one of the most comprehensive. The tcpdump data provided by the 1998 DARPA Intrusion Detection Evaluation network was processed and used for the 1999 KDD Cup contest at the Fifth International Conference on Knowledge Discovery and Data Mining. The learning task of this competition was to classify the preprocessed connection records into either normal traffic, or one out of the four given attack categories ('dos', 'probe', 'r2l', 'u2r').

Preprocessing of the data for the KDD Cup '99 competition was done with the MADAMID framework described in [12]. Each connection record contains 41 input features grouped into *basic features* and *higher-level features*. The dataset provides the training and testing datasets in a full set, and also a '10%' subset version with modified class distributions.

During the KDD Cup '99 competition, 24 entries were submitted. The first three places were occupied by entries that used variants of decision trees and showed only marginal differences in performance [13, 14, 15]. In ninth place in the challenge, was the 1-nearest neighbour classifier. The first 17 submissions of the competition were all considered to perform well and are summarised by [16].

Observing feature reduction on the KDD Cup '99 datasets, the majority of published results are trained and tested on the '10%' training set only [17, 18, 19]. Some researchers used custom-built datasets, with 11,982 random records extracted from the '10%' KDD Cup '99 training set [20, 21, 22].

After the challenge, a number of new results using learning algorithms on the KDD Cup '99 data were published. In the following papers, the authors used the same training and testing data as requested in the challenge, and provided partially comparable results: [23] evaluated a comprehensive set of machine learning algorithms and suggest a multi-classifier model with a multi-class topology. [24] applied the classical Adaboost algorithm and a modified version of the same to the KDD Cup '99 datasets. Decision stumps were chosen as a weak classifier and given as input to Adaboost. [25] demonstrated a genetic programming approach for large datasets comparing the results of using the first 8 basic features only with using all features of

[1] `http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/`

[2] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

the dataset. A machine learning approach, based on unsupervised presentation of data, is applied by [26]. They used a multi-layer, self-organising feature-map hierarchy with customised datasets.

There are also a number of interesting publications where the results are not directly comparable due to the use of different training and test datasets. In an early paper, [27] suggested genetic algorithms and decision trees for automatic rule generation for an expert system that enhances the capability of an existing IDS. [28] observed a nonparametric density estimation approach, based on Parzen-window estimators with Gaussian kernels. [29] compared the performance of a linear genetic programming approach to artificial neural networks and support vector machines.

[30] investigated the results of linear genetic programming and multi-expression programming. Other hybrid approaches combine neural networks and support vector machines [31], artificial neural networks and a fuzzy inference system [20], decision trees and support vector machines [32], and neural networks with recurrent neural networks [33].

[34] and [35] suggested the use of neural networks as components of intrusion detection systems; and an application of recurrent neural networks is proposed by [36]. [37] compared the performance of a selection of neural network architectures for statistical anomaly detection to datasets from four different scenarios. The use of hidden Markov models to detect complex multistage Internet attacks that occur over extended periods of time is described by [38]. An event classification scheme based on Bayesian networks is proposed by [39]. An approach using a modified Jordan recurrent neural network is proposed by [40]. Another approach [41] successfully applies Jordan recurrent neural networks to detect SQL-based attacks.

A framework for unsupervised learning, with two feature maps mapping unlabelled data elements to a feature space, is suggested by [42]. [43] further illustrated that neural networks can be efficiently applied to network data in both a supervised and an unsupervised learning approach. [44] demonstrated that supervised learning techniques applied to the KDD Cup '99 training data significantly outperform unsupervised methods. The best performance is achieved by non-linear methods.

With a focus on sequential relations between events [45] show that hidden Markov models outperform neural networks; and [46, 47] show that recurrent neural networks outperform neural networks likewise. [48] show that recurrent neural networks perform well in both anomaly and misuse detection.

A short time after the 1998 and 1999 DARPA intrusion detection system evaluations, [49] wrote a detailed critique identifying shortcomings of the provided datasets. The primary criticism of the paper was that the evaluation failed to verify that the network realistically simulated a real-world network. [50] looked more closely on the content of the 1999 DARPA evaluation tcpdump data and discovered that the simulated traffic contains problematic irregularities. The authors state that many of the network attributes, which have a large range in real-world traffic, have a small and fixed range in the simulation. Since the 1998 evaluation data was generated by the same framework, it can be assumed that it suffers from similar problems.

[51] investigated why classifiers fail to detect most of 'r2l' and 'u2r' attacks in the KDD Cup '99 datasets. They conclude that it is not possible for any classifier to accomplish an acceptable detection rate of these two attack classes. The authors admit that this might be not the case when the KDD Cup '99 datasets are used in an anomaly detection context.

[52] applied the tcpdump traffic data files provided with DARPA datasets to the Snort intrusion detection system (see [53]). The performance of this mainly signature-based intrusion detection system was rather poor. The authors reason that it is due to the fact that it is difficult to detect 'dos' and 'probe' attacks with a fixed signature. So the detection on the 'r2l' and 'u2r' attacks is in contrast much better. The paper emphasises the need to build a more realistic intrusion detection dataset with focus on false positive evaluation and more recent attacks.

[54] argue that although the KDD Cup '99 datasets suffer from various problems [49], they are still an effective benchmark to compare different intrusion detection methods. To address some of the known issues the authors created a revised version of the datasets, called NSL-KDD[3]. The authors changed the class distributions by cleaning the training and testing datasets from redundant records, and then adding records inversely proportional to their percentage in the original KDD data set. This was done to prevent learning algorithms to be biased towards the more frequent records.

3 LSTM-RNN BACKGROUND

Recurrent neural networks (RNNs) are dynamic systems; they have 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.

Below we roughly cover simple RNNs and point out their limitations prior explaining the LSTM-RNN architecture and their extensions.

3.1 RNN Architecture

RNNs range from partly to fully connected, and two simple RNNs are suggested by [55, 56]. The Elman network is similar to a three-layer neural network, but additionally, the outputs of the hidden layer are saved in so-called 'context cells'. The output of a context cell is circularly fed back to the hidden neuron along with the originating signal. Every hidden neuron has its own context cell and receives input both from the input layer and the context cells. Elman networks can

[3] http://iscx.cs.unb.ca/NSL-KDD/

be trained with standard error backpropagation, the output from the context cells being simply regarded as an additional input. Jordan networks have a similar structure to Elman networks, but the context cells are instead fed by the output layer.

RNNs need to be trained differently to the feed-forward neural networks (FFNNs). The most common and well-documented learning algorithms for training RNNs in temporal, supervised learning tasks are *backpropagation through time* (BPTT) and *real-time recurrent learning* (RTRL). In BPTT, the network is unfolded in time to construct an FFNN. Then, the generalised delta rule is applied to update the weights. This is an offline learning algorithm in the sense that we first collect the data and then build the model from the system. In RTRL, the gradient information is forward propagated. Here, the data is collected online from the system and the model is learned during collection. Therefore, RTRL is an online learning algorithm.

3.2 Vanishing Error Problem

In RTRL all the information necessary to compute the activity gradient is collected as the input stream is presented to the network. This makes a dedicated training interval obsolete. The algorithm comes at significant calculation cost per update cycle, and the stored information is non-local. But the memory required depends only on the size of the network. A detailed description of the RTRL algorithm is given in [57, 58].

Standard RNN cannot bridge more than 5–10 time steps. Error signals tend to either blow-up or vanish. Blown-up error signals lead straight to oscillating weights, whereas with a vanishing error, learning takes an unacceptable amount of time, or does not work at all. A detailed theoretical analysis of the problem with long-term dependencies is presented in [59]. The paper also briefly outlines several proposals on how to address this problem.

3.3 Standard LSTM

One solution that addresses the vanishing error problem is a gradient-based method called *long short-term memory* (LSTM) published by [6, 7, 8]. LSTM can learn how to bridge minimal time lags of more than 1,000 discrete time steps [6]. The solution uses *constant error carousels* (CECs), which enforce a constant error flow within special cells. Access to the cells is handled by multiplicative gate units, which learn when to grant access.

In the absence of new inputs to the cell, we know that the CEC's backflow remains constant. However, as part of a neural network, the CEC is not only connected to itself, but also to other units in the neural network. We need to take these additional weighted inputs and outputs into account. Incoming connections to neuron j can have conflicting weight update signals, because the same weight is used for storing and ignoring inputs. For weighted output connections from neuron j, the same weights can be used to both retrieve j's contents and prevent j's output flow to other neurons in the network.

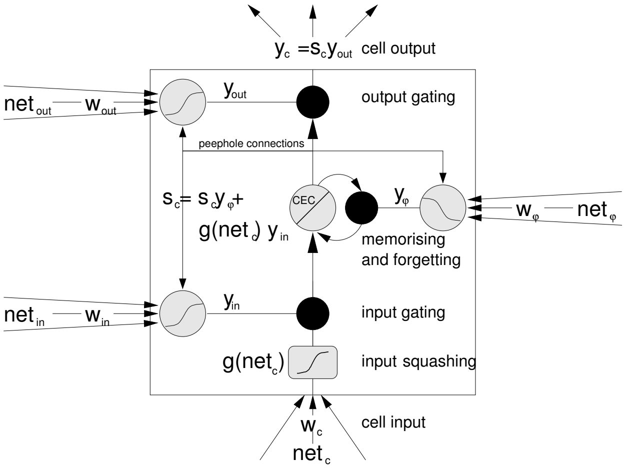

Figure 1: A standard LSTM memory cell with forget gate and peephole connections. The state of the cell is denoted as s_c. Read and write access is regulated by the input gate, y_{in}, and the output gate, y_{out}; cell reset is handled by the forget gate y_φ. The internal cell state is calculated by multiplying the result of the squashed input, g, by the result of the input gate, y_{in}, and then adding the state of the last time step, $s_c(t-1)$. Finally, the cell output is calculated by multiplying the cell state, s_c, by the activation of the output gate, y_{out}.

To address the problem of conflicting weight updates, LSTM extends the CEC with *input and output gates* connected to the network input layer and to other memory cells. This results in a more complex LSTM unit, called a *memory cell*; its standard architecture is shown in Fig. 1.

3.4 LSTM Extensions

The self-connection in a standard LSTM network has a fixed weight set to '1' in order to preserve the cell state over time. Unfortunately, the cell states s_c tend to grow linearly during the progression of a time series presented in a continuous input stream. The main negative effect is that the entire memory cell loses its memorising capability, and begins to function like an ordinary RNN network neuron. To address this problem, [7] suggested that an adaptive forget gate could be attached to the self-connection. Forget gates can learn to reset the internal state of the memory cell when the stored information is no longer needed.

In standard LSTM, gates do not have direct access to the internal cell state. They are connected only to the input units and the cell outputs. As long as the output gate is closed, the output of the cell is close to zero. Until the output gate starts to open, none of the gates have any information about the state of the CEC they should control. To address this problem, [8] suggested adding weighted connections from the internal cell state to all gates within a memory block. These connections are labelled as *peephole connections* which allow gates to learn to protect the internal cell state from unwanted inputs during the forward pass, whereas in the backward pass, they learn to protect the internal cell state from unwanted error signals.

The original LSTM training algorithm as described in [8] used a combination of truncated BPTT and RTRL. In [60] the authors suggest to use full BPTT to ease the implementation. They found that the full variant is slightly faster without changing results.

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 and Costs

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 two types of errors can have different costs. A cost matrix specifies the costs associated with different error conditions. All elements of the confusion matrix are then multiplied by the corresponding value of a cost matrix. The total cost of a model is the sum of all these products. The average cost is the total cost divided by the total number of classifications.

The confusion matrix with exemplary cost values shown in Table 1 is for a two-class problem. In this case a type I error is multiplied with 4 and a type II error with 1.

pred. → **actual** ↓	**1** positive	**2** negative
1 positive	a (0)	b (1)
2 negative	c (4)	d (0)

Table 1: A confusion matrix with cost values for a binary 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 algorithm 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 then a high accuracy 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 meaningful.

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 test set; 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 the 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.

In our experiments we will use ROC-curve and the resulting AUC value as prefered performance measures.

5 IDS-DATASETS

The choice of training data available for machine learning in the field of network intrusion detection systems is very limited. Two of the few labelled datasets are the DARPA[4]/KDD Cup '99 datasets[5] [11, 12] and the UNB ISCX 2012[6] as presented in [61]. The ISCX 2012 was collected more recently in 2010 and was intended as a replacement for the KDD Cup '99. Unfortunately availability of the dataset is discontinued. We used the DARPA/KDD Cup '99 datasets since they are still the most comprehensive and widely used.

5.1 DARPA/KDD Cup '99 Dataset

These datasets are called DARPA/KDD Cup '99 datasets because their generation was sponsored by the *Defence Advanced Research Projects Agency* (DARPA ITO) and the *Air Force Research Laboratory* (AFRL/SNHS). The 1998 DARPA Intrusion Detection Evaluation network simulated an air force base local area network. Seven weeks of training data and two weeks of testing data were collected. The total collected data contains more than 200 instances of 39 mostly network-based attack types embedded in background traffic similar to that of an air force base local area network.

All traffic is either classified as ('normal') or as one of various attack types. The attack types are grouped into the four attack categories: *denial-of-service* ('dos'), *network probe* ('probe'), *remote-to-local* ('r2l') and *user-to-root* ('u2r') attacks. In addition, the data contains anomalous user behaviour, such as a normal user acting like a privileged user.

The aim of 'dos' attacks is to prevent users accessing a service. 'TCP syn floods' are an example of this type of attack. 'Probe' attacks, such as 'port scans' and 'ip sweeps' are used to collect information about potential targets. Attackers on a remote machine using 'r2l' attacks try to gain user access on a machine they do not have access to. This can be achieved by, for example, dictionary attacks based on password guessing. A 'u2r' attack occurs when an attacker who has already achieved user access on a system tries to gain privileged access. Various buffer overflow attacks against system services fall in this category.

Attackers often use combinations of the attack types classified above. In the majority of cases, attackers follow a 'probe' → 'r2l' → 'u2r' pattern of behaviour.

The data provided by the 1998 DARPA Intrusion Detection Evaluation network was further processed and used for the 1999 KDD Cup contest at the fifth International Conference on Knowledge Discovery and Data Mining[7]. The learning task of this competition was to classify the preprocessed connection records into either normal traffic or one out of the four given attack categories ('dos', 'probe', 'r2l', 'u2r').

The seven weeks of network traffic collected from the DARPA training data were preprocessed into five million labelled and categorised connection records of approximately 100 bytes each; and the two weeks of training data were processed into two million unlabelled connection records. Preprocessing of the DARPA data for the 1999 KDD Cup contest was done with the MADAMID framework described in [12]. The

[4] `http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/`

[5] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

[6] `http://iscx.ca/datasets`

[7] The KDD Cup is an annual *Knowledge Discovery and Data Mining competition* organised by the ACM Special Interest Group on Knowledge Discovery and Data Mining.

KDD Cup '99 datasets are available from the UCI KDD Archive as the 1999 KDD Cup Dataset[8].

A connection record summarises the packets of a 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 labelled connection records in the training set are either categorised as 'normal' or indicate one of 22 types of attack. As far as we know, the KDD Cup '99 dataset is, as of today, still the most thorough observed and freely available dataset; with fully labelled connection records spanning several weeks of network traffic and a large number of different attacks.

Each connection record contains 41 input features—34 continuous- and 7 discrete-valued—grouped into *basic features* and *higher-level features*. The *basic features* are directly extracted or derived from the header information of IP packets and TCP/UDP segments in the tcpdump files of each session. This was done by using a modified version of the freely available *Bro Intrusion Detection System*[9] presented in [2]. Each connection record was produced when either the connection was terminated or Bro was closed. The listfiles for tcpdump from the DARPA training data where used to label the connection records.

The so-called *content-based higher-level features* use domain knowledge to look specifically for attacks in the actual data of the segments recorded in the tcpdump files. These address 'r2l' and 'u2r' attacks, which sometimes either require only a single connection or are without any prominent sequential patterns. Typical features include the number of failed login attempts and whether or not root access was obtained during the session.

Furthermore, there are *time-based and connection-based derived features* to address 'dos' and 'probe' attacks. *Time-based features* examine connections within a time window of two seconds and provide statistics about these. To provide statistical information about attacks exceeding a two-second time-window, such as slow probing attacks, *connection-based* features use a connection window of 100 connections. Both are further split into *same-host features*, which provide statistics about connections with the same destination host, and *same-service features*, which examine only connections with the same service.

The KDD Cup '99 competition provides the training and testing datasets in a full set, and also a so-called '10%' subset version. The '10%' subset was created due to the huge amount of connection records present in the full set; some 'dos' attacks have millions of records. For this reason, not all of these connection records were selected. Furthermore, only connections within a time-window of five minutes before and after the entire duration of an attack were added into the '10%' datasets. To achieve approximately the same distribution of intrusions and normal traffic as the original DARPA dataset, a selected set of sequences with 'normal' connections were also left in the '10%' dataset. Training and test sets have different probability distributions.

The full training dataset contains nearly five million records. The full training dataset and the corresponding '10%' both contain 22 different attack types in the order that they were used during the 1998 DARPA experiments.

The full test set, with nearly three million records, is only available unlabelled; but a '10%' subset is provided both as unlabelled and labelled test data. It is specified as the 'corrected' subset, with a different distribution and additional attacks not part of the training set. For the KDD Cup '99 competition, the '10%' subset was intended for training. The 'corrected' subset can be used for performance testing; it has over 300,000 records containing 37 different attacks. It is to be noticed that the sample distribution of 'probe', 'r2l' and 'u2r' attacks varies strongly between the training sets and the test set.

In the following, we roughly cover the applied feature selection methods.

5.2 Extracting Salient Features

For feature selection we built and examined post-pruned decision trees and backward elimination, as covered in detail in [10]. The applied J4.8 decision tree algorithm implements subtree raising as a pruning operation. In subtree raising, the algorithm moves nodes up towards the root of the tree and discards other nodes on the way.

After the first build from the training set using all features, we removed features from the dataset that were not part of the tree. Then we continued with a leave-one-out reduction until the removal of any feature led to significant performance loss in any of the five applied classifiers. We used true positive rate, false positive rate, precision, accuracy and costs as performance metrics in each traffic class. We also frequently estimated the ROC curve and calculated the *area under curve* (AUC) value using the Mann Whitney statistic. All values, except costs, were provided by WEKA. Costs were manually calculated using the suggested values provided by the KDD Cup '99 as shown in Table 2.

Table 2: The cost matrix provided by the KDD Cup '99 challenge with a minor modification. To ensure comparability, we increased the cost for classifying 'u2r' attacks as 'normal' traffic from 3 to 4.

		prediction				
		normal	probe	dos	u2r	r2l
actual	normal	0	1	2	2	2
	probe	1	0	2	2	2
	dos	2	1	0	2	2
	u2r	4*	2	2	0	2
	r2l	4	2	2	2	0

*was 3 in original KDD Cup '99 matrix

To limit the number of iterations, our leave-one-out approach was biased. By default, we kept features

[8] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[9] http://bro-ids.org/

close to the root of the tree, and one-by-one, removed features close to or at leaves. We preferred the removal of features that require domain knowledge or detailed traffic data analysis to features easily extracted from network data. We also frequently observed the classification and run-time performance of the five applied classifiers.

From the observed subsets, in every run with improved or comparable performance, we picked the best-performing attribute set. We declared the absent attribute of the best-performing subset as an unimportant attribute. We tested the performance of the final minimal feature set against the KDD Cup '99 test set.

The first approach resulted in a set of 11 selected features, which consisted of 7 basic features and 4 higher-level features [9]. The selected minimal features were 'duration', 'protocol_type', 'service', 'source_bytes', 'dst_bytes', and 'wrong_fragment'. Chosen higher-level features were 'serror_rate', 'dst_host_srv_count', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', and 'dst_host_rerror_rate'.

The exhaustive, feature-by-feature reduction of our second approach led to 8 important features, in which we identified the 4 most important minimal features. The 4 features are 'service', 'src_bytes', 'dst_host_diff_srv_rate', and 'dst_host_rerror_rate'. Additional important features are 'dst_bytes', 'hot', 'num_failed_logins', and 'dst_host_srv_count'.

6 EXPERIMENTS

Experiments were run with different parameters and structures of an LSTM recurrent neural network; such as the number of memory blocks and the cells per memory block, the learning rate, and the number of passes through the data. We also ran experiments with a layer of hidden neurons, with peephole connections, with forget gates, and with LSTM shortcut connections; all being extensions of LSTM as documented in [7].

For training we used a combination of truncated BPTT and RTRL, as originally described in [8]. The input squashing function of the LSTM memory cells was in the range [-2,2], and the output squashing function in the range [-1,1]. For the gates and standard neurons we used an logistic sigmoid activation in the range [0,1].

6.1 LSTM Network Parameters

To identify suitable network parameters and structure, experiments were started with a basic LSTM network structure using 43 input neurons connected to a hidden layer of two memory blocks with two cells each and peephole connections. The input neurons were fully connected to the hidden layer. The output was provided by five target neurons.

The '10%' KDD Cup '99 training dataset was used for training, and the corresponding '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. The number of iterations was fixed to 50 training cycles (referred to as epochs). The performance evaluation of the learned networks was supported by manual observation of the confusion matrix and by accuracy. We trained neural networks and ran performance tests to find optimal parameters for learning rate, network type, and LSTM features. We varied the learning rate in the interval [0.01–0.5], and we did not use any weight decay.

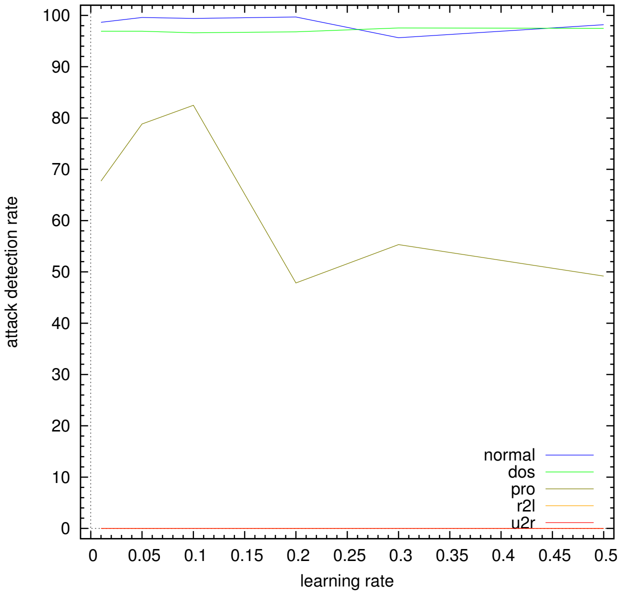

Figure 2: The detection rate results for all traffic types using well-performing LSTM recurrent neural networks, containing two memory blocks with two cells each. We trained the networks for 50 epochs, with different learning rates, in the interval between [0.01–0.5]. From the results, we can conclude that we can already get good results classifying the traffic classes 'normal', 'dos' and 'probe' when using a learning rate of around 0.1.

We experimented with different 'pure' feed-forward networks and hybrid networks, including hidden neurons and LSTM memory blocks. The number of hidden layers was fixed to one in all experiments. For both types of networks we ran experiments with 5–86 hidden neurons. Each experiment consisted of eight trials. Finally, we consecutively added forget gates (no manual reset), peephole connections and shortcuts to the basic LSTM network to assess their impact on learning performance.

Experiments with lower learning rates showed a slightly better classification performance. Naturally, for a low learning rate (0.01), the expected number of required iterations for low frequency attacks are very large ($\geq$ 10,000 epochs). As a trade-off between training time and classification performance, we decided to set the learning rate to not lower than 0.1 for following experiments. The best detection rate is shown in Fig. 2. We conclude from these results that 50 epochs are not sufficient to train 'r2l' and 'u2r' attacks using any learning rate; but for the traffic classes 'normal', 'dos' and 'probe', a learning rate of around 0.1 then already provides good results.

All LSTM-hybrid-networks showed good performance in terms of accuracy. Learning of hybrid networks was noticeably faster as well. The 'best' results were achieved using one feed-forward network layer with 20 hidden neurons. All trained networks achieved 'good' results with an accuracy value of over >90%, but these results were slightly less accurate than results possible with a standard LSTM network. From this we conclude that adding hidden neurons makes LSTM prone to over-fitting. Furthermore, we noted that 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 further experiments forget gates and shortcuts were used. Peephole connections were only activated in experiments on datasets using all features.

6.2 LSTM Network Structure Identification

To find a suitable network structure for training the KDD Cup '99 data, experiments were run with LSTM networks using four different topologies:

- Two memory blocks with two cells each,
- four memory blocks with two cells each,
- four memory blocks with four cells each, and
- eight memory blocks with four cells each.

All LSTM networks used forget gates and shortcuts; additionally we activated peephole connections when training with all features. The learning rate was fixed at 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' test set datasets with all features.

To find the minimum number of required iterations the training data was presented from five to up to one thousand epochs to each of the four observed network structures. We ran eight trials of each network setup. Results with good accuracy for attack detection (attack/normal two class classification) 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 generalisation performance decreases for that specific traffic type.

After 25–90 epochs, most networks learned DoS attacks. With further training the detection rate first peaks at 125 epochs, and then again at about 500 epochs. Network probes are mostly learned after 50–125 epochs and also peak at about 500 epochs. The rare attack categories, 'r2l' and 'u2r', need many more presentations of the training data. Attacks of the class 'r2l' need 200–1,000 epochs, and 'u2r' attacks need 125–1,000 epochs before they are learned as well as possible. Rare attacks might require more than 1,000 presentations of the training data; but it remains questionable if they can be learned at all using the available training data.

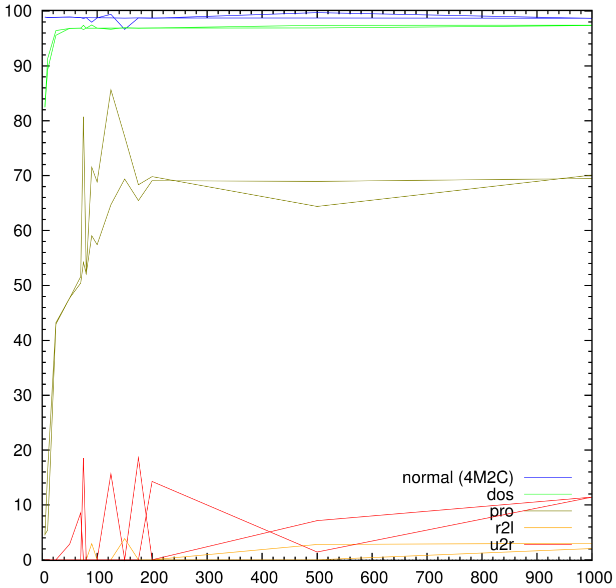

Figure 3: This figure shows the performance of the LSTM network structure with four memory blocks, containing two memory cells each (labelled as 4M2C), which proved to be a good compromise between computational cost and detection performance. Detection rate performance was measured for each traffic class according to the number of epochs trained.

After 500–600 epochs, approximately 50% of the trials achieve results with a low error rate. The performance of all networks decreases after further training. In comparison to standard neural networks with a hidden layer, LSTM is much more prone to over-fitting. After learning a specific traffic type, the network improves in memorising the learned traffic types in the training data, and the generalisation performance for these continuously decreases. Naturally, the detection performance of trained networks on the test set quickly degrades after reaching peak performance for the most frequent traffic type, which are, in this case, DoS attacks.

With increasing size of the LSTM network, learning requires more presentations of the training data. Rare 'r2l' attacks require more than 1,000 epochs on the two larger LSTM networks. On the other hand, the small LSTM network with two memory blocks has problems learning the very rare and difficult-to-learn 'u2r' attacks.

We think that networks with four memory blocks containing two cells each offer a good compromise between computational cost and detection performance. This type of network structure was used by us for our next experiments. Fig. 3 shows a performance comparison of the best results of the four different network types evaluated.

6.3 Classifier Performance Analysis

As learned from previous experiments in which we evaluated different network parameters and network structures, LSTM networks with four memory blocks containing two memory cells each were built. We used forget gates and shortcut connections. For experiments training on all features we activated peephole connections. Fig. 4 outlines the applied LSTM network.

We applied learning rates of 0.5 and 0.1. Additionally, we applied in selected experiments an exponential learning decay of 0.99 or 0.999 consecutively. For training the '10% training' dataset was used. The performance was tested on the training set and as well on the '10% corrected' test set. The networks were trained for up to 1,000 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 1,000 epochs. Every experiment contained 30 trials.

At the end of every trial a 5×5 confusion matrix was generated, accuracy and mean-squared error were calculated, and for every target neuron detection rate, precision and the AUC value were calculated. As required for ROC analysis no classification threshold was applied. All output values of all five target neurons for every tested pattern were recorded. Every target neuron represented one of the five target classes. The target output with the highest numerical value was used for traffic classification.

For the ROC calculations, we used the library version of the *proproc.2.8.0* software [62] with an non-parametric estimate for our curve calculations. The *proproc* software and support was kindly provided without cost on request by the developers.

Since multi-class ROC graphs are not plotable we examined the five target neurons of every trained LSTN network separately. We generated an ROC-Graph for every target neuron, representing each of network traffic classes: 'normal', 'dos', 'probe', 'r2l' and 'u2r'.

We trained LSTM networks and ran performance tests for

- all features for all attacks in a single network,
- all features for all attacks in different networks,
- minimal features for all attacks in a single network, and
- minimal features for individual attacks in individual networks.

6.4 Experiments Using All Features

We ran two experiment series, using all features, for training LSTM recurrent neural networks with all attack classes. The aim of the first experiment series was twofold: On the one hand, we plotted ROC curves to confirm that LSTM is actually able to correctly classify all five traffic classes in the training data. On the other hand, we estimated the minimum required training epochs for each traffic class.

We trained networks for 20–1,000 epochs, with 30 trials each, adding up to a total of 480 experiments. We applied a fixed learning rate of 0.1. At the end of each trial, we tested the performance of the trained neural network. For every test, we calculated the AUC values for every target neuron and saved the sequence of empirical operating points to plot the ROC curves.

We used all features of the preprocessed datasets, except two features with non-changing values in the training data. This left us with 39 input features, which we mapped to an input layer of size 40. We mapped the target feature, categorising one of the five traffic classes, to five target neurons.

In the second experiment series, we focused on training well-performing networks. Here, we directly calculated the AUC values, without saving the parametric points for the ROC curves. This dramatically reduced processing time and the amount of collected data. We kept the previously selected network architecture and parameters, but did not anymore consider peephole connections because they showed no benefit. We also added mean-squared error, to accuracy and AUC, as a standard performance measure.

Furthermore, we added an exponential decay to the learning rate. We initialised the learning rate to either 0.5 or 0.1, and set a decay of either 0.99 or 0.999 consecutively. Generally, training was stopped after 1,000 epochs. We also changed our testing procedure by freezing the weights after each epoch and testing the performance on the training and the test set, which led us to much better results.

We first analysed the results of the first experiment to find the best-performing LSTM network trained using all features and attack types. We observed all the trained LSTM networks in terms of ROC performance. We examined the five target neurons of every trained LSTM network separately. For each neuron, we generated a separate ROC graph and calculated the corresponding AUC value, representing one of the five traffic classes: 'normal', 'dos', 'probe', 'r2l' and 'u2r'.

For the majority of trained networks, we found that the target neurons representing normal traffic and network probes showed an excellent ROC performance. The 10 highest AUC values achieved for normal traffic results were in the range between [0.9665–0.9716], and for network probes, in the range between [0.9679–0.9826]. The corresponding ROC curves are shown in Figures 5 and 6.

The target neuron representing 'dos' attacks achieves a close to perfect discrimination between 'dos' attacks and other traffic classes (perfect = AUC value equal to 1.0) in well-performing networks. Here, the 10 highest AUC values are in the range between [0.9950–0.9971]. The corresponding ROC curves are shown in Fig. 7.

For 'u2r' attacks, we still achieved a good ROC performance. The 10 networks with the highest AUC values have a range of [0.8766–0.8909]. 'r2l' attacks proved to be the most difficult to classify in the test set. The 10 highest AUC values, in the range between [0.5380–0.5627], all show a poor performance. We also note that only 60% of all trained networks showed any classification performance better than random guessing.

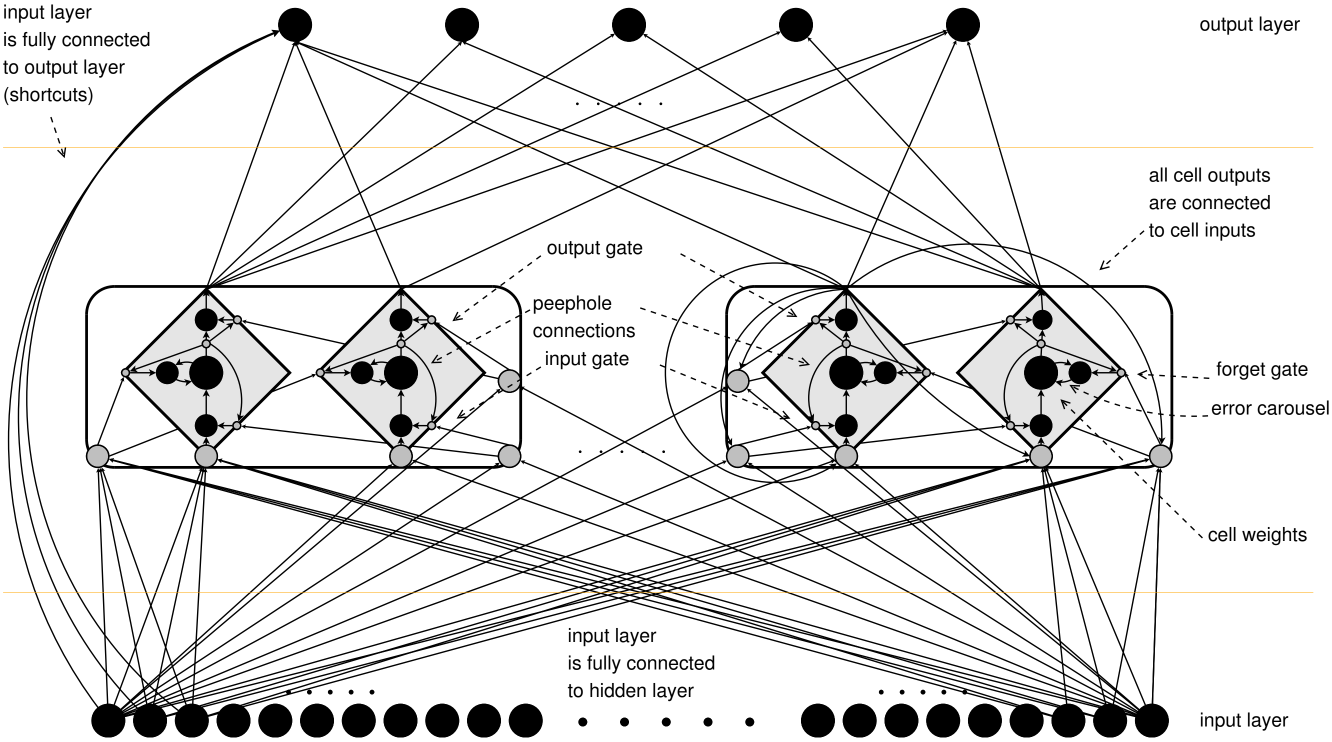

Figure 4: LSTM neural network with two memory blocks containing two cells each. The input layer is fully connected to the hidden and the output layers. This network has peephole connections and shortcuts. For reasons of clarity, not all connections are shown.

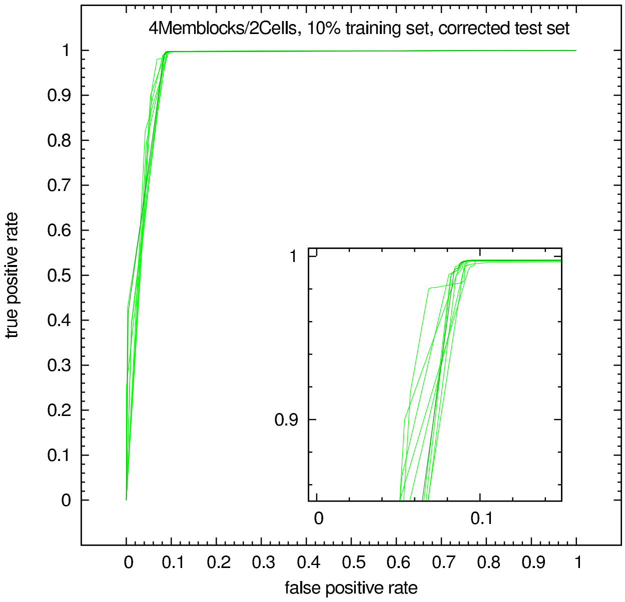

Figure 5: ROC curves of 10 well-performing networks for the target neuron representing the traffic class containing 'normal' traffic. The corresponding AUC values are in the range between [0.9665–0.9716]. This shows a very good classification performance in terms of AUC.

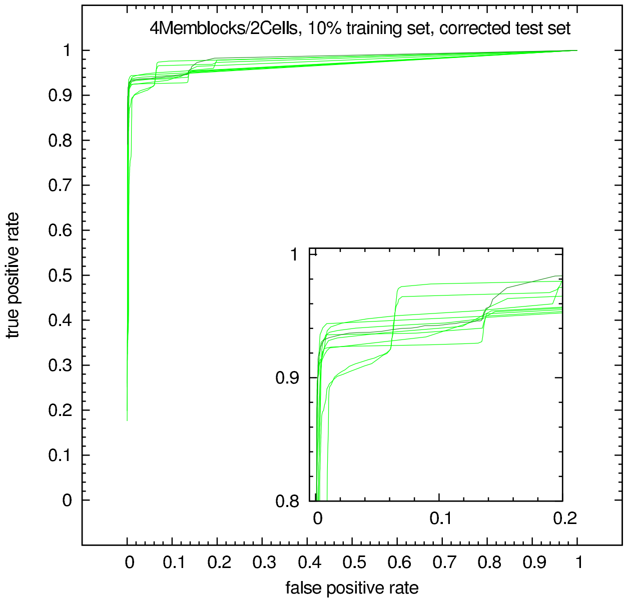

Figure 6: ROC curves of 10 well-performing networks for the target neuron representing the traffic class containing network probes. The corresponding AUC values are in the range [0.9679–0.9826]. In terms of AUC, this shows a very good classification performance. The bumpy curves are an indication that this class contains distinguishable subclasses of traffic.

Figures 8 and 9 show the corresponding ROC plots.

A derivation of the curves from a straight diagonal between the coordinates [0,0] and [1,1] to a curve bowing into the corner [1,0] of the graph is recognisable in all graphs. This shows that the trained networks were actually able to learn at least parts of all five traffic classes.

The partially bumpy ROC curves are expected, since we present whole classes of traffic where every class contains various subclasses of traffic. During the learning process, distinguishable subclasses do appear as bumps in the ROC graph. The curves for network probes and 'dos' attacks show that the classifier sepa-

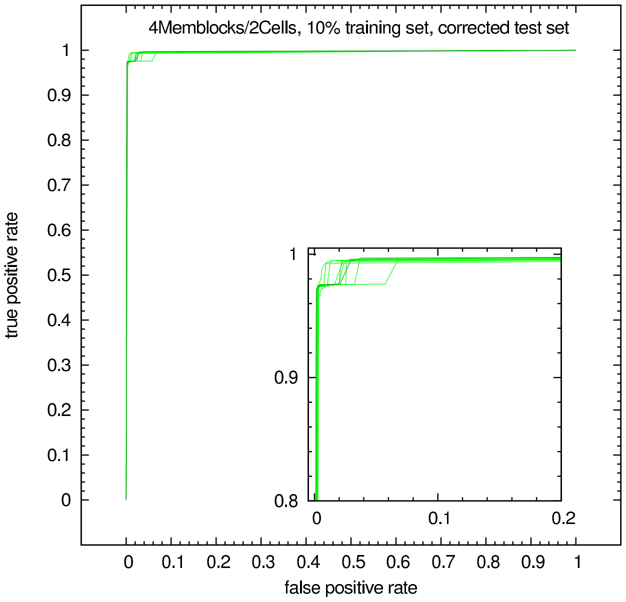

Figure 7: ROC curves of 10 well-performing networks for the target neuron representing the traffic class containing ‘dos’ attacks. The corresponding AUC values are in the range [0.9950–0.9971], which shows a close to perfect discrimination between ‘dos’ attacks and other traffic classes.

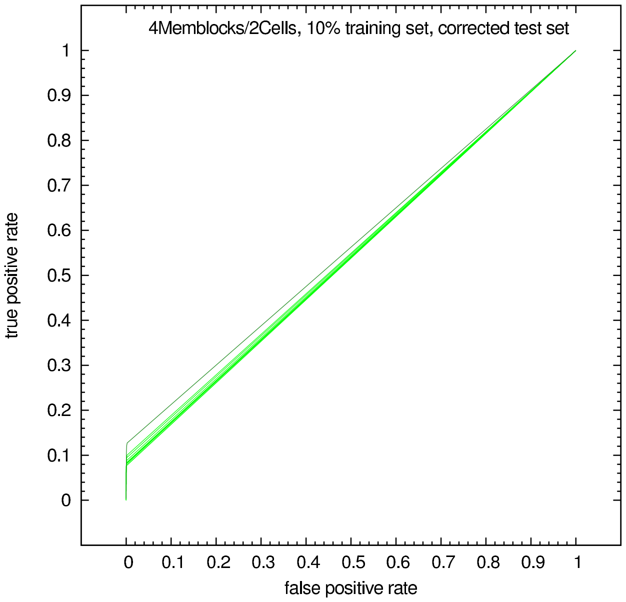

Figure 8: ROC curves of the 10 LSTM networks with the highest AUC values for the target neuron representing the traffic class containing ‘r2l’ attacks. The corresponding AUC values are in the range between [0.5380–0.5627]. This shows a rather poor ROC performance.

rates at least two subclasses.

In our second experiment series we focused on total MSE, accuracy, AUC per traffic class as performance measures. The investigated experimental configuration for training LSTM networks with all features and all attacks showed good results in terms of MSE and accuracy. The 10 lowest MSEs achieved on the test data in 30 trials are in the range [0.0259–0.0293]. The average MSE, picking the result with the lowest MSE of each trial, is 0.0308.

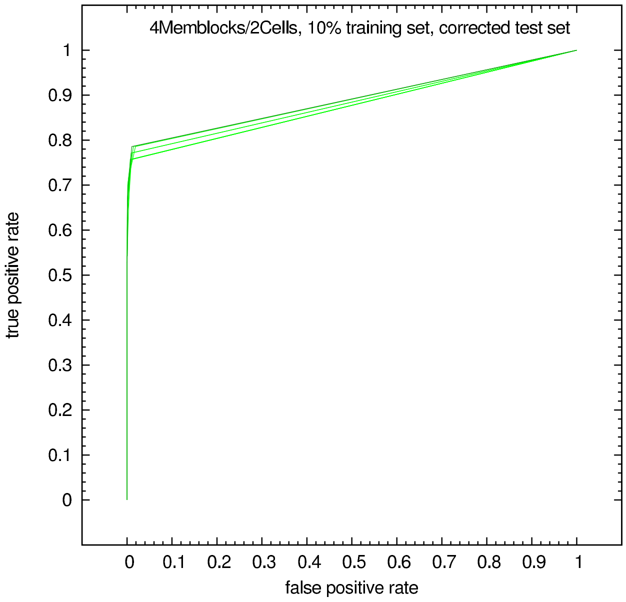

Figure 9: ROC curves of 10 well-performing networks for the target neuron representing the traffic class containing ‘u2r’ attacks. The corresponding AUC values are in the range between [0.8766–0.8909], which shows an acceptable ROC performance.

The average AUC values for the five traffic classes are 0.961 (‘normal’), 0.991 (‘dos’), 0.969 (‘probe’), 0.321 (‘r2l’) and 0.842 (‘u2r’). This shows an excellent performance of the trained LSTM networks, which successfully classified ‘dos’ attacks.

The performance in detecting ‘normal’ traffic, network probes and ‘u2r’ attacks is likewise very good. The AUC performance in classifying ‘r2l’ attacks is poor. Only a few networks learn to classify ‘r2l’ attacks with a performance better than guessing.

Table 3 shows the top five best-performing values in terms of MAUC. The table also includes the corresponding AUC and MSE values, and the minimum, average and 95% confidence interval over 30 trials of the test data.

Picking a network with a very low MSE of 0.0259, the five AUC values are 0.959 (‘normal’), 0.997 (‘dos’), 0.955 (‘probe’), 0.241 (‘r2l’) and 0.821 (‘u2r’). Here, we see an exceptional performance of the LSTM network on ‘dos’ attacks and an above-average performance on normal traffic. This is expected since most connection records in training and test data are related to either ‘dos’ attacks or normal traffic. All other traffic classes show an AUC performance below average. This also reflects in a MAUC value of 0.949, which is slightly below average.

Nevertheless, in terms of costs and accuracy, our trained network still outperforms the high-scoring entries of the KDD Cup ’99 challenge. With 93.82% accuracy and 22.13 cost, our network is far ahead in first place in comparison to the winning entries of the KDD Cup ’99 challenge. LSTM correctly classified 291,811 out of 311,029 instances from the test set. Our result outperforms the KDD Cup ’99 winning entry by impressive 3,462 instances.

Table 3: Summary of test results for training LSTM networks with all features for all traffic classes in one network. The classification performance in terms of AUC for 'normal' traffic, network probes and 'r2l' attacks is very good. The performance for classifying 'dos' attacks is exceptional. Only a few networks learn to classify 'r2l' traffic with a performance beyond guessing.

#	MSE	AUC					MAUC
		normal	dos	probe	r2l	u2r	
1	.030	.982	.994	.944	.617	.842	.971
2	.030	.954	.995	.990	.564	.796	.964
3	.031	.956	.989	.992	.604	.927	.962
4	.029	.976	.994	.965	.430	.833	.960
5	.031	.966	.990	.913	.473	.897	.957
min.	.029	.937	.982	.908	.186	.584	.935
avg.	.031	.961	.991	.969	.321	.842	.950
95% conf.		.956	.989	.960	.276	.812	.946
interval		.965	.992	.979	.366	.872	.953

Table 4: Confusion matrix of LSTM network trained with all features and all attacks having the lowest MSE on test data. True positive rates and precision of 'dos' attacks and 'normal' traffic are exceptional. True positive rate and precision for network probes and 'u2r' attacks are also acceptable. The result for 'r2l' attacks does not excel in this network.

		prediction					TPR	AUC
		normal	probe	dos	u2r	r2l	(DR)	
actual	normal	60272	233	81	6	1	0.995	0.959
	probe	898	3156	111	1	0	0.758	0.955
	dos	1262	233	228358	0	0	0.993	0.997
	u2r	58	0	0	12	0	0.171	0.821
	r2l	15828	106	399	1	13	0.001	0.241
	PRECISION	0.770	0.847	0.997	0.600	0.929	COST:	**0.2213**
	FPR (FAR)	0.072	0.002	0.007	0.000	0.000	ACC:	**93.82%**

Observing the results in the confusion matrix shown in Table 4, we note the very good true positive rates and precision of 'dos' attacks and 'normal' traffic. The true positive rate and precision for network probes and 'u2r' attacks are also acceptable. Unfortunately, the result for 'r2l' attacks does not excel at all in this network.

6.5 Performance Analysis with Minimal Feature Sets

For the detection of all attack classes using minimal feature sets, we used extracted minimal sets presented in [9] and [10]. The minimal sets used for training all attacks in a single LSTM network consisting of 8 and 4 features. We ran 30 trials to train the LSTM networks with the 8 and 4-feature sets for 1,000 epochs each. There we applied a learning rate of 0.1 and a decay of 0.999.

Using the 8-feature set, all trials show good results in terms of accuracy and MSE. With the 4-feature set, we still get good results for approximately half of the trials. The lowest MSE found for 8 and 4 features are 0.029, 0.025 and 0.024 respectively, with an average MSE of 0.031, 0.029 and 0.070. Here, we see the tendency of a performance improvement towards the smaller feature sets. But with the 4-feature set, there also comes the disadvantage of a decreasing yield of well-performing networks. This shows that LSTM has increasing difficulty learning to classify the data by using only 4 features.

In terms of MAUC, the LSTM networks trained with 8 features show the best results. In terms of AUC, the results show that LSTM learns to classify the traffic classes 'normal', 'probe' and 'u2r' very well. Once again, the results for classifying 'dos' attacks are exceptional. Few networks learn to classify few 'r2l' attacks. The top five networks are shown in Table 5.

We note that the LSTM networks trained with the 4 and 8-feature sets for all attacks in one network are superior to the results of other non-LSTM classifiers and the winning entries of the KDD Cup '99 challenge in terms of accuracy and costs. The results are 93.69% accuracy and 22.29 costs for the 8-feature set, and 93.72% accuracy and 22.24 costs for the 4-feature set. Compared to the LSTM network trained with all features, we note that these values almost match its performance.

The confusion matrices for the networks trained with 4 and 8 features show that the outstanding total performance is mainly due to an excellent detection of 'dos' attacks. The true positive rate for network probes is acceptable, taken into account that some probes are wrongly categorised as 'dos' attacks. The detection of 'r2l' attacks is poor. The 'u2r' attacks are not detected by LSTM using any of the reduced feature sets for all attacks in one network.

The results of the best-performing LSTM networks trained with the 8 and 4-feature sets for all attacks in one network are presented in the form of confusion

Table 5: Summary of test results for LSTM training with 8 features for all traffic classes in one network. In terms of AUC LSTM learns very well to classify the traffic classes ‘normal’, ‘probe’ and ‘u2r’. The results for classifying ‘dos’ attacks are exceptional. Only some networks learn to classify few ‘r2l’ attacks.

Rank	MSE	AUC					MAUC
		normal	dos	probe	r2l	u2r	
1	.025	.985	.999	.875	.811	.939	.985
2	.030	.981	.998	.928	.788	.772	.983
3	.029	.973	.995	.952	.720	.804	.976
4	.026	.969	.995	.947	.517	.954	.964
5	.031	.984	.991	.949	.423	.809	.959
min.	.025	.962	.970	.714	.128	.678	.932
avg.	.029	.977	.990	.920	.338	.819	.952
95% conf.		.974	.986	.902	.268	.790	.947
interval		.979	.993	.939	.408	.849	.957

Table 6: Confusion matrix of LSTM network trained with 8 features and all attacks with the lowest MSE on test data. This network shows an outstanding overall performance due to an excellent detection of ‘dos’ attacks and an acceptable classification performance of network probes. The performance of detecting ‘r2l’ and ‘u2r’ attacks is poor; but the high AUC values suggest there is still potential to learn them.

		prediction					TPR	AUC
		normal	probe	dos	u2r	r2l	(DR)	
actual	normal	60161	223	181	0	28	0.993	0.985
	probe	1001	2546	619	0	0	0.611	0.875
	dos	1217	226	228408	2	0	0.994	0.999
	u2r	70	0	0	0	0	0.000	0.939
	r2l	15687	169	199	4	288	0.018	0.811
PRECISION		0.770	0.805	0.996	0.000	0.911	COST:	**0.2229**
FPR (FAR)		0.072	0.002	0.012	0.000	0.000	ACC:	**93.69%**

Table 7: Confusion matrix of LSTM network trained with 4 features and all attacks with the lowest MSE on test data. This network shows an outstanding overall performance due to an excellent detection of ‘dos’ attacks and a still acceptable performance for classifying network probes.

		prediction					TPR	AUC
		normal	probe	dos	u2r	r2l	(DR)	
actual	normal	60182	154	221	0	36	0.993	0.967
	probe	889	2348	928	0	1	0.564	0.870
	dos	723	195	228935	0	0	0.996	0.993
	u2r	68	0	2	0	0	0.000	0.751
	r2l	16229	9	81	0	28	0.002	0.304
PRECISION		0.771	0.868	0.995	-	0.431	COST:	**0.2264**
FPR (FAR)		0.072	0.001	0.015	0.000	0.000	ACC:	**93.72%**

matrices in Tables 6 and 7.

6.6 Classifier Performance Comparison

The per-class performance results of LSTM networks trained with all features and minimal feature sets, in comparison with other trained classifiers, are summarised in Table 8. The results confirm that in terms of true positive rate, false positive rate, precision, accuracy and cost, the performance of the LSTM classifier is superior at detecting DoS and network probes. The results for detecting exploit-based attacks (‘r2l’ and ‘u2r’) are indeed very competitive, but do not match the very well-performing decision tree classifier.

For network probes, DoS and ‘r2l’ attacks, LSTM sacrifices precision for an improved true positive rate, when training with the minimal sets. Only for ‘u2r’ attacks does LSTM improve noticeably with a reduced feature set. Compared to the performance of other trained classifiers on the minimal set, we note that for ‘dos’ attacks and network probes, LSTM is superior in terms of true positive rate, false positive rate, precision, accuracy and cost. For ‘r2l’ attacks trained with 6 and 14 features, LSTM clearly outperforms the two other neural network-based classifiers (SVM, MLP).

7 CONCLUSIONS

We applied our own implementation of the *LSTM recurrent neural network* classifier to intrusion detection data. The results show that the LSTM classifier provides a superior performance in comparison to the results of the KDD Cup ’99 challenge and as well other tested strong static classifiers. The strengths are in the detection of ‘dos’ attacks and network probes, which both produce a distinctive time series of events. The performance on the attack classes that produce only a few events is comparable to the results of the other tested classifiers.

Performance is measured in terms of mean-squared error, confusion matrix, accuracy, ROC-curve and the corresponding AUC value. Our results show that ROC-curves are well suited for selecting well performing networks, although not common in related publications. The derivation 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 clearly recognisable in all graphs. Successfully trained LSTM-networks learned the five traffic classes at least partially.

We reasoned the selected experimental parameters on a number of chosen LSTM-RNN network topologies in detail. Furthermore, we make suggestions on improving LSTM performance. We used both the full feature and minimal feature sets.

The LSTM classifier shows its strength when training ‘dos’ attacks and network probes. The target neuron representing DoS attacks even achieves close to perfect discrimination between attacks and other traffic. These traffic classes tend to generate a high volume of consecutive connection records. Here, LSTM can strongly benefit from the fact that it can look back in time and learn to correlate these connections.

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. These two exploit-based attack classes generate in most cases only one connection record. If there is any time series information related to theses attacks hidden between other connection records, it seems to be very difficult to extract.

It is of lesser importance for our experiments that the attacks contained in the DARPA datasets are not recent. Reason is that the attacks are grouped into traffic-classes which are still valid. Therefore we expect that our results are applicable to more recent attacks; although this needs to be proven in future research. Admittedly on novel attacks unrelated to these four trained attack classes, the learned classifiers will likely fail.

We finally conclude that LSTM is very suitable for classifying high-frequency attacks. For low-frequency attacks, the benefit of using LSTM vanishes. Although we stress that the results achieved by LSTM are very competitive. This is the first reported demonstration of the successful application of LSTM recurrent neural networks to intrusion detection.

8 FUTURE RESEARCH

The nature of computer attacks is that they are very dynamic. Most of the attacks generated during the DARPA IDS evaluation were already well-known and outdated in the year 1998 when the evaluation was conducted. Until today, approaches taken for detecting and addressing attacks have changed several times. Today’s novel attacks are very different from the attacks of five years ago; and in five years time, they will be very different from today. And it is safe to assume that they will benefit from different features to be learned successfully.

Future intrusion detection systems supported by machine learning algorithms will need to deal with recent data and process it efficiently into connection records. We need to develop a framework that supports an expert to easily enrich the generated connection records with additional information. This is for the intuitive building of new features considered to be relevant for the detection of novel attacks. This could be supported by log information, network flows and alarms provided by hosts, syslog servers, switches, routers, firewalls, intrusion detection systems, and penetration testing tools.

Finally the expert will need to label connection records in an efficient way. Using current tools, this task is so time consuming that once all traffic necessary to train a supervised machine learner has been labelled, the traffic is already far outdated. We challenge that LSTM-RNN will perform well on more recent data. Creation of such a test set and comparison will be a future project.

Table 8: The table shows the winning results of the KDD Cup '99 challenge presented in [16] namely decision trees (commercial C5-variant), desision forest (DF), and the PNrule framework and the 1-nearest neighbour classifier. Additionally we trained Decision Trees (free C4.5-variant) and standard Neural Networks (MLP) classifier on the original KDD Cup '99 dataset. Then we present the full results of Decision Trees (C4.5) and standard Neural Networks (MLP) on the in comparison with the dynamic Long Short-Term Memory (LSTM) classifier trained with the preprocessed full-feature set (39p), 8-feature set, and 4-feature set.

		normal		probe		dos		u2r		r2l			
		TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	ACC	COST
KDD Cup '99 competition results (first three places + 1-nearest neighbour)													
all	C5	.995	.082	.833	.006	.971	.003	.132	.000	.084	.000	92.71%	.2336
	DF	.994	.085	.845	.002	.975	.003	.118	.000	.073	.000	92.92%	.2362
	PNr	.995	.089	.730	.001	.970	.001	.066	.000	.107	.001	92.59%	.2387
	1-NN	.996	.091	.750	.002	.973	.005	.035	.000	.006	.000	92.33%	.2530
Results of strong static classifiers as published in [23]													
all	MLP	n/a	n/a	.887	.004	.972	.003	.132	5E-4	.056	1E-4	n/a	n/a
	GAU	n/a	n/a	.902	.113	.824	.009	.228	.005	.096	.001	n/a	n/a
	K-M	n/a	n/a	.876	.026	.973	.004	.298	.004	.064	.001	n/a	n/a
	NEA	n/a	n/a	.888	.005	.971	.003	.022	6E-6	.034	1E-4	n/a	n/a
	RBF	n/a	n/a	.932	.188	.730	.002	.061	4E-4	.058	.003	n/a	n/a
	LEA	n/a	n/a	.838	.003	.972	.003	.066	3E-4	.001	3E-5	n/a	n/a
	HYP	n/a	n/a	.848	.004	.972	.003	.083	9E-5	.010	5E-5	n/a	n/a
	ART	n/a	n/a	.772	.002	.970	.003	.061	1E-5	.037	4E-5	n/a	n/a
	C4.5	n/a	n/a	.808	.007	.970	.003	.018	2E-5	.046	5E-5	n/a	n/a
Results of LSTM-RNN in comparison to two strong static classifiers													
all	C4.5	.995	.089	.747	.002	.973	.003	.086	.000	.058	.000	92.58%	.2421
	MLP	.984	.090	.725	.001	.973	.011	.086	.000	.056	.000	92.37%	.2484
	LSTM	.995	.072	.758	.002	.993	.007	.171	.000	.001	.000	**93.82%**	**.2213**
8p	C4.5	.995	.088	.776	.002	.971	.007	.257	.000	.055	.000	92.50%	.2450
	MLP	.995	.092	.648	.001	.973	.012	.000	.000	.000	.000	92.18%	.2541
	LSTM	.993	.072	.611	.002	.994	.012	.000	.000	.018	.000	**93.69%**	**.2229**
4p	C4.5	.993	.088	.766	.002	.973	.006	.043	.000	.037	.001	92.46%	.2488
	MLP	.987	.090	.654	.005	.970	.019	.000	.000	.000	.000	91.76%	.2625
	LSTM	.993	.072	.564	.001	.996	.015	.000	.000	.002	.000	**93.72%**	**.2264**

ACKNOWLEDGEMENTS

This research was partially funded by the fellowship programs of the University of the Western Cape (2006), National Research Foundation (2006), Rhodes University (2013) and the University of South Africa (2014). We would like to thank Christian W. Omlin for his guidance and advice.

REFERENCES

[1] P. Garcia-Teodoro, J. Diaz-Verdejo and E. Macia-Fernandez, G. andVazquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges". *Computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009. DOI `http://dx.doi.org/10.1016/j.cose.2008.08.003`.

[2] V. Paxson. "Bro: A system for detecting network intruders in real-time". *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999. DOI `http://dx.doi.org/10.1016/S1389-1286(99)00112-7`.

[3] S. Axelsson. "The base-rate fallacy and the difficulty of intrusion detection". *ACM transactions on information and system security*, vol. 3, no. 3, pp. 186–205, 2000. DOI `http://dx.doi.org/10.1145/357830.357849`.

[4] W. Lee, W. Fan, W. Miller, S. Stolfo and E. Zadok. "Toward cost-sensitive modeling for intrusion detection and response". *Journal of computer security*, vol. 10, no. 1/2, pp. 5–22, 2002.

[5] Y. Liao and V. Vemuri. *Enhancing computer security with smart technology*, chap. Machine learning in intrusion detection, pp. 93–124. Auerbach Publications, 2006.

[6] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

[7] F. Gers, J. Schmidhuber and F. Cummins. "Learning to forget: Continual prediction with LSTM". Tech. Rep. IDSIA-01-99, IDSIA, Lugano, Lugano, CH, 1999.

[8] F. Gers, N. Schraudolph and J. Schmidhuber. "Learning precise timing with LSTM recurrent networks". *Journal of machine learning research*, vol. 3, pp. 115–143, 2002.

[9] R. C. Staudemeyer and C. W. Omlin. "Feature set reduction for automatic network intrusion detection with machine learning algorithms". In *Proceedings of*

the southern African telecommunication networks and applications conference (SATNAC). 2009.

[10] R. C. Staudemeyer and C. W. Omlin. "Extracting salient features for network intrusion detection using machine learning methods". *South African computer journal*, 2014.

[11] R. Lippmann, J. Haines, D. Fried, J. Korba and K. Das. "The 1999 DARPA off-line intrusion detection evaluation". *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000. DOI `http://dx.doi.org/10.1016/S1389-1286(00)00139-0`.

[12] W. Lee and S. Stolfo. "A framework for constructing features and models for intrusion detectionsystems". *ACM transactions on information and system security*, vol. 3, no. 4, pp. 227–261, 2000. DOI `http://dx.doi.org/10.1145/382912.382914`.

[13] B. Pfahringer. "Winning the KDD99 classification cup: Bagged boosting". *SIGKDD explorations newsletter*, vol. 1, pp. 65–66, 2000. DOI `http://dx.doi.org/10.1145/846183.846200`.

[14] M. Vladimir, V. Alexei and S. Ivan. "The MP13 approach to the KDD'99 classifier learning contest". *SIGKDD explorations newsletter*, vol. 1, pp. 76–77, 2000. DOI `http://dx.doi.org/10.1145/846183.846202`.

[15] R. Agarwal and M. Joshi. "PNrule: A new framework for learning classier models in data mining". Tech. Rep. 00-015, Department of Computer Science, University of Minnesota, 2000.

[16] C. Elkan. "Results of the KDD'99 classifier learning". *SIGKDD explorations newsletter*, vol. 1, pp. 63–64, 2000. DOI `http://dx.doi.org/10.1145/846183.846199`.

[17] S. Sung, A.H. Mukkamala. "Identifying important features for intrusion detection using support vector machines and neural networks". In *Proceedings of the symposium on applications and the Internet (SAINT)*, pp. 209–216. IEEE Computer Society, 2003. DOI `http://dx.doi.org/10.1109/saint.2003.1183050`.

[18] H. Kayacik, A. Zincir-Heywood and M. Heywood. "Selecting features for intrusion detection: A feature relevance analysison KDD 99 intrusion detection datasets". In *Proceedings of the third annual conference on privacy, security and trust (PST)*. 2005.

[19] C. Lee, S. Shin and J. Chung. "Network intrusion detection through genetic feature selection". In *Seventh ACIS international conference on software engineering, artificial intelligence, networking, and parallel/distributed computing (SNPD)*, pp. 109–114. IEEE Computer Society, 2006.

[20] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham and S. Sanyal. "Adaptive neuro-fuzzy intrusion detection systems". In *Proceedings of the international conference on information technology: Coding and computing (ITCC)*, vol. 1, pp. 70–74. IEEE Computer Society, 2004. DOI `http://dx.doi.org/10.1109/itcc.2004.1286428`.

[21] S. Chebrolu, A. Abraham and J. Thomas. "Feature deduction and ensemble design of intrusion detection systems". *Computers & security*, vol. 24, no. 4, pp. 295–307, 2005. DOI `http://dx.doi.org/10.1016/j.cose.2004.09.008`.

[22] Y. Chen, A. Abraham and J. Yang. "Feature selection and intrusion detection using hybrid flexible neural tree". In *Advances in neural networks (ISNN)*, vol. 3498 of *Lecture notes in computer science*, pp. 439–444. Springer Berlin / Heidelberg, 2005. DOI `http://dx.doi.org/10.1007/11427469_71`.

[23] M. Sabhnani and G. Serpen. "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context". In *International conference on machine learning, models, technologies and applications (MLMTA)*, pp. 209–215. CSREA Press, 2003.

[24] W. Hu and W. Hu. "Network-based intrusion detection using Adaboost algorithm". In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, pp. 712–717. IEEE Computer Society, 2005.

[25] D. Song, M. Heywood and A. Zincir-Heywood. "Training genetic programming on half a million patterns: An example from anomaly detection". *IEEE transactions on evolutionary computation*, vol. 9, no. 3, pp. 225–239, 2005. DOI `http://dx.doi.org/10.1109/TEVC.2004.841683`.

[26] H. Kayacik, A. Zincir-Heywood and M. Heywood. "A hierarchical SOM-based intrusion detection system". *Engineering applications of artificial intelligence*, vol. 20, no. 4, pp. 439–451, 2007. DOI `http://dx.doi.org/10.1016/j.engappai.2006.09.005`.

[27] C. Sinclair, L. Pierce and S. Matzner. "An application of machine learning to network intrusion detection". In *Proceedings of the 15th annual computer security applications conference (ACSAC)*, pp. 371–377. IEEE Computer Society, 1999. DOI `http://dx.doi.org/10.1109/csac.1999.816048`.

[28] D. Yeung and C. Chow. "Parzen-window network intrusion detectors". In *Proceedings of the 16th international conference on pattern recognition*, vol. 4, pp. 385–388. 2002.

[29] S. Mukkamala, A. Sung and A. Abraham. "Modeling intrusion detection systems using linear genetic programming approach". In *Innovations in applied artificial intelligence*, vol. 3029 of *Lecture notes in computer science*, pp. 633–642. Springer Berlin / Heidelberg, 2004. DOI `http://dx.doi.org/10.1007/978-3-540-24677-0_65`.

[30] A. Abraham and C. Grosan. "Evolving intrusion detection systems". In *Genetic systems programming*, vol. 13 of *Studies in computational intelligence*, pp. 57–79. Springer Berlin / Heidelberg, 2006. DOI `http://dx.doi.org/10.1007/3-540-32498-4_3`.

[31] S. Mukkamala, A. Sung and A. Abraham. *Intelligent systems design and applications*, chap. Intrusion Detection Using Ensemble of Soft Computing Paradigms, pp. 239–248. Springer New York, 2003.

[32] S. Peddabachigari, A. Abraham, C. Grosan and J. Thomas. "Modeling intrusion detection system using hybrid intelligent systems". *Journal of network and computer applications*, vol. 30, no. 1, pp. 114–132, 2007. DOI `http://dx.doi.org/10.1016/j.jnca.2005.06.003`.

[33] V. Golovko, P. Kachurka and L. Vaitsekhovich. "Neural network ensembles for intrusion detection". In *4th IEEE workshop on intelligent data acquisition and advanced computing systems: Technology*

and applications, 2007. IDAACS 2007., pp. 578 – 583. sept. 2007. DOI http://dx.doi.org/10.1109/idaacs.2007.4488487.

[34] H. Debar, M. Becker and D. Siboni. "A neural network component for an intrusion detection system". In *Proceedings of the IEEE computer society symposium on research in security and privacy*, pp. 240–250. IEEE Computer Society, 1992. DOI http://dx.doi.org/10.1109/risp.1992.213257.

[35] J. Cannady. "Artificial neural networks for misuse detection". In *Proceedings of the 1998 national information systems security conference (NISSC)*, pp. 443–456. Citeseer, 1998.

[36] H. Debar and B. Dorizzi. "An application of a recurrent network to an intrusion detection system". In *International joint conference on neural networks, 1992. IJCNN.*, vol. 2, pp. 478 –483 vol.2. jun 1992. DOI http://dx.doi.org/10.1109/ijcnn.1992.226942.

[37] Z. Zhang, J. Lee, C. Manikopoulos, J. Jorgenson and J. Ucles. "Neural networks in statistical anomaly intrusion detection". *Neural network world*, vol. 11, no. 3, pp. 305–316, 2001.

[38] D. Ourston, S. Matzner, W. Stump and B. Hopkins. "Applications of hidden Markov models to detecting multi-stage network attacks". In *Proceedings of the 36th annual Hawaii international conference on system sciences (HICSS)*, pp. 10–15. 2003. DOI http://dx.doi.org/10.1109/hicss.2003.1174909.

[39] C. Kruegel, D. Mutz, W. Robertson and F. Valeur. "Bayesian event classification for intrusion detection". In *Proceedings of the 19th annual computer security applications conference (ACSAC)*, pp. 14–23. 2003. DOI http://dx.doi.org/10.1109/csac.2003.1254306.

[40] J.-S. Xue, J.-Z. Sun and X. Zhang. "Recurrent network in network intrusion detection system". In *Proceedings of 2004 international conference on machine learning and cybernetics.*, vol. 5, pp. 2676 – 2679 vol.5. aug. 2004.

[41] J. Skaruz and F. Seredynski. "Recurrent neural networks towards detection of SQL attacks". In *IEEE international parallel and distributed processing symposium, 2007. IPDPS 2007.*, pp. 1 – 8. march 2007. DOI http://dx.doi.org/10.1109/ipdps.2007.370428.

[42] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo. *Applications of data mining in computer security*, chap. A geometric framework for unsupervised anomaly detection, pp. 77–101. Kluwer Academic Pub, 2002.

[43] A. Bivens, C. Palagiri, R. Smith, B. Szymanski and M. Embrechts. "Network-based intrusion detection using neural networks". In *Proceedings of the artificial neural networks in engineering conference (ANNIE)*, vol. 12, pp. 579–584. Citeseer, 2002.

[44] P. Laskov, P. Dussel, C. Schafer and K. Rieck. "Learning intrusion detection: Supervised or unsupervised?" In *Image analysis and processing (ICIAP)*, vol. 3617 of *Lecture notes in computer science*, pp. 50–57. Springer Berlin / Heidelberg, 2005.

[45] M. Al-Subaie and M. Zulkernine. "Efficacy of hidden Markov models over neural networks in anomaly intrusion detection". In *30th annual international computer software and applications conference, 2006. COMPSAC '06.*, vol. 1, pp. 325 –332. sept. 2006. ISSN 0730-3157. DOI http://dx.doi.org/10.1109/compsac.2006.40.

[46] M. Al-Subaie and M. Zulkernine. "The power of temporal pattern processing in anomaly intrusion detection". In *IEEE international conference on communications, 2007. ICC '07.*, pp. 1391 –1398. june 2007. DOI http://dx.doi.org/10.1109/icc.2007.234.

[47] N. Chowdhury and M. Kashem. "A comparative analysis of feed-forward neural network & recurrent neural network to detect intrusion". In *International conference on electrical and computer engineering, 2008. ICECE 2008.*, pp. 488 –492. dec. 2008. DOI http://dx.doi.org/10.1109/icece.2008.4769258.

[48] P. Kachurka and V. Golovko. "Neural network approach to real-time network intrusion detection and recognition". In *IEEE 6th international conference on intelligent data acquisition and advanced computing systems (IDAACS), 2011*, vol. 1, pp. 393 – 397. sept. 2011. DOI http://dx.doi.org/10.1109/idaacs.2011.6072781.

[49] 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*, vol. 3, no. 4, pp. 262–294, 2000. DOI http://dx.doi.org/10.1145/382912.382923.

[50] M. Mahoney and P. Chan. "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection". In *Recent advances in intrusion detection*, vol. 2820 of *Lecture notes in computer science*, pp. 220–237. Springer Berlin / Heidelberg, 2003.

[51] M. Sabhnani and G. Serpen. "Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set". *Intelligent data analysis*, vol. 8, no. 4, pp. 403–415, 2004.

[52] S. Brugger and J. Chow. "An assessment of the DARPA IDS evaluation dataset using snort". Tech. Rep. CSE-2007-1, Department of Computer Science, University of California, Davis (UCDAVIS), 2005.

[53] M. Roesch. "Snort–lightweight intrusion detection for networks". In *Proceedings of the 13th USENIX conference on system administration*, pp. 229–238. Seattle, Washington, 1999.

[54] M. Tavallaee, E. Bagheri, W. Lu and A. A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set". In *IEEE symposium on computational intelligence for security and defense applications*, Cisda, pp. 1–6. IEEE, Jul. 2009. DOI http://dx.doi.org/10.1109/cisda.2009.5356528.

[55] M. Jordan. "Attractor dynamics and parallelism in a connectionist sequential machine". In *Proceedings of the eighth annual conference of the cognitive science society*, pp. 531–546. 1986.

[56] J. Elman. "Finding structure in time". *Cognitive science*, vol. 14, pp. 179–211, 1990. DOI http://dx.doi.org/10.1207/s15516709cog1402_1.

[57] R. Williams and D. Zipser. "A learning algorithm for continually running fully recurrent neural networks". *Neural computation*, vol. 1, pp. 270–280,

1989. DOI http://dx.doi.org/10.1162/neco.1989.1.2.270.

[58] R. Williams and D. Zipser. *Backpropagation: Theory, architectures, and applications*, chap. Gradient-based learning algorithms for recurrent networks and their computational complexity, pp. 433–486. Lawrence Erlbaum, 1995.

[59] S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber. "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies". In *A field guide to dynamical recurrent neural networks*. IEEE Press, 2001.

[60] A. Graves and J. Schmidhuber. "Framewise phoneme classification with bidirectional LSTM networks". In *Proceedings of the international joint conference on neural networks*, vol. 4, pp. 2047–2052. 2005. DOI http://dx.doi.org/10.1109/ijcnn.2005.1556215.

[61] A. Shiravi, H. Shiravi, M. Tavallaee and A. A. Ghorbani. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". *Computers & security*, vol. 31, no. 3, pp. 357–374, May 2012. DOI http://dx.doi.org/10.1016/j.cose.2011.12.012.

[62] L. Pesce and C. Metz. "Reliable and computationally efficient maximum-likelihood estimation of proper binormal ROC curves". *Academic radiology*, vol. 14, no. 7, pp. 814–829, 2007. DOI http://dx.doi.org/10.1016/j.acra.2007.03.012.