Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Network anomaly detection with the restricted Boltzmann machine

Ugo Fiore [a,*], Francesco Palmieri [b], Aniello Castiglione [c], Alfredo De Santis [c]

[a] *Centro di Ateneo per i Servizi Informativi, Università di Napoli Federico II, Napoli, Italy*
[b] *Dipartimento di Ingegneria dell'Informazione, Seconda Università di Napoli, Aversa, Italy*
[c] *Dipartimento di Informatica, Università di Salerno, Fisciano, Italy*

A R T I C L E I N F O

Article history:
Received 30 September 2012
Received in revised form
23 November 2012
Accepted 26 November 2012

Keywords:
Anomaly detection
Restricted Boltzmann machine
Semi-supervised learning
Intrusion detection
Energy-based models

A B S T R A C T

With the rapid growth and the increasing complexity of network infrastructures and the evolution of attacks, identifying and preventing network abuses is getting more and more strategic to ensure an adequate degree of protection from both external and internal menaces. In this scenario many techniques are emerging for inspecting network traffic and discriminating between anomalous and normal behaviors to detect undesired or suspicious activities. Unfortunately, the concept of normal or abnormal network behavior depends on several factors and its recognition requires the availability of a model aiming at characterizing current behavior, based on a statistical idealization of past events. There are two main challenges when generating the training data needed for effective modeling. First, network traffic is very complex and unpredictable, and second, the model is subject to changes over time, since anomalies are continuously evolving. As attack techniques and patterns change, previously gained information about how to tell them apart from normal traffic may be no longer valid. Thus, a desirable characteristic of an effective model for network anomaly detection is its ability to adapt to change and to generalize its behavior to multiple different network environments. In other words, a self-learning system is needed. This suggests the adoption of machine learning techniques to implement semi-supervised anomaly detection systems where the classifier is trained with "normal" traffic data only, so that knowledge about anomalous behaviors can be constructed and evolve in a dynamic way. For this purpose we explored the effectiveness of a detection approach based on machine learning, using the Discriminative Restricted Boltzmann Machine to combine the expressive power of generative models with good classification accuracy capabilities to infer part of its knowledge from incomplete training data.

1. Introduction

The main goal of a network anomaly detection system is to discriminate the occurrence of hostile activities from the normal network traffic, and such analysis must be accomplished in a sufficiently flexible and effective way to keep up with the continuously evolving world of cybersecurity where new, previously unknown, anomalies can continuously emerge over time. In doing this, it must either try to model any kind of attack or anomalous event that can affect the network (there are thousands of known ones) or simply construct a sufficiently general model describing the normal traffic.

Such model is usually built on the basis of training data, and used in classifying previously unseen or suspicious events. Classification is the fundamental task in unattended detection, by which the system "learns" to automatically recognize complex traffic patterns, to distinguish between different events based on the corresponding patterns, and to make "intelligent" decisions. Specific machine learning techniques, such as Neural Networks or Support Vector Machines can be used to develop a generalization capability from training data needed to correctly classify future data as normal or abnormal. These resulting approaches can be categorized as generative or discriminative. A generative approach builds a model solely based on normal training examples and evaluates each testing case to see how well it fits the model. A discriminative approach, on the other hand, attempts to learn the distinction between the normal and abnormal classes. Thus, based on the characteristics of training data used to build the model, anomaly detection can be divided into three broad classes [1]:

- *Supervised anomaly detection*: In this class, a training set containing labeled instances for both the normal and anomalous class is available.
- *Semi-supervised anomaly detection*: The training here only contains instances for the normal class. Anything that cannot be characterized as normal is thus marked as anomalous.
- *Unsupervised anomaly detection*: No training set is available nor is it needed.

Obviously, the quality of classification crucially depends on the accurateness and comprehensiveness of the model and hence of

* Corresponding author. Tel.: +39 392 8888562.
E-mail addresses: ufiore@unina.it, ugo.fiore@unina.it (U. Fiore).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.11.050

the training data on which the model is built. If a complete set of pre-classified or "labeled" categories of normal (or anomalous) behavior is included in the training set (which can be difficult to achieve and even harder to maintain), all the corresponding (or "matching") instances will be correctly classified. But if some of such categories are not described in the training data, the corresponding instances may be classified incorrectly. In particular, anomalous events are harder to describe, partly because of their negative definition (*a-nomalous* literally *means* non-normal). Anomalous events both appear less frequently than normal events and embrace a huge variety of aspects. In fact, one technique for producing a labeled training set consists in creating anomalies artificially, injecting anomalous events in a stream containing normal data. But the description of anomalies obtained in such a way is forcibly limited to the scope and characteristics of the artificial anomalies used. Labeled anomalous data gathered from operational, "real-world" networks are not readily available, for a number of reasons, including the sheer amount of effort needed to produce such data, and the reluctance of network administrators to divulge data that could compromise the privacy of their clients or exfiltrate privileged information about the internal structure of their networks. The immense amount of data to be potentially examined, combined with their complexity and with the level of expert knowledge that would be needed for the analysis is a strong motivator for making the training process as independent as possible from the availability of labeled anomalous data. Furthermore, these data are generated mainly through human intervention, as soon as the community becomes aware of a new menace and detailed information about the attack dynamics and behavior becomes available. This may require a not negligible time that is clearly unacceptable when real-time or timely response to anomalies is strictly necessary. Consequently, anomaly detection systems should avoid being limited by the knowledge of any predefined set of anomalies and should be able to flexibly recognize/classify any unknown event affecting the network operations according to a self-learning semi-supervised or better unsupervised detection approach, so that the knowledge of traffic behavior on which the model is based can be progressively constructed and dynamically change/evolve over time. However, building and training *in situ* such a generative self-learning model is a lengthy and costly process. Even when the semi-supervised model is used, the overall result will depend on the completeness of the training data, since it is difficult to represent all the features that normal behavior can have. Thus, in this work we focus on semi-supervised anomaly detection, with a perspective aiming at investigating whether normal traffic behavior (and, conversely, anomalous behavior) shares some inherent similarity that we can use to characterize it. The purpose of this analysis is not to concentrate on a near real-time intrusion detection and reaction system but, in a medium-term perspective, to work towards a better and more adequate description of network traffic, also aiming at being as adaptive as possible. The tool which has been selected for this analysis is the Discriminative Restricted Boltzmann Machine, a network of stochastic neurons behaving according to an energy-based model. These networks couple the ability to express much of the variability of data, given by generative models, with the good classification accuracy derived from discriminative classifiers. The main advantages of this approach are that in line of principle it is not restricted to any specific environment, or *a priori* knowledge base, and that it can enable the detection of any type of unknown anomalous events, being effective in coping with the so-called *zero-day* attacks.

In Section 2, related work is reviewed. An introductory summary of classical neural networks, energy-based models, and Boltzmann Machines is the subject of Section 3. The proposed model and its assumptions are detailed in Section 4. Sections 5 and 6 are dedicated to the description of the experiments and the discussion of their results. Section 7 contains some concluding remarks and directions for future research.

2. Related work

Anomaly detection in computer networks is a long-established area, with more than 40 years of evolution [2] and contributions that have explored many approaches [1,3]. In a wider perspective, other researchers have considered the time-dependent connection between events, focusing on the casual relationship between an event and its consequences. For example, the problem of correlating events from different sources to isolate the root cause has been investigated in [4]. The study of effects produced by hidden dynamics has been the object of works based on nonlinear analysis, first targeted to traffic classification [5] and then focused on the specific issue of network anomaly detection [6]. Feature-modeling anomaly detection techniques such as FRaC [7] focus instead on the linkage between individual features and attempt to build predictive models for each feature, based on the others. Features that deviate from this prediction indicate an anomaly. Semi-supervised learning has received attention by the research community recently. Mao et al. [8] have proposed a co-training framework based on multi-view data, semi-supervised learning and active learning. Their method requires user intervention. Chen et al. [9] evaluated the application of spectral graph transduction and Gauss random fields to the detection of unknown attacks. Besides classification, they also proposed a semi-supervised method for clustering. An unsupervised clustering algorithm based on competitive learning neural network is described in [10], where instability is reduced by means of a reward-punishment update rule.

3. Background

3.1. Anomaly detection

From a theoretical point of view our network anomaly detection problem can be formulated as follows [11].

A collection of traffic data measurements is described by a scalar time series $\{x_t\}_{t=1}^{T}$ governed by a probability distribution $p(\cdot)$. Although all these measurements are associated to the occurrence of specific events within the event space S, the correspondence between them may not be known in advance. We are interested in partitioning the event space S into two subspaces corresponding to the normal and the anomalous network traffic conditions. Also, we need to infer the membership of a particular event in one of the above subspaces starting from the corresponding time series values. To accomplish this task, since the probability distribution p describing the behavior of the time series is unknown, we can use a mechanism enabling the reconstruction of its volumetric representation from the collection $\{x_t\}_{t=1}^{T}$. A general approach to the problem of identifying this representation is based on building a Minimum Volume Set (MVS) characterized by a probability mass $0<\beta<1$ associated to the distribution p for a volume measure μ [12], that is:

$$G_{\beta}^{*} = \arg\min\{\mu(G) : p(G) \geq \beta, \quad G \text{ measurable}\} \tag{1}$$

In the most of the common cases μ can be chosen to be the *Lebesgue* measure, although such technique extend easily to other measures. The parameter β can be chosen by the user to reflect a desired false alarm rate of $1-\beta$.

These minimum volumes summarize the regions of greatest probability mass of the distribution p, and are useful for detecting

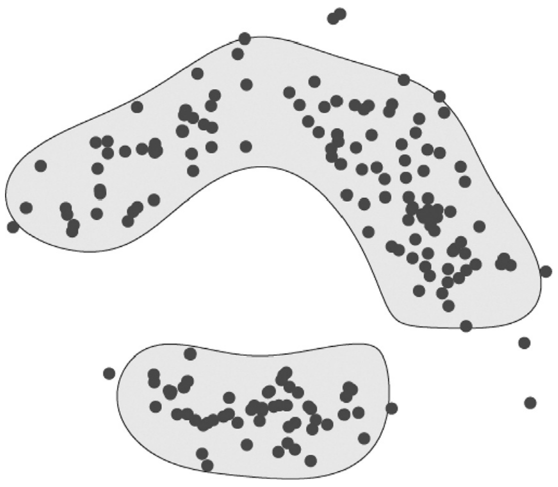

Fig. 1. Minimum volume sets example with $\beta=0.9$.

anomalies and constructing confidence regions. Hence, online evaluation of MVSs satisfying Eq. (1) inherently allows the identification of the highest density regions where the mass of p is concentrated. All the points falling outside these regions and the associated events can be declared as anomalous. It can be observed that, if p is a multivariate Gaussian distribution and μ is the Lebesgue measure, then the MVSs are represented as ellipsoids (see Fig. 1).

The most common available methods for estimating G come from the use of self-learning systems, whose fundamental solutions can be found in the world of neural networks.

3.2. Neural network-based learning models

In many problems of practical interest, the entities to be modeled are complex, with statistical regularities, characteristics, and interactions between them. When modeling such a system as a network of interconnected units, this complexity reflects, in turn, into an intricate mesh of relationships between units. These units can be arranged in a few layers (even one) with many units, or in multiple smaller layers. Theoretical results suggest [13] that functions admitting a compact representation in an architecture with a given number of layers would require a very large number of units in shallower architectures. However, finding the appropriate values for the parameters in these architectures is far more challenging than in the single-layer case, because there is no information about the expected behavior of intermediate layers [14]. In classical neural networks, training algorithms akin to backpropagation only try to model the dependence of the output from the input. Restricted Boltzmann Machines (RBMs), instead, are networks of stochastic neurons that can be trained in a greedy fashion. Deep Belief Networks [15,14] are obtained by stacking RBMs on one another so that the input to one layer is given by the hidden units of the adjacent layer, as if they were data, and adding a last discriminative layer. A hybrid system based on a combination of Support Vector Machines (SVMs) and DBNs has been proposed [16], with a primary focus on feature reduction.

3.2.1. Classical neural networks

The term “neural network” has its origins in attempts to find mathematical representations of information processing in biological systems [17]. Modeling cognitive processes is at the basis of building intelligent information systems capable, to an extent, of activities such as analysis, reasoning, interpretation, and forecasting [18].

Three dimensions are typically needed to describe a neural network:

- *Architecture*: Specifies what variables are involved in the network and their topological relationships, for example, the variables involved in a neural network might be the weights of the connections between the neurons, along with the activities of the neurons.
- *Activity rule*: Most neural network models have short timescale dynamics: local rules define how the activities of the neurons change in response to each other. Typically the activity rule depends on the weights (the parameters) in the network.
- *Learning rule*: The learning rule specifies the way in which the neural network's weights change with time. This learning is usually viewed as taking place on a longer time scale than the time scale of the dynamics under the activity rule. Usually the learning rule will depend on the activities of the neurons. It may also depend on target values supplied by a teacher and on the current value of the weights.

The multilayer perceptron is perhaps the simplest specific class of neural network, yet it has proven to be of great practical value. A multilayer feedforward network consists of a set of neurons that are logically arranged in layers. There are at least two layers, the input and output ones. The outputs (activations) of neurons in the input layer are determined by the network's input. Usually, one or more hidden layers are located between the input and output layers. Data flow in one direction (hence the term feedforward): from the outputs of the previous layer to the inputs of the next layer, so the output of the network is a function of its inputs. The output of such a network can be computed in a single deterministic pass. Every single neuron, represented in Fig. 2 as a circle, has n inputs $\mathbf{x}=(x_1,\ldots,x_n)$, plus another dummy input always valued at 1, acting as a bias b. Every neuron is characterized by n weights $\mathbf{w}=(w_1,\ldots,w_n)$ and an activation function f that is applied to the weighted sum of the inputs, yielding as result:

$$f\left(\sum_{i=1}^{n} w_i\cdot x_i+b\right)=f(\mathbf{w}^T\mathbf{x}+b).$$

The operational behavior of the network is principally determined by the weights: the exact shape of the activation function only affects the expressive power of the network in a minor way; instead it influences the convergence of the training procedure. The simplest form of nontrivial activation function is the threshold function $f_a=1$ if $x\geq a$ and 0 otherwise. In general, a differentiable activation function is preferred. A common family of activation functions is the sigmoid functions. An example of sigmoid function is the logistic sigmoid:

$$\mathrm{sigm}(x)=\frac{1}{1+e^{-x}}$$

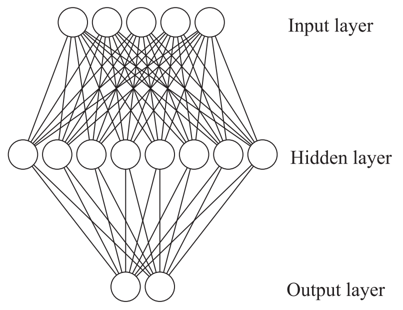

Fig. 2. A typical three-layer network, with five inputs, eight hidden units, and two outputs.

whose derivative is always positive and given by the logistic equation

$$\frac{d}{dx}f(x) = f(x)\cdot(1-f(x))$$

Other examples of sigmoid functions are the hyperbolic tangent and the scaled arctangent. Clearly none of these sigmoid functions can reach their respective theoretical bounds, so usually one assumes that a neuron is activated when its output value is about 0.9 and that a neuron is turned off when its output is about 0.1. It is reasonable, instead, to use the extremes as inputs to the network. In the field of neural networks there are very few theoretical strict rules on how to define the architecture of the network. One of these says that all problems that can be solved by a neural network do not require more than two layers. In practice, for the majority of the problems one hidden layer of hidden neurons is enough.

A good method for the determination of the hidden layers is to start with a network with a single hidden and few hidden neurons. If the networks perform poorly, increase the number of hidden neurons. When the number of neurons gets large and the performances of the network do not improve, it can be more productive to introduce a new hidden layer and reduce the total number of hidden neurons. Choosing the right number of neurons for a hidden layer is also not trivial. If there are too few neurons, the network has not enough resources to learn the general features of the training population. Using too many neurons increases the training time, and this not necessarily yields a corresponding increase of performance. In fact, too many neurons can cause over-fitting: the network can capture a large quantity of information not only about important general features, but also specific features of the training set. In passing, one should note that reasoning about the choice of a sensible number of hidden neurons heavily depends on the task the network is designed for: practical rules valid in discriminative machine learning may be inadequate in the generative case [19]. In the first context, a training sample imposes on the parameters a constraint given by the number of bits needed to specify the label. As labels typically carry a few bits of information, using more parameters than training samples will typically cause severe over-fitting. In generative models, what matters is the entire sample. Thus, it is the information carried by a sample that determines how much constraint is added on the parameters by each training sample. Redundancy in the entire training set will reduce the total amount of constraint, still this can be much larger than the number of bits it takes to identify a label.

Basically, there are two ways to train a neural network. The most common is supervised training. A training set of sample data that completely specifies the inputs, along with the desired output is needed. Once the training set is defined, it is presented to the network, one sample at a time. The error between the desired output and the network's output is evaluated and the weights of the network are updated in order to reduce the error. Each cycle through the training set updating the weights is called an *epoch*. This cycle is repeated until the wanted precision is attained or a predetermined number of iterations is reached. The other training method is unsupervised training. As in supervised training, there is a training set, but in this case the desired outputs are not needed. The process of training the network is to let it discover the main features of the training set, and using these features, to group the inputs into classes that the network finds distinct. A hybrid, training method, called Reinforcement Learning is unsupervised in that the desired outputs are unknown and, at the same time, it is supervised in that when the network evaluates an output, it is told whether its response is good or bad.

The most common training algorithm is the back-propagation of errors [20]. The name comes from the fact that output-layer errors are back-propagated through the network. Its most basic form is a gradient descent algorithm. After an initial random assignment of weights, the gradient of the error with respect to weights is computed, and a step is taken to reduce the error. The length of the step, called learning rate, will directly influence the convergence of the algorithm. Too small a step slows convergence down, whereas with a larger step there will be wild jumps and the algorithm may never converge. In addition, the training algorithm should avoid getting stuck in local minima.

It has been observed [21] that standard gradient descent is not a good performer when the number of hidden layer gets high. On the other hand, networks with few hidden layers have less expressive power than networks that have many of these layers [13]. In the same study [21], standard activation functions such as sigmoids and *tanh* are shown to be outperformed by the softsign function:

$$f(x) = \frac{x}{1+|x|}.$$

The softsign has the same range of the *tanh* and is similar to it, but has a softer asymptotic behavior.

3.2.2. *Boltzmann machine*

General (unrestricted) Boltzmann machines belong to the class of stochastic Energy-Based Models. In energy-based models [13], an *energy* is associated to each configuration (state) of the system under analysis. To describe such system state, observable variables $\mathbf{v} = (v_1, \dots, v_n)$ are usually complemented by non-observable (*hidden*) variables $\mathbf{h} = (h_1, \dots, h_m)$. Hidden variables participate in determining the evolution of the system over time, according to a set of (unknown) differential equations, but remain impermeable to observation. The relation between visible and latent variables is bidirectional, i.e., the former are influenced by the latter, but also influence them. Pictorially, this is represented with a graph similar to that in Fig. 2. The probability of a state $P(\mathbf{v},\mathbf{h})$ depends only on the energy of that state $\mathcal{E}(\mathbf{v},\mathbf{h})$, with a Boltzmann distribution function:

$$P(\mathbf{v},\mathbf{h}) = \frac{\exp(-\mathcal{E}(\mathbf{v},\mathbf{h}))}{\sum_{\mathbf{v},\mathbf{h}}\exp(-\mathcal{E}(\mathbf{v},\mathbf{h}))} \tag{2}$$

where the normalization factor at the denominator is usually denoted by Z (this comes from Statistical Mechanics, where Z is termed *Partition Function*[1]), i.e.,

$$Z = \sum_{\mathbf{v},\mathbf{h}}\exp(-\mathcal{E}(\mathbf{v},\mathbf{h})).$$

$P(\mathbf{v},\mathbf{h})$ is the joint probability of the visible and the hidden variables, that is, the probability of every possible pair of a vector of visible units and a vector of hidden units. Since only the $\mathbf{v}$ are actually observed, we are interested in the marginal distribution $P(\mathbf{v})$ summing over all configurations of the latent variables:

$$P(\mathbf{v}) = \sum_h P(\mathbf{v},\mathbf{h}) = \sum_h \frac{\exp(-\mathcal{E}(\mathbf{v},\mathbf{h}))}{Z}. \tag{3}$$

Observing that

$$Z = \sum_{x,h}\exp(-\mathcal{E}(x,h)) = \sum_x\sum_h \exp(-\mathcal{E}(x,h)),$$

Eq. (3) can be transformed into a form similar to Eq. (2) by introducing the notion of *Free Energy* $\mathcal{F}$:

$$\exp(-\mathcal{F}(x) = \sum_h \exp(-\mathcal{E}(x,h)), \tag{4}$$

so that

$$P(x) = \frac{\exp(-\mathcal{F}(x))}{\sum_x \exp(-\mathcal{F}(x))} = \frac{\sum_x \exp(-\mathcal{F}(x))}{Z}.$$

[1] Z comes from the German word *Zustandssumme*, sum over states.

To minimize the negative data log-likelihood, its gradient with respect to the parameters of the model, θ, must be considered:

$$-\frac{\partial \log P(\mathbf{v})}{\partial\theta} = \sum_h P(\mathbf{h}|\mathbf{v})\frac{\partial\mathcal{E}(\mathbf{v},\mathbf{h})}{\partial\theta} - \sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v},\mathbf{h})\frac{\partial\mathcal{E}(\mathbf{v},\mathbf{h})}{\partial\theta}$$
$$= \mathbb{E}\left[\frac{\partial\mathcal{E}(\mathbf{v},\mathbf{h})}{\partial\theta}\middle|\mathbf{v}\right] - \mathbb{E}\left[\frac{\partial\mathcal{E}(\mathbf{v},\mathbf{h})}{\partial\theta}\right]$$

where $\mathbb{E}$ denotes the expectation operator. To overcome the difficulty of calculating this gradient analytically (the expectation involves all possible input configurations), *Restricted Boltzmann Machines*(RBMs) were introduced.

In RBMs, no intra-layer links are allowed [19]. A hidden variable still does influence visible variables (and vice versa) but is not allowed to depend on the value of another hidden variable. The same is true for the visible variables. An RBM can be represented by a network of stochastic binary neurons, whose states are observable, which are connected to stochastic, unobservable, hidden units. Connections are bidirectional. Neurons have two states and will be activated with a probability which is a smooth function. The energy function of a RBM is

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{h}^T\mathbf{W}\mathbf{v} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h}, \tag{5}$$

where $\mathbf{h}^T$ is the transpose of $\mathbf{h}$, and $\mathbf{b}$ and $\mathbf{c}$ are the biases, respectively, of the visible and the hidden units. The structure of a RBM, with no intra-layer dependence, allows us to write

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) \quad \text{and} \quad p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}). \tag{6}$$

The most commonly used case involves binary units. However, RBMs can be generalized to unrestricted real-valued inputs. With binary units, Eqs. (6) can be rewritten as

$$P(v_i = 1|\mathbf{h}) = \text{sigm}\left(b_i + \sum_j w_{j,i}h_j\right)$$
$$ptP(h_j = 1|\mathbf{v}) = \text{sigm}\left(c_i + \sum_i w_{j,i}v_j\right).$$

This results in a parametric model of the joint distribution between hidden variables and the observable inputs. The parameters are $\Theta = (\mathbf{W},\mathbf{b},\mathbf{c})$. Training means finding the values of these parameters that correspond to desirable values for the energy, usually such that the energy is minimized. Thus, a possible training strategy may aim at minimizing the log-likelihood of the training data that is estimating its gradient with respect to the model parameters. While an exact computation is intractable, the gradient can be estimated using a method called *contrastive divergence* (CD). CD replaces the expectation with a sample taken over a limited number of Gibbs sampling steps. Samples of $p(v)$ can be obtained by running a Markov chain to convergence, by using Gibbs sampling as the transition operator. Visible units are sampled simultaneously, given fixed values of the hidden units (recall that visible units are conditionally independent).

Similarly, hidden units are sampled simultaneously, given the visible variables. At iteration i, the hidden units $\mathbf{h}^{(i)}$ are sampled, i.e., chosen at random with probability $\text{sigm}(\mathbf{W}\mathbf{v}^{(i-1)} + \mathbf{c})$. This is called *positive phase*, because it decreases the energy (increases the probability) of training data. Then, the network is allowed to set the values of the visible variables. This also corresponds to random sampling from the distribution of $p(\mathbf{v}|\mathbf{h}^{(i-1)})$ and is called *negative phase*, because it increases the energy (decreases the probability) of samples generated by the model. In theory, each parameter update in the learning process would require waiting for convergence of one Markov chain. In practice, after initializing the chain without using a random value but with a training example, even a single step of Gibbs sampling has been shown to yield good results.

Training is obtained by cycling through the training data and updating the parameters with the values obtained by CD multiplied by the learning rate λ.

3.2.3. *Discriminative restricted Boltzmann machines*

The discriminative RBM is a promising tool in statistical machine learning [22]. In contrast to the Restricted Boltzmann Machine (RBM), which is a generative classifier aimed at producing a model that describes inputs, capturing as much of their variational potential as possible, the discriminative RBM aims at combining the descriptive power with a sharp classification ability. In order to make an RBM operate in a supervised fashion, we introduce an additional input containing the targets. The datasets will thus be structured as sequences of pairs $(\mathbf{v},y)$ comprising an input vector $\mathbf{v}$ and a class $y\in\{1,\dots,C\}$. It is actually convenient to "vectorize" the class, introducing a vector $\mathbf{y}$ whose j-th component is $\delta_{j,C}$, using the Kronecker delta.[2]

In a DRBM, the joint distribution of Eq. (2) becomes then

$$p(\mathbf{v},\mathbf{h},y) \propto \exp(-E(\mathbf{v},\mathbf{h},y))$$

and the energy of Eq. (5) becomes

$$E(\mathbf{v},\mathbf{h},y) = -\mathbf{h}^T\mathbf{W}\mathbf{v} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{d}^T\mathbf{y} - \mathbf{h}^T\mathbf{U}\mathbf{y}.$$

What matters most in a classification task is not a correct joint distribution, but the accuracy for classification. In Discriminative RBMs, we optimize directly $p(y|\mathbf{v})$ instead of $p(y,\mathbf{v})$. The loss function will thus be

$$\ell_{\text{discr}} = \sum_{i=1}^{N_T} \log p(y^{(i)}|\mathbf{v}^{(i)}) \tag{7}$$

and the gradient can be computed exactly

$$\frac{\partial \log p(y_i|v_i)}{\partial\theta} = \sum_j \text{sigm}(o_{y,j}(v_i))\frac{\partial o_{y,j}(v_i)}{\partial\theta} - \sum_{j,y*}\text{sigm}(o_{y*,j}(v_i))\frac{\partial o_{y*,j}(v_i)}{\partial\theta}$$

where $o_{y,j}(\mathbf{v}) = c_j + \sum_k W_{j,k}v_k + U_{y,j}$.

As it can be seen, DRBMs have many things in common with feedforward neural networks. The added value of an energy-based model, such as the one at the basis of RBMs, is that it introduces generative capabilities in an explicit, controllable, way [22].

Whereas the use of binary variables has been previously assumed, RBMs can easily accommodate different variables. For example, units that can assume one state out of K distinct states are dealt with *softmax* units, whose probability of activation is

$$P_j = \frac{e^{x_j}}{\sum_{i=1}^K e^{x_i}}$$

4. DRBM for semi-supervised anomaly detection

Even when we are able to observe anomalous data, it is often difficult or somewhat impossible to identify the anomalous training samples *a priori*. In other words, we must be able to detect anomalies without knowing what they look like. In semi-supervised anomaly detection systems, the classifier is trained with data in the normal class only. In a sense, this presents an interesting analogy with the *self-model* of the human immune system [23, Chapter 7]. The immune system has indeed inspired a rich line of research in network security (see, for example, [24,25]). To be effective, the immune defenses should be capable to attack

[2] $\delta_{i,j} = 1$ iff $i = j$, otherwise 0.

foreign cells. At the same time, attacking all cells indiscriminately would mean bringing harm exactly to the organism (the *self*) which the immune system is intended to protect (as it happens in autoimmune diseases). Through processes known as positive and negative selection, T-cells lymphocytes are selected in a gland known as thymus so that they react to proteins which are found on all cells of the body (so that body cells are not felt as "too foreign"), but not react too much as to attack them (hence acting against the organism itself). Anything that is not known (not corresponding to the image of self) is labeled as anomalous. Naturally, this does not mean that everything not matching with the model *is* actually anomalous. Besides the effects of the dynamic, adaptive nature of the network traffic (and of biological systems), the possibility that the model does not completely describe all the instance of normal behavior (a task that can be overwhelmingly difficult and lengthy) should be considered. This approach focuses on understanding the process generating the traffic data, and is very useful to ease the construction of the model of normal or anomalous behavior, since it allows the system to learn about the occurrence of a specific phenomenon via inductive inference based on observing the empirical data that represent incomplete information about the phenomenon itself.

Even though training data includes only normal events, and thus any deviation from such "normality" would trigger an alert, the overall performance of classifiers in this group would not necessarily be remarkable. In fact, the very notion of normality is vague in itself and it is hard to model all the facets that normality can have regardless of the effort that can be put into making the collection of training data as extensive and comprehensive as possible, it is perfectly reasonable that many instances belonging to the normal class show distinctive features that separate them from those found in the training set and thus are flagged as anomalous. Thus, a high number of classification errors can be expected of these systems.

Remark 1. In semi-supervised anomaly detection, classification errors most likely are normal events incorrectly classified as anomalous, i.e., false positives.

It should be noted that false positives, although less harmful than false negatives, are undesirable in the context of anomaly detection. Events flagged as anomalous should be analyzed more closely (perhaps by humans), thus absorbing inordinate amount of resources. Starting from these considerations, some questions immediately arise. Can we do better? Is there some form of "inherent" similarity between all normal traffic? Do all normal traffic flows share some common value ranges for a significant set of features, as well as correlation and sequencing between them? Notably, a recent paper confirms that anomalous traffic has intrinsic properties regarding self-similarity which are different from those of normal traffic [26]. The idea that we intend to test is whether there is a deep similarity between normal behaviors, so that a learning model with high expressive power can describe all the nuisances of normal traffic when comparing it against unseen anomalous traffic. The DRBM, which combines the expressive power of generative models with good classification accuracy, has been selected as a tool to test this hypothesis.

4.1. Assumptions

The following assumptions are taken as valid hereinafter:

- No packet payload is available. There are many reasons for that: to begin with, some data sources that can likely be used for the purpose of this analysis natively do not contain any payload. Typical examples include NetFlow records and Network Address and Port Translation (NAPT) log files [27]. In addition, even if the data source is a packet capture, data can be limited to protocol headers or be truncated to a few bytes. This may happen for privacy concerns (users are very sensitive about the content of their traffic being captured) or simply for the sake of saving storage space.
- It is anticipated that the parameters characterizing a baseline may change over a sufficient long time. Thus, training should be repeated at regular intervals. This theme will be the object of future research and will not be further expanded here.

5. Experiments

A set of experiments was performed by testing the trained network against the processed data from two real-world traces, each one describing all the traffic involving a specific workstation for 24 h. Traffic on one host was normal, while the other one had been infected by a bot [29]. The bot communicated via IRC with other hosts in its botnet. All this traffic was defined as anomalous. This had considerable implications, which will be discussed later. Anomalous connections were flagged manually. The "clean" dataset contains 12,056 connections, while of the 12,317 connection in the second dataset, 4182 were anomalous. The first experiment in this set had the goal of testing the accuracy of the DRBM to recognize anomalous traffic on real data.

Another experiment involved the "cross" use of training and test data: the DRBM was trained with KDD data (using the 10% training KDD dataset), and tested against real data. All tests were repeated 10 times, randomizing the order of the test data.

The KDD '99 dataset[3] is the data set used for the *Third International Knowledge Discovery and Data Mining Tools Competition*, which was held in conjunction with the *Fifth International Conference on Knowledge Discovery and Data Mining* (KDD '99). The raw training data amounts to about 4 GB of compressed packet capture data from seven weeks of network traffic. This was processed into 4,898,431 connection records, with a set of 41 features describing various aspects. Similarly, the two weeks of test data yielded 2,984,154 (unlabeled) connection records. A smaller training dataset, containing 10% of the connections (but representative of the anomalies), is also available (Table 1). Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes. Attacks fall into four main categories:

(i) *DoS*: denial-of-service, e.g., SYN flood: the attack objective is to prevent legitimate users from accessing a service;
(ii) *R2L*: unauthorized access from a remote machine, e.g., guessing password: the attack objective is to get access to a machine from outside;
(iii) *U2R*: unauthorized access to local superuser (root) privileges, e.g., "buffer overflow" attacks: the attack objective is to obtain elevated privileges;
(iv) *Probe*: reconnaissance and other probing, e.g., port scanning – the attack objective is to gain information about target host or networks.

We have to chose how many features must be used for classification, and how many patterns are needed for each class. In our experiments, we use only part of the total training data, namely, those containing "normal" connections. This leads to 97,278 instances.

Remark 2. While the KDD dataset has been used in a substantial number of research works, it has been heavily criticized [28], as being excessively "artificial", in the sense that normal traffic into

[3] http://www.sigkdd.org/kddcup/index.php?section=1999

which anomalies were injected does not resemble real traffic; this may add an undesired bias to the data, weakening the results deriving from experiments upon them.

5.1. Features

In order to construct the needed traffic measurement space, we have to determine a basic set of features describing the traffic pattern in a specific time interval (the observation epoch). Let F denote the feature space describing the network traffic. We use a multi-dimensional feature vector $\mathbf{f} \in F$ built from the fundamental traffic observations (e.g., the overall byte rate or the average packet inter-arrival time) by computing on each epoch some traffic behavior discriminators.

The number of traffic features that can be simultaneously evaluated introduces great control granularity in the detection process by adding multiple points of observation, and hence new dimensions in the decision space that can ease correlation and inference activities in the DRBM-based binary classification process. Having multiple different observations associated to the individual traffic components may also be helpful in spotting and describing the nature and behavior of the observed anomalous phenomena (e.g., protocols affected, transport facilities used, traffic volumes distribution). This last issue can be of fundamental importance in the development of countermeasures or reaction strategies that can be a very interesting subject for further research.

In the experiments, 28 out of 41 features were used, namely, those related to network traffic. These features are listed in Table 2.

Most features take a value in a numerical range, and are indicated as "continuous". Other features are nominal, i.e., they assume one value from a discrete set of possible values. These features are tagged as "discrete". Note that some features are derived, i.e., calculated starting from the values of other features. Features can be lumped into three groups: the first group (lines 1–7 of Table 2) includes basic features of individual TCP connections; the second group (lines 8–18 of Table 2) includes time-based features computed over a two-second interval a set of counters for connection having specific characteristics; the third group (lines 19–28 of Table 2) includes derived host-based features, which were computed using, instead of a time window, a window of 100 connections to the same host. Within the second group, the "same host" features examine only the connections in the past 2 s that have the same destination host as the current connection, and calculate statistics related to protocol, service or behavior. The similar "same service" features examine only the connections in the past 2 s that have the same service as the current connection.

5.2. Preprocessing

A preprocessing step was needed to transform the packet traces into a connection-based format such as the one used in the KDD dataset. For this purpose, the `bro` IDS[4] has been used, together with a properly crafted trace parsing tool. `bro` is a powerful network monitoring framework, originally developed by Vern Paxson at the International Computer Science Institute and the National Center for Supercomputing Applications, that, while focusing on network security issues, provides a comprehensive platform for more general network traffic analysis. After processing the traffic traces with `bro`, a further step, encoding the nominal traffic features with arbitrary numeric values (tokens), is performed. For example, the `protocol_type` (e.g., `tcp`) is encoded by using the IANA-assigned protocol numbers, and so on.

A forest structure has been used in parsing the connection data from `bro` to make the preprocessing phase more efficient. During the tokenization, two search trees are built, structured according to the scheme reported in Fig. 3. There are three types of node: *source-Ip*, *destination-Ip*, *destination-Port*. Each path within a tree identifies a connection. A set of counters $C = \{c_1, \dots c_n\}$ (with $n = |C|$) is associated to each node: for instance, if there are 5 connections from the source IP `192.168.0.1`, the `connections` counter in the `192.168.0.1`-node will have value 5. It is trivial to note that the sum of the counters c_i associated to the children of a particular node is equal to the corresponding counter c_i on the node itself. When a new connection is found in the traffic trace, the respective path is added, but if the first part of the path is already in the tree, only the second one is added to it, and the counters of the first one are simply incremented. In this way, the value of the traffic features for each connection can be computed in a straightforward way.

For instance, the feature `count` is the number of connections to the same host (during the current connection in the past 2 s), so to compute this feature, the only thing to do is to read the value of the specific counter in the destination node of the path from source to destination. Another forest with a specular structure (i.e., with paths going from nodes describing the destination to nodes describing the source) is needed, because most features, to be computed, need the destination-IP as the search base. For instance if we want to know the number of connections from a specific host, say X, to a destination Y, using the destination-IP tree, the only thing to do is to read the associated counter value in the node X, chosen between the sons of Y. With this approach, it is possible to compute all the needed features: some of them are equal to the values of specific counters in the tree nodes, but other ones can be easily derived from them, like the percentage of connections that were reset or the connections with SYN errors (e.g., these connections will have status `S0` in `bro`).

5.3. Methodology

The success of a classifier meant to be used in practice depends ultimately on its ability to perform well not only with the data used for its testing, but most of all with real-world data. Although a methodologically sound procedure ensures that the classifier is tested against data it has never seen before (i.e., in the training phase), there is no guarantee whatsoever that testing data will be representative of, and resemble closely, real-world data. Such data surely are not available at the time of testing. On the other hand, when using a portion of the training data as a validation dataset to verify and refine the selection of model hyper-parameters, training data could "leak" some information about the validation conditions if the partitioning of data into training and validation sets is not done properly so as to ensure that instances relative to the same (or very close) conditions go in the same part. Connections recorded with the same conditions might, in fact, share some

Table 1
KDD 1999 data set.

Class	*Training set*	*10% Training set*
Normal	972,781	97,278
Probe	41,102	4107
DoS	3,883,370	391,458
R2L	1126	1126
U2R	52	52
Total	4,898,431	49,4021

[4] http://www.bro-ids.org/

Table 2
Features used in the experiments.

No.	Feature name	Description	Data type
1	duration	Length (number of seconds) of the connection	continuous
2	protocol_type	Type of the protocol, e.g., tcp or udp	discrete
3	service	Network service on the destination, e.g., http or telnet	discrete
4	flag	Normal or error status of the connection	discrete
5	src_bytes	Number of data bytes from source to destination	continuous
6	dst_bytes	Number of data bytes from destination to source	continuous
7	land	1 if connection is from/to the same host/port; 0 otherwise	discrete
8	wrong_fragment	Number of "wrong" fragments	continuous
9	urgent	Number of urgent packets	continuous
10	count	Number of connections to the same host as the current connection in the past two seconds	continuous
11	srv_count	Number of connections to the same service as the current connection in the past two seconds	continuous
12	serror_rate	% of same-host connections that have "SYN" errors	continuous
13	srv_serror_rate	% of same-service connections that have "SYN" errors	continuous
14	rerror_rate	% of same-host connections that have "REJ" errors	continuous
15	srv_rerror_rate	% of same-service connections that have "REJ" errors	continuous
16	same_srv_rate	% of same-host connections to the same service	continuous
17	diff_srv_rate	% of same-host connections to different services	continuous
18	srv_diff_host_rate	% of same-service connections to different hosts	continuous
19	dst_host_count	Count of connections having the same destination host	continuous
20	dst_host_srv_count	Count of connections having the same destination host and using the same service	continuous
21	dst_host_same_srv_rate	% of connections having the same destination host and using the same service	continuous
22	dst_host_diff_srv_rate	% of different services on the current host	continuous
23	dst_host_same_src_port_rate	% of connections to the current host having the same src port	continuous
24	dst_host_srv_diff_host_rate	% of connections to the same service coming from different hosts	continuous
25	dst_host_serror_rate	% of connections to the current host that have an S0 error	continuous
26	dst_host_srv_serror_rate	% of connections to the current host and specified service that have an S0 error	continuous
27	dst_host_rerror_rate	% of connections to the current host that have an RST error	continuous
28	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error	continuous

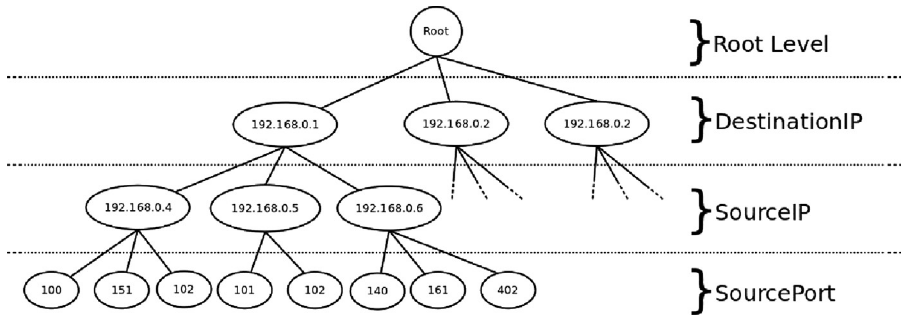

Fig. 3. Tree-like structure for the efficient computation of features.

similarity that might unnoticeably cause over-fitting when two instances taken in the same day are used both in the training and validation sets. The classification accuracy would thus be over-estimated. Connection that are comparable in terms of amount of data being exchanged might be too similar to one another for the validation to avoid biasing. Thus, a check that has been added is to subdivide connections into categories based on data volume, and to ensure that all the connections belonging to the same category are used either as training or as validation inputs.

5.4. Evaluation criteria

The performances of a network must be evaluated by testing it on a different data set than the one on which it was trained. The relatively large number of parameters involved (all the weights) means that it is too easy for the network to acquire unique characteristic of the training set (over-fitting), rather than generalizing on properties of the population. While the mean square error is easy to compute, its most obvious disadvantage is that it is a purely mathematical construct, which may have little meaning with respect to the practical task performed by the network. If the task is to classify a pattern into one of the several categories, the mean square error tells us nothing about expected frequency of misclassification. A more important problem with the MSE as a guide to performance is that it fails to distinguish minor and serious errors. If the task of the network is to classify cases into one of the several categories, examining a confusion matrix can be highly informative. A confusion matrix is a matrix containing as many rows and columns as there are classes. For every case, its class is chosen to be that corresponding to the output neuron that has the maximum activation. The content of the entry at the i-th row and the j-th column of the confusion matrix is the number of test cases that truly belong to class i but which were classified into class j. Ideally, one would obtain a strictly diagonal matrix. Quantities in off-diagonal positions represent misclassifications. The principal strength of the confusion matrix is that it clearly identifies the nature of errors, as well as their quantity. The experimenter is then free to evaluate the performances of the network in terms of relative severity of misclassifications. The evaluation parameters that have been selected are thus:

- *Accuracy*: Is the reliability of the rule, usually represented by the proportion of correct classifications, although it may be that some errors are more serious than others, and it may be important to control the error rate for some key class.
- *Speed*: In some circumstances, the speed of the classifier is a major issue. A classifier that is 90% accurate may be preferred over one that is 95% accurate if the former is 100 times faster in testing (and such differences in time-scales are not uncommon in neural networks). Such considerations would be even more important when the number of samples to be processed is very large.
- *Comprehensibility*: If it is a human operator who must apply the classification procedure, the procedure must be easily understood, else mistakes will be made in applying the rule. It is important, also, that human operators believe and trust the system. An oft-quoted example is the Three-Mile Island case, where the automatic devices correctly recommended a shutdown, but this recommendation was not acted upon by the human operators who did not believe that the recommendation was well founded.
- *Time to learn*: Especially in a rapidly changing environment, it may be necessary to learn a classification rule quickly, or make adjustments to an existing rule in real time. “Quickly” might imply also that we need only a small number of observations to establish our rule. In addition, retraining can be performed quickly, and this in turn allows it to be performed more often, and our results in Section 6 show that this may very frequently be the case.

5.5. *Early stopping*

Early stopping is a technique to concentrate training efforts. Typically it involves partitioning the data set into three parts: training, validation and testing. Gradient descent on the objective function is performed with data in the training set. In parallel, the resulting model is tested against the validation set (not the testing set) to see how well it is performing in unseen data. Samples in the validation set can be used during training, because they would not be included in the testing set. The training-validation partitioning can also be done in a similar way to that done in n-fold cross validation. The reconstruction error should normally decay steeply at the beginning of training, and then “stabilize” as the effects of noise in the gradient calculations begin to appear. Thus, when accuracy ceases to improve, or starts dropping, optimization is stopped.

5.6. *Choice of hyper-parameters*

In the training phase of Experiment 1, normal traffic of the first real trace was used for training, and traffic from the second trace for testing. The same testing set has been used in Experiment 2. However, in Experiment 2, KDD data purified from attacks were used as training data. Thus, the neural network was allowed to recognize a set of instances of normal traffic. The learning rate has been set to 0.1, while the number of training epochs was initially chosen as 15, trying also 10 and 5. A range of different values were used for the other hyper-parameters. The values obtained with the best 10 combinations have been used for the charts.

6. Results

In Fig. 4, the free energy of the RBM is shown for both normal and anomalous connections in the real data set. As it can be observed, the free energy of normal connections tends to oscillate between two intervals. Anomalous behavior, instead, shows two distinctive characteristics. Its free energy reaches extreme values (the positive and negative spikes) or it stays very stably in a narrow interval. The second characteristic can be traced back to routine communication between the malware and its controlling nodes. While the free energy for this activity lies in the range of the free energies associated to normal traffic, values in the specific interval for anomalous traffic are rarely found in normal traffic. The spikes are instead associated with actual commands sent out to the malware agent and to the return of a quantity of data gathered on the infected PC and sent back to a different (possibly up the chain of command) node.

The training times were significant. Training in Experiment 1 over 15 epochs took 186.4 h on a HP DL380 G3.

In Fig. 5, the accuracies obtained for the various experiments are reported. The graphs use boxplot format of Tukey and Chambers and illustrate the distribution of data. The central box is bounded by the first and third quartiles, and thus it encloses the middle 50% of the data. The “whiskers” extend to the minimum and maximum values. A “+” sign indicates the mean, whereas the

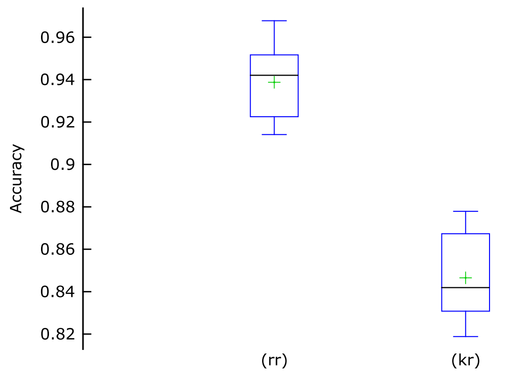

Fig. 5. Performance when training on real, testing on real (rr); training on KDD, testing on real (kr).

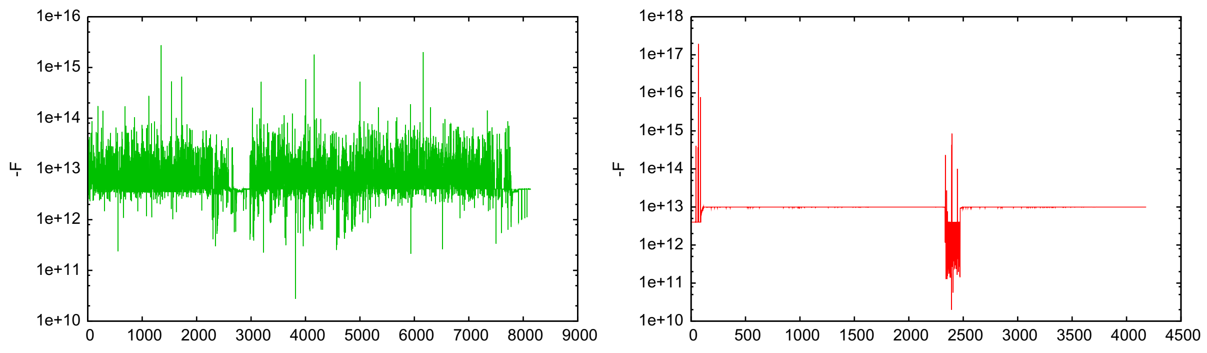

Fig. 4. Free energies for normal (left) and anomalous (right) data, real dataset.

horizontal line across the box is drawn at the median. Each box is relative to an experiment. The first (rr) gives the results for the experiment where the DBRM was trained and tested on the real data and the rightmost (kr) the experiment where the DBRM was trained on the KDD dataset and tested on the real data. Recall that traffic labeled as anomalous contained both "noisy" events that are noticeable for their characteristics, and "silent" events whose low volume and frequency makes them harder to detect. This may explain, in part, the performance in the first experiment. By comparing the accuracy obtained in Experiments 1 and 2, we can see the inadequacy of the training on the KDD dataset to be exported to another dataset. The lower values in the (kr) box suggest that the use of highly dishomogeneous data for training and testing decreases the classification performance. This may be due to over-fitting, but it should be observed that the parameters of operational networks reflect policies, usage habits, and conditions that have an organizational, regional, and cultural nature. Thus it is natural to expect that diverse networks tend to exhibit their own traffic parameters. In addition, the discredit that the KDD dataset has received from the network security community, mostly based on the fact that recorded traffic is not representative of a real network traffic, may explain the result. To improve classification accuracy, a baseline of normal traffic should definitely be isolated specifically for the network where the analysis is to be performed.

7. Conclusions

In this paper, the Discriminative Restricted Boltzmann Machine, a recently proposed classifier based on the family of energy-based models, has been applied to network anomaly detection in a semi-supervised fashion. In particular we were interested in decoupling the training data from the testing scenario to assess the generalization abilities of neural networks. The DRBM has been chosen for its ability to combine generative power, to capture the inherent aspects of the normal traffic class, and classification accuracy.

Experiments confirm that, when the classifier is tested in a network widely different from the one where training data were taken from, the performance suffers. This suggests the need for further investigation over the nature of anomalous traffic and the intrinsic differences with normal traffic.

We can consider that the nonparametric nature of the above techniques can simultaneously become their main strength and Achille's heel: a nonparametric method is not based on any known form or distribution of the sample observations and consequently it is more robust. At the same time being nonparametric means that no prior knowledge about anomalous phenomena is available to be incorporated into the detection model, greatly reducing its effectiveness. This implies that, since the detection capability is directly associated with the correctness of the underlying self-learnt traffic model, if the training set does not accurately represent the real network normal traffic (i.e., it is not realistic), we may overestimate or underestimate anomalous phenomena. Thus, artificial traffic models are error prone and should not be used for effective semi-supervised training purposes. However, also real traffic patterns used for characterizing the normal behavior can be affected by unpredictable network noise phenomena largely due to the randomness and burstiness of the traffic behavior that can adversely affect the anomaly detection accuracy. Hence, in future applications, to enhance the performance of the detection phase we will try to improve the overall "signal-to-noise ratio" through signal pre-processing by limiting the influence the "noisy" components and unwanted dependencies according to some de-noising technique such as nonparametric regression or other available ones. Directions for future research also include the analysis of feature-modeling detectors, combined with the study of the discriminatory power of single features.

Acknowledgments

The authors would like to thank A. Serra, L. Puglia, M. Fratello, D. Cavaliere who contributed some tools which were used in this work.

References

[1] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. 41 (3) (2009) 15:1–15:58.
[2] J P. Anderson, Computer Security Threat Monitoring and Surveillance, Technical Report, Computer Security Division of the Information Technology Laboratory, National Institute of Standards and Technology, 1980.
[3] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, E. Vazquez, Anomaly-based network intrusion detection: techniques, systems and challenges, Comput. Secur. 28 (1–2) (2009) 18–28.
[4] M. Ficco, L. Romano, A generic intrusion detection and diagnoser system based on complex event processing, in: Proceedings – 1st International Conference on Data Compression, Communication, and Processing, CCP 2011, IEEE, 2011, pp. 275–284.
[5] F. Palmieri, U. Fiore, A nonlinear, recurrence-based approach to traffic classification, Comput. Networks 53 (6) (2009) 761–773.
[6] F. Palmieri, U. Fiore, Network anomaly detection through nonlinear analysis, Comput. Secur. 29 (7) (2010) 737–755.
[7] K. Noto, C. Brodley, D. Slonim, FRaC: a feature-modeling approach for semi-supervised and unsupervised anomaly detection, Data Mining knowl. Discovery 25 (1) (2012) 109–133.
[8] C.-H. Mao, H.-M. Lee, D. Parikh, T. Chen, S.-Y. Huang, Semi-supervised co-training and active learning based approach for multi-view intrusion detection, in: Proceedings of the 2009 ACM symposium on Applied Computing, SAC '09, ACM, 2009, pp. 2042–2048.
[9] C. Chen, Y. Gong, Y. Tian, Semi-supervised learning methods for network intrusion detection, in: IEEE International Conference on Systems, Man and Cybernetics, 2008, SMC '08, IEEE, 2008, pp. 2603–2608.
[10] J.Z. Lei, A.A. Ghorbani, Improved competitive learning neural networks for network intrusion and fraud detection, Neurocomputing 75 (1) (2012) 135–145.
[11] T. Ahmed, B. Oreshkin, M. Coates, Machine learning approaches to network anomaly detection, in: Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques, SYSML'07, USENIX Association, Berkeley, CA, USA, 2007, pp. 7:1–7:6.
[12] M. Davenport, R. Baraniuk, C. Scott, Learning minimum volume sets with support vector machines, in: Proceedings of the 2006 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing, 2006, IEEE, 2006, pp. 301–306.
[13] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[14] G.E. Hinton, To recognize shapes, first learn to generate images, Prog. Brain Res. 165 (2007) 535–547.
[15] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[16] M.A. Salama, H.F. Eid, R.A. Ramadan, A. Darwish, A.E. Hassanien, Hybrid intelligent intrusion detection scheme, Soft Comput. Ind. Appl. (2011) 293–303.
[17] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[18] L. Ogiela, M.R. Ogiela, Advances in Cognitive Information Systems Cognitive Systems Monographs, vol. 17, Springer, 2012.
[19] G.E. Hinton, A Practical Guide to Training Restricted Boltzmann Machines, Momentum, vol. 9, 2010, pp. 1–21.
[20] D.E. Rumelhart, G.E. Hintont, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[21] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, J. Mach. Learn. Res. – Proc. Track 9 (2010) 249–256.
[22] H. Larochelle, Y. Bengio, Classification using Discriminative Restricted Boltzmann Machines, in: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008), ACM, 2008, pp. 536–543.
[23] C. Janeway, P. Travers, M. Walport, J. Capra, Immunobiology: The Immune System in Health and Disease, Garland Science, 2001.
[24] F. Palmieri, U. Fiore, Automated detection and containment of worms and viruses into heterogeneous networks: a simple network immune system, Int. J. Wireless Mobile Comput. 2 (1) (2007) 47–58.
[25] C.-M. Ou, Host-based intrusion detection systems adapted from agent-based artificial immune systems, Neurocomputing 88 (0) (2012) 78–86.
[26] J.-S.R. Lee, H.-D.J. Jeong, D.C. McNickle, K. Pawlikowski, Self-Similar Properties of Spam, in: Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS, IEEE, 2011, pp. 347–352.

[27] A. Castiglione, A. De Santis, U. Fiore, F. Palmieri, Device tracking in private networks via NAPT log analysis, in: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, IEEE, 2012, pp. 603–608.

[28] S.T. Brugger, J. Chow, An Assessment of the DARPA IDS Evaluation Dataset Using Snort, Technical Report, University of California, Davis, Department of Computer Science, 2007.

[29] A. Castiglione, R. De Prisco, A. De Santis, U. Fiore, F. Palmieri, A botnet-based command and control approach relying on swarm intelligence, J. Network Comput. Appl., http://dx.doi.org/10.1016/j.jnca.2013.05.002, in press.

Ugo Fiore leads the Network Operations Center at the Federico II University, in Naples. He began his career with Italian National Council for Research and has also more than 10 years of experience in the industry, developing software support systems for telco operators. His research interests focus on optimization techniques and algorithms aiming at improving the performance of high-speed core networks. He is also actively pursuing two other research directions: the application of nonlinear techniques to the analysis and classification of traffic; security-related algorithms and protocols.

Francesco Palmieri is an assistant professor at the Engineering Faculty of the Second University of Napoli, Italy. His major research interests concern high performance and evolutionary networking protocols and architectures, routing algorithms and network security. Since 1989, he has worked for several international companies on networking-related projects and, starting from 1997, and until 2010 he has been the Director of the telecommunication and networking division of the Federico II University, in Napoli, Italy. He has been closely involved with the development of the Internet in Italy as a senior member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has published a significant number of papers in leading technical journals and conferences and given many invited talks and keynote speeches.

Aniello Castiglione joined the Dipartimento di Informatica ed Applicazioni “R. M. Capocelli” of University of Salerno in February 2006. He received a degree in Computer Science and his Ph.D. in Computer Science from the same university. He is a reviewer for several international journals (Elsevier, Hindawi, IEEE, Springer, Inderscience) and he has been a member of international conference committees. He is a Member of various associations, including IEEE (Institute of Electrical and Electronics Engineers), of ACM (Association for Computing Machinery), of IEEE Computer Society, of IEEE Communications Society, of GRIN (Gruppo di Informatica) and of IISFA (International Information System Forensics Association, Italian Chapter). He is a Fellow of FSF (Free Software Foundation) as well as FSFE (Free Software Foundation Europe). For many years, he has been involved in forensic investigations, collaborating with several Law Enforcement agencies as a consultant. His research interests include Data Security, Communication Networks, Digital Forensics, Computer Forensics, Security and Privacy, Security Standards and Cryptography.

Alfredo De Santis received a degree in Computer Science (cum laude) from the University of Salerno in 1983. Since 1984, he has been with the Dipartimento di Informatica ed Applicazioni of the Università di Salerno. Since 1990 he is a Professor of Computer Science. From November 1991 to October 1995 and from November 1998 to October 2001 he was the Chairman of the Dipartimento di Informatica ed Applicazioni, University of Salerno. From November 1996 to October 2003 he was the Chairman of the Ph.D. Program in Computer Science at the University of Salerno. From September 1987 to February 1990 he was a Visiting Scientist at IBM T. J. Watson Research Center, Yorktown Heights, New York. He spent August 1994 at the International Computer Science Institute (ICSI), Berkeley CA, USA, as a Visiting Scientist. From November 2009 he is in the Board of Directors of Consortium GARR (the Italian Academic & Research Network). His research interests include Algorithms, Data Security, Cryptography, Information Forensics, Communication Networks, Information Theory, and Data Compression.