A Systematic Approach for the Application of Restricted Boltzmann Machines in Network Intrusion Detection

Arnaldo Gouveia Miguel Correia

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Abstract. A few exploratory works studied Restricted Boltzmann Machines (RBMs) as an approach for network intrusion detection, but did it in a rather empirical way. It is possible to go one step further taking advantage from already mature theoretical work in the area. In this paper, we use RBMs for network intrusion detection showing that it is capable of learning complex datasets. We also illustrate an integrated and systematic way of learning. We analyze learning procedures and applications of RBMs and show experimental results for training RBMs on a standard network intrusion detection dataset.

1 Introduction

Deep neural networks have become increasingly popular due to their success in machine learning. Their history goes as far back as 1958 when Rosenblatt published his work on the perceptron concept [19]. Present day forms of deep learning networks include Hopfield Networks, Self-Organizing Maps, Boltzmann Machines, Multi-Layer Perceptrons, Autoencoders or Deep Belief networks.

Most of present day machine learning algorithms are not classifiable as *deep* since they use at most one layer of hidden variables. Bengio and LeCun [2, 3], have shown that the internal representations learned by such systems are necessarily simple due to their simple internal structure, being incapable of extracting certain types of complex structure. However this limitation does not apply to specific types of energy-based learning approaches.

Problem with simple DL networks with one layer of hidden variables.

Two important classes of Boltzmann Machine (BMs) are the Restricted Boltzmann Machine (RBM) described by a complete bipartite graph, and the Deep RBM that is composed of several layers of RBMs. The BM derived from Hopfield networks and in its initial form was fully node-connected. The concept of RBMs came about due to the difficulty of training a fully connected BM in classification problems. RBMs are designated *restricted* due to the fact that there are no connections among the hidden layer nodes or among the visible layer nodes.

Recently the notion of deep learning gained a lot of attention as a method to model high-level abstractions by composing multiple non-linear layers [15]. Several deep learning network architectures, like deep belief networks [8], deep

BMs [20], convolutional neural networks [15], and deep denoising auto-encoders [26], have shown their advantages in specific areas.

Despite their interest, these approaches have barely been applied in Network Intrusion Detection Systems (NIDSs), i.e., for detecting cyber-attacks by inspecting computer network traffic. This paper aims to contribute for closing this gap by introducing a systematic approach for training RBMs for network intrusion detection. The approach considers three important aspects: weight initialization, pre-training, and fine-tuning.

The paper seeks to demonstrate the effectiveness of the approach with an analysis based on a dataset carefully crafted for this purpose: the UNB ISCX intrusion detection evaluation dataset [21]. This dataset is reasonably recent but has been gaining increasing adoption for evaluating NIDSs.

2 The Ising Model

The RBM formalism, specifically in terms of synonymous of objective or loss function, is similar to the Ising Model formalism, so it is relevant to address the similarities. Energy-based models are popular in machine learning due to the elegance of their formulation and their relationship to statistical physics. Among these, the Restricted Boltzmann Machine (RBM) is the focus of our work.

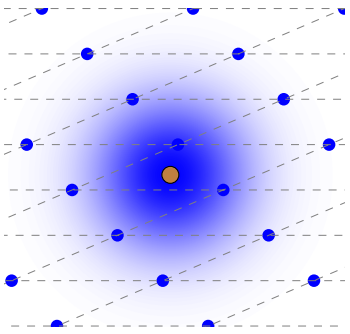


Fig. 1. Illustrating a 2-dimensional interaction with only the nearest nodes

The Ising model was defined by Lenz in 1920 and named after his student Ising, who developed the model in his PhD thesis [14]. Ising solved the model optimization problem in one dimension. The two-dimensional square lattice Ising model was given an analytic description much later [18]. Such physical models, having two alternate states in an array with mutual interactions, are currently described in physics as spinor Ising models. The mutual interaction among spin units is modeled by an interaction parameter commonly named coupling. Interactions between particles is small and restricted to their neighbourhood. In this model the states align among themselves spontaneously in a way that minimizes a global parameter, e.g., the global energy. In 2D the model topology is usually described as a regular lattice as illustrated in Figure 1.

3 Restricted Boltzmann Machines

3.1 Energy-Based Models (EBM)

Energy-based models associate an energy figure to each configuration of the state variables. Energy in this context is synonymous of objective or loss function.

Learning, again in this context, corresponds to modifying the energy function as to find minima. In the probabilistic model associated with RBMs the associated probability distribution is described through an energy type function:

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{1}$$

The normalizing factor Z is the *partition function* in the context of physical systems. The normalization is achieved by summing across all available sates and divide.

$$Z = \sum_{x} e^{-E(x)} \tag{2}$$

An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data. As for logistic regression, we will first define the log-likelihood and then the loss function as being he negative log-likelihood.

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log p(x^{(i)}) \tag{3}$$

The stochastic gradient is $-\frac{\partial \log p(x^{(i)})}{\partial \theta}$ where θ are the model parameters. However, computationally this is not the best option for this type of energy function (see Section 3.4).

3.2 The Concept of Restricted Boltzmann Machine

From the physical analogy, BMs model the data with an *Ising model* that is in thermal equilibrium. In this analogy the equivalent to physical spins are called RBM units or nodes. The set of nodes that encode the observed data and the output are called the visible units $\{v_i\}$, whereas the nodes used to model the latent concept and feature space are called the hidden units $\{h_i\}$. For the purpose of explanation, we assume that the visible and hidden units are binary. Alternatives are discussed later.

The RBM concept is similar to the BM concept [1], except that no connections between neurons of the same layer are allowed. Figure 2 depicts the architecture of a RBM, consisting of two layers: the visible layer $\{v_i\}$ and the hidden layer $\{h_i\}$, with N_v and M_h nodes, respectively. Hidden units are used to capture higher level correlations in the data, and the visible units to mirror the data itself. Connections between nodes are restricted so that there are no visible-visible and hidden-hidden connections. Hidden-visible connections are strictly symmetrical. Hence, we have a restricted BM. An RBM is a bi-partite (visible and hidden) BM with full interactivity between visible and hidden units, and no interactivity between units of the same type. RBMs are usually described as energy-based stochastic neural networks composed by two layers of neurons

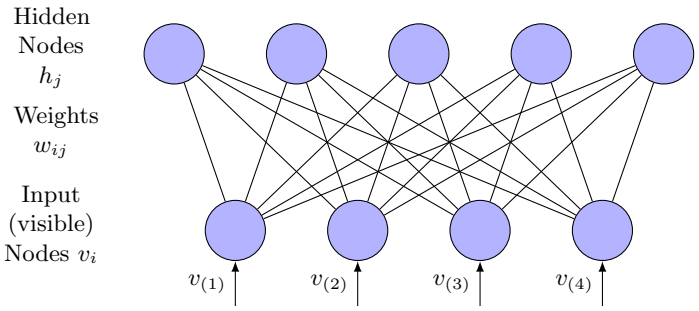


Fig. 2. A RBM with 4 inputs and 5 hidden nodes.

(visible and hidden), in which the learning phase is conducted in an unsupervised fashion. RBMs are a variant of the BM model [1, 7]. Here we consider the case where h_j can only take binary values, and σ_i as the standard deviation for each input dimension.

Convergence in BMs can be slow, particularly when the number of units and layers increases. RBMs were introduced to mitigate this issue [23]. The simplification consists of having only one layer of visible and one layer of hidden units with links between units on the same layer erased, allowing for parallel updates of hidden and visible units (Figure 2).

In Figure 2, $\mathbf{v} = (v_1, v_2 \cdots v_n)$ and $\mathtt{h} = (h_1, h_2 \cdots h_m)$ are the visible and the hidden vectors, a_i and b_j are their biases, n and m are the dimension of the visible layer and the hidden layer, and w_{ij} is the connection weight matrix between the visible layer and the hidden layer. The visible stochastic binary variables $\mathbf{v} \in \{0,1\}^N$ are connected to hidden stochastic binary variables $\mathbf{h} \in \{0,1\}^M$.

For binary RBMs the energy $E(\mathbf{v}, \mathbf{h})$, which defines the bipartite structure, is given by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \,\epsilon\, visible} a_i v_k - \sum_{j \,\epsilon\, hidden} b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j \tag{4}$$

or equivalently:

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

The weight matrix $\mathbf{W}$, the visible bias vector $\mathbf{b}$ and the hidden bias vector $\mathbf{c}$ are the parameters of the model.

RBM satisfies a Boltzmann-Gibbs distribution over all its units. The joint probability of $(\mathbf{v}, \mathbf{h})$ is given by a probability distribution function $P(\mathbf{v}, \mathbf{h})$, where Z is the normalization term to obtain a proper probability distribution function:

$$P(\mathtt{v}, \mathtt{h}) = \frac{1}{Z} \mathrm{e}^{-E(\mathbf{v}, \mathbf{h})} \tag{6}$$

By definition the partition function, which sums over all possible visible and hidden states, is given by:

$$Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \tag{7}$$

The normalizing factor Z is called the partition function by analogy with physical systems. To find $p(v)$, we marginalize over the hidden units. Given a set of training vectors, V, to train a RBM, one aims to maximize the average probability, $p(\mathbf{v}), \mathbf{v} \in V$, where

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{h} e^{-E(\mathbf{v},\mathbf{h})} \tag{8}$$

which can also be written as

$$p(\mathbf{v}) = \frac{1}{Z} e^{-F(\mathbf{v})} \tag{9}$$

where $F(v)$ is the logarithm of the energy function summed over h:

$$F(\mathbf{v}) = -log \sum_{h} e^{-E(\mathbf{v},\mathbf{h})} \tag{10}$$

For training efficiency, BMs can be restricted to a bipartite graph with one set of visible neurons and one set of hidden neurons. As shown in Figure 2 there are only visible-hidden and hidden-visible connections (still symmetric). Therefore hidden units h_j only depend on the visible units v_j and vice-versa, with b_j as the biases for the visible units and c_j for the hidden units:

$$p(h_j = 1|\mathbf{v}) = \sigma(c_j + \sum_{i} w_{ij} v_i) \tag{11}$$

$$p(v_j = 1|\mathbf{h}) = \sigma(b_j + \sum_{i} w_{ij} h_i) \tag{12}$$

3.3 Gibbs sampling

Gibbs sampling is commonly used for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution, when direct sampling is difficult. This sequence can be used to approximate a joint distribution function, e.g., the joint distribution function expressed in Eq. 6 in our case.

In its simplest theoretical description this is how Gibbs sampling would work, with all updates done in parallel, as illustrated in Figure 3. The most common algorithm for Gibbs sampling is *Contrastive Divergence* (CD), used inside a gradient-descent. A single-step contrastive divergence (CD-1) procedure for a single training example can be summarized as follows [9]:

1. Sample hidden units **h** from training example **v**

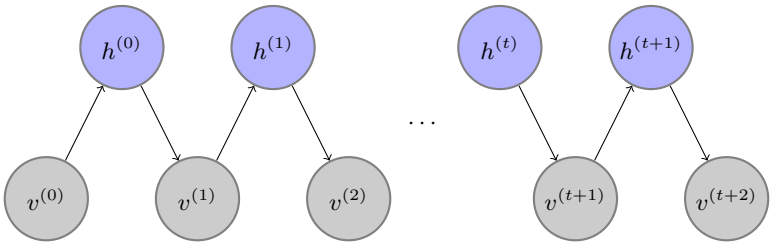


Fig. 3. Gibbs sampling in a RBM

2. Sample reconstruction $\mathbf{v}'$ of visible units using $\mathbf{h}$ and then resample $\mathbf{h}'$ from it. (Gibbs sampling step)
3. $w_{ij} \leftarrow w_{ij} - \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{sampled})$

In the previous equation $< . >$ denotes an internal product. In practice, waiting till $t \rightarrow \infty$ is not practical. However, an alternative consists in modulating the number of iterations for a limited number and extract a final sample:

- After t steps, $(h^{(t)}, v^{(t)})$ is available for sampling
- Sample $(h^{(t)}, v^{(t)})$ assuming a sample accurate enough as an approximation of $P(v, h)$ as $t \rightarrow \infty$

As $t \rightarrow \infty, (v^{(t)}, h^{(t)})$, v^t and h^t will be ever more accurate samples drawn from the RBM's distribution. Nevertheless, one can further speed up the process by using the contrastive divergence (CD) algorithm as explained next.

3.4 Contrastive Divergence Algorithm in Detail

It is easy to calculate $< v_i h_j >_{data}$ because there is no direct connections among the hidden units. However, it is difficult to get an unbiased sample of $< v_i h_j >_{model}$. Hinton proposed a faster learning algorithm with contrastive divergence (CD) learning and the change of learning parameter [10]. The partial derivative of the log probability of Eq. 8 with respect to a weight is given by:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = (< v_i h_j >_{data} - < v_i h_j >_{model}) \tag{13}$$

where the angle brackets $< v_i h_j >_{data}$ and $< v_i h_j >_{model}$ are used to denote expectations of the distribution specified by the subscript *data* and *model*. In the log probability, a very simple learning rule for performing stochastic steepest ascent is given by:

$$\Delta w_{ij} = \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{model}) \tag{14}$$

where ε is a learning rate.

The CD algorithm computes an approximation of the gradient by performing Gibbs sampling for a finite number of steps. This involves initializing the RBM

with a training example $v \epsilon V$ and running the RBM for k (often with $k = 1$) steps. CD-k is a generalization k iterations by repeating the sampling process k times. CD makes the following simplifications for computing the gradient:

1. Replace the first term (expectation over all input samples) with a single sample.
2. For the second term, run the chain for fixed k steps:

$$\Delta w_{ij} = \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{sampled}), \tag{15}$$

The bias updates for the visible and hidden layers respectively can be defined by these expressions:

$$\Delta a_i = \varepsilon(< v_i >_{data} - < v_i >_{sampled}), \tag{16}$$

$$\Delta b_j = \varepsilon(< h_j >_{data} - < h_j >_{sampled}) \tag{17}$$

where $< v_i h_j >_{sampled}$, or *reconstructed* by means of the CD algorithm can be computed more efficiently than $< v_i h_j >_{model}$.

3.5 Applying RBMs to Continuous-Valued Inputs

With the binary units introduced for RBMs in [8], one can handle continuous-valued inputs by scaling them to the $[0, 1]$ interval and considering each input as the probability for a binary random variable to take the value 1. Previous work on continuous-valued input in RBMs include [4], in which noise is added to sigmoidal units, and the RBM forms a special form of Diffusion Network [16]. The approach followed in this paper starts by acquiring the RBM weights in the pre-training phase by using Rectified Linear Units (for the hidden units) and training with Sigmoid units, which has shown to work very well [6], something that our results confirm.

A continuous RBM (RBM) is a form of RBM that accepts continuous inputs via a different type of contrastive divergence sampling. This allows the CRBM to handle things like image pixels or word-count vectors that are normalized to decimals between zero and one.

4 A Systematic Approach for Training RBMs

This section describes an optimized approach for choosing the training parameters of the specific RBMs configured. It takes advantage of Hinton et al.'s results [11]. An optimization approach is important in the context of RBMs due to the complexity of assuring the learning process convergence and for avoiding overfitting. In the end we aim at showing the validity of our approach in the context of network intrusion detection by using a specific dataset. The approach comprehends three major aspects – Weight initialization, Pre-training, and Fine-Tuning – plus a few other parameter optimization choices (see Table 1).

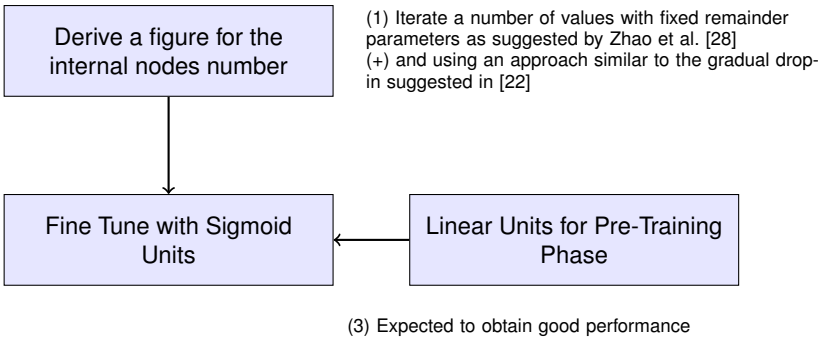


Fig. 4. Approach for training RBMs

4.1 The Three Major Aspects

Pre-Training As an alternative or in addition to weight initialization techniques, layer-wise unsupervised pre-training can be used to initialize the weights for fine-tuning. For every combination of two adjacent layers, an RBM is trained for a certain number of epochs, whereas an epoch consists of training the network on batches of samples.

Fine-Tuning After pre-training, the instance is fine-tuned using one of several fine-tuning functions (more on those in the next section).

Choosing the Number of Hidden Nodes Dimensioning the RBM internals involves testing a set of dimensioning options and extracting the best option using the accuracy maximization criteria.

4.2 Using Rectified Linear Units

Hinton et al. have shown that for problems involving real valued data rectified linear units provide better results [17]. Although their research has been over image datasets, there is enough numerical similitude to both problems as the results obtained have been based on a feature based real valued dataset. The present proposal uses Rectified Linear Units for the hidden layer during pre-training. A suggestion over the usage of sigmoid units has been put forward by pre-training the RBM with Rectified Linear Units and fine tuning with Sigmoid Units. The advantages of this approach come in the type of output produced belonging to $\{0, 1\}$ instead of $[0, 1]$.

4.3 Parameter Optimization

As stated previously we do not try to optimize the whole set of RBM parameters as it would show to be exhaustive, but instead choose a number of parameters as optimization enablers, based on the literature. The specific values for each parameter used in the experiment is in Table 1.

Table 1. Experimental Parameters

Fine Tuning parameters	Value	Fine Tuning parameters	Value
rbm.batchSize	100	DArch.initialMomentum	0.5
rbm.lastLayer	True	DArch.finalMomentum	0.9
rbm.learnRate	0.001	DArch.momentumRampLength	0.5
rbm.weightDecay	2e-04	DArch.unitFunction	sigmoidUnitRbm
rbm.initialMomentum	0.5	bp.learnRate	0.001
rbm.finalMomentum	0.9	DArch.dropout	0
rbm.unitFunction	linearUnitRbm	DArch.dropout.oneMaskPerEpoch	False
rbm.updateFunction	rbmUpdate	DArch.isClass	True
rbm.numCD	10	DArch.numEpochs	50
rbm.numEpochs	50	DArch.errorFunction	Mean Square Error
rbm.momentumRampLength	0.5	retainData	True
DArch.batchSize	100	normalizeWeights	True
bootstrap	True	preProc.params	method="range"
DArch.fineTuneFunction	backpropagation	generateWeightsFunction	generateWeightsGlorotUniform

Number of Training Epochs. Regularization of neural networks is used to improve generalization by preventing over-fitting. The most straightforward way of preventing over-fitting is early stopping. One of the options for the regularization approaches used limits the number of iterations is choosing the number of epochs for the training phase to 50. In practice this number has been chosen as a limit value based on empirical criteria since it marked the beginning of an asymptotic behaviour for the error.

Mini batches Updating the weights using small "mini-batches" of 10 to 100 samples in size has shown to allow better results. In the experiment we used a mini batch size of 100.

Momentum By choosing a momentum value it is possible to modulate the speed of learning in RBM training. At the start of learning, the random initial parameter values may create very large gradients and the system is unlikely to be at a minimum, so it is usually best to start with a low momentum of 0.5. This very conservative momentum typically makes the learning more stable than no momentum at all [11].

Learning Rate A good rule of thumb for setting the learning rate ε is to look at a histogram of the weight updates and a histogram of the weights [27]. The updates should be about 10^{-3} times the weights (to within about an order of magnitude). When a unit has a very large fan-in, the updates should be smaller since many small changes in the same direction can easily reverse the sign of the gradient. Conversely, for biases, the updates can be bigger: $\Delta w_{ij} = \varepsilon(< v_i h_j >_{data} - < v_i h_j >_{sampled})$

Weight Decay Weight-decay works by adding an extra term to the normal gradient. The extra term is the derivative of a function that penalizes large weights. The simplest penalty function, called L2, is half of the sum of the squared weights times a coefficient which will be called the weight-cost. For an RBM, sensible values for the weight-cost coefficient for L2 weight-decay typically range from 0.01 to 0.00001.

Weight initialization Weight initialization values represents the starting value of the RBM weights that amplify or mute the input signal coming into each node. Proper weight initialization help training training time. In our case a trial has been made with Glorot uniform weight initialization as described in [25], in the perspective of achieving faster convergence.

5 Dataset Description

The UNB ISCX Intrusion Detection Evaluation Dataset was developed in order to provide a quality dataset for network intrusion detection research [21]. The approach for defining this dataset involved identifying features that would allow effective detection, while minimizing processing costs. Each record of the dataset is characterized by features that fall into three categories: basic, content, and traffic. These features are described in [5, 24].

The UNB ISCX dataset is composed of sequences of entries in the form of records labeled as either *normal* or *attack*. Each entry contains a set of characteristics of a *flow*, i.e., of a sequence of IP packets starting at a time instant and ending at another, between which data flows between two IP addresses using a transport-layer protocol (TCP, UDP) and an application-layer protocol (HTTP, SMTP, SSH, IMAP, POP3, or FTP). The dataset is fairly balanced with prior class probabilities of 0.466 for the *normal* class and 0.534 for the *anomaly* class.

This dataset is composed of two sub-datasets: a *train dataset*, used for training a NIDS, and a *test dataset*, used for testing. Both have the same structure and contain all four types of attacks. However, the test dataset has more attacks as shown in Table 2, to allow evaluating the ability of algorithms to generalize. The train dataset has around 2.2 GB of data; the test dataset has 0.8 GB.

Table 2. Attacks in the UNB ISCX train / test datasets (all attacks from the first exist also in the second)

Class	Train dataset attacks	Test dataset only attacks
Probing	portsweep, ipsweep, satan, guesspasswd, spy, nmap	snmpguess, saint, mscan, xsnoop
DoS	back, smurf, neptune, land, pod, teardrop, buffer overflow, warezclient, warezmaster	apache2, worm, udpstorm, xterm
R2L	imap, phf, multihop	snmpget, httptunnel, xlock, sendmail, ps
U2R	loadmodule, ftp write, rootkit	sqlattack, mailbomb, processtable, perl

6 The Experiment

We used the DArch R Package package [12, 13] in the experiments. This package allows generating deep architecture networks and training them. All parameters not explicitly referred took the default values.

Table 3. Error Estimation (averages for 5 trials)

	16	17	18	19	20	25	30
Training MSE	0.137	0.120	0.122	0.123	0.109	0.115	0.106
Training classification error	5.55%	5.92%	6.06%	5.79%	5.86%	5.94%	5.75%
Validation MSE	0.137	0.120	0.122	0.123	0.110	0.115	0.107
.632+ MSE	0.137	0.120	0.122	0.123	0.110	0.115	0.107
Validation classification error	5.65%	5.88%	5.99%	5.73%	5.89%	5.94%	5.75%
.632+ classification error	5.61%	5.89%	6.02%	5.75%	5.88%	5.95%	5.75%

In order to make these features suitable inputs for the visible layer of the RBM, we normalized them to the range [0,1], and treated them as continuous values. Cross-validation, sometimes called rotation estimation, is a validation technique for assessing how the results of a statistical analysis will generalize to an independent dataset. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (the training set), and validating the analysis on the other subset (the validation or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

For the starting RBM hidden nodes dimensioning in the experiment, a technique similar to the gradual drop-in of nodes (increasing the number of nodes in each step of the experience) was used with reference values near the $\lceil v/2 \rceil$. The numbers of nodes used in the experience were $16, 17, 18, 19, 20$, plus $25, 30$ for comparison purposes. This has been suggested to be a sufficient approximation for obtaining good results in training RBMs [28]. The most relevant parameters for the RBM as used are depicted in Table 1. This thesis has been validated since the best performance figures are shown to be obtained for an internal node number of 16.

We performed an empirical study to compare the .632+ bootstrap estimator with the repeated 10-fold cross-validation. The results for the pre-training and fine tuning phases were obtained with 125973 samples and 40 predictor variables. The accuracy was used to select the optimal model using the largest value. Preprocessing was used to re-scale the features to the range $[0, 1]$. Bootstrapping was started with 125973 samples and resulted in 79598 unique training samples and 46375 validation samples for this run. Training data was shuffled before each epoch and the final result for each run was taken from the average results of each epoch. The results for the classification and network errors are summarized in Table 3 and put in contrast in Figures 5 and 6. By looking at the .632+ statistics value we can observe that a negligible amount of bias for the error values is present, since this statistics results similar to the respective values.

In Figure 5 the training and validation Classification Errors are presented as a graphical example of the results from Table 3. The classification error is the error of the classification given by the RBM for a validation and a training set. In Figure 6(a) an example of error convergence for the case of an RBM with 16 internal nodes is presented. Similarly, in Figure 6 the training and validation Mean Square Errors are presented as a graphical example of the results from

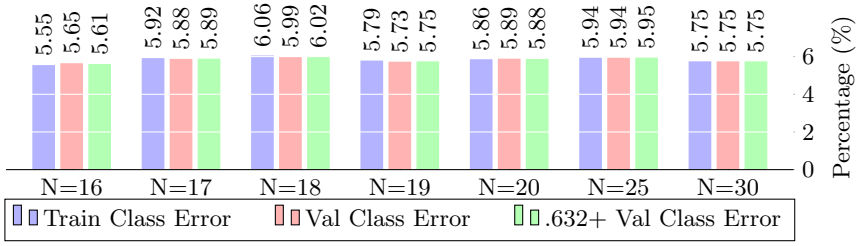


Fig. 5. Experimental results for training and validation classification errors (Table 3)

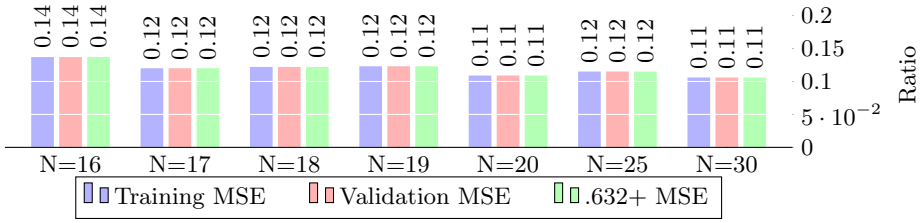


Fig. 6. Experimental results for network training and validation errors from Table 3

Table 3. The MSE is the average difference between the expected and the actual output, and it contains both the variance and bias of the estimator. In Figure 7(b) an example of MSE convergence for the case of a RBM with 16 internal nodes is presented.

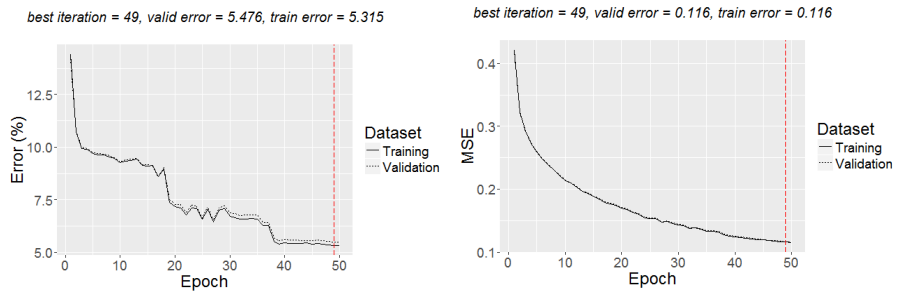
Figure 7 presents the results for an example with internal node number of 16, showing good convergence properties (with maximum of 50 epochs).

7 Conclusions

The application of Restricted Boltzmann Machines in the network intrusion detection field has been gaining momentum. In this work it has been shown that it is possible to surpass some performance obstacles by means of proper RBM optimization given a set of optimized choices for the remaining parameters. Among these the most troublesome is the fact that for using RBMs it is relevant to process real valued features in the $[0, 1]$ range therefore the need for transforming the original values accordingly for this range. Good results have been obtained

Table 4. Performance metrics

Metrics	Drop = 0	Drop = 0.1	Drop = 0.5
Accuracy	0.7609	0.7634	0.7549
Kappa	0.5400	0.5414	0.5258
Specificity	0.9666	0.9368	0.9353
Sensitivity	0.6063	0.6331	0.6194
Neg Pred Value	0.6485	0.6574	0.6487
Pos Pred Valuee	0.9603	0.9302	0.9272
Balanced Accuracy	0.7865	0.7850	0.7773



(a) (b)

Fig. 7. Errors for iterations with 16 internal nodes: (a) Classification error; (b) MSE

for the Network Training MSE and Network Validation MSE. In terms of classification the results obtained were also comparable to the best obtained till now in NIDS using RBMs [5].

Acknowledgements. This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

1. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.
2. Y. Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.
3. Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.
4. H. Chen and A. F. Murray. Continuous restricted Boltzmann machine with an implementable training algorithm. *Vision, Image and Signal Processing, IEEE Proceedings*, 150(3):153–158, 2003.
5. U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomput.*, 122:13–23, Dec. 2013.
6. G. Hinton. A practical guide to training restricted Boltzmann machines. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *LNCS*, pages 599–619. Springer, 2012.
7. G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
8. G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.
9. G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs.NE]*, July 2012.
10. G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

11. G. E. Hinton. A Practical Guide to Training Restricted Boltzmann Machines. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *LNCS*, chapter 24, pages 599–619. Springer, 2nd edition, 2012.
12. G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.
13. G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
14. E. Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.
15. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, 2012.
16. J. R. Movellan, P. Mineiro, and R. J. Williams. A monte carlo em approach for partially observable diffusion processes: Theory and applications to neural networks. *Neural Computation*, 14:1507–1544, 2002.
17. V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In J. Frnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814. Omnipress, 2010.
18. L. Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.
19. F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.
20. R. Salakhutdinov. Learning in markov random fields using tempered transitions. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 1598–1606. Curran Associates, 2009.
21. A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, May 2012.
22. L. N. Smith, E. M. Hand, and T. Doster. Gradual dropin of layers to train very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4763–4771, 2016.
23. P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 194–281. MIT Press, 1986.
24. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, pages 53–58, 2009.
25. Y. W. Teh and D. M. Titterington, editors. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.
26. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, Dec. 2010.
27. M. Welling and Y. W. Teh. Linear response algorithms for approximate inference in graphical models. *Neural Computation*, 16:197–221, 2004.
28. X. Zhao, Y. Hou, Q. Yu, D. Song, and W. Li. Understanding deep learning by revisiting Boltzmann machines: An information geometry approach. *CoRR*, abs/1302.3931, 2013.