2019 IEEE European Symposium on Security and Privacy Workshops

Exploiting the Auto-Encoder Residual Error for Intrusion Detection

Giuseppina Andresini, Annalisa Appice, Nicola Di Mauro, Corrado Loglisci, Donato Malerba
Department of Computer Science
University of Bari Aldo Moro
Bari, Italy
Email: {giuseppina.andresini,annalisa.appice,nicola.dimauro,corrado.loglisci,donato.malerba}@uniba.it

***Abstract*—Intrusion Detection Systems aim to address the problem of correctly identifying unforeseen network attacks. The attack detection problem has been already tackled through supervised and unsupervised machine learning approaches. While the former methods lead to models very accurate on already seen samples, the latter provide models robust on unforeseen samples by trading-off a high accuracy on seen ones. In this paper, we combine deep unsupervised neural networks with supervised neural networks aiming at improving the classification accuracy on unforeseen attacks. Auto-encoders neural networks are used both for feature engineering and as anomaly detectors. Experimental results on a challenging dataset prove the validity of the proposed approach when compared to other state-of-the-art methods.**

I. Introduction

Identifying various network attacks—especially unforeseen ones—represents one of the main key problems in ensuring information security, a problem generally addressed adopting a network Intrusion Detection System (IDS). Many shallow supervised Machine Learning approaches have been widely used in predicting various types of attacks [1], [2], [3], relying on a well done feature engineering and selection. However, supervised machine learning techniques usually fail in identifying unknown attacks that are not present in the training data. On the contrary, anomaly based IDSs are more robust against new attacks (anomalies) since they are learnt to identify normal behaviour patterns.

Recent Deep Learning architectures have achieved remarkable improvements in many real-world scenario such as speech recognition [4], image recognition [5], and natural language processing [6]. Since deep learning models are able to learn better representations from data, here we propose a new deep learning approach for an intrusion detection system using auto-encoders. An auto-encoder is an artificial neural networks (NNs) consisting of an encoder function $\mathbf{h} = f(\mathbf{x})$—mapping the input $\mathbf{x}$ to a hidden code $\mathbf{h}$—and a decoder producing the reconstructed input $\hat{\mathbf{x}} = g(\mathbf{h})$, learned by minimizing a loss function $\mathcal{L}(\mathbf{x}, g(f(\mathbf{x})) = \mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ penalising $\mathbf{x}$ for being dissimilar from $\hat{\mathbf{x}}$ such as $\mathcal{L}_{\text{mse}}(\mathbf{x}, \hat{\mathbf{x}}) = 1/n \sum_i ||x_i - \hat{x_i}||^2$.

Given a set $\mathbf{X} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ of N data samples, whose binary labels y_i denote a *normal* or an *attack* sample and each $\mathbf{x}_i$ denotes an input sample defined over M features, our main assumption is that the normal data samples share a distribution $p_n(\mathbf{x})$. Hence, the first task we investigate is designing a generative approach able to estimate the $p_n(\mathbf{x})$ distribution from $\mathbf{X}$ and assess whether a sample is normal or not, as already done for positive-unlabelled learning [7]. Instead of learning a probability distribution, the same problem can be tackled exploiting an auto-encoder, which maps the normal data samples into a low-dimensional manifold, where attack data samples cannot be well mapped, as already done for anomaly detection [8]. Indeed, after having learned such an auto-encoder from normal data samples, we can improve its discriminating ability by making it "aware" on the reconstruction error through a feature augmentation step. Thus, new features built on the residual error $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$, associated with each data point, can be exploited to distinguish normal from attack data in an unsupervised fashion.

Differently from a classical anomaly detection approach [8], here we argue that this first unsupervised step can be followed by a supervised one. Indeed, the residual error is an indicator—like a new hidden label—of a normal data, and each training point $\mathbf{x}$ can be augmented by adding this new learnt feature $e = \mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ leading to the augmented dataset, denoted as $\mathbf{X}_{\cup\{e\}}$. Finally, a multi-layer perceptron is fine-tuned for classifying normal and attack data on this new augmented dataset. The idea is to integrate the ability of the auto-encoder in detecting anomalies into the supervised task.

Furthermore, as we will prove in the experimental evaluation section, even if this newly learned feature is able to improve the score of the classifier at testing, there are still some data points that are not correctly classified, which typically follow the data distribution of unforeseen attacks. Mis-classifications can be reduced by using an additional step of auto-encoding—learnt on the same augmented normal data points—as an anomaly detector.

The experimental results on a challenging dataset prove the validity of our proposed approach when compared to other state-of-the art deep learning based methods.

The key contributions of our work are *a)* the combination of deep supervised and unsupervised NNs for intrusion detection; *b)* the use of the residual error of auto-encoders as feature engineering; and *c)* an automatic method to define a threshold criteria to be used by the auto-encoder based anomaly detector.

This paper is organised as follows. The related literature is illustrated in the next Section. The formulated intrusion detection method is described in Section III. The findings in

DOI 10.1109/EuroSPW.2019.00038

the evaluation of the proposed method are discussed in Section IV-D. Finally, Section V refocuses on the purpose of the research, draws conclusions and proposes future developments.

II. Related work

The prior research on intrusion detection is populated with approaches based on traditional supervised and unsupervised machine learning algorithms working on raw data of computer network activities [1], [9], [2], [3], [10], which have turn out to be poorly performant as the attacks become more sophisticated. The reason can be found in the non-linearity property which always more frequently characterises the attacks and which has opened to the research line on Deep Learning due to the peculiarities to deal with the non-linearity of high-dimensional data and to learn new representational forms of the attacks [11].

Among the recentest contributions, [12] combines a neural feature learning schema with conventional Support Vector Machines (SVMs) and prove that SVMs perform better when compared to shallow learning algorithms. The new feature representation is built using a sparse auto-encoder mechanism on the whole NSL-KDD dataset[1]—including both training and testing samples—without considering the labelling information. Sparse auto-encoder based solutions has been tested also in conjunction with recurrent neural networks (RNNs) [13], but [14] proved that the auto-encoders are not so beneficial for RNNs. The authors tested different configurations—by setting hidden layers and learning rate—which perform better than traditional classifiers, but requiring greater learning times. [15] carries on a more comprehensive experimental study aiming at strengthening anomaly detectors with respect to unseen data. To do that, testing samples are removed from initial input data and used later on for the evaluation. They tested classifiers designed with different neural models, but achieve the better performance and model generality with Long short-term memory (LSTM) RNNs. Also in this case, no preliminary operation of feature engineering or dimensionality reduction has been adopted, so the original representational space is forced to work on different input layers—sequential input layers with LSTM or two-dimensional inputs for Convolutional neural networks (CNNs). Similarly, [16] separates training and testing portions and use the unlabelled training data to build new features through a neural approach—auto-encoders. Finally, they add the label information to the new representation to learn a conventional soft-max predictor. [17] tries to capture the structural content of the network activities through an image conversion format, but, as also proved in [18], they ascertain that there is no a CNN-solution which offers stably good results compared to alternative neural models or even traditional classifiers. This confirms that combining two neural processing stages—one to determine a new feature space and the other one to train the anomaly detector—is beneficial rather than using neural models only for learning raw data. However, while the above mentioned studies basically converge on the necessity to map the original features into a reduced set of latent characteristics, few interest has been addressed to account for performances of feature learning stage, which, especially in presence of data imbalance, could under-fit the types of attacks affected by data scarcity. Techniques of feature augmentation could provide a viable approach through the construction of additional attributes. In domains different from intrusion detection, these techniques have led to consistent reduced prediction error, even in the semi-supervised setting [19], while, in the domain of information security, we can mention only the method of [20] designed for conventional SVMs. No attempt seems have been done with Deep learning solutions.

[1]https://www.unb.ca/cic/datasets/nsl.html

III. The proposed method

In this section we describe the proposed supervised IDS, named AIDA (Auto-encoding-based Anomaly Detection for Intrusion Detection via ClAssification). AIDA performs a three-stage training phase (see Figure 1). In the first stage, it applies a feature selection mechanism, in order to reduce the number of features by discarding the less relevant ones. In the second stage, AIDA learns a classification model able to distinguish attacks data samples from normal ones. The classification model is learned from a training set augmented with the residual error of an auto-encoder. Finally, in the last stage, AIDA builds an anomaly detector able to identify attack samples that have been classified as normal. The anomaly detection stage works as a post-classification step aiming to improve the IDS performance by reducing the number of attacks missed by the classifier. Both the classifier and the anomaly detection model will be jointly used in cascade in order to predict the target label of unseen test samples.

A. Feature selection

Feature selection consists of detecting the relevant features and discarding the irrelevant ones with the goal of obtaining a subset of features that describe properly the given problem with a negligible degradation of performance. This stage is performed according to literature studies [21], [22], which have identified feature selection as a pre-processing phase in attack defence that can potentially increase classification accuracy and reduce computational complexity.

We rank features according to how they gain in classification accuracy and retain the top-ranked features only. Similarly to [23], [24], we compute the Information Gain (IG) in order to measure the worth of every input feature. The IG is based on the concept of entropy, that can be viewed as a measure of uncertainty of the system—the higher the entropy the more the information content. A feature that does not have much effect on the data classification has very small IG and it can be ignored without affecting the accuracy of a classifier. Based upon this consideration, features are ranked into the training set according to the IG measure so that the top-k features can be retained.

After the feature selection process, all the symbolic features have been transformed into a numeric form using a one-

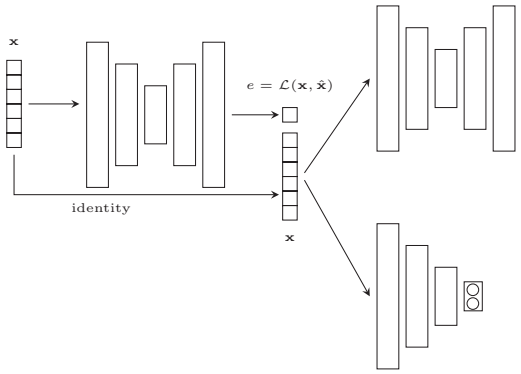


Fig. 1. *The AIDA model*. The model takes as input all the available training samples $\mathbf{X}$. The normal samples are used to learn the first auto-encoder (left), that is then employed—for each training sample $\mathbf{x}$, both normal and attack samples—in order to compute $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$. Each training sample is then augmented with a new feature corresponding to this lastly computed residual error. This new augmented dataset is used to learn both the new auto-encoder (top-right)—again using the normal samples only—leading to an anomaly detector, and the classifier (bottom-right).

hot encoding mapping—a transformation also usually used in intrusion detection literature [25], [26], [27], [28]. While, numeric features have been scaled in order to reduce the distances between the points and put them on similar scale [25], [27].

B. Residual-error feature augmentation

By accounting for the auto-encoding theory [29], the loss, or the residual error of each sample, of an auto-encoder can be used as a measure to detect anomalies. In particular when the auto-encoder is trained on normal samples only, it should be used to detect attack samples. The underlying assumption is that an auto-encoder trained from normal samples will re-create attack samples poorly [30], [31]—with a high loss. Based upon this assumption, although auto-encoding is mostly used in the literature for feature learning and dimensionality reduction, we are going to use it as an anomaly detector exploiting the residual error of each sample like in [8].

Let $\mathbf{X} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$ be a set of N training samples, whose binary labels y_i denote a *normal* or an *attack* sample and each $\mathbf{x}_i$ denotes an input sample defined over M features—constructed during the feature selection stage. Furthermore, we denote with $\mathbf{X}^n$ the subset of $\mathbf{X}$ restricted to the samples $\mathbf{x}_i$ whose corresponding label y_i is normal, i.e. $\mathbf{X}^n = \{\mathbf{x}_i \in \mathbf{X} | y_i = \text{normal}\}$. We train two auto-encoders, denoted as $\mathcal{A}_1$ and $\mathcal{A}_2$, which empower, respectively, the classification and the anomaly detection stage. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ are trained from $\mathbf{X}^n$ by minimising their mean squared reconstruction errors. This choice depends on the fact that both auto-encoders are intended as unsupervised models of the normal behaviour.

The auto-encoder $\mathcal{A}_1$ trained from $\mathbf{X}^n$ is used to engineer a new feature e_i for each sample $\mathbf{x}_i$ useful in the classification stage. In particular, after having trained $\mathcal{A}_1$, e_i represents the residual error $\mathcal{L}_{\mathcal{A}_1}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$ computed with $\mathcal{A}_1$. The rationale of this feature engineering process can be found in the assumption, reported above, on the residual error of auto-encoders trained from normal data samples. Our hypothesis (that will be empirically checked) is that the entire intrusion detection model can actually gain in accuracy in case the residual errors of the auto-encoder is explicitly enclosed in classification model.

We denote by $\mathbf{X}_{\oplus e}$ the augmented dataset obtained from $\mathbf{X}$ exploiting the auto-encoder $\mathcal{A}_1$ learned from the training samples in $\mathbf{X}$, i.e., $\mathbf{X}_{\oplus e} = \{\mathbf{x}_i \oplus (e_i), y_i | e_i = \mathcal{L}_{\mathcal{A}_1}(\mathbf{x}_i, \hat{\mathbf{x}}_i\}_{i=1}^{N}\}$.

The training samples in the augmented dataset $\mathbf{X}_{\oplus e}$ are used to train both the classifier and a second auto-encoder $\mathcal{A}_2$. The classifier has been obtained with a multilayer perceptron with a final softmax layer. The auto-encoder $\mathcal{A}_2$ is used as an anomaly detector for the testing samples, which are predicted as normal by the classification model. Let $e'_i = \mathcal{L}_{\mathcal{A}_2}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$, for each $\mathbf{x}_i \in \mathbf{X}_{\oplus e}$, be the residual error when a sample is reconstructed with $\mathcal{A}_2$. The anomaly-based post-classification stage uses a threshold-based approach and post-classifies as attack samples all the samples classified as normal by the classifier when their residual error is greater than (or equal to) an anomaly threshold (see Section III-C).

C. Residual-error anomaly-based post-classification

To learn the anomaly-based post-classification model, we first train the auto-encoder $\mathcal{A}_2$ from the training normal samples in $\mathbf{X}_{\oplus e}$.

Then we use $\mathcal{A}_2$ to compute the residual error $e'_i = \mathcal{L}_{\mathcal{A}_2}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$ for each training sample in $\mathbf{X}_{\oplus e}$. Finally, we learn a threshold α so that the rule "if $e'_i \geq \alpha$ then attack else normal" can be applied, in order to post-classify any sample that is initially detected as normal from the classifier.

Let $\mathbf{e}' = \{e'_i\}_{i=1}^{T}$ the set of residual errors computed on the $T < N$ training samples in $\mathbf{X}_{\oplus e}$. The anomaly threshold α is automatically determined as a value in the interval $]0, \alpha_{IG}]$, where α_{IG} is the threshold of the rule "if $e'_i \geq \alpha_{IG}$ then attack else normal" that maximises the following Information Gain criterion:

$$\alpha_{IG} = \underset{\alpha' \in \mathbf{e}'}{\operatorname{argmax}} IG(Y, \alpha') = H(Y) - \underset{\alpha' \in \mathbf{e}'}{\operatorname{argmax}} H(Y|\alpha'), \quad (1)$$

where, Y is the random variable corresponding to the training labels y_i, and $H(Y)$ is the entropy estimator derived from the empirical class probabilities, i.e.,

$$H(Y) = -p_n(Y)\log p_n(Y) - p_a(Y)\log p_a(Y),$$

being $p_n(Y) = \sum_{y\in Y} \mathbb{1}(y = \text{normal})$ and $p_n(Y) = \sum_{y\in Y} \mathbb{1}(y = \text{attack})$; while,

$$H(Y|\alpha') = \frac{-|Y_{\leq\alpha'}|H(Y_{\leq\alpha'}) - |Y_{>\alpha'}|H(Y_{>\alpha'})}{T},$$

where, $Y_{\circ\alpha'} = \{y_i \in Y | e'_i \circ \alpha\}$.

We note that Equation 1 guarantees that rule "if $e'_i \geq \alpha_{IG}$ then attack else normal" achieves the highest accuracy if it is used to detect attack samples. Although this consideration suggests that α_{IG} may be a suitable anomaly threshold for the post-classification stage, we do not actually use it in AIDA. This decision is motivated by the risk of yielding a post-classification rule that overfits training data if we roughly use the threshold identified with the Information Gain criterion. The problem with the overfitting risk is that it may yield poor performance when the possibly overfitted threshold is expected to recognise unseen zero-day attacks (which may be so close to normal training data to exhibit a residual error also lower than α_{IG}).

To actually diminish the overfitting risk, we evaluate the series of post-classification rules which may be formulated with α ranging from α_{IG} down to zero. For each candidate rule, we measure its training classification accuracy. We determine $avgAcc$ that is the average accuracy for this series of rules. Formally,

$$avgAcc = \underset{\alpha\in]0,\alpha_{IG}]}{\text{avg}} \; accuracy(c_\alpha), \tag{2}$$

where c_α denotes the classification rule "if $e'_i \geq \alpha$ then attack else normal" and $accuracy(c_\alpha)$ denotes the training classification accuracy of c_α. Finally, we choose an anomaly threshold α that is the highest value of e'_i, selected in the interval $]0, \alpha_{IG}]$, which is associated with the rule whose training accuracy is lower than $avgAcc$. In particular,

$$\alpha = \underset{\alpha'\in]0,\alpha_{IG}]}{\text{argmax}} \; accuracy(c_{\alpha'}) \leq avgAcc, \tag{3}$$

In the implementation of this search, we evaluate candidate thresholds, which are equally spaced from α_{IG} down to zero with a user-defined space size. For example, in Figure 2 we report the threshold α learned for KDDTrain$^+$ (see Section IV-D) by exploring the interval $]0, \alpha_{IG}]$ with $\alpha_{IG} = 0.009$ and space size equal to 0.001.

D. Prediction

Let us consider a new unseen sample $\mathbf{x_i}$. We apply the cascade composed of both the classification model c: $\mathbf{X}_{\oplus e} \mapsto \{attack, normal\}$ and the anomaly-based post classification rule "if $A_2 \geq \alpha$ then attack else normal" as they have been trained by AIDA. First, we calculate residual error e_i of the reconstruction of $\mathbf{x_i}$ with auto-encoder $\mathcal{A}_1$. Then we apply classification model c so that if c classifies $(\mathbf{x_i}, e_i)$ as an attacking sample, we definitely label the sample under consideration with class 'attack'. Otherwise, we post-process the sample according to the anomaly-based rule. This means that we calculate residual error e'_i of the reconstruction of $(\mathbf{x_i}, e_i)$ with auto-encoder $\mathcal{A}_2$. If $e'_i \geq \alpha$, we re-classify the sample under consideration in class 'attack'; otherwise we leave the sample labelled with class 'normal'.

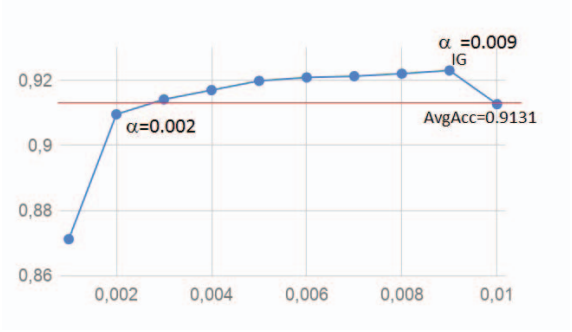


Fig. 2. Anomaly threshold α (axis X) learned on the training set KDDTrain$^+$ based on Overall accuracy (axis Y).

IV. Empirical Study

In this Section we firstly describe—in Section IV-A—the benchmark NSL-KDD dataset adopted in our empirical study. Then—in Section IV-B—we illustrate the result of the feature selection step when applied on the adopted dataset. In Section IV-C we report the implementation details of the artificial neural networks considered to train both the auto-encoders and the classifier. Finally—in Section IV-D—we discuss the obtained results comparing the performance of AIDA against state-of-the-art competitors.

A. The NSL-KDD Dataset

We consider the NSL-KDD dataset[2] introduced in [32], a commonly used benchmark dataset for the evaluation of the accuracy of intrusion detection systems—even recently [27], [28], [33], [34]. It is derived from the KDDCUP'99 dataset, in order to overcome the learning difficulties posed by the huge number of observed redundant data samples. This redundancy condition can in fact lead to bias the performance of learning algorithms towards the most frequent data samples [32]. In addition, the KDDCUP'99 dataset is unbalanced—19.69% of normal samples and 80.31% of attack ones—while removing redundancy, it becomes a quite balanced one—53.46% of normal samples and 46.54% of attack ones.

Tavallaee et al. [32] proposed a specific data setting to perform the experiments with NSL-KDD. This setting includes one training set, denoted as KDDTrain$^+$, and two testing sets, denoted as KDDTest$^+$ and KDDTest^{-21}. Both testing sets comprise data samples belonging to attack families for which no sample is available in the training set, thus simulating a *zero-day* attack condition. KDDTest^{-21} is a reduced and more

[2]https://www.unb.ca/cic/datasets/nsl.html/.

TABLE I
NUMBER OF SAMPLES IN BOTH THE TRAINING AND TESTING SETS

Dataset	Total	Non-Attacks	Attacks
KDDTrain$^+$	125973	67343	58630
KDDTest$^+$	22544	9711	12833
KDDTest^{-21}	11850	2152	9698

complex version of KDDTest$^+$, obtained by removing samples in KDDTest$^+$ correctly classified by the IDS proposed in [32]. We replicate the data setting described in [32] for our empirical setting since it is commonly adopted as benchmark in the literature [12], [14], [26], [15].

The number of samples collected in both the training and testing sets is reported in Table I. Each sample consists in a vector of 41 input features—6 are binary, 3 are symbolic, and 32 are quantitative, as described in Table II—and 1 output label. The input features are categorised into three types:

- *basic*: 1 – 10, extracted from the TCP/IP connection;
- *content*: 11 – 22, describing a suspicious behaviour; and
- *traffic*: 23 – 41, accumulated in a window interval for connections which involve either the same destination host or the same service.

B. Feature selection

The feature selection stage is performed by retain the 10 top-ranked features according to Information Gain (IG) as reported in [23], thus comprising two symbolic features and eight quantitative features as described in Table II (selected features are marked with (*)). After applying the one-hot encoder mapping to transform the selected symbolic features in quantitative ones, an input feature space with 89 quantitative features is finally constructed. This input feature space is expanded with the addition of the residual error feature that is engineered using the auto-encoder trained on the non-attacking training data.

C. Implementation details

AIDA has been implemented in Python whose source code is available at https://github.com/giusy123/AIDA. The neural networks of both the auto-encoders and classifiers architectures are developed in Keras 2.2[3]—a high-level neural network API—with TensorFlow[4] as back-end.

The auto-encoder architecture consists of five hidden layers. The encoder reduces the input feature space through layers with 60, 30 and 10 neurons in the bottleneck layer. The decoder maps the bottleneck signals back to the input space through two layers with 30 and 60 neurons. The mean squared error (*mse*) has been used as the loss function to be optimised and the hyperbolic tangent (*tanh*) has been selected as activation function for each layer.

The classification architecture is a deep neural network with three intermediate layers (with 80, 40 and 20 neurons) equipped with *tanh* as activation function and a softmax layer. This architecture assigns samples to classes 'normal' or 'attack' by minimising the binary-cross entropy as loss function. A dropout layer is placed before the the softmax layer, in order to perform data regularisation and prevent the overfitting.

Both architectures described above are trained for a maximum number of epochs equal to 150, retaining the best models achieving the lowest loss on a validation set—fixed as 10% of the training set.

D. Results and discussion

1) Evaluation metrics: We evaluate the intrusion detection accuracy in terms of overall accuracy (OA), average accuracy (AA), precision (P), recall (R), F-measure (F) and false alarm rate (FAR). These metrics are selected as they are commonly used in the evaluation of intrusion detection systems. A short description of these metrics is reported in Table III.

In addition, we evaluate the intrusion detection efficiency in terms of both training and testing time. The training time is measured as the computational time spent in seconds to complete the training of the IDS model. The testing time is measured as the average computational time spent in predicting each testing sample. All the experiments have been executed on a Linux machine with an Intel CORE i7-4700HQ CPU @ 2.40GHZ and 16GB RAM.

2) Component analysis: We study the effectiveness of both the classification and anomaly-based post classification stage in AIDA. To this purpose, we consider five configurations, which are defined as follows:

- $\mathcal{A}_1$: samples are classified exploiting the auto-encoder $\mathcal{A}_1$ as a threshold-based anomaly-detector;
- $\mathcal{C}_1$: samples are classified using a classifier trained on $\mathbf{X}$;
- $\mathcal{C}_1 + \mathcal{A}_1$: samples are firstly classified with a classifier trained on $\mathbf{X}$ and then all the samples predicted as normal are post-processed using the auto-encoder $\mathcal{A}_1$ as a threshold-based anomaly-detector;
- $\mathcal{C}_2$: sample are classified using a classifier trained on the augmented dataset $\mathbf{X}_{\oplus e}$;
- AIDA: the sample are firstly classified as in $\mathcal{C}_2$, but then all the samples predicted as normal are post-processed using the auto-encoder $\mathcal{A}_2$ as a threshold-based anomaly-detector.

We start analysing the accuracy of the configurations $\mathcal{A}_1$, $\mathcal{C}_1 + \mathcal{A}_1$ and AIDA. All these configurations employ the threshold-based anomaly detection as a stage of the intrusion detection system. We expect that the performance of this stage may depend on the selected anomaly threshold. Based upon this consideration, we explore the sensitivity of the accuracy of these configurations w.r.t. the value of the anomaly threshold. In the case of $\mathcal{A}_1$ and $\mathcal{C}_1 + \mathcal{A}_1$, this analysis aims to select the best anomaly threshold (i.e., the threshold that achieves the highest accuracy for each configuration under consideration) to be considered in the subsequent comparative study (see results in Table IV). In the case of AIDA, this analysis aims to provide an empirical evidence of the effectiveness of the

[3]https://keras.io/
[4]https://www.tensorflow.org/

TABLE II

FEATURE DESCRIPTION.FOR EACH SYMBOLIC FEATURE, THE NUMBER OF DISTINCT VALUES IS REPORTED BETWEEN PARENTHESES. FEATURE IDENTIFIED AS RELEVANT DURING THE FEATURE SELECTION PHASE ARE MARKED WITH (*).

No.	Feature	Type	No.	Feature	Type	No.	Feature	Type
1	duration	Quantitative	15	su_attempted	Binary	29	same_srv_rate (*)	Quantitative
2	protocol_type	Symbolic (3)	16	num_root	Quantitative	30	diff_srv_rate (*)	Quantitative
3	service (*)	Symbolic (70)	17	num_file_creations	Quantitative	31	srv_diff_host_rate	Quantitative
4	flag (*)	Symbolic (11)	18	num_shells	Quantitative	32	dst_host_count	Quantitative
5	src_bytes (*)	Quantitative	19	num_access_files	Quantitative	33	dst_host_srv_count (*)	Quantitative
6	dst_bytes (*)	Quantitative	20	num_outbound_cmds	Quantitative	34	dst_host_same_srv_rate (*)	Quantitative
7	land	Binary	21	is_host_login	Binary	35	dst_host_diff_srv_rate (*)	Quantitative
8	wrong_fragment	Quantitative	22	is_guest_login	Binary	36	dst_host_count_src_port_rate	Quantitative
9	urgent	Quantitative	23	count	Quantitative	37	dst_host_srv_diff_host_rate	Quantitative
10	hot	Quantitative	24	srv_count	Quantitative	38	dst_host_serror_rate (*)	Quantitative
11	num_failed_login	Quantitative	25	serror_rate	Quantitative	39	dst_host_srv_serror_rate	Quantitative
12	logged_in	Binary	26	srv_serror_rate	Quantitative	40	diff_host_rerror_rate	Quantitative
13	num_compromised	Quantitative	27	rerror_rate	Quantitative	41	dst_host_srv_rerror_rate	Quantitative
14	root_shell	Binary	28	srv_rerror_rate	Quantitative			

TABLE III

ACCURACY METRICS: OVERALL ACCURACY (OA), AVERAGE ACCURACY (AA), PRECISION (P), RECALL (R), F-MEASURE (F) AND FALSE ALARM RATE (FAR). THESE METRICS ARE COMPUTED BY ACCOUNTING FOR THE NUMBER OF TRUE POSITIVE – TP (NUMBER OF ATTACKS CORRECTLY DETECTED), THE NUMBER OF TRUE NEGATIVE – TN (NUMBER OF NON-ATTACKS CORRECTLY DETECTED), NUMBER OF FALSE POSITIVE – FP (NUMBER OF NON-ATTACKS INCORRECTLY DETECTED AS ATTACKS) AND NUMBER OF FALSE NEGATIVE – FN (NUMBER OF NON-ATTACKS INCORRECTLY DETECTED AS ATTACKS.

Accuracy metric	Mathematical formulation
OA	$\frac{TP+TN}{TP+TN+FP+FN}$
AA	$\frac{AC_P+AC_N}{2}$
P	$\frac{TP}{TP+FP}$
R	$\frac{TP}{TP+FN}$
F	$2 \cdot \frac{P \cdot R}{P+R}$
FAR	$\frac{FP}{FP+FN}$

strategy described in Section III-C to correctly select the anomaly threshold.

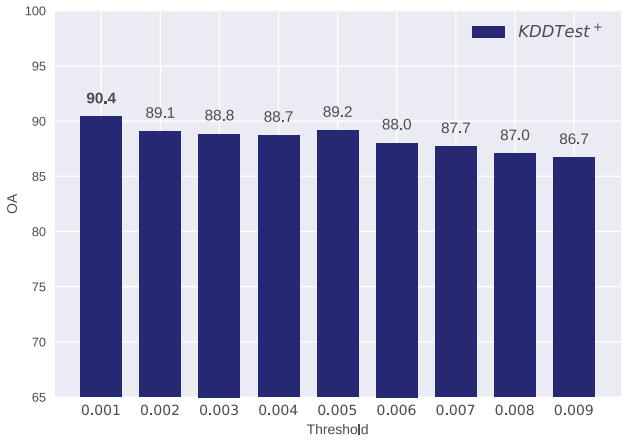
We analyse the OA of $\mathcal{A}_1$, $\mathcal{C}_1+\mathcal{A}_1$ and AIDA with the anomaly threshold ranging from 0.001 to 0.009. We select 0.009 as the maximum of the interval since the Information Gain criterion suggests this value as the ideal threshold to separate training attack samples from training normal samples in all these three configurations. Results, reported in Figure 3, show that $\mathcal{A}_1$ and $\mathcal{C}_1 + \mathcal{A}_1$ achieve the highest OA with the anomaly threshold equal to 0.001. On the other hand, the OA of AIDA always achieves the peak when testing samples are post-processed with anomaly threshold equal to 0.002. This value corresponds to the anomaly threshold that is automatically identified by the proposed threshold learning strategy (see details in Figure 2). To confirm this conclusion, we also report ROC curves computed for AIDA by varying the anomaly threshold. They are plotted in Figures 4(a) and 4(b) for KDDTest$^+$ and KDDTest^{-21}, respectively. Red points marked on ROC curves correspond to the configuration with anomaly threshold automatically set equal to 0.002. We note that high separability between classes is already achieved at these points. In addition, the area under the ROC curves (0.94 for KDDTest$^+$ and 0.85 for KDDTest^{-21}) confirms that AIDA achieves good separability between classes in both testing sets, although the accuracy is the highest in the simpler scenario – KDDTest$^+$.

We proceed comparing the accuracy of all the selected configurations. Accuracy metrics are collected in Table IV. The analysis of OA, AA and F confirms that AIDA outperforms all its baseline configurations. As these metrics express how each configuration is able to identify the correct class (both 'attack' and 'normal') of each testing sample, this result indicates that the best trade-off between accurately detecting both attack and normal samples is actually achieved by considering the auto-encoding information and performing both the classification and the anomaly-based post-classification stage. Additional considerations to support this evidence can be drawn from the joint analysis of P, R and FAR. Even if the configurations $\mathcal{C}_1$ and $\mathcal{C}_2$ outperform AIDA in terms of P and FAR, they also achieve the lowest R. High precision and low false alarm rate indicates that $\mathcal{C}_1$ and $\mathcal{C}_2$ yield a low number of

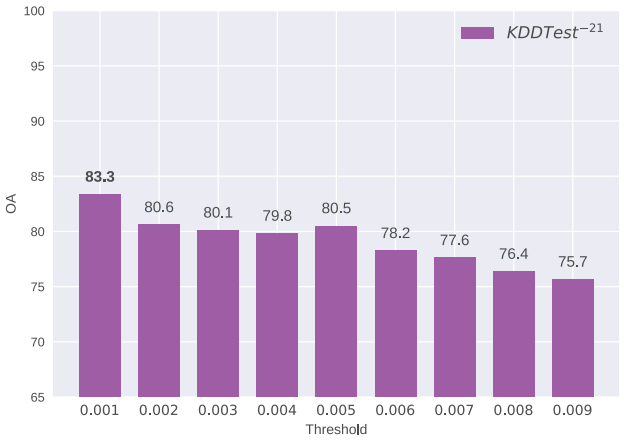
TABLE IV

ACCURACY ANALYSIS OF $\mathcal{A}_1$, $\mathcal{C}_1$, $\mathcal{C}_1 + \mathcal{A}_1$, $\mathcal{C}_2$ AND AIDA: OVERALL ACCURACY (OA), AVERAGE ACCURACY (AA), PRECISION (P), RECALL (R), F-MEASURE (F) AND FALSE ALARM RATE (FAR). FOR EACH ACCURACY METRIC, THE BEST CONFIGURATION IS IN BOLD.

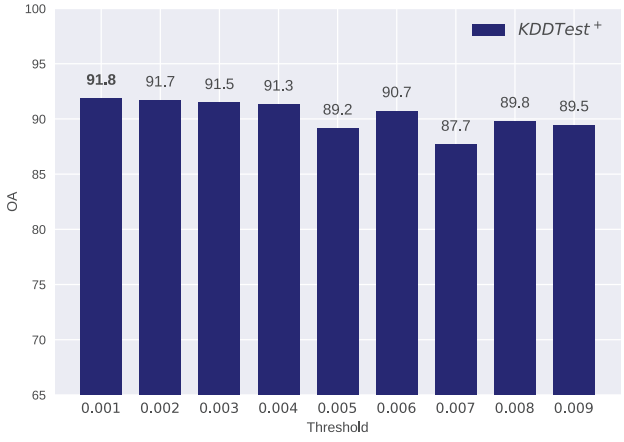
Testing Set	Model	OA	AA	P	R	F	FAR
KDDTest$^+$	$\mathcal{A}_1$	90.40	89.78	89.55	94.06	91.75	14.50
	$\mathcal{C}_1$	81.23	83.13	**96.63**	69.45	80.82	**3.20**
	$\mathcal{C}_1 + \mathcal{A}_1$	91.85	91.05	89.70	**96.82**	93.10	14.70
	$\mathcal{C}_2$	81.53	83.34	96.28	70.26	81.24	3.58
	AIDA	**92.41**	**92.48**	94.52	92.00	**93.24**	7.04
KDDTest^{-21}	$\mathcal{A}_1$	83.35	67.91	88.05	92.15	90.05	56.31
	$\mathcal{C}_1$	58.62	70.12	**95.21**	52.05	67.31	**11.8**
	$\mathcal{C}_1 + \mathcal{A}_1$	86.30	69.54	88.38	**95.87**	91.97	56.78
	$\mathcal{C}_2$	58.70	69.89	94.95	52.33	67.47	12.55
	AIDA	**87.20**	**82.76**	94.35	89.73	**91.98**	24.21



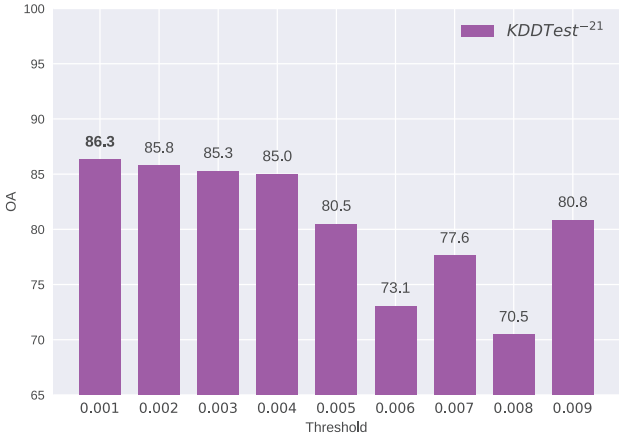
(a) $\mathcal{A}_1$ on KDDTest^{+}



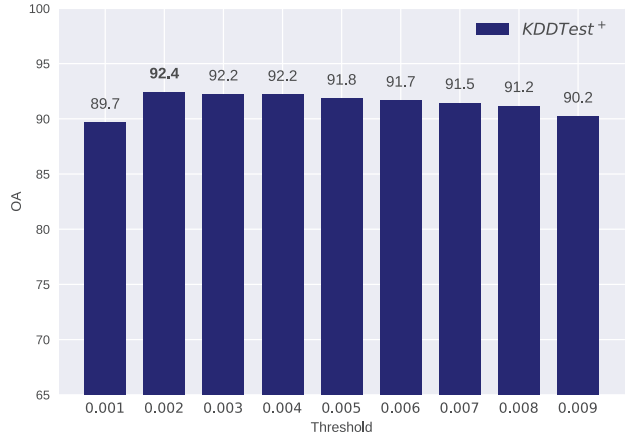
(b) $\mathcal{A}_1$ on KDDTest^{-21}



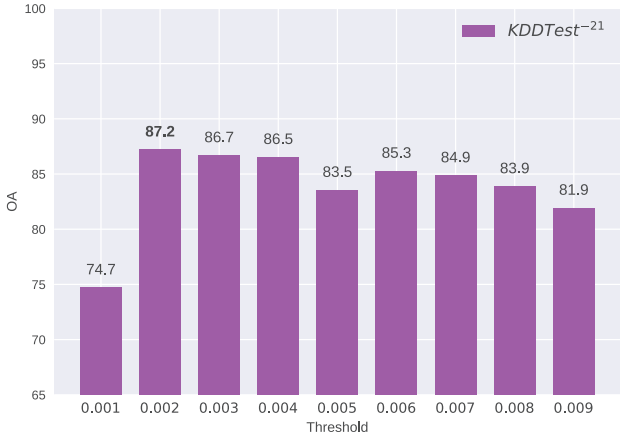
(c) $\mathcal{C}_1 + \mathcal{A}_1$ on KDDTest^{+}



(d) $\mathcal{C}_1 + \mathcal{A}_1$ on KDDTest^{-21}

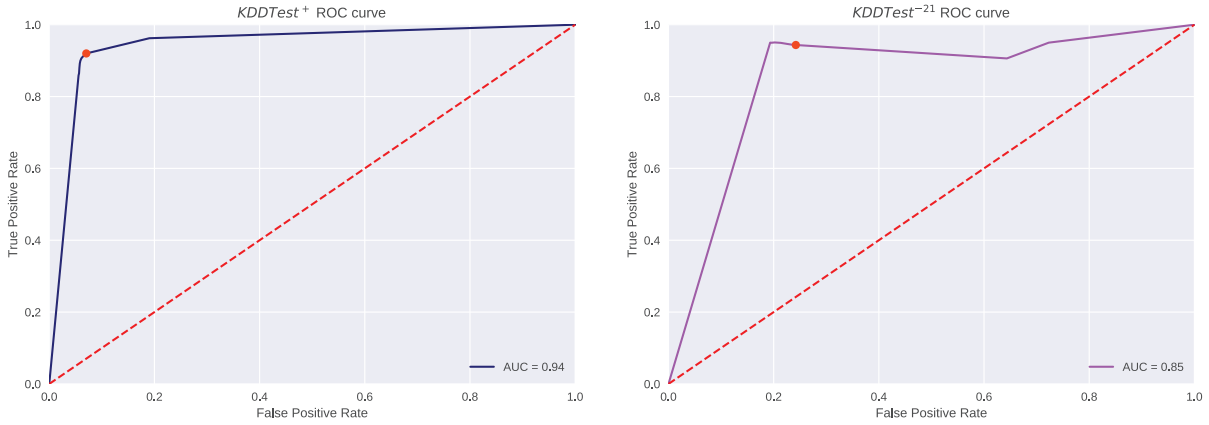


(e) AIDA on KDDTest^{+}



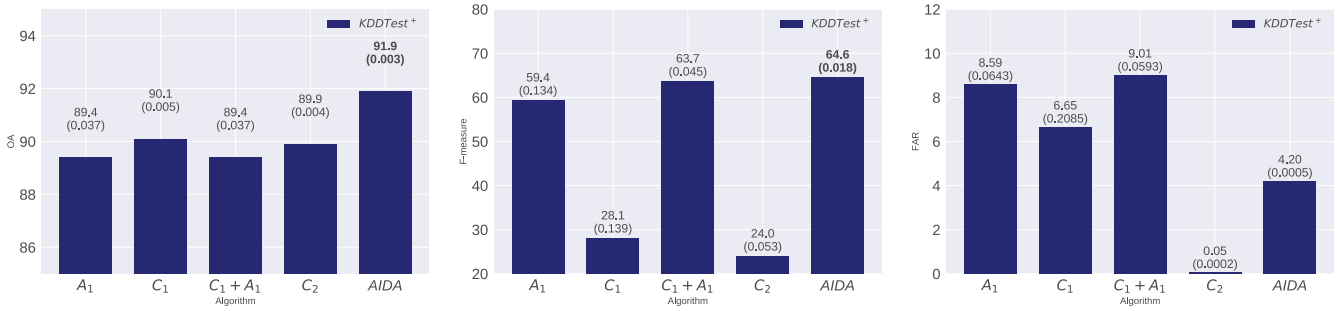
(f) AIDA on KDDTest^{-21}

Fig. 3. Overall accuracy (axis Y) of $\mathcal{A}_1$ (Figure 3(a) and 3(b)), $\mathcal{C}_1 + \mathcal{A}_1$ (Figure 3(c) and 3(d)) and AIDA (Figures 3(e) and 3(f)) by varying the anomaly threshold (axis X). The bars in bold correspond to the highest OA. They represent the configurations that are considered for the comparative study reported in Table IV.



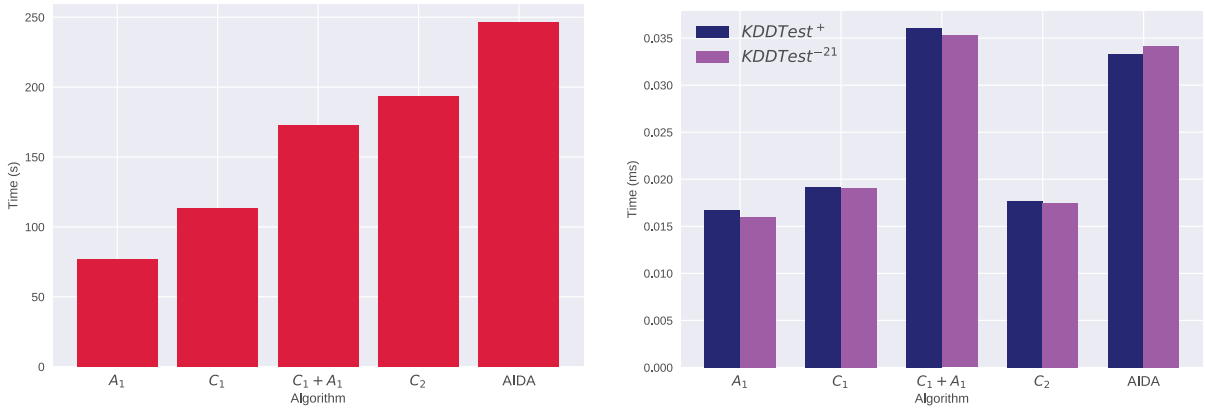
(a) ROC curve on KDDTest+ (b) ROC curve on KDDTest-21

Fig. 4. ROC curves and AUC of AIDA computed by varying the anomaly threshold. Red points marked on curves correspond to configurations of AIDA considered in the comparative study reported in Table IV .



(a) OA (b) F-measure (c) False Alarm Rate

Fig. 5. Overall accuracy (Figure 5(a)), F-Measure (Figure 5(b)) and FAR (Figure 5(c)) (axis Y) of $\mathcal{A}_1$, $\mathcal{C}_1$, $\mathcal{C}_1 + \mathcal{A}_1$, $\mathcal{C}_2$ and AIDA evaluated in an imbalanced scenario (only 10% of attacks are retained for both training and testing phases). For each metric, the length of each bar measures the average metric on ten trials. For each configuration, the mean and standard deviation of the metric computed on ten trials is reported on the top of the bar. For each metric, the best result is in bold.



(a) Training time on KDDTrain+ (b) Testing time on KDDTest+ and KDDTest-21

Fig. 6. Computation time (axis Y) spent by $\mathcal{A}_1$, $\mathcal{C}_1$, $\mathcal{C}_1 + \mathcal{A}_1$, $\mathcal{C}_2$ and AIDA (axis X): total computation time spent in seconds learning the intrusion detection model (Figure 6(a)) and average computation time spent in milliseconds classifying each testing sample (Figure 6(b)).

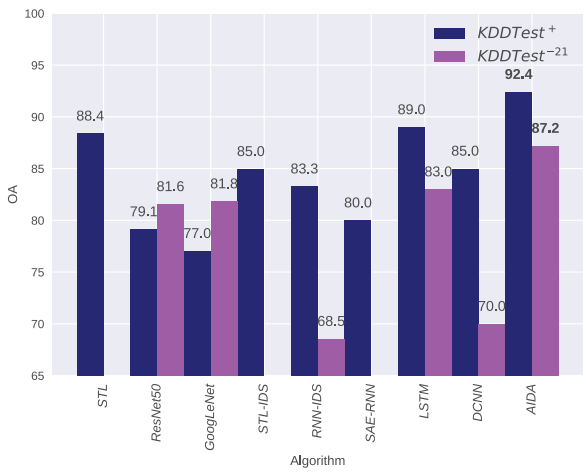


Fig. 7. Overall Accuracy (axis Y) of *STL* [16], *ResNet50* [26], *GoogLeNet* [26], *STL-IDS* [12], *RNN-IDS*[14], *SAE+RNN* [13], *LSTM* [15], *DCNN* [15] and AIDA (axis X). The methods are evaluated with KDDTest$^+$ and KDDTest^{-21}. The OA of *STL*, *ResNet50*, *GoogLeNet*, *STL-IDS*, *SAE+RNN*, *LSTM*, *RNN-IDS*, *DCNN* is collected for the reference papers.

normal samples wrongly alerted as attacks. In any case, the effectiveness of these configurations is drastically reduced by their low recall that highlights how they also suffer of the highest number of attacks, which are wrongly mis-classified as normal samples. On the other hand, configurations $\mathcal{A}_1$ and $\mathcal{C}_1 + \mathcal{A}_1$, which outperform AIDA in terms of R, achieve the lowest performance in terms of P and FAR. Therefore, reversing the analysis performed above, $\mathcal{A}_1$ and $\mathcal{C}_1 + \mathcal{A}_1$ are able to correctly detect the highest number of attacks, but this at the cost of the lowest precision and the highest false alarm rate. The conclusion of this analysis can be drawn by re-focusing the attention on metric F that, computed as an harmonic mean of P and R, describes the overall performance in terms of precision and recall jointly. In fact, the highest F score is always achieved by AIDA. This result is an additional confirmation of the overall superiority of AIDA compared to its baseline configurations.

To validate the results of the accuracy analysis also in an imbalanced setting, we perform an additional experiment by drastically diminishing the number of attacks in both training and testing sets. For this experiment, we consider KDDTest$^+$ that is a super-set of KDDTest^{-21}. We randomly select 10% of attacks from both training and test sets, while we retain all non-attack samples in both sets. This procedure allows us to retain 67343 non-attacks and select 5863 attacks (instead of original 58630 attacks) for training, retain 9711 non-attacks and select 1283 attacks (instead of original 12833 attacks) for testing. We repeat this procedure on 10 trials, in order to build 10 different pairs of imbalanced training and testing sets. We run AIDA and its competitors on these trials and evaluate the accuracy performance. In particular, Figures 5(a), 5(b) and 5(c) report the mean and standard deviation of OA, F-measure and FAR computed on ten trials. Interestingly, results confirm conclusions drawn considering the entire datasets. Although $\mathcal{C}_2$ outperforms AIDA in terms of FAR, this happens since $\mathcal{C}_2$ assigns the most part of samples (included a high number of attacks) to the majority non-attack class. Differently, AIDA outperforms all competitors in terms of both OA and F. As these general metrics, and in particular F, combine the ability of correctly detecting high number attacks without high false alarm rate, we can conclude that also the analysis performed in the imbalanced scenario shows that AIDA reports the best trade-off between correctly detecting attack and non attacks.

To complete this analysis, we analyse the efficiency of the compared configurations. The computational time (in seconds) spent to train the model is plot in Figure 6(a). The average time (in milliseconds) spent assigning each testing sample to a category ('attack', 'normal') is plot in Figure 6(b). As expected computational time spent by AIDA performing both training and testing operations is greater than that spent by the baseline competitors. However, the training time is less than 5 minutes on the entire training set, while the testing time is less than .035 milliseconds per sample. On the other hand, the higher computation burden is counter-balanced by the higher accuracy.

3) Competitor analysis: We compare the overall accuracy of AIDA to that of several competitors (see description in Section II). In particular, we consider the following IDS competitors based respectively on:

- Convolutional Neural Networks (CNNs): ResNet50, GoogLeNet [26] and DCNN [15];
- Long short term memory: LSTM [15];
- Recurrent Neural Networks (RNNs): RNN-IDS [14] and SAE+RNN [13] ;
- auto-encoder used in combination with SVMs: STL-IDS [12];
- auto-encoder used in combination with RNNs: SAE+RNN [13] ;
- auto-encoder used in combination with softmax regression: STL [16];

The accuracy of these competitors is collected from the reference studies where they are all evaluated on KDDTest$^+$ and KDDTest^{-21}. We note that the accuracy of STL, STL-IDS and SAE+RNN is not available for KDDTest^{-21}.

Accuracy results of all compared methods are plot in Figure 7. They highlight that AIDA outperforms all competitors in this study. In particular, AIDA achieves an accuracy equal to 92.4% in KDDTest$^+$ and 87.2% in KDDTest^{-21}, while the runner-up competitor— LSTM—achieves accuracy equal to 89.0% in KDDTest$^+$ and 83.0% in KDDTest^{-21}. Interestingly AIDA outperforms STL-IDS, SAE+RNN and STL, which also use auto-encoding as a part of their intrusion detection model. However, these competitors differ from AIDA in the auto-encoding information used. In fact, STL-IDS, SAE+RNN and STL use auto-encoding in the traditional fashion, i.e. as a means to perform dimensionality reduction in a deep learning architecture, while AIDA uses residual errors to augment the predictive power of classification, as well as to gain in accuracy with anomaly detection.

V. Conclusions

Many machine learning methods proposed in the literature show good performances when applied to detect attacks that follow the same distribution of the training data, while they often poorly perform when work on unforeseen attacks. In this paper, we try to address this issue by formulating a deep learning method that resorts to a feature augmentation step, in order to elicit the data distribution of unforeseen attacks. In this way, we allow the learning process to profitable account for the mis-classifications that unforeseen attacks can introduce. The new feature represents the residual error produced by an auto-encoder learnt from normal samples and used to map attacks also. We feed normal and attack samples, described in the augmented feature representation, to two deep learners arranged in a cascade. They improve the ability of detecting unseen attacks by augmenting the number of attacks correctly detected. The empirical evaluation is performed by considering the benchmark NSL-KDD dataset. Results show with evidence the advantage of accounting the residual errors of auto-encoders in cascading learners, as well as the gain in accuracy, in comparison with recent deep learning methods (the smallest difference is 92.4% vs 89% in KDDTest^{+} and 87.2% vs 83.0% in KDDTest^{-21}). As future works, we plan to further investigate different strategies for residual-error based feature augmentation instead of considering the simplest approach of taking the mean squared as proposed here. Furthermore, we can investigate how both the auto-encoders and the classifier could be learned in an end-to-end fashion.

References

[1] R. R. Reddy, Y. Ramadevi, and K. V. N. Sunitha, "Effective discriminant function for intrusion detection using SVM," in *ICACCI*, 2016, pp. 1148–1153.

[2] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002.

[3] N. Sharma and S. Mukherjee, "Layered approach for intrusion detection using naïve bayes classifier," in *ICACCI*, 2012, pp. 639–644.

[4] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[6] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Comp. Int. Mag.*, vol. 13, no. 3, pp. 55–75, 2018.

[7] T. M. Basile, N. D. Mauro, F. Esposito, S. Ferilli, and A. Vergari, "Ensembles of density estimators for positive-unlabeled learning," *Intelligent Information Systems*, 2019.

[8] D. Y. Oh and I. D. Yun, "Residual error based anomaly detection using auto-encoder in SMD machine sound," *Sensors*, vol. 18, no. 5, p. 1308, 2018.

[9] R. A. R. Ashfaq, X. Wang, J. Z. Huang, H. Abbas, and Y. He, "Fuzziness based semi-supervised learning approach for intrusion detection system," *Inf. Sci.*, vol. 378, pp. 484–497, 2017.

[10] E. de la Hoz Correa, E. de la Hoz Franco, A. Ortiz, J. Ortega, and B. Prieto, "PCA filtering and probabilistic SOM for network intrusion detection," *Neurocomputing*, vol. 164, pp. 71–81, 2015.

[11] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] M. Al-Qatf, L. Yu, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with SVM for network intrusion detection," *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[13] N. N. Kherlenchimeg Zolzaya, "Network intrusion classifier using autoencoder with recurrent neural network," in *ICESS*, 2018.

[14] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[15] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48 231–48 246, 2018.

[16] A. Y. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," *ICST Trans. Security Safety*, vol. 3, no. 9, 2016.

[17] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *ICONIP*, 2017, pp. 858–866.

[18] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, 2018, pp. 1595–1598.

[19] C. Loglisci, A. Appice, and D. Malerba, "Collective regression for handling autocorrelation of network data in a transductive setting," *J. Intell. Inf. Syst.*, vol. 46, no. 3, pp. 447–472, 2016.

[20] B. Luo and J. Xia, "A novel intrusion detection system based on feature generation with visualization strategy," *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4139–4147, 2014.

[21] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "Feature selection and classification in multiple class datasets: An application to kdd cup 99 dataset," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5947 – 5957, 2011.

[22] R. Zuech and T. Khoshgoftaar, "A survey on feature selection for intrusion detection," in *ISSAT International Conference on Reliability and Quality in Design*, 01 2015, pp. 150–155.

[23] N. Shahadat, I. Hossain, A. Rohman, and N. Matin, "Experimental analysis of data mining application for intrusion detection with feature reduction," in *2017 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, 2017, pp. 209–216.

[24] M. el Boujnouni and M. Jedra, "New intrusion detection system based on support vector domain description with information gain metric," *I. J. Network Security*, vol. 20, pp. 25–34, 2018.

[25] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2986–2998, 2016.

[26] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds., 2017, pp. 858–866.

[27] T. Kenaza, K. Bennaceur, and A. Labed, "An efficient hybrid svdd/clustering approach for anomaly-based intrusion detection," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18, 2018, pp. 435–443.

[28] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with svm for network intrusion detection," *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[29] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, pp. 78 – 96, 2018.

[30] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *ICKDM*. ACM, 2017, pp. 665–674.

[31] P. Madani and N. Vlajic, "Robustness of deep autoencoder in intrusion detection under adversarial contamination," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS '18, 2018, pp. 1–8.

[32] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *CISDA*, 2009, pp. 1–6.

[33] F. Hachmi, K. Boujenfa, and M. Limam, "Enhancing the accuracy of intrusion detection systems by reducing the rates of false positives and false negatives through multi-objective optimization," *J. Network Syst. Manage.*, vol. 27, no. 1, pp. 93–120, 2019.

[34] B. Yan and G. Han, "Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system," *IEEE Access*, vol. 6, pp. 41 238–41 248, 2018.