AFEKA
The Academic College of Engineering in Tel Aviv

Deep Learning Course, 65021

Dr. Yehudit Aperstien

Network Intrusion Detection Using Deep Belief Network as an Anomaly Detector

Oded Milman, Liran Shani, Maoz Koren

Abstract

In recent years, there has been a significant increase in cybersecurity threats around the globe, targeting a range of organizations, from private companies to critical infrastructure facilities.

To identify and repel those threats, a lot of resources are invested in developing efficient network intrusion detection systems (NIDSs) that are designed to monitor traffic flows within network and alert cybersecurity personnel if a threat is identified.

The most advanced NIDS methods today use machine learning (ML) algorithms, and more particular deep learning methods.

Since NIDS systems need human intervention for analysis and decision making for some of the identified threats, it can be extremely helpful if a network's decision of identifying a threat could come with an interpretable explanation.

In this paper, we will lay out our proposal for a system that uses deep belief networks (DBNs) and SHAP [1], a model explanation framework, for a novel network intrusion detection system.

A. Introduction

In cybersecurity, there are two main approaches in designing a NIDS. The first approach attempts to identify a network intrusion by deciding if the traffic flow has features that are known to be correlated with a known type of attack, this type of method is known as supervised learning.

The second approach uses anomaly detection methods to learn statistical representations of a normal flow and decide if the new flow exhibits a different behavior. Thus, it is considered as an unsupervised learning method. New types of attacks are constantly and rapidly being developed, and although supervised learning methods for NIDS are relatively efficient in identifying known types of attacks, they often fail when presented with new types of attacks. Recent works, adopting anomaly detection methods for NIDS have shown promising results for identifying new types of attacks.

In the domain of cybersecurity, there is a lot of weight for human decision making.

Thus, having a NIDS that can explain why it has decided to tag a certain network flow as malicious can be very helpful (for example, IT managers can focus on the most important features, i.e ports, IP addresses).
SHapley Additive exPlanations, or SHAP, is a game theoretic approach to explain the output of any machine learning model.

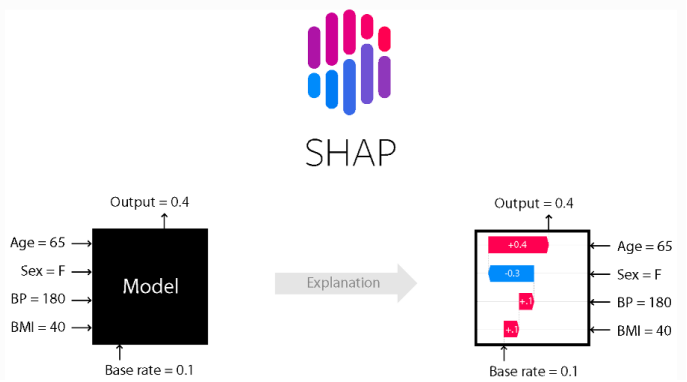


Figure 1: SHAP explanation model

In the example shown in figure 1, a ML model is deployed to find the best credit allocation (Interest rate) using four characteristics (features) of loaners. In the right box of figure 1, there's a bar plot that shows the significance of each of the features in predicting credit allocation, and the direction of its effect on the outcome of the model.

In their paper from 2022, Pieter et al [2] proposed a novel architecture for a NIDS, using SHAP explanation model.

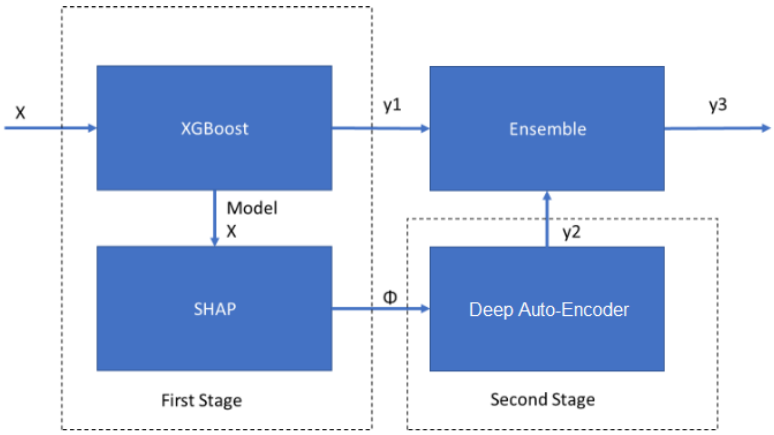


Figure 2: Pieter et al's NIDS Architecture

They have created an ensemble model that uses XGBoost classifier to classify malicious / benign network flow from the NSL-KDD dataset, while performing SHAP analysis on XGBoost model and training a Deep Autoencoder on SHAP values to perform unsupervised classification on new data flow. As the results of XGBoost are difficult to interpret, the authors used SHAP for post-hoc explanation of the tree-based model's results.

Our proposed solution involves Deep Belief Networks (DBN) [3] as a feature detector. A DBN is assembled from a collection of Restricted Boltzmann Machines (RBMs).
RBM is a statistical model that is trained by learning a probability distribution over a set of inputs.

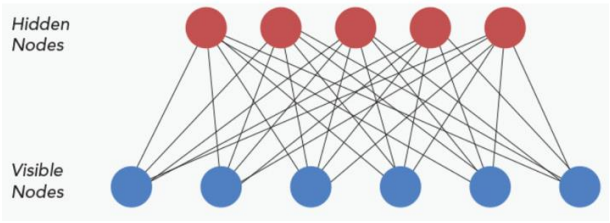


Figure 3: Restricted Boltzmann Machine Architecture

As shown in figure 3, a single RBM is constructed out of a visible layer and a hidden layer.
The network's weights are learned by a process called Constructive Divergence, which reconstructs the inputs and enables gradient descent by calculating the error between input and reconstructed input.

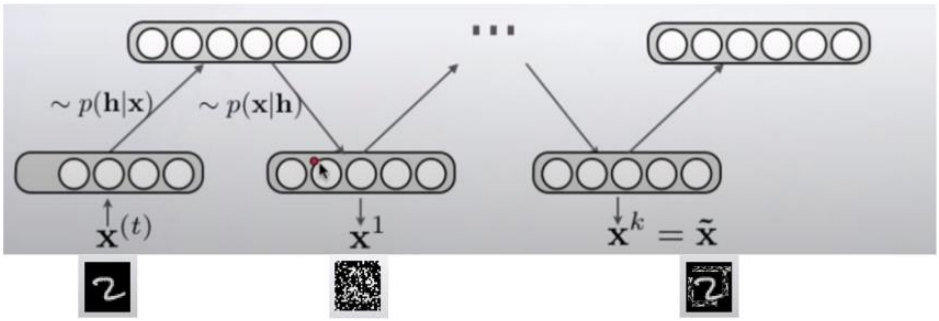


Figure 4: Example of reconstruction process of image input in a RBM

As mentioned before, a DBN is a collection of RBMs that enables detecting more complicated features than a single RBM.

B. Proposed Solution

Pieter et al had used a deep autoencoder to reconstruct SHAP values of new network flow.
An autoencoder [4] is a type of neural network where the output layer has the same dimensionality as the input layer. An autoencoder replicates the data from the input to the output in an unsupervised manner by passing it through the network, reducing the input's size in the middle layers to smaller representations and then reconstructing the input from this reduced representation. The input passes through the encoder, and is coded into a distorted, small representation of the data ("code"), to be reconstructed by the decoder.
Our hypothesis is that by reconstructing its input, a DBN can perform as an autoencoder, reconstructing SHAP values and enabling anomaly detection.
The NIDS consists of 2 main parts, a supervised model and unsupervised model.
The architecture of our proposed model can be seen in figure 5.

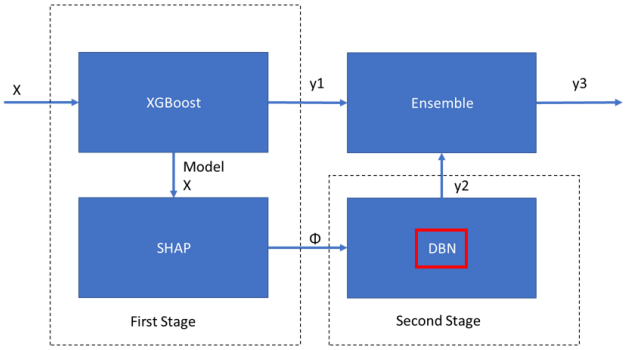


Figure 5: Proposed model architecture. X is the set of features derived from the dataset. The SHAP model input is the XGBoost model and the feature set. The DBN's input is the SHAP values for all features.

The supervised model (XGBoost) will be trained on known attacks and will be responsible for recognizing them. We used the XGBoost model as it is a widely used and well-known classifier. Of course, other classifiers can be used.

To better explain the results of the model, we used an XAI (explainable AI) model, known as SHAP. In this model each sample (set of features) of the training set is assigned a vector of feature importance scores which represent the magnitude and direction by which each feature impacts the outcome of the model for that sample. In figure 6 we can see the outcome of the SHAP model for a particular sample, which our model predicted as malicious. Features which have a positive impact on the model's outcome are shown next to red bars, while features that have a negative impact on the model's outcome are shown next to blue bars. The topmost feature is the one with most influence on the model's outcome (in our case, "src_bytes", with value 18, raised the probability by 0.13%. We can see other features that have negligible contribution). Since every sample has SHAP scores for all its features, we get a new dataset which is as large as the original dataset.

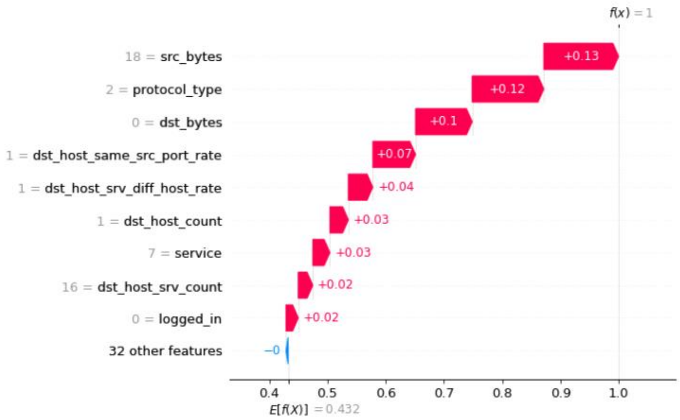


Figure 6: SHAP values of a particular sample.

The second part of the architecture is an unsupervised model used as an anomaly detector for new attacks (known as zero-day attacks). This part is used to differentiate between previously seen behaviors and new ones. This model is trained on the new dataset, obtained from the SHAP model. The outcome of both models is ensembled and the result is the final binary output of the model. (As depicted in figure 5).

C. Dataset

The dataset used in this paper is the NSL-KDD dataset [5]. This dataset is used by many papers as the standard benchmark for Intrusion Detection Systems (IDS). The dataset has 41 features and has 5 classes of traffic: 4 kinds of attacks (each with several subclasses) and benign traffic. The dataset is divided into train and test data. As mentioned in [6] the main difference between these subsets is the fact that test data has subclasses that are not included in the train data. The reason for this distinction is that the attacks which exist only in the test data represent new attacks that the IDS needs to be able to cope with. In our work we used the whole train dataset for training the XGBoost supervised model and divided the original NSL-KDD test data into validation and test data, by a 1:5 ratio, as shown in figure 5. In this work we only recognize 2 kinds of classes: malicious traffic and benign traffic. Actual data for training is from the KDDTrain+.txt file, and for validation and testing is from the KDDTest+.txt.

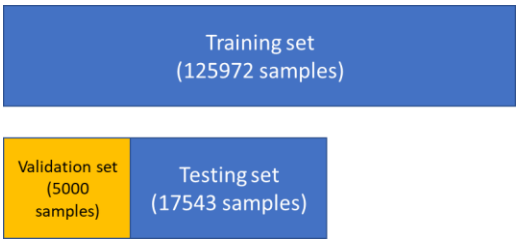


Figure 5: Division of dataset for training, validation, and test data.

D. Model Implementation and Methodology

Data Preprocessing

As mentioned in section C the dataset for the first stage was taken from NSL-KDD. To prepare the data for the XGBoost model we transformed all the categorical features into numerical features with LabelEncoder, from python's Scikit-Learn software. The dataset for the second stage was the SHAP values from the SHAP XAI model. The DBN's input should be in the range of [0,1], so we scaled the SHAP values using MinMaxScaler.

First stage

The supervised model is implemented with XGBoost using default hyperparameters. The output from the model is denoted as y_1. The metric for evaluation of XGBoost was standard cross-entropy.
The SHAP model used the TreeExplainer kernel, which is optimized for tree algorithms like XGBoost.

Second stage

The unsupervised model was a Deep Belief Network (DBN) implemented by [7], with 2 layers each with 41 hidden units. We used the reconstruction property of the DBN to observe anomalies from the regular data. To calculate the reconstruction, we implemented new methods in the unsupervisedDBN

class[1]. The input to the DBN is the SHAP values from the SHAP model, ϕ and the reconstructed features output from the DBN, $\phi`$ are used to calculate the absolute reconstruction error (ARE),

$$ARE = \frac{\sum|\phi_i - \phi`_i|}{N}$$

When the ARE is greater than a certain threshold, the sample is considered as malicious. The output from this model is denoted as y_2. To optimize this model, we used the validation dataset, with a greedy cross validation procedure for hyperparameter tuning. The metric for choosing optimized values was accuracy. Cross validation results are given in the set of tables below.

To get the final model result the two outputs, y_1, y_2 were combined to create the outcome y_3, called the ensembled model. The combining method was a simple OR gate. For all outputs we calculated accuracy, precision, and recall.

Training Epochs	Number of layers	ARE	Ensemble accuracy
25	2	0.776	0.75
50	**2**	**0.866**	**0.89**
75	2	0.86	0.89
100	2	0.86	0.88

Table 1: Cross validation over number of training epochs of the DBN model. The optimized result is 50 epochs.

Training Epochs	Number of layers	ARE	Ensemble accuracy
50	1	0.87	0.87
50	**2**	**0.86**	**0.89**
50	3	0.864	0.89
50	4	0.861	0.89

Table 2: Cross validation over the number of layers in the DBN model. Because of the minor difference in ARE score between 2 and 3 layers, and for simplicity of the model, we chose 2 layers.

[1] The complete code is available from: https://github.com/liranDev/dbn_nslkdd

Training Epochs	Learning rate	ARE	Ensemble accuracy
50	0.01	0.851	0.85
50	**0.06**	**0.86**	**0.89**
50	0.08	0.86	0.89
50	0.1	0.869	0.88

Table 3: Cross validation over learning rate in the DBN model. The optimized result is 0.06.

E. Results and Discussion

In this chapter we will examine the conclusions as they are reflected from the results.

Our idea to use DBN reconstruction capability as an encoder-decoder replacement has demonstrated overall good performance, as shown in table 4.

Method	Accuracy (%)	Recall (%)	Precision (%)
XGBoost	80	82	83
DBN	86	81	94
Ensemble	91	91	92

Table 4: Results of final models, over test set.

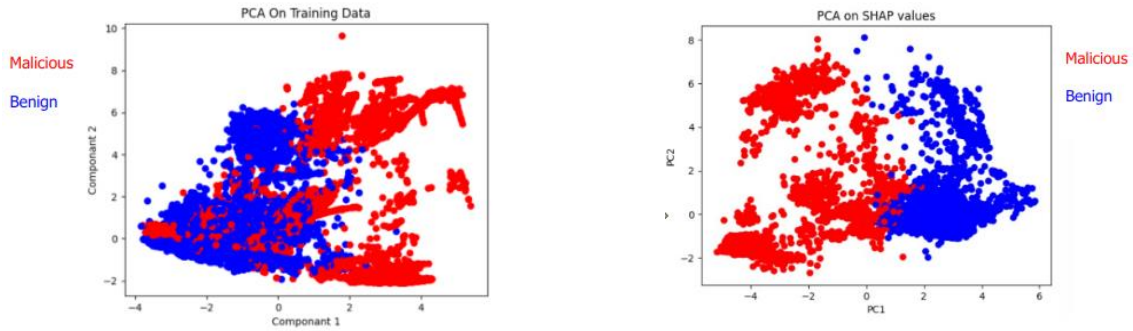
We managed to achieve test accuracy of 91% with the ensemble model. This model combines the results from the XGboost and the DBN. Ensemble models tend to have better accuracy and this work showed that indeed the combined model achieved better results. The following table summarizes our results against other benchmarks.

Method	Accuracy (%)	Recall (%)	Precision (%)	XAI
Pieter et al	93.28	97.81	91.05	✓
Proposed Solution	91.00	91.00	90.5	✓
Yang et al	89.36	84.86	95.98	×
Javaid et al	88.39	95.95	85.44	×
Wang et al	80.6	80.6	82.8	✓
Marino et al	95.5	-	-	✓

Table 5: Performance comparison with other implementations of NIDS

To explain the results, we applied PCA to both training raw data and the SHAP values. Figure 6a shows clear overlap between benign and malicious data. After using the SHAP values, in figure 6b we can see **good separation** between malicious and benign samples.

A possible explanation can be that SHAP values formulated a complex model to a simplified linear model, i.e SHAP values disentangle some of the non-linearities that exist in the model.



(a) *(b)*

Figure 6: Plots of major components after PCA, for the original data (on left) and for SHAP values (on right).

As shown in the figure 7, The model with SHAP values managed to better explain the percentage of cumulative variance, 40% versus 32% in the original data.

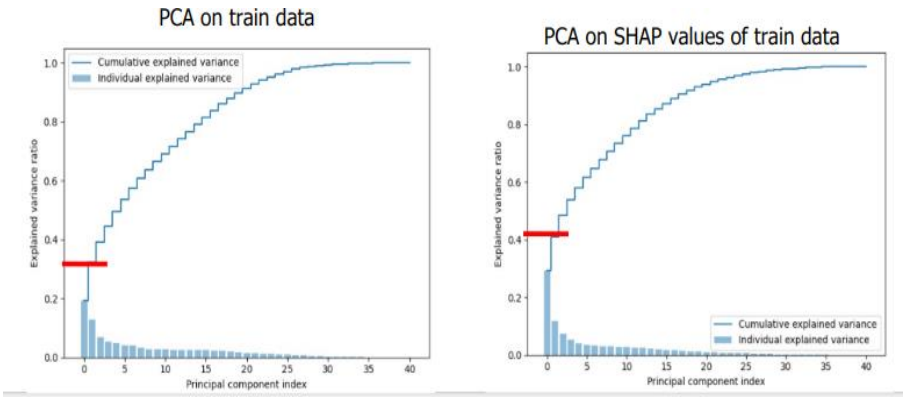


Figure 7: Accumulated percentage of variance on raw train data

F. Conclusion and Future Work

In this work we introduced a novel NIDS architecture, based on a DBN as an anomaly detector. Together with a supervised model and an XAI model based on SHAP we show good accuracy results as compared to other models. For future work, we can add another supervised model like SVM, and combine them to a majority vote ensemble. Another aspect for further study can be a better way to ensemble the two models, for example a method that takes into account the error of each model.

References

[1] https://shap.readthedocs.io/en/latest/

[2] Pieter Barnard *et al*., Robust Network Intrusion Detection through Explainable Artificial Intelligence (XAI), *JOURNAL OF LATEX CLASS FILES*, VOL. 14, NO. 8, FEBRUARY 2022 2

[3] Geoffrey E. Hinton and Simon Osindero, A fast learning algorithm for deep belief nets, Neural Computation 2006

[4] https://www.mygreatlearning.com/blog/autoencoder/

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

[6] Arnaldo Gouveia et al., Network Intrusion Detection with XGBoost, Recent Advances in Security, Privacy, and Trust for Internet of Things (IoT) and Cyber-Physical Systems (CPS) (pp.137-166), November 2020

[7] https://github.com/albertbup/deep-belief-network