Hindawi
Security and Communication Networks
Volume 2020, Article ID 3512737, 15 pages
https://doi.org/10.1155/2020/3512737

WILEY | Hindawi

Research Article

Neural Network-Based Voting System with High Capacity and Low Computation for Intrusion Detection in SIEM/IDS Systems

Nabil Moukafih, Ghizlane Orhanou, and Said El Hajji

Laboratory of Mathematics, Computing and Applications-Information Security, Faculty of Sciences, Mohammed V University in Rabat, BP1014 RP, Rabat, Morocco

Correspondence should be addressed to Nabil Moukafih; moukafih.nab@gmail.com

Received 24 December 2019; Revised 23 June 2020; Accepted 29 June 2020; Published 16 July 2020

Academic Editor: Mamoun Alazab

Integrating intelligence into intrusion detection tools has received much attention in the last years. The goal is to improve the detection capability within SIEM and IDS systems in order to cope with the increasing number of attacks using sophisticated and complex methods to infiltrate systems. Current SIEM and IDS systems have many processes involved, which work together to collect, analyze, detect, and send notification of failures in real time. Event normalization, for example, requires significant processing power to handle network events. So, adding heavy deep learning models will invoke additional resources for the SIEM or IDS tool. This paper presents a majority system based on reliability approach that combines simple feedforward neural networks, as weak learners, and produces high detection capability with low computation resources. The experimental results show that the model is very suitable for modeling a classification model with high accuracy and that its performance is superior to that of complex resource-intensive deep learning models.

1. Introduction

It is no secret that Internet access has become an indispensable part of life. In fact, most businesses and government institutions operate online. However, in addition to the important benefits and services offered daily by computer networks, they also raise network security issues as many unscrupulous cyberattackers are also active on the Web, waiting to hit vulnerable systems. The integration of cybersecurity tools and threat detection has become increasingly important to prevent downtime. Security devices such as Security Information and Event Management, or SIEM [1], and Intrusion Detection Systems, or IDS [2], have become a core part of monitoring and defending networks and hosts against intrusions.

Unfortunately, this has become quite difficult as attacks are evolving rapidly in terms of complexity and sophistication, especially attacks with signatures that are not recorded in public databases (0-day attacks) and those that target specific systems and vulnerabilities. Such attacks can be used to go unnoticed by most organizations' defense mechanisms and infiltrate the target network. Indeed, in the 2018 data breach investigations, we see that 68% of breaches last year took months or longer to be discovered [3], and these breaches happen within few minutes or even seconds.

Under these constraints, researchers and security experts try to provide intelligence, adaptation, and pattern recognition for SIEM and IDS systems. In particular, they use machine learning models to improve the efficiency and accuracy of these systems by providing historical data to these models. This gives the algorithm or the model more "experience," which can, in turn, be used to make better decisions or predictions. For this reason, machine learning techniques represent the best choice over traditional rule-based algorithms and even human operators [4], and they are widely used in multiple fields and industries [5]. The problem is that machine learning models have some particularly demanding needs in terms of computational resources to train and calibrate. On the other hand, SIEM/IDS have other resource-intensive processes such as collecting

and normalizing events that are running with other advanced analytics modules. This makes the use of machine learning in such systems more challenging. This research presents a new approach to create a machine learning model, by combining small submodels, with low computation and high detection rate.

1.1. Motivation: Why Combining Models Is Interesting? There is an interesting area in the field of machine learning called "*ensembling*," where researchers try to increase model capacity without a proportional increase in computation. This is due to the fact that some models, especially neural networks, need significant computational resources to train when facing large datasets or complex classification problems. In fact, some researchers even started investigating other approaches to trade accuracy of the classifier for speed and memory usage [6]. Others focused instead on externalizing some of the heavy important processes in SIEM systems in order to leave resources available for correlation and advanced data analysis techniques [7].

Model ensembling is a very powerful technique to increase accuracy of models on a variety of machine learning tasks. It consists of several base models combined together in order to produce one optimal model that will best predict the desired outcome. Some ensembling methods require the base models to be pretrained to only combine their predictions on test set; other methods require multiple rounds of training on different chunks of the train set. This paper examines the first category, especially Majority Vote Systems, where the prediction of the ensemble represents the majority prediction of the base models. The benefits of majority vote can be clearly seen below.

Suppose a SIEM/IDS system uses three neural-based binary classifiers with 70% of classification accuracy. For an observed event, the neural classifier can predict four scenarios:

(i) Scenario 1: all three classifiers are correct:

$$0.7 \times 0.7 \times 0.7 = 0.3429. \quad (1)$$

This value can be interpreted as the probability that the three classifications predict the same class.

(ii) Scenario 2: two classifiers are correct and one is wrong:

$$0.7 \times 0.7 \times 0.3 \times 0.7 \times 0.3 \times 0.7 + 0.3 \times 0.7 \times 0.7 = 0.4409. \quad (2)$$

This value can be interpreted as the probability that the two classifications predict the same class. This means that the majority vote corrects an error most of the times (≈44%).

(iii) Scenario 3: the *probability* that two classifiers are wrong and one is correct:

$$0.3 \times 0.3 \times 0.7 + 0.3 \times 0.7 \times 0.3 + 0.7 \times 0.3 \times 0.3 = 0.189. \quad (3)$$

(iv) Scenario 4: the *probability* that three classifiers are wrong:

$$0.3 \times 0.3 \times 0.3 = 0.027. \quad (4)$$

So, by adding the outputs from the first and second scenarios, we see that the output of the majority vote ensemble can be correct with an average of ≈78%.

The combination of models or learners can sometimes improve the overall accuracy; that is, the prediction accuracy of the ensemble model is greater than that of all the other base models. But that is true only when two things are met:

(a) The base models are *weak learners*. This means that they are classifiers that are only slightly correlated with the target label (they can often label examples better than random guessing). They also have the distinction of being computationally simple, unlike strong classifiers that are arbitrarily well correlated with the actual classification.

(b) The weak learners should be *uncorrelated* in their predictions. This is really important because if all learners predict the same class (even with different probabilities), then the final prediction of the ensemble model will always predict the same class. So, a lower correlation between ensemble model members is needed in order to increase the error-correcting capability.

1.2. Contributions. This paper presents a machine learning model that can be integrated with SIEM/IDS systems for intrusion detection. Our model treats security events from different angles and using different weak models and, according to the majority of predictions, we classify the nature of the event. This article presents the following main findings:

(i) A simple and effective intrusion detection model is developed using a majority system and based on reliability. This model can be easily integrated in current SIEM and IDS systems as it was developed following best practices in machine learning.

(ii) An investigation of some of the techniques used to improve the overall accuracy of the whole model using weak learners is presented.

(iii) A novel approach is used to create weak learners using only artificial neural networks.

(iv) The proposed model proved to be of high capacity without a proportional increase in the resources used. The model has also shown promise when compared to other current complex models.

The remainder of this paper is structured as follows: Section 2 examines the related work to intrusion detection using neural networks in SIEM and IDS systems. In Section 3, we present the components of our majority model based on reliability. The model combines the outputs of weak neural network using a novel approach. In Section 4, we

implement our approach and compare it with the most relevant related work. Finally, we conclude the paper in Section 5.

2. Related Work

There is a decent effort to apply neural networks in SIEM/IDS systems. These applications can be classified into two groups: intrusion detection and user behavior analysis. The goal is to create a neural network-based system that automatically learns complex normal behavior (or a normal security event) and at the same time knows what suspicious activity (or an attack signature) looks like.

Generally speaking, before developing a machine learning model, the data should follow certain preprocessing steps before feeding them to the model. The authors in [8] applied different preprocessing and discretization techniques to the NSL-KDD dataset and found that these steps have a big effect on the execution of the classification algorithms. Also, we argue that each preprocessing step has a specific purpose and interpretation and cannot be used out of context if the final goal is to have a machine learning model that operates well on production and not just on the used dataset. In this context, we will divide past literature on the applications of neural networks in intrusion detection or in user behavior analysis into two categories based on the preprocessing steps used, specifically, depending on how the authors transform the nominal or categorical features of the dataset. We distinguish works that use one-hot encoding as a preprocessing step and others that implement other techniques such as integer encoding to transform nominal features because it affects the relationship between variables in the data. This will be discussed in later sections.

(1) One-Hot Encoding (Used Preprocessing Step)

The authors in [9] investigated the impact of the cost and the training function (optimizer) on the accuracy of neural network classifiers within SIEM/IDS systems. This work evaluates 37 feedforward neural networks, where each model contains different cost and training functions. The best model, combined with Support Vector Machines (SVM), achieves an accuracy of 81.8% on the famous NSL-KDD dataset [20].

In [10], the authors used a deep learning approach by implementing a two-stage classifier: a sparse autoencoder and softmax regression (or SMR) based NIDS. The sparse autoencoder is used for feature learning that outputs relevant features to the softmax regression for classification using the NSL-KDD dataset. For 2-class classification, the authors achieved a total classification accuracy of 78.06% (first stage, 88.39% accuracy rate, and SMR achieved 78.06%).

Another work that used a similar approach was done by the authors in [11]. This work used a variant of sparse autoencoders for unsupervised feature learning and a logistic classifier is then utilized for classification on NSL-KDD dataset. The authors achieved an accuracy of 87.2%.

In another effort to propose a suitable intrusion detection model using deep learning approach, the authors proposed in [12] RNN-IDS an intrusion detection using recurrent neural networks that can improve the accuracy of current intrusion detection systems. The model achieved 81.29% accuracy using 80 hidden nodes in binary classification.

An effective network intrusion detection system, using an architecture called Channel Boosted and Residual learning based deep Convolutional Neural Network (CBR-CNN), is developed using one-class classification in [13]. The model uses a *Reconstructed Feature Space* that reconstructs a feature space using only normal traffic. The anomalies in the train set are generated far away in the reconstructed feature. Next, a *Channel Boosting* is used to improve classifier performance by increasing diversity in the classifier's input feature space. Finally, a CNN classifier is used to classify events. The model achieved an accuracy of 89.41%. The authors showed that these results are better than other classification algorithms such as J48, Naive Bayes, random forest, and multilayer perceptron. This work was implemented using the NSL-KDD dataset.

The authors in [14] proposed a deep neural network to develop an effective IDS that classifies and predicts attacks in an Internet of Medical Things (IoMT) environment. In this work, the authors also argue that it is recommended to use one-hot encoding on the categorical characteristics in order to avoid creating numerical values that can be misunderstood by the algorithm due to some ordering issues. After preprocessing, optimizing, and tuning of the deep neural network by hyperparameter selection methods, the model was tested on a Kaggle intrusion detection dataset, where it showed an increase of 15% of accuracy and a reduction of 32% of the time required for training and classification.

(2) Other Preprocessing Steps

The authors in [15] used neural networks as an event classifier within SIEM systems. The classifier, called CONTEXTUAL, is used with another subsystem called GENIAL based on genetic programming to improve the correlation engine of SIEM systems. The neural network-based subsystem classifies all the events collected by the SIEM system and GENIAL generates new correlation rules according to the neural network classification. In their paper, the authors discussed the performance of their correlation engine and have not provided information about parameters of the neural network.

The authors in [16] proposed a user behavior classifier based on neural networks to detect malicious activities that use valid credentials and standard administrative tools to evade detection. In their

work, they implemented several feedforward and recurrent neural networks with different hidden layers and on different number of epochs in order to identify the suitable model for user behavior analysis. According to the authors, the best feedforward neural network achieved 98% accuracy and the best recurrent neural network achieved 97% accuracy.

The authors in [17] proposed an anomaly network intrusion detector constructed on the KDD dataset. The model was built by studying 48 IDSs (neural networks) and by studying the most important parameters that, according to the authors, are input features, normalization function, number of hidden nodes, and the activation function. At the end of the study, the authors selected the two best models and implemented them as a network intrusion detection system.

On a different dataset, the authors in [18] proposed an anomaly intrusion detector using hybrid principal component analysis- (PCA-) firefly based machine learning model to classify intrusions and attacks. The hybrid PCA and firefly algorithm was used to reduce dimensionality and training time. This will help the classifier, XGBoost in this case, to work better on the reduced dataset that has better features and lower complexity.

Machine learning was also applied in cybercrime classification. For instance, the work in [19] proposed a framework by combining multiple models: Naive Bayes is used for classification, k-means is used for clustering, and the TFIDF or tf-idf vector process is used for feature extraction. The goal was to support analytics regarding the identification, detection, and classification of the integrated cybercrime offenses.

The literature survey is summarized in Table 1.

From this literature review, regardless of the used preprocessing steps, we can make the following remarks:

(a) Compared to other machine learning techniques, neural networks have the important advantage of being flexible and can be adapted to particular use cases of intrusion detection in general. But this leads directly to the question of tuning to determine the optimal hyperparameters of a system based on a neural network.

(b) In order to tune an intrusion detection model, most researchers train many different candidate networks and only select one (the best) and discard the rest. This raises three issues: First, all of the effort and resources dedicated to train the remaining networks are wasted. Second, the selected model that had the best performance on the validation set might not be the one with the best performance on test data or new unseen ones. Third, sometimes even unselected models can outperform the best one when labeling rarely observed events.

In this work, instead of working with a single model, we will use three models that will process security events from different angles and classify them according to the decision made by the majority.

3. Majority System Design Based on Reliability

In this section, with the purpose of explaining the different modules of our proposed system, we will use a chronological methodology to justify the final model shown in Figure 1.

The proposed system has three major components that work together to achieve a reliable event classifier:

(i) Data preparation module: this module will apply different preprocessing techniques to the used dataset before feeding them to the proposed model. Our goal is to propose a model that can be easily integrated into a SIEM/IDS system. In this context, we have followed the best practices in this phase, since we noticed that some preprocessing techniques were applied out of context.

(ii) Weak neural networks module: this uses feedforward neural networks as weak learners in our framework. We will detail later the structure of the neural networks used and how we managed to create weak learners using NNs.

(iii) Ensemble module: this module contains the ensemble model that combines the predictions of the weak learners in a way the majority vote will be produced. This module also considers the reliability of the base learners.

More details on these modules will be given below. For the ensemble module, we will also present other models that were implemented during the experiment because they will play a role in the justification of the choice of a majority function based on a reliability system as an ensemble module.

3.1. Data Preparation Module. In this study, we used the NSL-KDD (Network Security Layer-Knowledge Discovery Database) dataset as an input. It is a refined version of its predecessor KDD'99 dataset. This database was collected from the 1998 DARPA Intrusion Detection Evaluation Program that was prepared and managed by MIT Lincoln Labs to survey and evaluate research in intrusion detection.

3.1.1. Dataset. NSL-KDD is a dataset proposed by Tavallaee after criticizing the inherent problems of KDD'99 in [21]. Because KDD'99 contains many duplicate records, a machine learning model is likely to learn high-frequency attacks that can affect test-process evaluation results and prevent it from learning infrequent records, which are usually more damaging to networks. As a result, NSL-KDD comes with the following improvement:

Table 1: Summary of the literature survey.

Reference	Model	Dataset	Accuracy (%)
El Hajji et al. [9]	NN with best cost and training function	NSL-KDD	81.8
Javaid et al. [10]	Sparse autoencoder with SMR	NSL-KDD	78.06
Gurung et al. [11]	Sparse autoencoder with LR	NSL-KDD	87.2
Yin et al. [12]	RNN based IDS	NSL-KDD	81.29
Chouhan et al. [13]	CBR-CNN based IDS	NSL-KDD	89.41
Maddikunta et al. [14]	DNN with PCA-GWO	Kaggle dataset	99.9
Suarez-Tangil et al. [15]	NN and GP	–	–
Ussath et al. [16]	RNN and NN	—	89
Chiba et al. [17]	NN	KDD	99.62
Bhattacharya et al. [18]	PCA-firefly based XGBoost	Kaggle dataset	99.9
Gadekallu et al. [19]	Naive Bayes	Kaggle and CERT-In repositories	99.9

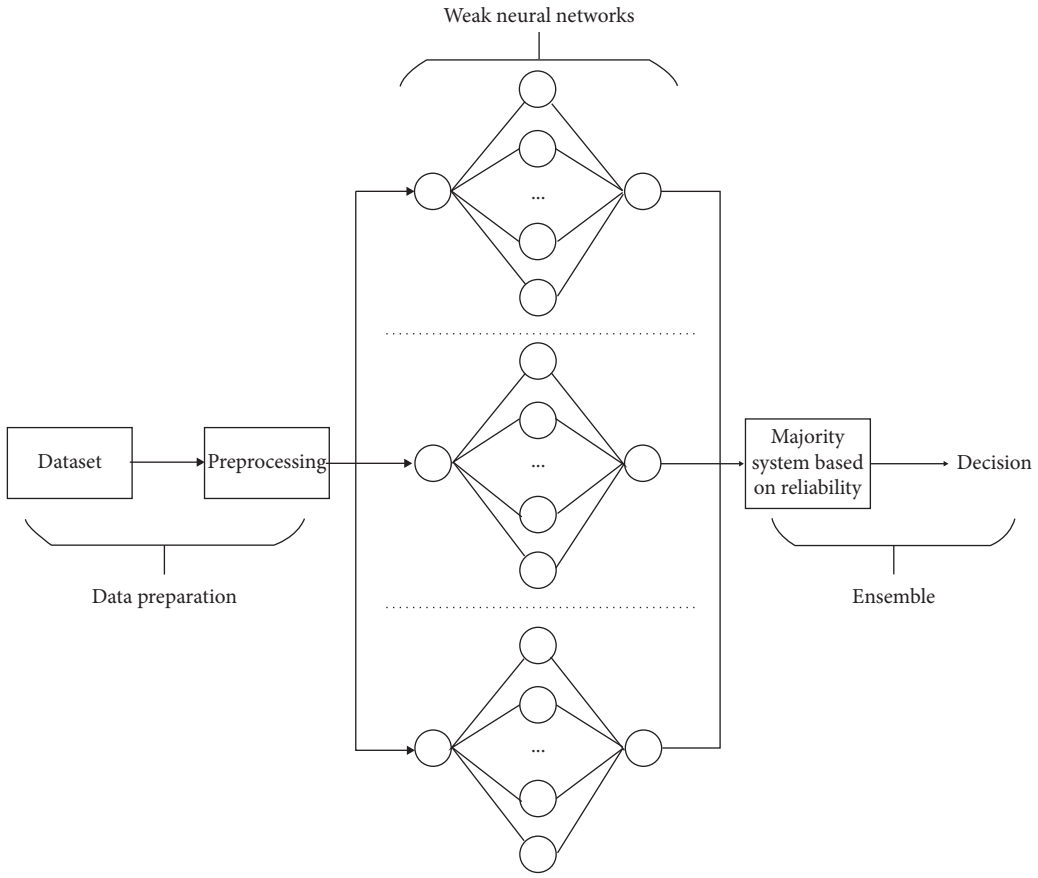

Figure 1: The new voting neural network intrusion detection system.

(i) It does not include redundant and duplicate records in the train and test sets. Thus, the classifiers will not be biased towards more frequent records, and the performances of the learners are not biased by the methods that have better detection rates on the frequent records.

(ii) The number of records in the train and test sets is reasonable, which allows for affordable experiments on the complete set without the need to randomly select a small portion. As a result, the evaluation results of different research studies will be consistent and comparable. More information on the improvements can be found in [21].

NSL-KDD is a collection of downloadable files available to researchers. They are listed in Table 2.

3.1.2. Data Preprocessing. In each record of the NSL-KDD dataset, there are 41 different features of the stream and a label to indicate if a record is an attack or a normal flow. Each feature can have a categorical/nominal value such as the protocol used in the connection *(Protocol_type = TCP, UDP or ICMP)* or a numeric value like the length of time duration of the connection. In this subsection, we will detail the steps we followed to prepare the dataset:

Table 2: List of NSL-KDD dataset files and their description.

File name	Description
KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format
KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
KDDTrain+_20Percent.ARFF	A 20% small subset of the KDDTrain+.arff file
KDDTrain+20Percent.TXT	A 20% subset of the KDDTrain+.txt file
KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format
KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

(a) Input Features

The first important phase in constructing an effective intrusion detection model is feature selection. It is the process of selecting a subset of relevant features that will allow learning models to perform faster and more efficiently in order to improve the classification accuracy.

For the NSL-KDD dataset, a number of researchers have studied the impact of some features on the accuracy of classification. For instance, the authors in [22] showed that the attributes 9, 20, 15, 17, 19, 21, and 40 have minimum or no role in detection of attack. The authors in [23] mentioned that features 7, 8, 11, and 14 are not useful, since there are almost all zero values in the dataset. In this context, these features were removed from the train and test sets in this study to retain only 29 features in total.

(b) Categorical Encoding

Categorical encoding refers to the process of assigning numeric values to nominal (nonnumeric) features in order to facilitate the processing task. Since we are using only neural networks in our model, it is necessary to convert the textual data of the dataset into numerical values. Generally speaking, there are two approaches to encode categorical variables:

(1) One-hot (binary) encoding: it is a binary representation of nominal features where the categorical value is removed and a new binary variable is added for each unique nominal value
(2) Integer encoding: it refers to the process of coding a categorical variable using integers such as 1, 2, and 3

The main difference between the two approaches lies in the existence of an order relationship between the values of the nominal feature in question. Integer encoding only makes sense if the categorical variable has an order; that is, if we are studying a dataset that follows the Syslog format, then the ordinal feature *Severity_Level* will be encoded with numeric values from 0 to 7 according to the predefined levels of the log format (from Emergency to Debug). On the other hand, because one-hot encoding implies an assumption of independence between the records of the dataset, it is used on nominal attributes with no order.

All the nominal features of the NSL-KDD dataset are independent and do not follow any order. For this reason, we used one-hot encoding to transform the nominal features. For example, *Protocol_Type* is encoded into 3 binary variables (tcp: (1, 0, 0); udp: (0, 1, 0); icmp: (0, 0, 1)). By applying this transformation to all the categorical features in the dataset, each connection record in NSL-KDD will be represented by 110 coordinates instead of 29.

As a final remark, there are many researchers who worked on NSL-KDD (or KDD dataset) and used numerical encoding to represent the nominal features of the dataset. Having done so, the model will assume a natural ordering between categories, which may lead to poor performance or unexpected results (predictions halfway between categories) [24]. With this in mind, we will not include the papers with this preprocessing step in our comparison.

(c) Data Normalization

The goal of this step is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. This is interesting only when features have different ranges. Data normalization has the advantage of speeding up some machine learning algorithms.

Generally, the two most effective attribute normalization methods for NSL-KDD preprocessing are mean range [0, 1] (min–max normalization) and statistical normalization (Z-score normalization) as mentioned by the authors in [25]. For our model, we have used Z-score normalization to bring all values into the range [0, 1].

(d) Dimensionality Reduction

It is pretty clear that one-hot encoding linearly increased the size of the dataset. This is because some nominal features have a lot of values: the feature *Service* has 70 types of attributes, and the feature *Flag* has 11 types of attributes.

For this, we have used principal component analysis (PCA) as a final preprocessing step to work on a lower dimension space and also speed up the training time of our neural networks. It is considered as one of the effective techniques used to transform a number of correlated variables into a number of uncorrelated variables called principal components

(PCs) without losing the primary information. The main goal of PCA is to reduce dimensionality of the initial features, while retaining as much as possible the variance (99% in our case) present in these samples.

So, applying PCA to our dataset has reduced the coordinates of each connection record from 110 to 90.

3.2. Pushing the Limits of Feedforward Neural Networks. The goal of our study is to propose a voting system that uses neural networks to classify events based on majority decision. This suggests, from the preliminaries section, that the three neural networks should be weak learners with uncorrelated predictions. In this context, initial studies and tests were performed in order to create weak neural networks that differ by using the following:

(i) Different number of hidden layers
(ii) Different number of hidden units per layer
(iii) Different types of activation functions
(iv) Different learning algorithms or optimizers
(v) Different sizes of input features

As soon as each neural network was trained, their predictions were combined by a simple averaging function. These tests revealed that the accuracy of the averaging function is a simple mean of the accuracy of each classifier without improvement. There are several possible explanations for this outcome:

(1) Neural networks are simply strong learners that try their best to approximate a function or minimize the error to achieve the desired output, regardless of the dimension space or the used parameters/constraints
(2) This is a consequence of the first explanation: even by varying the parameters of each model, the predictions of each trained model are still correlated. This was discussed before; even with the different predictions, all the three models predict the same class (for most observations)

Note that, in these initial experiments, we even varied the input features for each model, using a number of feature selection algorithms for each model. Our initial hypothesis was that prediction of the base learners would be uncorrelated if they were trained on different dimension spaces. Although this is not completely false, it is hard to achieve when the used base models are strong learners such as neural networks.

This explains why neural networks are not used as weak learners in model ensembling in many studies. However, in the documentation of JMP (John's Macintosh Project), which is a suite of computer programs for statistical analysis developed by the JMP business unit of SAS Institute [26], it is mentioned that neural networks can be used as base learners if the base model is a single-layer model with 1 to 2 nodes. Otherwise, the benefit of faster fitting can be lost if a large number of models are specified. For the NSL-KDD, using 1 or 2 nodes did not create a weak learner but rather an unstable one. So, in order to create a weak learner using neural networks, we propose the following approach depicted in Figure 2.

The input features will be divided into three subsets; each neural network will use one subset during training and drop the rest: during training, the first neural network will use the first 30 features (0, 1, . . ., and 29) and drop the rest, the second neural network will use the second 30 features (30, 31, . . ., and 59) and drop the rest, and finally the last one will drop the first 60 features and use the features 60, 61, . . ., and 89. This approach is like the dropout regularization technique but with the following differences:

(i) In the dropout regularization, nodes are dropped randomly at each epoch. Using our approach, the learner will drop the same input nodes in every epoch. We used this modified version of dropout regularization to train the neural networks on different subset of features and then test the trained model on the whole set of features (the original dimension space, using 90 features).
(ii) The goal of dropout regularization is to improve the generalization of the model, where the goal here is to improve the specialization of each model.

It could be argued that if each neural network was trained on different features, then the predictions of the models will differ, which could improve the performance of the ensemble model. However, the initial results revealed that this is not the case; this is because even if the model was trained on different subset of features, the test set will also be mapped through the same transformation (e.g., will have the same features used during training). This detail is important for strong learners such as neural networks.

3.3. Ensemble Module. In our initial tests, we used a simple averaging function so we can propose an approach to create weak learners based on neural networks that differ from each other. After this has been done, we can now dive more into the different techniques to combine the predictions of the pretrained neural networks. In this research, we have studied the following ensemble techniques:

(a) Hard Voting
(b) Mixture of Experts
(c) Majority Function
(d) Weighted Average
(e) Majority System Based on Reliability

Each technique was implemented separately in order to choose the best one for our study. Details of each ensemble techniques are discussed as follows:

(a) Hard Voting

This is the simplest case of majority voting. Classes or labels are predicted via plurality (majority) voting of each classifier. Note that here we consider the classes and not the probability prediction of the classifier.

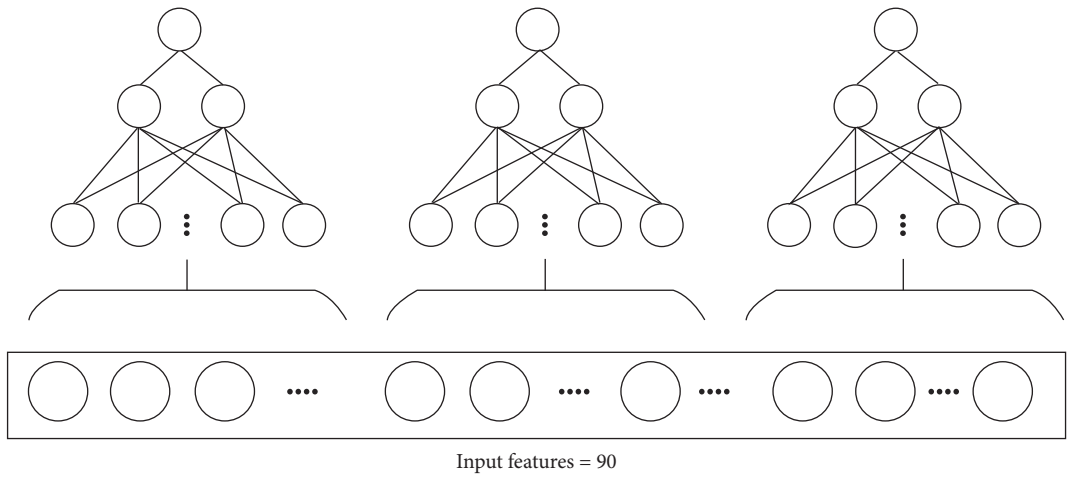

Figure 2: The used approach to create weak neural networks.

(b) Mixture of Experts Layer

In this approach, each model, called expert, is made to focus on predicting the right label for the cases where it is already doing better than other experts, which causes specialization. This concept also aims to create specialized base neural networks, except that here it uses a *manager* function (another neural network) to do so. This *manager* monitors the expert's predictions to improve the overall accuracy and performance of the whole model as shown in Figure 3.

In this context, the authors in [27] proposed the Mixture of Experts (MoE) layer, which consists of a number of experts (namely, four), each a simple feedforward neural network, and a trainable gating network, which selects a sparse combination of the experts to process each input. All parts of the network are trained jointly by back-propagation. The MoE layer was also implemented in this study and has been modified accordingly to use our 3 specialized weak neural networks, instead of using the same feedforward neural networks with the same architecture.

(c) Majority Function

Each neural network will process an event using its defined architecture and parameters and produce a probability p. This probability represents the classification accuracy of a neural network for the processed event.

The objective of this phase is to use a majority voting system to classify security events. Since each neural network uses a different subset of input features, each model will produce a different classification accuracy p_i. If the majority of the neural networks classify an event as an attack, then the final decision should represent the majority vote.

For this, we propose the following steps:

(1) Order the results (p_1, p_2, p_3) of each neural network in ascending order (r_1, r_2, r_3).

(2) Take the function

$$w(r) = \begin{cases} 0, & \text{if } r < 0.4, \\ 5r - 2, & \text{if } 0.4 \le r \le 0.6, \\ 1, & \text{if } r > 0.6. \end{cases} \tag{5}$$

(3) Calculate the weighted average:

$$f(r_1, r_2, r_3) = \frac{1 - w(r_2)}{2} r_1 + 0.5 r_2 + \frac{w(r_2)}{2} r_3. \tag{6}$$

Varying the input features and the parameters of each model is an important key factor, as it will guarantee the absence of a relationship and equivalence between the existing probabilities of each neural network. By ordering the result of each neural network, r_2 becomes the decision between learning "attack" or "normal." If $r_2 > 0.5$, the final output is labeled "attack," $r_2 < 0.5$, and the event is considered "normal."

(i) In our proposed function, we have augmented the decision boundary from 0.5 to 0.6 to have a clear decision on the processed event. For instance, if $r_2 > 0.6$, then this is a clear consensus of two algorithms," as it is codified in $w(r)$ as a clear "attack" by two algorithms. This same reasoning is true for $r_2 < 0.4$ (two algorithms labeled the event as "normal").

(ii) In the gray area of $w(r)$, where $0.4 \le r_2 \le 0.6$, the formula gives a weight to both extreme answers (r_1 and r_3), which takes into account the fact that that we do not really have a consensus. As r_2 changes from 0.4 to 0.6, the weight shifts gradually from r_1 to r_3, making sure the function f is continuous.

(d) Weighted Average

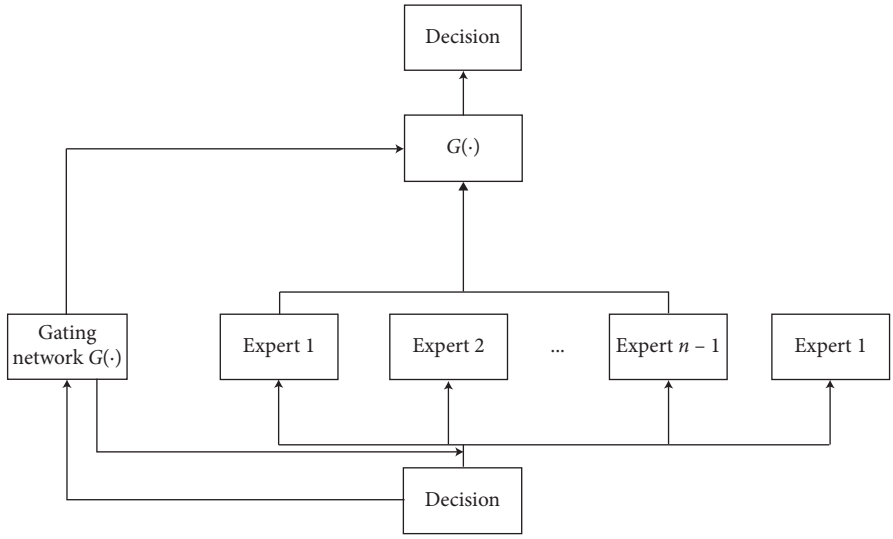

Figure 3: Mixture of experts layer.

One issue we have noticed when using our formula is that the function becomes noncontinuous when r_2 crosses 0.5. For example, if $r_1 = 0$, $r_2 = 0.49$, and $r_3 = 1$, applying the average of the consensus votes ("normal") would result in $(r_1 + r_2)/2 = 0.47$. If r_2 changes slightly to $r_2 = 0.51$, the consensus vote changes to "attack," so the average of the consensus votes would be $(r_2 + r_3)/2 = 0.53$, which is a big change.

To solve this problem, we decided to design a kind of generalization of equation (2) by adding a fully connected final layer with softmax activation. The new designed function will have the following expression:

$$f(p_1, p_2, p_3) = \alpha p_1 + \beta p_2 + \gamma p_3, \tag{7}$$

where α, β, and γ are trainable parameters. The architecture of this ensemble module is depicted in Figure 4. The main difference between this architecture and that illustrated previously in Figure 1 is that here the neural networks are trained twice.

The first time is when the 3 NNs are pretrained independently until an acceptable classification accuracy is reached. The second time is that when we build a larger neural network with the 3 NNs in parallel by adding a softmax layer, we will train the last layer so that the parameters α, β, and γ are configured to always follow the rule of the majority.

(e) Majority System Based on Reliability

Another alternative solution to the issue of the Majority Function (M.F) is to have one reliable voter decide the class of an observed event when r_2 is close to 0.5. This is done using a reliable system, as depicted in Figure 5, which will be used interchangeably with the Majority Function.

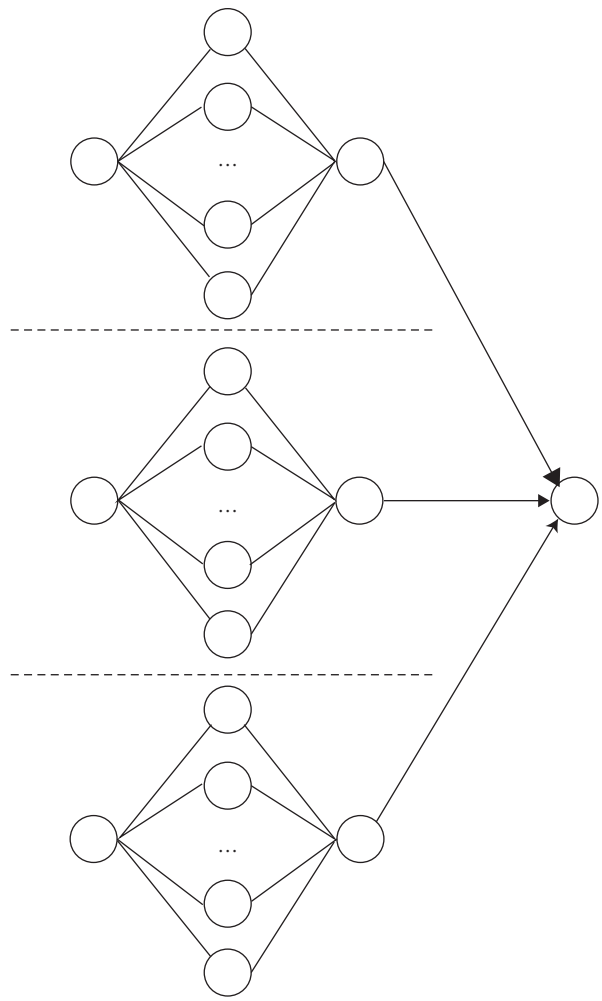

Figure 4: Ensemble module: weighted average layer.

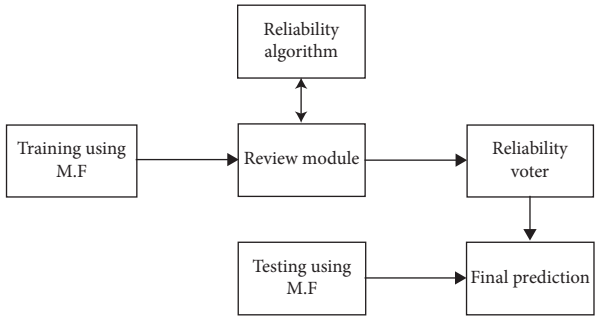

Figure 5: Majority function with reliability system.

For this, we propose the following steps with a modified version of the Majority Function:

(1) Order the results (p_1, p_2, p_3) of each neural network in ascending order (r_1, r_2, r_3).
(2) Take the function

$$w(r) = \begin{cases} 0, & \text{if } r < 0.4, \\ \text{Review Reliable Algorithm}, & \text{if } 0.4 \le r \le 0.6, \\ 1, & \text{if } r > 0.6. \end{cases} \tag{8}$$

(3) Calculate the weighted average:

$$f(r_1, r_2, r_3) = \frac{1 - w(r_2)}{2} r_1 + 0.5 r_2 + \frac{w(r_2)}{2} r_3. \tag{9}$$

(i) By following the same reasoning as before, the final decision of the Majority Function is clear when $r_2 > 0.6$ (or $r_2 < 0.4$).
(ii) If r_2 is near 0.5, the most reliable voter, which correctly labeled most of the events during training, will classify events in test phase.

This approach proved to be the best and was used in the final model. The details of operation and reasoning of the majority function based on reliability are presented in the implementation section.

4. Experimentation and Discussion

In order to analyze our proposed approach, we implemented the whole model in Python using PyTorch library. Details of the hardware and software used during the implementation are given as follows:

(i) *Hardware*: Dell XPS 12 (9Q33)-12.5″-Core i7 4500U-8 GB RAM
(ii) *Software*: Jupyter Notebook 6.0.0, Python v2.7.15+
(iii) *Python Libraries*: PyTorch v1.2.0 and Pandas v0.25.3 for preprocessing

4.1. Experimentation. We have used the files KDDTrain+.csv and KDDTest+.csv for training and testing, respectively. The sample sets were randomized in order to avoid any possible bias in the presentation order of the sample patterns to the ANNs. 20% of the training set was used for validation. Details of the architecture of each specialized neural network are shown in Table 3.

Note that we have tried to vary most of the parameters studied by authors interested in intrusion detectors based on a neural network. The variation of these parameters will also make the predictions of the different learners. However, since we combine these predictions, we have not modified some parameters such as the activation function, so that the predictions are in the same range of values.

Figure 6 shows the performance of the three base learners. Each neural network has been trained for 30 epochs. Note that although we used a validation set, we did not perform any regularization techniques such as early stopping, because our goal is not to improve the generalization of models but to create specialized models. Thus, we might see some overfitting phenomena for some base learners (e.g., NN3). We remind the reader that our approach to creating weak neural networks follows the same training/testing procedure used in dropout regularization:

(i) The input features are dropped only during training; the base learners are tested on the full 90 features
(ii) Since each model drops 60 features ($P_{\text{drop}} = 2/3$), the weights are scaled by the chosen dropout rate (units are multiplied by $P_{\text{keep}} = 1 - P_{\text{drop}}$ at test time)

To evaluate the performance of the weak neural networks and overall modules, we used 4 evaluation metrics that take the following parameters into account:

(i) True positive (TP) represents the correct classification of an intrusion
(ii) False positive (FP) is the incorrect classification of a normal user taken as an attack
(iii) True negative (NP) represents a correct classification of a normal activity
(iv) False negative (FN) is an instance where the intruder is incorrectly classified as a normal activity

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{10}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{11}$$

Recall (sensitivity) measures the proportion of the positive values that are correctly classified:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{12}$$

Table 3: Parameters of the base neural network learners.

Parameter/base learner	NN1	NN2	NN3
No. of input features during training	0, 1, . . ., 29	30, 31, . . ., 59	50, 51, . . ., 89
No. of hidden layers		Single hidden layer	
No. of hidden nodes	10	20	25
Optimizer	Stochastic gradient descent	Resilient back-propagation	Adam
Learning rate	$1e-3$	$3e-4$	$1e-4$
Batch size	256	Full batch	512
Hidden activation function		*RelU* activation function	
Loss function		*Cross Entropy* loss function	
Activation function		*Sigmoid* activation function	

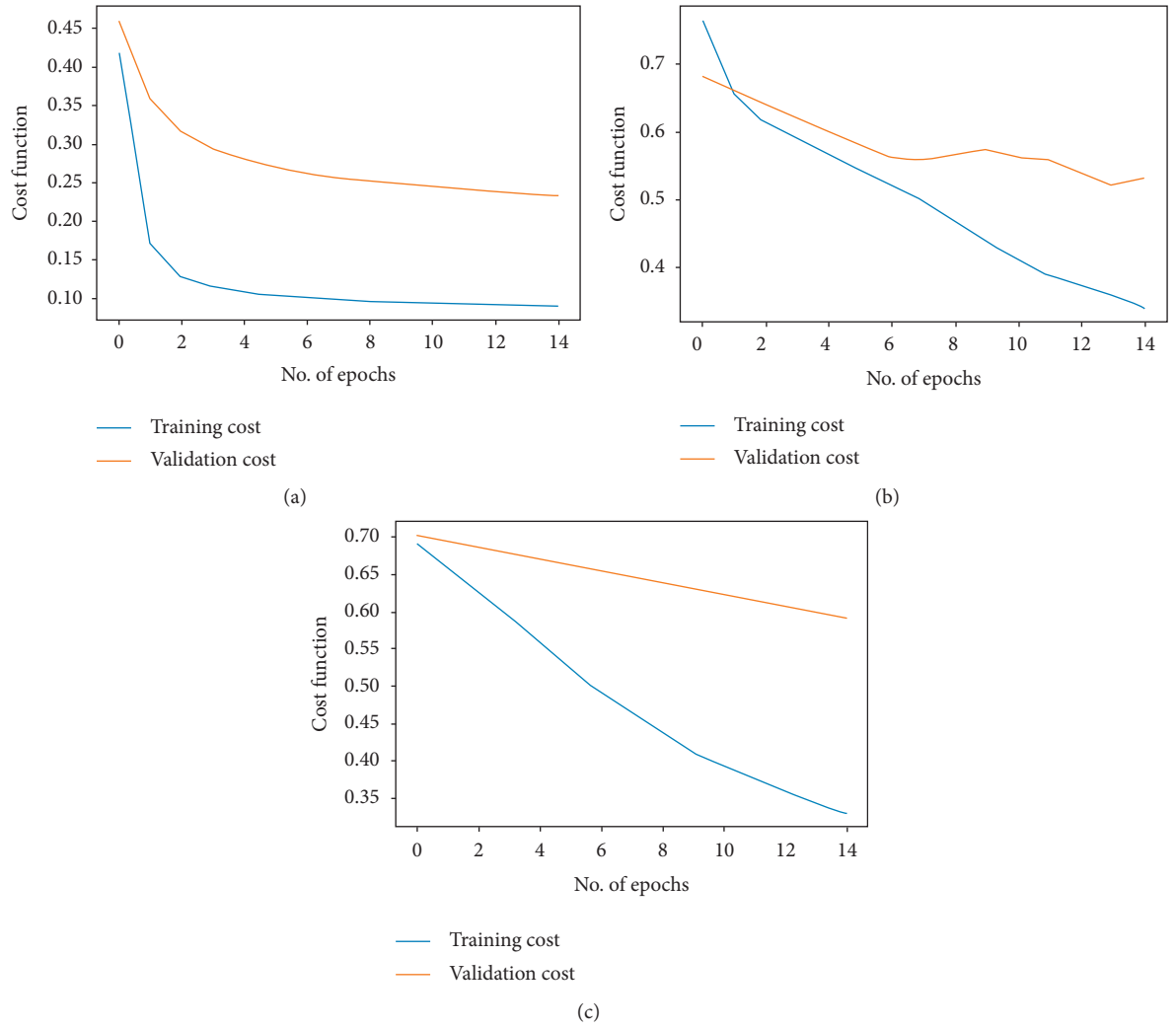

Figure 6: Performance of the three neural networks. (a) NN1. (b) NN2. (c) NN3.

F-score is the weighted average of precision and recall. Therefore, this score takes both false positives and false negatives into account:

$$F-\text{score} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}. \quad (13)$$

Table 4 shows the classification accuracy of the base learners.

Table 4: Classification accuracy of the base learners.

Base learner	Accuracy (%)
NN1	73
NN2	59
NN3	70

Next, each ensemble model was implemented and compared; results are summarized in Table 5.

According to Table 5, the worst performance of the ensemble models is the Hard Voting approach. This is because it considers less information compared to other approaches (the binary output of the voter).

The weighted layer on the other hand showed a moderate improvement in all performance metrics, but the results were not significant enough to be considered. Also, the weighted layer required a second round of training to calibrate the weighted or the trainable parameters α, β, and γ, which does not meet the purpose of this study.

The most interesting ensemble approaches for us were the majority function and the MoE layer because they both improved the classification accuracy of the voters without proportionally using more resources for training. Although we initially expected that the MoE layer will significantly improve the results because the whole model was designed for this specific purpose, this was not the case and we explain this by the following reason: the MoE layer uses the Gater in order to train only a tiny part of the network (namely, 3 experts) given one training example, whereas, usually, we would have to propagate the input through the whole network (in this case all experts). Each expert sees, therefore, not all samples but just a subset and specializes (becomes an expert) on those. In other words, the learners in the MoE layer become experts *after* the training. In addition, our approach uses learners trained only on a subset of features, so they become experts on these features. So, using learners that are *already experts* on a subset of features to recognize a certain type of observations did not help the overall systems, which is the reason why we decided to move on and try to enhance the Majority Function instead.

One issue we mentioned in the Majority Function, which the weighted layer did not solve, is how the decision changes when one of the voters is in the gray area (near 0.5) and the other voters clearly choose different classes (one voter votes >0.6 and the vote of the second model is <0.4). So, for this issue, instead of trying to find optimal weights α, β, and γ, where $\alpha\, p_1 + \beta p_2 + \gamma\, p_3$, we propose to have one voter, which proved to be reliable during training in the gray decision area and have that voter decide for the other voters the nature of processed events, hence proposing a reliability system illustrated in Figure 5.

4.1.1. Operation of the Majority Function Based on Reliability. The goal of the reliability system is to help the ensemble model (Majority Function) overcome the gray area that the voters reach. Specifically, when two voters are decisive and vote for different classes and one voter is indecisive (gives a probability near 0.5), the reliability system is used to choose the most reliable voter of the two decisive voters. The goal is to have the reliable voter decide the nature of the event in gray areas.

The review module is launched using training data. The voters (base learners) are trained and the ensemble is created using the Majority Function proposed before. Next, the review module is executed to check if the final prediction (on training data) mislabeled an event and the issue mentioned before is present. In this case, the reliability algorithm is launched to reward voters when they correctly classified an event. The reward points are different; the highest point is awarded to the voter who correctly predicts an event and the others do not.

In testing phase, the Majority Function is used *only* when there is a clear consensus of two voters (at least two votes have probabilities >0.6 or <0.4). If one vote is in the gray area, the reliability system is executed to proceed with the most reliable voter during training.

By adding this reliability module to the Majority Function, we got significant improvements in the predictions of the ensemble model. Table 6 shows the results of the Majority Function based on reliability.

The formal time complexity of both the review module and the reliability algorithm is $O(N)$, where N is the number of training examples. Since the review module calls the reliability algorithm, the overall complexity is $O(N2)$.

As for our weak neural networks, since the models are trained in parallel, we will only consider the model with the most complex architecture, which is NN3. So, the time complexity of NN3 which has the architecture 90-25-1 is

$$\begin{aligned} &O\left(\text{num}_{\text{epochs}} \times N \times (90 \times 25 + 25 \times 1)\right) \\ &\quad = O(30 \times N \times (90 \times 25 + 25 \times 1)) = \text{O}(68250 \times N) \\ &\quad = O(N). \end{aligned} \tag{14}$$

4.2. Discussion. The final step is to present a qualitative discussion of the relevance of the proposed model by comparing our approach to the most relevant related work presented so far. Note that, as previously mentioned, we exclude works for which one-hot encoding was not used as a preprocessing step for the reasons discussed before. In addition, the scope of this comparison only focuses on works based neural networks.

Table 7 summarizes our findings during this qualitative analysis, mainly focusing on examining the model capacity

Table 5: Performance of the studied ensemble models.

Ensemble	Accuracy	Precision (%)	*F*-score (%)	Recall (%)
Majority Function	74	93	72	59
Weighted Average	73	92	73	60
Hard Voting	72	93	69	55
MoE layer	75	91	74	63

Table 6: Performance of the studied ensemble models.

Ensemble	Accuracy (%)	Precision (%)	*F*-score (%)	Recall (%)
M.F and reliability	89	91	90	88

Table 7: Comparison with other approaches.

Approach	Model capacity	Computational cost	Accuracy (%)	Precision (%)	F-score (%)	Recall (%)
NN with best cost and training function [9]	Low	Low	81.8	92.5	—	67.9
Sparse autoencoder with SMR [10]	Low	High	78.06	96.56	76.8	63.73
Sparse autoencoder with logistic regression [11]	High	High	87.2	84.6	—	92.8
RNN based IDS [12]	Low	High	81.29	—	—	—
CBR-CNN based IDS [13]	High	High	89.41	92.63	—	—
This research	*High*	*Low*	*89*	*91*	*90*	*88*

and the computational cost of the approaches. Model capacity in our study is the ability to generalize well with test data. So, in our case, the higher the classification accuracy of the model, the higher the capacity. The computational cost criterion, as the name suggests, is the amount of resources that should be allocated to train a model. Thus, deep neural networks with large number of nodes and layers require significant processing power to train and tune.

The authors in [9] used a neural network with an optimal parameter focusing mainly on the cost function and the optimizer algorithm. The model has a small architecture with one hidden layer and 25 hidden neurons, so training this model is not heavy on resources, which decreases the computation cost factor. The accuracy, however, is not that high.

The model proposed in [10] is a two-stage classifier, so it requires two training cycles before testing the model. The authors also mentioned that they used 10-fold cross-validation on the training data on some cases, which makes training longer. The final classification accuracy (including the output from the sparse autoencoder) is around 78%, which makes the capacity of the model very low.

The model in [11] has a high capacity compared to the precious work; it provides a classification accuracy of 87% but with a low precision. The model, however, uses many layers of features learning, an intermediate classifier, feature stacking, and finally a deep net. So, training such model is computationally heavy.

The use of recurrent neural networks in [12] for intrusion detection gave a moderate classification accuracy (81.29%). Compared to feedforward neural network, RNNs are difficult to train because they require memory-bandwidth-bound computation. The authors also used an important number of hidden nodes (80) considering the modest performance of the model.

Finally, the highest evaluation metrics were found in [13] and in this research. The main difference is simplicity: in our approach the results are obtained using weak feedforward neural learners with a small number of parameters, while their approach is based on conventional neural network that involves channel boosting, autoencoders, and stacked autoencoders and many intermediate functions. Also, tuning small neural networks is not computationally complex because the largest model uses only 25 hidden neurons.

We also want to mention that most of these works were compared with other techniques according to different criteria, from which we cite [28–30]. Note that, in terms of classification accuracy rate, our work also outperformed these models.

Finally, some authors followed the same preprocessing steps we applied on the NSL-KDD dataset but used different machine learning models such as SVMs [31], unsupervised learning [32], and reinforcement learning [33]. These models were not included in the comparison because the authors have used a sampling process during training and testing, while we used the full dataset. Consider the following:

(i) The authors in [31] proposed an attack detection system using SVMs which enables early discovery of the APT attack. To develop the model, they used one-hot encoding for preprocessing and PCA for dimensional reduction; next they compared the classification accuracy of four classifiers: Support Vector Machine, Naive Bayes classification, the decision tree, and neural networks. The authors claim that the SVM achieved 97.22% accuracy, but the model was trained and tested on two different subsets from the original training set (KDDTrain+) and not on the test set (KDDTest+).

(ii) In [32], the authors used an unsupervised learning algorithm autoencoder and achieved 91.70% classification accuracy. To preprocess the NSL-KDD dataset, they used one-hot encoding on the categorical features and then applied PCA for dimensionality reduction. The authors mentioned that they used 85% of the dataset for training and 15% for testing, which means that they did not work on the complete dataset as we did in this work.

(iii) For the work in [33], the authors used deep reinforcement learning on two datasets: NSL-KDD and AWID. For each dataset, they compared four reinforcement learning algorithms: Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Policy Gradient (PG), and Actor-Critic (AC). The best results are obtained for the DDQN algorithm. The authors also used a sampling function on the datasets during the preparation, but, in order to compare the results with other works, they ran the tests on the full datasets. For the NSL-KDD dataset, they achieved 89% accuracy, 91% *F*-score, 89% precision, and 93% recall. We can see that the metrics are really close to the ones we found with simpler weak neural networks.

5. Conclusion

SIEM and IDS systems are tools that process a huge amount of data on a daily basis. This involves collecting, normalizing, and detecting attacks and sending notifications in real time. This research takes into account the processing power that is dedicated to the operation of SIEM/IDS systems and proposes a new machine learning model with high detection capability and low computation resources.

Our model is based on reliability system that processes events using different simple feedforward neural networks and predicts the output using majority functions. We have also proposed a new way to create weak learners using neural networks by making them "experts" on small subsets of the input feature space. Our model achieved an accuracy of 89%, which is superior to many works and competes with deep learning approaches.

In a future study, we will evaluate the proposed system in a real network. We will also increase the number of base learners and evaluate the performance on multiclassification.

Data Availability

The datasets used to support the findings of this study are available at https://www.unb.ca/cic/datasets/nsl.html.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] D. Miller, *Security Information and Event Management (SIEM) Implementation*, McGraw-Hill, New York, NY, USA, 2011.

[2] K. Scarfone and P. Mell, *Guide to Intrusion Detection and Prevention Systems (IDPS)*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2007, http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf.

[3] Verizon, "2018 data breach investigations report 11th edition," 2018, https://www.verizonenterprise.com.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[5] M. Alazab, S. Khan, S. S. R. Krishnan, Q.-V. Pham, M. P. K. Reddy, and T. R. Gadekallu, "A multidirectional LSTM model for predicting the stability of a smart grid," *IEEE Access*, vol. 8, pp. 85454–85463, 2020.

[6] J. Huang, V. Rathod, C. Sun et al., "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017.

[7] N. Moukafih, G. Orhanou, and S. Elhajji, "Mobile agent-based SIEM for event collection and normalization externalization," *Information & Computer Security*, vol. 28, no. 1, pp. 15–34, 2019.

[8] S. S. Panwar and Y. P. Raiwani, "Performance analysis of NSL-KDD dataset using classification algorithms with different feature selection algorithms and supervised filter discretization," in *Advances in Intelligent Systems and Computing*, pp. 497–511, Springer International Publishing, Berlin, Germany, 2019.

[9] S. El Hajji, N. Moukafih, and G. Orhanou, "Analysis of neural network training and cost functions impact on the accuracy of IDS and SIEM systems," in *Codes, Cryptology and Information Security*, pp. 433–451, Springer International Publishing, Berlin, Germany, 2019.

[10] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," *Proceedings of the 9th EAI International Conference on Bio-nspired Information and Communications Technologies (Formerly BIONETICS)*, New York, NY, USA, 2016.

[11] S. Gurung, M. Kanti Ghose, M. Kanti Ghose, and A. Subedi, "Deep learning approach on network intrusion detection system using NSL-KDD dataset," *International Journal of Computer Network and Information Security*, vol. 11, no. 3, pp. 8–14, 2019.

[12] 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. 21954–21961, 2017.

[13] N. Chouhan, A. Khan, and H.-u.-R. Khan, "Network anomaly detection using channel boosted and residual learning based deep convolutional neural network," *Applied Soft Computing*, vol. 83, Article ID 105612, 2019.

[14] R. M. Swarna Priya, P. K. R. Maddikunta, M. Parimala et al., "An effective feature engineering for DNN using hybrid PCA-GWO for intrusion detection in IoMT architecture," *Computer Communications*, vol. 160, pp. 139–149, 2020.

[15] G. Suarez-Tangil, E. Palomar, A. Ribagorda, and I. Sanz, "Providing SIEM systems with self-adaptation," *Information Fusion*, vol. 21, pp. 145–158, 2015.

[16] M. Ussath, D. Jaeger, F. Cheng, and C. Meinel, "Identifying suspicious user behavior with neural networks," in *Proceedings of the 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, New York, NY, USA, 2017.

[17] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection," *Computers & Security*, vol. 75, pp. 36–58, 2018.

[18] S. Bhattacharya, S. R. K. S, P. K. R. Maddikunta et al., "A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU," *Electronics*, vol. 9, no. 2, p. 219, 2020.

[19] R. Ch, T. R. Gadekallu, M. H. Abidi, and A. Al-Ahmari, "Computational system to classify cyber crime offenses using machine learning," *Sustainability*, vol. 12, no. 10, p. 4087, 2020.

[20] C. Carlet, S. Guilley, A. Nitaj, and E. M. Souidi, "Codes, cryptology and information security," *Lecture Notes in Computer Science*, Springer International Publishing, Berlin, Germany, 2019.

[21] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, Ottawa, Canada, 2009.

[22] K. Bajaj and A. Arora, "Improving the intrusion detection using discriminative machine learning approach and improve the time complexity by data mining feature selection methods," *International Journal of Computer Applications*, vol. 76, no. 1, pp. 5–11, 2013.

[23] B. Ingre and A. Yadav, "Performance analysis of NSL-KDD dataset using ANN," in *Proceedings of the International*

Conference on Signal Processing and Communication Engineering Systems (SPACES), Vijayawada, India, 2015.

[24] J. Brownlee, *Long short-term memory networks with python: develop sequence prediction models with deep learning*, in Machine Learning Mastery, 2017.

[25] W. Wang, X. Zhang, S. Gombault, and S. J. Knapskog, "Attribute normalization in network intrusion detection," in *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, Kaohsiung, Taiwan, 2009.

[26] "Overview of neural networks," http://www.jmp.com/support/help/Overview_of_Neural_Networks.shtml.

[27] N. Shazeer, A. Mirhoseini, K. Maziarz et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-Of-Experts Layer," 2017, https://arxiv.org/abs/1701.06538.

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

[29] M. Panda, A. Abraham, and M. R. Patra, "Discriminative multinomial naive bayes for network intrusion detection," in *Proceedings of the Sixth International Conference on Information Assurance and Security (IAS)*, 2015.

[30] R. S. Naoum, N. A. Abid, and Z. N. Al-Sultani, "An enhanced resilient backpropagation artificial neural network for intrusion detection system," *International Journal of Computer Science and Network Security*, vol. 12, no. 11, 2012.

[31] W.-L. Chu, C.-J. Lin, and K.-N. Chang, "Detection and classification of advanced persistent threats and attacks using the support vector machine," *Applied Sciences*, vol. 9, no. 21, p. 4579, 2019.

[32] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5597–5621, 2019.

[33] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Systems with Applications*, vol. 141, Article ID 112963, 2020.