VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

INFORMATION SECURITY ANALYSIS AND AUDIT [CSE3501]

BIBHU BHUSHAN [18BIT0134]

RAHUL ANAND **RISHU RAJPUT** **MAYANK RAJ**

INTRUSION DETECTION USING DEEP LEARNING (Deep Auto- encoder)

Paper-1

The purpose of this survey is to classify and summarize the machine learning-based IDSs proposed to date, abstract the main ideas of applying machine learning to security domain problems, and analyze the current challenges and future developments. For this survey, we selected representative papers published from 2015 to 2019, which reflect the current progress. Several previous surveys [3–5] have classified research efforts by their applied deep learning.

Architecture/ model/ pseudo code developed

Deep brief networks (DBNs)
Deep neural networks (DNNs)
Convolutional neural networks (CNNs)
Recurrent neural networks (RNNs) are supervised learning models, while auto-encoders, restricted Boltzmann machines (RBMs), and generative adversarial networks (GANs) are unsupervised learning models.

Auto- encoder.

An auto-encoder contains two symmetrical components, an encoder and a decoder, as shown in Figure 3. The encoder extracts features from raw data, and the decoder reconstructs the data from

the extracted features. During training, the divergence between the input of the encoder and the output of the decoder is gradually reduced. When the decoder succeeds in reconstructing the data via the extracted features, it means that the features extracted by the encoder represent the essence of the data. It is important to note that this entire process requires no supervised information. Many famous auto-encoder variants exist, such as denoising auto-encoders [14,15] and sparse auto-encoders .

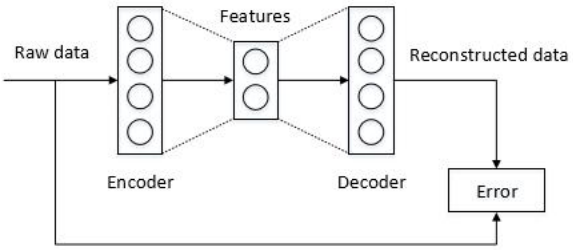


Figure 3. The structure of an autoencoder.

Datasets analyzed in the paper with the performance results

(1) DARPA1998
(2) KDD99
(3) NSL-KDD
(4) UNSW-NB1

Comparison

Table 4. Comparison of various deep learning models

Algorithms	Suitable Data Types	Supervised or Unsupervised	Functions
Autoencoder	Raw data; Feature vectors	Unsupervised	Feature extraction; Feature reduction; Denoising
RBM	Feature vectors	Unsupervised	Feature extraction; Feature reduction; Denoising
DBN	Feature vectors	Supervised	Feature extraction; Classification
DNN	Feature vectors	Supervised	Feature extraction; Classification
CNN	Raw data; Feature vectors; Matrices	Supervised	Feature extraction; Classification
RNN	Raw data; Feature vectors; Sequence data	Supervised	Feature extraction; Classification
GAN	Raw data; Feature vectors	Unsupervised	Data augmentation; Adversarial training

Paper-2

In this paper, They adopt deep Auto-Encoder and LightGBM algorithms to construct model. There are two phases in this model. In the first phase, They applied the clean dataset into deep Auto-Encoder network. They only save the intermediate results as the part of input data of the second phase. In the second phase, The concatenate the previous result and the clean dataset into new dataset. Then the new dataset is inputted in the LightGBM model. In Section II, They analyse the related work. In Section III, They introduce the detail of our design. In Section IV, They introduce raw data, data pre-processing and evaluation metrics. In Section V, They give the results of the experiment and compare performance of various methods. Finally, in Section VI, They conclude this paper.

Auto-Encoder

An Auto-Encoder includes two parts.

Encoder: This part compresses the input into a latent representation, which can be represented by the encoding function $h = (x)$.

Decoder: This part can reconstruct the input from the latent representation, which can be represented by the decoding function $y = (h)$

LightGBM

SVM, GBDT, XGBoost, etc., are used frequently in IDS, but they will become very slow during training phase. Besides, their efficiency and scalability are worrying when the number of features or data is so large. The reason is that when splitting a node, they need to scan all the data to estimate the splitting gain of all possible segmentation point for each feature to get obtain the maximum information gain. So we chose LightGBM algorithm. LightGBM mainly includes two improved algorithms: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

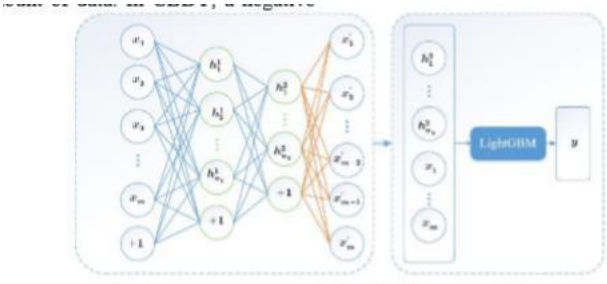


Figure 1: Deep Auto-Encoder based LightGBM architecture.

In this experiment, they evaluate their proposed model on use KDD 99 dataset which is the benchmark dataset in intrusion detection system.

Table 1: data distribution.

Label	Training set	Test set
Normal	97278	60593
DOS	391458	229853
R2L	1126	16189
U2R	52	228
Probe	4107	4166

Table 2: mainstream models comparison.

model	Training Time (s)	Acc (%)
LightGBM	422	94.7
XGBoost	4395	92.5
SVM	-	87.7
CNN	-	91.3
LR	79	86.3
RF	159	90.5

where '-' means these models are extremely slow to train.

Paper-3

Auto –Encoder

Self- taught learning

In the proposed Deep Neural Network and adaptive Self- Taught based Transfer Learning (DST-TL) technique, we have exploited the concept of self-taught learning to train Deep Neural Networks for reliable network intrusion detection. In the proposed method, a pre-trained network on regression related task is used to extract features from NSL-KDD dataset.
It is experimentally observed that the auto-encoder trained on the combined original and extracted features is stable and offer good generalization in comparison to the sparse auto-encoder trained on original features alone.

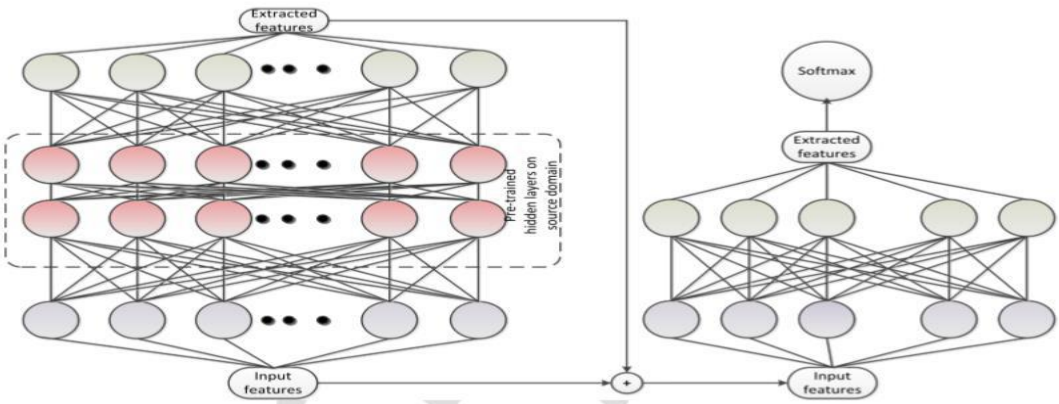


Figure 2: Use of self-taught learning in training of deep sparse auto-encoder

Dataset

Table 1: Percentage of positive and negative samples in $KDDTest^{+}$ and $KDDTest^{-21}$ dataset

Dataset	Total samples	Normal Samples	Abnormal Samples
20% of $KDDTrain^{+}$	25193	13449 (53.4%)	11744 (46.6%)
$KDDTest^{+}$	22544	9711 (44%)	12833 (56%)
$KDDTest^{-21}$	11850	2152 (19.2%)	9698 (81.8%)

Table 2: Types of features present within NSL-KDD dataset

Feature number	Type		Feature name
1	Basic Features		Duration
2			Protocol_type
3			Service
4			Flag
5			Src_bytes
6			Dst_bytes
7			Land
8			Wrong_fragment
9			Urgent
10			Hot
11	Content features		Num_failed_logins
12			Logged_in
13			Num_compromised
14			Root_shell
15			Su_attempted
16			Num_root
17			Num_file_creations
18			Num_shells
19			Num_access_files
20			Num_outbound_cmds
21			Is_hot_login
22			Is_guest_login
23	Traffic features	Same service features	Count
24			Srv_count
25			Serror_rate
26			Srv_serror_rate
27			Rerror_rate
28			Srv_rerror_rate
29			Same_srv_rate
30			Diff_srv_rate
31			Srv_diff_host_rate
32		Same host features	Dst_host_count
33			Dst_host_srv_count
34			Dst_host_same_srv_rate
35			Dst_host_diff_srv_rate
36			Dst_host_same_src_port_rate
37			Dst_host_srv_diff_host_rate
38			Dst_host_serror_rate
39			Dst_host_srv_serror_rate
40			Dst_host_rerror_rate
41			Dst_host_srv_rerror_rate

All the attacks in the NSL-KDD data belong to the following four attack categories:

Table 3: Different types of attacks in NSL - KDD dataset

PROB	**IP-sweep**		**Port-sweep**		**NMAP**		**Satan**	
R2L	Warezmaster	Wareclient	Phf	SPY	Multi-HOP	Guess-password	FTP-Write	IMAP
U2R	Perl		Root-kit		Load-module		Butter-overflow	
DOS	Smurf	Land	Ping-0f-death		Neptune	Back	Teardrop	

Table 4: Parameters setting of deep sparse auto-encoder (for extracting features) using hidden layers trained on source domain task

Parameters	**Layer trained according to the target domain**	**Trained layers on regression related task in the source domain**		**Layer trained according to the target domain**
	Layer 1	Layer 2	Layer 3	Layer 4
Coefficient of L_2 weight Rregularization	0.0002	0.00003	0.00001	0.00001
Coefficient of Sparsity Regularization	4	4	4	4
Maximum number of epochs	200	500	250	150
Number of neurons	124	300	220	200

Table 5: Parameters setting of the sparse auto-encoder trained on the combined original and extracted features

Parameters	**Without self-taught learning approach**		**With self-taught learning approach**	
	Layer 1	Layer 2	Layer 1	Layer 2
Coefficient of L_2 weight regularization	0.0002	0.00002	0.00002	0.00002
Coefficient of Sparsity Regularization	5	4	4	3
Maximum number of epochs	100	200	200	100
Number of neurons	100	80	166	180

	AUC-ROC		**AUC-PR**		**Detection Rate**		**False Alarm Rate**	
	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$
MLP	0.88±0.02	0.75±0.02	0.92±0.01	0.93±0.01	0.65±0.01	0.28±0.01	0.38±0.02	0.51±0.03
DBN	0.85±0.03	0.74±0.07	0.90±0.02	0.92±0.02	0.65±0.01	0.27±0.01	0.40±0.01	0.53±0.02
NLPCA	0.89±0.03	0.64±0.03	0.92±0.01	0.91±0.01	0.66±0.01	0.25±0.01	0.35±0.01	0.47±0.02

Table 7: Parameter setting of MLP, NLPCA, and DBN during training

MLP	**DBN**							**NLPCA**
Number of neurons in the hidden layer	No of neurons			No of epochs	Batch size	Momentum	Learning-rate	No of neurons in encoding layer
	Layer1	Layer2	Layer3					
25	80	75	50	50	2	0.04	0.001	18

Comparison

Performance evaluation using detection and false alarm rate

Table 8: Performance comparison of the trained sparse auto-encoders for ten independent runs

	Spars auto-encoder without TL		**Sparse auto-encoder with TL**	
	Detection rate	False alarm rate	Detection rate	False alarm rate
$KDDTest^{+}$	0.86±0.09	0.13±0.09	0.85 ±0.05	0.14±0.05
$KDDTest^{-21}$	0.81±0.13	0.18±0.13	0.80±0.06	0.19±0.06

Performance evaluation using AUC-ROC, AUC-PR, and accuracy

Table 9: Performance comparison of trained sparse auto-encoders on test data

	Sparse deep auto-encoder without TL		**Deep sparse auto-encoder with TL**	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
$KDDTest^{+}$	0.89±0.01	0.91±0.01	0.91± 0.01	0.93±0.01
$KDDTest^{-21}$	0.61±0.03	0.88±0.01	0.69±0.01	0.91±0.01

Table 10: Performance comparison for ten independent runs of the trained sparse auto-encoders in terms of accuracy

	Sparse auto-encoder without TL (Accuracy)		**sparse auto-encoder trained using TL (Accuracy)**	
	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$
1	**85.30**	**74.50**	**84.60**	**79.90**
2	84.50	72.70	81.70	71.40
3	80.60	63.30	82.40	69.40
4	74.30	78.30	82.40	72.40
5	80.60	63.40	82.60	71.50
6	75.80	75.50	83.80	74.90
7	72.50	80.70	82.20	72.10
8	76.50	73.90	83.80	72.30
9	77.40	57.40	81.80	69.40
10	73.30	79.20	82.40	77.70
	78.08±4.27	**71.89±7.43**	**82.77±0.91**	**73.1±3.09**

Performance comparison of the proposed DST-TL technique with state-of-the-art techniques

Table 11: Performance comparison of the proposed technique with the existing methods

Classifier	Accuracy	
	$KDDTest^{+}$	$KDDTest^{-21}$
J48	81.05	63.97
Naive Bayes	76.56	55.77
NB Tree	82.02	66.16
Random forest	80.67	63.25
Random tree	81.59	58.51
MLP	77.41	57.34
NLPCA	76.73	56.58
DBN	76	54.55
SVM	69.52	42.29
Fuzziness based IDS (Experiment2)	84.12	68.82
DST-TL(best accuracy in ten independent runs)	**84.60**	**79.90**
DST-TL(average performance)	**82.77±0.91**	**73.1±3.09**

Paper-4

Algorithm Using :

- Deep Neural Network for Network Intrusion Detection
  Auto- encoder for Network Intrusion Detection

This chapter looks at using auto encoders for the task of network intrusion detection. An auto encoder is a type of neural of network that is trained in such a way that it aims to copy its input to its output. An auto encoder is designed so that it reproduce its input at the outer layer. One of the main difference between an auto encoder and a multilayer perception, or deep neural network, is that the number of input neurons is equal to the number of output neurons .

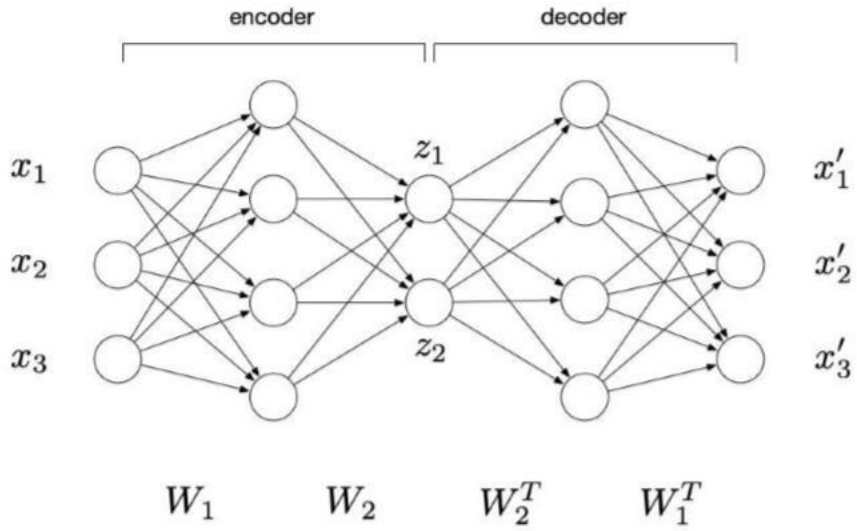


Figure 4.1: Example neural network structure for Autoencoder

Datasets

Table 2.3: CIC IDS 2017 Dataset Overview

Date	# of Flows	# of Attacks	Description
Monday	529,918	0	Normal network activities
Tuesday	445,909	7,938	FTP-Patator
		5,897	SSH-Patator
Wednesday	692,703	5,796	DoS slowloris
		5,499	DoS Slowhttptest
		231,073	DoS Hulk
		10,293	Dos GoldenEye
		11	Heartbleed
Thursday Morning	170,366	1507	Web Attack - Brute Force
		652	Web Attack - XSS
		21	Web Attack - SQL Injection
Thursday Afternoon	288,602	36	Infiltration
Friday Morning	191,033	1966	Bot
Friday Afternoon 1	286,467	158,930	PortScan
Friday Afternoon 2	225,745	128,027	DDoS
Total	**2,830,743**	**557,646**	**19.70% attack traffic**

Table 2.2: Description of Features for ISCX IDS 2012 Dataset

No.	Feature	Description	Type	Unique Values
1	SrcIP	Source IP address	Categorical	2,478
2	DstIP	Destination IP address	Categorical	34,552
3	SrcPort	Source port for TCP and UDP	Categorical	64,482
4	DstPort	Destination port for TCP and UDP	Categorical	24,238
5	AppName	Application name	Categorical	107
6	Direction	Direction of flow	Categorical	4
7	Protocol	IP protocol	Categorical	6
8	Duration	Flow duration in fractional seconds	Continuous	N/A
9	TotalSrcBytes	Total source bytes	Continuous	N/A
10	TotalDstBytes	Total destination bytes	Continuous	N/A
11	TotalBytes	Total bytes	Continuous	N/A
12	TotalSrcPkts	Total source packets	Continuous	N/A
13	TotalDstPkts	Total destination packets	Continuous	N/A
14	TotalPkts	Total packets	Continuous	N/A

Table 4.1: Autoencoder Evaluation Results - CIC IDS 2017

Metric	Result
True Negative	681, 307
False Positive	89
False Negative	128, 065
True Positive	38, 902
Area Under the Curve	0.6164
Accuracy	0.8489
Error Rate	0.1512
True Positive Rate (Sensitivity, Recall, Detection Rate)	**0.2330**
True Negative Rate (Specificity)	0.9999
False Positive Rate	**0.00013**
False Negative Rate	**0.7670**
Precision	0.9977
F1 Measure	**0.3778**

Comparison

Table 3.10: CIC IDS 2017 Result Comparison [13]

Technique	Avg. DR	Avg. FPR
Hypbrid IDS - Decision Tree + Rule-based [13]	0.94475	.01145
WISARD [30]	0.48175	0.02865
Forest PA [12]	.92920	0.03550
J48 Consolidated [52]	0.92020	0.06645
LIBSVM [19]	0.54595	0.05130
FURIA [50]	0.90500	0.03165
Random Forest [13]	0.93050	0.01880
REP Tree [13]	0.91640	0.04835
MLP [13]	0.77830	0.07350
Naive Bayes [13]	0.82510	0.33455
Jrip [13]	0.93400	0.04470
J48 [13]	0.91990	0.05040
DNN with IPs	**0.9993**	**0.0003**
DNN without IPs	**0.9677**	**0.0052**

Table 3.11: CIC IDS 2017 Result Comparison [95]

Technique	Precision	Recall	F1 Score
KNN [95]	0.96	.96	.96
Random Forest (RF) [95]	0.98	0.97	.97
ID3 [95]	.98	0.98	.98
Adaboost [95]	0.77	0.84	.77
MLP [95]	**0.77**	**0.83**	**.76**
Naive-Bayes [95]	0.88	0.04	.04
Quadratic Discriminant Analysis (QDA) [95]	0.97	0.88	.92
DNN with IPs	**0.9987**	**0.9993**	**0.9990**
DNN without IPs	**0.9784**	**0.9677**	**0.9730**

Paper-5

Algorithm Used

Deep learning
Self- taught learning
Deep Belief Network
Convulation Neural Network
Recurrent neural network

Auto- Encoder

This is a very popular technique used in deep learning research. It is an unsupervised neutral network-based feature extraction algorithm, which learns the best parameters required to reconstruct its output as close as to the input as possible.

NSL-KDD was proposed to overcome the limitation of KDD Cup dataset. A sparse autoencoder is a neural network consisting of an input, a hidden, and an output layer. The input and output layers contain N nodes and the hidden layer contains K nodes. The target values in the output layer set to the input values, i.e.,

$x\hat{i} = xi$

The sparse auto-encoder network finds the optimal values for weight matrices, $W \in \mathbb{R}K\times N$ and $V \in \mathbb{R}N\times K$ and bias vectors, $b1 \in \mathbb{R}K\times 1$ and $b2 \in \mathbb{R}KN\times 1$ using back-propagation algorithm while trying to learn the approximation of the identity function, i.e., output $x\hat{}$ similar to x. Sigmoid function, $g(z) = 1\ 1+e-z$ is used for the activation, hw,b of the nodes in the hidden and output layers:

$hW,(x) = g(W\ x + b)$ (1)

$J = 1\ /2m\sum \|xi - x\hat{i}\|2m\ i=1+\lambda\ 2(\sum W2 + \sum V2\ n,k\ ,n + \sum b1\ 2\ + k\ \sum b2\ 2\ n\ ) + \beta\sum KL(\rho\|\rho\hat{j})K\ j=1$

The cost function to be minimized in sparse auto-encoder using back-propagation is represented by Eqn. 2.

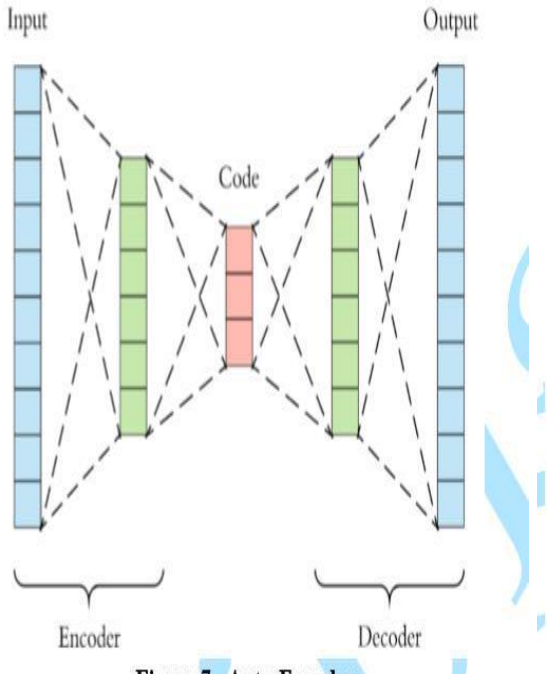


Figure 7. Auto-Encoder

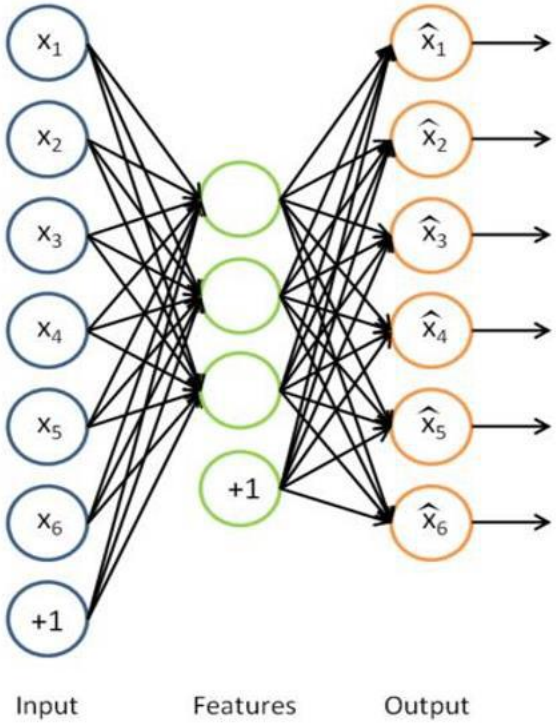


Figure 8. Encoder

This is done by the use of back propagation and setting the target value to be equal to the input, trying to learn an approximation to the identity function. An Auto-encoder typically has an input layer, output layer (the same dimension as the input layer) and a hidden layer. The hidden layer normally has a smaller dimension than that of an input (known as under complete or sparse auto-encoder).

INTRUSION DETECTION USING DEEP LEARNING TECHNIQUE: A REVIEW

Olatunde Iyaniwura [1], Festus Ayetiran [2], Alaba T. Owoseni [1]

[1] Department of Mathematical Sciences, Kings University, Odeomu, Nigeria
[2] Department of Computer Science, Elizade University, Ilara-Mokin, Nigeria

Corresponding Author: Olatunde Iyaniwura, niwura_51@yahoo.com

ABSTRACT: This work is a survey paper that represents the review of the current research in deep learning. Developing a flexible and efficient NIDS for unforeseen and unpredictable attacks, deep learning based provides more efficient result. Shallow learners mostly depend on the features used for creating the prediction model. On the other hand, deep learners have the potential to extract better representations from the raw data to create much better models. The self-taught learning algorithms work best when they are allowed to learn rich models using large amounts of unlabeled data (millions of examples). The STL model could be implemented in an environment where data is unclean. Though, an important consideration is to recognize that the best practice for applying any model, would be to ensure that the data is clean. Deep learning give room for the elimination of manual work done by system administrator.

KEYWORDS: Intrusion Detection, shallow learning deep learning, Self-Taught Learning Encoder.

1. INTRODUCTION

The effectiveness of Intrusion Detection System [IDS] depends on its "configurability" (ability to define and add new specifications attack), robustness (fault tolerance) and the small amount of false positive and negative [FA12]. This is not so because of the IDS which often required to evaluate events on the network in real time. This requirement cannot be met because of the large volume of events on the network, high false error rate (and associate cost), difficulty in obtaining reliable training data, longevity of training data and behavioral dynamics of the system [S+17].

For the above, mostly shallow machine is being used for building the anomaly detection models. Shallow learners mostly depend on the features used for creating the prediction models. The current situation will reach a point whereby reliance on such techniques leads to inaccurate and ineffective detection.

Because of the above challenges, there is need to create an effective and widely accepted anomaly detection technique capable of overcoming limitation induced by the ongoing changes occurring in modern networks the following are some of the machine learning technique being used, Artificial Neural Network, Support Vector Machine, Decision Tree etc.

The application of these techniques has offered improvement in detection accuracy but the limitations with their usage are as follows: -

Comparatively high level of human expert interaction is required. Expert knowledge needed to process data e.g. identifying useful data and patterns. This is not only labour intensive and expensive but also error prone [N+15]. Similarly, a large quantity of training data is required for operation (with associated time overheads) which can become challenging in a heterogenous and dynamic environment.

The use of deep learning, which is a sub-set of machine learning can overcome some of the above limitations. Deep learning-based algorithms provide better result than the existing machine learning in this area. Intrusion Detection involves monitoring network traffic, detecting attempts to gain unauthorized access to a system or resources and alerting the appropriate person so that countermeasures can be taken [PB13].

Traffic monitoring is in general the responsibility of Intrusion Detection System. It captures packets at particular time frame, namely every millisecond from those captured packets, IDs extracts more detail information such as packet size, the origin of IP address, the attacked port number and also its packet type like, ICMP, TCP, UDP as well.

IDS can be classified according to its detection system, the source of data and behavior. The detection system is made up of Misuse (Knowledge/Signature) and Anomaly (Heuristic). The Misuse detection is normally used for detecting known attacks. It requires that all known threats will be defined first; and the information regarding this threat be submitted to the IDS. The misuse detection is as good as its database. It has a relatively low rate of false alarms, and it cannot detect a new attack for which a signature is not yet installed.

The Anomaly detection is based on defining the

network behavior. There are two main approaches used in anomaly detection: self-learning approach or programmed anomaly detection.

In the self-learning approach, the anomaly detection system will begin to automatically monitor events such as, live network traffic on the environment it has been implemented on and attempt to build information on what is considered normal. This is online learning,

The programmed approach or offline learning, anomaly-based IDS must manually learn what is considered normal behavior by having a user or some form of function teaching the system through input of information [PB13].

Anomaly detection can also detect malformed packets. Anomaly system detects anomalous behavior. It must just the trained to recognize normal system activity. The two phases of anomaly detection system consist of the training phase and testing phase. The training phase consists of building a profile of normal behaviors and testing phase has the existing traffic being compared with the profile created in the training phase. [Van17].

The anomaly detection is computational expensive because every metrics are often maintained that need to be updated every system activity and due to insufficient data, they may be trained incorrectly to recognize an intrusive behavior as normal due to insufficient data.

2. RELATED WORK

[JR13]; due to the limitation above, SVM cannot be used for IDS domain without a variant in SVM framework to address the mentioned limitations.

For solving the features selections of SVM, [E+10] made use of Principal Component Analysis (PCA) with SVM as an approach to select optimum subset.

Intrusion detection based on SVM optimized with Swarm intelligence. [EP14], stated in their paper that SVM performance depends on selection of the appropriate parameters. The IDS model based on Information Gain for features selection combined with the SVM classifiers. The parameters for SVM are selected by Swarm optimization or artificial bee colony, using the NSL-KDD, dataset. The model achieved higher detection rate and lower false alarm rate than regular SVM.

[MW14], in a paper proposed intrusion detection system using data mining techniques. The SVM and PSO (particle swarm optimization) parameter optimization was first performed by the PSO using SVM in optimizing the value of cost and gammar parameter. This was followed by the PSO performing feature optimization as to obtain optimized feature. The SVM was finally used to optimize the parameters and features to obtain higher degree of accuracy.

In 2014, [Ift15] made use genetic algorithm to search the genetic principal components that offer a subset of features with optimal sensitivity and the highest discriminatory power. The SVM was used as the classifier. The results show that the proposed method enhanced SVM performance in intrusion detection.

[U+12] mentioned the two main challenges when generating the training data needed for IDS modeling. First, network traffic is very complex and unpredictable and model in subject to change over time since anomalies are continuously evolving.

As attack techniques and patterns change, previously gained information about how to tell them apart from normal traffic may no longer valid. To overcome this, machine learning technique was adopted to implement semi-supervised anomaly detection system where the classifier was trained with 'normal' traffic data only, so that knowledge about anomalous behaviour can be constructed and evolve in a dynamic way. Using the machine learning, Discriminative Restricted Boltzmann machine with expressive power of generative models with good classification accuracy capability to infer part of its knowledge from incomplete training data.

[WP15] observed that a single artificial neural network produces over fitting on intrusion detection system. Their work used two of ANN namely Lavenberg- Marquaralt and Quaxi-Newton to overcome the issue. Both algorithms are used to detect computer networks from attack. In addition, they use Possibilistic Fuzz C- means (PFCM) before going into the neural network ensemble Naïve Bayesian Classification method in used. The outcome shows that the neural network ensembles method produces a better average accuracy than previous researches.

[QPM14] implemented a fuzzy logic-risk analysis technique for analyzing the alarms generated. This is because of serious concern in information security false alarm which in having severe impact through the distribution of information availability

[S+11] in their work on new forms of attacks on the computer system. They discovered that these attacks gave room for high rate of false positive alarms. They tried to reduce the false alarms using alert clustering mechanism and system hibernation capabilities Alert clustering and authentication algorithms were used

Detecting new attacks in real time is a big problem. Alrawashden et. al. approach the problem by using deep learning method anomaly detection using a Restricted Boltzmann machine (RBM) and deep

belief network. This approach was able to perform self-learning to extract features from unlabeled data. With this, self-learning in deep learning is essential in the design of online intrusion detection system. With this, human involvement is eliminated.

[AA18], discussed about attackers always changing their tools techniques in IDS on a challenge talk. Making use of various machine learning classifiers based on KDD intrusion dataset, several experiments were performed and tested to evaluate the efficiency and the performance of J48, Random forest, Random time, Decision table, MLP, Naïve Bayes and Bayes network. The experiment resulted in that no simple machine learning algorithm can handle efficiently all types of attacks. Furthermore, to save the availability and confidentiality of the networks resources, the true positive and average accuracy rate are not sufficient to detect the intrusion false negative and false positive rates are also needed to be taken into consideration.

Many challenges arrive while developing a flexible and effective NIDS. For unforeseen and unpredictable attacks, Ponkarthika et.al made use of deep learning-based approach to implement long short-term memory (LSTM) architecture applied to recurrent neural network (RNN) and train the IDS model using KDD cup 99 dataset. The proposed method is to detect the network behavior whether it is normal or affected based on the past observations. After the experiments by comparing it to IDS classifiers the LSTM-RNN is a better classifier.

[AJ15] brought in the use of particle swarm optimization (PSO) (another optimization approach based on the behavioral study of animals/birds) for the development of reliable and intelligent intrusion detection system.

The main advantage of PSO is that it is easy to implement and only a few input parameters are needed to be adjusted and is in non-linear optimization problem. In their research, the following factors affected the performance of IDS. First is the selection and extraction of relevant feature. If all features are evaluated, then it degrades the IDS performance Hybridization of different supervised machine better clarification.

Hybridization of PSO with Roughset, ANN, SVM were made use of for better result with this, a scalable solution for detecting network-based anomaly was obtained. Based on the behavioral study of animals/birds.

[Ift15] worked on the maximization of features selection. Optimal feature subset is worked upon as to improve the classifier performance, principal component Analysis is used to obtain the subset of the features. There is possibility to miss several important features and include irrelevant features in the subset during the process. This process selected those features which had highest eigen values (most significant) and ignored those features which had lower eigen values. A method of features selection in intrusion detection of wireless sensor network proposed based on PSO which selects optimal subset of features from the principal space or the pert space

3. INTRUSION DETECTION USING DEEP LEARNING

Machine learning is used to build anomaly detection models and there are two approaches. Shallow learning and Deep learning. Shallow learners mostly depend on the features used for creating the prediction model. On the other hand, deep learners have the potential to extract better representations from the raw data to create much better models.

3.1. DEEP LEARNING

Deep learners can learn better because they are composed of multiple hidden layers. At each layer the model can extract a better representation from the features set when composed to shallow listeners who don't have hidden layers [G+18].

A deep network can be thought of as a program in which the functions computed by the lower layered neuron can be thought of as subroutine. These routines are revised many times in the computational of the final program. Whereas, using a shallow network is similar to the writing a program without the ability of calling subroutine. Without this ability, at any place we could otherwise call the subroutines There is need to explicitly write the code for the subroutine. In terms of number lines of codes, the program for shallow network is therefore longer than a deep network. Worse, the execution time is also longer because the computation of subroutine is not properly used [Le15].

Deep learning is a complex version of machine learning with multiple levels of abstractions of data at multiple processing layers. Deep learning can learn intricate structures in the data set through back propagation and indicate how machine changes the internal parameters at each layer. Deep learning exploits many layers of non-linear information processing for supervised or unsupervised feature extracts and transformation [DY13].

Deep neural network is now an integral part of network security because of its ability to work on large volume of data generated on the network today. Deep neural network has the potential to ensure exhaustive and conclusive evaluation of the network.

Deep learning Architectures are made up of multiple

layers of non-linear operation similar to neural networks with many hidden layers.
The different between multi-layer perception and Deep network is their training procedure. Deep learning is a class of machine learning techniques whose classification is conducted by training data with many layers [H+15].
The key difference between machine learning and deep learning is the change in the performance as the scale of the data increases. Deep learning algorithms requires a large amount of data to find the patterns in the network while machine learning requires the less data.
The deep learning algorithm are able to perform self-learning by extracting features from unlabeled data example of such algorithms are the SVM and Auto-Encoder algorithms. Both algorithms can extract important features from unlabeled data. The difference between Autoencoder and Multi-layer Perceptron is that they may have the same number of inputs and outputs, instead of predicting y, encoders try to reconstruct x [L+12].
The weights are initialized properly. This is a great advantage since most of the data in real life are unlabeled and are always with larger volume of features, and the data are always in large volume. Feature selection helps in the elimination of the possibility of incorrect features and noises [N+16].

3.2. SELF-TAUGHT LEARNING

SLT is a deep learning-based technique. Challenges arising while developing an effective and flexible NIDS for unknown attacks are, first, proper feature selection from the network traffic dataset for anomaly detection is difficult. As attack scenarios are continuously changing and evolving. The features selected for one class of attack may not be well for others classes of attacks.
Secondly, unavailability of labeled traffic dataset from the real networks for developing an NIDS. Great efforts are required to produce such a labeled dataset from the raw network traffic traces collected over a period or in the real-time and this serves as the reason behind the second challenge [N+15].
The STL consists of two stages for classification:
First, a good feature representation is learnt from a large collection of unlabeled data, xu termed as Unsupervised Feature Learning (UFL). Secondly, this learnt representation is applied to label data xL and used for classification task.
Sparse auto-encoder used for UFL. A sparse auto encoder is a neural network consist of an input, a hidden and out layer. The input and output layers contain N nodes and the hidden layer contain K nodes.
The target values in the output layer set to the inputs value i.e:

$$\bar{x} = x_i$$

Where, $\bar{x}$ is the target.
Deep learning method based on neural network are now very popular of recent because deep learning algorithm are able to perform self- learning by extracting features from unlabeled data. Auto-encoder is one of such algorithms. The algorithm can extract important features from unlabeled data.. Self-taught of the deep learning is an example of unsupervised learning. Human intervention is eliminated from the process. Extraction of important features are done through self-taught learning This process is good for online intrusion detection system. Semi-supervised layer of back propagation is to finetune the result obtained so as to obtain higher detection rate and the SVM as the classifier.

3.3. DEEP NETWORK

The deep network is presented in Figure 1.

3.3.1. Generative Architecture

A Generative Model is a powerful way of learning any kind of data distribution using unsupervised learning and it has achieved tremendous success in just few years. All types of generative models aim at learning the true data distribution of the training set so as to generate new data points with some variations.
Generative models are associated with supervised learning since their training does not depend on the labels of the data. For classification purposes the models go through a pre-training stage (unsupervised learning). During this process, each of the lower layers are trained separately from the other layers which allows the other layers to be trained in a greedily layer by layer from bottom to up. All other layers are trained after pre-training [H+15].

3.4. RECURRENT NEURAL NETWORK

They are networks with loops in them, allowing information to persist. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for Machine Learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements of Deep Learning in the past few years.
A multi-layer perceptron is built with an input layer, a hidden layer with certain activations and finally we receive an output.

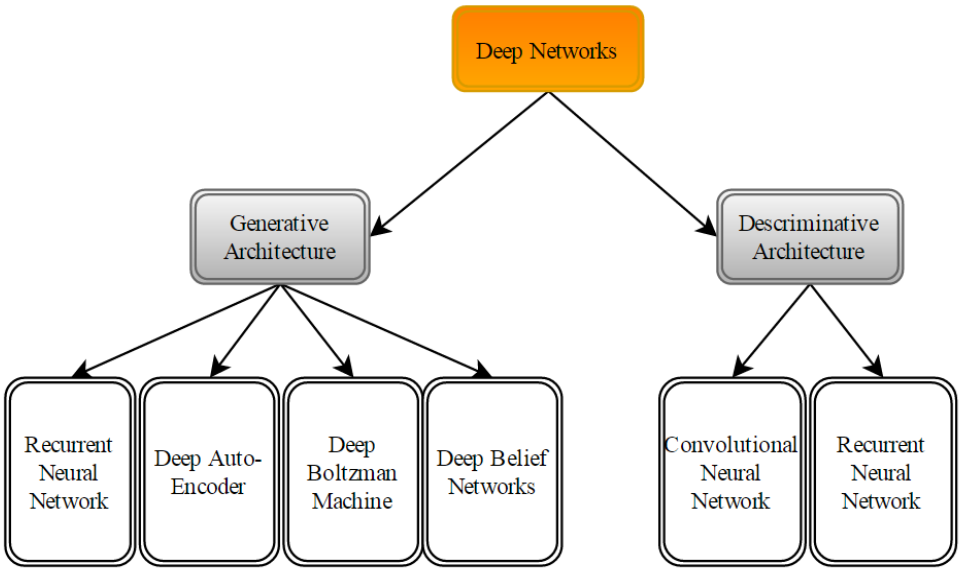


Figure 1. The deep method

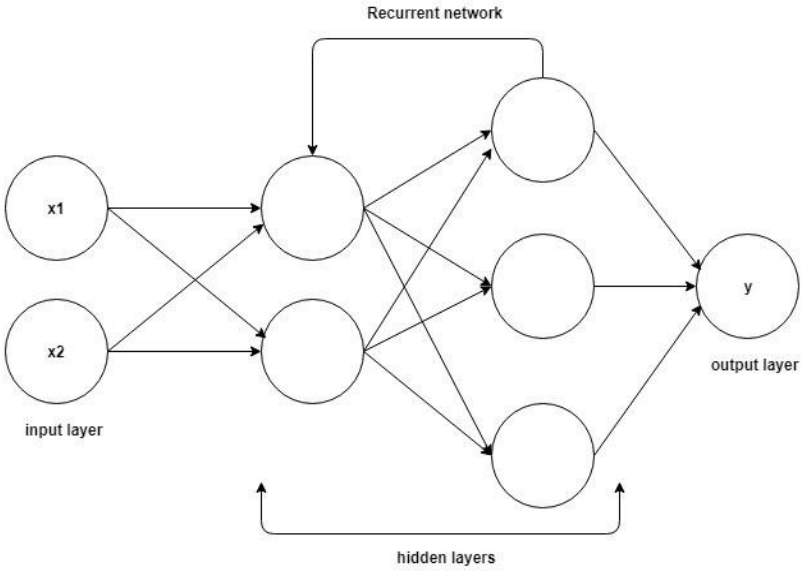


Figure 2. Recurrent Network

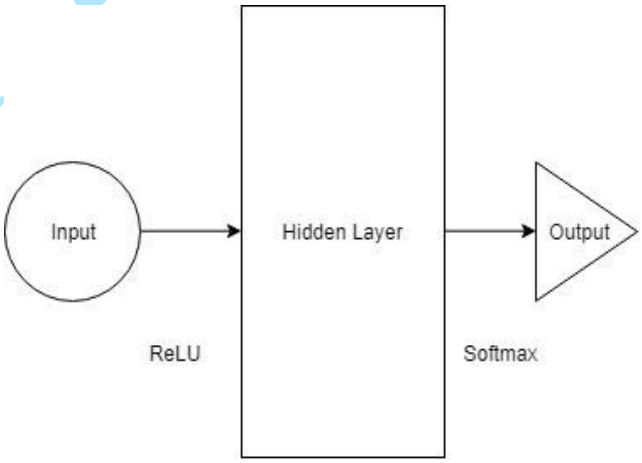


Figure 3. Recurrent Neural Network Architecture

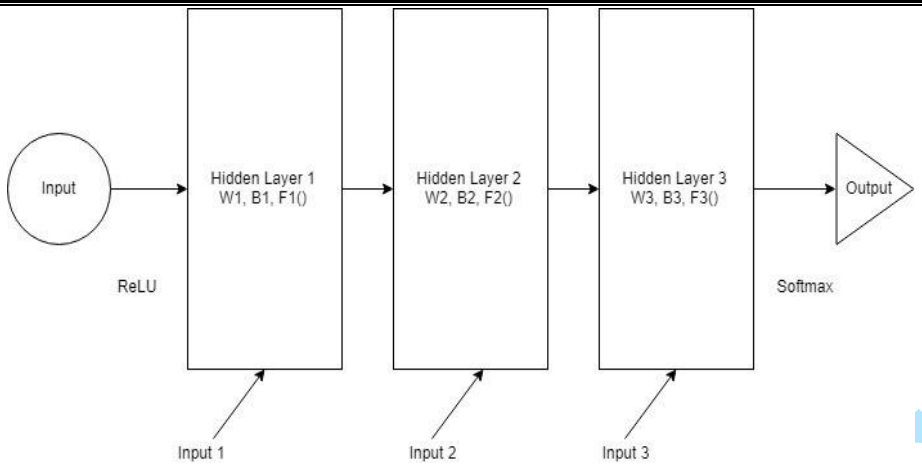


Figure 4. Increasing number of Layers

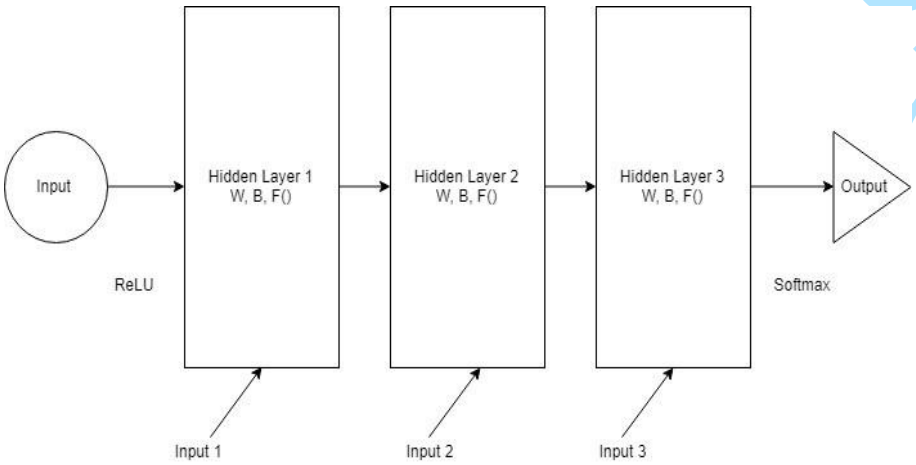


Figure 5. Attaching weights

Increasing the number of layers in the above example, input layer takes the input. Then the first hidden layer does the activation passing onto the next hidden layers and so on. Finally, it reaches the output layer which gives the output. Each hidden layer has its own weights and biases (see Figure 4).

Each layer has its own weight (W), biases (B), Activation Functions (F). These layers behave differently and technically would be challenging to merge together. To be able to merge them, lets replace all the layers with the same weights and biases. It will look something like Figure 5.

Merging all the layers together, the hidden layers can be combined into a single recurrent layer to obtain a structure as Figure 6.

Input is provided to the hidden layer at each step. A recurrent neuron now stores all the previous step input and merges that information with the current step input. Thus, it also captures some information regarding the correlation between current data step and the previous steps. The decision at a time step t-1 affects the decision taken at time t [DEB18].

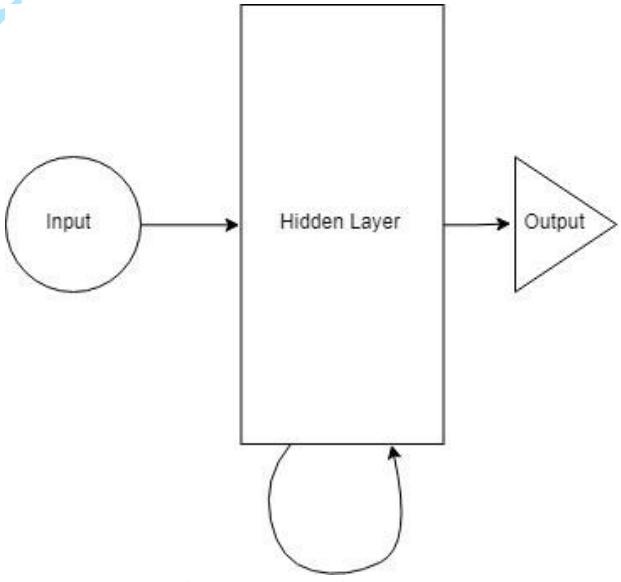


Figure 6. Merging Layers

Deep Auto Encoder

Autoencoder is an unsupervised learning neural net. Autoencoders are deep neural networks used to reproduce the input at the output layer i.e. the number of neurons in the output layer is exactly the same as the number of neurons in the input layer.

3.4.1. Auto-Encoder

This is a very popular technique used in deep learning research. It is an unsupervised neutral network-based feature extraction algorithm, which learns the best parameters required to reconstruct its output as close as to the input as possible. One of the its desirable characteristics is the capability to provide a more powerful and non-linear generalization than the Principle Component Analysis (PCA).
This is done by the use of backpropagation and setting the target value to be equal to the input, trying to learn an approximation to the identity function. An Auto-encoder typically has an input layer, output layer (the same dimension as the input layer) and a hidden layer. The hidden layer normally has a smaller dimension than that of an input (known as undercomplete or sparse auto-encoder).

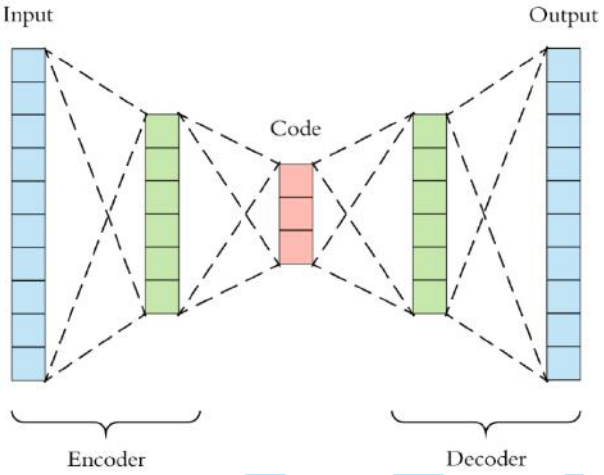


Figure 7. Auto-Encoder

To be able to discover more interesting data structure, use the autoencoder as a non-linear transformation by imposing other constrains on the network, then compare the result with those of the PCA (linear transformation). This method is based on encoder-decoder paradigm.
The input is first transformed into a typically lower dimension space (encoder) and then expanded to reproduce the initial data (decoder). Once a layer is trained, its code is fed to the next, to better model highly non-linear dependence in the input [GBV15].. Auto-encoder are considered an unsupervised learning technique since they don't need explicit labels to train on. But to be more precise they are Self-Supervised because they generate their own labels from the training data.
In self-taught, we first trained a sparse Autoencoder (sparse: to have many of the activation exactly to zero). On unlabeled data, then, given a new example x, we used the hidden layer to extract features a.

3.4.2. Methodologies

The auto-encoder learns features from unlabeled data from the feature learnt from the main data, this trained data can now be trained in Sparse Autoencoder. This training is done with the $W^{(1)}$, $b^{(1)}$, $W^{(2)}$,$b^{(2)}$ as parameters (w=weight , b= base).
If the input is X_u (u for unlabeled), w, b, are used for train *a* to obtain the target x (a better representation of X_{II}).

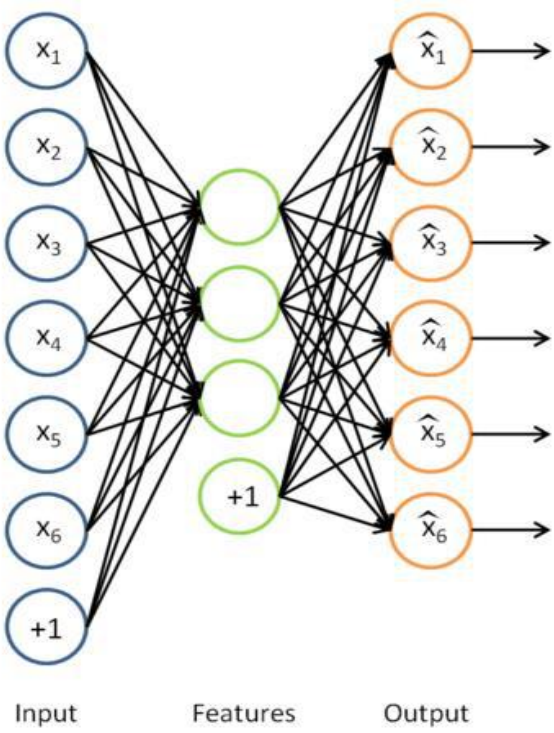


Figure 8. Encoder

After this, we have a labeled training set

$(X_L^{(l)}, y^{(l)}), (X_l^{(2)}, y^{(2)})\ldots\ldots (X_1^{(ml)}, y^{(ml)})$

m_l examples where l stands for labelled.
Instead of representing the first training example $x_l^{(l)}$, let the autoencoder be fed with $x_l^{(l)}$, $a_l^{(l)}$ is obtained as the corresponding vector of the activation.
Representing this example, the original feature can either be either just be replaced with the vector $a_l^{(l)}$ or alternately the two features vectors can be concatenated together, getting $(x_l^{(l)}, a_l^{(l)})$.
The training set is now for representation $(a_l^{(1)}, y^{(1)})$, $(a_l^{(2)}, y^{(2)})\ldots\ldots$ $(a_l^{(ml)}, y^{(ml)})$ for replacement representation and use a_l for l-th training example.
For concatenation representation

$\{(x_l^{(1)}, a_l^{(1)})\ y^{(1)}\}, \{(x_l^{(2)},a_l^{(2)},$
$y^{(2)}\}\ldots\ldots\{(x_l^{(ml)},a_l^{(l)},y^{ml})\}$

Concatenation is better, but for memory or computation representation, replacement is often used.

Training and Testing the Support Vector Machine as Classifier

Support Vector Machine (SVM) is trained as the supervised learning to obtain a function that makes predictions on the y values.

Given a test example x_{test} with above examples, this is fed to auto-encoder as to get a_{test}.

Feeding either a_{test} (as in replacement) or (x_{test}, a_{test}) for the trained classifier SVM to get prediction.

NSL-KDD was proposed to overcome the limitation of KDD Cup dataset.

A sparse autoencoder is a neural network consisting of an input, a hidden, and an output layer. The input and output layers contain N nodes and the hidden layer contains K nodes. The target values in the output layer set to the input values, i.e.,

$$\hat{x}_i = x_i$$

The sparse auto-encoder network finds the optimal values for weight matrices, $W \in \mathbb{R}^{K\times N}$ and $V \in \mathbb{R}^{N\times K}$ and bias vectors, $b_1 \in \mathbb{R}^{K\times 1}$ and $b_2 \in \mathbb{R}^{KN\times 1}$ using back-propagation algorithm while trying to learn the approximation of the identity function, i.e., output $\hat{x}$ similar to x.

Sigmoid function, $g(z) = \frac{1}{1+e^{-z}}$ is used for the activation, $h_{w,b}$ of the nodes in the hidden and output layers:

$$h_{W,b}(x) = g(W_x + b) \quad (1)$$

$$J = \frac{1}{2m}\sum_{i=1}^{m}\|x_i - \hat{x}_i\|^2 + \frac{\lambda}{2}\left(\sum_{k,n} W^2 + \sum_{n,k} V^2 + \sum_k b_1^2 + \sum_n b_2^2\right) + \beta\sum_{j=1}^{K} KL\left(\rho\|\hat{\rho}_j\right) \quad (2)$$

The cost function to be minimized in sparse auto-encoder using back-propagation is represented by Eqn. 2.

The Support Vector Machine is finally used as to fine tune the output of the result of the regression.

3.4.3. Deep Boltzmann Machine

DBM is a unidirectional graphical model. Currently there exist no connection between units on the same layer but between the input units and the hidden units. DBM when trained with a large supply of unlabeled data and fine-tuned with labelled data acts as a good classifier [F+17].

A reduction in the number of hidden layers of a DBM to one form is a Restricted Boltzmann Machine (RBM).

In detecting new attack in real time for anomaly detection, Restricted Boltzmann Machine and Deep Belief Network (DBN), one-hidden layer of RBM was used to perform unsupervised feature reduction. Resultant weight from this RBM are passed to another RBM producing DBN. The pre-trained weights are passed into a fine-tuning layer consisting of a Logistic Regression (LR) classifier with multi-class Soft-Max.

3.4.4. Deep Belief Network

Deep Belief Network integrates unsupervised learning into its network training. Deep Belief Network can be viewed as a composition of unsupervised Restricted Boltzmann Machines, where each Restricted Boltzmann Machine's hidden layer serves as the visible layer for the next. This leads to a layer-by-layer unsupervised training procedure and supervised learning is only applied at the end to fine-tune the network parameters and convert the learned representation into probability predictions (for example, softmax function is used in the last layer to convert the outputs from hidden layers to probability predictions) [F+17]. Therefore, Deep Belief Network has less dependence on initial labels and we expect it to perform better for our problem with limited labels.

3.4.5. Convolutional Neural Network

[D+05] propose an IDS platform based on convolutional neural network (CNN) called IDS-CNN to detect DoS attack. Experimental results show that CNN based DoS detection obtains high accuracy at most 99.87%. Moreover, comparisons with other machine learning techniques including KNN, SVM, and Naïve Bayes demonstrate that the proposed method outperforms traditional ones.

4. CONCLUSION

The advent of big data, the speed at which they come, has great effect on the efficiency of making use of shallow network in the intrusion detection. The noisy data that distorts the performance of the IDS when shallow machine language is made use of is no problem for the deep network. This paper gives an over view of what is happening in the development of Intrusion Detection System with Deep Network in recent time. The use of each deep network mentioned is carefully explained and mention is made of their characteristics.

The elimination of human intervention is of vital importance. The self-taught learning in deep network encoder, using Restricted Boltzmann

Machine, Deep Belief Network and similar algorithm have made intrusion detection performance more efficient.

REFERENCES

[AA18] **Almseidin M., Alkasassbh M.** - *Machine Learning Methods for Network Intrusion Detection*, 2018.

[AJ15] **Asgharzadeh P., Jamal S.** - *A Survey on Intrusion Detection System Based Support Vector Machine Algorithm*, International Journal of Research in Computer Application and Robotics, Vol. 3, Issue 12, Pp. 42-50, www.ijrcar.com, 2015.

[Deb18] **Debarko D.** - *What is a Recurrent Neural Network or RNN, how it works*, https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobsa9afbb00e860, 2018.

[DY13] **Deng L., Yu D.** - *Deep Learning: Methods and Applications*. Foundations and Trends in Signal Processing, vol. 7, nos. 3–4, pp. 197–387, 2013.

[D+05] **Dongseong K., Hanam N., Syngyup O., Jongsou P.** - *Fusions of GA and SVM for Anomaly Detection in Intrusion Detection System*, Advances in Neural Networks - ISNN 2005, Second International Symposium on Neural Networks, Chongqing, China, May 30 - June 1, 2005, Proceedings, Part III, 2005.

[EP14] **Enache A., Patriciu V. V.** - *Intrusions detection based on Support Vector Machine optimized with swarm intelligence*, 9th International Symposium on Applied Computational Intelligence and Informatics, 2014.

[E+10] **Eid H. F., Darwish A., Hassanien A. E., Abraham A.** - *Principle Components Analysis and Support Vector Machinebased Intrusion Detection System*, 10th International Conference on Intelligent Systems Design and Applications, Cairo, November 29, 2010-December 1, 2010, pp. 363-367.

[FA12] **Farhaou Y., Asimi A.** - *Creating a Complete Model of an Intrusion Detection System Effective on the LAN*, International Journal of Advanced Computer Science and Application, Vol. 3, 2012.

[F+17] **Feng W., Wu S., Li X., Kunkle K.** - *A Deep Belief Network Based Machine Learning System for Risky Host Detection*, arXiv.org > cs > arXiv:1801.00025, 2017.

[GBC16] **Goodfellow Y., Bensio Y., Courville** - *Deep Learning*. MIT Press; https//www.deeplearningbooks.org, 2016.

[G+18] **Green C., Lee B., Amarex S., Wengals D.** - *Comparative Study of Deep Learning Models for Network Intruder Detection*, SMU. Data Science Review Vol.1, No.1 Art g, 2018.

[H+15] **Hodo E., Bellekens X., Hamilton A., Tachtatzis C., Atkinson R.** - *Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey*, 2015.

[Ift15] **Iftkhar A.** - *Feature Selection using Particle Swarm Optimization in Intrusion Detection*, International Journal of Distributed Sensor Networks 2015(2), 1-8, 2015.

[JR13] **Jha J., Ragha** L. - *Intrusion Detection System using Support Vector Machine*, International Conference & Workshop on Advanced Computing (ICWAC 2013), www.ijais.org, 2013.

[Le15] **Le Q. V.** - *A Tutorial on Deep Learning. Part1: Nonlinear Classifiers and The Backpropagation Algorithm*. Available on: http://ai.stanford.edu/~quocle/tutorial1.pdf, 2015.

[L+12] **Le Q. V., Ranzato M. A., Monga R., Devin M., Chen K., Corrado G. S., Dean J., Ng A. Y.** - *Building high level features using large scale unsupervised learning*, Proceedings of the 29th International Conference on Machine Learning, Edinburgh, Scotland, UK, 2012.

[MW14] **Manekar V., Waghmare K.** - *Intrusion Detection System using support vector machine (SVM) and particle Swarm Organization, PSO*, International Journal of Advanced Computer Research, Vol. 4, Number 3, Issue 16, 2014.

[N+15] **Niyaz Q., Sun W., Javaid A. Y., Alam M.** - *A deep learning approach for Network Intrusion Detection System*, BICT 2015, December 03-05, New York City, United States, 2015.

[N+16] **Nadeem M., Marshal O., Singh S., Fangand X., Yuan X.** - *Semi-supervised Deep Neural Network for Network Intrusion Detection*, KSU Conference on Cybersecurity Education, Research and Practice, 2016.

[PB13] **Patel K. K. Buddhadev B. V.** - *An Architecture of Hybrid Intrusion Detection System.* International Journal of Information & Network Security (IJINS), vol. 2, no.2, pp. 197-202, 2013.

[QPM14] **Qassim Q., Palid A., Mohozun A.** - *Strategy to reduce False alarms in Intrusion Detection and Prevention System*, International Arab Journal of Information Technology, Vol. II, Issue 5, 2014.

[S+11] **Salama M. A., Eid H. F., Ramadan R. A.., Darwish A., Hassanien A. E.** - *Deep Belief for Clustering and classification of continuous data*, International Journal of Advanced Research in Computer Science, Vol. 4, No. 6, 978-1-4244-9991-5/11 IEEE, 2011.

[S+17] **Shone N., Nguyen Ngoc T., Phai V. D., Shi Q.** - *A Deep Learning Approach to Network Intrusion Detection*, IEEE Transactions On Emerging Topics In Computational Intelligence, vol. 2, issue 1, 2017.

[U+12] **Ugo F., Palmieri F., Castiglione A., De Santi A.** - *Network anomaly detection with restricts Boltzmann Machine*, www.relsevier.com/locate/neucom, 2012.

[Van17] **Vani R.** - *Toward Efficient Intrusion Detect using deep learning technique: A Review*, International Journal of Advance Research in Computer and Communication Engineering 180. 3297, Vol.6 Issue 10, 2017.

[WJ17] **Wang L., John R.** - *Big Data Annalistic for N.I. Detection: A Survey*. International Journal of Networks and Communication, P-ISSN: 2168-4936, E-ISSN: 2168-4944 7(I): 24-31, 2017.

[WP15] **Wiharto A. A., Permana U.** - *Improvement of performance intrusion Detection System (TDS) Using Artificial Neuval Network Ensemble*, Journal of Theoretical and Applied Information Technology, Vol. 80, No. 2, 2015.

A Deep Auto-Encoder based LightGBM Approach for Network Intrusion Detection System

Kun Mo[1], Jian Li[1]
[1]*School of Computer Science, BeiJing University of Posts and Telecommunications, Beijing, China*

Keywords: Intrusion detection systems, Deep auto-encoder, LightGBM, Cyber security, Multi-classification algorithm.

Abstract: With the development of the network in recent years, cyber security has become one of the most challenging aspects of modern society. Machine learning is one of extensively used techniques in Intrusion Detection System, which has achieved comparable performance. To extract more important features, this paper proposes an efficient model based Auto-Encoder and LightGBM to classify network traffic. KDD99 dataset from Lee and Stolfo (2000), as the benchmark dataset, is used for computing the performance and analyse the metrics of the method. Based on Auto-Encoder, we extract more important features, and then mix them with existing features to improve the effectiveness of the LightGBM (Ke et al., 2017) model. The experimental results show that the proposed algorithm produces the best performance in terms of overall accuracy.

1 INTRODUCTION

Network intrusion detection system takes an important role in cyber security. But with the development of internet, various forms of network attacks are emerging one after another. In the past few years, network criminals have been constantly renovating the attack means, in addition to the use of zero-day vulnerabilities, malicious mining extortion software has been rampant; DDoS attack has made a breakthrough in TB level, and the attack channel is increasingly changing; 53% medium-sized companies have suffered security attacks; industrial network has become the focus of illegal hacker attack. So the whole industry network security environment brings new challenges.

The Intrusion Detection System (IDS) is a system which can detect network traffic for suspicious activity (Bai & Kobayashi, 2003). IDS has made great progress in past 20 years that various fine-designed machine learning algorithms have been proposed to model network traffic and applied on commercial applications, such as SVM, XGBoost, LightGBM and CNN. For example, Aastha Puri and Sharma (2017) used a novel algorithm named SVM-CART to combine its output from SVM and CART. Currently, existing IDS algorithm techniques can fall into two categories. The first category is called traditional machine learning model which is easy to train with better interpretability. Based on the characteristics of KDD99 dataset and the existing experimental results, traditional machine learning is extremely suitable for IDS to detect malicious behavior. The second category is deep learning method, which tends to excavate potential time or space structure characteristics. Because of lack of sufficient theoretical support, the second category is like a black box. In recent years, deep learning has developed rapidly so that it has many mature application scenarios, e.g., Computer Vision (CV) (Umbaugh, 1997), Natural Language Processing (NLP) (Manning & Schütze, 1999), autopilot (Ribler et al., 1998). They are proven to be capable of extracting high-order features and the correlation between features or adjacent data. Inspired by this, we can apply deep learning techniques to the feature extraction of network traffic data. This paper aims to combine the advantages of the two categories to achieve higher accuracy. The performance was then evaluated on KDD99 dataset and compared with other proposed model.

In this paper, we adopt deep Auto-Encoder and LightGBM algorithms to construct model. There are two phases in this model. In the first phase, we applied the clean dataset into deep Auto-Encoder network. We only save the intermediate results as the part of input data of the second phase. In the second phase, we concatenate the previous result and the

Mo, K. and Li, J.
A Deep Auto-Encoder based LightGBM Approach for Network Intrusion Detection System.
DOI: 10.5220/0008098401420147
In *Proceedings of the International Conference on Advances in Computer Technology, Information Science and Communications (CTISC 2019)*, pages 142-147
ISBN: 978-989-758-357-5

clean dataset into new dataset. Then the new dataset is inputted in the LightGBM model. In Section II, we analyse the related work. In Section III, we introduce the detail of our design. In Section IV, we introduce raw data, data pre-processing and evaluation metrics. In Section V, we give the results of the experiment and compare performance of various methods. Finally, in Section VI, we conclude this paper.

2 RELATED WORK

The concept of intrusion detection system from a technical report submitted to the US Air Force by Anderson (1980), which details what is intrusion detection. The core of intrusion detection is to use existing computer technology to analyse and detect network traffic, and then take corresponding measures according to certain rules. After more than 30 years of development, the intrusion detection technology has achieved many exciting results (Aljawarneh, Aldwairi, & Yassein, 2018). The existing mainstream intrusion detection methods are based on different machine learning algorithms and typical neural network algorithms, such as support vector machine (SVM) (Mahmood, 2018), Naive Bayes Multiclass Classifier, DNN, CNN (Nguyen et al., 2018).

One of the earliest work found in literature used SVM with various Kernel functions and regularization parameter C values for the design of the model (Kim & Park, 2003). In its paper, Dong Seong Kim and Jong Sou Park used 10% of the KDD 99 training dataset for training and all test dataset for testing. As expected, the training data was divided into train set and validation set. Instead of k-fold cross validation, they optimized the model by repeatedly sampling training set randomly. It is worth noting that the experimental results are improved a lot by proper feature selection. The experimental results proved that SVM achieved a high accuracy in IDS.

Inspired by SVM model, Sungmoon Cheong proposed new model named SVM-BTA to improve SVM (Cheong, Oh, & Lee, 2004). He produced a novel structure which includes SVM and decision tree. This work built a binary decision tree which each node was a SVM classifier. Besides of this, Sungmoon Cheong produced a modified SOM to convert multi-class tree into binary tree. As expected, this method got took advantage of both the efficient computation of the tree architecture and high accuracy.

Deep learning has become more and more popular since researchers were satisfied with data and computation. Javaid et al. (2016) proposed a 2-level deep learning structure. In the first level, there is a self-taught learning model, or more specifically, a Sparse Auto-Encoder for a more expressive feature representation. Sparse Auto-Encoder can excavate more relationships between features and labels. The second level is a softmax regression classifier. Moreover, Farahnakian and Heikkonen (2018) recently have proposed a deep Auto-Encoder based approach for IDS. There are five Auto-Encoder stacked together in their model. Then they used a supervised learning algorithm to avoid overfitting and local optima. Finally, a softmax classifier is added to get the results.

In this paper, we proposed an deep Auto-Encoder and LightGBM based approach for improving IDS performance. Our main contributions are as follows:

Firstly, an Auto-Encoder model is added to discover efficient feature representations.

Secondly, our model concatenated the intermediate result of the deep Auto-Encoder and the clean dataset into new dataset so that we can avoid the loss of feature transformation and feature reduction. We employed a LightGBM model to classify data.

Finally, the performance of our model is evaluated by KDD-CUP'99 dataset. A series of experiments is conducted to explore the performance of different parameters.

3 DEEP AUTO-ENCODER BASED LIGHTGBM MODEL

3.1 Auto-Encoder

An Auto-Encoder is a deep learning model which uses a backpropagation algorithm to make the output value equal to the input value. It first compresses the input into a latent spatial representation and then reconstructs the output by this characterization. An Auto-Encoder includes two parts.

Encoder: This part compresses the input into a latent representation, which can be represented by the encoding function $h = f(x)$.

Decoder: This part can reconstruct the input from the latent representation, which can be represented by the decoding function $y = g(h)$.

Auto-Encoder is an unsupervised learning algorithm whose structure is consistent with BP neural network, but its objective function is different:

$$y_i = x_i \quad (1)$$

For a given dataset $X = \{x_1, x_2, x_3, \dots, x_n\}$, where there is a constraint $x_i \in R^m$, Auto-Encoder first maps it to a hidden representation h by the function denoted as

$$h = f(x) = sigmoid(W_i \cdot x + b_i), \quad (2)$$

where $W_i \in R^{l_1 \times m}, b_i \in R^{l_1}$, l_1 is the number of hidden units. In this function, 0077e use a sigmoid function as the activation function to get the nonlinear correlation between features:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

In the decoding stage, the model maps the encoding result to a reconstructed feature $y \in R^{l_1 \times o_1}$ as

$$y = g(x) = sigmoid(W_o \cdot h + b_o) \quad (4)$$

In next stage, the parameters are optimized such that the error of this model is minimized. The traditional squared error is used frequently:

$$L(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left\| x_i - y_i \right\|^2 \quad (5)$$

n is the number of training dataset.

Finally, back propagation algorithm is used to optimize parameters.

3.2 LightGBM

SVM, GBDT, XGBoost, etc., are used frequently in IDS, but they will become very slow during training phase. Besides, their efficiency and scalability are worrying when the number of features or data is so large. The reason is that when splitting a node, they need to scan all the data to estimate the splitting gain of all possible segmentation point for each feature to get obtain the maximum information gain. So we chose LightGBM algorithm. LightGBM mainly includes two improved algorithms: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

GOSS is proposed to filter most data and only keep a small amount of data. In GBDT, a negative gradient is used for fitting the residual. A sample with a gradient close to zero indicates that the sample has been training well, and the subsequent decision tree will focus on training samples with larger absolute gradient values. The improvement of the GOSS algorithm is to sort the gradients, then select the samples with larger gradients and randomly extract a part of the samples with smaller gradients, and make up a constant for the small gradients, thus greatly reducing the training data.

EFB algorithm can avoid unnecessary computation for zero feature values by bundling exclusive features. The so-called mutual exclusion means that in the feature space, there is only one non-zero values at the same time among the set of features. This algorithm reduces the optimal bundling problem to a graph colouring problem. Firstly the algorithm constructs a graph with weighted edges, whose value is the number of conflicts between features. Then it sort these weights in the descending order. Due to the interference of noise, EFB algorithm solve the problem by tolerating skirmishes. Finally, it checks each feature to create bundles.

Besides, LightGBM adopts histogram-based algorithm (Lee & Goo, 2018) to speed up the training period.

3.3 The Proposed Model

In this section, we introduce how Deep Auto-Encoder based LightGBM model (DAEL) actually works. DAEL mainly consists of two stages. In the first stage, we learn expressive feature representation from a deep Auto-Encoder model that includes two hidden layers. Secondly, we use the intermediate result of the second hidden layer and the clean data as the input of the LightGBM model for final classification. The data is defined as: $D = \{(x_1, y_1), \dots, (x_m, y_m)\}, x \in R^m$. Figure 1. shows the architecture of the proposed model.

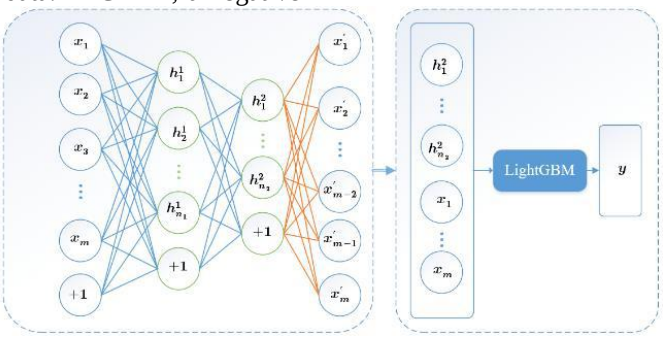


Figure 1: Deep Auto-Encoder based LightGBM architecture.

In the first stage, the deep Auto-Encoder consists of four layers: input layer, two hidden layers, output layer. Unlike other deep neural network, deep Auto-encoder is trained layer by layer. Firstly, the input layer gets input from clean data that includes 117 features. The first hidden layer contain 20 nodes, so we create the first simple Auto-Encoder model whose intermediate result is 20-dimensional vector. Then we use this intermediate result as the input of next hidden layer. In the second hidden layer, 20 features are compressed into 10 features. In this case, 10 features are presented the clean data in the whole deep Auto-encoder architecture.

In the second stage, there is a LightGBM model for the classification task. First of all, the intermediate result of the second layer and the clean dataset is concatenated into new dataset. Then, the new data are inputed into LightGBM model. Due to the imbalance of the data, macro-F1 (Yang, 1999) is adopted as the evaluation standard for training.

4 PERFORMANCE EVALUATION

4.1 Dataset

In this experiment, we evaluate our proposed model on use KDD 99 dataset which is the benchmark dataset in intrusion detection system. This dataset was derived from an intrusion detection assessment project at MIT Lincoln Laboratory (Sabhnani & Serpen, 2003). The dataset contains more than 5 million data, each of which contains 41 features and 1 label. We uses 494,021 samples for training the model and 311,029 samples for evaluating the model. It is common to use 10% of the origin data for training model. In the data, labels can be divided into two categories: Normal and attacks. More specifically, Attacks can be broken down into four categories. The detail of the dataset is as shown in the following Table 1.

Table 1: data distribution.

Label	Training set	Test set
Normal	97278	60593
DOS	391458	229853
R2L	1126	16189
U2R	52	228
Probe	4107	4166

4.2 Data Pre-processing

The KDD 99 dataset includes non-numerical features and duplicates, so this dataset is preprocessed before being inputted into models.

Firstly, the data are deduplicated and disambiguated. Because these duplicate data cause that the model assign a bigger weight to the more frequent data. We need to ensure that there is only one result for a piece of data, these data are disambiguated. After this work, the training dataset consists of 145585 samples and test dataset consists of 77291 samples.

Then, data transformation is applied to the experiment. The symbolic features (protocol_type, services and flag) are mapped to numeric feature by One-Hot Encoding (Buckman et al., 2018). For example, the feature 'protocol_type' contains three values: tcp, udp and icmp. These values are mapped to (1, 0, 0), (0, 1, 0) and (0, 0, 1) in turn. However, the label is mapped to numeric feature by Label-Encoding (Zhang et al., 2018). As we have seen in Table 1, the 'label' field contains five values, which are mapped to 0, 1, 2, 3 and 4 from top to bottom.

Since Auto-Encoder is used in the framework, it is necessary to standardize the data to eliminate differences caused by the different value scales between features. In this experiment, z-score method is adopted. The standard score of a raw score is calculated as

$$z = \frac{x - \mu}{\sigma} \quad (6)$$

where μ is the mean of population and σ is the standard deviation of the population.

4.3 Evaluation Metrics

We evaluate the performance of the proposed model based on the following metrics: Accuracy, macro-F1.

For the binary classification problems, it can be divided into the following four cases according to the real categories of data and the predicted results of the classifier:

- TP (True Positive): The positive class is predicted to be positive class.
- FP (False Positive): The negative class is predicted to be positive class.
- FN (False Negative): The positive class is predicted to be negative class.
- TN (True Negative): The negative class is predicted to be negative class.

Precision: Proportion of samples with positive correct predictions for all samples with positive predictions:

$$P = \frac{TP}{TP + FP} \times 100\% \quad (7)$$

Recall: Proportion of samples with positive correct predictions for all positive samples:

$$R = \frac{TP}{TP + FN} \times 100\% \quad (8)$$

F1-measure: Harmonic mean of Precision and Recall:

$$F1 = \frac{2 \times P \times R}{P + R} \quad (9)$$

In the multi-classification problem, it is assumed that there are k real categories, and the formula for calculating macro F1 rate is

$$F1_{Macro} = \frac{1}{\mathrm{n}} \sum_{i=1}^{k} F1_i \quad (10)$$

5 EXPERIMENTAL RESULTS AND ANALYSIS

This experimental environment for the CPU: Razen1600X, GPU: GTX1070Ti, 16 g memory, operating system for Ubuntu. In order to prove the validity of LightGBM model, we compare it with some existing mainstream models. Table 2 shows the training time and accuracy for multi-classification.

Table 2: mainstream models comparison.

model	Training Time (s)	Acc (%)
LightGBM	422	94.7
XGBoost	4395	92.5
SVM	-	87.7
CNN	-	91.3
LR	79	86.3
RF	159	90.5

where '-' means these models are extremely slow to train.

According to the experimental results, the LightGBM algorithm has the best comprehensive performance. While ensuring high training efficiency, the accuracy of the model far exceeds the above other classifiers. Based on the characteristics of data imbalance, macro F1 is used as the evaluation standard in the training phase of this model. LightGBM training process F1 Score is shown in Figure 2:

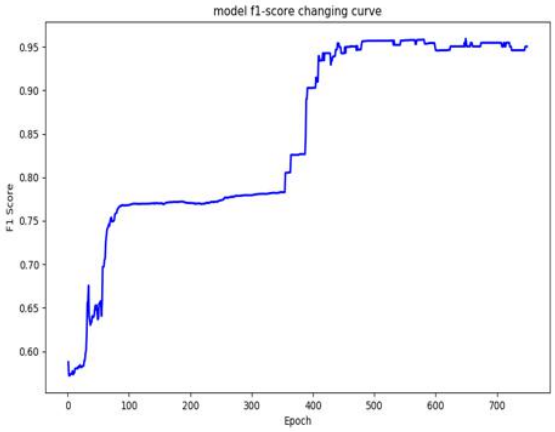


Figure 2: macro F1 changing curve.

As we have seen in Figure 2, LightGBM is hard to optimize after 500 epochs.

In order to prove the effectiveness of our DAEL model on IDS, we compared our algorithm with the previous implemented work. The Table 3 shows that our proposed model get better accuracy and Macro F1 score.

Tbale 3: model improvement.

Models	Acc (%)	Macro F1 (%)
LightGBM	94.7	95.6
DAE	94.2	93.5
DAEL	95.3	96.2

6 CONCLUSION

In this paper, we presented a deep Auto-Encoder based LightGBM approach for improving the intrusion detection system. A deep Auto-Encoder is used for digging some key features. Then LightGBM model is used to automatic feature selection and classify these data. The encoder idea is one of the most useful in the field deep learning and the LightGBM approach is widely used in various practical problems, so we can take full advantage of deep learning and traditional machine learning. KDD 99 dataset is used for evaluating the performance of our proposed model. The experimental result showed that our proposed method achieved accuracy 95.3% on the test dataset.

In future, we will further explore more deep learning methods to represent correlations between features and labels. Additionally, we will further analyse how to evaluate the performance of intrusion detection system more effectively.

REFERENCES

Aljawarneh, S., Aldwairi, M. & Yassein, M. B., 2018. 'Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model'. *Journal of Computational Science*. 25, 152-160.

Anderson, J. P., 1980. 'Computer security threat monitoring and surveillance'. *Technical Report*, James P. Anderson Company.

Bai, Y. & Kobayashi, H., 2003. 'Intrusion detection systems: technology and development'. In *International Conference on Advanced Information Networking & Applications*. IEEE, 710-715.

Buckman, J., et al., 2018. 'Thermometer encoding: One hot way to resist adversarial examples'. In *ICLR*.

Cheong, S., Oh, S. H. & Lee, S. Y., 2004. 'Support vector machines with binary tree architecture for multi-class classification'. *Neural Information Processing-Letters and Reviews*, 2(3), 47-51.

Farahnakian, F. & Heikkonen, J., 2018. 'A deep auto-encoder based approach for intrusion detection system'. In *20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 178-183.

Javaid, A., et al., 2016. 'A deep learning approach for network intrusion detection system'. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, 21-26.

Ke, G., et al., 2017. 'Lightgbm: A highly efficient gradient boosting decision tree'. In *Advances in Neural Information Processing Systems*. 3146-3154.

Kim, D. S. & Park, J. S., 2003. 'Network-based intrusion detection with support vector machines. International Conference on Information Networking'. In *International Conference on Information Networking*. Springer, Berlin, Heidelberg.747-756.

Lee, K. B. & Goo, H. W., 2018. 'Quantitative image quality and histogram-based evaluations of an iterative reconstruction algorithm at low-to-ultralow radiation dose levels: a phantom study in chest CT'. *Korean journal of radiology*, 19(1), 119-129.

Lee, W. & Stolfo, S. J., 2000. *A framework for constructing features and models for intrusion detection systems*. 3, 227-261.

Mahmood, H. A., 2018. 'Network Intrusion Detection System (NIDS) in Cloud Environment based on Hidden Naïve Bayes Multiclass Classifier'. *Al-Mustansiriyah Journal of Science*, 28(2), 134-142.

Manning, C. D. & Schütze, H., 1999. *Foundations of statistical natural language processing*, MIT press.

Nguyen, S. N., et al., 2018. 'Design and implementation of intrusion detection system using convolutional neural network for DoS detection. Proceedings of the 2nd International Conference on Machine Learning and Soft Computing'. In *Proceedings of the 2nd International Conference on Machine Learning and Soft Computing*. ACM, 34-38.

Puri, A. & Sharma, N., 2017. 'A novel technique for intrusion detection system for network security using hybrid svm-cart'. *International Journal of Engineering Development and Research*. Vol. 5.

Ribler, R. L., et al., 1998. 'Autopilot: Adaptive control of distributed applications'. In *IEEE International Symposium on High Performance Distributed Computing*. IEEE.

Sabhnani, M. & Serpen, G., 2003. 'Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context'. *MLMTA*, 209-215.

Umbaugh, S. E., 1997. C*omputer vision and image processing: a practical approach using cviptools with cdrom*, Prentice Hall PTR.

Yang, Y., 1999. 'An evaluation of statistical approaches to text categorization'. *Information retrieval*, 1(1-2), 69-90.

Zhang, Q., et al., 2018. *Category coding with neural network application*. arXiv preprint arXiv:1805.07927.

Deep Learning Approaches for Network Intrusion Detection

Gabriel Fernandez

DEEP LEARNING APPROACHES FOR NETWORK INTRUSION DETECTION

by

GABRIEL C. FERNÁNDEZ, B.S.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Shouhuai Xu, Ph.D., Chair
Greg White, Ph.D.
Wenbo Wu, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2019

DEDICATION

I would like to dedicate this thesis to my wife, for all of her love and support.

ACKNOWLEDGEMENTS

First of all, I would like to thank my wife and family for all the love and support they've provided as I've embarked on my career and postgraduate education.

I would also like to thank my colleagues at USAA for their guidance and mentorship as I've worked through the M.S. Computer Science program, including Chuck Oakes, Dr. Barrington Young, Dr. Michael Gaeta, Maland Mortensen, Debra Casillas, Brad McNary, and many others. In addition, I would like to thank my advisor Dr. Shouhuai Xu, committee members Dr. Greg White and Dr. Wenbo Wu, and my colleagues in the Laboratory for Cybersecurity Dynamics at UTSA for their guidance, fellowship, and collaboration as I worked toward completion of this research, including Dr. Marcus Pendelton, Richard Garcia-LeBron, Jose Mireles, Eric Ficke, and Dr. Sajad Khorsandroo. I would also like to thank all of my professors throughout my master's curriculum at both UTSA and TAMU-CC Departments of Computer Science for their dedication and excellence in teaching. Furthermore, I'd like to thank my colleagues at Texas A&M – Corpus Christi, including Dr. Liza Wisner, Dr. Julie Joffray, Dr. Laura Rosales, and Lori Blades for assisting in my growth and professional development as part of the ELITE team. Lastly, I would like to thank my Uncle, Dr. John Fernandez, for his guidance and mentorship during this endeavor.

May 2019

DEEP LEARNING APPROACHES FOR NETWORK INTRUSION DETECTION

Gabriel C. Fernández, M.Sc.
The University of Texas at San Antonio, 2019

Supervising Professor: Shouhuai Xu, Ph.D.

As the scale of cyber attacks and volume of network data increases exponentially, organizations must develop new ways of keeping their networks and data secure from the dynamic nature of evolving threat actors. With more security tools and sensors being deployed within the modern-day enterprise network, the amount of security event and alert data being generated continues to increase, making it more difficult to find the needle in the haystack. Organizations must rely on new techniques to assist and augment human analysts when dealing with the monitoring, prevention, detection, and response to cybersecurity events and potential attacks on their networks.

The focus for this Thesis is on classifying network traffic flows as benign or malicious. The contribution of this work is two-fold. First, a feedforward fully connected Deep Neural Network (DNN) is used to train a Network Intrusion Detection System (NIDS) via supervised learning. Second, an autoencoder is used to detect and classify attack traffic via unsupervised learning in the absence of labeled malicious traffic. Deep neural network models are trained using two more recent intrusion detection datasets that overcome limitations of other intrusion detection datasets which have been commonly used in the past. Using these more recent datasets, deep neural networks are shown to be highly effective in performing supervised learning to detect and classify modern-day cyber attacks with a high degree of accuracy, high detection rate, and low false positive rate. In addition, an autoencoder is shown to be effective for anomaly detection.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

The Internet has revolutionized society — with more and more people connecting everyday, it is fast becoming a necessity of daily life and a mainstay for conducting day-to-day business. Continued growth in both network access and speed of network connectivity has facilitated wide-spread adoption by the world at large. While the growth of the Internet continues to enable breakthrough innovations and life-changing benefits to society, it also opens the possibility for adversaries to conduct malicious activity in this digital arena. These adversaries primarily consist of three groups: Nation-states, cybercriminals, and activist groups (e.g. Anonymous). Their motivations include espionage, political and ideological interests, and financial gain [10]. While their motivations may be varied, their aims are the same: leverage the connectivity of society through the Internet to carry out a malicious goal. These goals can vary from theft of intellectual property, denial of service, disruption of business, theft of personally identifiable information (PII) or payment card information (PCI), financial fraud, demanding a ransom (i.e. ransomware), destruction of physical property (e.g. Conflicker Warm), and other nefarious purposes.

Due to the opportunity existing for bad actors to conduct this malicious activity, it is imperative that all cyber infrastructure be secured and protected from misuse. Among the many cyber infrastructure systems that exist (e.g. critical infrastructure, cyber-physical systems, SCADA systems, Internet of Things, etc.), this Thesis focuses on the protection of networks maintained and operated by enterprises, both large and small, from being exploited by bad actors. Often, bad actors seek to gain access to enterprise networks for a variety of reasons including but not limited to theft of intellectual property, access to trade secrets, insider information for illegal stock trading, disruption of business (e.g. Sony attacks in 2016), theft of financial information (e.g. Target breach in 2016).

In order to combat bad actors, a wide array of approaches have been developed in order to stay one step ahead of the adversary. The best overall approach for tackling this problem consists of a defense-in-depth strategy, whereby various security tools, techniques, and mechanisms are employed throughout an organization's ecosystem, both horizontally and vertically at different levels.

It is commonly understood that there is no such thing as 100% security. The aim instead is towards managing risk and reducing the surface area available for attack. Security is an intractable problem, as it is impossible to think of all the possible ways an attacker may break through the defenses. A preferred strategy, as suggested by MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) [85], is to minimize the attack surface, and manage risk by employing a defense-in-depth strategy. Various techniques can be used to reduce surface area vulnerable to attack (e.g., by enforcing access control, multi-factor authentication, network segmentation, and continuous patching) as well as reducing risk by deploying tools at various stages, from the exterior-facing network to the interior network, and down to the individual host-level workstations on the network.

The challenge inherent in information systems and networks from being compromised is the fact that they are built upon complex layers of software. Due to its growing complexity, software often contains vulnerabilities that can be found and exploited by an attacker. Even if the software of a given security tool uses proven algorithms and standards for security, it can still suffer from a bad implementation that leaves a security hole. These risks can be mitigated by putting in place strategies and best practices, such as continuous patching, bug bounty programs, red/blue team exercises, threat hunting, honeypots, honeynets, moving target defense, and vulnerability management programs, to name a few. However, attackers continually try new avenues to compromise defenses by altering their attack strategies and using never before seen techniques. Commonly referred to as a zero-day attack, these types of attacks can be very damaging and frustrating to the defender, as the attacker has developed a new exploit that bypasses the defenses of a given system or software. Therefore, as mentioned previously, the best strategy is to implement a defense-in-depth strategy and expect the inevitable result that with enough time and resources, an attacker will inevitably gain access to the network somewhere along the way. It is paramount that when this occurs, the attack is discovered promptly and quarantined or eliminated before any material harm is done.

One of the most effective ways to protect the confidentiality, integrity, and availability of information and enterprise systems once an attacker has compromised its defenses is to deploy Intru-

sion Detection Systems (IDS). Intrusion Detection Systems are defined by the National Institute of Technology (NIST) as "software or hardware systems that automate the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems" [17]. Intrusion Detection is the art and science of finding attackers that have bypassed preventive defense mechanisms such as firewalls, access control, and other protection mechanisms further up or down the stack. More formally, Intrusion Detection is defined by NIST as the "process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices" [83]. There are two main types of Intrusion Detection Systems: Host-based and Network-based. Host-based intrusion detection systems monitor and control data coming from an individual workstation using tools and techniques such as host-based firewalls, anti-virus/anti-malware agents, data-loss prevention agents, and via monitoring system call trees. Network-based defenses monitor and control network traffic flows via firewalls, anti-virus, proxies, and network intrusion detection techniques. Network Intrusion Detection Systems (NIDSs) are essential security tools that help increase the security posture of a computer network. NIDSs have become necessary, along with firewalls, anti-virus, access control, and other common defense-in-depth strategies towards helping cyber threat operations teams become aware of attacks, security incidents, and potential breaches occurring on their networks. The focus of this research is on advancing NIDSs, by leveraging recent advances in deep learning.

There are two main types of Network Intrusion Detection Systems: signature/misuse based and anomaly based. Signature based systems generate alarms when a known misuse or bad activity occurs. These systems use techniques to measure the difference between input events and signatures of known bad intrusions. If the input event shares patterns of similarity with known bad intrusions, then the systems flags these events as malicious. These systems are effective in finding known bad attacks, and can flag them with a low false positive rate. The downside to these systems is that they are not able to detect novel attacks [35]. Anomaly based systems trigger alarms when observed events are behaving substantially differently from previously defined known good patterns. The

advantage of these systems is that unlike signature based systems, they are able to detect novel and evolving attacks. An anomaly, by definition is anything that deviates from what is considered standard, normal, or expected. Anomalies are rare in occurrence, and deviate from the normal, expected behavior. The goal of an anomaly detection system is to identify any event, or series of events, that fall outside a predefined set of normal behaviors. It is important to note that not all anomalies are necessarily malicious. By definition, anomalies are just deviations from expected normal behavior. Once an event or pattern is deemed to be an anomaly, it can be further labeled as either benign or malicious. Therefore, one of the main challenges in anomaly based systems is the problem of generating a high rate of false positives, as well as a high rate of false negatives.

As described, there are numerous host-based and network-based security tools put in place at different layers to detect attacks — these tools generate security events, which must then be evaluated either systematically or by a human analyst. Often times, these security events are centralized in a Security Incident and Event Monitoring (SIEM) system, where they can be triaged by a team of security analysts. These SIEMs contain a combination of signatures, rules, and anomaly detection modules that correlate the myriad of security events, triggering alerts to be worked by a cybersecurity analyst. A common problem faced by these SIEMs is a high rate of false positives. The sheer number of events, and thus alerts generated can overwhelm a security operations team. This results in "alert fatigue" [69] and ultimately makes it increasingly difficult to triage the alerts. As a result, true positive alerts can become buried in a sea of false positives, resulting in an attacker flying under the radar and not being detected until it is too late and material harm has been done. Thus, an ongoing cyber attack campaign can go undetected, leading to a multitude of negative outcomes.

1.1 Thesis Contribution

This Thesis approaches the challenge of detecting attacks using network intrusion detection in a two-fold manner. First, a fully connected *Deep Neural Network (DNN)* is used to train a NIDS with supervised learning using labeled benign and malicious network traffic data. Newer bench-

mark datasets produced by the Canadian Institute of Cybersecurity from the University of New Brunswick (CIC UNB) are used which are more representative of modern day network traffic and attacks [94,95] and do not have drawbacks of previous datasets commonly used in the field [96,97]. After learning these patterns of malicious and benign by training a fully connected neural network, the system can reliably and effectively detect and classify modern attack traffic with a high degree of accuracy, high rate of recall, and a low rate of false positive rate. This is considered to be a form of pattern-based detection because the system is trained on known good and known bad patterns and taught to detect these patterns in future, unseen network flows.

Second, an alternative deep learning approach, known as *Autoencoder*, is used to detect and classify attack traffic in the case where there isn't any labeled malicious training data. This approach is important because in practice it may be difficult to obtain labeled training data in order to train a supervised deep learning algorithm on malicious and benign traffic. In addition, the nature of the adversary is that they are constantly evolving and attempting new attacks, for which a pattern-based system may not be effective since new attacks may have patterns that are vastly different than what has been seen historically. This second approach is considered an unsupervised anomaly based approach, as the learning algorithm will put the traffic into clusters, whereby anomalous activity (e.g. outliers) will stand out from the normal traffic.

1.2 Thesis Organization

In the following chapters, experiments will be outlined that implement deep learning approaches for network intrusion detection, in an attempt to detect and classify malicious traffic. Chapter 2 reviews the preliminary knowledge in the field of machine learning as well as the subfield of deep learning and how it can be used for network intrusion detection. Then the evaluation metrics are described, and the two datasets that will be used in experiments are introduced. Chapter 3 presents a case study on using a fully connected feedforward neural network to perform a classification task on network traffic flows on the two datasets. Chapter 4 describes another case study which utilizes an Autoencoder to perform anomaly detection. Chapter 5 reviews and compares with related prior

work. This work is concluded in Chapter 6 with a discussion on the use these deep learning approaches to network intrusion detection, a review of the insights gained, and suggestions for future work in this field.

Chapter 2: PRELIMINARIES AND DATASETS

This chapter covers a background on the field of artificial intelligence, machine learning, and deep learning and how it relates to the problem of network intrusion detection. In addition, metrics are reviewed that will be used to evaluate the deep learning algorithms used in this work. Lastly, the datasets that will be used for our experiments are described in detail, as well as how features are setup for the learning algorithm.

2.1 Preliminaries

Deep learning sits nestled within the field of machine learning, and machine learning is a subset of Artificial Intelligence (Figure 2.1). Deep learning is a subfield of machine learning that deals with utilizing neural networks containing a large number of parameters and layers. Therefore, a background on artificial intelligence and machine learning concepts will be reviewed, then a framework for applying deep learning for network intrusion detection will be discussed.

2.1.1 Artificial Intelligence

The field of Artificial Intelligence (AI) was born in the 1950s when computer scientists set out to determine if computers could "think" like humans. Artificial Intelligence is defined by MIT's Marvin Minksy as "the science of making machines do things that would require intelligence if done by men" [18]. Another similar definition for the field of artificial intelligence provided by Chollet in [27] is that it is simply "the effort to automate intellectual tasks normally performed by humans." Therefore, artificial intelligence not only encompasses the subfields of machine learning and deep learning, but also many other approaches for enabling the goal of automating intellectual tasks normally performed by humans. These other approaches, however, do not involve the task of having the computer learn. These other systems rely on rules that are explicitly programmed by humans, and are known as *symbolic* artificial intelligence. These systems perform well for for solving well-defined logical tasks such as playing chess; however, they are ill-equiped to deal with more complex tasks such as image classification and language translation. Thus, a new approach

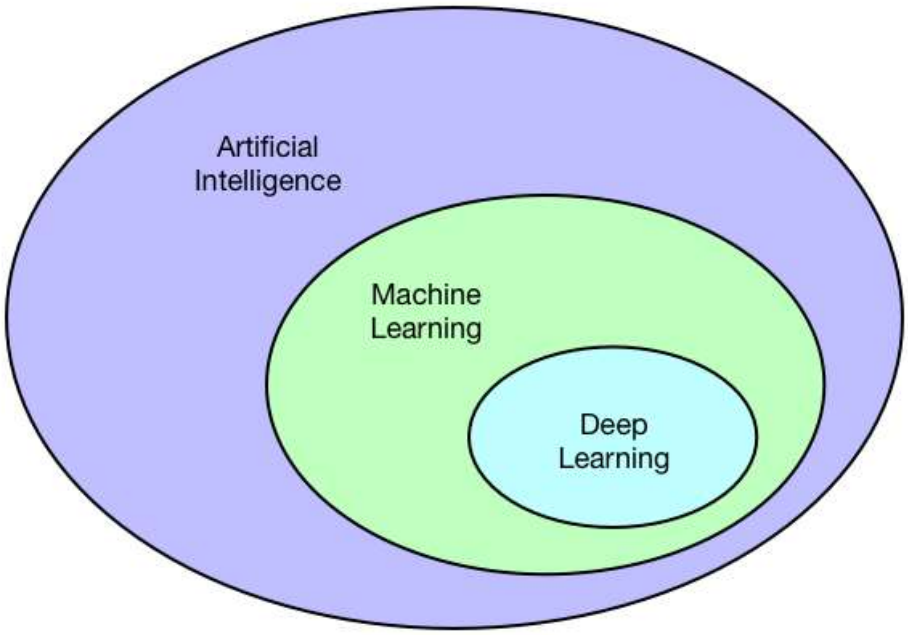


Figure 2.1: Relationship between Artificial Intelligence, Machine Learning, and Deep Learning. (Figure adapted from [27])

to artificial intelligence, termed *machine learning*, gained traction over the previous approaches of symbolic AI.

The field of intrusion detection is another area where existing approaches often rely on rules programmed by humans. While there is a place for these existing intrusion detection systems, and they do work well for enforcing specific parameters and blocking known attack signatures, they are challenged with being able to adapt to new, unseen attacks that do not fall within the strict ruleset defined. A machine learning based system can learn what patterns constitute benign and malicious traffic, and when new traffic comes through, the learning model can determine whether this new traffic looks benign or looks similar to an attack, based on what it has learned about what constitutes benign versus malicious, based on the complex patterns it has learned from the data.

2.1.2 Machine Learning

Alan Turing in his seminal 1950 paper "Computing Machinery and Intelligence" [101] came to the conclusion that general purpose computers could learn and be capable of originality. This opened up questions of whether computers could learn on their own to perform a specific task — can

computers learn rules by looking at data, instead of having humans input the rules manually? These questions gave rise to the subfield of machine learning. Machine learning algorithms are learning algorithms that learn and adjust from data. Instead of manually programming the computer and telling it explicitly what to do, machine learning algorithms enable the program to learn what output to produce, implicitly based on examples and data. By learning based on examples and data, this allows the computer to make decisions and perform tasks on new inputs it has never seen before.

Mitchell [74] defines a learning algorithm as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." In the context of this work, the task is for the learning-based algorithm to classify a network flow as being either benign or malicious. In this case, the individual network flow is an **example** input to the machine learning based algorithm for a classification task. Each example is represented by a set of **features** in certain quantitative or categorical measurements, leading to a vector $\boldsymbol{x} \in \mathbb{R}^n$ where each entry x_i in the vector corresponds to an individual feature (assuming only numeric features are present). The features for describing netflow examples are described in more detail in Section 2.2.

While there are several types of tasks a machine learning algorithm can accomplish, we focus on the task of classification, which has two variants: **binary** and **multiclass** (also called **multinomial**). In either of these two types of classification, the learning algorithm's goal is to determine a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$, where $k = 2$ for binary classification and $k \geq 3$ for multinomial classification. For a function $y = f(\boldsymbol{x})$, the machine learned model assigns to a given input $\boldsymbol{x}$ a numerical value y, representing the output class. When $k = 2$, the output class y implies that the netflow represented by input $\boldsymbol{x}$ is either **benign** or **malicious**; when $k \geq 3$, the output class y implies that the netflow represented by input $\boldsymbol{x}$ is either one among a particular set of attacks, such as Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R), or the network flow is of a benign class. Machine learning algorithms can be placed into two main categories of either **supervised** or **unsupervised**, which will be described in the upcoming section.

2.1.2.1 Supervised Learning

Supervised learning involves the program (system) observing many examples of a given input vector **x** along with an associated output vector **y** containing corresponding labels. The learning algorithm intuits how to predict **y** when given **x**, often by estimating $p(\mathbf{y} \mid \mathbf{x})$. Supervised learning uses an algorithm to learn a function that maps the input to the output, in the form $Y = f(X)$. The goal is to learn the mapping function in such a way that when a new input sample (x) is run through the function (f), it can predict an output (Y) that is correct. This process is referred to as supervised learning because the process can be thought of as a teacher supervising a student. As the 'student' iteratively makes predictions, the 'teacher' supervises and informs the 'student' whether these predictions are correct. The learning algorithm adjusts itself until it reaches an acceptable level of performance.

Fundamentally, supervised machine learning and deep learning are based on this concept of conditional probability following the equation

$$P(E \mid F) \tag{2.1}$$

where, *E* is the label (in this case benign or malicious), and *F* represents the various attributes or features that describe the example or entity for which we are predicting *E*. A common application of conditional probabilities is what is known as **Bayes' Theorem**, which is defined as the formula for any two events, *A* and *B* as:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.2}$$

Where

- **P(A | B)** is the probability of hypothesis A, given an event/data B. This is also referred to as conditional probability.
- **P(B | A)** is the probability of event/data B, given that the hypothesis A was true.

- **P(A)** is the probability of hypothesis A being true (regardless of associated event/data). This is referred to as the prior probability of A
- **P(B)** is the probability of the event/data (regardless of the hypothesis).

Probability is a centerpiece of neural networks and deep learning due to how it enables feature extraction and classification [87].

The foundation of how machine learning works is based on linear algebra and solving systems of linear equations. The most basic form of these equations is:

$$Ax = b \tag{2.3}$$

Data is represented in this equation using scalars, vectors, matrices, and *tensors*. A tensor is the basic data structure for modern machine learning systems. Fundamentally, a tensor is a container for data. A matrix can be described as being a two-dimensional tensor. Simply put, tensors are a generalization of matrices that can be extended to an arbitrary number of dimensions. In the equation above, A is a matrix containing all of the input examples as row vectors with the different features as scalar values, and b is a column vector which has the output labels for each training example, or vector, in the A matrix. The goal in this example is to solve for the coefficient x, in this case a parameter vector, which produces the desired output b. This is accomplished by changing the values in this parameter vector iteratively until the equation generates a desirable outcome as close to the known output b as possible. These parameters are adjusted in a weight matrix iteratively after a loss function calculates the loss between the calculated output, and the actual value, also referred to as *ground truth*. The goal when solving this system of linear equations is to minimize the error, or loss, via *optimization*.

A common method for optimization when solving this system of linear equations (a way in which the algorithm *learns*) is based on the iterative process called *Stochastic Gradient Descent (SGD)*. When performing optimization, the learning algorithm is searching through the hypothesis space for which parameter values (coefficients) map the input to the output with the least amount of

error, or loss. There is a fine balance in this process, as the learning algorithm should not underfit nor overfit the training data. Instead, the parameters should be optimized in a way that the learned function generalizes well to the overall general population of data for the given problem set.

As discussed, there are three primary levers at play within machine learning optimization: (1) *parameterization*, which helps translate input to output for determining probability for classification or regression task; (2) *loss function*, which is used to measure how well the parameters classify (reduce error, minimize loss) at each training step; (3) *optimization function*, which helps adjust the parameters towards a point of minimized error.

Convex optimization is one type of optimization that deals with a convex cost function. In 3-dimensional space, this can be imagined as a sheet that is being held high at each of the four corners, with each corner sloping down to form a cup shape in the bottom. See Figure 2.2 for a visual representation of this cost function. The bottom of the convex shape represents the global minima, or a 0 cost.

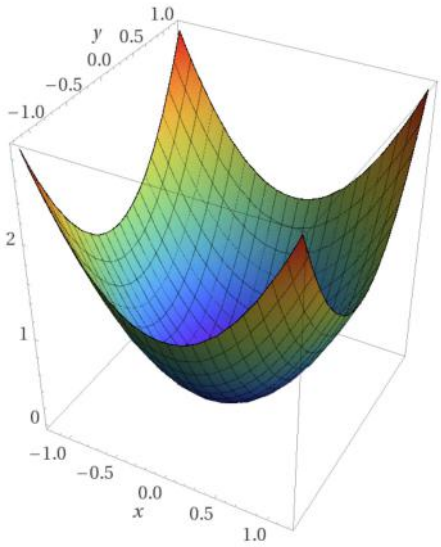


Figure 2.2: Example convex optimization function (computed on WolframlAlpha)

Gradient descent is one optimization algorithm that can determine the slope of the valley (or hill) at a given point based on the weight parameter of the cost function. The gradient descent algorithm then adjusts the weights (parameters) on the function towards reaching a lower cost.

It determines which direction to adjust the weights based on the direction of the slope that it calculates, towards the goal of reaching a zero cost. Gradient descent is able to calculate the slope of the function by taking the derivative of the cost function. The derivative of a function is equivalent to the the slope of the function. For a two dimensional loss function, for example, the derivative (or slope) of the parabolic function $y = 4x^2$ is calculated to be $8x$. The derivative then becomes the tangent of any point on the parabola. On a parabolic function, the slope at any given point is a line tangential to that point. The gradient descent algorithm takes the derivative of the loss function to determine the gradient. This gradient provides the direction of the slope and thus informs the algorithm on how to adjust its weights (parameters) in order to calculate a loss that approaches zero on each successive step.

A variant of gradient descent is **Stochastic Gradient Descent (SGD)**, where the gradient is calculated after each training example is run. This variant is commonly used as it has shown to increase training speed and its computation can also be distributed via parallel computing. An alternative is to perform *mini-batch SGD*, which takes in a small number of training examples for each iteration of loss calculations. This has shown to be more effective than computing the gradient update only one training example at a time.

2.1.2.2 Unsupervised Learning

The main difference between supervised learning and unsupervised learning is that with supervised learning there are given labels or targets corresponding to the input data, and with unsupervised learning the algorithm is given no corresponding labels for its input data. Unsupervised learning is commonly used within the data analytics space and is often used as a means to better understand a dataset prior to using it within a supervised learning algorithm through dimensionality reduction. In the field of exploratory data analysis and data visualization, since humans can only comprehend data that is represented in three dimensions, when a given dataset has 50 or 100 dimensions, it becomes impossible to visualize and make sense of the data in these hyper dimensions. This is why dimensionality reduction is so useful, as it enables humans to visualize high-dimensional data and discover patterns and clusters within the data. Another benefit to dimensionality reduction

is that if the size of the data can be reduced, it can be processed faster. In addition, reducing dimensions also minimizes noise present in the data. When the data is compressed to a smaller number of dimensions, the amount of room available to represent that data is limited, therefore removing unnecessary noise.

As previously described, unsupervised learning techniques can be leveraged in order to help separate the signal from the noise in a given dataset. The hypothesis is that by reducing the dimensions and removing noise from the signal, a supervised deep learning classifier can perform better, as it will mainly be learning from the signal, without additional noise getting in the way. Chapter 4 explores the use of unsupervised deep learning techniques, namely autoencoders, to perform dimensionality reduction and train a neural network to reconstruct its inputs, instead of predicting a class label as in supervised learning. By learning the representation of the input data for normal network flows, a reconstruction error is calculated on never-before-seen test inputs, whereby higher reconstruction errors above a set threshold are flagged as anomalous. In this way, unsupervised deep learning, and specifically autoencoders, can be a powerful engine for an anomaly detection system.

2.1.3 Deep Learning

Deep learning, a subfield of machine learning, excels in generalizing to new examples when the data is complex in nature and contains a high level of dimensionality [45]. In addition, deep learning enables the scalable training of nonlinear models on large datasets [45]. This is important in the domain of network intrusion detection because not only is it dealing with a large amount of data, but the model generated by the deep learning system will need to be capable of generalizing to new forms of attacks not specifically represented in the currently available labeled data. Ideally, the model could generalize and be effective in new, never-before seen network environments, or at a minimum be leveraged in a machine learning pipeline as part of a transfer learning step when used with data from a different computer network environment.

While deep learning has gained popularity in recent years, it has been around for a long time and its origin dates back to the 1940s [45]. Through its history, it has gone by different names such

as “cybernetics” in the 1940-1960 timeframe, and “connectionism” in the 1980-1990s, to what it is known by today as “deep learning” with renewed interested starting back up in 2006. Some of the early algorithms in deep learning were biologically inspired by computational models of the human brain, thereby popularizing algorithms with names such as **artificial neural networks** (ANNs) and by describing computational nodes as *neurons*. While the neuroscientific perspective is considered an important source of inspiration for deep learning, it is no longer the primary basis for the field — there simply does not yet exist a full understanding of the inner workings and algorithms run by the brain. This is an active and ongoing area of research being conducted within the field of “computational neuroscience.” While models of the brain such as the perceptron and neuron have influenced the architecture and direction of deep learning over the years, it is by no means a rigid guide. Modern deep learning instead is based more on the principle of *multiple levels of composition* [45].

One of the catalysts in the resurgence of deep learning in the 2000s was due to a combination of the increase in computational power, along with the increase in available data. Deep learning excels when there exists a large amount of data for which the algorithm can learn from. According to [45], the general rule of thumb as of 2016 is that supervised deep learning algorithms will achieve good performance with at least 5,000 labeled examples per category. They will also exceed human performance when they are trained with a dataset that has at least 10 million labeled examples. In the field of network intrusion detection, the most common benchmark datasets that have been used in the past such as the NSL-KDD ‘99 dataset are smaller in size, containing a total of 148,517 training examples, with 77,054 being benign, and 71,463 being attack. The newer benchmark datasets used in this work such as ISCX IDS 2012 and CIC IDS 2017 are much larger. The ISCX IDS 2012 dataset contains over 2.54M examples, with over 2.47M being benign, and 68,910 being malicious. Similarly, the CIC IDS 2017 dataset contains over 2.83M examples with over 2.27M being benign, and 557,646 being malicious. These larger datasets have many more examples for the neural network to learn from, and therefore can be used to experiment on the effectiveness of using deep neural network architectures for classifying flows as benign or malicious.

While these datasets don't quite have 10M examples, they are much larger than any IDS datasets used in the past, and can be used to experiment and determine the effectiveness of deep learning architectures and algorithms as applied to the domain of network intrusion detection. The amount of data available in practice in an enterprise network is enormous and highly dimensional, often not only containing raw PCAP and/or network flow data, but also including application event logs, host-based logs, security event data, and a myriad of other log data from workstations, servers, sensors, and other appliances spread throughout the network. Furthermore, there exists expert human analysts which can provide ongoing feedback to a learning-based system. This immense amount of data is suited well for utilization by deep learning technologies to help find malicious activity buried within the haystack of network traffic and log data on an enterprise network.

As described earlier, the underlying technology and algorithms in deep learning are based on the utilization of neural network architectures consisting of multiple layers of neurons. In the next section, we will provide some background on neural networks, then describe distinctions inherent within deep neural network architectures.

2.1.3.1 Neural Networks

A neural network, also referred to as an *artificial neural network*, is an information processing system that has certain performance characteristics similar to biological neural networks [37]. It it composed of simple, individual computing *units*, also called *nodes* or *neurons*. Each individual neuron is connected to other neurons by a direct communication link (or *synapse*), and each synapse has its own associated weight. There are different types of neural networks, and they can be categorized by the following [37]:

1. *Architecture*, or pattern by which neurons are connected

2. *Learning algorithm*, or the way in which values for weights on the communication links are determined

3. *Activation function(s)* used by the individual layers of the neural network

Each individual neuron maintains its own internal state, which is determined by the activation function applied to its inputs. The neuron sends its *activation* to all the other neurons to which it is directly connected to downstream in the next layer of the network. Common activation functions for a neuron include the **sigmoid function**, and more recently the **rectified linear unit (ReLU) function**. These two activation functions are shown in Figure 2.7.

An example of a very simple neural network is shown in Figure 2.3. More specifically, this architecture is representative of a single-layer perceptron, which was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt, and funded by the U.S. Office of Naval Research [86]. In this example, the neuron Y is receiving inputs x_1, x_2, and x_3. Each of these three connections between the inputs x_1, x_2, x_3, to Y are represented by weighted variables in this first hidden layer as $w_1^{[1]}$, $w_2^{[1]}$, and $w_3^{[1]}$. Therefore, the input, y_{in}, to neuron Y is calculated as the sum of the three weighted input signals x_1, x_2, and x_3.

$$y_{in} = w_1^{[1]}x_1 + w_2^{[1]}x_2 + w_3^{[1]}x_3 \tag{2.4}$$

This generalizes to n number of inputs as:

$$y_{in} = \sum_{i=1}^{n} x_i w_i^{[l]} \tag{2.5}$$

In this example, it can be supposed that the activation function on the hidden layer neuron will be a sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

Therefore, after neuron Y performs its activation function on the input, it has an output or activation of y_{out}. In this example, neuron Y is connected to output neurons Z_1 and Z_2, each having weights $w_1^{[2]}$ and $w_2^{[2]}$ respectively. Neuron Y then sends the output of its activation function to all neurons in the next layer, in this case neurons Z_1 and Z_2. The values that these two downstream neurons receive as input will vary, as each of the connections to these neurons has a different

weight associated, $w_1^{[2]}$ and $w_2^{[2]}$.

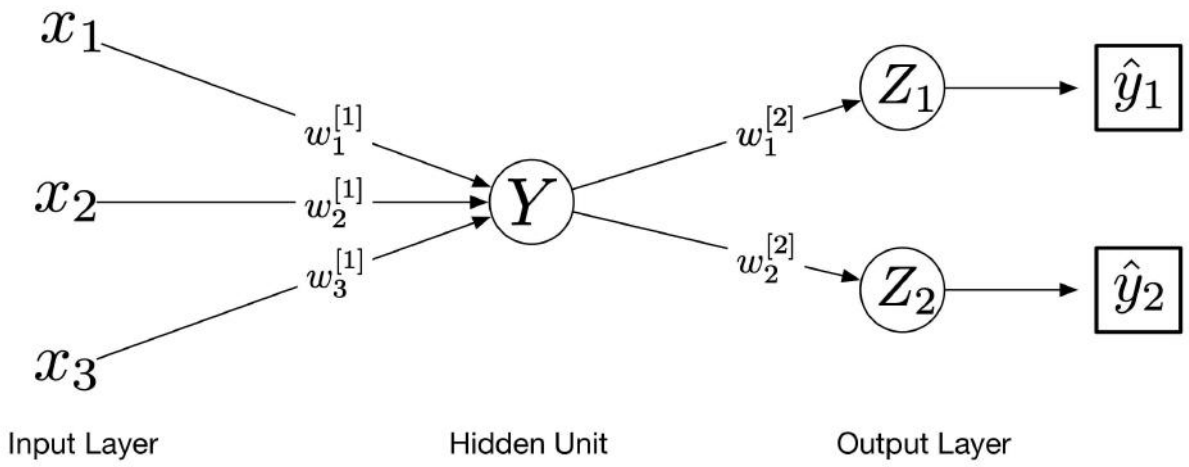


Figure 2.3: Simple neural network

At this last output layer, Z_1 and Z_2 have their own activation function which generates the final output from the network, $\hat{y}_1$ and $\hat{y}_2$ respectively. In the case of a classification problem, $\hat{y}_1$ and $\hat{y}_2$ can each be the respective probabilities that the given input X is either class 0 or 1.

At each training step, the neural network adjusts the weights between its connection in an effort to minimize the loss of the cost function. Updating weights is the primary way in which the neural network learns. The neural adjusts the weights using an algorithm called *backpropagation learning*. The backpropagation learning technique uses gradient descent (described earlier) on the weight values of the neural network connections in order to minimize the error on the output generated by the network.

The underpinning of the backprogagation algorithm is based on the *chain rule* from calculus, which tells us that a chain of functions can be derived using the following identity:

$$f(g(x)) = f'(g(x)) * g'(x) \tag{2.7}$$

Therefore, a neural network can update the weights for each neuron by using the backpropagation algorithm, which starts with the final loss value and working backwards from the top layer down to the bottom layers of the network. At each backward step, the chain rule is applied to deter-

mine the contribution that each individual parameter (at each neuron) had in determining the loss value. Using gradient descent, the weights are updated accordingly, with the goal of optimizing the loss value at the end of the training cycle [86].

2.1.3.2 Deep Neural Networks (DNNs)

Deep neural networks (DNNs), also called deep feedforward networks or feedforward neural networks or multilayer perceptrons (MLPs), are a powerful mechanism for supervised learning. DNNs are one type of deep learning architecture, in addition to recurrent deep neural networks (RNNs) and convolutional deep neural networks (CNNs). This research focuses on the use of DNNs for the task of network intrusion detection. DNNs can represent functions of increasing complexity, by inclusion of more layers and more units per layer in a neural network [45]. In the context of NIDSs, DNNs can be used to discover patterns of benign and malicious traffic hidden within large amounts of structured log data. According to [82], a neural network is considered deep if it contains more than three layers, including input and output layers. Therefore, any network with at least two hidden layers is considered a deep neural network.

An example of standard deep learning representations can be seen in Figures 2.4 and 2.5. The former shows a deep, fully connected neural network, as each of the neurons in the input layer are connected to every other neuron at each successive layer. The latter is a more simplified representation of a two layer fully connected neural network. These figures convey common notation used for representing deep neural networks [81], and will be the notation followed for the rest of this work. Nodes represent inputs, and edges represent weights or biases. The superscript (i) denotes the i^{th} training example, and superscript $[l]$ denotes the l^{th} layer.

The basic technical approach of deep learning for neural networks has been around for decades, so why has this area been gaining so much attention in recent years? The main reason for this is due to an increase in scale of both amounts of data and computational power available. A larger amount of available data, combined with larger neural networks has led to an increase in performance of deep neural network learning algorithms, specifically in the context of supervised learning [81]. This concept can be seen depicted in Figure 2.6.

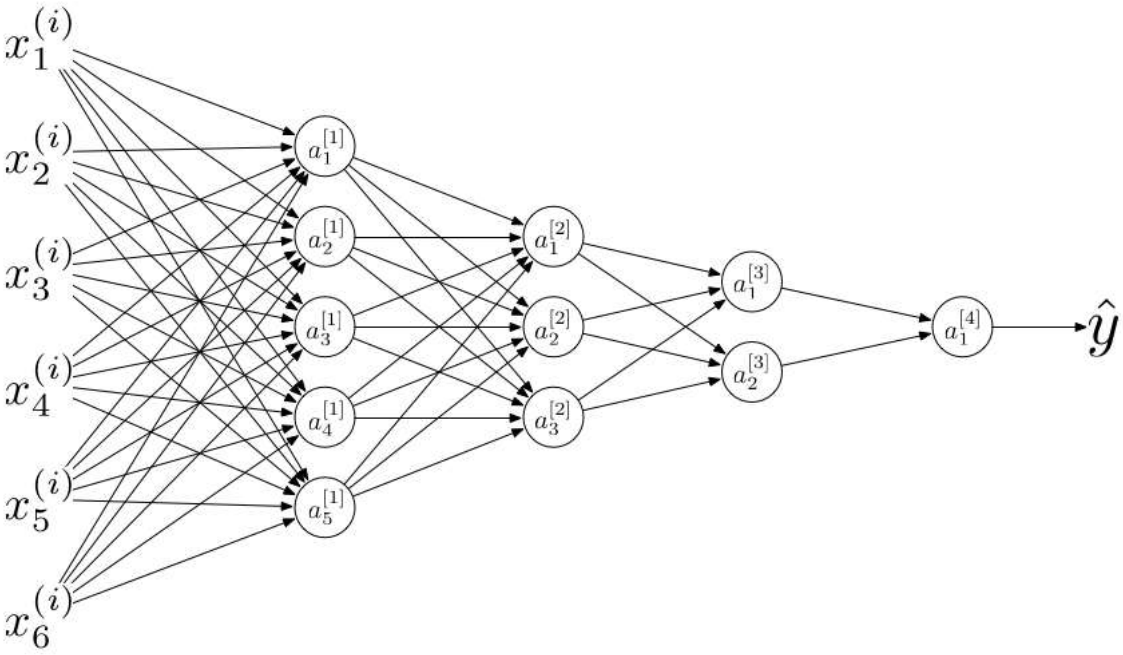


Figure 2.4: Comprehensive neural network representation

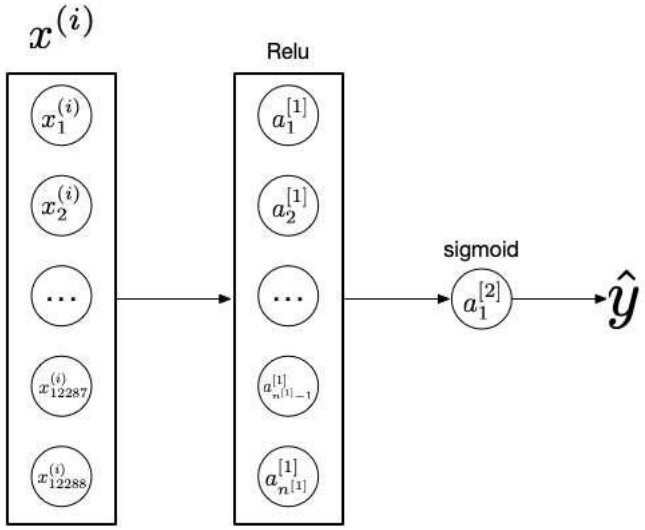


Figure 2.5: Simplified neural network representation

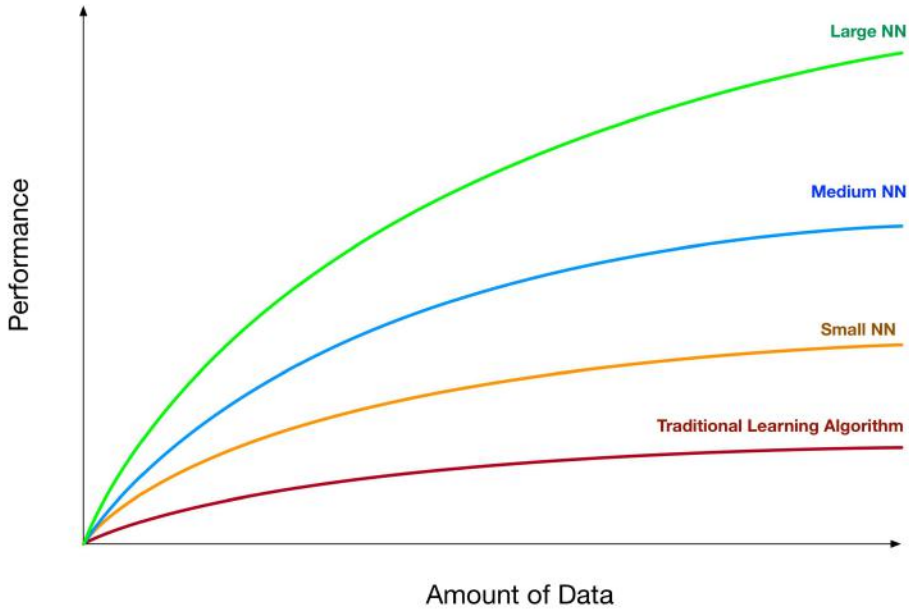


Figure 2.6: Scale drives deep learning performance (Figure adapted from [81])

As described, an improvement in performance can be gained by increasing both the amount of data and the size of the neural network. Once the amount of data is maximized, then the size of the network can continue to be increased until the performance of the neural network levels off. With an increased network size comes increased length of computation times.

Another important factor that has helped deep neural networks become more useful in recent years is due to advances and innovations in algorithms, helping drive more efficient computation and enabling neural networks to run much faster. Previously, the sigmoid activation function (equation 2.6) was most commonly used. One drawback to the sigmoid activation function is that there are regions of the function where the slope (gradient) is nearly zero. This often results in a learning algorithm taking a long time to converge (in minimizing the loss). Later, a new activation function called Rectified Linear Unit (ReLU) became more widely used.

$$R(z) = max(0, z) \tag{2.8}$$

Figure 2.7 pictorially compares these two activation functions.

Innovations such as the preceding one enable machine learning computations to run much

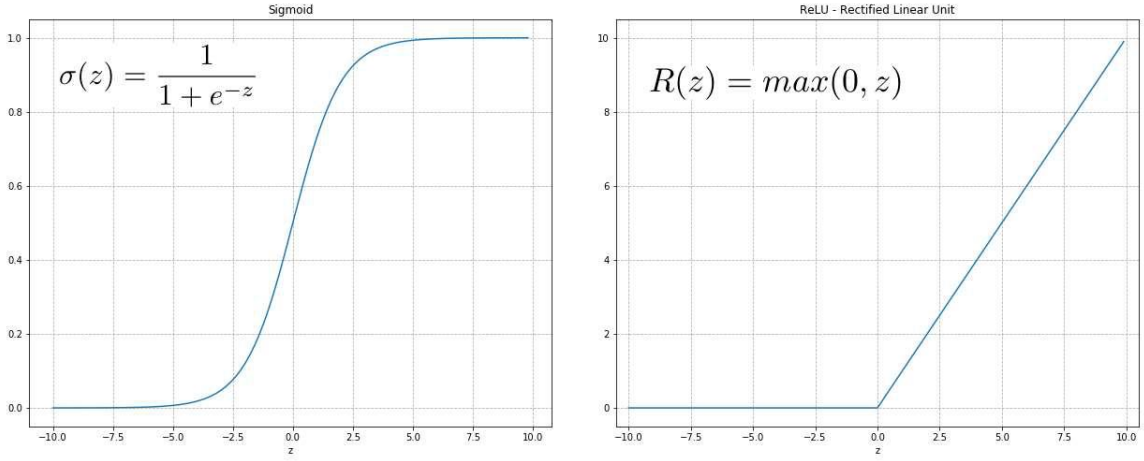


Figure 2.7: Sigmoid vs. ReLU activation functions

faster, allowing for training on much larger networks within a reasonable amount of time. This faster computation is important because the training of neural networks is an iterative process. As illustrated in Figure 2.8, a machine learning practitioner first has an idea for a particular neural network architecture. Subsequently, they implement their idea in code. Finally, they run an experiment to determine how well the neural network performs. Based on the performance of the experiment, the practitioner modifies the architecture, hyperparameters, and/or code, and runs another experiment. This iterative process is run repeatedly until best results are achieved. If each experiment takes an exhorbitant amount of time to run, the productivity of the practitioner is impacted, thus inhibiting their ability to achieve useful results for their machine learning application. Therefore, the ability to iterate quickly with larger neural networks, coupled with larger amounts of data, and new algorithmic innovations has led to higher performance than previously possible.

One of the hallmarks of deep learning is its ability to take complex raw data and create higher-order features automatically in order to make the task of generating a classification or regression output simpler [86]. In the field of cybersecurity and specifically network intrusion detection, the amount of data being generated is continually increasing. Coupled with the continued growth in computational power, deep neural networks can be an effective tool in performing supervised learning for the task of network intrusion detection.

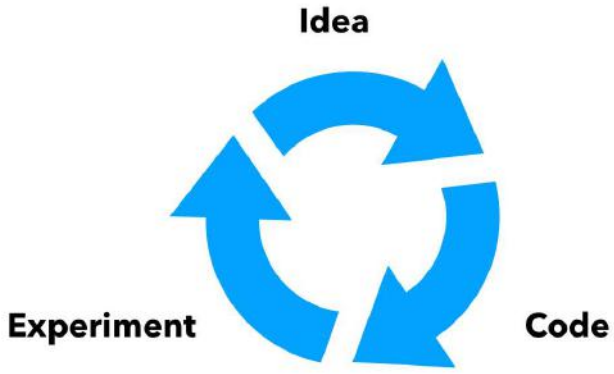


Figure 2.8: Illustration of the iterative process for using Machine Learning in practice (Figure adapted from [81])

2.1.3.3 Unsupervised Deep Learning: Autoencoders and Restricted Boltzmann Machines

Autoencoders and Restricted Boltzmann Machines (RBMs) are two types of neural network architectures that are considered building blocks of larger deep networks [86]. Often, these types of networks are used in a pretraining phase used to extract features and pretrain weight parameters for a follow-on network(s). They are considered unsupervised because they do not use labels (ground-truth) as part of their training. A common use case for using unsupervised pretraining is when there exists a lot of unlabeled data, along with a relatively smaller set of labeled training data [86]. This is a common scenario for enterprise network security use cases, as often times there is a subset of labeled training data that has been reviewed, processed, and labeled by a human analyst, yet there is still an enormous amount of unlabeled data for which there is not enough manpower to review. The downside to having this pretraining step is the extra amount of overhead in terms of network tuning and added training time.

Autoencoders are useful in cases where there are lots of examples of what normal data looks like, yet it is difficult to explain what represents anomalous activity. For this reason, Autoencoders can be powerful when used in anomaly detection systems. Autoencoders applied to network intrusion detection are described in more depth and used in experiments in Chapter 4.

2.1.4 Evaluation Metrics

The primary goal of a classification algorithm in the context of network intrusion detection is to achieve the highest level of accuracy with the lowest number of false positives. In addition, the **True Positive Rate (TPR)** (also referred to as **Detection Rate**, **Recall**, or **Sensitivity**) is an important metric for network intrusion detection as it indicates the number of malicious examples that are correctly identified. A number of common metrics are used to evaluate the effectiveness of the deep learning approaches in this work [24, 88]. The basic terminology will be described first.

- **True Positives** (TP) are the number of samples that are *correctly predicted as positive* (e.g. ground truth is 'malicious' and the prediction is also 'malicious').

- **True Negatives** (TN) are the number of samples that are *correctly predicted as negative* (e.g. ground truth is 'benign' and the prediction is also 'benign').

- **False Positives** (FP) are the number of samples that are negative but predicted as positive (e.g. ground truth is 'benign' and prediction is 'malicious').

- **False Negatives** (FN) are the number of samples that are positive but are predicted as negative (e.g. ground truth is 'malicious' and prediction is 'benign').

The performance of a supervised learning classification algorithm can be depicted via a **confusion matrix**, an example of which is shown in Figure 2.9. The rows indicate the ground truth and the columns indicate predicted class.

In the context of network intrusion detection, metrics can be defined as follows [24, 38, 88].

- **True Positive Rate** (TPR) or **Recall**: $TPR = Recall = \frac{TP}{TP+FN}$.

  Note: True Positive Rate, or Recall, is also referred to by other studies as *Detection Rate*; therefore, results in this work are compared to these other studies where appropriate using the term Detection Rate, interchangeable for True Positive Rate.

- **True Negative Rate** (TNR): $TNR = \frac{TN}{TN+FP}$.

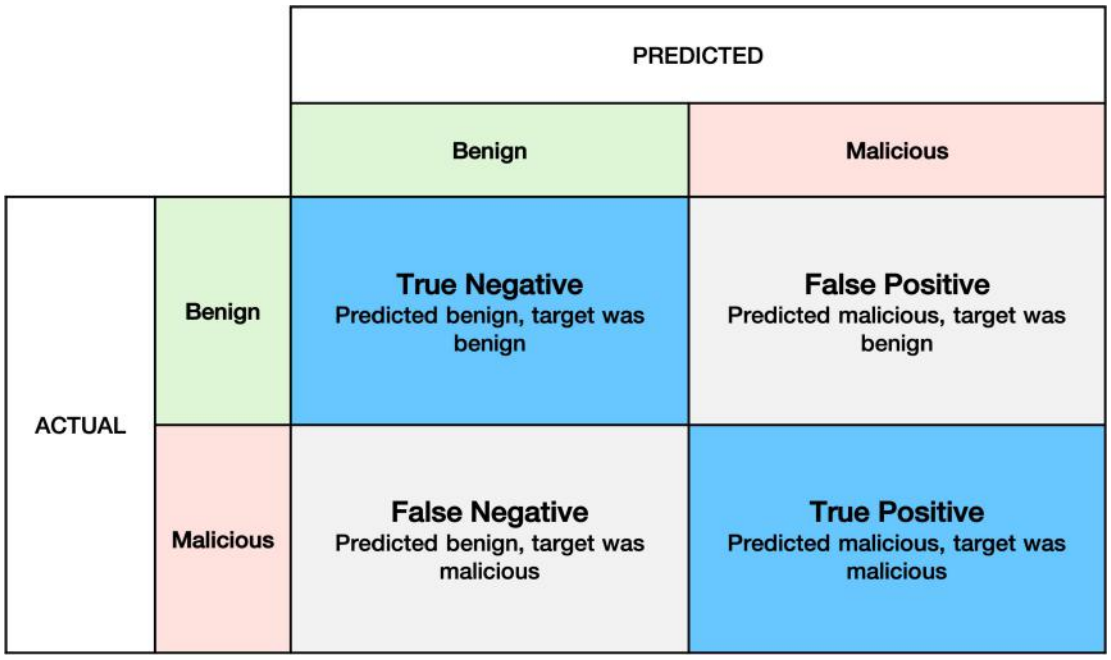


Figure 2.9: Confusion Matrix

- **False Positive Rate** (FPR): $FPR = \frac{FP}{FP+FN}$.
- **False Negative Rate** (TNR): $FNR = \frac{FN}{FN+TP}$.
- **Accuracy** is the ratio of the number of total correct predictions made $TP + TN$ to all predictions made $TP + FP + FN + TN$, namely

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.9}$$

- **Precision**, which is also known as Bayesian Detection Rate or Positive Predictive Value, is the ratio of the total number of correctly predicted positive classes TP to the total number of positive predictions made $TP + FP$, namely

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

- **F1 Score** is the harmonic mean of the Precision and the Recall (i.e., TPR), namely

$$F1\ Score = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{2.11}$$

- The **Receiver Operating Characteristics** (ROC) curve is a plot of True Positive Rate (TPR) on the y-axis against False Positive Rate (FPR) on the x-axis.

As shown above, there are several metrics that can be used for evaluating the performance of a given machine learning classifier. The F1 Score, which combines precision and recall, is a single evaluation metric that is focused on in order to measure and compare the performance and effectiveness of the neural network classifiers used in this work. In addition, the True Positive Rate (Recall, Detection Rate) is another key metric that is focused on in comparing the deep learning approaches in this Thesis to other works in the field.

2.2 Datasets

A primary and ongoing challenge in the field of network intrusion detection is the lack of publicly available, labeled datasets that can be used for effective testing, evaluation, and comparison of techniques [84, 96]. Often times, the most useful datasets for network intrusion detection are those containing captures of real network environments. These datasets are not easily shared with the public, as they contain details of an organization's network topology, and more importantly sensitive information about the traffic activity of the users on the respective network. Furthermore, the effort required to create a labeled dataset from the raw network traces is an immense undertaking.

As a consequence, researchers often resort to suboptimal datasets, or datasets that cannot be shared amongst the research community. Granted, publicly labeled datasets are available, such as CAIDA [51], DARPA/Lincoln Labs packet traces [66, 67], KDD '99 Dataset [1], and Lawrence Berkeley National Laboratory (LBNL) and ICSI Enterprise Tracing Project [14]; however, these datasets are mostly anonymized and do not contain valuable payload information, making them less useful for research purposes [96]. While these datasets have proven useful, there are some

arguments as to the validity of using them in present day research — they may be better suited for the purposes of providing additional validation and cross-checking of a novel technique [97].

This work focuses on using newer benchmark datasets that have recently become available to the research community, specifically the UNB ISCX IDS 2012 and UNB CIC IDS 2017 datasets, which will be described in more detail in the following sections.

2.2.1 UNB ISCX IDS 2012 Dataset

One of the datasets analyzed in this thesis was provided by Lashkari, on behalf of the authors of [96], who with the University of New Brunswick's Information Security Center of Excellence (ISCX) developed a systematic approach to generating benchmark datasets for network intrusion detection. Their approach creates datasets by first statistically modeling a given network environment, and then creating agents that replay that activity on a testbed network. Using this systematic approach, Lashkari et. al. created the UNB ISCX IDS 2012 dataset, which consists of network traffic generated on a testbed environment in their laboratory. The testbed network consists of 21 interconnected Windows workstations. Windows was chosen for the workstations because of the availability of exploits for running attacks. These workstations were divided into four distinct LANs in order to represent a real network configuration. A fifth LAN was setup containing both Linux (Ubuntu 10.04) and Windows (Server 2003) servers for providing web, email, DNS, and Network Address Translation (NAT) services. When compared with the widely used, but more outdated datasets [1, 66, 67], this dataset has the following characteristics [96]: (i) realistic network configuration because of the use of real testbed; (ii) realistic traffic because of the use of real attacks/exploits (to the extent of the specified profiles); (iii) labeled dataset with ground truth of benign and malicious traffic; (iv) total interaction capture of communications; (v) diverse/multiple attack scenarios are involved. The reader is directed to [96] for full details of the testbed configuration.

For generating the dataset, two kinds of profiles, dubbed α-profiles and β-profiles, were used [96]. The α-profiles reflect attacks by specifying attack scenarios in a clear format, easily interpretable and reproducible by a human agent. The β-profiles reflect benign traffic by specifying

statistical distributions or behaviors, represented as procedures with *pre* and *post* conditions (e.g., the number of packets per flow, specific patterns in a payload, protocol packet size distribution, and other encapsulated entity distributions). The β-profiles are represented by a *procedure* in a programming language and executed by an agent, either systematic or human. This profile generation methodology was created in an attempt to resolve issues seen in other network security datasets. The main objective of Shiravi et al. in [96] was to establish a systematic approach for generating a dataset containing background traffic (β-profiles) reflective of benign traffic while being complimentary to malicious traffic generated from executing legitimate attack scenarios (α-profiles). The UNB ISCX IDS 2012 dataset therefore contains properties that make it useful as a benchmark dataset, and resolves issues seen in other intrusion detection datasets.

Table 2.1: ISCX IDS 2012 Dataset Overview

Date	# of Flows	# of Attacks	Description
6/11/2012	474,278	0	Benign network activities
6/12/2012	133,193	2,086	Brute-force against SSH
6/13/2012	275,528	20,358	Infiltrations internally
6/14/2012	171,380	3,776	HTTP DoS attacks
6/15/2012	571,698	37,460	DDoS using IRC bots
6/16/2012	522,263	11	Brute-force against SSH
6/17/2012	397,595	5,219	Brute-force against SSH
Total	**2,545,935**	**68,910**	**2.71% attack traffic**

Table 2.1 gives an overview of the dataset. This dataset contains over 2.5 million flows. The dataset contains labels for both benign and malicious traffic flows, which are described in an XML file. After processing the XML flow records, the number of flows and attacks per day can be seen in Figures 2.10 and 2.11 respectively.

The dataset is made up of captured traffic for a seven day period starting at 00:01:06 on Friday, June 11th, 2012 and running continuously until 00:01:06 on Friday June 18th, 2012, while noting that attack activities occurred on days two through seven only (i.e., no attacks on day one). The dataset contains the following types of attacks [96]:

1. **Brute-force against SSH:** This attack attempts to gain SSH access by running a brute-force

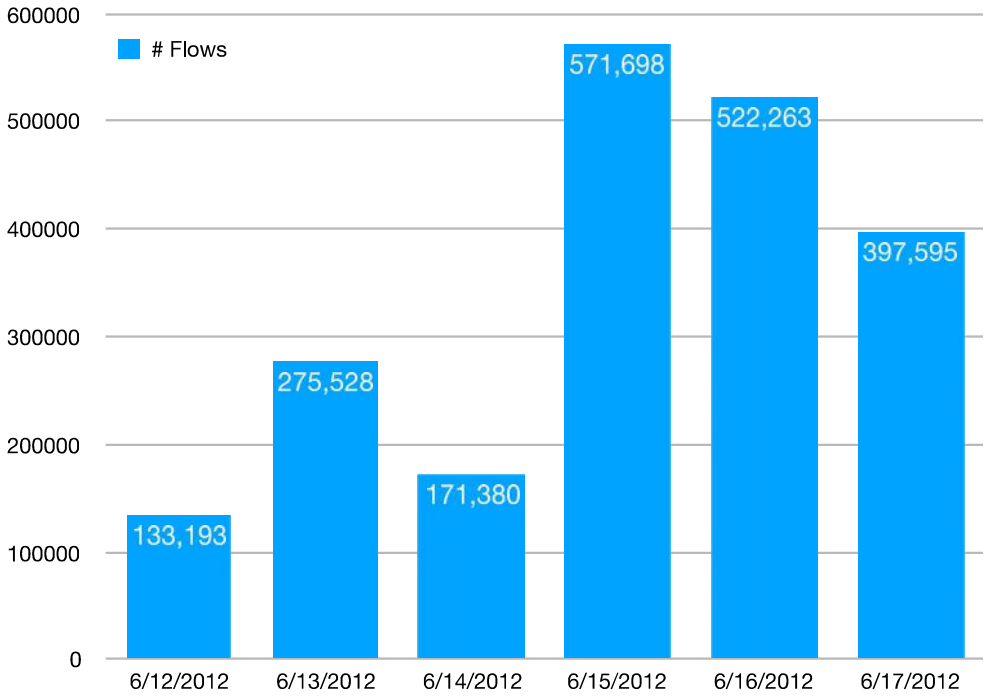


Figure 2.10: ISCX IDS 2012 Dataset: Number of flows per day

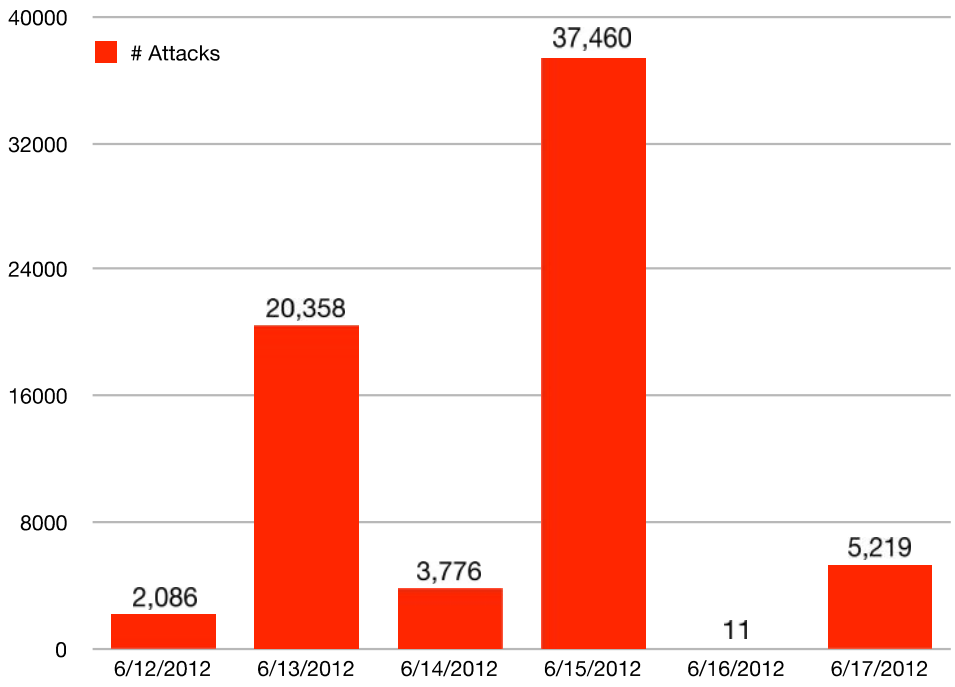


Figure 2.11: ISCX IDS 2012 Dataset: Number of attacks per day

dictionary attack by guessing username/password combinations. The *brutessh* tool [5] was used for this attack, and ran for a period of 30 minutes until successfully obtaining superuser credentials. Those credentials were used to download `/etc/passwd` and `/etc/shadow` files from a server.

2. **Infiltrating the network from the inside:** This attack attempts to gain access to a host on the inside (due to a buffer overflow exploit on a PDF file) and establishes a reverse shell. The attacker pivots from the host machine, scanning other internal servers for vulnerabilities and installing a backdoor.

3. **HTTP denial of service (DoS) attacks:** This is a stealthy, low bandwidth denial of service attack without needing to flood the network. The `Slowloris` tool [4] was used to perform this attack, which holds a TCP connection open with a webserver by sending valid, incomplete HTTP requests, keeping open sockets from closing. Leveraging a backdoor established by the aforementioned attack of *infiltrating the network from the inside*, `Slowloris` was deployed on multiple hosts to perform the attack.

4. **Distributed denial of service (DDoS) using IRC bots:** Leveraging the backdoor established by the aforementioned attack of *infiltrating the network from the inside*, an Internet Relay Chat (IRC) server is installed, and IRC bots are deployed to infected machines on the network. Within a period of 30 minutes, bots installed on seven users' machines connect to the IRC server awaiting commands. These bots are then instructed to download a program for making HTTP GET requests, and are then commanded to flood an Apache Web server with requests for a period of 60 minutes causing a distributed denial of service.

Table 2.2 lists the features for the ISCX IDS 2012 Dataset. Each feature is discrete or continuous, depending on the type of information it contains. A **discrete** feature (also called **categorical**) is one which has a finite or countably finite number of states. Discrete features can be either integers or named states represented as strings that do not have a numerical value. A **continuous** feature is one that can be represented as a real number. In the context of network intrusion detec-

tion, it is important to configure each feature correctly based on domain knowledge. For example, while certain features such as source/destination port appear numerical and potentially continuous in nature, they should be setup as categorical variables since the value '80' corresponds to the HTTP protocol, and not the continuous value of 80. The full description and configuration of features for the ISCX IDS 2012 dataset is shown in Table 2.2.

Table 2.2: Description of Features for ISCX IDS 2012 Dataset

No.	Feature	Description	Type	Unique Values
1	SrcIP	Source IP address	Categorical	2,478
2	DstIP	Destination IP address	Categorical	34,552
3	SrcPort	Source port for TCP and UDP	Categorical	64,482
4	DstPort	Destination port for TCP and UDP	Categorical	24,238
5	AppName	Application name	Categorical	107
6	Direction	Direction of flow	Categorical	4
7	Protocol	IP protocol	Categorical	6
8	Duration	Flow duration in fractional seconds	Continuous	N/A
9	TotalSrcBytes	Total source bytes	Continuous	N/A
10	TotalDstBytes	Total destination bytes	Continuous	N/A
11	TotalBytes	Total bytes	Continuous	N/A
12	TotalSrcPkts	Total source packets	Continuous	N/A
13	TotalDstPkts	Total destination packets	Continuous	N/A
14	TotalPkts	Total packets	Continuous	N/A

Each flow record in the original dataset is represented by these 14 high-level features. Of these 14 high-level features, seven of them are categorical and seven are continuous. Therefore, the actual number of feature columns expands to a maximum of the sum of all the unique categories existent for each categorical feature. For example, the high-level feature of 'SrcIP' contains 2,478 unique source IP addresses, while the 'DstIP' feature consists of 34,552 distinct destination IP addresses. The 'SrcPort' feature contains 64,482 unique values, and 'DstPort' feature contains 24,238 distinct values. As a result, if each categorical feature is expanded out based on the unique possible value, there ends up being a total of $2,478 + 34,552 + 64,482 + 24,238 + 107 + 4 + 6 =$ **125,867** possible features. In the case study section of Chapter 3 a number of experiments are conducted using the maximum number of features, as well as a subset of these expanded features. The features can be reduced by first removing IP address and ports completely from

the dataset, but experiments are also conducted that used a dense vector representation of the features for IP addresses and ports. This methodology and its results are expanded upon in Chapter 3. The 'AppName' feature contains 107 unique values, and consists of values such as 'SSH', 'HTTPWeb', 'IMAP', 'DNS', 'FTP', etc. corresponding to the type of application traffic traversing between source and destination IP/port for a given flow record. The 'Direction' feature contains four unique values, consisting of 'L2L', 'L2R', 'R2L', and 'R2R' which stand for local-to-local, local-to-remote, remote-to-local, and remote-to-remote respectively. The 'IP Protocol' feature consists of six unique values of 'tcp_ip', 'udp_ip', 'icmp_ip', 'ip', 'igmp', 'ipv6icmp', which indicate the type of protocol used for the given flow record. The remaining features in the dataset are continuous and are statistics of the flow record, including total source and destination bytes, as well as total number of source and destination packets that occurred for a given flow. The two features of 'TotalBytes' and 'TotalPackets' are engineered features not present in the original dataset, which are a sum of the source and destination bytes and source and destination packets respectively. In addition, this dataset contains labels of *benign* and *malicious* for each flow record example. The class label *benign* is represented with the numerical value 0, while *malicious* is represented with the numerical value 1.

2.2.2 UNB CIC IDS 2017 Dataset

This second dataset analyzed in this work was also provided by Lashkari, on behalf of the authors of [95]. It comes from a collaboration between the Canadian Institute for Cybersecurity (CIC) and University of New Brunswick's Information Security Center of Excellence (ISCX). The dataset was created in 2017 and published for the research community to use in 2018. In their work, they study and compare eleven available datasets that have been used for the research and development of intrusion detection and intrusion prevention algorithms, including the ISCX IDS 2012 dataset. They outline some of the same points that have been discussed in Section 2.2 in regards to the issue of existing datasets being out of date and not fit for current research and future advancement of the field of network intrusion detection. This dataset is improved from previous datasets in that it contains more recent attack traffic from seven different attack methods, along with benign traffic.

In addition, the published dataset includes not only the raw PCAP data, but also pre-processed netflow data from the PCAP data that was processed using publicly available CICFlowMeter software [55]. This dataset was generated over a period of five days, Monday through Friday. The distribution of benign and malicious flows can be seen in Table 2.3. It turns out that there are a total of $2,830,743$ flows generated over the five days, with a total of $557,646$ of those flows being attack flows. This results in 19.70% of the flows being malicious traffic, which is much larger than the ISCX IDS 2012 dataset which had 2.71% of the flows labeled as malicious traffic.

This pre-processed netflow data is provided as CSV files that can be more easily fed into the machine learning pipeline, as opposed to having to start with the raw PCAP files (or in the case of ISCX IDS 2012, a custom XML file). Furthermore, the pre-processed netflow data has 83 columns (plus one label column and one flow ID column) that can be used as potential features, which is advantageous for evaluating various features within deep learning approaches for NIDSs. As mentioned, there are seven different attack types that are labeled as such, which enables experimentation with a multinomial classifier, as opposed to just a binary classifier in the case of the labeled ISCX IDS 2012 dataset. The other main limitation of the ISCX IDS 2012 dataset is that there are no HTTPS protocols in the dataset, which is an important point since over 70% of traffic on the Internet is now traversing the HTTPS protocol [95]. In addition, as stated in [95, 96], the distribution of the simulated attacks in the ISCX IDS 2012 dataset is not based on real world statistics.

In order to generate this benchmark dataset, Sharafldin et. al. implmented a comprehensive testbed that consisted of two networks which they named the *Attack-Network* and *Victim-Network*. The Victim-Network was built to represent a modern-day highly secure network environment, complete with routers, firewalls, switches, and different versions of modern operating systems, including Linux, Windows, and MacOS. In addition, the Victim-Network had an agent that performed the benign behaviors on each workstation on the network. The Attack-Network was built on a completely separate network infrastructure, complete with router and switch and various PCs on multiple public IPs. These PCs were loaded with varying operating systems and software nec-

essary for launching malicious attacks. The reader is directed to Figure 1 in [95] for a detailed diagram of the testbed architecture. The way in which the actual PCAP data was captured for this dataset was via a span/mirror port setup on the Victim-Network to record all send and receive network traffic.

An important component of this dataset is the degree to which it represents real network traffic that would be naturally generated by a live network, commonly referred to as **background traffic**. In order to accomplish this, Sharafaldin et. al. created a benign profile agent, based on their previously proposed β-profile system in [94]. Similar to the background traffic generation in ISCX IDS 2012, this system profiles normal human interactions on the network in order to be used for benign background traffic generation at a later time. Using the β-profile system for this dataset, 25 users' behavior was profiled based on their use of HTTP, HTTPS, FTP, SSH, and email (SMTP). As detailed in Section 2.2.1, the β-profiles are created using machine learning and statistical analysis techniques to obtain distributions of packets, packet sizes, protocol use, etc. These generated β-profiles are then used by an agent written in Java in order to create realistic background traffic on the Victim-Network based on the real 25 users' prior behavior.

As detailed in Table 2.3 the CIC IDS 2017 dataset contains over 2.8 Million flows. This dataset also aims to cover an up-to-date and diverse set of attacks that are seen in modern day networks. Therefore, this dataset contains the following seven types of attack profiles and scenarios:

1. **Brute Force Attack:** A common attack where an attacker repeatedly attempts to guess a password by 'brute-forcing' a large number of attempts, one after another until they succeed with a correct username/password combination. Not only is this technique used for credentials, but it can also be used to 'brute-force' a web application or server, trying to find hidden pages or directories.

2. **Heartbleed Attack:** This attack exploits a vulnerability in the OpenSSL implementation of the TLS protocol. It allows the attacker to send a heartbeat payload (intended to be used to check that a server or service is still active and 'alive'), which causes the OpenSSL library to return more data to the requestor (attacker) than was designed. This enables an attacker to

Table 2.3: CIC IDS 2017 Dataset Overview

Date	# of Flows	# of Attacks	Description
Monday	529,918	0	Normal network activities
Tuesday	445,909	7,938	FTP-Patator
		5,897	SSH-Patator
Wednesday	692,703	5,796	DoS slowloris
		5,499	DoS Slowhttptest
		231,073	DoS Hulk
		10,293	Dos GoldenEye
		11	Heartbleed
Thursday Morning	170,366	1507	Web Attack - Brute Force
		652	Web Attack - XSS
		21	Web Attack - SQL Injection
Thursday Afternoon	288,602	36	Infiltration
Friday Morning	191,033	1966	Bot
Friday Afternoon 1	286,467	158,930	PortScan
Friday Afternoon 2	225,745	128,027	DDoS
Total	**2,830,743**	**557,646**	**19.70% attack traffic**

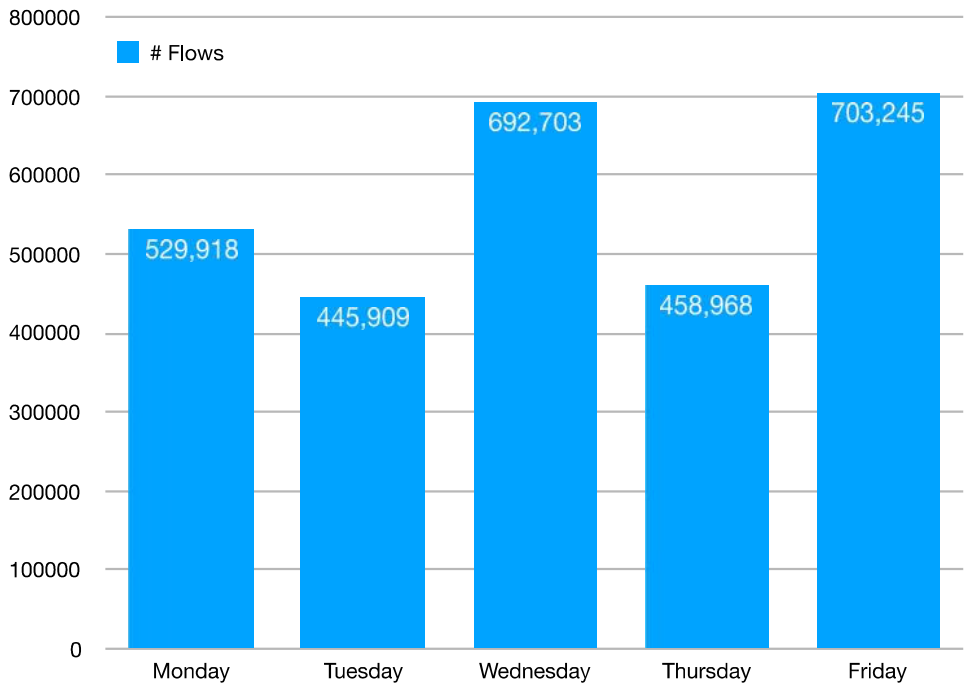


Figure 2.12: CIC IDS 2017 Dataset: Number of flows per day

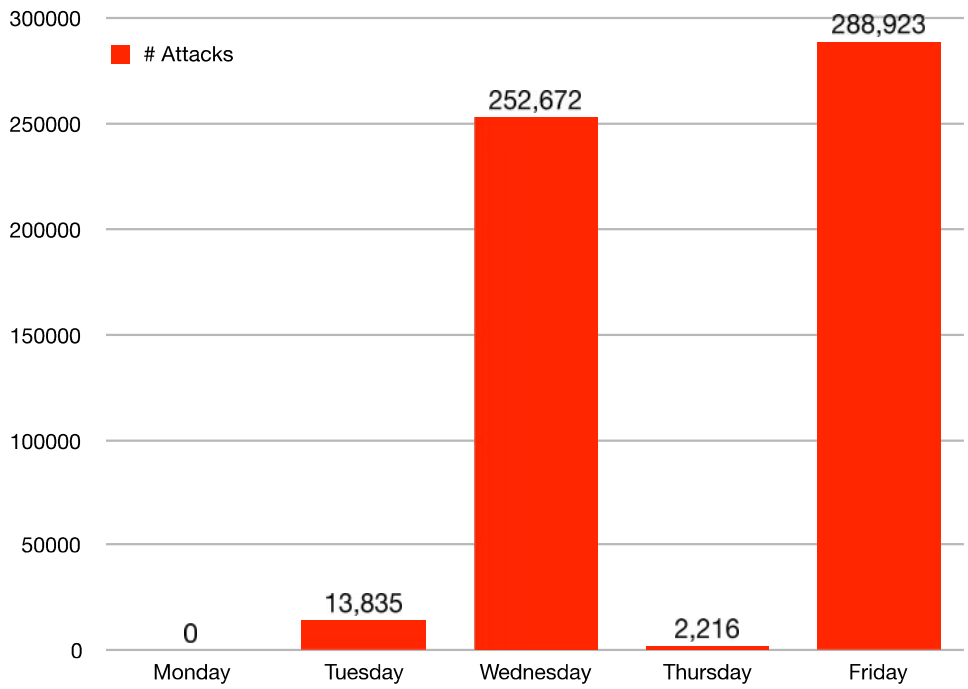


Figure 2.13: CIC IDS 2017 Dataset: Number of attacks per day

steal sensitive information such as private key material, which could later be used to decrypt confidential information.

3. **Botnet:** A botnet consists of a large number of 'zombie' hosts that have been infected with a piece of malware, whereby a **Command and Control (C&C)** server can send instructions to the bots to perform a specific command or series of commands.

4. **DoS Attack:** A Denial of Service attack is one in which the attacker intends to hinder the *availability* piece of the CIA (confidentiality, integrity, availability) triangle, causing the service to go down often by flooding the system with an abundance of requests beyond which the system can respond to.

5. **DDoS Attack:** A Distributed Denial of Service attack is similar to a DoS attack, with the only difference being that the attack is now carried out by multiple, distributed hosts (often facilitated via a botnet). These are more difficult to contain, as the source of the attack is not concentrated.

6. **Web Attack:** Web attacks can take a variety forms, and they are always evolving. In this dataset, some of the most popular forms of web attacks are performed, including SQL Injection, Cross-Site Scripting (XSS), and Brute-force password guessing. SQL Injection is a type of fuzzing attack where the attacker injects (or appends) additional string values into a form field that if not properly checked for by the web application, would trigger the database to perform commands it was not intended to perform. This is commonly a scenario in which many web applications leak sensitive data inadvertently. Cross Site Scripting (XSS) occurs when a web application contains form fields that don't properly sanitize the input, allowing an attacker to run malicious scripts on the server. Brute-force password guessing is similar to Brute-force SSH attacks, however they are run over the HTTP/S protocol against a web application/server.

7. **Infiltration Attack:** This is a dangerous attack where an external bad actor is able to gain unauthorized access to the internal network. This is often accomplished via a social engineering attack where the attacker will send a phishing email to a victim, and convince the victim to click a link that leads to a malicious website that launches an exploit, or has the victim open a malicious attachment that contains a zero-day attack that allows the bad actor to compromise the victim's computer via establishing a back door. Through this backdoor, the attacker can run commands remotely, which can be anything from performing reconnaissance on the topology of the network, looking for vulnerable services to perform lateral movement, or anything else they desire.

Each flow in this dataset contains 85 columns, including one column for the label, and another column for the FlowID. Therefore, there are a total of 83 available features. This dataset contains more features than the previous dataset because the authors provided not only raw PCAP, but also a CSV of the PCAP that had already been converted to flow records using the CICFlowMeter tool [33,55]. For the ISCX IDS 2012 dataset, the flow records were provided as an XML document for which 14 main features were extracted, as well as a label indicating whether the flow was benign or malicious. While they do also provide the raw PCAP data for ISCX IDS 2012, the labeled flow

record version did not provide the breadth of features that can be made available when converting from PCAP to flows. For the CIC IDS 2017 dataset, when the authors converted the PCAP to flow records using the CICFlowMeter tool, they output the full 85 columns of the flow record made available by the tool. In addition, this output includes a column with a label indicating whether the flow is benign, or one of 14 different attack types, which fall into one of the seven attack categories described in section previously and shown in Table 2.3. For this reason, this dataset is more useful to the research community as a benchmark dataset. The previous ISCX IDS 2012 dataset only provided labeled data in the form of an XML file, generated from the IBM QRadar SIEM appliance. Therefore, that data is not as rich in features, since many of the network protocol features that could be generated from raw PCAP were not provided in the labeled data set. Granted, ISCX IDS 2012 also does provide raw PCAP data which could be converted to flow records using the CICFlowMeter tool to acquire the same 83 features of the flow record. However, those flow records would then need to be matched up with the XML flow records to acquire labels of ground truth for the generated flow records. Results could vary from one researcher to the next once PCAP has been converted to flow records and combined with labels from the given XML file. Therefore, for the ISCX IDS 2012 dataset, experiments in this work use the given XML flow records with labels, thus having a total of 14 primary features available for use in learning.

All of the 14 primary features used in the ISCX IDS 2012 dataset are also present and used in the CIC IDS 2017 dataset. Five of the features are categorical, and the remaining 78 features are continuous. The five categorical features are 'SourceIP', 'DestinationIP', 'SourcePort', 'DestinationPort', and 'Protocol'. These overlap with the ISCX IDS 2012 dataset as expected, as they are common and necessary elements in network flow records.

There are $17,0002$ unique values for 'SourceIP', and $19,112$ unique values for 'DestinationIP'. For 'SourcePort', there are a total of $64,638$ unique values, and for 'DestinationPort' there are a total of $53,791$ unique values. For the last categorical feature 'Protocol' there are three unique values of '6.0', '0.0', and '17.0' which map to internet protocol types as defined by the Internet Assigned Numbers Authority (IANA) [7]. A value of '6.0' indicates TCP traffic, a value of '17.0'

indicates UDP traffic, and while '0.0' translates to IPv6 'Hop-by-Hop Option', it is assumed that a value of '0.0' is undefined since this dataset is dealing with IPv4 traffic. In addition, the number of flows that are of UDP type is $99,476$, and the number of flows for TCP traffic is $1,826,704$, leaving only $1,696$ flows with protocol label '0.0'. Therefore, with these five categorical features, the CIC IDS 2017 dataset can be expanded to a maximum of $17,002 + 19,112 + 64,638 + 53,791 + 3 =$ **154,546** features. The full list of features in the CIC IDS 2017 dataset are detailed in Table A.1 within Appendex A.

In this chapter, the background and preliminaries in regards to machine learning, deep learning, evaluation metrics, as well as details on the NID datasets being used in this Thesis were provided. In the next chapter, the details of the experiments, implementation, and their results will be reviewed.

Chapter 3: USING DEEP NEURAL NETWORKS FOR NETWORK INTRUSION DETECTION

3.1 Introduction

In this chapter a series of case studies will be performed using a deep fully connected feedforward neural network to perform classification on two different network intrusion detection benchmark datasets.

In this chapter, the following contributions are provided:

1. Application of a fully connected deep neural network to network intrusion detection. Through experimentation, it is determined the set of hyperparameters and network configuration that produces optimal results in terms of evaluation metrics. In addition, comparison is performed between including and excluding IP address as features.

2. Validation of the approach by evaluating results using two benchmark intrusion detection datasets. The two benchmark datasets used in this research are recent and contain modern-day attacks. Using these datasets with the deep neural network approach in this chapter shows that this technique can be applied to modern-day enterprise networks.

3. Evaluation and comparison of this deep learning approach for network intrusion against other previous techniques. There are a number of studies that have now been performed using the two intrusion detection benchmark datasets used in this Thesis. Therefore, the results achieved in this work are compared with other approaches in the field.

3.2 Case Study Preparation

For this case study, experiments are performed with the previously mentioned two datasets – ISCX IDS 2012 benchmark dataset [96] and CIC IDS 2017 benchmark dataset [95]. Each of these datasets contain ground truth labels which is necessary for carrying out supervised learning. While the version of the ISCX IDS 2012 dataset used in this work has binary class labels for *benign* and

malicioius, the CIC IDS 2017 dataset contains multinomial class labels for each of the attack types carried out in the flow records. Each dataset is used within a common machine learning workflow to apply the deep neural network approach to network intrusion detection. The results of each of the experiments will be analyzed and evaluated, then the effectiveness of this approach against previous approaches in the field will be compared.

3.2.1 Experiment Setup

To implement and evaluate the deep learning algorithms in this work, experiments were performed on the Texas Advanced Computing Center (TACC) resources. To perform the deep learning tasks, experiments leverage the popular open source TensorFlow machine learning framework [11], along with the Keras high level framework [28] which uses TensorFlow on the backend. The experiments were run on a Dell R630 server with 24 cores setup in dual socket Intel Xeon E5-2670 v3 "Haswell" processors (each with 12 cores @ 2.3GHz) and 128 GB of RAM.

3.2.2 Machine Learning Workflow

Figure 3.1 highlights a supervised machine learning pipeline. This workflow is utilized (minus the model deployment step) in the experiments with each of the benchmark datasets.

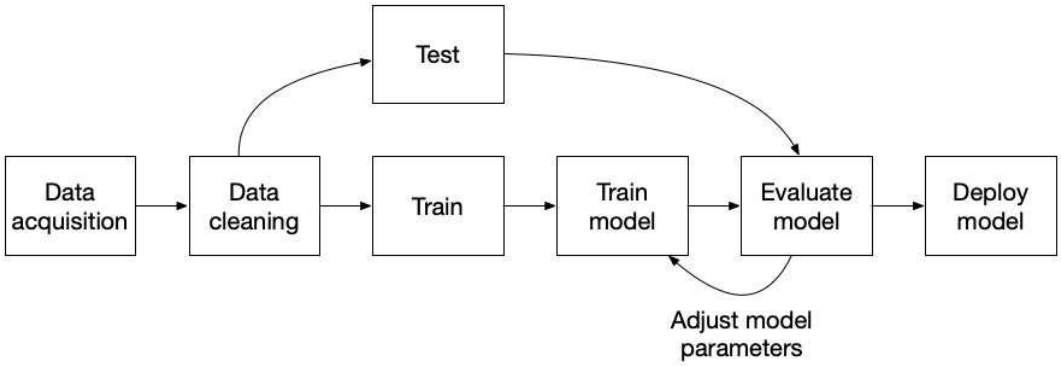


Figure 3.1: Supervised Machine Learning pipeline

3.2.3 Network Traffic Data: PCAP vs NetFlow

Network traffic data is often collected at network switches or routers in the form of raw packet capture (PCAP). PCAP data contains the full TCP/IP packet data transmitted or received on a given network device. While full packet capture information can be helpful in certain scenarios, it does bring with it a high cost in terms of space. An alternative (or complement) to PCAP data is NetFlow, which serves to summarize the PCAP data in terms of higher level network flows. NetFlow data offers the advantages of more breadth and lower disk space requirements, whereas PCAP data has more depth, but is more expensive in terms of disk space. NetFlow records typically consist of source/destination IP, source/destination port, source interface, protocol, and number of bytes transmitted. It can have many more fields extracted from the PCAP based on the configuration of software used for converting PCAP to NetFlow records.

The machine learning algorithms used in this work utilize NetFlow records as input. Therefore, if there is any PCAP input data, it must first be converted to NetFlow. In practice, running an algorithm that takes NetFlow as input is ideal — the NetFlow protocol was created by Cisco, and is commonly used by many network switches, routers, and firewalls to export raw traffic data into flow data for further analysis. An emerging IETF standard called IPFIX, short for IP Flow Information Export, is similar to the NetFlow standard, but allows for additional information that is normally sent to syslog, as well as variable length fields for collecting information such as URLs, messages, HTTP hosts, and more [2, 6]. For the purposes of this work, the summarized traffic records will simply be referred to as 'flow data'.

In order to convert raw PCAP data to flow data, there are a number of tools available. One tool is called YAF, short for Yet Another Flowmeter. YAF was created by Computer Emergency Response Taskforce (CERT) and is a part of their Network Situational Awareness (NetSA) Security Suite of tools used for monitoring large-scale networks. YAF processes packet data from PCAP into bidirectional flows, then exports the flows to IPFIX-based file format.

Another tool that can be used for converting PCAP data to network flow data is the CICFlowMeter, created by University of New Brunswick [33, 55] . For the CIC IDS 2017 Dataset,

the authors provide the dataset in the form of both PCAP as well as computed flow records via CICFlowMeter. The CICFlowMeter is a network traffic flow generator written in Java that takes raw PCAP as input and generates bidirectional flows where the first packet determines the forward (source to destination) and backward (destination to source) directions. It generates 77 statistical features (the features available for CIC IDS 2017 specifically are detailed in Table A.1), such as duration, number of packets, number of bytes, length of packets, which are also calculated separately in the forward and backwards directions. Along with these statistical features for each flow, CICFlowMeter provides the corresponding source/destination IP address and port, protocol number, timestamp, label, and a unique FlowID for each flow record.

3.2.4 Case Study I: ISCX IDS 2012 Dataset

The following subsections detail a deep learning approach applied in experiments using the ISCX IDS 2012 benchmark intrusion detection dataset.

3.2.4.1 Data Acquisition

The ISCX IDS 2012 Dataset contains raw PCAP files of the network traffic, as well as flow records in the form of an XML file that was processed through an IBM QRadar network security appliance. Each of the flows in the XML file were labeled as either *benign* or *malicious*. For this experiment, instead of utilizing a tool such as YAF or CICFlowMeter to convert the PCAP data into flow records, the provided XML flow records are used as a starting point. The reason for this is that the labels for each of the network flows are only provided through the XML files. Therefore, a reliable way to obtain ground truth labels from this dataset is to use these XML flows, as opposed to generating the network flow records from the raw PCAP files. In order to process the XML records, a custom parser was developed that iterates through all the XML records and outputs structured CSV files that can more easily be used as input into a machine learning pipeline. The labeled XML flows were only provided for days two through seven, for which there was attack traffic. Therefore, for this experiment, day one is being omitted and only days two through seven are being used.

3.2.4.2 Data Cleaning and Preprocessing

Once the data is in the form of flow data, a process is performed to further clean, transform, and prepare the data for use in the deep learning algorithm. This step includes common tasks such as making sure there are no erroneous characters in the data, removing fields that contain all zero or null values, and removing or modifying NaN (not a number) values. Often, a large amount of time can be spent in this stage to clean and prepare the data. This is due not only to the various formats and locations of target data, but also to ensure the accurateness and effectiveness of the model that is being trained on this data.

During this stage, it is common practice to also normalize or scale the continuous values amongst all the features, so that the machine learning algorithm trains on data that is all within the same feature space. There are two main types of normalization that are commonly used within the machine learning pipeline: *Standardization* (or *Z-score normalization*) and *Min-Max scaling*.

Standardization results in the features being rescaled so that they will have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, where μ is the mean (average) and σ is the standard deviation from mean. The standard scores (also called *z scores*) of the features are calculated as follows:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

Min-Max scaling, commonly referred to as normalization, consists of data being scaled to a fixed range, typically [0,1]. The only issue with this type of scaling method is that there will be smaller standard deviations, which can work to suppress the effectiveness of outliers. Min-Max scaling is performed using the following formula:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.2}$$

In this work, Min-Max scaling is used to normalize the features during this step.

3.2.4.3 Feature Selection

The amount of features available in this first dataset of converted flow records is relatively small at a total of 14. As described in Section 2.2.1, the main features provided in this dataset after processing the labeled XML flows are as follows: Source IP, Destination IP, Source Port, Destination Port, App Name, Direction, Protocol, Duration, Total Source Bytes, Total Destination Bytes, Total Bytes, Total Source Packets, Total Destination Packets, Total Packets. Of these features, Source IP, Destination IP, Source Port, Destination Port, App Name, Direction, and Protocol are categorical features. As described earlier in Section 2.2.1, the number of unique possible values across all of these categorical features is $2,478+34,552+64,482+24,238+107+4+6+7 =$ **125,867**. Therefore, to experiment with using these high dimensional categorical features, they must be converted to floating point values for use in the neural network. There are a number of different approaches for accomplishing this, ranging from simple to more complex. The following techniques for working with these high-cardinality categorical variables are described: one-hot encoding, one-hot hash trick, and entity embedding.

One-hot encoding is a common technique for converting categorical feature variables into continuous features for use with a machine learning algorithm. This process works by creating a new variable for every unique value possible for the original categorical feature. An example of how this works is if there is a categorical feature named 'animal' with three different possible values, such as 'cat', 'dog', 'bird'. Since the machine learning algorithm only works with continuous floating point values, one-hot encoding can encode this 'animal' feature as shown in tables 3.1 and 3.2.

Using one-hot encoding in the context of neural networks has two main problems [32]:

1. One-hot encoding of high cardinality features can be inefficient and require a large amount of computational power to complete.

2. One-hot encoding ignores the relationships between categorical features, and treats them completely independent of one another.

Table 3.1: One-hot encoding example - before encoding

No.	[Other columns]	animal	[Other columns]
1	···	cat	···
2	···	cat	···
3	···	bird	···
4	···	dog	···
5	···	bird	···
6	···	cat	···
···	···	···	···
n	···	dog	···

Table 3.2: One-hot encoding example - after encoding

No.	[Other columns]	animal_cat	animal_dog	animal_bird	[Other columns]
1	···	1	0	0	···
2	···	1	0	0	···
3	···	0	0	1	···
4	···	0	1	0	···
5	···	0	0	1	···
6	···	1	0	0	···
···	···	···	···	···	···
n	···	0	1	0	···

A variant of one-hot encoding is a technique called the *one-hot hashing trick*. This is useful when the number of possible values for the categorical variable is large. Instead of assigning an index to each value and keeping track of it in a vector as is done with one-hot encoding, the hashing trick uses a simple hash function to hash each possible value into a vector of a fixed size. The main advantage of this method is that there is no longer the need to maintain an explicit index of possible values (a "word index" when used in the context of natural language processing and text). This provides a savings in terms of memory, and also allows for on-the-fly creation of hash values ("tokens") before needing to see the entirety of possible values for the categorical variable. The main disadvantage of this technique is that if the size of the hashing table is not large enough to represent the possible number of values for the categorical variable, the problem of *hash collisions* occurs. In a hash collision, two different values would map to the same value. This is a problem for machine learning, as the algorithm would see these two distinct values as being the same. One way to prevent hash collisions is to make sure the dimensionality of the hashing space is much larger than the maximum number of values for a given categorical variable [28].

Entity embedding is a technique that can overcome the shortcomings of one-hot encoding and one-hot hashing trick. It is useful when the number of unique values for a categorical feature is of high cardinality [48]. Using embeddings for categorical variables is a key technique for leveraging the power of deep learning when working with tabular data [100]. This idea draws from word embedding techniques, such as *Word2Vec*, where similar words are grouped together. Using this technique, relationships between different categories can be captured. Embeddings map categorical values to low-dimensional real vectors in such a way that similar values are close to each other [46]. The lower-dimensional space preserves semantic relationships. The structure in which embeddings are mapped is geared toward what is trying to be accomplished with the underlying categorical variables. The key categorical variables in network intrusion detection consist of the source/destination IP addresses and Ports. With embeddings, vectors for similar categorical values are closer together, and unrelated values have vectors that are further away. An example for word embeddings would be 'puppy', 'kitten', 'fawn', and 'mountain'. In this example, the vectors that

represent the first three baby animals would be closer together, and the vector for the embedding that represents ‘mountain‘ would be further away.

Each category is represented by a dense vector of floating point numbers. The values in the vector are learned. Embeddings are able to capture richer, more meaningful and complex relationships, as opposed to raw categories. Embeddings are more useful than one-hot encoding as the embedding technique provides an extra level of relationship information and can group values together into a smaller number of categories [100]. In addition, embeddings allow for the reduction of the dimensionality of the feature space for the categorical variables, which helps in reducing overfitting [99].

According to Cheng Guo and Felix Berkhan [48], embeddings can help models to generalize better when data is sparse and statistics are unknown. They mention that this technique is especially useful for datasets which contain high cardinality features, where other methods would tend to overfit. For the datasets used in this work, this is especially applicable, as there are thousands of possible values for each of the source IP, destination IP, source port, and destination port features. Therefore in this work, the embedding technique is used for these high cardinality features. The initial embedding dimensions used are according to the following rule of thumb [46]:

$$dimensions = \left\lceil \sqrt[4]{possible\ values} \right\rceil \tag{3.3}$$

Another rule of thumb for number of reduced dimensions used by [100] is as follows:

$$dimensions = min(50, \frac{possible\ values}{2}) \tag{3.4}$$

Therefore, for the ISCX IDS 2012 dataset, the high cardinality features are reduced in dimension using entity embeddings, following the first rule of thumb, as shown in Table 3.3.

In effect, this borrows from the technique used in word embedding, and performs IP embedding and Port embedding to utilize these features within a neural network. Specifically, the high cardinality categorical variables are each mapped to an integer index. Then, each of the categor-

Table 3.3: Embedding Categorical Variables - ISCX IDS 2012 Dataset

Feature	Possible Values	Embedded Dimensions
Source IP	2,478	7
Destination IP	34,552	13
Source Port	64,482	16
Destination Port	24,238	12

ical values are represented by a dense vector of size $\sqrt[4]{possible\ values}$. When these categorical variables are embedded, they are then represented by a dense vector of floating point numbers according to the dimension that are calculated. The parameters (weights) for the dense vector representation are initialized using a random uniform distribution in the range $[-0.05, 0.05]$. An example of the embedded representation of sample source IP addresses into an 8-dimensional vector is shown in Table 3.4. This dense representation is not only more computationally efficient, but also holds inherent relationship information between the various IP addresses, the other features in the dataset, and the label. As these vectors are inputs into the first layer of the neural network, their weights are updated during the back-propagation step at each epoch.

Table 3.4: Embedded Categorical Features - Source IP Address Example

Index	Latent Factors							
1	-0.041	0.028	-0.013	0.038	-0.037	-0.035	-0.026	-0.033
2	0.004	-0.024	-0.050	-0.032	0.005	0.021	0.009	-0.044
3	-0.017	0.015	0.042	0.008	0.023	0.047	-0.009	0.036
4	0.041	0.007	0.029	-0.010	-0.013	0.025	0.035	0.015
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
n	0.043	-0.049	-0.009	0.024	0.035	0.003	-0.026	-0.041

3.2.4.4 Training, Validation, and Test Split

Following best practices, the dataset is separated into three separate training, validation, and test sets. First, the dataset is separated out with 33% of the data put into a test set, and 67% into a training set. In addition, since this dataset is highly imbalanced, it is important to ensure there is the same proportion of malicious flows in each training and test sample as there are in the dataset as a whole. In order to accomplish this, the dataset is stratified to ensure the distribution of benign

and malicious traffic is equivalent in both training and test data sets. In addition to the standard training and test split, a separate hold out validation set is used during the training iterations. This is a standard methodology used when there is plenty of data available [28]. Therefore, the training split is further separated out into a validation set according to the same parameters as the original training and test split, including stratification. This results in 67% of the training dataset set aside for training, and 33% of the training dataset used for validation during the training iterations.

The reason for having a validation set is to help in tuning the configuration of the model. For example, modifying the number of layers, or units per layer can have an effect on the performance of the model. This tuning is performed as a feedback loop using the model's performance on the validation dataset. Once the model performs well on the validation set, it is run against the separate test dataset which it has never before seen. This ensures the model is not overfitting the training and validation set. The knobs that are tuned during this training and validation cycle are called *hyperparameters*, which will be covered in the next section.

3.2.4.5 Hyperparameters

Hyperparameters refer to the parameters that are set before the learning process begins. These hyperparameters are tuned based on a feedback loop of the model's performance on the validation set. Hyperparameters fall into the following main categories [86]:

- Layer size, number of units per layer
- Magnitude (momentum, learning rate)
- Regularization (dropout, L1, L2)
- Activation functions (sigmoid, relu, tanh, etc.)
- Weight initialization
- Loss functions
- Number of epochs to train, and batch size per epoch

- Data normalization scheme

Throughout the training and validation cycle, these hyperparameters are tuned in order to achieve highest and best performance for the machine learning model. The most effective neural network architecture and hyperparameters used in this case study are described next.

The configuration for the neural network as shown in Figure 3.2 gave best results in the experiments. The first hidden layer for the continuous input features is much larger in this case at 128 neurons, than the number of inputs at 7. Smaller numbers of neurons were also tested, and found to achieve comparable results. Therefore, to keep architectures consistent, this same general configuration is also used in the next case study with CIC IDS 2017, which has many more continuous input features. It should also be noted that it is recommended future work to perform an in-depth study that performs further hyperparameter optimization to determine the neural network architecture and combination of features that performs best for each of the given datasets.

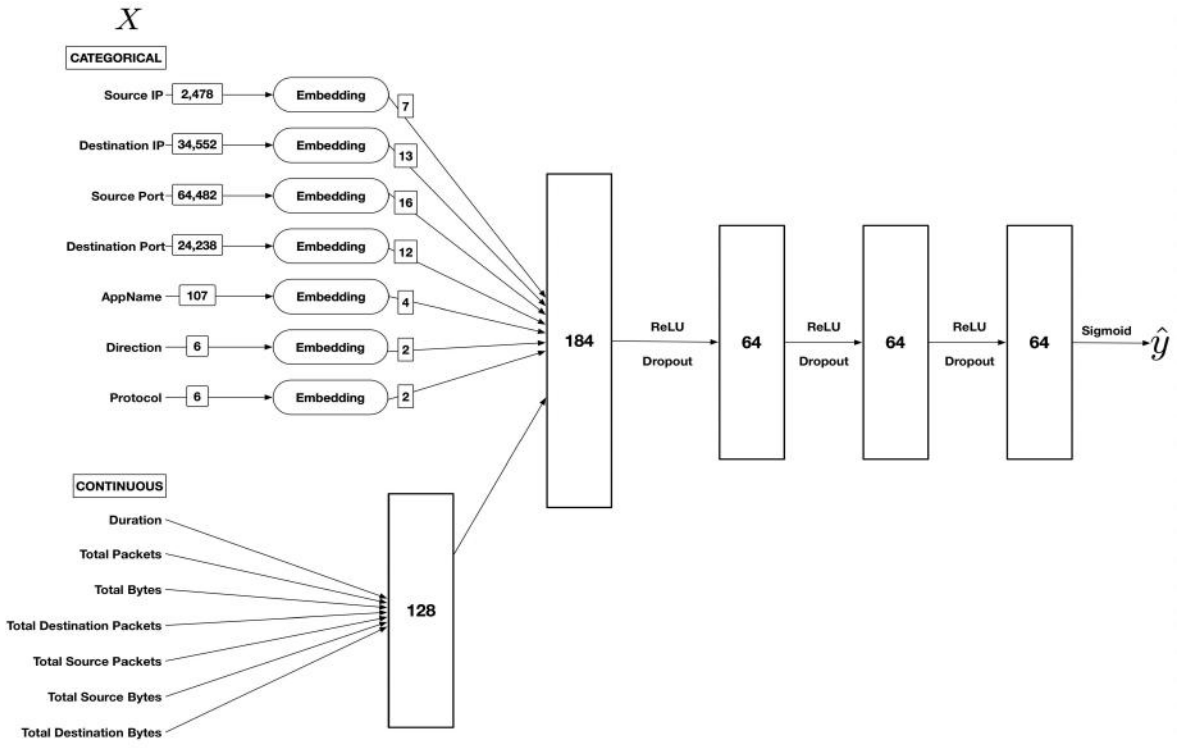


Figure 3.2: Deep Neural Network Architecture for ISCX IDS 2012 Dataset

The neural network that was chosen consists of three layers of hidden units, consisting of 64

units per layer. The activation function on each layer is the ReLU activation function, with the last output layer having a sigmoid activation function. The optimizer used is the RMSProp optimizer, with a default learning rate of 0.001. After initially using the Adam optimizer, it was found that RMSProp performed better at optimizing the loss function, enabling better results with validation and test sets. The loss function used is the binary crossentropy:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \tag{3.5}$$

where y is the label (1 for malicious and 0 for benign), and $p(y)$ is the predicted probability of the given flow being malicious for all N flows. This formula for binary crossentropy shows that for each malicious point ($y = 1$), it adds $\log(p(y))$ to the loss, which is the log probability of the example flow being malicious. Conversely, it also adds $\log(1 - p(y))$, which is the log probability of the flow being benign, for each benign flow ($y = 0$).

3.2.4.6 Evaluation and Results

In this section, the results and evaluation of experiments with the ISCX IDS 2012 dataset are described. The experiments show that by using deep fully connected feedforward neural networks for classification of network traffic flows as either benign or malicious, and using the embedding technique for the high cardinality categorical features, high performance can be achieved in terms of evaluation metrics. A variety of hyperparameters were iterated with along with different combinations of features in order to arrive at the best results.

As a review, this dataset is highly imbalanced, with only **2.71%** of the traffic flows being attack traffic, as shown in Figure 3.3.

The confusion matrix in Figure 3.4 shows the results of using this neural network architecture on the ISCX IDS 2012 dataset. In addition, the technique of using embeddings for the categorical variables was used. The detailed metrics are also shown in Table 3.5.

The comparison of Precision vs Recall is shown in Figure 3.7. A Precision/Recall Curve is the preferred metric to use when the classes are highly imbalanced, as is the case in this case study.

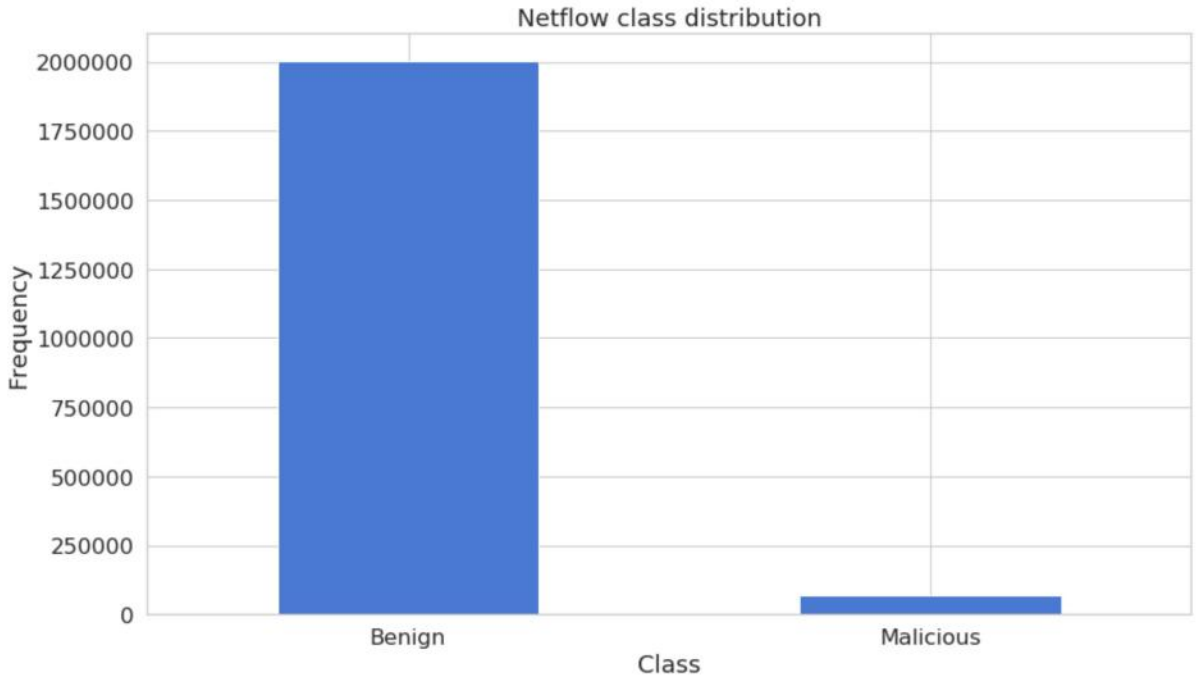


Figure 3.3: ISCX IDS 2012 Class Distribution

Table 3.5: ISCX IDS 2012 Evaluation Results - Metrics

Metric	**Result**
True Negative	659,138
False Positive	1,474
False Negative	889
True Positive	22,043
Area Under the Curve	0.9795
Accuracy	0.9965
Error Rate	0.0035
True Positive Rate (Sensitivity, Recall, Detection Rate)	**0.9612**
True Negative Rate (Specificity)	0.9978
False Positive Rate	**0.0022**
False Negative Rate	**0.0388**
Precision	0.9373
F1 Measure	**0.9491**

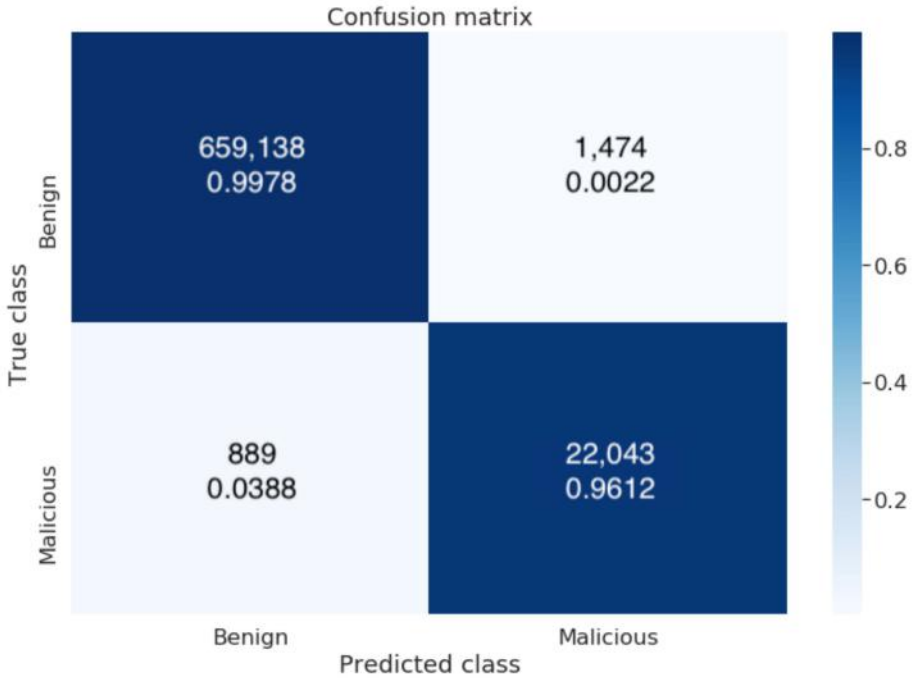


Figure 3.4: ISCX IDS 2012 Confusion Matrix using Embeddings with IP Address

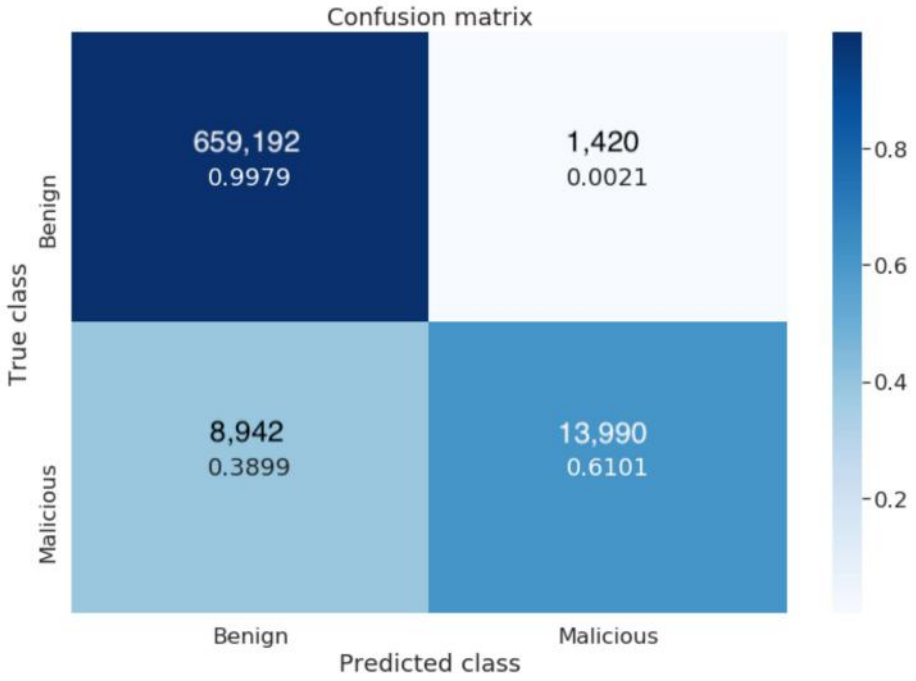


Figure 3.5: ISCX IDS 2012 Confusion Matrix - without IP Address

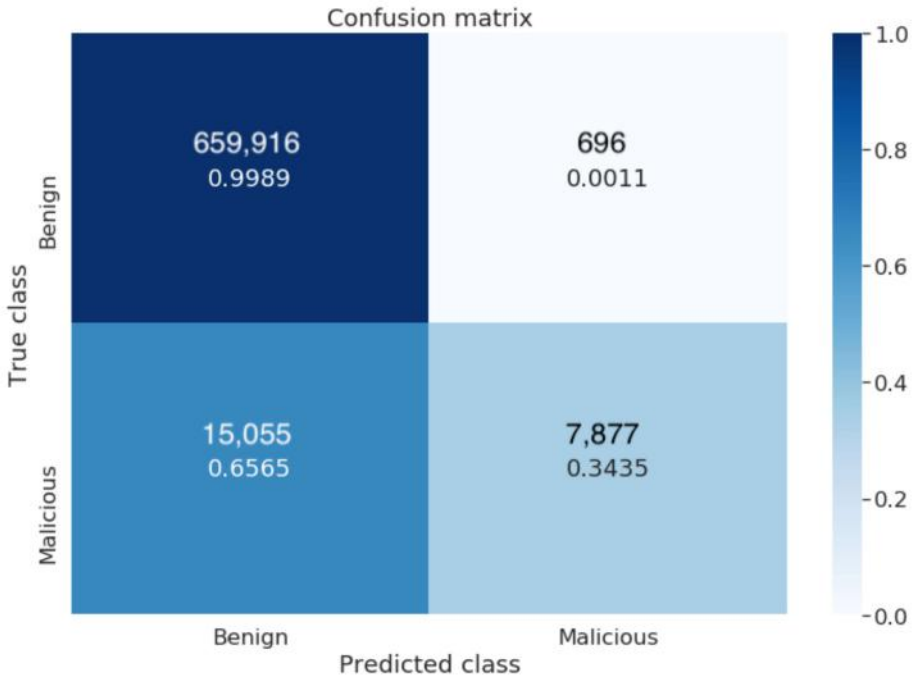


Figure 3.6: ISCX IDS 2012 Confusion Matrix - without IP Address, AppName, Direction

The higher towards the right of the Precision/Recall Curve, the better. As seen in Figure 3.7, the Precision/Recall curve for this case study is favorable.

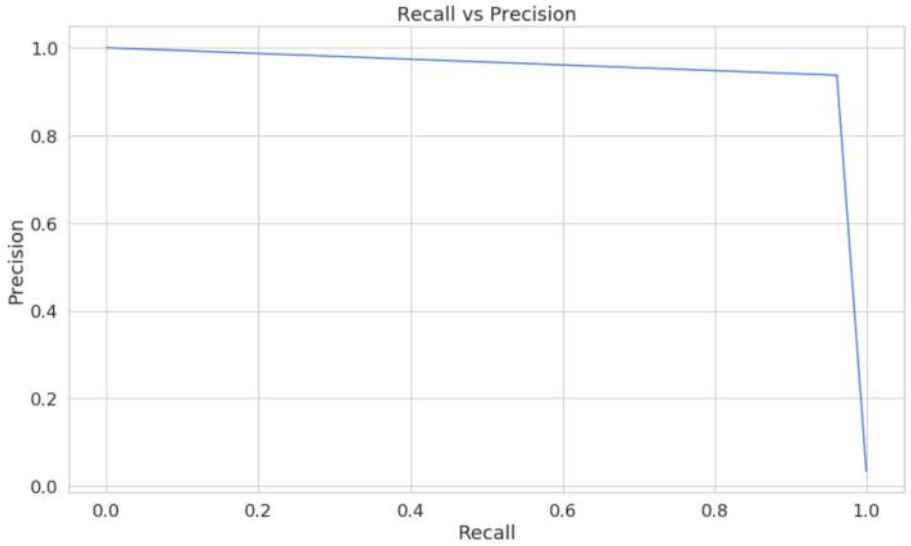


Figure 3.7: ISCX IDS 2012 Recall vs Precision

In addition, the F1 Measure, which is the one of the primary metrics which is being evaluated in this work, is above 90% at a value of 0.9491.

The false positive rate for this experiment reaches a low of 0.0022, which is lower than other leading machine learning techniques used on this same dataset. In the work of [16] published in 2017, Atli compared their technique along with other leading machine learning approaches against the ISCX IDS 2012 dataset. The results of these other techniques in terms of detection rate and false positive rate is shown in Table 3.6. As can be seen, the method used in this work outperforms all the others in terms of the False Positive Rate metric, reaching a low of 0.0022. For the average detection rate, also referred to as True Positive Rate or Recall (a measure combining the true positive rate and false negative rate), our method achieves a detection rate of 0.9612.

Table 3.6: ISCX IDS 2012 Result Comparison [16]

Technique	**Avg. DR**	**Avg. FPR**
KMC+NBC [119]	**0.98**	0.02
Bagged-NB [54]	0.45	0.09
Boosted-NB [54]	0.43	0.09
AMGA2-NB [54]	0.94	0.07
Decision Tree Classifier with Snort Priority [15]	**0.98**	0.06
CAGE-MetaCombiner [42]	0.76	0.06
CAGE-MetaCombiner + Specialized Ensemble [42]	0.83	0.09
EMD (low threshold, only on DoS attacks) [98]	0.90	0.08
EMD (high threshold, only on DoS attacks) [98]	0.44	0.01
ELM+PT 1 [16]	0.90	0.09
ELM+PT 2 [16]	0.91	0.09
ELM+PT 1 (subset of dataset) [16]	**0.994**	0.016
ELM+PT 2 (subset of dataset) [16]	**0.997**	0.008
DNN	**0.9612**	**0.0022**

In looking at the detection rate of other methods, [119], [15], and the last two methods of [16] have higher detection rates. When looking at [119], the authors only selected incoming malicious and benign packets for certain days for a particular host. Similarly, the authors in [54] only randomly selected a subset of benign and malicious attack traffic to create their training and test dataset. For [15], the authors used Snort to generate ground truth labels for flow data. Therefore, [16] used a subset of the dataset to emulate the dataset version used by [15], and they got the resulting highest results for detection rate. For this Thesis, the ground truth that was evaluated was from the creators of the ISCX IDS 2012 dataset itself, which generated flow data from an

IBM QRadar appliance. The main takeaway from this is that while the ISCX IDS 2012 dataset is a good first step towards a benchmark intrusion detection dataset, there still exists the problem that a common, ground truth set of flow data was not provided in the form of CSV files for which various studies can perform true apples to apples comparison. Each of the studies in this comparison table used their own method of converting PCAP to flow records, and furthermore used subsets of the dataset for performing evaluation and analysis. These issues were addressed when a newer benchmark dataset, CIC IDS 2017 was released. Another point to mention is that in achieving their top results, [16] used an Extreme Learning Machine (ELM), which is a single hidden layer feedforward neural network, that was configured with 200 hidden neurons. Therefore, future work should look at performing more in-depth hyperparameter optimization to determine the best deep neural network configuration for achieving the best results. This also lends further credence to the utility of neural networks for the field of network intrusion detection. While the comparisons for the ISCX IDS 2012 are not ideal due to the varying ways in which the ground truth data for each comparative study was generated, the results in this Thesis could be improved further by performing more in-depth hyperparameter optimization. As will be seen in the next case study, it is found that the performance of the deep neural network increases further by having more malicious examples available for training.

For this first case study, a minimal number of features where used which represent what is commonly available from the NetFlow format and in typical security appliances. In particular, it did not include detailed flow statistics, which will be used in the next case study. Also, other experiments were run without using IP Address as features, and also without using the categorical features of 'AppName' and 'Direction'. These experiments showed that IP address is an important feature for achieving the best results. When all three of these aforementioned categorical variables are removed, the performance of the classifier drops sharply.

Insight 1. *Despite the smaller number of features in this dataset, the deep neural network still performs well by using the embedding technique to include high-dimensional categorical variables. Using all the features, the neural network achieves a lower false positive rate than other leading*

methods at 0.0022*, and has a detection rate of* 96.12%*. In particular, using the IP address as a feature is important for achieving best results, especially when there are a smaller number of other features available. When removing the IP address, the performance of the neural network drops sharply.*

3.2.5 Case Study II: CIC IDS 2017 Dataset

The following subsections detail a deep learning approach applied in experiments using the CIC IDS 2017 benchmark intrusion detection dataset.

3.2.5.1 Data Acquisition

The main difference with the CIC IDS 2017 dataset as compared to the ISCX IDS 2012 dataset is the number of network flow features available. The ISCX IDS 2012 labeled dataset has 14 total features, while the labeled CIC IDS 2017 dataset has a total of 83 available features. These additional features primarily consist of detailed flow statistics and calculations regarding the packet details for each flow, which were generated using the open source CICFlowMeter tool. These features are discussed in more detail in Chapter 2 and can also be reviewed in Appendix A. Another main difference in the CIC IDS 2017 dataset is that it is made available in the form of CSV files with labels, containing pre-processed packet data in the form of flow records. As described earlier, for the ISCX IDS 2012 dataset, the labeled data was provided in the form of an XML document that was produced via an IBMQRadar appliance. Therefore, there was some additional XML parsing that had to be done in order to prepare the ISCX IDS 2012 dataset for use in the machine learning algorithm.

3.2.5.2 Data Cleaning and Preprocessing

After inspecting the dataset, it is found that there are a total of $289,960$ rows that contain at least one NaN value. Within these, there were a total of $288,602$ rows with all values containing NaN values. Therefore, all these rows were dropped from the dataset, leaving a total of $1,358$ rows that had at least one 'NaN' value in it. The only column that contained remaining 'NaN' values was 'Flow Bytes/s'. In addition, there is a value of 'Infinity' in a total of $2,867$ rows, specifically in the

column 'Flow Packet/s'. Therefore, to clean this data, the 'Infinity' values are converted to NaN, then these $2,867$ rows are dropped that had contained NaN and/or 'Infinity' values in any of the columns.

In addition, after reviewing the statistics of the dataset and features such as mean, standard deviation, min, max, there are a number of columns that have all 0 values for every single example. Therefore, since these features provide no information to the learning algorithm, the following columns are dropped from the dataset: *Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate*.

As discussed in the previous case study, it is imperative that the continuous feature values all be normalized into the same feature space. This is important when using features that have different measurements, and is a general requirement of many machine learning algorithms. Therefore, the values for this dataset are also normalized using the *Min-Max scaling* technique, bringing them all within a range of [0, 1].

3.2.5.3 Feature Selection

In this dataset, after cleaning the data, there are a total of 74 useable features, with five of them being categorical. The categorical features are *Source IP, Source Port, Destination IP, Destination Port, and Protocol*. As described in Chapter 2, the total number of unique values across these five categorical variables is $17,002 + 19,112 + 64,638 + 53,791 + 3 =$ **154,546** features. In order to use the features, they must be converted to a floating point value. Following the same technique used for converting categorical variables in the ISCX IDS 2012 dataset, these features are converted to floating point numbers using the categorical embedding technique. As described, there are two general techniques for calculating the number of embedded dimensions. For the experiments, both techniques were utilized and compared to see if it had an effect on the performance of the classifier. One technique, referred here to as *Embedding Technique A*, as described is $dimensions = min(50, \frac{possible\ values}{2})$. Another technique, referred here to as *Embedding Technique B*, reduces dimensions using the formula $dimensions = \left\lceil \sqrt[4]{possible\ values} \right\rceil$. Each of

these techniques generates a dense representation of each categorical variable with vectors of reduced dimensionality, with the latter reducing even further as shown in Table 3.7. Comparing these two embedding techniques, it was found that on this dataset, they perform nearly identical, with the latter technique of using the fourth square of the possible values slightly outperforming the former technique. In one experiment, the former achieved an F1 measure of 0.9989440, and the latter achieved an F1 measure of 0.9989576. This leads to the following insight:

Insight 2. *The rule of thumb for embedding categorical variables of using the fourth square of the number of possible categorical variables (Embedding Technique B) performs just slightly better than Embedding Technique A for this given dataset. This allows for a smaller, more dense amount of features, which reduces the number of parameters in the neural network and thus reduces the computational overhead for training, while also providing better performance.*

All the useable numeric features (full list in Appendix A) as well as the five categorical features of *Source IP, Source Port, Destination IP, Destination Port, and Protocol* serve as input to the neural network.

Table 3.7: Embedding Categorical Variables - CIC IDS 2017 Dataset

Feature	Possible Values	Embedded Dimensions
Source IP	17,002	12
Destination IP	19,112	12
Source Port	64,638	16
Destination Port	53,791	15

This dataset contains labels for not only benign and malicious, but also indicates the type of malicious attack that was carried out for each malicious flow. Therefore, this data can be looked at as either a binary classification problem, or a multinomial classification problem. There are a total of fourteen different labels, one *benign*, and thirteen with various attack types. For the binary classification problem, the labels are converted to 0 and 1, with 0 representing the *benign* class, and 1 representing the *malicious* class. For performing multinomial classification, the labels can be vectorized using the one-hot encoding technique. As described in Chapter 2, one-hot encoding consists of mapping a unique integer index with each label, then turning a given integer index i into

a binary vector of size *n* (number of labels of 14 in this case). For this dataset, one-hot encoding consists of embedding each of the labels into an all zero vector of size 14, with a single 1 value in the place of the index for the index corresponding to the label. For example, this dataset contains the following class labels as strings: *BENIGN, FTP-Patator, SSH-Patator, DoS slowloris, DoS Slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed, Web Attack Brute Force, Web Attack XSS, Web Attack Sql Injection, Infiltration, Bot, PortScan, DDoS*.

For a multinomial classification problem, this dataset can be converted using one-hot encoding, with the label for the $benign$ class represented as the following vector (tensor): [1,0,0,0, 0,0,0,0,0,0,0,0,0,0], and one of the the attack class labels of 'DoS GoldenEye' represented by the following one-hot encoded vector: [0,0,0,0,0,0,1,0,0,0,0,0,0,0]. However, for this first experiment, this dataset is evaluated as a binary classification problem, where all the malicious flows are converted to the value 1, and all benign flows are converted to the value 0.

3.2.5.4 Train, Validation, and Test Split

Similar to the previous case study, this dataset is split into separate training, validation, and test sets with the original data split into 67% training and 33% test; then this training set is further split up into 80% for final training set, and 20% for hold-out validation set. This dataset is also imbalanced, but not quite as imbalanced as the ISCX IDS 2012 dataset. Instead, there is a total of 19.70% malicious flows, and 80.30% benign flows. Therefore, when splitting the dataset, it must be stratified so that there is a consistent proportion of malicious and benign flows within each training, validation, and test split.

3.2.5.5 Hyperparameters

After running several experiments tweaking various parameters, high performance is achieved using a neural network configuration as shown in Figure 3.8.

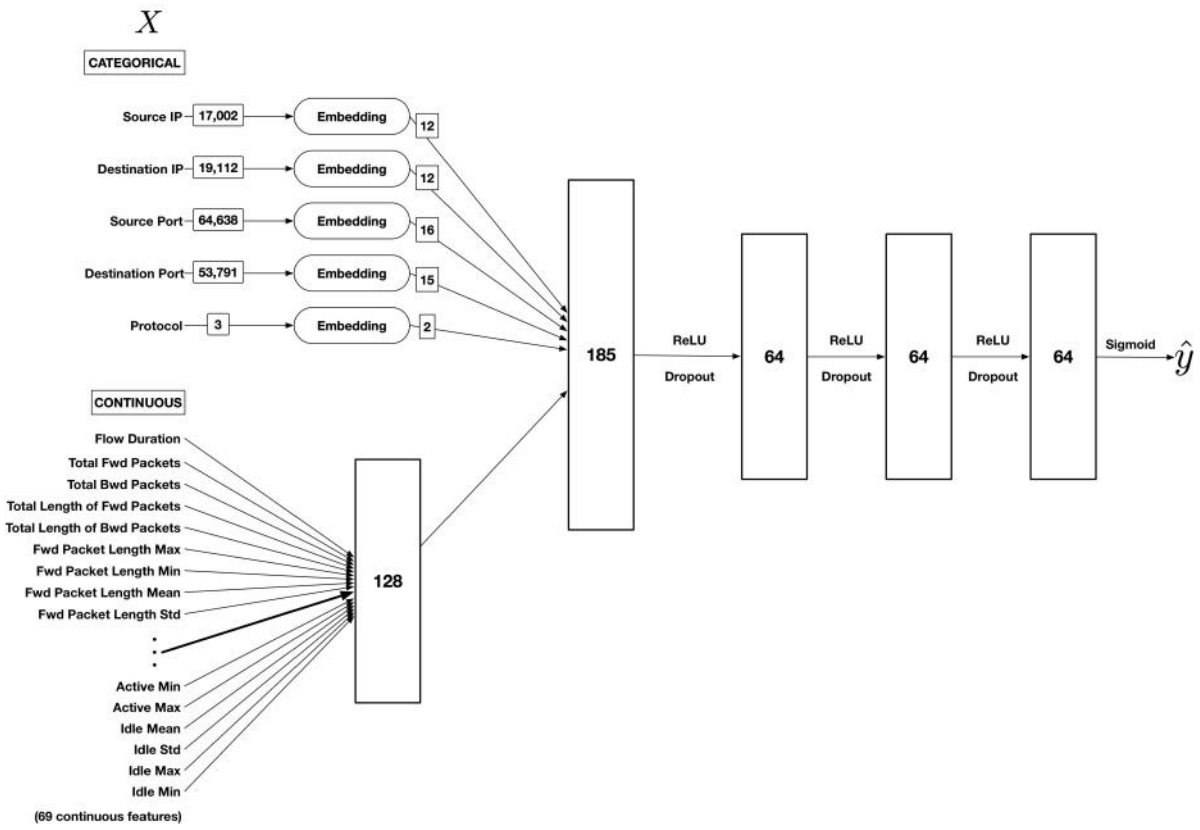


Figure 3.8: Deep Neural Network Architecture for CIC IDS 2017 Dataset

3.2.5.6 Evaluation and Results

In this section, evaluation and results of experiments are provided from using a deep feedforward fully connected neural network classifier on the CIC IDS 2017 dataset. The class distribution for the CIC IDS 2017 dataset can be seen in Figure 3.9.

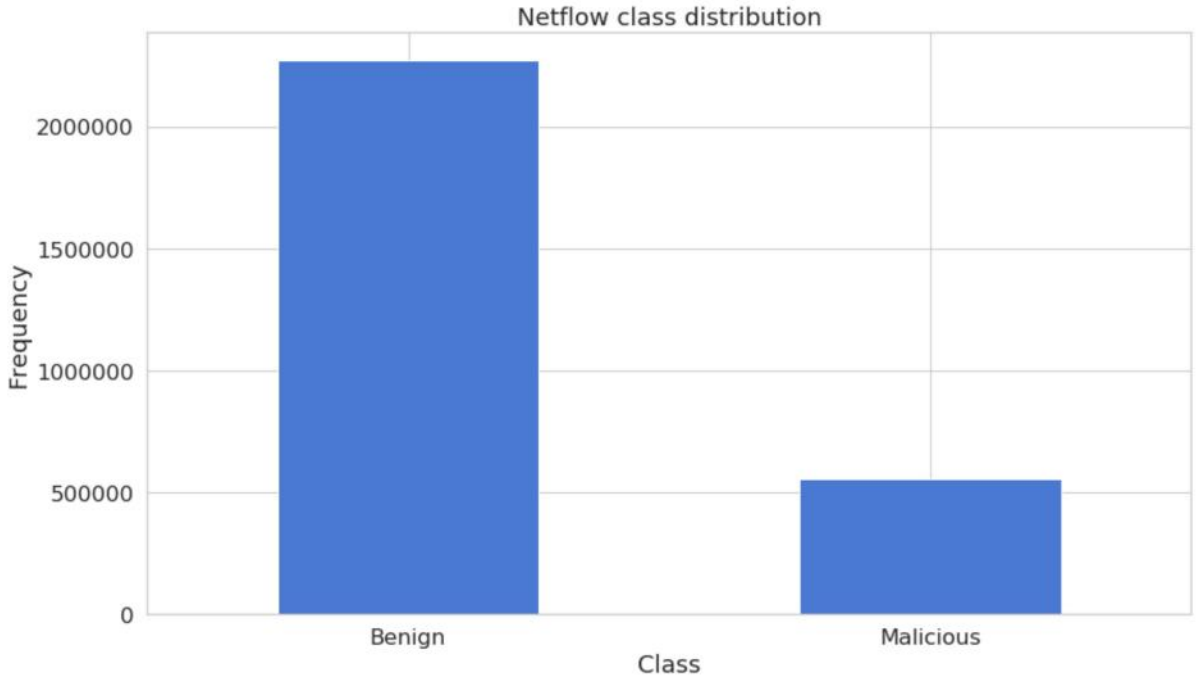


Figure 3.9: CIC IDS 2017 Class Distribution

A number of experiments and trials were run using the CIC IDS 2017 dataset, similar to the previous case study. Various neural network configurations and hyperparameters were used in order to achieve the best results, which are shown in Table 3.8. Using a neural network with three hidden layers of 64 units per layer, using the ReLU activation function for each hidden layer, and with loss being calculated using the softmax cross entropy loss function. The optimizer used in obtaining these results is the Adagrad optimizer.

As can been seen by the results, the deep neural network performs well in its ability to classify network flows as malicious and benign, achieving a low false positive rate of only 0.000127 and achieving an F1 score of 0.9980. This demonstrates that using the detailed flow statistics, in addition to using the embedding technique for the categorical variables of source/destination IP

address and ports enables this classifier to achieve acceptable results.

Insight 3. *When using embeddings for the categorical variables of Source/Destination IP Address and Ports, neural network performs exceptionally well at classifying malicious and benign flows. In addition, in experiments that omit the Source/Destination IP address features, but still utilizes detailed flow statistics, the neural network performs nearly as well, achieving an F1 Measure of* 0.9730 *in comparison to the experiment that included the IP address features, which achieved the* 0.9980 *F1 score. Therefore, training a neural network by omitting IP address features, given it contains detailed flow statistics, may be a viable way to use the trained model on a different computer network it has not seen before.*

Table 3.8: CIC IDS 2017 Evaluation Results - Metrics

Metric	**With IPs**	**Without IPs**
True Negative	$749,299$	$745,606$
False Positive	237	$3,930$
False Negative	137	$5,930$
True Positive	$183,527$	$177,734$
Area Under the Curve	0.9995	0.9812
Accuracy	0.9996	0.9894
Error Rate	0.0008	0.0106
True Positive Rate (Sensitivity, Recall, Detection Rate)	**0.9993**	**0.9677**
True Negative Rate (Specificity)	0.9997	0.9948
False Positive Rate	**0.0003**	**0.0052**
False Negative Rate	**0.0007**	**0.0323**
Precision	0.9987	0.9784
F1 Measure	**0.9990**	**0.9730**

Insight 4. *When using the IP address as a feature and representing it as a dense embedded vector at the first layer of the deep neural network, the best results are achieved. As the dense vector representation is among the first layer of the neural network, its representation gets updated at every epoch, in relation to the other features in the dataset as well as the supervised label. Therefore, it is theorized that the neural network forms a type of memory of the IP address in relation to the other features and the label, whereby it enables the highest performance for the classification task.*

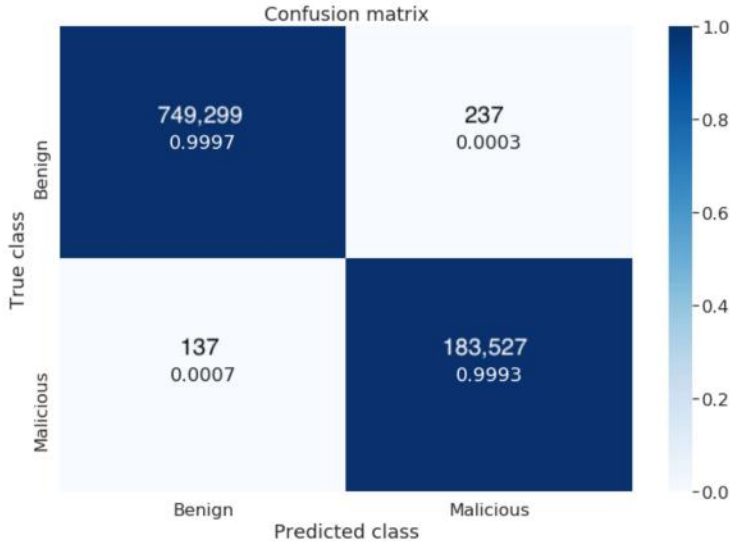


Figure 3.10: CIC IDS 2017 Confusion Matrix using Embeddings with IP Address

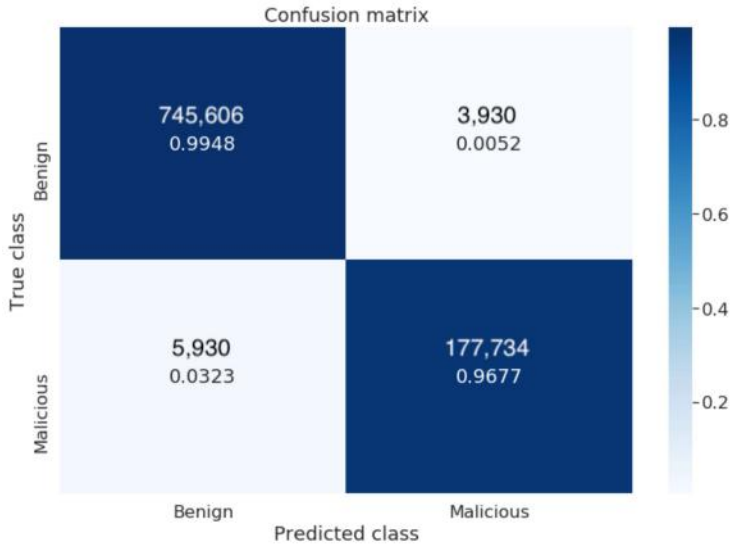


Figure 3.11: CIC IDS 2017 Confusion Matrix using Embeddings without IP Address

It should be noted, however, that the IP address feature is highly relevant to the given dataset from which it originates, and may therefore not generalize well to other datasets.

In reviewing these results, it shows that the using the IP address as a feature enables better performance of the classification task. However, in practice, the IP address can often change due to DHCP. Most often, the IP address will remain constant for the first three octets in a network environment, as this is determined by the configuration of routers and switches and is seldom changed. Instead, it is the last octet that can often change for a workstation on a network environment. Therefore, the key question is can IP address be used reliably as a feature in practice? How can the IP address be used while still accounting for its possibility to change due to DHCP? One approach is to use only the first three octets, instead of the entire IP address.

Figure 3.12 shows the results of removing the first three octets for source and destination IP address:

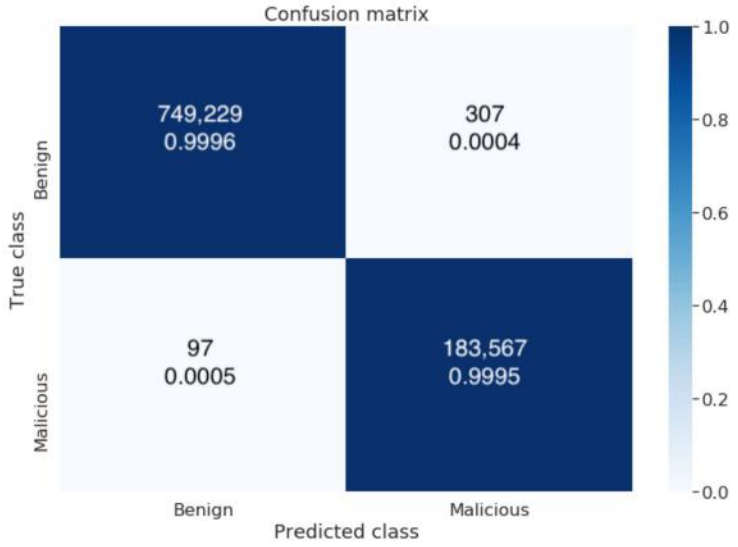


Figure 3.12: CIC IDS 2017 Confusion Matrix - First 3 Octets of IP address

Insight 5. *Using only the first three octets of the IP address performs just as well as using the full IP address. This naturally captures interactions between different local subnets, and between local and remote hosts. This methodology can be used to reliably include IP address as feature and overcome limitations imposed by DHCP.*

In addition to binary classification, the same neural network model was evaluated for multinomial classification. The results of these experiments can be seen in the confusion matrix in Figures 3.13 and 3.14. The multinomial classifier also performs well, for those categories for which it has a sufficient number of examples of each class. For categories 7-12, the confusion matrix shows low performance. However, for those categories, there is only 4, 497, 215, 7, 12, and 645 amount of support (examples) in the dataset for each respective category (see Table 3.9). Therefore, this shows that the neural network needs a sufficient number of examples to learn from in order to perform classification accurately.

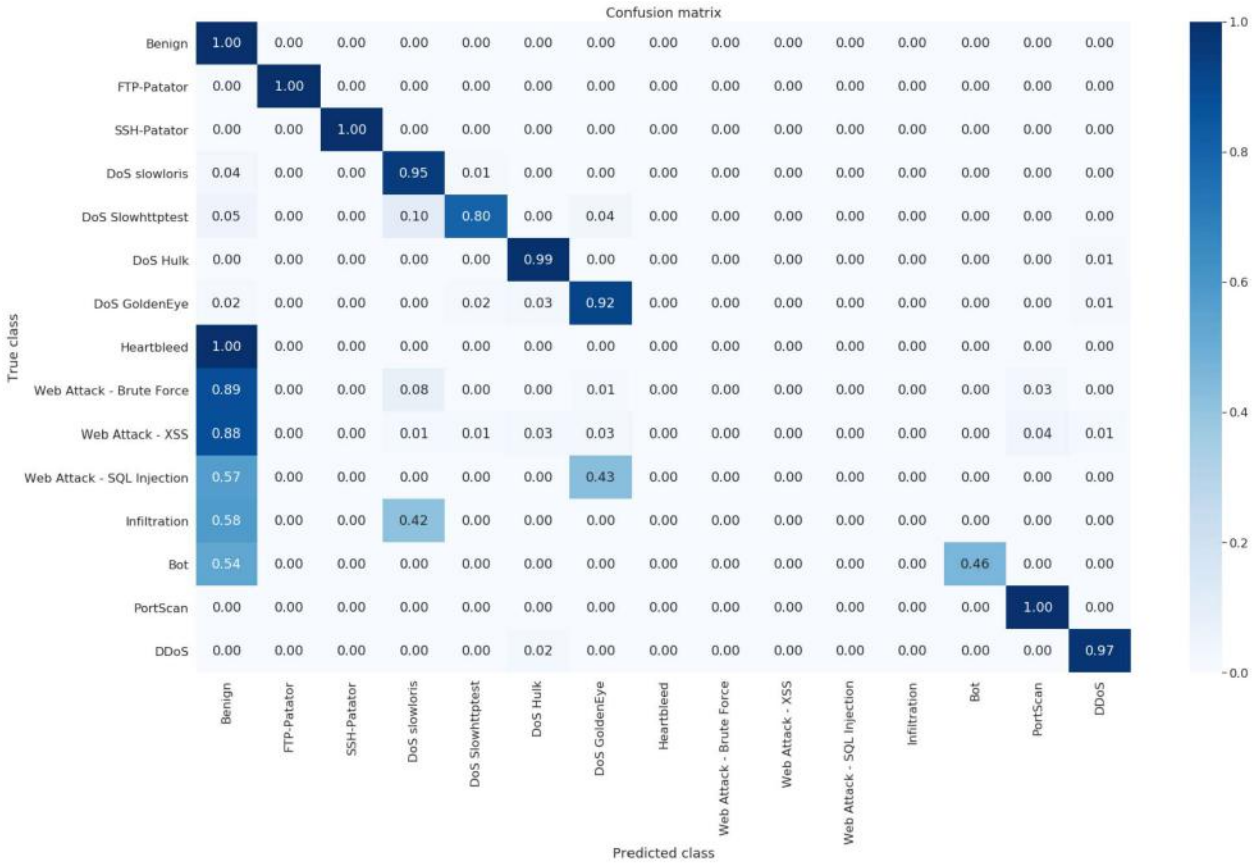


Figure 3.13: Confusion Matrix - Multinomial Classification using full IP address

As shown in Figure 3.14, the neural network performs just as well for multinomial classification using just the first 3 octets of the IP address as when using the full IP address. In some cases, such as for Bot, DoS, and DDoS, it performs just slightly better.

In comparing the two case studies, the performance of the neural network was best for the

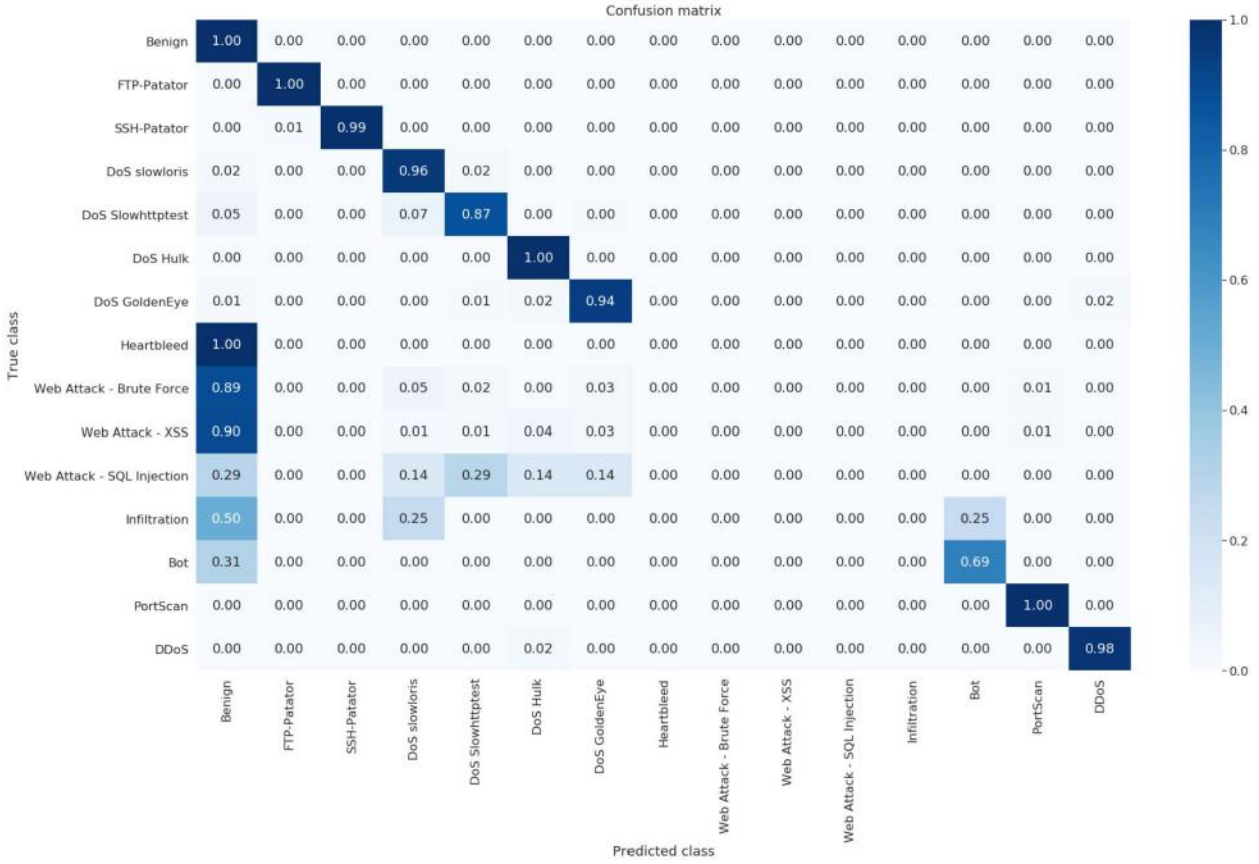


Figure 3.14: Confusion Matrix - Multinomial Classification using first 3 octets

Table 3.9: CIC IDS 2017 Multinomial Classification - Support Numbers

Class	**Support**
Benign	$749,536$
FTP-Patator	$2,619$
SSH-Patator	$1,946$
DoS Slowloris	$1,913$
DoS Slowhttptest	$1,815$
DoS Hulk	$75,941$
DoS Goldeneye	$3,397$
Heartbleed	**4**
Web Attack - Brute Force	**497**
Web Attack - XSS	**215**
Web Attack - SQL Injection	**7**
Infiltration	**12**
Bot	**645**
PortScan	$52,405$
DDoS	$42,248$

second case study which used the CIC IDS 2017 dataset. For the second case study, there was a larger amount of training data available. In particular, the second case study used a dataset that had a higher amount of malicious training examples to learn from. The first case study with the ISCX IDS 2012 dataset contained $2,545,935$ total flows, of which $68,910$ (2.91%) were malicious. The second case study with the CIC IDS 2017 dataset contained $2,830,743$ total flows, of which $557,646$ (19.70%) were malicious. In addition, the dataset for the first cased study had less features (a total of 14, consisting of 7 continuous and 7 categorical), compared to the second case study dataset (a total of 74, consisting of 69 continuous and 5 categorical), with the second dataset containing more continuous features made up of detailed flow statistics for each training example. Even when removing features from CIC IDS 2017 to make the two datasets have parity, the performance of the neural network was higher on CIC IDS 2017 with an F1 Score of 0.9990, compared to ISCX IDS 2012 with an F1 Score of 0.9491. The primary difference in this case was that CIC IDS 2017 contained a higher number of attacks to learn from. In addition, the CIC IDS 2017 dataset itself is more robust in how it was generated, as it adheres to the 11 criteria for building a robust benchmark dataset, as according to [95].

Insight 6. *Given more attack examples for the neural network to train on, the performance of the neural network increases.*

There are a number of studies that have been using the CIC IDS 2017 dataset for evaluating approaches and algorithms for network intrusion detection. Tables 3.10 and 3.11 compare other studies' results to the results achieved in this study.

3.3 Case Study Limitations

These experiments show promising results, especially when including the source and destination IP address as features. In some environments, it may be viable to include the IP address as a feature; however, in other environments it is often the case that IP addresses change due to DHCP. The experiments without IP addresses still performed well, but could not reach the level of results as when IP addresses were included as embedded categorical features. One way to overcome the

Table 3.10: CIC IDS 2017 Result Comparison [13]

Technique	Avg. DR	Avg. FPR
Hypbrid IDS - Decision Tree + Rule-based [13]	0.94475	.01145
WISARD [30]	0.48175	0.02865
Forest PA [12]	.92920	0.03550
J48 Consolidated [52]	0.92020	0.06645
LIBSVM [19]	0.54595	0.05130
FURIA [50]	0.90500	0.03165
Random Forest [13]	0.93050	0.01880
REP Tree [13]	0.91640	0.04835
MLP [13]	0.77830	0.07350
Naive Bayes [13]	0.82510	0.33455
Jrip [13]	0.93400	0.04470
J48 [13]	0.91990	0.05040
DNN with IPs	**0.9993**	**0.0003**
DNN without IPs	**0.9677**	**0.0052**

Table 3.11: CIC IDS 2017 Result Comparison [95]

Technique	Precision	Recall	F1 Score
KNN [95]	0.96	.96	.96
Random Forest (RF) [95]	0.98	0.97	.97
ID3 [95]	.98	0.98	.98
Adaboost [95]	0.77	0.84	.77
MLP [95]	**0.77**	**0.83**	**.76**
Naive-Bayes [95]	0.88	0.04	.04
Quadratic Discriminant Analysis (QDA) [95]	0.97	0.88	.92
DNN with IPs	**0.9987**	**0.9993**	**0.9990**
DNN without IPs	**0.9784**	**0.9677**	**0.9730**

limitation of IP addresses changing due to DHCP is to use only the first 3 octets of the IP address as a feature. Experiments show that this methodology performs just as well as using the complete 4 octets of the IP address.

3.4 Summary

Deep Neural Networks are effective when working with tabular data that consists of lots of examples, and with categorical variables of high cardinality, which are present in the domain of Cybersecurity and Network Intrusion Detection. The technique for embedding high cardinality categorical variables, which are common in cybersecurity data, leverages the power of deep neural networks to achieve better results than other leading techniques without needing to perform much in the way of hand-engineered features, especially when the amount of training data is larger. Furthermore, it was found that using the IP Address as an embedded categorical feature enabled the best performance for the deep neural network. It is theorized that by using the embedding technique and updating the weights of the IP address representation at each epoch, the neural network forms a type of inherent memory about the IP address feature in relation to the other features and the given label. Additionally, high performance was still achieved when using only the first three octets of the IP address are used, as opposed to the full IP address, allowing for a more robust feature that withstands changes that would typically be caused by updates from a DHCP server.

In this chapter, two case studies were presented that evaluated the use of deep fully connected neural networks for classifying network flows as benign or malicious. To perform the case studies, two recent benchmark datasets were used. The experiments showed results that outperformed other leading techniques when using deep neural networks for classification of network flows as benign or malicious. Further, it was shown that the use of flow statistics generated by the CICFlowMeter, in addition to a technique of dimensionality reduction of the categorical variables using the embedding technique, enabled the deep neural network to achieve an F1 measure of 0.9990, a detection rate of 0.9993, and a false positive rate of only 0.0003.

Chapter 4: USING AUTOENCODERS FOR NETWORK INTRUSION DETECTION

4.1 Introduction

This chapter looks at using unsupervised learning with an autoencoder for detecting anomalies for the task of network intrusion detection.

In this chapter, the following key contributions are made:

1. Application of unsupervised learning using deep autoencoder for anomaly detection, and applying to field of network intrusion detection. Using only normal flows, this chapter evaluates how an autoencoder can be trained to detect anomalous flows based on reconstruction error.

2. Validation of the approach by evaluating results using CIC IDS 2017 benchmark intrusion detection dataset. As this benchmark dataset was captured from real network traffic as opposed to being simulated or replayed from packet captures, it is representative of normal (benign) modern day network traffic. Therefore, it is useful for evaluation where the autoencoder trains only on benign traffic flows. In addition, this dataset contains malicious flows generated by modern-day attack scenarios, which are representative of attacks seen on typical enterprise networks.

4.2 Background on Autoencoder

This chapter looks at using autoencoders for the task of network intrusion detection. An autoencoder is a type of neural network that is trained in such a way that it aims to copy its input to its output [45]. An autoencoder is designed so that it reproduces its input at the output layer. One of the main differences between an autoencoder and a multilayer perceptron, or deep neural network, is that the number of input neurons is equal to the number of output neurons. In addition, instead of generating an output of $\hat{y}$ (i.e. representing a class of *benign* or *malicious* as in the previous case

study), the output generated is X', a reconstruction of the original input X. An autoencoder's primary utility is to find a lower dimensional, latent space representation of the input data. It does this in a non-linear fashion, unlike other popular dimensionality reduction techniques such as Principle Component Analysis (PCA). Therefore, a key application of autoencoders is to use them for pre-training a neural network, and thus use them to improve the performance of supervised learning. In addition, they can be used to power an anomaly detection system.

The key idea behind autoencoders is to take an input vector and map it to a latent space (encode) of lower dimensionality, then from that latent space, map it back to an output (decode) using the low dimensional latent space as input. The module that is capable of taking this latent space of lower dimensions and mapping it back to an output of with the same higher dimensions of the original input is called a *decoder*. This technique is commonly used for image generation; however, this chapter looks at applying this technique for the purposes of network intrusion detection, and classifying network flows as either malicious or benign based on reconstruction error.

Variational autoencoders are one type of autoencoder that are well suited for learning latent spaces that are well structured, whereby specific directions are able to encode useful axis of variation of the data. Therefore, VAEs can be well suited for the task of network intrusion detection. Generative Adversarial Networks (GANs) are another type of generative deep learning that maps an input space of high dimensionality into a lower dimensional latent space; the main difference with a GAN is that the latent space it creates and uses tends not have as much structure and continuity as one created by a VAE. This chapter uses a standard autoencoder which functions to reconstruct its inputs from a compressed representation.

Figure 4.1 shows an example standard autoencoder neural network architecture, where the number of input neurons is equal to the number of output neurons. The way an autoencoder works is that it takes an input vector and maps it to a latent vector space using an encoder module, then decodes back to an output vector using a decoder module. The decoded output has the same dimensions as the original input vector. Then, the neural network is trained using target data that is the same as the original input vector, so that it learns how to reconstruct the original inputs. Using

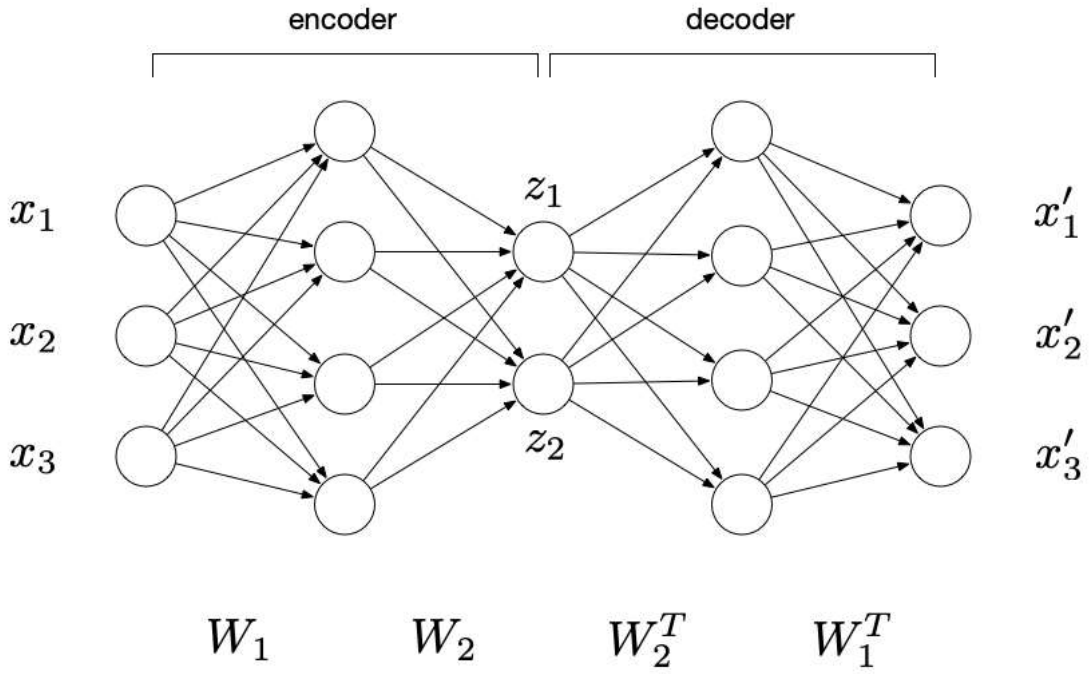


Figure 4.1: Example neural network structure for Autoencoder

this process, and by placing certain constraints on the encoded output, the autoencoder can learn interesting latent representations of the original input data. The most common constraint being limiting the encoded output to be of low dimensions and sparse (mostly containing zeros). This causes the encoder to enforce compression on the input data, resulting in it having a lower amount of bits.

One way to think about an autoencoder is to consider that it is simply a neural network that is making predictions about itself, instead of training itself to predict labeled output data. In other words, instead of calling `model.train(X, Y)` followed by `model.predict(X)`, an autoencoder is the process of a neural network learning to model its own inputs, by calling `model.train(X, X)`.

For the cost function, since the inputs are real values that are trying to be predicted, instead of classification where the network is trying to predict a binary value, the squared error can be used as the cost function, and it can be treated as a regression problem. Alternatively, cross-entropy can also be used as the loss function, with X being a value in the range $[0, 1]$. For the hidden layer and

output layer, the activation function can be either sigmoid or ReLU, so that the outputs are always going to be within the range $[0, 1]$.

For the autoencoder, there are different bias terms for the hidden layer b_h and the output layer b_o. One thing to consider when using an autoencoder is the notion of shared weights. Therefore, for the weight at the output layer, it is just a transpose of the weight of the input layer. This also serves as a type of regularization, as it reduces the number of parameters, which in turn reduces the probability of overfitting.

$$Z = sigmoid(X.dot(W) + b_h) \tag{4.1}$$

$$\hat{X} = sigmoid(Z.dot(W^T) + b_o) \tag{4.2}$$

The objective function for the autoencoder used in this case study will be the squared error. Written out in terms of weights and inputs, this function is shown in equation 4.3.

$$J = |X - \hat{X}|_F^2 = |X - sigmoid(sigmoid(X \cdot W)W^T)|_F^2 \tag{4.3}$$

4.3 Case Study with CIC IDS 2017 Dataset

In this case study, using unsupervised deep learning with an autoencoder is evaluated on the the CIC IDS 2017 dataset. The ability for the autoencoder to be used as an anomaly detection mechanism is demonstrated when training only on the benign traffic flows. Autoencoders can learn a compressed representation of the input data, where the output of the autoencoder is a reconstruction of the input data in its most efficient form. By minimizing the error when reconstructing the normal input, the neural network learns to modify its weights for reconstructing benign flows. Then when the neural network sees malicious flows, the hypothesis is that the reconstruction error will be higher, as the malicious flows are anomalous when compared to the benign flows in terms of encoding and decoding.

The configuration of the autoencoder used in this case study is shown in Figure 4.2.

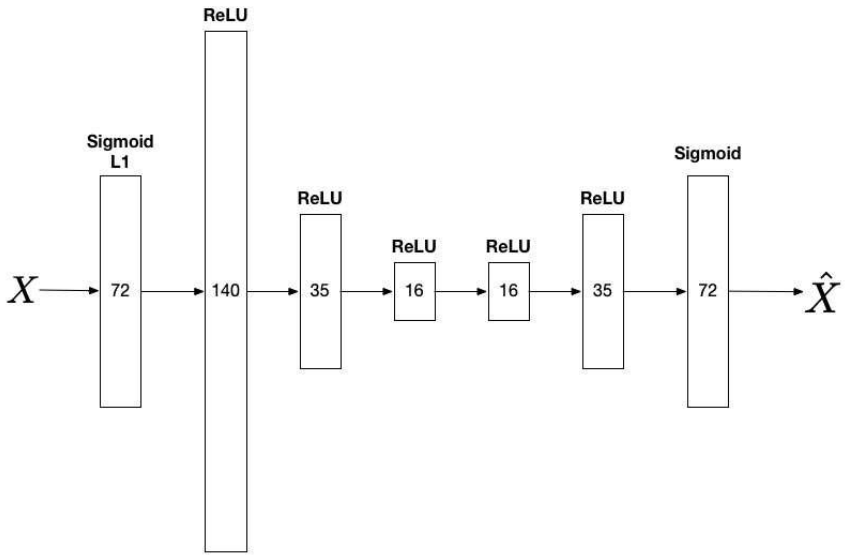


Figure 4.2: CIC IDS 2017 Autoencoder Configuration

The results of this experiment are shown in the tables and figures below. As shown in Figure 4.3, there is a higher reconstruction error for the malicious traffic flows as compared to the benign flows.

Insight 7. *The autoencoder is trained to reconstruct inputs from a latent space representation based on only benign flows. Therefore, when it receives malicious flows as input, it is more difficult for the neural network to reconstruct the malicious flows, thus resulting in a higher reconstruction error. When there is sufficient amount of benign training examples, an autoencoder can be used to detect anomalous flows based on reconstruction error and setting a threshold.*

Looking at this further, when setting a threshold of 0.03 for the reconstruction error, the neural network predicts malicious and benign flows with the metrics shown in Table 4.1. As can be seen, a high number of malicious flows are detected as such, with a minimal number of false positives. However, there is a relatively large number of false negatives, malicious flows that are classified as benign. This can also be observed in Figure 4.3, where a number of malicious flows have a similar reconstruction error as benign flows. Notwithstanding, the autoencoder shows effectiveness in being able to detect malicious flows it has never seen before, and does so with a low false positive rate. However, it does so at the cost of a high false negative rate, whereby when setting a given

threshold, the neural network sees a large amount of malicious flows as benign.

Insight 8. *Autoencoders can be effective as anomaly detection mechanism for network intrusion detection (with training on normal traffic only). They can be used to detect malicious flows that have never been encountered before with a low false positive rate. Importantly, since it exhibits a low false positive rate, this technique can be useful for bringing the anomalies to the attention of a human analyst for review. However, this it at the cost of a high false negative rate, and should be used along with other supervised techniques as part of a defense-in-depth strategy.*

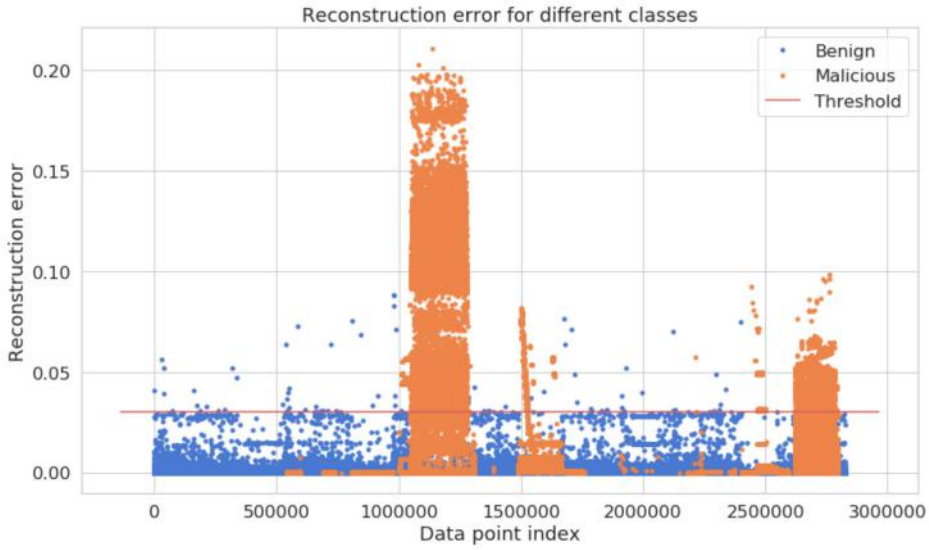


Figure 4.3: CIC IDS 2017 Autoencoder Reconstruction Error with Threshold

4.4 Case Study Limitations

As shown in this case study, autoencoders can be useful for anomaly detection in terms of achieving a low false positive rate. Using only one class of benign flows, autoencoders can learn how to reconstruct the benign class. Therefore, when new flows are observed, if the reconstruction loss is above a set threshold, anomalies can be detected. The limitations here are that there must be data existing that is known to be normal or benign. In addition, the experiments show that a threshold can be set where there are minimum number of false positives, yet still a relatively large number of false negatives. However, this is may be preferred to have a lower number of false

Table 4.1: Autoencoder Evaluation Results - CIC IDS 2017

Metric	**Result**
True Negative	681,307
False Positive	89
False Negative	128,065
True Positive	38,902
Area Under the Curve	0.6164
Accuracy	0.8489
Error Rate	0.1512
True Positive Rate (Sensitivity, Recall, Detection Rate)	**0.2330**
True Negative Rate (Specificity)	0.9999
False Positive Rate	**0.00013**
False Negative Rate	**0.7670**
Precision	0.9977
F1 Measure	**0.3778**

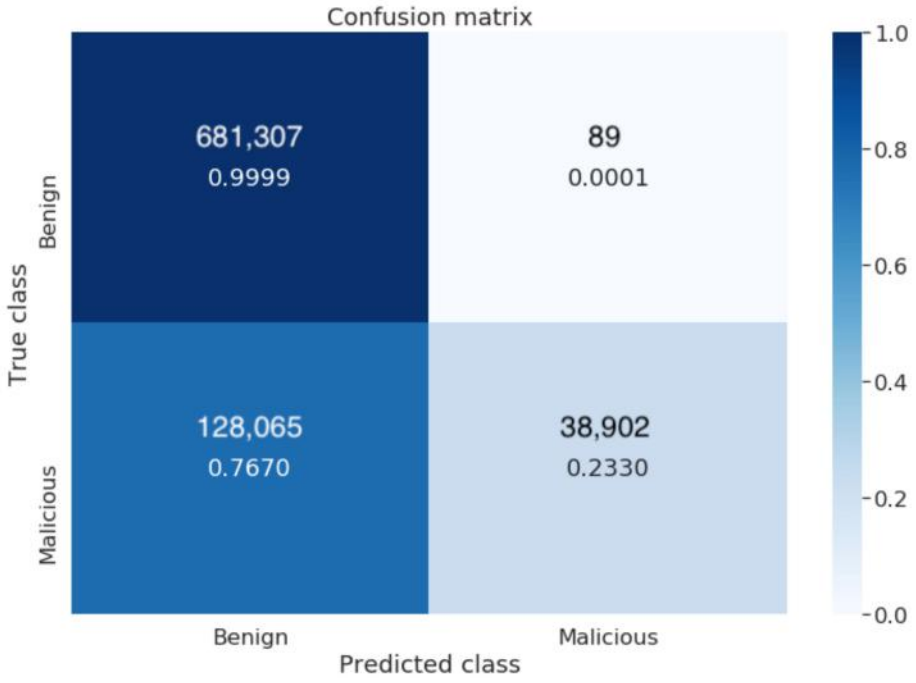


Figure 4.4: CIC IDS 2017 Autoencoder Confusion Matrix

positives than false negatives, as once an anomalous flow is detected, it is a signal to an ongoing attack that is causing multiple malicious flows in the network. In addition, from a practitioner's perspective, it may be preferable for the human analyst to be reviewing anomalous alerts that have a high probability of being true positive. Importantly, this shows that autoencoders as an anomaly detection mechanism are just one technique that can be used to surface never-before-seen malicious activity, in combination with other tools within a defense-in-depth strategy. It is also important to note that this case study used only the flow statistics as features. This is one contributing factor to the lower performance in terms of the experiments exhibiting a high false negative rate. In the previous chapter, it was shown that the IP address had a strong impact on the performance of the neural network for supervised learning. Therefore, future work on autoencoders should investigate the use of incorporating categorical variables along with the flow statistics to improve performance, especially in achieving a lower false negative rate.

Insight 9. *It is important to note that not all anomalous flows indicate malicious flows. Rather, these are just flows that have an inherently different structure than the benign flows that the neural network was trained on. Regardless, the experiments in this case study resulted in a low false positive rate, and the majority of the anomalous flows are in fact verified as malicious. Incorporating categorical variables is a good next step for improving the performance of the autoencoder in terms of false negative rate.*

4.5 Summary

In this chapter, the use of an autoencoder for anomaly detection was evaluated on the CIC IDS 2017 benchmark intrusion detection dataset. It was shown that when training an autoencoder on only benign flows, a threshold can be set on the reconstruction error, and anomalies can be detected. These anomalies were verified to be attack traffic (with a low false positive rate) which the neural network had never seen before. This shows a use of an unsupervised deep neural network for the use case where there is lots of labeled *benign* network flows, and how never-before-seen *malicious* traffic flows can then be detected as anomalies based on reconstruction error.

Chapter 5: RELATED WORK

Intrusion detection is an important field of cybersecurity data analytics, which is a core pillar underlying the Cybersecurity Dynamics framework [109–112] that aims to understand, model, characterize and quantify cybersecurity from a holistic perspective. The other two pillars are known as first-principle modeling and analysis [29, 49, 61, 65, 68, 106, 108, 111, 113–116, 123, 124] and cybersecurity metrics [20–22, 25, 26, 34, 40, 43, 71, 73, 88].

There are many sub-fields in cybersecurity data analytics. The present Thesis falls into prescriptive cybersecurity data analytics aim to detect network attacks, which are then treated properly by the defender; therefore, this sub-field of cybersecurity data analytics may be called *reactive cybersecurity data analytics* (Figure 5.1).

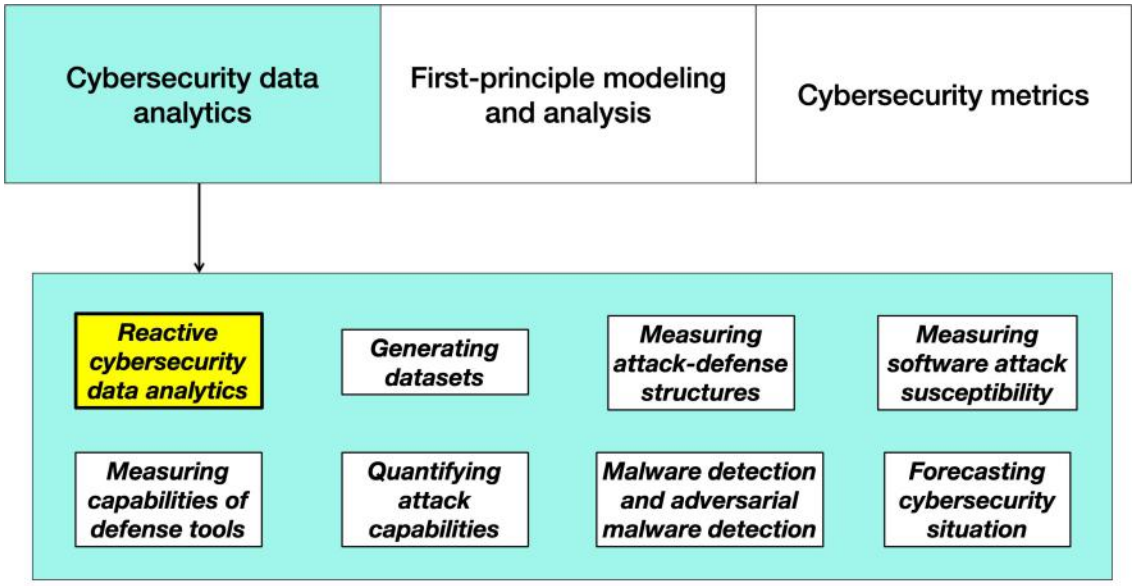


Figure 5.1: Cybersecurity Dynamics Framework

Intrusion detection can be host-based or network-based. The present Thesis falls into the latter scenario. There have been many approaches to Network Intrusion Detection developed over the years. The general study of anomaly detection dates back as early as the 19th century with origins in the statistics community [36]. The study of intrusion detection for cybersecurity has roots in a paper published by Denning in 1987 [31]. Fiore et. al. [41] explored the use of a semi-

supervised model for network intrusion detection, using a Discriminative Restricted Boltzmann Machine. They trained their classifier on "normal" traffic only, with the goal of being able to detect anomalous behaviors that may evolve and change over time. They argue that it is often difficult to train a model with anomalous training samples *a priori*. For their experiments, they used the classic KDD'99 dataset. While this is a common benchmark dataset that has been used in many research papers in the field, it has also been criticized for its lack of relevancy with current day attacks. Sommer et. al suggest that this dataset may be better suited for only providing additional validation and cross-checking of a novel technique [97].

Other sub-field of cybersecurity data analytics (shown in Figure 5.1) aim to understand, characterize, and predict cybersecurity behaviors and events exhibited by cybersecurity data, including: generating datasets [89, 95, 96]; measuring attack-defense structures in complex networks [21,22,44]; measuring the susceptibility of software to attacks [3,8,9,47,62–64,80,117,118]; measuring capabilities of defense tools [20,34,39,71]; quantifying capabilities of attacks [70,102,103]; malware detection and adversarial malware detection [53, 56–60, 72, 75–79, 92, 93, 102, 104, 120]; predicting or forecasting the cybersecurity situation based on the monitored cybersecurity data streams [23, 90, 91, 105, 107, 121, 122].

Chapter 6: DISCUSSION AND CONCLUSION

6.1 Conclusion

Deep learning models are able to learn features on their own by having lots of training data available – labeled training data. Neural networks are capable of extracting features directly out of the data itself, instead of relying on hand-engineered features. Hand-engineered features are still important, as they allow for the solving of the problem with less resources and in a more elegant manner. Good hand-engineered features based on domain knowledge also help when there is far less data available to train on.

In the field of network intrusion detection, there is an abundance of data available from packet captures of network flows. These packet captures can be converted into network flows that contain rich metadata about the statistics of each flow, which are composed of the captured packet data. These flows are structured in the form of tabular data, and contain both continuous and categorical features. The categorical features for network flows are often times high-dimensional, as was shown in the two case studies in this work. Deep learning can be a powerful tool when used with tabular data, especially when it contains categorical features of high dimensionality. Using an embedding technique, these high cardinality categorical variables can be converted to dense floating point vectors that are orders of magnitude smaller.

Using two recent benchmark intrusion detection datasets, the effectiveness of using feedforward fully connected deep neural networks for classifying malicious and benign flows using both supervised and unsupervised techniques was explored. It was shown that supervised learning using deep neural networks is highly effective for network intrusion detection. Using embeddings for the high dimensional categorical variables of source and destination IP addresses and ports, the neural network performs exceptionally well at classifying malicious and benign flows using supervised learning. The performance of the neural network was also effective on the UNB ISCX 2012 dataset, which had less examples and less features per example (specifically there were far fewer flow statistics available). In particular, it performed better than other leading techniques as

shown in table 3.6. Furthermore, on the CIC IDS 2017 dataset, which contained more examples for each class, and many more features composed of flow statistics, the performance of the neural network increased. Also, a technique for embedding IP address features using only the first three octets was discovered that overcomes the limitations imposed by DHCP – using only the first three octets performed just as well as using the full IP address.

In this work, it was also shown that unsupervised learning using autoencoders can be a useful approach and use of deep neural networks when there exists lots of data for one class. By training the neural network to reconstruct its inputs, it can learn the latent representation of the data based on the reconstruction error. When the trained autoencoder receives new inputs it has never seen before, it reconstructs those inputs based on its prior learned representation. If there is a high reconstruction error, that input is deemed anomalous (above a certain threshold). It was shown in Chapter 4 that the autoencoder reconstructed malicious inputs to contain a high reconstruction error and thus be deemed anomalies, with a low false positive rate. Autoencoders can be effective in modeling the normal flows, and by setting a threshold on the reconstruction error, can alert when anomalous (malicious) flows are detected due to a high reconstruction error.

Deep neural networks can be used for both supervised and unsupervised learning. In the case studies performed, it was shown that a deep feedforward fully connected neural network can achieve excellent results for supervised learning. It was also shown that autoencoders can be used for anomaly detection when they are trained on benign flows, enabling the discovery of never-before-seen malicious flows with a low false positive rate.

6.2 Future Work

There are a number of future directions that can be pursued, including the following:

1. Use time domain information as an embedded categorical variable. For example, use different resolutions such as day of week, hour of day, or minute of hour as embedded categorical variables. This is highly dependent on the signature of an attack, but if an adversary chooses to attack at a certain time, this may help increase precision and recall. In addition, evaluate

these features on their ability to improve the performance of the autoencoder.

2. Perform additional strategies for embedding the IP address as a categorical variable. For environments where the IP address frequently changes, other techniques for including the IP address as a categorical embedded feature may be useful.

3. Explore Recurrent Neural Network structures and how they can be used to leverage the time domain for classification of benign and malicious flows.

4. Leverage categorical variables with the autoencoder approach to improve its performance in detecting anomalous flows. The current case study in this Thesis which evaluated an autoencoder utilized only the continuous flow statistics as features.

5. Perform a systematic evaluation of varying features and hyperparameter optimizations to determine the deep neural network configuration that produces the best results (both supervised and unsupervised) on the benchmark datasets used in this Thesis.

6. UNB CIC recently released a new dataset in 2018 that was created using the same framework for building the CICIDS2017 dataset [95]. Therefore, training a deep neural network on one computer network traffic environment, and determining how well it generalizes to classifying flows on an entirely different environment will be a good next step. It would be of interest to determine algorithms and techniques in which a neural network trained on one network intrusion detection dataset can be generalized to perform effectively on a new dataset coming from a different computer network environment.

Appendix A: DATASET DETAILS

A.1 CIC IDS 2017 Dataset Features

Table A.1: Description of Features for CIC IDS 2017 Dataset

No.	Feature	Description	Type
1	FlowID	Unique identifier for flow record	N/A
2	SourceIP	Source IP address	Categorical
3	DestinationIP	Destination IP address	Categorical
4	SourcePort	Source Port number	Categorical
5	DestinationPort	Destination Port number	Categorical
6	Protocol	Protocol type (e.g. tcp, udp)	Categorical
7	Feduration	Duration of the flow in Microsecond	Continuous
8	total_fpackets	Total packets in the forward direction	Continuous
9	total_bpackets	Total packets in the backward direction	Continuous
10	total_fpktl	Total size of packet in forward direction	Continuous
11	total_bpktl	Total size of packet in backward direction	Continuous
12	min_fpktl_in	Minimum size of packet in forward direction	Continuous
13	min_bpktl	Minimum size of packet in backward direction	Continuous
14	max_fpktl	Maximum size of packet in forward direction	Continuous
15	max_bpktl	Maximum size of packet in backward direction	Continuous
16	mean_fpktl	Mean size of packet in forward direction	Continuous
17	mean_bpktl	Mean size of packet in backward direction	Continuous
18	std_fpktl	Standard deviation size of packet in forward direction	Continuous

Table A.1: Continued

19	std_bpktl	Standard deviation size of packet in backward direction	Continuous
20	total_fiat	Total time between two packets sent in the forward direction	Continuous
21	total_biat	Total time between two packets sent in the backward direction	Continuous
22	min_fiat	Minimum time between two packets sent in the forward direction	Continuous
23	min_biat	Minimum time between two packets sent in the backward direction	Continuous
24	max_fiat	Maximum time between two packets sent in the forward direction	Continuous
25	max_biat	Maximum time between two packets sent in the backward direction	Continuous
26	mean_fiat	Mean time between two packets sent in the forward direction	Continuous
27	mean_biat	Mean time between two packets sent in the backward direction	Continuous
28	std_fiat	Standard deviation time between two packets sent in the forward direction	Continuous
29	std_biat	Standard deviation time between two packets sent in the backward direction	Continuous
30	fpsh_cnt	Number of times the PSH flag was set in packets traveling in the forward direction (0 for UDP)	Continuous

Table A.1: Continued

31	bpsh_cnt	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)	Continuous
32	furg_cnt	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)	Continuous
33	burg_cnt	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)	Continuous
34	total_fhlen	Total bytes used for headers in the forward direction	Continuous
35	total_bhlen	Total bytes used for headers in the backward direction	Continuous
36	fPktsPerSecond	Number of forward packets per second	Continuous
37	bPktsPerSecond	Number of backward packets per second	Continuous
38	flowPktsPerSecond	Number of flow packets per second	Continuous
39	flowBytesPerSecond	Number of flow bytes per second	Continuous
40	min_flowpktl	Minimum length of a flow's packets	Continuous
41	max_flowpktl	Maximum length of a flow's packets	Continuous
42	mean_flowpktl	Mean length of a flow's packets	Continuous
43	std_flowpktl	Standard deviation length of a flow's packets	Continuous
44	min_flowiat	Minimum inter-arrival time of packet	Continuous
45	max_flowiat	Maximum inter-arrival time of packet	Continuous
46	mean_flowiat	Mean inter-arrival time of packet	Continuous
47	std_flowiat	Standard deviation inter-arrival time of packet	Continuous

Table A.1: Continued

48	flow_fin	Number of packets with FIN	Continuous
49	flow_syn	Number of packets with SYN	Continuous
50	flow_rst	Number of packets with RST	Continuous
51	flow_psh	Number of packets with PUSH	Continuous
52	flow_ack	Number of packets with ACK	Continuous
53	flow_urg	Number of packets with URG	Continuous
54	flow_cwr	Number of packets with CWE	Continuous
55	flow_ece	Number of packets with ECE	Continuous
56	downUpRatio	Download and upload ratio	Continuous
57	avgPacketSize	Average size of packet	Continuous
58	fAvgSegmentSize	Average size observed in the forward direction	Continuous
59	fAvgBytesPerBulk	Average number of bytes bulk rate in the forward direction	Continuous
60	fAvgPacketsPerBulk	Average number of packets bulk rate in the forward direction	Continuous
61	fAvgBulkRate	Average number of bulk rate in the forward direction	Continuous
62	bAvgSegmentSize	Average size observed in the backward direction	Continuous
63	bAvgBytesPerBulk	Average number of bytes bulk rate in the backward direction	Continuous
64	bAvgPacketsPerBulk	Average number of packets bulk rate in the backward direction	Continuous

Table A.1: Continued

65	bAvgBulkRate	Average number of bulk rate in the backward direction	Continuous
66	sflow_fpacket	The average number of packets in a sub flow in the forward direction	Continuous
67	sflow_fbytes	The average number of bytes in a sub flow in the forward direction	Continuous
68	sflow_bpacket	The average number of packets in a sub flow in the backward direction	Continuous
69	sflow_bbytes	The average number of bytes in a sub flow in the backward direction	Continuous
70	min_active	Minimum time a flow was active before becoming idle	Continuous
71	mean_active	Mean time a flow was active before becoming idle	Continuous
72	max_active	Maximum time a flow was active before becoming idle	Continuous
73	std_active	Standard deviation time a flow was active before becoming idle	Continuous
74	min_idle	Minimum time a flow was idle before becoming active	Continuous
75	mean_idle	Mean time a flow was idle before becoming active	Continuous
76	max_idle	Maximum time a flow was idle before becoming active	Continuous

Table A.1: Continued

77	std_idle	Standard deviation time a flow was idle before becoming active	Continuous
78	Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction	Continuous
79	Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction	Continuous
80	Act_data_pkt_forward	Count of packets with at least 1 byte of TCP data payload in the forward direction	Continuous
81	min_seg_size_forward	Minimum segment size observed in the forward direction	Continuous
82	packet_length_variance	Variance of packet length	Continuous
83	label	Label indicating whether flow is benign or one of fourteen different attack types	N/A

BIBLIOGRAPHY

[1] KDD Cup Dataset. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[2] RFC 7011 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, 2013. https://tools.ietf.org/html/rfc7011.

[3] Rough Audit Tool for Security, 2014. https://code.google.com/archive/p/rough-auditing-tool-for-security/.

[4] Slowloris HTTP DoS, Apr 2015. https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris/.

[5] ArchLinux User Repository - Brutessh, Jun 2016. https://aur.archlinux.org/packages/brutessh/.

[6] What is IPFIX - Netflow's main Contender in Traffic Analysis, Jul 2016. https://www.pcwdld.com/what-is-ipfix.

[7] Internet Protocol Numbers, 2017. https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.

[8] Checkmarx, 2018. https://www.checkmarx.com/.

[9] FlawFinder, 2018. http://www.dwheeler.com/flawfinder.

[10] U.S. Department of Homeland Security: Cybersecurity Strategy, 2018. https://www.dhs.gov/publication/dhs-cybersecurity-strategy.

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore,

Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from `https://tensorflow.org/`.

[12] Md Nasim Adnan and Md Zahidul Islam. Forest pa: Constructing a decision forest by penalizing attributes used in previous trees. *Expert Systems with Applications*, 89:389–403, 2017.

[13] Ahmed Ahmim, Leandros Maglaras, Mohamed Amine Ferrag, Makhlouf Derdour, and Helge Janicke. A novel hierarchical intrusion detection system based on decision tree and rules-based models. *arXiv preprint arXiv:1812.09059*, 2018.

[14] Mark Allman, Mike Bennett, Martin Casado, Scott Crosby, Jason Lee, Boris Nechaev, Ruoming Pang, Vern Paxson, and Brian Tierney. Lbnl/icsi enterprise tracing project, 2005.

[15] Adel Ammar. A decision tree classifier for intrusion detection priority tagging. *Journal of Computer and Communications*, 3(04):52, 2015.

[16] Buse Atli et al. Anomaly-based intrusion detection by modeling probability distributions of flow characteristics. 2017.

[17] R Bace and P Mell. Intrusion detection systems, national institute of standards and technology (nist). *Technical Report 800-31*, 2001.

[18] M.A. Boden. *Artificial Intelligence and Natural Man*. Computer science/psychology. Basic Books, 1977.

[19] Chih-Chung Chang and Chih-Jen Lin. Libsvm: Alibraryforsupportve ctormachines. *Availableat: http://www. csie. ntu. edu. tw/scjlin/libsvm*, 2001.

[20] John Charlton, Pang Du, Jin-Hee Cho, and Shouhuai Xu. Measuring relative accuracy of malware detectors in the absence of ground truth. In *IEEE Military Communication Conference (MILCOM 2018)*, 2018.

[21] Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. Quantifying the security effectiveness of firewalls and dmzs. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security (HoTSoS'2018)*, pages 9:1–9:11, 2018.

[22] Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. Quantifying the security effectiveness of network diversity: poster. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security (HoTSoS'2018)*, page 24:1, 2018.

[23] Yu-Zhong Chen, Zi-Gang Huang, Shouhuai Xu, and Ying-Cheng Lai. Spatiotemporal patterns and predictability of cyberattacks. *PLoS One*, 10(5):e0124472, 05 2015.

[24] J. Cho, S. Xu, P. Hurley, M. Mackay, T. Benjamin, and M. Beaumont. Stram: Measuring the trustworthiness of computer-based systems. *ACM Comput. Surv. (accepted for publication)*.

[25] J. Cho, S. Xu, P. Hurley, M. Mackay, T. Benjamin, and M. Beaumont. Stram: Measuring the trustworthiness of computer-based systems. manuscript in submission, 2018.

[26] Jin-Hee Cho, Packtrick Hurley, and Shouhuai Xu. Metrics and measurement of trustworthy systems. In *IEEE Military Communication Conference (MILCOM 2016)*, 2016.

[27] F. Chollet. *Deep Learning with Python*. Manning Publications Company, 2018.

[28] François Chollet et al. Keras, 2015. `https://keras.io`.

[29] Gaofeng Da, Maochao Xu, and Shouhuai Xu. A new approach to modeling and analyzing security of networked systems. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security (HotSoS'14)*, pages 6:1–6:12, 2014.

[30] Massimo De Gregorio and Maurizio Giordano. An experimental evaluation of weightless neural networks for multi-class classification. *Applied Soft Computing*, 72:338–354, 2018.

[31] D E Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987.

[32] Georgios Drakos. Entity Embeddings of Categorical Variables in Neural Networks, 2018. https://towardsdatascience.com/decoded-entity-embeddings-of-categorical-variables-in-neural-networks-

[33] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *ICISS 2016*, 2016.

[34] Pang Du, Zheyuan Sun, Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. Statistical estimation of malware detection metrics in the absence of ground truth. *IEEE Trans. Information Forensics and Security*, 13(12):2965–2980, 2018.

[35] Sumeet Dua and Xian Du. *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications, Boston, MA, USA, 1st edition, 2011.

[36] FY Edgeworth. Xli. on discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 23(143):364–375, 1887.

[37] Laurene V Fausett et al. *Fundamentals of neural networks: architectures, algorithms, and applications*, volume 3. Prentice-Hall Englewood Cliffs, 1994.

[38] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[39] E. Ficke, K. Schweitzer, R. Bateman, and S. Xu. Characterizing the effectiveness of network-based intrusion detection systems. In *IEEE Milcom'2018*. 2018.

[40] Eric Ficke, Kristin M. Schweitzer, Raymond M. Bateman, and Shouhuai Xu. Characterizing the effectiveness of network-based intrusion detection systems. In *2018 IEEE Military*

Communications Conference, MILCOM 2018, Los Angeles, CA, USA, October 29-31, 2018, pages 76–81, 2018.

[41] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomputing*, 122:13–23, 2013.

[42] Gianluigi Folino, Francesco Sergio Pisani, and Pietro Sabatino. A distributed intrusion detection framework based on evolved specialized ensembles of classifiers. In *European Conference on the Applications of Evolutionary Computation*, pages 315–331. Springer, 2016.

[43] Richard Garcia-Lebron, David J. Myers, Shouhuai Xu, and Jie Sun. Node diversification in complex networks by decentralized coloring). manuscript under review, 2018.

[44] Richard Garcia-Lebron, Kristin Schweitzer, Raymond Bateman, and Shouhuai Xu. A framework for characterizing the evolution of cyber attacker-victim relation graphs. In *IEEE Milcom'2018*. 2018.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] Google. Machine learning crash course: Embeddings. `https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture`, 2019.

[47] Gustavo Grieco, Guillermo Luis Grinblat, Lucas C. Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, New Orleans, LA, USA*, pages 85–96, 2016.

[48] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

[49] Yujuan Han, Wnelian Lu, and Shouhuai Xu. Characterizing the power of moving target defense via cyber epidemic dynamics. In *Proc. 2014 Symposium and Bootcamp on the Science of Security (HotSoS'14)*, pages 10:1–10:12, 2014.

[50] Jens Hühn and Eyke Hüllermeier. Furia: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319, 2009.

[51] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Colleen Shannon, Matthew Luckie, and K Claffy. The caida ipv4 routed/24 topology dataset. *URL http://www. caida. org/data/active/ipv4_routed_24_topology_dataset. xml*, 2011.

[52] Igor Ibarguren, Jesús M Pérez, Javier Muguerza, Ibai Gurrutxaga, and Olatz Arbelaitz. Coverage-based resampling: Building robust consolidated decision trees. *Knowledge-Based Systems*, 79:51–67, 2015.

[53] Erhan J. Kartaltepe, Jose Andre Morales, Shouhuai Xu, and Ravi S. Sandhu. Social network-based botnet command-and-control: Emerging threats and countermeasures. In *ACNS*, pages 511–528, 2010.

[54] Gulshan Kumar and Krishan Kumar. Design of an evolutionary approach for intrusion detection. *The Scientific World Journal*, 2013, 2013.

[55] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of tor traffic using time based features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 253–262. INSTICC, SciTePress, 2017.

[56] L.Chen, S. Hou, Y. Ye, and S. Xu. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In *Proc. 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages ??–??, 2018.

[57] Justin Leonard, Shouhuai Xu, and Ravi S. Sandhu. A first step towards characterizing stealthy botnets. In *Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009, March 16-19, 2009, Fukuoka, Japan*, pages 106–113, 2009.

[58] Justin Leonard, Shouhuai Xu, and Ravi S. Sandhu. A framework for understanding botnets. In *Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009*, pages 917–922, 2009.

[59] D. Li, Q. Li, Y. Ye, and S. Xu. Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge. In *AAAI-2019 Workshop on Artificial Intelligence for Cyber Security (AICS'2019)*.

[60] Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. *CoRR*, abs/1809.06498, 2018.

[61] Xiaohu Li, Paul Parker, and Shouhuai Xu. A stochastic model for quantitative security analyses of networked systems. *IEEE Transactions on Dependable and Secure Computing*, 8(1):28–43, 2011.

[62] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Hanchao Qi, and Jie Hu. Vulpecker: An automated vulnerability detection system based on code similarity analysis. In *Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, CA, USA*, pages 201–213, 2016.

[63] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. Sysevr: A framework for using deep learning to detect software vulnerabilities. *CoRR*, abs/1807.06756, 2018.

[64] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. In

25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018, 2018.

[65] Z. Lin, W. Lu, and S. Xu. Unified preventive and reactive cyber defense dynamics is still globally convergent. *IEEE/ACM Transactions on Networking*, 2019 (accepted for publication).

[66] Richard Lippmann, Robert K Cunningham, David J Fried, Isaac Graf, Kris R Kendall, Seth E Webster, and Marc A Zissman. Results of the 1998 darpa offline intrusion detection evaluation. In *Proc. Recent Advances in Intrusion Detection*, 1999.

[67] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[68] Wenlian Lu, Shouhuai Xu, and Xinlei Yi. Optimizing active cyber defense dynamics. In *Proceedings of the 4th International Conference on Decision and Game Theory for Security (GameSec'13)*, pages 206–225, 2013.

[69] Freda Lundy. *Cyber Threat Alert Fatigue and Reduction Methods*. PhD thesis, 2017. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2018-03-02.

[70] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Learning to detect malicious urls. *ACM TIST*, 2(3):30:1–30:24, 2011.

[71] Jose Mireles, Eric Ficke, Jin-Hee Cho, Patrick Hurley, and Shouhuai Xu. Metrics towards measuring cyber agility. manuscript in submission, 2018.

[72] Jose David Mireles, Jin-Hee Cho, and Shouhuai Xu. Extracting attack narratives from traffic datasets. In *2016 International Conference on Cyber Conflict, CyCon U.S. 2016, Washington, DC, USA, October 21-23, 2016*, pages 118–123, 2016.

[73] Jose David Mireles, Eric Ficke, Jin-Hee Cho, Patrick Hurley, and Shouhuai Xu. Metrics towards measuring cyber agility. IEEE Transaction on Information Forensics & Security, 2019 (accepted for publication).

[74] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[75] J.A. Morales, R. Sandhu, and Shouhuai Xu. Evaluating detection and treatment effectiveness of commercial anti-malware programs. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 31–38, 2010.

[76] Jose Morales, Shouhuai Xu, and Ravi Sandhu. Analyzing malware detection efficiency with multiple anti-malware programs. In *Proceedings of 2012 ASE International Conference on Cyber Security (CyberSecurity'12)*, 2012.

[77] Jose Andre Morales, Areej Al-Bataineh, Shouhuai Xu, and Ravi S. Sandhu. Analyzing and exploiting network behaviors of malware. In *SecureComm*, pages 20–34, 2010.

[78] Jose Andre Morales, Erhan J. Kartaltepe, Shouhuai Xu, and Ravi S. Sandhu. Symptoms-based detection of bot processes. In *MMM-ACNS*, pages 229–241, 2010.

[79] Jose Andre Morales, Michael Main, Weiliang Luo, Shouhuai Xu, and Ravi S. Sandhu. Building malware infection trees. In *6th International Conference on Malicious and Unwanted Software (MALWARE'2011)*, pages 50–57, 2011.

[80] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA*, pages 529–540, 2007.

[81] Andrew Ng. Neural networks and deep learning, 2018.

[82] Chris V. Nicholson and Adam Gibson. Introduction to deep neural networks (deep learning).

[83] SP NIST. 800-94, guide to intrusion detection and prevention systems (idps). *Information Technology Laboratory, National Institute of Standards and Technology, USA*, 2007.

[84] Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS), BICT-15*, volume 15, pages 21–26, 2015.

[85] Bhaskar Pant, Daniela Rus, and Howard Shrobe. Cybersecurity: Technology, application and policy, Spring 2016.

[86] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 1st edition, 2017.

[87] W. Patterson. *Mathematical Cryptology for Computer Scientists and Mathematicians*. Rowman and Littlefield, 1987.

[88] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. A survey on systems security metrics. *ACM Comput. Surv.*, 49(4):62:1–62:35, December 2016.

[89] Marcus Pendleton and Shouhuai Xu. A dataset generator for next generation system call host intrusion detection systems. In *2017 IEEE Military Communications Conference, MILCOM 2017, Baltimore, MD, USA, October 23-25, 2017*, pages 231–236, 2017.

[90] Chen Peng, Maochao Xu, Shouhuai Xu, and Taizhong Hu. Modeling and predicting extreme cyber attack rates via marked point processes. *Journal of Applied Statistics*, 44(14):2534–2563, 2017.

[91] Chen Peng, Maochao Xu, Shouhuai Xu, and Taizhong Hu. Modeling multivariate cybersecurity risks. *Journal of Applied Statistics*, 0(0):1–23, 2018.

[92] Moustafa Saleh, Tao Li, and Shouhuai Xu. Multi-context features for detecting malicious programs. *J. Computer Virology and Hacking Techniques*, 14(2):181–193, 2018.

[93] Moustafa Saleh, E. Paul Ratazzi, and Shouhuai Xu. A control flow graph-based signature for packer identification. In *2017 IEEE Military Communications Conference, MILCOM 2017, Baltimore, MD, USA, October 23-25, 2017*, pages 683–688, 2017.

[94] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, Ali A. Ghorbani, , , and and. Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2017(1):177–200, 2017.

[95] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *4th International Conference on Information Systems Security and Privacy*, pages 108–116. SCITEPRESS - Science and Technology Publications, 2018.

[96] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.*, 31(3):357–374, May 2012.

[97] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316. IEEE, 2010.

[98] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, Ren Ping Liu, and Jiankun Hu. Detection of denial-of-service attacks based on computer vision techniques. *IEEE transactions on computers*, 64(9):2519–2533, 2015.

[99] Florian Teschner. Exploring embeddings for categorical variables with keras. `http://flovv.github.io/Embeddings_with_keras/`, 2018.

[100] Rachel Thomas. An introduction to deep learning for tabular data. `https://www.fast.ai/2018/04/29/categorical-embeddings/`, 2018.

[101] A.M. Turing. *Computing Machinery and Intelligence*. Mind: a quarterly review. Blackwell for the Mind Association, 1950.

[102] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. Cross-layer detection of malicious websites. In *Third ACM Conference on Data and Application Security and Privacy (ACM CODASPY'13)*, pages 141–152, 2013.

[103] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. An evasion and counter-evasion study in malicious websites detection. In *IEEE Conference on Communications and Network Security (CNS'14)*, pages 141–152, 2013.

[104] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. An evasion and counter-evasion study in malicious websites detection. In *IEEE Conference on Communications and Network Security (CNS'14)*, pages 265–273, 2014.

[105] M. Xu, K. M. Schweitzer, R. M. Bateman, and S. Xu. Modeling and predicting cyber hacking breaches. *IEEE Transactions on Information Forensics and Security*, 13(11):2856–2871, Nov 2018.

[106] Maochao Xu, Gaofeng Da, and Shouhuai Xu. Cyber epidemic models with dependences. *Internet Mathematics*, 11(1):62–92, 2015.

[107] Maochao Xu, Lei Hua, and Shouhuai Xu. A vine copula model for predicting the effectiveness of cyber defense early-warning. *Technometrics*, 59(4):508–520, 2017.

[108] Maochao Xu and Shouhuai Xu. An extended stochastic model for quantitative security analysis of networked systems. *Internet Mathematics*, 8(3):288–320, 2012.

[109] Shouhuai Xu. Cybersecurity dynamics publications. `http://www.cs.utsa.edu/~shxu/socs/`.

[110] Shouhuai Xu. Cybersecurity dynamics. In *Proc. Symposium and Bootcamp on the Science of Security (HotSoS'14)*, pages 14:1–14:2, 2014.

[111] Shouhuai Xu. Emergent behavior in cybersecurity. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security (HotSoS'14)*, pages 13:1–13:2, 2014.

[112] Shouhuai Xu. Cybersecurity dynamics: A foundation for the science of cybersecurity. In Zhuo Lu and Cliff Wang, editors, *Proactive and Dynamic Network Defense*. Springer New York, 2018 (to appear).

[113] Shouhuai Xu, Wenlian Lu, and Hualun Li. A stochastic model of active cyber defense dynamics. *Internet Mathematics*, 11(1):23–61, 2015.

[114] Shouhuai Xu, Wenlian Lu, and Li Xu. Push- and pull-based epidemic spreading in arbitrary networks: Thresholds and deeper insights. *ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS)*, 7(3):32:1–32:26, 2012.

[115] Shouhuai Xu, Wenlian Lu, Li Xu, and Zhenxin Zhan. Adaptive epidemic dynamics in networks: Thresholds and control. *ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS)*, 8(4):19, 2014.

[116] Shouhuai Xu, Wenlian Lu, and Zhenxin Zhan. A stochastic model of multivirus dynamics. *IEEE Transactions on Dependable and Secure Computing*, 9(1):30–45, 2012.

[117] Fabian Yamaguchi, Markus Lottmann, and Konrad Rieck. Generalized vulnerability extrapolation using abstract syntax trees. In *28th Annual Computer Security Applications Conference, Orlando, FL, USA*, pages 359–368, 2012.

[118] Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon, and Konrad Rieck. Chucky: Exposing missing checks in source code for vulnerability discovery. In *2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany*, pages 499–510, 2013.

[119] Warusia Yassin, Nur Izura Udzir, Zaiton Muda, Md Nasir Sulaiman, et al. Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In *Proc. 4th Int. Conf. Comput. Informatics, ICOCI*, volume 49, pages 298–303, 2013.

[120] Y. Ye, S. Hou, L. Chen, X. Li, L. Zhao, S. Xu, J. Wang, and Q. Xiong. Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information

network. In *Proceedings of the 34nd Annual Conference on Computer Security Applications, ACSAC 2018*, pages 131–140, 2018.

[121] Zhenxin Zhan, Maochao Xu, and Shouhuai Xu. Characterizing honeypot-captured cyber attacks: Statistical framework and case study. *IEEE Transactions on Information Forensics and Security*, 8(11):1775–1789, 2013.

[122] Zhenxin Zhan, Maochao Xu, and Shouhuai Xu. Predicting cyber attack rates with extreme values. *IEEE Transactions on Information Forensics and Security*, 10(8):1666–1677, 2015.

[123] Ren Zheng, Wenlian Lu, and Shouhuai Xu. Active cyber defense dynamics exhibiting rich phenomena. In *Proc. 2015 Symposium and Bootcamp on the Science of Security (HotSoS'15)*, pages 2:1–2:12, 2015.

[124] Ren Zheng, Wenlian Lu, and Shouhuai Xu. Preventive and reactive cyber defense dynamics is globally stable. *IEEE Trans. Network Science and Engineering*, 5(2):156–170, 2018.

VITA

Gabriel Carlos Fernández is a M.Sc. student in Computer Science at the University of Texas at San Antonio. His research focus is on cybersecurity data analytics, attack and defense, and machine learning.

Gabriel is currently employed at USAA where he has worked primarily as a Research Engineer focused on cybersecurity and fraud research, and more recently as a Software Engineer working on projects within the same domain. During his tenure at USAA, he has been granted twelve patents to date, and has more than five patents in application.

He has two papers: "My brain is my passport. Verify me," which was published in proceedings at the 2016 IEEE International Conference on Consumer Electronics (ICCE); and "Addressing the vulnerabilities of pass-thoughts," which was published in proceedings of Signal Processing, Sensor/Information Fusion, and Target Recognition XXV (SPIE Defense + Security 2016).

applied sciences

MDPI

Review

Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey

Hongyu Liu * and Bo Lang

State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China; langbo@buaa.edu.cn

* Correspondence: liuhongyu@buaa.edu.cn

Received: 14 September 2019; Accepted: 11 October 2019; Published: 17 October 2019

Abstract: Networks play important roles in modern life, and cyber security has become a vital research area. An intrusion detection system (IDS) which is an important cyber security technique, monitors the state of software and hardware running in the network. Despite decades of development, existing IDSs still face challenges in improving the detection accuracy, reducing the false alarm rate and detecting unknown attacks. To solve the above problems, many researchers have focused on developing IDSs that capitalize on machine learning methods. Machine learning methods can automatically discover the essential differences between normal data and abnormal data with high accuracy. In addition, machine learning methods have strong generalizability, so they are also able to detect unknown attacks. Deep learning is a branch of machine learning, whose performance is remarkable and has become a research hotspot. This survey proposes a taxonomy of IDS that takes data objects as the main dimension to classify and summarize machine learning-based and deep learning-based IDS literature. We believe that this type of taxonomy framework is fit for cyber security researchers. The survey first clarifies the concept and taxonomy of IDSs. Then, the machine learning algorithms frequently used in IDSs, metrics, and benchmark datasets are introduced. Next, combined with the representative literature, we take the proposed taxonomic system as a baseline and explain how to solve key IDS issues with machine learning and deep learning techniques. Finally, challenges and future developments are discussed by reviewing recent representative studies.

Keywords: machine learning; deep learning; intrusion detection system; cyber security

1. Introduction

Networks have increasing influences on modern life, making cyber security an important field of research. Cyber security techniques mainly include anti-virus software, firewalls and intrusion detection systems (IDSs). These techniques protect networks from internal and external attacks. Among them, an IDS is a type of detection system that plays a key role in protecting cyber security by monitoring the states of software and hardware running in a network.

The first intrusion detection system was proposed in 1980 [1]. Since then, many mature IDS products have arisen. However, many IDSs still suffer from a high false alarm rate, generating many alerts for low nonthreatening situations, which raises the burden for security analysts and can cause seriously harmful attack to be ignored. Thus, many researchers have focused on developing IDSs with higher detection rates and reduced false alarm rates. Another problem with existing IDSs is that they lack the ability to detect unknown attacks. Because network environments change quickly, attack variants and novel attacks emerge constantly. Thus, it is necessary to develop IDSs that can detect unknown attacks.

To address the above problems, researchers have begun to focus on constructing IDSs using machine learning methods. Machine learning is a type of artificial intelligence technique that can

automatically discover useful information from massive datasets [2]. Machine learning-based IDSs can achieve satisfactory detection levels when sufficient training data is available, and machine learning models have sufficient generalizability to detect attack variants and novel attacks. In addition, machine learning-based IDSs do not rely heavily on domain knowledge; therefore, they are easy to design and construct. Deep learning is a branch of machine learning that can achieve outstanding performances. Compared with traditional machine learning techniques, deep learning methods are better at dealing with big data. Moreover, deep learning methods can automatically learn feature representations from raw data and then output results; they operate in an end-to-end fashion and are practical. One notable characteristic of deep learning is the deep structure, which contains multiple hidden layers. In contrast, traditional machine learning models, such as the support vector machine (SVM) and k-nearest neighbor (KNN), contain none or only one hidden layer. Therefore, these traditional machine learning models are also called shallow models.

The purpose of this survey is to classify and summarize the machine learning-based IDSs proposed to date, abstract the main ideas of applying machine learning to security domain problems, and analyze the current challenges and future developments. For this survey, we selected representative papers published from 2015 to 2019, which reflect the current progress. Several previous surveys [3–5] have classified research efforts by their applied machine learning algorithms. These surveys are primarily intended to introduce different machine learning algorithms applied to IDSs, which can be helpful to machine learning researchers. However, this type of taxonomic system emphasizes specific implementation technologies rather than cyber security domain problems. As a result, these surveys do not directly address how to resolve IDS domain problems using machine learning. For coping with this problem, we propose a new data-centered IDS taxonomy in this survey, and introduce the related studies following this taxonomy.

Data objects are the most basic elements in IDS. Data objects carry features related to attack behaviors. Feature types and feature extraction methods differ among different data elements, causing the most appropriate machine learning models to also differ. Therefore, this survey thoroughly analyzes the data processed in cyber security and classifies IDSs on the basis of data sources. This taxonomy presents a path involving data–feature–attack behavior–detection model, which is convenient for readers to find study ideas for particular domain problems. For example, this taxonomic system can answer the following problems: (1) What features best represent different attacks? (2) What type of data is most suitable for detecting certain attacks? (3) What types of machine learning algorithms are the best fit for a specific data type? (4) How do machine learning methods improve IDSs along different aspects? These problems appeal to cyber security researchers. Finally, the challenges and future development of machine learning methods for IDS are discussed by summarizing recent representative studies.

The rest of this paper is organized as follows: Section 2 introduces the key concepts and the taxonomy of IDS. Section 3 introduces the frequently used machine learning algorithms in IDS, their metrics, and common benchmark datasets. Section 4 classifies IDS according to data sources and sums up the process of applying machine learning to IDSs. Section 5 discusses the challenges and future directions of machine learning-based IDSs, and Section 6 concludes the paper.

2. Concept and Taxonomy of IDS

For an IDS, an intrusion means an attempt to access information about computer systems or to damage system operation in an illegal or unauthorized manner. An IDS is a computer-security application that aims to detect a wide range of security violations, ranging from attempted break-ins by outsiders to system penetrations and abuses by insiders [6]. The main functions of IDSs are to monitor hosts and networks, analyze the behaviors of computer systems, generate alerts, and respond to suspicious behaviors. Because they monitor related hosts and networks, IDSs are typically deployed near the protected network nodes (e.g., the switches in major network segments).

There are two types of IDS classification methods: detection-based method and data source-based methods. Among the detection-based methods, IDSs can be divided into misuse detection and

anomaly detection. Among the data source-based methods, IDSs can be divided into host-based and network-based methods [7]. This survey combines these two types of IDS classification methods, taking the data source as the main classification consideration and treating the detection method as a secondary classification element. The proposed taxonomy is shown in Figure 1. Regarding detection methods, the survey concentrates on machine learning methods. We introduce how to apply machine learning to IDS using different types of data in detail in Section 4.

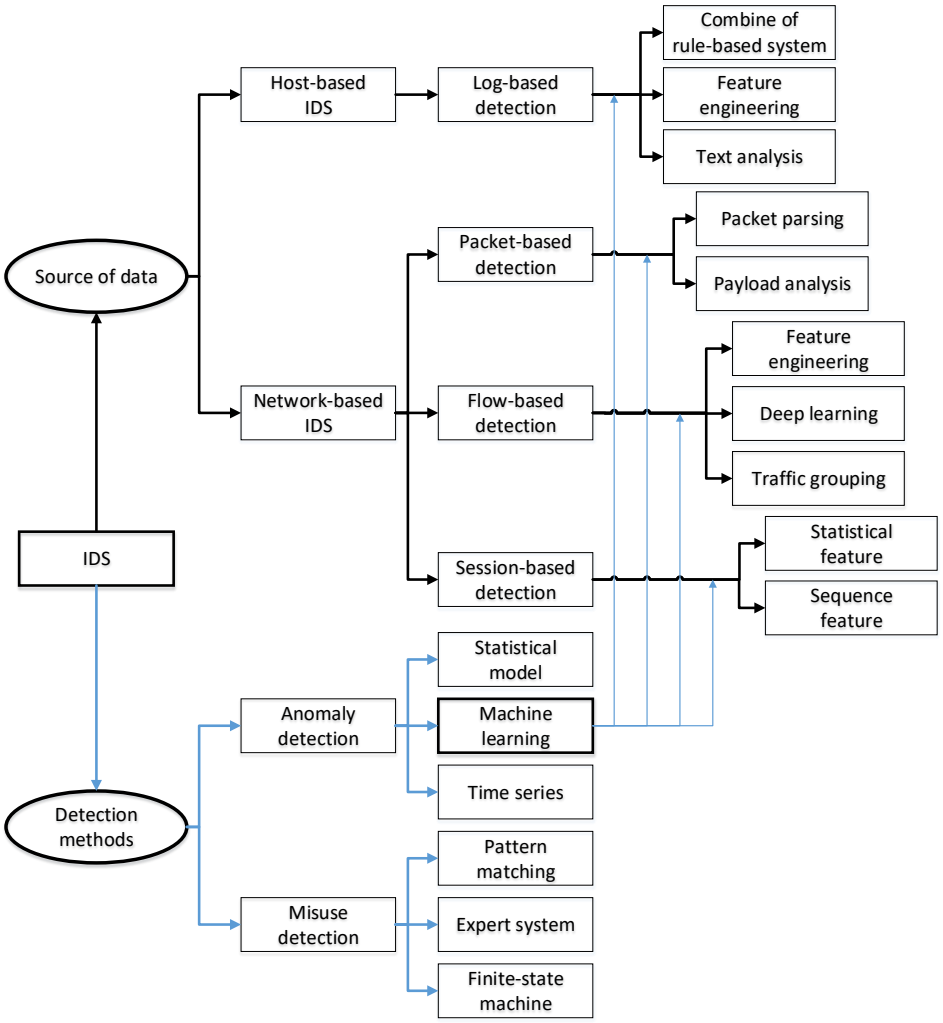


Figure 1. Taxonomy system of IDS.

2.1. Classification by Detection Methods

Misuse detection is also called signature-based detection. The basic idea to represent attack behaviors as signatures. The detection process matches the signatures of samples using a signature database. The main problem in constructing misuse detection systems is to design efficient signatures. The advantages of misuse detection are that it has a low false alarm rate and it reports attack types as well as possible reasons in detail; the disadvantages are that it has a high missed alarm rate, lacks the ability to detect unknown attacks, and requires maintaining a huge signature database. The design idea behind anomaly detection is to establish a normal behavior profile and then define abnormal behaviors by their degree of deviation from the normal profile. Thus, the key to designing an anomaly detection system is to clearly define a normal profile. The benefits of anomaly detection are strong generalizability and the ability to recognize unknown attacks, while its shortcomings are a high false alarm rate and an inability to provide possible reasons for an abnormality. The main differences between misuse detection and anomaly detection are listed in Table 1.

Table 1. Differences between misuse detection and anomaly detection.

	Misuse Detection	Anomaly Detection
Detection performance	Low false alarm rate; High missed alarm rate	Low missed alarm rate; High false alarm rate
Detection efficiency	High, decrease with scale of signature database	Dependent on model complexity
Dependence on domain knowledge	Almost all detections depend on domain knowledge	Low, only the feature design depends on domain knowledge
Interpretation	Design based on domain knowledge, strong interpretative ability	Outputs only detection results, weak interpretative ability
Unknown attack detection	Only detects known attacks	Detects known and unknown attacks

As shown in Figure 1, in detection method-based taxonomy, misuse detection includes pattern matching-based, expert system, and finite state machine-based methods. Anomaly detection includes statistical model-based, machine learning-based, and time series-based methods.

2.2. Classification by Source of Data

An advantage of a host-based IDSs is that it can locate intrusions precisely and initiate responses because such IDSs can monitor the behaviors of significant objects (e.g., sensitive files, programs and ports). The disadvantages are that host-based IDSs occupy host resources, are dependent on the reliability of the host, and are unable to detect network attacks. A network-based IDS is usually deployed on major hosts or switches. A majority of network-based IDSs are independent of the operating system (OS); thus, they can be applied in different OS environments. Furthermore, network-based IDSs are able to detect specific types of protocol and network attacks. The drawback is that they monitor only the traffic passing through a specific network segment. The main differences between host-based IDS and network-based IDS are listed in Table 2.

Table 2. Differences between host-based and network-based IDSs.

	Host-Based IDS	Network-Based IDS
Source of data	Logs of operating system or application programs	Network traffic
Deployment	Every host; Dependent on operating systems; Difficult to deploy	Key network nodes; Easy to deploy
Detection efficiency	Low, must process numerous logs	High, can detect attacks in real time
Intrusion traceability	Trace the process of intrusion according to system call paths	Trace position and time of intrusion according to IP addresses and timestamps
Limitation	Cannot analyze network behaviors	Monitor only the traffic passing through a specific network segment

As shown in Figure 1, a host-based IDS uses audit logs as a data source. Log detection methods are mainly hybrids based on rule and machine learning, rely on log features, and use text analysis-based methods. A network-based IDS uses network traffic as a data source—typically packets, which are the basic units of network communication. A flow is the set of packets within a time window, which reflects the network environment. A session is a packet sequence combined on the basis of a network information 5-tuple (client IP, client port, server IP, server port, protocol). A session represents high-level semantic information of traffic. Packets contain packet headers and payloads; therefore, packet detection includes parsing-based and payload analysis-based methods. Based on feature

extraction, flow detection can be divided into feature engineering-based and deep learning-based methods. In addition, traffic grouping is a unique approach in flow detection. Based on whether sequence information is used, session detection can be divided into statistical feature-based and sequence feature-based methods.

3. Common Machine Learning Algorithms in IDS

3.1. Machine Learning Models

There are two main types of machine learning: supervised and unsupervised learning. Supervised learning relies on useful information in labeled data. Classification is the most common task in supervised learning (and is also used most frequently in IDS); however, labeling data manually is expensive and time consuming. Consequently, the lack of sufficient labeled data forms the main bottleneck to supervised learning. In contrast, unsupervised learning extracts valuable feature information from unlabeled data, making it much easier to obtain training data. However, the detection performance of unsupervised learning methods is usually inferior to those of supervised learning methods. The common machine learning algorithms used in IDSs are shown in Figure 2.

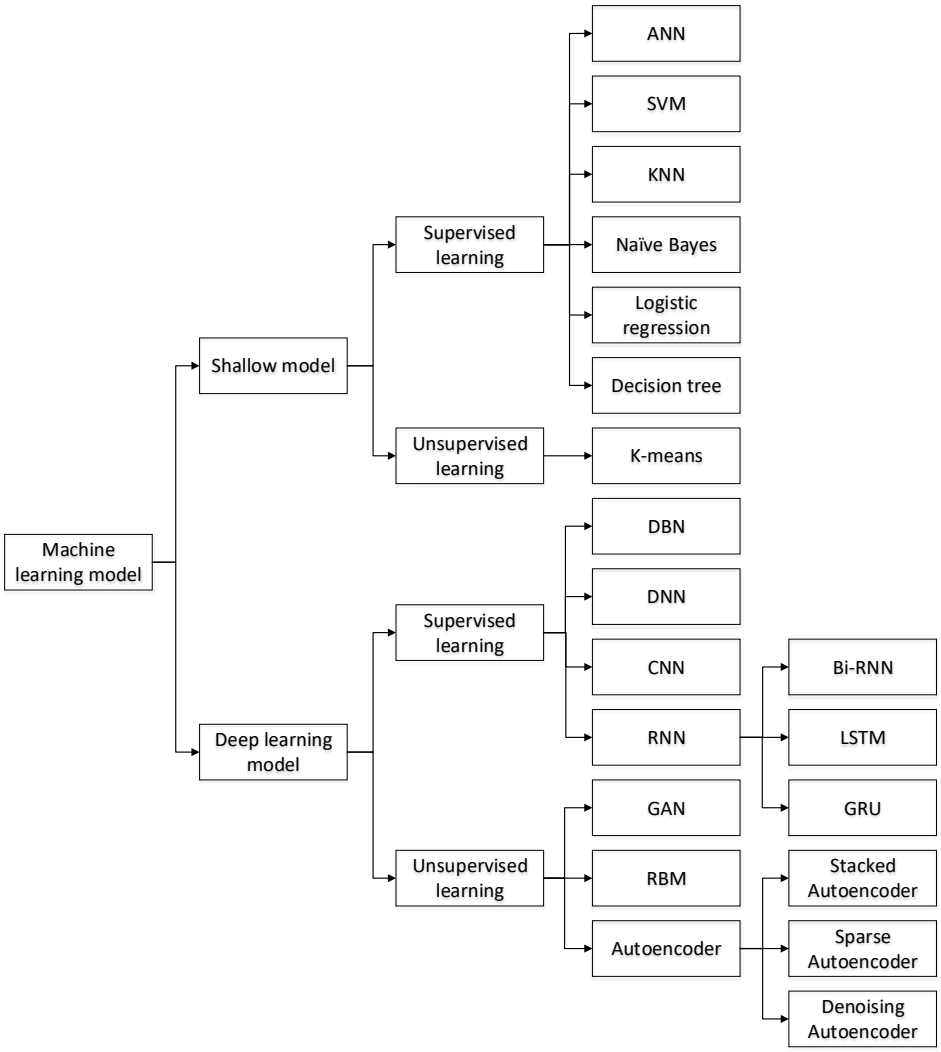


Figure 2. Taxonomy of machine learning algorithms.

3.1.1. Shallow Models

The traditional machine learning models (shallow models) for IDS primarily include the artificial neural network (ANN), support vector machine (SVM), K-nearest neighbor (KNN), naïve Bayes, logistic regression (LR), decision tree, clustering, and combined and hybrid methods. Some of these methods have been studied for several decades, and their methodology is mature. They focus not only on the detection effect but also on practical problems, e.g., detection efficiency and data management. The pros and cons of various shallow models are shown in Table 3.

Table 3. The pros and cons of various shallow models.

Algorithms	Advantages	Disadvantages	Improvement Measures
ANN	Able to deal with nonlinear data; Strong fitting ability	Apt to overfitting; Prone to become stuck in a local optimum; Model training is time consuming	Adopted improved optimizers, activation functions, and loss functions
SVM	Learn useful information from small train set; Strong generation capability	Do not perform well on big data or multiple classification tasks; Sensitive to kernel function parameters	Optimized parameters by particle swarm optimization (PSO)[8]
KNN	Apply to massive data; Suitable to nonlinear data; Train quickly; Robust to noise	Low accuracy on the minority class; Long test times; Sensitive to the parameter *K*	Reduced comparison times by trigonometric inequality; Optimized parameters by particle swarm optimization (PSO) [9]; Balanced datasets using the synthetic minority oversampling technique (SMOTE) [10]
Naïve Bayes	Robust to noise; Able to learn incrementally	Do not perform well on attribute-related data	Imported latent variables to relax the independent assumption [11]
LR	Simple, can be trained rapidly; Automatically scale features	Do not perform well on nonlinear data; Apt to overfitting	Imported regularization to avoid overfitting [12]
Decision tree	Automatically select features; Strong interpretation	Classification result trends to majority class; Ignore the correlation of data	Balanced datasets with SMOTE; Introduced latent variables
K-means	Simple, can be trained rapidly; Strong scalability; Can fit to big data	Do not perform well on nonconvex data; Sensitive to initialization; Sensitive to the parameter *K*	Improved initialization method [13]

Artificial Neural Network (ANN). The design idea of an ANN is to mimic the way human brains work. An ANN contains an input layer, several hidden layers, and an output layer. The units in adjacent layers are fully connected. An ANN contains a huge number of units and can theoretically approximate arbitrary functions; hence, it has strong fitting ability, especially for nonlinear functions. Due to the complex model structure, training ANNs is time-consuming. It is noteworthy that ANN models are trained by the backpropagation algorithm that cannot be used to train deep networks. Thus, an ANN belongs to shallow models and differs from the deep learning models discussed in Section 3.1.2.

Support Vector Machine (SVM). The strategy in SVMs is to find a max-margin separation hyperplane in the n-dimension feature space. SVMs can achieve gratifying results even with small-scale training sets because the separation hyperplane is determined only by a small number of support vectors. However, SVMs are sensitive to noise near the hyperplane. SVMs are able to solve linear

problems well. For nonlinear data, kernel functions are usually used. A kernel function maps the original space into a new space so that the original nonlinear data can be separated. Kernel tricks are widespread among both SVMs and other machine learning algorithms.

K-Nearest Neighbor (KNN). The core idea of KNN is based on the manifold hypothesis. If most of a sample's neighbors belong to the same class, the sample has a high probability of belonging to the class. Thus, the classification result is only related to the top-k nearest neighbors. The parameter k greatly influences the performance of KNN models. The smaller k is, the more complex the model is and the higher the risk of overfitting. Conversely, the larger k is, the simpler the model is and the weaker the fitting ability.

Naïve Bayes. The Naïve Bayes algorithm is based on the conditional probability and the hypothesis of attribute independence. For every sample, the Naïve Bayes classifier calculates the conditional probabilities for different classes. The sample is classified into the maximum probability class. The conditional probability formula is calculated as shown in Formula (1).

$$P(X = x|Y = c_k) = \prod_{i=1}^{n} P(X^{(i)} = x^{(i)}|Y = c_k) \quad (1)$$

When the attribute independence hypothesis is satisfied, the Naïve Bayes algorithm reaches the optimal result. Unfortunately, that hypothesis is difficult to satisfy in reality; hence, the Naïve Bayes algorithm does not perform well on attribute-related data.

Logistic Regression (LR). The LR is a type of logarithm linear model. The LR algorithm computes the probabilities of different classes through parametric logistic distribution, calculated as shown in Formula (2).

$$P(Y = k|x) = \frac{e^{w_k * x}}{1 + \sum_{k}^{K-1} e^{w_k * x}} \quad (2)$$

where $k = 1,2...K - 1$. The sample x is classified into the maximum probability class. An LR model is easy to construct, and model training is efficient. However, LR cannot deal well with nonlinear data, which limits its application.

Decision tree. The decision tree algorithm classifies data using a series of rules. The model is tree like, which makes it interpretable. The decision tree algorithm can automatically exclude irrelevant and redundant features. The learning process includes feature selection, tree generation, and tree pruning. When training a decision tree model, the algorithm selects the most suitable features individually and generates child nodes from the root node. The decision tree is a basic classifier. Some advanced algorithms, such as the random forest and the extreme gradient boosting (XGBoost), consist of multiple decision trees.

Clustering. Clustering is based on similarity theory, i.e., grouping highly similar data into the same clusters and grouping less-similar data into different clusters. Different from classification, clustering is a type of unsupervised learning. No prior knowledge or labeled data is needed for clustering algorithms; therefore, the data set requirements are relatively low. However, when using clustering algorithms to detect attacks, it is necessary to refer external information.

K-means is a typical clustering algorithm, where K is the number of clusters and the means is the mean of attributes. The K-means algorithm uses distance as a similarity measure criterion. The shorter the distance between two data objects is, the more likely they are to be placed in the same cluster. The K-means algorithm adapts well to linear data, but its results on nonconvex data are not ideal. In addition, the K-means algorithm is sensitive to the initialization condition and the parameter K. Consequently, many repeated experiments must be run to set the proper parameter value.

Ensembles and Hybrids. Every individual classifier has strengths and shortcomings. A natural approach is to combine various weak classifiers to implement a strong classifier. Ensemble methods train multiple classifiers; then, the classifiers vote to obtain the final results. Hybrid methods are designed as many stages, in which each stage uses a classification model. Because ensemble and hybrid

classifiers usually perform better than do single classifiers, an increasing number of researchers have begun to study ensemble and hybrid classifiers. The key points lie in selecting which classifiers to combine and how they are combined.

3.1.2. Deep Learning Models

Deep learning models consist of diverse deep networks. Among them, deep brief networks (DBNs), deep neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) are supervised learning models, while autoencoders, restricted Boltzmann machines (RBMs), and generative adversarial networks (GANs) are unsupervised learning models. The number of studies of deep learning-based IDSs has increased rapidly from 2015 to the present. Deep learning models directly learn feature representations from the original data, such as images and texts, without requiring manual feature engineering. Thus, deep learning methods can execute in an end-to-end manner. For large datasets, deep learning methods have a significant advantage over shallow models. In the study of deep learning, the main emphases are network architecture, hyperparameter selection, and optimization strategy. A comparison of various deep learning models is shown in Table 4.

Table 4. Comparison of various deep learning models

Algorithms	Suitable Data Types	Supervised or Unsupervised	Functions
Autoencoder	Raw data; Feature vectors	Unsupervised	Feature extraction; Feature reduction; Denoising
RBM	Feature vectors	Unsupervised	Feature extraction; Feature reduction; Denoising
DBN	Feature vectors	Supervised	Feature extraction; Classification
DNN	Feature vectors	Supervised	Feature extraction; Classification
CNN	Raw data; Feature vectors; Matrices	Supervised	Feature extraction; Classification
RNN	Raw data; Feature vectors; Sequence data	Supervised	Feature extraction; Classification
GAN	Raw data; Feature vectors	Unsupervised	Data augmentation; Adversarial training

Autoencoder. An autoencoder contains two symmetrical components, an encoder and a decoder, as shown in Figure 3. The encoder extracts features from raw data, and the decoder reconstructs the data from the extracted features. During training, the divergence between the input of the encoder and the output of the decoder is gradually reduced. When the decoder succeeds in reconstructing the data via the extracted features, it means that the features extracted by the encoder represent the essence of the data. It is important to note that this entire process requires no supervised information. Many famous autoencoder variants exist, such as denoising autoencoders [14,15] and sparse autoencoders [16].

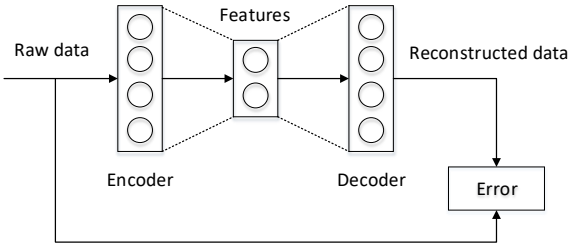


Figure 3. The structure of an autoencoder.

Restricted Boltzmann Machine (RBM). An RBM is a randomized neural network in which units obey the Boltzmann distribution. An RBM is composed of a visible layer and a hidden layer. The units in the same layer are not connected; however, the units in different layers are fully connected, as shown in Figure 4. where v_i is a visible layer, and h_i is a hidden layer. RBMs do not distinguish between the forward and backward directions; thus, the weights in both directions are the same. RBMs are unsupervised learning models trained by the contrastive divergence algorithm [17], and they are usually applied for feature extraction or denoising.

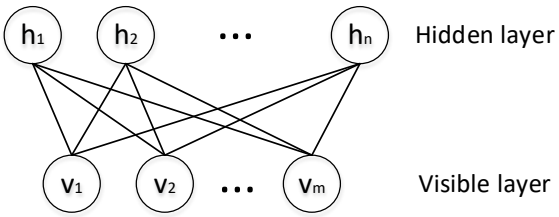


Figure 4. The structure of the RBM.

Deep Brief Network (DBN). A DBN consists of several RBM layers and a softmax classification layer, as shown in Figure 5. Training a DBN involves two stages: unsupervised pretraining and supervised fine-tuning [18,19]. First, each RBM is trained using greedy layer-wise pretraining. Then, the weight of the softmax layer are learned by labeled data. In attack detection, DBNs are used for both feature extraction and classification [20–22].

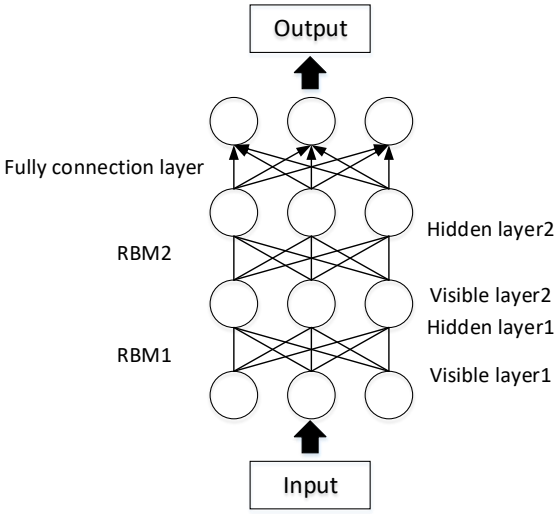


Figure 5. The structure of the DBN.

Deep Neural Network (DNN). A layer-wise pretraining and fine-tuning strategy makes it possible to construct DNNs with multiple layers, as shown in Figure 6. When training a DNN,

the parameters are learned first using unlabeled data, which is an unsupervised feature learning stage; then, the network is tuned through the labeled data, which is a supervised learning stage. The astonishing achievements of DNNs are mainly due to the unsupervised feature learning stage.

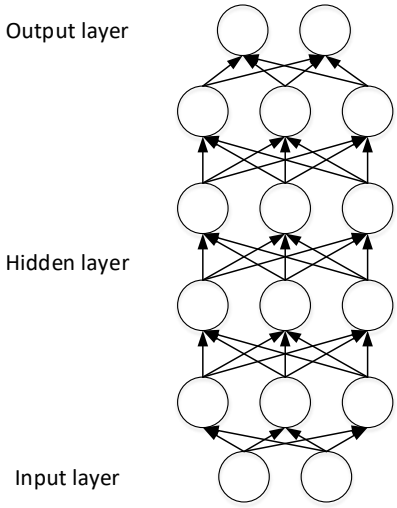


Figure 6. The structure of the DNN.

Convolutional Neural Network (CNN). CNNs are designed to mimic the human visual system (HVS); consequently, CNNs have made great achievements in the computer vision field [23–25]. A CNN is stacked with alternate convolutional and pooling layers, as shown in Figure 7. The convolutional layers are used to extract features, and the pooling layers are used to enhance the feature generalizability. CNNs work on 2-dimensional (2D) data, so the input data must be translated into matrices for attack detection.

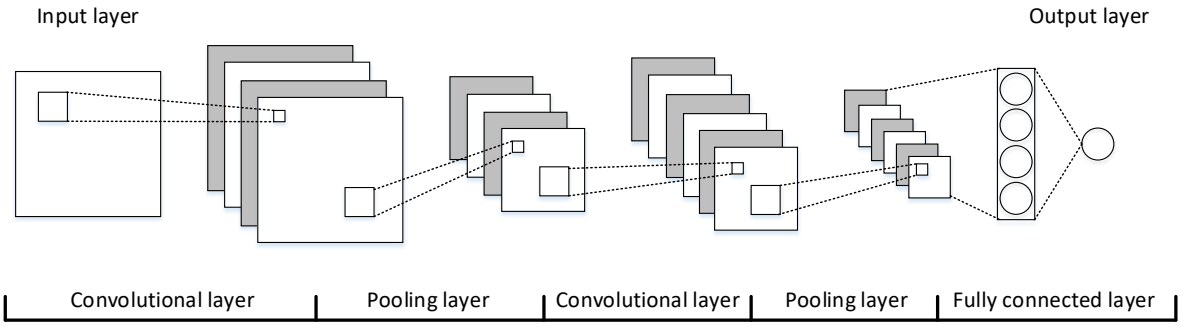


Figure 7. The structure of a CNN.

Recurrent Neural Network (RNN). RNNs are networks designed for sequential data and are widely used in natural language processing (NLP) [26–28]. The characteristics of sequential data are contextual; analyzing isolated data from the sequence makes no sense. To obtain contextual information, each unit in an RNN receives not only the current state but also previous states. The structure of an RNN is shown in Figure 8. Where all the *W* items in Figure 8 are the same. This characteristic causes RNNs to often suffer from vanishing or exploding gradients. In reality, standard RNNs deal with only limited-length sequences. To solve the long-term dependence problem, many RNN variants have been proposed, such as long short-term memory (LSTM) [29], gated recurrent unit (GRU) [30], and bi-RNN [31].

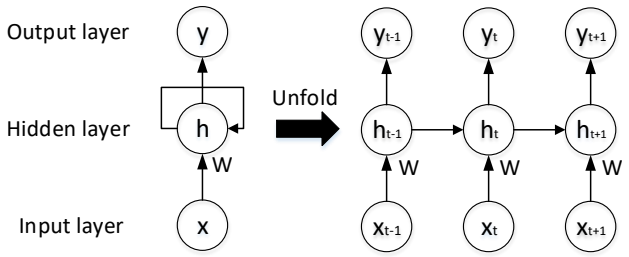


Figure 8. The structure of an RNN.

The LSTM model was proposed by Hochreiter and Schmidhuber in 1997 [29]. Each LSTM unit contains three gates: a forget gate, an input gate, and an output gate. The forget gate eliminates outdated memory, the input gate receives new data, and the output gate combines short-term memory with long-term memory to generate the current memory state. The GRU was proposed by Chung et al. in 2014 [30]. The GRU model merges the forget gate and the input gate into a single update gate, which is simpler than the LSTM.

Generative Adversarial Network (GAN). A GAN model includes two subnetworks, i.e., a generator and a discriminator. The generator aims to generate synthetic data similar to the real data, and the discriminator intends to distinguish synthetic data from real data. Thus, the generator and the discriminator improve each other. GANs are currently a hot research topic used to augment data in attack detection, which partly ease the problem of IDS dataset shortages. Meanwhile, GANs belong to adversarial learning approaches which can raise the detection accuracy of models by adding adversarial samples to the training set.

3.1.3. Shallow Models Compared to Deep Models

Deep learning is a branch of machine learning, and the effects of deep learning models are obviously superior to those of the traditional machine learning (or shallow model) methods in most application scenarios. The differences between shallow models and deep models are mainly reflected in the following aspects.

(1) Running time. The running time includes both training and test time. Due to the high complexity of deep models, both their training and test times are much longer than those of shallow models.

(2) Number of parameters. There are two types of parameters: learnable parameters and hyperparameters. The learnable parameters are calculated during the training phase, and the hyperparameters are set manually before training begins. The learnable parameters and hyperparameters in deep models far outnumber those in shallow models; consequently, training and optimizing deep models takes longer.

(3) Feature representation. The input to traditional machine learning models is a feature vector, and feature engineering is an essential step. In contrast, deep learning models are able to learn feature representations from raw data and are not reliant on feature engineering. The deep learning methods can execute in an end-to-end manner, giving them an outstanding advantage over traditional machine learning methods.

(4) Learning capacity. The structures of deep learning models are complex and they contain huge numbers of parameters (generally millions or more). Therefore, the deep learning models have stronger fitting ability than do shallow models. However, deep learning models also face a higher risk of overfitting, require a much larger volume of data for training. However, the effect of deep learning models is better.

(5) Interpretability. Deep learning models are black boxes [32–35]; the results are almost uninterpretable, which is a critical point in deep learning. However, some traditional deep learning algorithms, such as the decision tree and naïve Bayes, have strong interpretability.

3.2. Metrics

Many metrics are used to evaluate machine learning methods. The optimal models are selected using these metrics. To comprehensively measure the detection effect, multiple metrics are often used simultaneously in IDS research.

- **Accuracy** is defined as the ratio of correctly classified samples to total samples. Accuracy is a suitable metric when the dataset is balanced. In real network environments; however, normal samples are far more abundant than are abnormal samples; thus, accuracy may not be a suitable metric.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

- **Precision (P)** is defined as the ratio of true positive samples to predicted positive samples; it represents the confidence of attack detection.

$$P = \frac{TP}{TP + FP} \tag{4}$$

- **Recall (R)** is defined as the ratio of true positive samples to total positive samples and is also called the detection rate. The detection rate reflects the model's ability to recognize attacks, which is an important metric in IDS.

$$R = \frac{TP}{TP + FN} \tag{5}$$

- **F-measure (F)** is defined as the harmonic average of the precision and the recall.

$$F = \frac{2 * P * R}{P + R} \tag{6}$$

- **The false negative rate (FNR)** is defined as the ratio of false negative samples to total positive samples. In attack detection, the FNR is also called the missed alarm rate.

$$FNR = \frac{FN}{TP + FN} \tag{7}$$

- **The false positive rate (FPR)** is defined as the ratio of false positive samples to predicted positive samples. In attack detection, the FPR is also called the false alarm rate, and it is calculated as follows:

$$FPR = \frac{FP}{TP + FP} \tag{8}$$

where the TP is the true positives, FP is the false positives, TN is the true negatives, FN is the false negatives. The purpose of an IDS is to recognize attacks; therefore, attack samples are usually regarded as positives, and normal samples are usually regarded as negatives. In attack detection, the frequently used metrics include accuracy, recall (or detection rate), FNR (or missed alarm rate), and FPR (or false alarm rate).

3.3. Benchmark Datasets in IDS

The task of machine learning is to extract valuable information from data; therefore, the performance of machine learning depends upon the quality of the input data. Understanding data is the basis of machine learning methodology. For IDSs, the adopted data should be easy to acquire and reflect the behaviors of the hosts or networks. The common source data types for IDSs are packets, flow, sessions, and logs. Building a dataset is complex and time-consuming. After a benchmark dataset is constructed, it can be reused repeatedly by many researchers. In addition to convenience, there are two other benefits of using benchmark datasets. (1) The benchmark datasets are authoritative, and make experimental results more convincing. (2) Many published studies have been conducted

using common benchmark datasets, which allows new study results to be compared with those of previous studies.

(1) DARPA1998

The DARPA1998 dataset [36] was built by the Lincoln laboratory of MIT and is a widely used benchmark dataset in IDS studies. To compile it, the researchers collected Internet traffic over nine weeks; the first seven weeks form the training set, and the last two weeks form the test set. The dataset contains both raw packets and labels. There are five types of labels: *normal*, *denial of service (DOS)*, *Probe*, *User to Root (U2R)* and *Remote to Local (R2L)*. Because raw packets cannot be directly applied to traditional machine learning models, the KDD99 dataset was constructed to overcome this drawback.

(2) KDD99

The KDD99 [37] dataset is the most widespread IDS benchmark dataset at present. Its compilers extracted 41-dimensional features from data in DARPA1998. The labels in KDD99 are the same as the DARPA1998. There are four types of features in KDD99, i.e., *basic features*, *content features*, *host-based statistical features*, and *time-based statistical features*. Unfortunately, the KDD99 dataset includes many defects. First, the data are severely unbalanced, making the classification results biased toward the majority classes. Additionally, there are many duplicate records and redundant records exist. Many researchers have to filter the dataset carefully before they can use it. As a result, the experimental results from different studies are not always comparable. Last but not least, KDD data are too old to represent the current network environment.

(3) NSL-KDD

To overcome the shortcomings of the KDD99 dataset, the NSL-KDD [38] was proposed. The records in the NSL-KDD were carefully selected based on the KDD99. Records of different classes are balanced in the NSL-KDD, which avoids the classification bias problem. The NSL-KDD also removed duplicate and redundant records; therefore, it contains only a moderate number of records. Therefore, the experiments can be implemented on the whole dataset, and the results from different papers are consistent and comparable. The NSL-KDD alleviates the problems of data bias and data redundancy to some degree. However, the NSL-KDD does not include new data; thus, minority class samples are still lacking, and its samples are still out-of-date.

(4) UNSW-NB15

The UNSW-NB15 [39] dataset was compiled by the University of South Wales, where researchers configured three virtual servers to capture network traffic and extracted 49-dimensional features using tool named Bro. The dataset includes more types of attacks than does the KDD99 dataset, and its features are more plentiful. The data categories include normal data and nine types of attacks. The features include *flow features*, *basic features*, *content features*, *time features*, *additional features*, and *labeled features*. The UNSW-NB15 is representative of new IDS datasets, and has been used in some recent studies. Although the influence of UNSW-NB15 is currently inferior to that of KDD99, it is necessary to construct new datasets for developing new IDS based on machine learning.

4. Research on Machine Learning-Based IDSs

Machine learning is a type of data driven method in which understanding the data is the first step. Thus, we adopt the type of data source of as the main classification thread, as shown in Figure 1. In this section, we introduce various ways to apply machine learning to IDS design for different data types. The different types of data reflect different attack behaviors, which include host behaviors and network behaviors. Host behaviors are reflected by system logs, and network behaviors are reflected by network traffic. There are multiple attack types, each of which has a unique pattern. Thus, selecting appropriate data sources is required to detect different attacks according to the attack characteristics. For instance, one salient feature of a DOS attack is to send many packets within a very short period of

time; therefore flow data is suitable for detecting a DOS attack. A covert channel involves data-leaking activity between two specific IP addresses, which is more suited to detection from session data.

4.1. Packet-Based Attack Detection

Packets, which are the basic units of network communication, represent the details of each communication. Packets consist of binary data, meaning that they are incomprehensible unless they are first parsed. A packet consists of a header and application data. The headers are structured fields that specify IP addresses, ports and other fields specific to various protocols. The application data portion contains the payload from the application layer protocols. There are three advantages to using packets as IDS data sources: (1) Packets contain communication contents; thus, they can effectively be used to detect U2L and R2L attacks. (2) Packets contain IPs and timestamps; thus, they can locate the attack sources precisely. (3) Packets can be processed instantly without caching; thus, detection can occur in real time. However, individual packets do not reflect the full communication state nor the contextual information of each packet, so it is difficult to detect some attacks, such as DDOS. The detection methods based on packets mainly include packet parsing methods and payload analysis methods.

4.1.1. Packet Parsing-Based Detection

Various types of protocols are used in network communications, such as HTTP and DNS. These protocols have different formats; the packet parsing-based detection methods primarily focus on the protocol header fields. The usual practice is to extract the header fields using parsing tools (such as Wireshark or the Bro) and then to treat the values of the most important fields as feature vectors. Packet parsing-based detection methods apply to shallow models.

The header fields provide basic packet information from which feature can be extracted used with using classification algorithms to detect attacks. Mayhew et al. [40] proposed an SVM- and K-means-based packet detection method. They captured packets from a real enterprise network and parsed them with Bro. First, they grouped the packets according to protocol type. Then, they clustered the data with the K-means++ algorithm for the different protocol datasets. Thus, the original dataset was grouped into many clusters, where the data from any given cluster were homologous. Next, they extracted features from the packets and trained SVM models on each cluster. Their precision scores for HTTP, TCP, Wiki, Twitter, and E-mail protocols reached 99.6%, 92.9%, 99%, 96%, and 93%, respectively.

In packet parsing-based detection, unsupervised learning is a common way to solve the high false alarm rate problem. Hu et al. [41] proposed a fuzzy C-means based packet detection method. The fuzzy C mean algorithm introduces fuzzy logic into the standard K-means algorithm such that samples belong to a cluster with a membership degree rather than as a Boolean value such as 0 or 1. They used Snort to process the DARPA 2000 dataset, extracting Snort alerts, source IPs, destination IPs, source ports, destination ports, and timestamps. Then, they used this information to form feature vectors and distinguished false alerts from true alerts by clustering the packets. To reduce the influence of initialization, they ran the clustering algorithms ten times. The results showed that the fuzzy C-means algorithm reduced the false alarm rate by 16.58% and the missed alarm rate by 19.23%.

4.1.2. Payload Analysis-Based Detection

Apart from packet parsing-based detection, payload analysis-based detection places emphasis on the application data. The payload analysis-based methods are suitable for multiple protocols because they do not need to parse the packet headers.

As a type of unstructured data, payloads can be processed directly by deep learning models [42]. It should be noted that this method does not include encrypted payloads. Shallow models depend on manual features and private information in packets, leading to high labor costs and privacy leakage problems. Deep learning methods learn features from raw data without manual intervention. Min et al. [43] utilized a text-based CNN to detect attacks from payloads. They conducted

experiments on the ISCX 2012 dataset and detected attacks with both statistical and content features. The statistical features mainly came from packet headers and included protocols, IPs, and ports. The content features came from the payloads. First, payloads from different packets were concatenated. Next, the concatenated payloads were encoded by skip-gram word embedding. Then, the content features were extracted with a CNN. Finally, they trained a random forest model to detect attacks. The final model reached an accuracy of 99.13%.

Combining various payload analysis techniques can achieve comprehensive content information, which is able to improve the effect of the IDS. Zeng et al. [44] proposed a payload detection method with multiple deep learning models. They adopted three deep learning models (a CNN, an LSTM, and a stacked autoencoder) to extract features from different points of view. Among these, the CNN extracted local features, the RNN extracted time series features, and the stacked autoencoder extracted text features. The accuracy of this combined approach reached 99.22% on the ISCX 2012 dataset.

Extracting payload features with unsupervised learning is also an effective detection method. Yu et al. [45] utilized a convolutional autoencoder to extract payload features and conducted experiments on the CTU-UNB dataset. This dataset includes the raw packets of 8 attack types. To take full advantage of convolutions, they first converted the packets into images. Then, they trained a convolutional autoencoder model to extract features. Finally, they classified packets using learned features. The precision, recall and F-measure on the test set reached 98.44%, 98.40%, and 98.41% respectively.

To enhance the robustness of IDSs, adversarial learning becomes a novel approach. Adversarial learning can be used for attacks against IDS. Meanwhile, it is also a novel way to improve detection accuracy of IDS. Rigaki et al. [46] used a GAN to improve the malware detection effect. To evade detection, malware applications try to generate packets similar to normal packets. Taking the malware FLU as an example, the command & control (C & C) packets are very similar to packets generated by Facebook. They configured a virtual network system with hosts, servers, and an IPS. Then, they started up the malware FLU and trained a GAN model. The GAN guided the malware to produce packets similar to Facebook. As the training epochs increased, the packets blocked by the IPS decreased and packet that passed inspection increased. The result was that the malicious packets generated by the GAN were more similar to normal packets. Then, by analyzing the generated packets, the robustness of the IPS was improved.

4.2. Flow-Based Attack Detection

Flow data contains packets grouped in a period, which is the most widespread data source for IDSs. The KDD99 and the NSL-KDD datasets are both flow data. Detecting attacks with flow has two benefits: (1) Flow represents the whole network environment, which can detect most attacks, especially DOS and Probe. (2) Without packet parsing or session restructuring, flow preprocessing is simple. However, flow ignores the content of packets; thus, its detection effect for U2R and R2L is unsatisfactory. When extracting flow features, packets must be cached packets; thus, it involves some hysteresis. Flow-based attack detection mainly includes feature engineering and deep learning methods. In addition, the strong heterogeneity of flow may cause poor detection effects. Traffic grouping is the usual solution to this problem.

4.2.1. Feature Engineering-Based Detection

Traditional machine learning models cannot directly address flow data; therefore, feature engineering is an essential step before these models can be applied. Feature engineering-based methods adopt a "feature vectors + shallow models" mode. The feature vectors are suitable for most machine learning algorithms. Each dimension of the feature vectors has clear interpretable semantics. The common features include the average packet length, the variance in packet length, the ratio of TCP to UDP, the proportion of TCP flags, and so on. The advantages of these types of detection methods are that they are simple to implement, highly efficient, and can meet real-time requirements.

The existing feature engineering-based IDSs often have high detection accuracy but suffer from a high false alarm rate. One solution is to combine many weak classifiers to obtain a strong classifier. Goeschel et al. [47] proposed a hybrid method that included SVM, decision tree, and Naïve Bayes algorithms. They first trained an SVM model to divide the data into normal or abnormal samples. For the abnormal samples, they utilized a decision tree model to determine specific attack types. However, a decision tree model can identify only known attacks, not unknown attacks. Thus, they also applied a Naïve Bayes classifier to discover unknown attacks. By taking advantage of three different classifier types this hybrid method achieved an accuracy of 99.62% and a false alarm rate of 1.57% on the KDD99 dataset.

Another research objective is to accelerate the detection speed. Kuttranont et al. [48] proposed a KNN-based detection method and accelerated calculation via parallel computing techniques running on a graphics processing unit (GPU). They modified the neighbor-selecting rule of the KNN algorithm. The standard KNN selects the top K nearest samples as neighbors, while the improved algorithm selects a fixed percentage (such as 50%) of the neighboring samples as neighbors. The proposed method considers the unevenness of data distribution and performs well on sparse data. These experiments were conducted using the KDD99 dataset, achieving an accuracy of 99.30%. They also applied parallel computing and the GPU to accelerating calculation. The experimental results showed that the method with the GPU was approximately 30 times faster than that without the GPU.

The unsupervised learning methods are also applied to IDS, a typical way is to divide data with clustering algorithms. The standard K-means algorithm is inefficient on big datasets. To improve detection efficiency, Peng et al. [13] proposed an improved K-means detection method with mini batch. They first carried out data preprocessing on the KDD99 dataset. The nominal features were transformed into numerical types, and each dimension of the features was normalized by the max-min method. Then, they reduced the dimensions using the principal components analysis (PCA) algorithm. Finally, they clustered the samples with the K-means algorithm, but they improved K-means from two aspects. (1) They altered the method of initialization to avoid becoming stuck in a local optimum. (2) They introduced the mini-batch trick to decrease the running time. Compared with the standard K-means, the proposed method achieved higher accuracy and runtime efficiency.

4.2.2. Deep Learning-Based Detection

Feature engineering depends on domain knowledge, and the quality of features often becomes a bottleneck of detection effects. Deep learning-based detection methods learn feature automatically. These types of methods work in an end-to-end fashion and are gradually becoming the mainstream approach in IDS studies.

Deep learning methods can directly process raw data, allowing them to learn features and perform classification at the same time. Potluri et al. [49] proposed a CNN-based detection method. They conducted experiments on the NSL-KDD and the UNSW-NB 15 datasets. The data type in these datasets is a feature vector. Because CNNs are good at processing 2-dimensional (2D) data, they first converted the feature vectors into images. Nominal features were one-hot coded, and the feature dimensions increased from 41 to 464. Then, each 8-byte chunk was transformed into one pixel. Blank pixels were padded with 0. The end result was that the feature vectors were transformed into images of 8*8 pixels. Finally, they constructed a three-layer CNN to classify the attacks. They compared their model with other deep networks (ResNet 50 and GoogLeNet), and the proposed CNN performed best, reaching accuracies of 91.14% on the NSL-KDD and 94.9% on the UNSW-NB 15.

Unsupervised deep learning models can also be used to extract features; then, shallow models can be used to perform classification. Zhang et al. [50] extracted features with a sparse autoencoder and detected attacks with an XGBoost model. They used data from the NSL-KDD dataset. Due to the imbalanced nature of this dataset, they sampled the dataset using SMOTE. The SMOTE algorithm oversamples the minority classes and divides the majority classes into many subclasses so that every class is balanced. The sparse autoencoder introduces a sparsity constraint into the original autoencoder,

enhancing its ability to detect unknown samples. Finally, they classified the data using an XGBoost model. Their model achieved accuracies on the Normal, DOS, Probe, R2L, and U2R classes of 99.96%, 99.17%, 99.50%, 97.13%, and 89.00%, respectively.

Deep learning models have made great strides in big data analysis; however, their performances are not ideal on small or unbalanced datasets. Adversarial learning approaches can improve the detection accuracy on small datasets. Zhang et al. [51] conducted data augmentation with a GAN. The KDD99 dataset is both unbalanced and lacks new data, which leads to poor generalizability of machine learning models. To address these problems, they utilized a GAN to expand the dataset. The GAN model generated data similar to the flow data of KDD99. Adding this generated data to the training set allows attack variants to be detected. They selected 8 types of attacks and compared the accuracies achieved on the original dataset compared to the expanded dataset. The experimental results showed that adversarial learning improved 7 accuracies in 8 attack types.

4.2.3. Traffic Grouping-Based Detection

Flow includes all traffic within a period, and many types of traffics may act as white noise in attack detection. Training machine learning models with such data probably leads to overfitting. One natural approach is to group traffic to decrease heterogeneity. The grouping methods include protocol-based and data-based methods.

The traffic features of various protocols have significant differences; thus, grouping traffic by protocol is a valid step toward improving accuracy. Teng et al. [52] proposed an SVM detection method based on protocol grouping using the data of the KDD99 dataset, which involves various protocols. They first divided the dataset based on protocol type, and considered only TCP, UDP, and ICMP protocols. Then, according to the characteristics of these different protocols, they selected features for each subdataset. Finally, they trained SVM models on the 3 subdatasets, obtaining an average accuracy of 89.02%.

Grouping based on data characteristics is another traffic grouping approach. One typical method is clustering. Ma et al. [53] proposed a DNN and spectral clustering-based detection method. The heterogeneity of flow may cause low accuracy. Therefore, they first divided the original dataset into 6 subsets, in which each subset was highly homogeneous. Then, they trained DNN models on every subset. The accuracy of their approach on the KDD99 and the NSL-KDD datasets reached 92.1%.

4.3. Session-Based Attack Detection

A session is the interaction process between two terminal applications and can represent high-level semantics. A session is usually divided on the basis of a 5-tuple (client IP, client port, server IP, server port, and protocol). There are two advantages of detection using sessions. (1) Sessions are suitable for detecting an attack between specific IP addresses, such as tunnel and Trojan attacks. (2) Sessions contain detailed communications between the attacker and the victim, which can help localize attack sources. However, session duration can vary dramatically. As a result, a session analysis sometimes needs to cache many packets, which may increase lag. The session-based detection methods primarily include statistics-based features and sequence-based features.

4.3.1. Statistic-Based Feature Detection Methods

Session statistical information includes the fields in packet headers, the number of packets, the proportion of packets coming from different directions, and so on. This statistical information is used to compose feature vectors suitable for shallow models. The sessions have high layer semantics; thus, they are easily described by rules. Decision tree or rule-based models may be appropriate methods. Unfortunately, the methods based on statistical features ignore the sequence information, and they have difficulties detecting intrusions related to communication content.

Because statistical information includes the basic features of sessions, supervised learning methods can utilize such information to differentiate between normal sessions and abnormal sessions.

The existing session-based detection methods often face problems of low accuracy and have high runtime costs. Ahmim et al. [54] proposed a hierarchical decision tree method in which, reduce the detection time, they analyzed the frequency of different types of attacks and designed the detection system to recognize specific attacks. They used data from the CICIDS 2017 dataset that included 79-dimensional features and 15 classes. The proposed detection system had a two-layer structure. The first layer consisted of two independent classifiers (i.e., a decision tree and a rule-based model), which processed part of the features. The second layer was a random forest classifier, which processed all the features from the dataset as well as the output of the first layer. They compared multiple machine learning models on 15 classes; their proposed methods performed best on 8 of the 15 classes. Moreover, the proposed method had low time consumption, reflecting its practicability.

Session-based detection using supervised learning models depends on expert knowledge, which is difficult to expand to new scenarios. To address this problem, Alseiari et al. [55] proposed an unsupervised method to detect attacks in smart grids. Due to the lack of smart grid datasets, they constructed a dataset through simulation experiments. First, they captured and cached packets to construct sessions. Then, they extracted 23-dimensional features from the sessions. Next, they utilized mini batch K-means to divide the data into many clusters. Finally, they labeled the clusters. This work was based on two hypotheses. The first was that normal samples were the majority. The second one was that the distances among the normal clusters were relatively short. When the size of a cluster was less than 25% of the full sample amount or a cluster centroid was far away from all other the other cluster centroids, that cluster was judged as abnormal. No expert knowledge was required for any part of this process. The proposed methods were able to detect intrusion behaviors in smart grids effectively and locate the attack sources while holding the false alarm rate less to than 5%.

4.3.2. Sequence Feature-Based Detection

Different from flow, the packets in sessions have a strict order relationship. The sequence features mainly contain the packet length sequence and the time interval sequence. Analyzing the sequence can obtain detailed session interaction information. Most machine learning algorithms cannot deal with sequences, and related methods are relatively rare. At present, most sequence feature-based detection adopts the RNN algorithm.

Encoding raw data is a common preprocessing step for RNN methods. The bag of words (BoW) model is a frequently used text processing technology. Yuan et al. [56] proposed a DDOS detection method based on the LSTM using UNB ISCX 2012 dataset. They first extracted 20-dimensional features from the packets and encoded them with BoW. Then, they concatenated the packets in sequence, resulting in matrices with a size of m*n, where m was the number of packets in a session and n was the dimension of a packet, and both m and n were variable. Finally, they trained a CNN to extract local features and an LSTM to classify the sessions. They provided comprehensive experimental results, reaching accuracy, precision, recall, and F-measure scores of 97.606%, 97.832%, 97.378%, and 97.601%, respectively.

One of the drawbacks of the BoW is that it is unable to represent the similarity between words. Word embedding approaches overcome that problem. Radford et al. [57] proposed a session detection method based on a bi-LSTM. Because LSTMs had made great strides in NLP, they expressed the sessions as a specific language. They conducted experiments on the ISCX IDS dataset. First, they grouped packets on the basis of IP addresses to obtain sessions. Then, they encoded the sessions with the word embedding. Finally, they trained an LSTM model to predict abnormal sessions. To utilize the contextual information, they adopted a bi-LSTM model to learn the sequence features in two directions.

In addition to text processing technology, the character-level CNN is a novel encoding method. Wang et al. [58] proposed a hierarchical deep learning detection method in which a session contains not only packet contents but also the packet time sequence. Then, they designed a hierarchical deep learning method using a CNN to learn the low-level spatial features and an LSTM to learn the high-level time features, where the time features are based on the spatial features. They conducted

experiments on the DARPA 1998 and the ISCX 2012 datasets. They first applied the CNN to extract spatial features from packets. Next, they concatenated the spatial features in sequence and extracted time features using the LSTM model. The resulting model achieved accuracies between 99.92% and 99.96%, and detection rates between 95.76% and 98.99%.

4.4. Log-Based Attack Detection

Logs are the activity records of operating systems or application programs; they include system calls, alert logs, and access records. Logs have definite semantics. There are three benefits to using logs as a data source in IDSs. (1) Logs include detailed content information suitable for detecting SQL injection, U2R, and R2L attacks. (2) Logs often carry information about users and timestamps that can be used to trace attackers and reveal attack times. (3) Logs record the complete intrusion process; thus, the result is interpretable. However, one problem is that log analysis depends on cyber security knowledge. Additionally, the log formats of different application programs do not have identical formats, resulting in low scalability. The log-based attack detection primarily includes hybrid methods involving rules and machine learning, log feature extraction-based methods, and text analysis-based methods.

4.4.1. Rule and Machine Learning-Based Hybrid Methods

Hybrid methods combine rule-based detection and machine learning, which together achieve better performances than do single detection systems. Many rule-based detection systems (e.g., Snort) generate masses of alerts; however, most of the alerts involve only operations that do not match the rules; therefore, these are often not real intrusion behaviors. The hybrid methods take the log output of the rule-based systems as inputs; then, machine learning models are used to filter out the meaningless alerts.

Many IDSs suffer from high false alarm rates, which cause real attacks to be embedded among many meaningless alerts. Ranking alerts via machine learning models forms a possible solution. To reduce the false alarm rate, Meng et al. [59] proposed a KNN method to filter alarms. They conducted experiments in a real network environment and generated alerts using Snort. Then, they trained a KNN model to rank the alerts. There were 5 threat levels in total in their experiment, and the results showed that the KNN model reduced the number of alerts by 89%.

Some IDSs perform a function similar to human interaction, in which alerts are ranked by machine learning to reduce analyst workloads. McElwee et al. [60] proposed an alert filtering method based on a DNN. They first collected the log generated by McAfee. Then, they trained a DNN model to find important security events in the logs. Next, the extracted important events were analyzed by security experts. Then, the analysis results were used as training data to enhance the DNN model, forming an interaction and promotion cycle. The proposed hybrid system can reduce analyst workloads and accelerate security analyses.

4.4.2. Log Feature Extraction-Based Detection

This method involves extracting log features according to domain knowledge and discovering abnormal behaviors using the extracted features, which is suitable for most machine learning algorithms. Using a sliding window to extract features is a common approach. The sliding window makes use of the contextual information contained in logs. In addition, the sliding window is a streaming method that has the benefit of low delay.

Intrusion behaviors may leave traces of system calls, and analyzing these system calls with classification algorithms can detect intrusions. Tran et al. [61] proposed a CNN method to analyze system calls. Every underlying operation that involves the operating system will use system calls; thus, analyzing the system call path can reproduce the complete intrusion process. They conducted experiments on the NGIDS-DS and the ADFA-LD datasets, which include a series of system calls. First, they extracted features with a sliding window. Then, they applied a CNN model to perform

classification. The CNN was good at finding local relationships and detecting abnormal behaviors from system calls.

Model interpretation is another important research direction, which has attracted extensive attention. Tuor et al. [62] proposed an interpretable deep learning detection method using data from the CERT Insider Threat dataset, which consists of system logs. They first extracted 414-dimensional features using a sliding window. Then, they adopted a DNN and an RNN to classify logs. The DNN detected attacks based on the log contents, and the RNN detected attacks based on the log sequences. The proposed methods reduced the analysis workload by 93.5% and reached a detection rate of 90%. Furthermore, they decomposed the abnormal scores into the contributions of each behavior, which was a helpful analysis. Interpretable models are more convincing than are uninterpretable models.

Some logs lack labeled information; consequently, supervised learning is inappropriate. Unsupervised learning methods are usually used with unlabeled logs. Bohara et al. [63] proposed an unsupervised learning detection method in the enterprise environment. They conducted experiments on the VAST 2011 Mini Challenge 2 dataset and extracted features from the host and network logs. Due to the different influences of each feature, they selected features using the Pearson correlation coefficient. Then, they clustered the logs with the K-means and DBSCAN algorithms. By measuring the salient cluster features, the clusters were associated with abnormal behaviors. Finally, they analyzed the abnormal clusters manually to determine the specific attack types.

4.4.3. Text Analysis-Based Detection

The text analysis-based detection regards logs as plain text. The methods utilize mature text processing techniques such as the n-gram to analyze logs. Compared with log feature extraction-based methods, this method understands log content at the semantic level and therefore has stronger interpretability.

In log-based detection, extracting text features from logs and then performing classification is the usual approach. When analyzing texts, a small number of keywords have large impacts on the whole text. Thus, the keywords in the field of cyber security aid in improving the detection effect. Uwagbole et al. [64] proposed an SQL-injection detection method for the Internet of Things (IoT). They collected and labeled logs from a real environment. The logs provide the contextual information of the SQL injection attack. First, they extracted 479,000 high-frequency words from the logs and then added 862 keywords that appear in SQL queries to compose a dictionary. Then, they removed duplicate and missing records from the log and balanced the data with SMOTE. Next, they extracted features using the n-gram algorithm and selected features using Chi-square tests. Finally, they trained an SVM model to perform classification, achieving accuracy, precision, recall, and F-measure scores of 98.6%, 97.4%, 99.7% and 98.5%, respectively.

In an actual network environment, normal samples are in the majority, and abnormal samples are rare. One-class classification, a type of unsupervised learning method, uses only normal samples for training, which solves the problem of a lack of abnormal samples. Vartouni et al. [65] proposed a web attack detection method based on the isolate forest model. They used the data of the CSIC 2010 dataset. First, they extracted 2572-dimensional features from HTTP logs with the n-gram. Then, they utilized an autoencoder to remove irrelevant features. Finally, they trained an isolation forest model to discover abnormal webs, which reached an accuracy of 88.32%.

5. Challenges and Future Directions

Table 5 lists papers on machine learning based IDSs which are introduced in this survey. It shows that deep learning methods have become a research hotspot (26 papers are listed, 14 papers adopt deep learning methods). KDD99 and NSL-KDD datasets are still widespread used. Although machine learning methods have made great strides in the field of intrusion detection, the following challenges still exist.

Table 5. Methods and papers on machine learning based IDSs.

Methods	Papers	Data Sources	Machine Learning Algorithms	Datasets
Packet parsing	Mayhew et al. [40]	Packet	SVM and K-means	Private dataset
	Hu et al. [41]	Packet	Fuzzy C-means	DARPA 2000
Payload analysis	Min et al. [43]	Packet	CNN	ISCX 2012
	Zeng et al. [44]	Packet	CNN, LSTM, and autoencoder	ISCX 2012
	Yu et al. [45]	Packet	Autoencoder	CTU-UNB
	Rigak et al. [46]	Packet	GAN	Private dataset
Statistic feature for flow	Goeschel et al. [47]	Flow	SVM, decision tree, and Naïve Bayes	KDD99
	Kuttranont et al. [48]	Flow	KNN	KDD99
	Peng et al. [13]	Flow	K-means	KDD99
Deep learning for flow	Potluri et al. [49]	Flow	CNN	NSL-KDD and UNSW-NB15
	Zhang et al. [50]	Flow	Autoencoder and XGBoost	NSL-KDD
	Zhang et al. [51]	Flow	GAN	KDD99
Traffic grouping	Teng et al. [52]	Flow	SVM	KDD99
	Ma et al. [53]	Flow	DNN	KDD99 and NSL-KDD
Statistic feature for session	Ahmim et al. [54]	Session	Decision tree	CICIDS 2017
	Alseiari et al. [55]	Session	K-means	Private dataset
Sequence feature for session	Yuan et al. [56]	Session	CNN and LSTM	ISCX 2012
	Radford et al. [57]	Session	LSTM	ISCX IDS
	Wang et al. [58]	Session	CNN	DARPA 1998 and ISCX 2012
Rule-based	Meng et al. [59]	Log	KNN	Private dataset
	McElwee et al. [60]	Log	DNN	Private dataset
Log feature extraction with sliding window	Tran et al. [61]	Log	CNN	NGIDS-DS and ADFA-LD
	Tuor et al. [62]	Log	DNN and RNN	CERT Insider Threat
	Bohara et al. [63]	Log	K-means and DBSCAN	VAST 2011 Mini Challenge 2
Text analysis	Uwagbole et al. [64]	Log	SVM	Private dataset
	Vartouni et al. [65]	Log	Isolate forest	CSIC 2010 dataset

(1) Lack of available datasets. The most widespread dataset is currently KDD99, which has many problems, and new datasets are required. However, constructing new datasets depends on expert knowledge, and the labor cost is high. In addition, the variability of the Internet environment intensifies the dataset shortage. New types of attacks are emerging, and some existing datasets are too old to reflect these new attacks. Ideally, datasets should include most of the common attacks and correspond to current network environments. Moreover, the available datasets should be representative, balanced and have less redundancy and less noise. Systematic datasets construction and incremental learning may be solutions to this problem.

(2) Inferior detection accuracy in actual environments. Machine learning methods have a certain ability to detect intrusions, but they often do not perform well on completely unfamiliar data. Most the existing studies were conducted using labeled datasets. Consequently, when the dataset does not cover all typical real-world samples, good performance in actual environments is not guaranteed—even if the models achieve high accuracy on test sets.

(3) Low efficiency. Most studies emphasize the detection results; therefore, they usually employ complicated models and extensive data preprocessing methods, leading to low efficiency. However, to reduce harm as much as possible, IDSs need to detect attacks in real time. Thus, a trade-off exists between effect and efficiency. Parallel computing [66,67] approaches using GPUs [48,68,69] are common solutions.

From summarizing the recent studies, we can conclude that the major trends of IDS research lie in the following aspects.

(1) Utilizing domain knowledge. Combining domain knowledge with machine learning can improve the detection effect, especially when the goal is to recognize specific types of attacks in specific application scenarios.

- The rule-based detection methods have low false alarm rates but high missed alarm rates include considerable expert knowledge. In contrast, the machine learning methods usually have high false alarm rates and low missed alarm rates. The advantages of both methods are complementary. Combining machine learning methods with rule-based systems, such as Snort [70–73], can result in IDSs with low false alarm rates and low missed alarm rates.
- For specific types of attacks, such as DOS [74–79], botnet [80], and phishing web [81], proper feature must be extracted according to the attack characteristics that can be abstracted using domain knowledge.
- For specific application scenarios, such as cloud computing [82,83], IoT [84–86], and smart grids [87,88], domain knowledge can be used to provide the environmental characteristics that are helpful in data collection and data preprocessing.

(2) Improving machine learning algorithms. Improvements in machine learning algorithms are the main means to enhance the detection effect. Thus, studies involving deep learning and unsupervised learning methods has an increasing trend.

- Compared with shallow models, deep learning methods learn features directly from raw data, and their fitting ability is stronger. Deep learning models with deep structures can be used for classification, feature extraction, feature reduction, data denoising, and data augmentation tasks. Thus, deep learning methods can improve IDSs from many aspects.
- Unsupervised learning methods require no labeled data; thus they can be used even when a dataset shortage exists. The usual approach involves dividing data using an unsupervised learning model, manually labeling the clusters, and then training a classification model with supervised learning [89–92].

(3) Developing practical models. Practical IDSs not only need to have high detection accuracy but also high runtime efficiency and interpretability.

- In attack detection, the real-time requirement is essential. Thus, one research direction is to improve the efficiency of machine learning models. Reducing the time required for data collection and storage is also of concern.
- Interpretability is important for practical IDSs. Many machine learning models, especially deep learning models, are black boxes. These models report only the detection results and have no interpretable basis [93]. However, every cyber security decision should be made cautiously. An output result with no identifiable reason is not convincing. Thus, an IDS with high accuracy, high efficiency and interpretability is more practical.

6. Conclusions

The paper first proposes an IDS taxonomy that takes data sources as the main thread to present the numerous machine learning algorithms used in this field. Based on this taxonomy, we then analyze and discuss IDSs applied to various data sources, i.e., logs, packets, flow, and sessions. IDSs aim to detect attacks, therefore it is vital to select proper data source according to attack characteristics. Logs contain detailed semantic information, which are suitable for detecting SQL injection, U2R, and R2L attacks. And packets provide communication contents, which are fit to detect U2L and R2L attacks. Flow represents the whole network environment, which can detect DOS and Probe attack. Sessions, which reflect communication between clients and servers, can be used to detect U2L, R2L, tunnel and Trojan attacks. For IDSs using these different data types, the paper emphasizes machine learning techniques (especially deep learning algorithms) and application scenarios.

Deep learning models are playing an increasingly important role and have become an outstanding direction of study. Deep learning approaches include multiple deep networks which can be used to improve the performance of IDSs. Compared with shallow machine learning models, deep learning models own stronger fitting and generalization abilities. In addition, deep learning approaches are independent of feature engineering and domain knowledge, which takes an outstanding advantage over shallow machine learning models. However, the running time of deep learning models are often too long to meet the real-time requirement of IDSs.

By summarizing the recent typical studies, this paper analyzes and refines the challenges and future trends in the field to provide references to other researchers conducting in-depth studies. Lacking of available datasets may be the biggest challenge. So unsupervised learning and incremental learning approaches have broad development prospects. For practical IDSs, interpretability is essential. Because interpretable models are convincing and can guide users to make a decision. The interpretability of models may become an important research direction about IDSs in the future.

Author Contributions: Writing—original draft preparation, H.L.; writing—review and editing, H.L.; writing—review and editing, B.L.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Anderson, J.P. *Computer Security Threat Monitoring and Surveillance*; Technical Report; James P. Anderson Company: Philadelphia, PA , USA, 1980.
2. Michie, D.; Spiegelhalter, D.J.; Taylor, C. *Machine Learning, Neurall and Statistical Classification*; Ellis Horwood Series in Artificial Intelligence: New York, NY, USA, 1994; Volume 13.
3. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 1153–1176. [CrossRef]
4. Xin, Y.; Kong, L.; Liu, Z.; Chen, Y.; Li, Y.; Zhu, H.; Gao, M.; Hou, H.; Wang, C. Machine learning and deep learning methods for cybersecurity. *IEEE Access* **2018**, *6*, 35365–35381. [CrossRef]
5. Agrawal, S.; Agrawal, J. Survey on anomaly detection using data mining techniques. *Procedia Comput. Sci.* **2015**, *60*, 708–713. [CrossRef]
6. Denning, D.E. An intrusion-detection model. *IEEE Trans. Softw. Eng.* **1987**, 222–232. [CrossRef]

7. Heberlein, L.T.; Dias, G.V.; Levitt, K.N.; Mukherjee, B.; Wood, J.; Wolber, D. A network security monitor. In Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 7–9 May 1990; pp. 296–304.
8. Kuang, F.; Zhang, S.; Jin, Z.; Xu, W. A novel SVM by combining kernel principal component analysis and improved chaotic particle swarm optimization for intrusion detection. *Soft Comput.* **2015**, *19*, 1187–1199. [CrossRef]
9. Syarif, A.R.; Gata, W. Intrusion detection system using hybrid binary PSO and K-nearest neighborhood algorithm. In Proceedings of the 2017 11th International Conference on Information & Communication Technology and System (ICTS), Surabaya, Indonesia, 31 October 2017; pp. 181–186.
10. Pajouh, H.H.; Dastghaibyfard, G.; Hashemi, S. Two-tier network anomaly detection model: A machine learning approach. *J. Intell. Inf. Syst.* **2017**, *48*, 61–74. [CrossRef]
11. Mahmood, H.A. Network Intrusion Detection System (NIDS) in Cloud Environment based on Hidden Naïve Bayes Multiclass Classifier. *Al-Mustansiriyah J. Sci.* **2018**, *28*, 134–142. [CrossRef]
12. Shah, R.; Qian, Y.; Kumar, D.; Ali, M.; Alvi, M. Network intrusion detection through discriminative feature selection by using sparse logistic regression. *Future Internet* **2017**, *9*, 81. [CrossRef]
13. Peng, K.; Leung, V.C.; Huang, Q. Clustering approach based on mini batch kmeans for intrusion detection system over big data. *IEEE Access* **2018**, *6*, 11897–11906. [CrossRef]
14. Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.A. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, 5–9 July 2008, pp. 1096–1103.
15. Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **2010**, *11*, 3371–3408.
16. Deng, J.; Zhang, Z.; Marchi, E.; Schuller, B. Sparse autoencoder-based feature transfer learning for speech emotion recognition. In Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, Geneva, Switzerland, 2–5 September 2013; pp. 511–516.
17. Hinton, G.E. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin, Germany, 2012; pp. 599–619.
18. Hinton, G.E.; Osindero, S.; Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef] [PubMed]
19. Boureau, Y.l.; Cun, Y.L.; Ranzato, M.A. Sparse feature learning for deep belief networks. In Proceedings of the 21st Annual Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–10 December 2008; pp. 1185–1192.
20. Zhao, G.; Zhang, C.; Zheng, L. Intrusion detection using deep belief network and probabilistic neural network. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 21–24 July 2017; Volume 1, pp. 639–642.
21. Alrawashdeh, K.; Purdy, C. Toward an online anomaly intrusion detection system based on deep learning. In Proceedings of the 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, USA, 18–20 December 2016; pp. 195–200.
22. Yang, Y.; Zheng, K.; Wu, C.; Niu, X.; Yang, Y. Building an effective intrusion detection system using the modified density peak clustering algorithm and deep belief networks. *Appl. Sci.* **2019**, *9*, 238. [CrossRef]
23. Sharif Razavian, A.; Azizpour, H.; Sullivan, J.; Carlsson, S. CNN features off-the-shelf: An astounding baseline for recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 806–813.
24. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the 26th Annual Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105. [CrossRef]
25. Lawrence, S.; Giles, C.L.; Tsoi, A.C.; Back, A.D. Face recognition: A convolutional neural-network approach. *IEEE Trans. Neural Netw.* **1997**, *8*, 98–113. [CrossRef]
26. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada , 26–31 May 2013; pp. 6645–6649.

27. Graves, A.; Jaitly, N. Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 1764–1772.
28. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. In Proceedings of the Annual Conference on Neural Information Processing Systems 2014, Montreal, QC, Canada, 8–13 December 2014; pp. 3104–3112.
29. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
30. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, arXiv:1412.3555.
31. Schuster, M.; Paliwal, K.K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **1997**, *45*, 2673–2681. [CrossRef]
32. Ribeiro, M.T.; Singh, S.; Guestrin, C. Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144.
33. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. In Proceedings of the Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 4765–4774.
34. Li, J.; Monroe, W.; Jurafsky, D. Understanding neural networks through representation erasure. *arXiv* **2016**, arXiv:1612.08220.
35. Fong, R.C.; Vedaldi, A. Interpretable explanations of black boxes by meaningful perturbation. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 3429–3437.
36. DARPA1998 Dataset. 1998. Available online: http://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset (accessed on 16 October 2019).
37. KDD99 Dataset. 1999. Available online: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed on 16 October 2019).
38. NSL-KDD99 Dataset. 2009. Available online: https://www.unb.ca/cic/datasets/nsl.html (accessed on 16 October 2019).
39. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications And Information Systems Conference (MilCIS), Canberra, Australia, 10–12 November 2015; pp. 1–6.
40. Mayhew, M.; Atighetchi, M.; Adler, A.; Greenstadt, R. Use of machine learning in big data analytics for insider threat detection. In Proceedings of the MILCOM 2015-2015 IEEE Military Communications Conference, Canberra, Australia, 10–12 November 2015; pp. 915–922.
41. Hu, L.; Li, T.; Xie, N.; Hu, J. False positive elimination in intrusion detection based on clustering. In Proceedings of the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, China, 15–17 August 2015; pp. 519–523.
42. Liu, H.; Lang, B.; Liu, M.; Yan, H. CNN and RNN based payload classification methods for attack detection. *Knowl.-Based Syst.* **2019**, *163*, 332–341. [CrossRef]
43. Min, E.; Long, J.; Liu, Q.; Cui, J.; Chen, W. TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest. *Secur. Commun. Netw.* **2018**, *2018*, 4943509. [CrossRef]
44. Zeng, Y.; Gu, H.; Wei, W.; Guo, Y. *Deep − Full − Range*: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* **2019**, *7*, 45182–45190. [CrossRef]
45. Yu, Y.; Long, J.; Cai, Z. Network intrusion detection through stacking dilated convolutional autoencoders. *Secur. Commun. Netw.* **2017**, *2017*, 4184196. [CrossRef]
46. Rigaki, M.; Garcia, S. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 70–75.
47. Goeschel, K. Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive Bayes for off-line analysis. In Proceedings of the SoutheastCon 2016, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–6.
48. Kuttranont, P.; Boonprakob, K.; Phaudphut, C.; Permpol, S.; Aimtongkhamand, P.; KoKaew, U.; Waikham, B.; So-In, C. Parallel KNN and Neighborhood Classification Implementations on GPU for Network Intrusion Detection. *J. Telecommun. Electron. Comput. Eng. (JTEC)* **2017**, *9*, 29–33.

49. Potluri, S.; Ahmed, S.; Diedrich, C. Convolutional Neural Networks for Multi-class Intrusion Detection System. In *Mining Intelligence and Knowledge Exploration*; Springer: Cham, Switzerland, 2018; pp. 225–238.
50. Zhang, B.; Yu, Y.; Li, J. Network Intrusion Detection Based on Stacked Sparse Autoencoder and Binary Tree Ensemble Method. In Proceedings of the 2018 IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
51. Zhang, H.; Yu, X.; Ren, P.; Luo, C.; Min, G. Deep Adversarial Learning in Intrusion Detection: A Data Augmentation Enhanced Framework. *arXiv* **2019**, arXiv:1901.07949.
52. Teng, S.; Wu, N.; Zhu, H.; Teng, L.; Zhang, W. SVM-DT-based adaptive and collaborative intrusion detection. *IEEE/CAA J. Autom. Sin.* **2017**, *5*, 108–118. [CrossRef]
53. Ma, T.; Wang, F.; Cheng, J.; Yu, Y.; Chen, X. A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. *Sensors* **2016**, *16*, 1701. [CrossRef]
54. Ahmim, A.; Maglaras, L.; Ferrag, M.A.; Derdour, M.; Janicke, H. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 228–233.
55. Alseiari, F.A.A.; Aung, Z. Real-time anomaly-based distributed intrusion detection systems for advanced Metering Infrastructure utilizing stream data mining. In Proceedings of the 2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE), Offenburg, Germany, 20–23 October 2015; pp. 148–153.
56. Yuan, X.; Li, C.; Li, X. DeepDefense: identifying DDoS attack via deep learning. In Proceedings of the 2017 IEEE International Conference on Smart Computing (SMARTCOMP), Hong Kong, China, 29–31 May 2017; pp. 1–8.
57. Radford, B.J.; Apolonio, L.M.; Trias, A.J.; Simpson, J.A. Network traffic anomaly detection using recurrent neural networks. *arXiv* **2018**, arXiv:1803.10769.
58. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* **2017**, *6*, 1792–1806. [CrossRef]
59. Meng, W.; Li, W.; Kwok, L.F. Design of intelligent KNN-based alarm filter using knowledge-based alert verification in intrusion detection. *Secur. Commun. Netw.* **2015**, *8*, 3883–3895. [CrossRef]
60. McElwee, S.; Heaton, J.; Fraley, J.; Cannady, J. Deep learning for prioritizing and responding to intrusion detection alerts. In Proceedings of the MILCOM 2017—2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA, 23–25 October 2017; pp. 1–5.
61. Tran, N.N.; Sarker, R.; Hu, J. An Approach for Host-Based Intrusion Detection System Design Using Convolutional Neural Network. In Proceedings of the International Conference on Mobile Networks and Management, Chiba, Japan, 23–25 September 2017; Springer: Berlin, Germany, 2017; pp. 116–126.
62. Tuor, A.; Kaplan, S.; Hutchinson, B.; Nichols, N.; Robinson, S. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In Proceedings of the Workshops at the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
63. Bohara, A.; Thakore, U.; Sanders, W.H. Intrusion detection in enterprise systems by combining and clustering diverse monitor data. In Proceedings of the Symposium and Bootcamp on the Science of Security, Pittsburgh, PA, USA, 19–21 April 2016; pp. 7–16.
64. Uwagbole, S.O.; Buchanan, W.J.; Fan, L. Applied machine learning predictive analytics to SQL injection attack detection and prevention. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 1087–1090.
65. Vartouni, A.M.; Kashi, S.S.; Teshnehlab, M. An anomaly detection method to detect web attacks using Stacked Auto-Encoder. In Proceedings of the 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), Kerman, Iran, 28 February–2 March 2018; pp. 131–134.
66. Potluri, S.; Diedrich, C. Accelerated deep neural networks for enhanced Intrusion Detection System. In Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 6–9 September 2016; pp. 1–8.
67. Pektaş, A.; Acarman, T. Deep learning to detect botnet via network flow summaries. *Neural Comput. Appl.* **2018**, 1–13. [CrossRef]

68. Kim, J.; Shin, N.; Jo, S.Y.; Kim, S.H. Method of intrusion detection using deep neural network. In Proceedings of the 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju, Korea, 13–16 February 2017; pp. 313–316.
69. Shone, N.; Ngoc, T.N.; Phai, V.D.; Shi, Q. A deep learning approach to network intrusion detection. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 41–50. [CrossRef]
70. Ammar, A. A decision tree classifier for intrusion detection priority tagging. *J. Comput. Commun.* **2015**, *3*, 52. [CrossRef]
71. Patel, J.; Panchal, K. Effective intrusion detection system using data mining technique. *J. Emerg. Technol. Innov. Res.* **2015**, *2*, 1869–1878.
72. Khamphakdee, N.; Benjamas, N.; Saiyod, S. Improving intrusion detection system based on snort rules for network probe attacks detection with association rules technique of data mining. *J. ICT Res. Appl.* **2015**, *8*, 234–250. [CrossRef]
73. Shah, S.A.R.; Issac, B. Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Gener. Comput. Syst.* **2018**, *80*, 157–170. [CrossRef]
74. Fouladi, R.F.; Kayatas, C.E.; Anarim, E. Frequency based DDoS attack detection approach using naive Bayes classification. In Proceedings of the 2016 39th International Conference on Telecommunications and Signal Processing (TSP), Vienna, Austria, 27–29 June 2016; pp. 104–107.
75. Alkasassbeh, M.; Al-Naymat, G.; Hassanat, A.; Almseidin, M. Detecting distributed denial of service attacks using data mining techniques. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 436–445. [CrossRef]
76. Niyaz, Q.; Sun, W.; Javaid, A.Y. A deep learning based DDoS detection system in software-defined networking (SDN). *arXiv* **2016**, arXiv:1611.07400.
77. Yadav, S.; Subramanian, S. Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder. In Proceedings of the 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), New Delhi, India, 11–13 March 2016; pp. 361–366.
78. Nguyen, S.N.; Nguyen, V.Q.; Choi, J.; Kim, K. Design and implementation of intrusion detection system using convolutional neural network for dos detection. In Proceedings of the 2nd International Conference on Machine Learning and Soft Computing, Phu Quoc Island, Vietnam, 2–4 February 2018; pp. 34–38.
79. Bontemps, L.; McDermott, J.; Le-Khac, N.A. Collective anomaly detection based on long short-term memory recurrent neural networks. In Proceedings of the International Conference on Future Data and Security Engineering, Tho City, Vietnam, 23–25 November 2016; Springer: Cham, Switzerland, 2016; pp. 141–152.
80. Bapat, R.; Mandya, A.; Liu, X.; Abraham, B.; Brown, D.E.; Kang, H.; Veeraraghavan, M. Identifying malicious botnet traffic using logistic regression. In Proceedings of the 2018 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 27 April 2018; pp. 266–271.
81. Abdelhamid, N.; Thabtah, F.; Abdel-jaber, H. Phishing detection: A recent intelligent machine learning comparison based on models content and features. In Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, China, 22–24 July 2017; pp. 72–77.
82. Peng, K.; Leung, V.; Zheng, L.; Wang, S.; Huang, C.; Lin, T. Intrusion detection system based on decision tree over big data in fog environment. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 4680867. [CrossRef]
83. He, Z.; Zhang, T.; Lee, R.B. Machine learning based DDoS attack detection from source side in cloud. In Proceedings of the 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA , 26–28 June 2017; pp. 114–120.
84. Doshi, R.; Apthorpe, N.; Feamster, N. Machine learning ddos detection for consumer internet of things devices. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 29–35.
85. Meidan, Y.; Bohadana, M.; Mathov, Y.; Mirsky, Y.; Shabtai, A.; Breitenbacher, D.; Elovici, Y. N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Comput.* **2018**, *17*, 12–22. [CrossRef]
86. Diro, A.; Chilamkurti, N. Leveraging LSTM networks for attack detection in fog-to-things communications. *IEEE Commun. Mag.* **2018**, *56*, 124–130. [CrossRef]
87. Foroutan, S.A.; Salmasi, F.R. Detection of false data injection attacks against state estimation in smart grids based on a mixture Gaussian distribution learning method. *IET Cyber-Phys. Syst. Theory Appl.* **2017**, *2*, 161–171. [CrossRef]

88. He, Y.; Mendis, G.J.; Wei, J. Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism. *IEEE Trans. Smart Grid* **2017**, *8*, 2505–2516. [CrossRef]
89. Jing, X.; Bi, Y.; Deng, H. An Innovative Two-Stage Fuzzy kNN-DST Classifier for Unknown Intrusion Detection. *Int. Arab. J. Inf. Technol. (IAJIT)* **2016**, *13*, 359–366.
90. Farnaaz, N.; Jabbar, M. Random forest modeling for network intrusion detection system. *Procedia Comput. Sci.* **2016**, *89*, 213–217. [CrossRef]
91. Ravale, U.; Marathe, N.; Padiya, P. Feature selection based hybrid anomaly intrusion detection system using K means and RBF kernel function. *Procedia Comput. Sci.* **2015**, *45*, 428–435. [CrossRef]
92. Jabbar, M.; Aluvalu, R.; Reddy, S. Cluster based ensemble classification for intrusion detection system. In Proceedings of the 9th International Conference on Machine Learning and Computing, Singapore, 24–26 February 2017; pp. 253–257.
93. Guo, W.; Mu, D.; Xu, J.; Su, P.; Wang, G.; Xing, X. Lemna: Explaining deep learning based security applications. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15 October 2018; pp. 364–379.

Intrusion detection using deep sparse auto-encoder and self-taught learning

Article *in* Neural Computing and Applications · March 2019

DOI: 10.1007/s00521-019-04152-6

4 authors, including:

Aqsa Saeed
Pakistan Institute of Engineering and Applied Sciences
13 PUBLICATIONS **142** CITATIONS
SEE PROFILE

Asifullah Khan
Pakistan Institute of Engineering and Applied Sciences
214 PUBLICATIONS **3,256** CITATIONS
SEE PROFILE

Hanif Durad
Pakistan Institute of Engineering and Applied Sciences
35 PUBLICATIONS **149** CITATIONS
SEE PROFILE

Qureshi, A.S., Khan, A., Shamim, N. et al. Neural Comput & Applic (2019).
https://doi.org/10.1007/s00521-019-04152-6 (Accepted: 15 March 2019)

Intrusion detection using deep sparse auto-encoder and self-taught learning

Aqsa Saeed Qureshi[1], Asifullah Khan[*1,2], Nauman Shamim[1], and Hanif Durad[1]

[1] PR-Lab, Department of Computer and Information Science, Pakistan Institute of Engineering and the Applied Sciences, Nilore-45650, Islamabad; asif@pieas.edu.pk

[2]Centre for Mathematical Sciences, Pakistan Institute of Engineering and Applied Sciences, Nilore-45650, Islamabad;

Abstract

With the enormous increase in the use of the internet, secure transfer of data across networks has become a challenging task. Attackers are in continuous search of getting information from network traffic and this is the main reason that efficient intrusion detection techniques are required to identify different kinds of network attacks. In past, various supervised and semi-supervised methods have been developed for intrusion detection. Most of these methods require a large amount of data to develop an efficient Intrusion Detection System (IDS). In the proposed Deep Neural Network and adaptive Self-Taught based Transfer Learning (DST-TL) technique, we have exploited the concept of self-taught learning to train Deep Neural Networks for reliable network intrusion detection. In the proposed method, a pre-trained network on regression related task is used to extract features from NSL-KDD dataset. Original features along with extracted features from the pre-trained network are then provided as an input to the sparse auto-encoder. Self-taught learning based extracted features, when concatenated with the original features of NSL-KDD dataset, enhances the performance of the sparse auto-encoder. Performance of self-taught learning based approach is compared against several techniques using ten independent runs in terms of accuracy, false alarm and detection rate, area under the ROC, and PR curve. It is experimentally observed that the auto-encoder trained on the combined original and extracted features is stable and offer good generalization in comparison to the sparse auto-encoder trained on original features alone.

Keywords ---- Self-taught learning, deep sparse auto-encoder, intrusion detection system

1. Introduction

In the last decade, with the rapid growth in information technology, security has become one of the major concerns of almost all the institutes and companies. Intruders are in continuous search to access the network traffic and that is why security systems have to cope with different types of attacks [1]. Secure data transfer over a network has always been a challenging task. In some cases, such as banking transactions, defense-related communication, traffic of government-related networks, the security of network traffic becomes increasingly important. Consequently, it is imperative to have a secure network

monitoring and detection system that can efficiently detect different types of attacks and thus secure a network against malicious attacks. An Intrusion Detection System (IDS) monitors and detects violations regarding security policy and recognise any abnormal patterns in the network traffic [2]. In host-based IDS, only information related to host is monitored [3]. Whereas, in case of network-based IDS, instead of monitoring individual hosts in the network, an overall network flow is monitored and analyzed [4]. However, in application-based IDS, a system monitors any abnormal behavior of protocol between the different devices[5].

An IDS can be considered as an expert system. The system is first trained over a large dataset for selecting features. After training and recognizing patterns as normal or malacious, the working of IDS is assessed on unseen data. The performance of an IDS mostly depends on the selected features. Intrusion detection techniques are broadly categorized into two main groups. The first one is called anomaly-based IDS, in which those data patterns are detected, which show abnormal behavior[6]. The advantage of anomaly-based IDS is that zero-day attacks can be detected; however, it may have a high false positive rate. The second category of IDS is signature or misuse based IDS, in which patterns are known in advance. For example, a labeled intrusion detection dataset has instances, which are labeled as normal or intrusive. By using machine learning algorithms, a trained model can automatically detect intrusive instances [7]. However, zero-day attacks are difficult to be detected by signature-based IDS.

In past, many different intrusion detection techniques have been reported by different researchers. Malhotra et al. [8] used K Nearest Neighbor and Genetic Programming for developing a reliable IDS. To overcome the deficiencies of IDS associated with single-level structure technique, a hierarchical approach based on Neural Networks is recommended by Zhan et al. [9]. On the other hand, Panda et al. [10] used Naïve Bayes classifier for developing an efficient IDS. Zhang et al.[11] used Random Forest based classifier for the purpose of detecting an intrusion within the network. Portnoy et al. [12] proposed a clustering-based technique that uses unlabeled data for detecting an intrusion within the network. Similarly, an IDS trained in an unsupervised way was reported by Leung et al. [13]. Aslahi et al. [14] reported a hybrid approach for intrusion detection system. Another network-based IDS was developed by Ramakrishnan et al. [15] in which features are selected on the basis of entropy. Similarly, a technique that detects intrusion in wireless Ad-Hoc networks was proposed by Srinivasan et al.[16] whereby, Kohen self-organizing map identifies abnormal patterns within the network. In contrast, Puri et al. [17]

proposed a hybrid technique, that combines regression and classification trees along with Support Vector Machine (SVM) for identifying network attacks. In a different work, in order to ensure network security, SVM and Neural Network-based IDS is developed by Mukkamala et al.[18]

Recently, a fuzzy logic based semi-supervised algorithm was proposed for intrusion detection, in which a single layer Neural Network is used to obtain fuzzy membership function [19]. Similarly, Kim et al. [20] used Deep Neural Networks based approach for IDS. Another tree algorithm based approach for network intrusion detection was reported by Subasi et al.[21]. Almost all of these intrusion detection techniques uses machine learning algorithms, that need to be trained from scratch. In contrast, the proposed Deep Neural Network and adaptive Self-Taught based Transfer Learning (DST-TL) technique uses the concept of self-taught learning for developing efficient and reliable IDS. Instead of training the model with original features as in previously reported methods, the proposed DST-TL technique uses a self-taught learning approach for extracting feature through an already trained network (using unlabeled data). The extracted features are then combined with the original features and are used to train a sparse auto-encoder. The organization of the paper is as follows: Section 2 is related to the description of the dataset. Similarly, section 3 presents the proposed technique. Implementation and parameter details are discussed in section 4. Results and discussion are described in section 5. Whereas, section 6 is related to conclusions.

2. Details of Dataset:

In 1998, MIT-Lincoln Lab generated an environment in which raw data from TCP dump was gathered throw Local-Area-Network (LAN) that simulates Air Force LAN in U.S [22]. Each TCP connection within the network is labeled either as attacked or a normal. Moreover, training and test data contain five and two million records, respectively. In 1999, a version of the data collected from Lincoln Labs is used in KDD contest related to intrusion detection. The purpose of gathering the data was to analyze and evaluate the research related to detecting intrusion within the network. Whereas, the performance of proposed DST-TL is evaluated on NSL-KDD dataset (an updated version of KDD dataset). KDD dataset is among the few publically available network-based IDS datasets. There are some inherent deficiencies [23] in KDD dataset; some of which are:

a) Almost 78% and 75% of the samples are repeated in training and test sets, respectively.

b) Redundant samples in training set may make the classifier biased towards the most frequent samples, as a result, good performance may not be achieved on the test data.

To overcome the deficiencies of KDD dataset, NSL-KDD dataset was introduced by Tavallaee et al. [23]. All the redundant samples in KDD dataset were removed and only a single copy was kept in training and test sets. Moreover, in both training and test sets, difficulty level against each record of KDD data was also assigned. For that purpose, the training set was divided into three equal parts. After the division of training data, seven different classifiers were trained on different portions of the data; as a result, 21 different labels were predicted against test data records. From KDD dataset, after the removal of redundant records $KDDTest^{+}$ was generated. In addition to $KDDTest^{+}$, a new test set $KDDTest^{-21}$ was also generated containing only those records, which were not correctly classified by any of the trained classifiers. Whereas, for experimental purposes, only 20% of the $KDDTrain^{+}$ was reserved. Therefore, performance of a classifier needs to be evaluated on $KDDTest^{+}$ and especially, on $KDDTest^{-21}$. Percentages of the positive and negative samples in $KDDTest^{+}$ and $KDDTest^{-21}$ are shown in Table 1. In NSL-KDD dataset, each sample consists of 41 features, which are further categorized into three different groups.

i. **Basic features:**

The TCP/IP connection related attributes are included in basic features. The basic features are originated from the packet header, and out of 41 features in NSL-KDD dataset, ten features are categorized as basic features. For example, protocol type, service flags, etc are considered as basic features.

ii. **Content features:**

Content features are related to all of the suspicious information within the payload of a TCP packet. For example, the number of failed login efforts is considered as a content feature.

iii. **Traffic features:**

Traffic features are calculated with reference to the time interval and are further categorized into two groups:

a) **Same service features:**

Same service features are features that are evaluated across only those connections from past two seconds that contain the same service as current connections hold.

Qureshi, A.S., Khan, A., Shamim, N. et al. Neural Comput & Applic (2019).
https://doi.org/10.1007/s00521-019-04152-6 (Accepted: 15 March 2019)

b) Same host features:

Same host features are features (behavior, services of protocol, etc.) that are evaluated across those connections in past two seconds, which contain the same host as current connections hold.

Table 2. shows details of the 41 features present within NSL-KDD dataset.

Table 1: Percentage of positive and negative samples in $KDDTest^{+}$ and $KDDTest^{-21}$ dataset

Dataset	Total samples	Normal Samples	Abnormal Samples
20% of $KDDTrain^{+}$	25193	13449 (53.4%)	11744 (46.6%)
$KDDTest^{+}$	22544	9711 (44%)	12833 (56%)
$KDDTest^{-21}$	11850	2152 (19.2%)	9698 (81.8%)

Qureshi, A.S., Khan, A., Shamim, N. et al. Neural Comput & Applic (2019).
https://doi.org/10.1007/s00521-019-04152-6 (Accepted: 15 March 2019)

Table 2: Types of features present within NSL-KDD dataset

Feature number	Type		Feature name
1	Basic Features		Duration
2			Protocol_type
3			Service
4			Flag
5			Src_bytes
6			Dst_bytes
7			Land
8			Wrong_fragment
9			Urgent
10			Hot
11	Content features		Num_failed _logins
12			Logged_in
13			Num_compromised
14			Root_shell
15			Su_attempted
16			Num_root
17			Num_file_creations
18			Num_shells
19			Num_access_files
20			Num_outbound_cmds
21			Is_hot_login
22			Is_guest_login
23	Traffic features	Same service features	Count
24			Srv_count
25			Serror_rate
26			Srv_serror_rate
27			Rerror_rate
28			Srv_rerror_rate
29			Same_srv_rate
30			Diff_srv_rate
31			Srv_diff_host_rate
32		Same host features	Dst_host_count
33			Dst_host_srv_count
34			Dst_host_same_srv_rate
35			Dst_host_diff_srv_rate
36			Dst_host_same_src_port_rate
37			Dst_host_srv_diff_host_rate
38			Dst_host_serror_rate
39			Dst_host_srv_serror_rate
40			Dst_host_rerror_rate
41			Dst_host_srv_rerror_rate

All the attacks in the NSL-KDD data belong to the following four attack categories:

- Denial-of-Service (DOS) attacks
- User-to-root (U2R) attacks
- Remote-to-local (R2L) attacks
- Probing (PROB) attacks

Details related to all of the four categories of attacks are shown in Table 3. Another property of the NSL-KDD dataset is that an additional 14 new attacks, also are included in the test data.

Table 3: Different types of attacks in NSL - KDD dataset

PROB	**IP-sweep**		**Port-sweep**		**NMAP**		**Satan**	
R2L	Warezmaster	Wareclient	Phf	SPY	Multi-HOP	Guess-password	FTP-Write	IMAP
U2R	Perl		Root-kit		Load-module		Butter-overflow	
DOS	Smurf	Land	Ping-0f-death		Neptune	Back		Teardrop

3. Deep sparse auto-encoder:

Sparsity is used for reducing the links within the network and thus generally increases the generalization performance of the Machine Learning technique. In past, different researchers proposed the sparse representation based classification system for recognising face [24,25], object categorization [26], classification [27], and regression [28,29] related tasks. Lu et al. [24] proposed locality based Weighted Sparse Representation Classifier (WSRC) for the face recognition related task. Performance comparison of Lu's technique is checked on 15 different datasets from UCI repository. Experimental results depict the WSRC based method outperforms most of the state-of-the-art techniques. Work related to recognizing the partially occluded faces within images is proposed by Mi et al. [25]. In Mi's work after the division of images into blocks, the regression-based technique is used to find out the occluded blocks. In the end, Sparse Representation Classification (SRC) based strategy is used to detect the face. Comparison of Mi's technique against extended SRC (eSCR) shows that SCR technique is good not only in terms of accuracy but also computationally less extensive. Another researcher, Gui et al.[30] presented a survey related to sparsity-based feature selection strategies. In Gui's survey, mathematical representation and motivation behind the sparsity-based feature selection strategies have been discussed. Another work related to Group-Sparse-Multiview-Alignment-Framework (GSM-PAF) method for the purpose of selecting and extracting feature by introducing sparsity within the projection matrix is proposed [27]. GSM-PAF technique shows good performance on real-image-classification tasks. In the proposed DST-TL work, sparse auto-encoder based self-taught learning approach is used for the development of a reliable IDS.

An auto-encoder is a kind of neural network that tries to reproduce input to its output. Two steps are involved in the training of deep sparse auto-encoder. In the first step, unsupervised greedy layer-wise

pre-training is performed, in which unlabeled data is used to extract useful features at each layer. Encoding and decoding are important steps involved in the pre-training of deep sparse auto-encoder. An auto-encoder tries to reproduce its input to its output, during the pre-training phase, and that is why labeled data is not needed during pre-training phase. Let d_{inp} be the input to the encoder, which is encoded in the form of a function as shown in equation 1.

$$en_{inp} = func(d_{inp}) \quad (1)$$

After encoding the input data, next phase is to decode the encoded results (en_{inp}) using equation 2.

$$dec_{inp} = func(\mathrm{en}_{inp}) \quad (2)$$

Purpose of decoding is to reconstruct the original input. During the training phase, if auto-encoder only tries to reproduce its input to its output, then it may not be beneficial because it will lead to overfitting, and thus results on unseen samples may be poor. To overcome the problem of overfitting during the training of individual auto-encoder, a sparsity term is introduced in the loss function whih helps the auto-encoder to learn more generalized features during the pre-training phase.

A simple auto-encoder is comprised of an input layer, a hidden layer (for encoding the input data) and an output layer to reconstruct the original input. The loss function, used during pre-training of sparse auto-encoder, is described below.

$$Loss\ function = \frac{1}{S}\sum_{i=1}^{I}\sum_{j=1}^{J}(d_{inp_{ij}} - dec_{inp_{ij}})^2 + \lambda_L \Omega_{weight} + \beta_L \Omega_{sparsity} \quad (3)$$

$(\frac{1}{S}\sum_{i=1}^{I}\sum_{j=1}^{J}(d_{inp_{ij}} - dec_{inp_{ij}})^2)$ is the mean-squared-error between decoded (dec_{inp}) and actual (d_{inp}) inputs. Whereas Ω_{weight} is weight regularization with a coefficient λ_L, and $\Omega_{sparsity}$ is a sparsity regularization term with a coefficient β_L. L_2 weight regularization is expressed mathematically as:

$$\Omega_{weight} = \frac{1}{2}\sum_{l}^{L}\sum_{i}^{I}\sum_{j}^{J}(\mathrm{W}_{ij}^{\ l})^2 \quad (4)$$

In equation 4, *L*, *I*, and *J* show the number of hidden layers, instances, and variables, respectively, whereas W denotes weights of a link. L_2 weight regularization term in the loss-function helps to control the weights of the network and thus act as regularization term (learns optimal weights during training phase). The sparsity regularization term is expressed mathematically as:

$$\Omega_{sparsity} = \sum_{i'=1}^{s'} KL_{D'}(\frac{A'}{A'_{i'}}) = \sum_{i'=1}^{s'} A' \log(\frac{A'}{A'_{i'}}) + (1-A') \log(\frac{1-A'}{1-A'_{i'}}) \quad (5)$$

$A_i^{'}$ shows the average activation value of a neuron i, whereas $A^{'}$ depicts the desired activation value of i^{th} neuron. The activation value of a particular neuron is controlled by regulating the weights of the network. If the desired and the average activation values of a neuron are the same, then the sparsity regularization term will be zero. The sparsity regularization term increases with an increase in the difference between the values of $A_i^{'}$ and $A^{'}$ values.

To make the proposed network deep, the decoders are removed after the individual training of auto-encoder and the encoded input is provided as input to the next auto-encoder. When the required number of auto-encoders are trained, the next step is to stack the individually trained auto-encoders to form a feedforward network (with good initial weights). This feedforward network is then fine-tuned using back propagation based learning algorithm. In the proposed DST-TL technique, features are extracted by implementing only unsupervised pre-training phase (using unlabeled data). Sparsity within deep sparse auto-encoder helps to extract effective feature space and improves generalization performance of the trained network. Moreover, features extracted from the proposed DST-TL based approach, when combined with the original features of NSL-KDD dataset, generally lead to a diverse feature space.

3.1.Deep neural network and adaptive self-taught based transfer learning (DST-TL) for IDS:

The basic idea behind the proposed technique is the use of an unsupervised feature extraction using adaptive self-taught learning for developing an efficient IDS. Results depict that the proposed DST-TL technique provides good generalization in terms of different performance measures. Deep sparse auto-encoder is used in the proposed DST-TL technique for extracting effective features through unsupervised learning. Figure 1. shows a block diagram of the proposed DST-TL methodology.

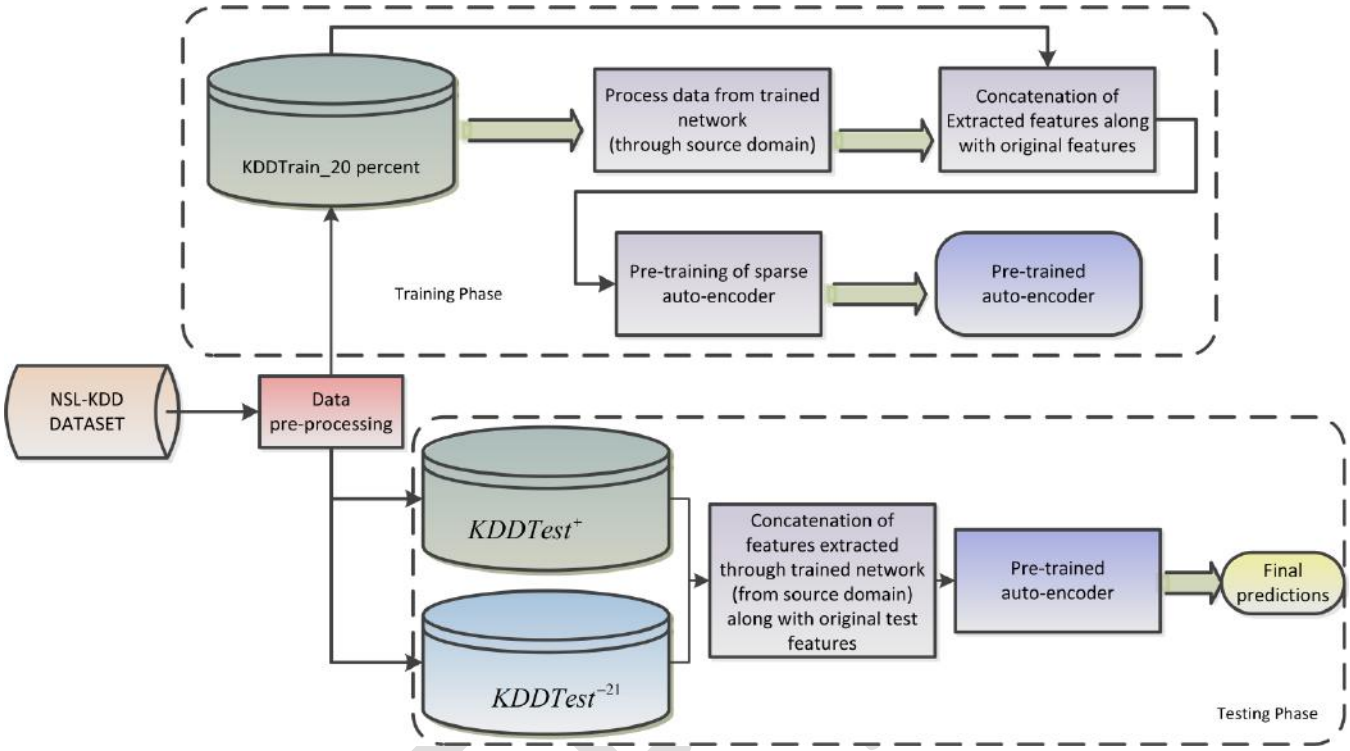


Figure 1: Block diagram of the proposed DST-TL methodology

3.2. Importance of transfer learning in the proposed DST-TL technique:

Leveraging the knowledge gained from one type of machine learning problem and applying it to another machine learning task is known as transfer learning. The domain from where the knowledge is extracted is called the source domain. Extracted information from the source domain is either in the form of features or in the form of learning weights. Domain, in which the knowledge gained from the source domain is applied, is known as the target domain. Transfer learning can help in providing good results, particularly in case of deep neural networks where a lot of calibrated effort is required during training. There are several reasons that urge the machine learning experts to apply transfer learning. Some of the reasons are listed below:

- Sometimes labeled data is not sufficient to train a network in the target domain.
- A lot of effort (for tuning parameters) is required in the target domain to achieve remarkable results.
- Training from scratch takes a lot of time.

In case of transfer learning, it is not always a necessary condition that target and source domains follow the same distribution. There are also some forms of transfer learning approaches in which the target and source tasks are not from the similar domain. Some of the transfer learning approaches are given below:

- Self-taught learning
- Multi-tasking
- Domain adaptation
- Unsupervised transfer learning

In past, various transfer learning based approaches [31–42] have been reported by many different researchers for machine learning-related tasks. In the proposed DST-TL technique, self-taught learning based robust IDS is developed.

3.2.1. Self-taught learning:

Typically, in transfer learning, target and source domain follow the same data distribution, and in comparison to the target domain, a sufficient amount of data is available in the source domain. In 2007, Rania et al.[31] presented the idea of the self-taught learning, in which unlabeled data in the source domain help in improving the performance of classification related task in the target domain. Moreover, target and source domain data may not follow the same distribution. Features thus extracted using unlabeled data from a source domain may assist to improve learning in the target domain task.

3.2.2. Exploiting adaptive self-taught learning using deep sparse auto-encoder:

Proposed DST-TL technique is comprised of two phases. In phase-1, a pre-trained deep sparse auto-encoder (using unlabeled data) on regression related task (wind power prediction in this case) is used as a transferable knowledge from source domain. Only first and last layer (using pre-training in target domain) are added to the already trained hidden layers (from source domain) to form a feed forward neural network. Feature extraction in the phase-1 is performed by just passing the target domain data through trained feed forward network.

In phase-2, original features along with extracted features (from phase-1) are provided as an input to train the sparse auto-encoder. Sparse auto-encoder is trained (using pre-training) such that only those features are extracted from the input data on which softmax classifier shows good performance (on

validation data). Performance of trained network is then evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ test data. Figure 2 shows self-taught learning based feature extraction methodology that is used in the proposed DST-TL technique.

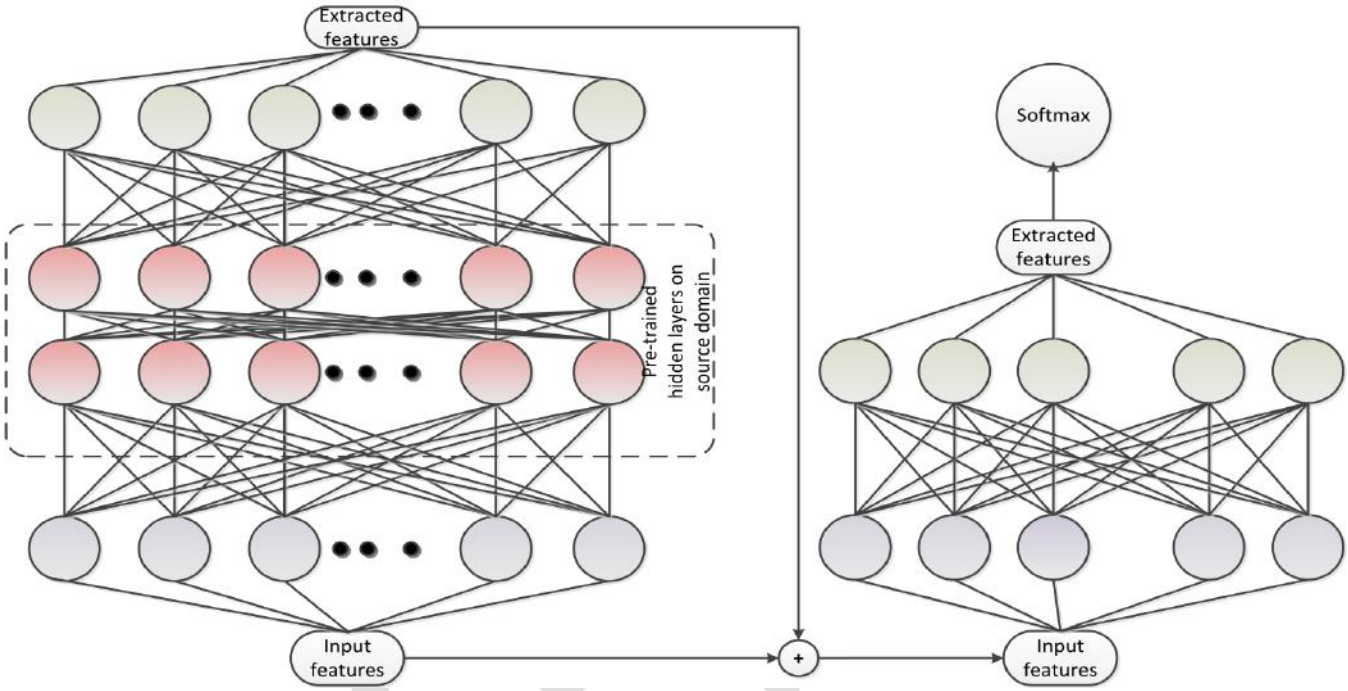


Figure 2: Use of self-taught learning in training of deep sparse auto-encoder

3.2.3. Knowledge transfer from source to target domains:

In self-taught learning, although the source task does not follow the same data distribution as that of the target domain, however both domains are somehow related to each other. In the proposed DST-TL technique, source domain data is collected from wind farms located near Europe, and the task was to perform power forecasting against the provided features. As power forecasting at time t depends on the climatic condition of the previous hours, that is why the predicted power and associated features of last twenty-four hours are considered as input data to predict the power of time t. In short, in the source domain, the dataset is of the time series nature. Moreover, wind power prediction is a challenging task because any sudden fluctuation in the geographical and climatic conditions can affect the generated power. On the other hand, the target domain task in the proposed DST-TL technique is related to intrusion detection, in which NSL-KDD is used as benchmark dataset. Discussion related to the NSL-KDD dataset in section 2 shows that traffic features are also of the time series nature and are calculated

on the basis of connections in the past two seconds. When an intruder attacks the network, it results in the sudden change in the behavior of the Network flow, so any sudden change in the behavior of the network may be the sign of intrusion within the network. Network intrusion detection, just like wind power, is of unpredictable nature. Thus the source domain task is related to target domain task in terms of the time-series nature of input features and unpredictable behavior of power and intrusion.

In the proposed DST-TL technique, because of the commonality between the target and source domain, trained network from source domain is efficient enough to predict the wind power despite of any abrupt change in the atmospheric behavior and thus share the gained knowledge to target domain.

4. Implementation details:

All the experimental work related to the proposed work was performed on a desktop computer, having 16GB RAM, Intel(R) Core(TM) i7-33770, 64-bit operating system, CPU@3.4 GHz. The operating system used was Windows 7 professional and Matlab 2016 a is used as a programming tool.

4.1.Parameters setting of the proposed technique:

In order to set the parameters of deep sparse auto-encoder, 10% of training data was used for validating the optimal parameters. Table 4 shows the parameter setting of the deep sparse auto-encoder, in which hidden layers are pre-trained on source domain task. Whereas, Table 5 illustrates the parameters setting during the training phase of sparse auto-encoder with (using original as well as extracted features from the source domain) or without (using only original features of NSL-KDD data) self-taught learning approach.

Table 4: Parameters setting of deep sparse auto-encoder (for extracting features) using hidden layers trained on source domain task

Parameters	**Layer trained according to the target domain**	**Trained layers on regression related task in the source domain**		**Layer trained according to the target domain**
	Layer 1	Layer 2	Layer 3	Layer 4
Coefficient of L_2 weight Rregularization	0.0002	0.00003	0.00001	0.00001
Coefficient of Sparsity Regularization	4	4	4	4
Maximum number of epochs	200	500	250	150
Number of neurons	124	300	220	200

Qureshi, A.S., Khan, A., Shamim, N. et al. Neural Comput & Applic (2019).
https://doi.org/10.1007/s00521-019-04152-6 (Accepted: 15 March 2019)

Table 5: Parameters setting of the sparse auto-encoder trained on the combined original and extracted features

Parameters	Without self-taught learning approach		With self-taught learning approach	
	Layer 1	Layer 2	Layer 1	Layer 2
Coefficient of L_2 weight regularization	0.0002	0.00002	0.00002	0.00002
Coefficient of Sparsity Regularization	5	4	4	3
Maximum number of epochs	100	200	200	100
Number of neurons	100	80	166	180

4.2.Performance evaluation:

To evaluate the proposed DST-TL technique, detection rate, false alarm rate, area under the ROC curve (AUC-ROC), area under the precision-recall curve (AUC-PR), and accuracy are used as evaluation measures. AUC-ROC is basically the plot of sensitivity and 1-specificity, at different values of the thresholds, however AUC-PR is the plot between Precision and Recall. Mathematically, the measures are defined in equations 6 to 11.

$$Sensitivity = \mathrm{Re}call = \frac{TP}{TP + FN} \quad (6)$$

$$1 - Sepecificity = \frac{FP}{FP + TN} \quad (7)$$

$$\mathrm{Pr}ecision = \frac{TP}{TP + FP} \quad (8)$$

$$Detection\,Rate = \frac{TP}{TP + FP} \quad (9)$$

$$False\,Alarm\,Rate = \frac{FP}{TN + FP} \quad (10)$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} * 100 \quad (11)$$

In the above equations, TP and TN are the number of positive and negative class samples, respectively that are correctly classified by the classifier. Whereas, FP and FN are the number of negative and positive class samples, respectively, which are classified wrongly by the classifier.

5. Results and Discussion:

In the proposed DST-TL technique, after the training of a sparse auto-encoder, the performance of the trained model is evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ data sets. To show the stability of the proposed DST-TL technique, performance in terms of detection and false alarm rates, AUC-ROC and AUC-PR, and accuracy for ten independent runs are shown in Table 8, Table 9, and Table 10, respectively. Before checking the performance of proposed DST-TL technique, the performance of the conventional classifiers (Multi-Layer-Perceptron (MLP), Non-linear Principal Component Analysis (NLPCA), and Deep Belief Network (DBN)) is evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ data sets. MLP is a commonly used classifier that uses back propagation learning algorithm during training. Whereas, DBN is a type of classifier which is formed by stacking of independently trained RBMs (Restricted Boltzmann Machine). After stacking of RBM, the backpropagation learning algorithm is used to fine tune the stacked feed-forward network. Principal-Component-Analysis (PCA) is a commonly used feature reduction technique, however, NLPCA is considered as a generalized form of PCA. In this case, NLPCA comprise of a simple auto-encoder having an encoding layer that contains less neurons in comparison to the number of neurons in input layer. Features that are extracted from the hidden layer are provided as input to a simple classifier for classifying the input data. Table 6. shows the performance of MLP, DBN, and NLPCA in terms of all the evaluation measures. Parameter that are used during the training are provided in Table 7.

Table 6: performance comparison of MLP, NLPCA, and DBN in terms of AUC-ROC, AUC-PR, Detection, and False Alarm Rates

	AUC-ROC		**AUC-PR**		**Detection Rate**		**False Alarm Rate**	
	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$
MLP	0.88±0.02	0.75±0.02	0.92±0.01	0.93±0.01	0.65±0.01	0.28±0.01	0.38±0.02	0.51±0.03
DBN	0.85±0.03	0.74±0.07	0.90±0.02	0.92±0.02	0.65±0.01	0.27±0.01	0.40±0.01	0.53±0.02
NLPCA	0.89±0.03	0.64±0.03	0.92±0.01	0.91±0.01	0.66±0.01	0.25±0.01	0.35±0.01	0.47±0.02

Table 7: Parameter setting of MLP, NLPCA, and DBN during training

MLP	**DBN**							**NLPCA**
Number of neurons in the hidden layer	No of neurons			No of epochs	Batch size	Momentum	Learning-rate	No of neurons in encoding layer
	Layer1	Layer2	Layer3					
25	80	75	50	50	2	0.04	0.001	18

5.1.Performance evaluation using detection and false alarm rate

Table 8 shows the comparison between detection and false alarm rates of a sparse auto-encoder with and without self-taught learning. The standard deviation of error for the ten independent runs is also shown. Table 8 illustrates that deep, sparse auto-encoder trained with self-taught learning approach shows better and more stable results (evident from the low values of standard deviation of error) in comparison to the deep sparse auto-encoder trained without self-taught learning.

Table 8: Performance comparison of the trained sparse auto-encoders for ten independent runs

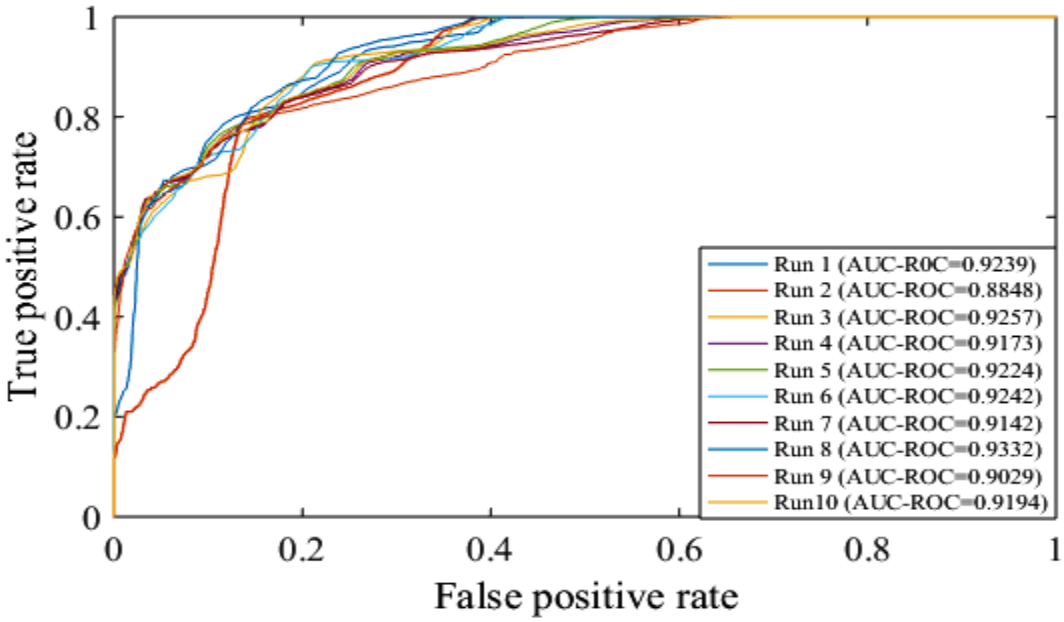
	Spars auto-encoder without TL		**Sparse auto-encoder with TL**	
	Detection rate	False alarm rate	Detection rate	False alarm rate
$KDDTest^{+}$	0.86±0.09	0.13±0.09	0.85 ±0.05	0.14±0.05
$KDDTest^{-21}$	0.81±0.13	0.18±0.13	0.80±0.06	0.19±0.06

5.2. Performance evaluation using AUC-ROC, AUC-PR, and accuracy

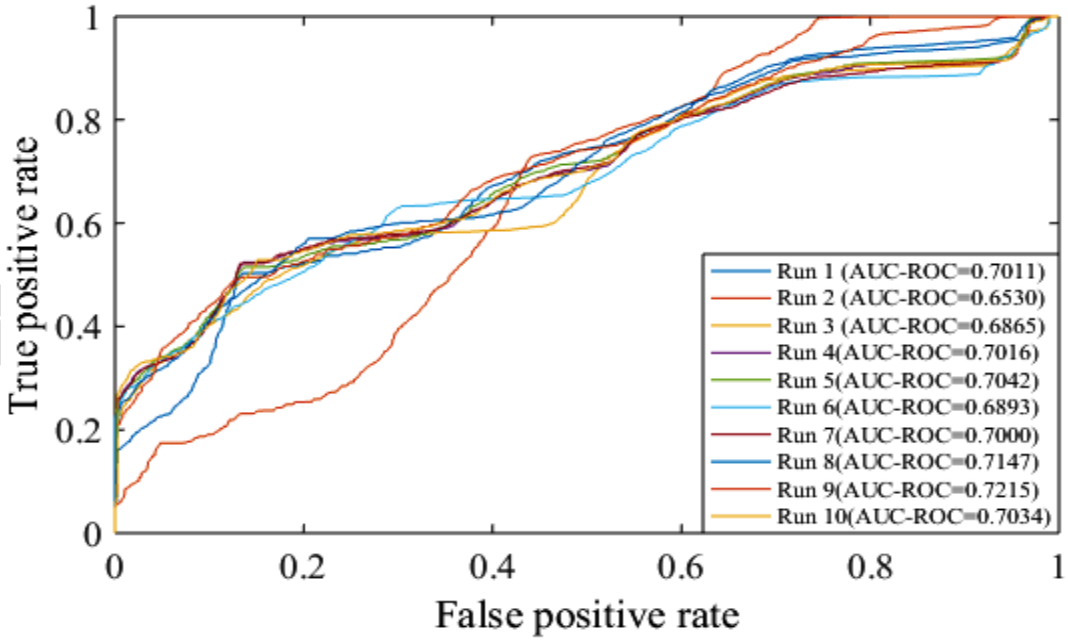
Table 9 shows the performance of the proposed DST-TL on $KDDTest^{+}$ and $KDDTest^{-21}$ data sets in terms of AUC-ROC and AUC-PR. It can be observed that the self-taught learning approach increases stability and performance of network on unseen data and also has a low standard deviation of error. Whereas, Figure 3 and Figure 4 graphically illustrate the AUC-ROC and AUC-PR on $KDDTest^{+}$ and $KDDTest^{-21}$ for ten independent runs.

Table 9: Performance comparison of trained sparse auto-encoders on test data

	Sparse deep auto-encoder without TL		**Deep sparse auto-encoder with TL**	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
$KDDTest^{+}$	0.89±0.01	0.91±0.01	0.91± 0.01	0.93±0.01
$KDDTest^{-21}$	0.61±0.03	0.88±0.01	0.69±0.01	0.91±0.01

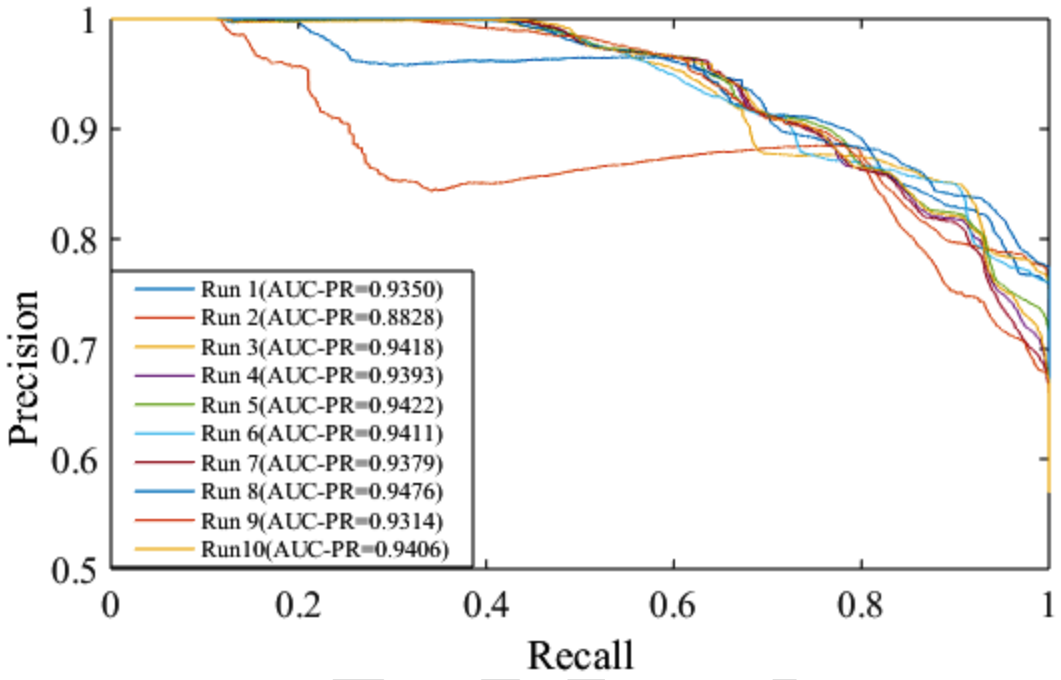


(a)

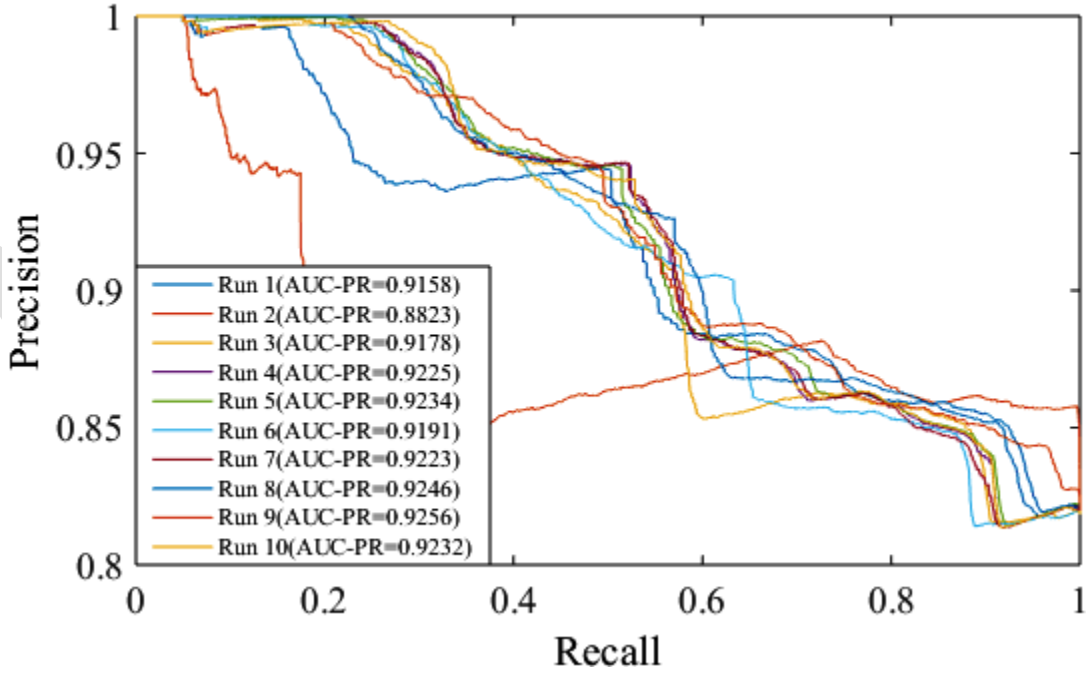


(b)

Figure 3: AUC-ROC of ten independent runs, (a): on $KDDTest^{+}$ (b): on $KDDTest^{-21}$



(a)



(b)

Figure 4:AUC-PR of ten independent runs, (a): on $KDDTest^{+}$,(b): on $KDDTest^{-21}$

Table 10 shows the accuracy of the proposed technique for ten independent runs. It is observed that features extracted through self-taught learning approach lead to the better training of the proposed DST-TL method. The self-taught learning approach helps in extracting robust features, which are then utilized by the deep sparse auto-encoder to learn optimal weights during training and thus increases the generalization performance.

Table 10: Performance comparison for ten independent runs of the trained sparse auto-encoders in terms of accuracy

	Sparse auto-encoder without TL (Accuracy)		**sparse auto-encoder trained using TL (Accuracy)**	
	KDDTest^{+}	*KDDTest*$^{-21}$	*KDDTest*$^{+}$	*KDDTest*$^{-21}$
1	**85.30**	**74.50**	**84.60**	**79.90**
2	84.50	72.70	81.70	71.40
3	80.60	63.30	82.40	69.40
4	74.30	78.30	82.40	72.40
5	80.60	63.40	82.60	71.50
6	75.80	75.50	83.80	74.90
7	72.50	80.70	82.20	72.10
8	76.50	73.90	83.80	72.30
9	77.40	57.40	81.80	69.40
10	73.30	79.20	82.40	77.70
	78.08±4.27	**71.89±7.43**	**82.77±0.91**	**73.1±3.09**

5.3. Performance comparison of the proposed DST-TL technique with state-of-the-art techniques

Table 11 shows a comparison of the proposed DST-TL method with different classifiers (J48, Naïve Bayes, NB tree, Random forest, Random tree, MLP, NLPCA, DBN, and SVM) and also with fuzzy based semi-supervised technique. Table 11 depicts that the performance of the proposed DST-TL technique is better in comparison to existing methods, especially on *KDDTest*$^{-21}$ dataset.

Table 11: Performance comparison of the proposed technique with the existing methods

Classifier	Accuracy	
	$KDDTest^{+}$	$KDDTest^{-21}$
J48	81.05	63.97
Naive Bayes	76.56	55.77
NB Tree	82.02	66.16
Random forest	80.67	63.25
Random tree	81.59	58.51
MLP	77.41	57.34
NLPCA	76.73	56.58
DBN	76	54.55
SVM	69.52	42.29
Fuzziness based IDS (Experiment2)	84.12	68.82
DST-TL(best accuracy in ten independent runs)	**84.60**	**79.90**
DST-TL(average performance)	**82.77±0.91**	**73.1±3.09**

6. Conclusion:

A novel network IDS based on deep sparse auto-encoder that exploits self-taught learning is proposed. In the first phase, feature extraction is performed by passing the original feature set of NSL-KDD through the pre-trained network. Then, a combination of original and extracted features are used to train the sparse auto-encoder. The combined features improve the effectiveness of the feature space and thus increase the performance of the sparse auto-encoder on test samples. It is experimentally shown that the proposed DST-TL technique yields improved performance, although adaptation of the network trained on regression related task is used to improve the performance of the IDS. The experimental comparison shows that the sparse auto-encoder trained on the improved feature space extracted through self-taught learning is more robust and stable in comparison to the sparse auto-encoder trained on the original feature space. Performance on test data shows that in comparison to previous techniques, the proposed DST-TL approach is robust and provides improved prediction accuracy. In future, we intend to apply deep neural networks for classifying different types of attacks.

ACKNOWLEDGMENT

This work is supported by the Higher Education Commission of Pakistan under the Indigenous Ph.D. Scholarship (PIN#213-54573-2EG2-097) Program. We also acknowledge Pakistan Institute of

Engineering and Applied Sciences (PIEAS) for healthy research environment which led to the research work presented in this article.

Conflict of Interest:

The authors declare that they have no conflict of interest.

References:

[1] Ashoor AS, Gore S. Importance of intrusion detection system (ids). Int J Sci Eng Res 2011;2:1–4.

[2] Scarfone K, Mell P. Guide to intrusion detection and prevention systems (idps). NIST Spec Publ 2007;800:94.

[3] Letou K, Devi D. Host-based Intrusion Detection and Prevention System (HIDPS). Int J Comput Appl 2015;69:975 – 88. doi:10.5120/12136-8419.

[4] Mukherjee B, Heberlein LT. Network Intrusion Detection. IEEE Netw 83 1994:26–41.

[5] Masri W, Podgurski A. Application-based anomaly intrusion detection with dynamic information flow analysis 5. Comput Secur 2008;27:176–87. doi:10.1016/j.cose.2008.06.002.

[6] Dı J. Anomaly-based network intrusion detection : Techniques , systems and challenges. Comput S Secur 2009;28:18–28. doi:10.1016/j.cose.2008.08.003.

[7] Diaz-Gomez PA, Hougen DF. Misuse Detection-An Iterative Process vs. A Genetic Algorithm Approach. ICEIS (2), 2007, p. 455–8.

[8] Malhotra S, Bali V, Paliwal KK. Genetic programming and K-nearest neighbour classifier based intrusion detection model. Cloud Comput. Data Sci. Eng. 2017 7th Int. Conf., IEEE; 2017, p. 42–6.

[9] Zhang C, Jiang J, Kamel M. Intrusion detection using hierarchical neural networks. Pattern Recognit Lett 2005;26:779–91. doi:10.1016/j.patrec.2004.09.045.

[10] Panda M, Patra MR. Network intrusion detection using naive bayes. Int J Comput Sci Netw Secur 2007;7:258–63.

[11] Zhang J, Zulkernine M. Network Intrusion Detection using Random Forests. PST, Citeseer; 2005.

[12] Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. Proc. ACM CSS Work. Data Min. Appl. to Secur. (DMSA-2001, Citeseer; 2001.

[13] Leung K, Leckie C. Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters. Proc. Twenty-eighth Australas. Conf. Comput. Sci., vol. 38, 2005, p. 333–42.

[14] Chizari BMARRM, Eslami AMM. A hybrid method consisting of GA and SVM for intrusion detection system. Neural Comput Appl 2016;27:1669–76. doi:10.1007/s00521-015-1964-2.

[15] Devaraju SRS. Attack ' s Feature Selection-Based Network Intrusion Detection System Using Fuzzy Control Language. Int J Fuzzy Syst 2017;19:316–28. doi:10.1007/s40815-016-0160-6.

[16] Srinivasan T, Vijaykumar V, Chandrasekar R. A Self-organized Agent-based architecture for Power-aware Intrusion Detection in wireless ad-hoc networks. 2006 Int. Conf. Comput. Informatics, 2006, p. 1–6. doi:10.1109/ICOCI.2006.5276609.

[17] Puri A, Sharma N. A novel technique for intrusion detection system for network security using hybrid svm-cart 2017.

[18] Mukkamala S, Janoski G, Sung A. Intrusion Detection Using Neural Networks and Support Vector Machines. Neural Networks, 2002. IJCNN'02. Proc. 2002 Int. Jt. Conf., 2002, p. 1702–7.

[19] Aamir R, Ashfaq R, Wang X, Zhexue J, Abbas H, He Y. Fuzziness based semi-supervised learning approach for intrusion detection system. Inf Sci (Ny) 2017;378:484–97. doi:10.1016/j.ins.2016.04.019.

[20] Kim J, Shin N, Jo SY, Kim SH. Method of intrusion detection using deep neural network. Big Data Smart Comput. (BigComp), 2017 IEEE Int. Conf., IEEE; 2017, p. 313–6.

[21] Kevric J, Jukic S, Subasi A. An effective combining classifier approach using tree algorithms for network intrusion detection. Neural Comput Appl 2016:1051–8. doi:10.1007/s00521-016-2418-1.

[22] Cup KDD. Data: Available on http://kdd. ics. uci. edu/database/kddcup99/Database/kddcu p99/kddcup99. html 2007.

[23] Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A Detailed Analysis of the KDD CUP 99 Data Set. Comput. Intell. Secur. Def. Appl., 2009, p. 1–6.

[24] Lu C, Min H, Gui J, Zhu L, Lei Y. Face recognition via Weighted Sparse Representation. J Vis Commun Image Represent 2013;24:111–6. doi:10.1016/j.jvcir.2012.05.003.

[25] Mi J, Lei D, Gui J. Optik A novel method for recognizing face with partial occlusion via sparse representation. Opt - Int J Light Electron Opt 2013;124:6786–9. doi:10.1016/j.ijleo.2013.05.099.

[26] Gui J, Liu T, Tao D, Sun Z, Tan T. Representative Vector Machines : A Unified Framework for Classical Classifiers. IEEE Trans Cybern 2016;46:1877–88.

[27] Gui J, Tao D, Sun Z, Luo Y, You X, Tang YY. Group Sparse Multiview Patch Alignment Framework With View Consistency for Image Classification. IEEE Trans IMAGE Process 2014;23:3126–37.

[28] Gui J, Sun Z, Hou G, Tan T. An optimal set of code words and correntropy for rotated least squares regression. Biometrics (IJCB), 2014 IEEE Int. Jt. Conf., IEEE; 2014, p. 1–6.

[29] Qureshi AS, Khan A, Zameer A, Usman A. Wind power prediction using deep neural network based meta regression and transfer learning. Appl Soft Comput J 2017;58:742–55. doi:10.1016/j.asoc.2017.05.031.

[30] Gui J, Sun Z, Ji S, Member S, Tao D, Tan T. Feature Selection Based on Structured Sparsity : A Comprehensive Study. 1490 IEEE Trans NEURAL NETWORKS Learn Syst 2017;28:1490–507.

[31] Raina R, Battle A, Lee H, Packer B, Ng AY. Self-taught learning: transfer learning from unlabeled data. Proc. 24th Int. Conf. Mach. Learn., ACM; 2007, p. 759–66.

[32] Maurer A, Pontil M, Romera-Paredes B. Sparse coding for multitask and transfer learning. Int. Conf. Mach. Learn., 2013, p. 343–51.

[33] Lim JJ, Salakhutdinov RR, Torralba A. Transfer learning by borrowing examples for multiclass object detection. Adv. Neural Inf. Process. Syst., 2011, p. 118–26.

[34] Hu Q, Zhang R, Zhou Y. Transfer learning for short-term wind speed prediction with deep neural networks. Renew Energy 2016;85:83–95. doi:10.1016/j.renene.2015.06.034.

[35] Du B, Zhang L, Tao D, Zhang D. Neurocomputing Unsupervised transfer learning for target detection from hyperspectral images. Neurocomputing J 2013;120:72–82. doi:10.1016/j.neucom.2012.08.056.

[36] Cao X. A Practical Transfer Learning Algorithm for Face Verification. IEEE Int. Conf. Comput. Vision(ICCV), 2013, p. 3208–15. doi:10.1109/ICCV.2013.398.

[37] Yang S, Lin M, Hou C. A general framework for transfer sparse subspace learning. Neural Comput Appl 2012:1801–17. doi:10.1007/s00521-012-1084-1.

[38] La L, Guo Q, Cao Q, Wang Y. Transfer learning with reasonable boosting strategy. Neural Comput Appl 2014:807–16. doi:10.1007/s00521-012-1297-3.

[39] Yang S, Hou C, Zhang C. Robust non-negative matrix factorization via joint sparse and graph regularization for transfer learning. Neural Comput Appl 2013:541–59. doi:10.1007/s00521-013-1371-5.

[40] Seera M, Peng C. Transfer learning using the online Fuzzy Min – Max neural network. Neural Comput Appl 2014:469–80. doi:10.1007/s00521-013-1517-5.

[41] Silva M, Cardoso JS. Multi-source deep transfer learning for cross-sensor biometrics. Neural Comput Appl 2017:2461–75. doi:10.1007/s00521-016-2325-5.

[42] Khan A, Sohail A, Ali A. A New Channel Boosted Convolution Neural Network using Transfer Learning. arXiv Prepr arXiv180408528 2018.