Building trustworthy AI-Assisted Network Intrusion Detection System

MSc Dissertation

Ka Leung, CHEUNG
Computer Science Department, Swansea University
29, Sep 2021 2021622

***Abstract*-- Artificial intelligence (AI) has gained remarkable attention from human experts in deciding high-stake scenarios. A key to the success of AI-assisted decision making, where human experts can draw on their domain knowledge complementary to the AI models to ensure task success, is to analyze the factors that potentially impact human trust in the AI models, improving the accuracy of decision making in case that the model performs poorly.**

The objective of the project is to evaluate a variety of AI models on potential network attack pattern detection and how to incorporate these factors to develop a more trustworthy AI-assisted decision-making system.

***Index Terms*—Artificial Neural Network, Big Data, Decision Making, trustworthy AI System, Deep Learning, Network Traffic, Data Analysis, Intrusion Detection System, Malicious Activity.**

I. Introduction

Internet-connected services change rapidly to meet the extremely high demands of network usage in the current digital era. By 2020, it is estimated that over fifty billion devices will be connected. (HSu10) Over the past 10 years, a variety of information and communication technology systems were invented and widely used such as IPv6, the internet of things (IoT), and 5G. However, despite the advanced technology systems have greatly enhanced our daily life, it also leads to increase exposure to cyber risks, causing significant substantial financial losses, economic and social consequences such as personal and business data leakage, and network communication failure. With significant improvements in technology, malware attacks have been developed in an even more complicated and dangerous context, hence, intrusion recognition has become a very difficult task and the loss of due to breaches of IT services or other consequences are expected to grow in the coming years. (SGo17)

The demand for building a critical system of intrusion detection system is high. To prevent the malicious threat and ensure improved Internet-connected security, worldwide IT enterprises have invested and developed more intelligent Intrusion Detection Systems (IDS) using the state-of-the-art Machine learning approach. Artificial intelligence (Artificial intelligence, 2021) has been in existence for over 50 years and have been rapidly rising in recent years and gained remarkable attention from human experts in term of decision making in high-stakes scenarios. As the improvement of supercomputer science and big data technology have greatly improved, the manufacturing industry today is experiencing a never seen increase in available data (J. F. & S., 2010, July) Machine learning methods are capable to learn and extract important and useful features from raw data efficiently and automatically, applying deep learning techniques to intrusion detection systems may be an efficient solution.

However, an incapable intrusion detection system not only puts user-sensitive information at risk but also breaks the trust in an AI-assisted system. Hence, this project aims to determine whether an AI model is trustable when applying deep learning techniques to intrusion detection systems to increase the confidence of the intrusion detection system. In this paper, we will assess the performance of the system based on the accuracy, speed, and dependability of detecting malicious activities. Exploring these three factors to see how these factors are reflected in the network security scenario.

To advance research on the use of AI for decision-making in the network traffic analysis of cybersecurity, this project will develop an AI-assisted decision system, Network Intrusion Detection System (NIDS) (Intrusion detection system, 2021). To find the reliability and accuracy of using AI systems by trying different kinds of data training models, such as K-Nearest Neighbors algorithm (KNN), Random Forests, Naive Bayes classifier, Multilayer Perceptron, Deep Neural Network (DNN), and long short-term memory algorithm (LSTM) to analysis the network malicious activities in details.

Utilizing Artificial intelligence helps to detect, classify and visualize the amoral network traffic data, hence find out how this Network Intrusion Detection System can help monitor a network for malicious activity. To see how this system can be trusted and be affected human decision marking. The paper offers related study research and AI-assisted decision systems implementation.

As a result, seeing this critical challenge for contemporary society, this project is inspired to analysis of network traffic data and state-of-the-art Intrusion Detection Systems (IDS), ensuring cybersecurity in information technology (IT) infrastructures.

Contents

Ka Leung, CHEUNG
Computer Science Department, Swansea University

Ka Leung, CHEUNG
Computer Science Department, Swansea University

II. Background research

Hassan et al proposed using a DNN classifier to capture the feature of the packet. They mentioned using two DNN classifiers. The first classifier is limited to only determining if the traffic is normal or suspicious. Upon detection of suspicious activity, the second DNN will further classify the network traffic based on their malware dataset. The second DNN is a multiclass classifier that tries to assign the name of known malware, from the dataset it was trained on, to the input data, p 53 (AL-Maksousy, 2018). This shows a successful case of applying DNN to data analysis.

To detect and detect and classify malware, other researchers used different kinds of deep learning models to increase its accuracy and reliability.

Tao et al. proposed that they tried to use a Fisher and Deep Auto-Encoder to capture the significant features. (Xiaoling Tao, Deyan Kong, Yi Wei, & Yong Wang, 2016). Kim et al. proposed that using the Long-Short-Term-Memory algorithm (LSTM) with the Gradient Descent Optimization and achieve an accuracy of 97.54% in detecting malware IP packets and a recall of 98.95%. (T. Le, J. Kim, & H. Kim, 2017).

Furthermore, Georgy and Richard proposed a model that uses the Random Forest classifier with the Information Gain algorithm. Since the dataset in their work is not large enough. In their work achieved an accuracy of 97% and 0.03% false positives. Although the accuracy rate is high, the result is not reliable since the small dataset. (Georgy Evgen'evich SHILOV & Richard A SILVERMAN., 1971). Gonzalo et al. published showed the performance of DeepMAL can achieve an accuracy of 99.9% in Rbot botnet, while Neris and Virut achieve 63.5% and 54.7% respectively. (Gonzalo Mar, Pedro Casas, & German Capdehourat, 2020)

Experimentally, we explored and compared these types of classic neural networks.

A research article from Discrete Dynamics in Nature and Society proposed using combining Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) to detect and classify network security data (Lan Liu, Jun Lin, Pengcheng Wang, Langzhou Liu, & Rongfu Zhou, 2020,). This approach is different from traditional methods. It is because normally CNN is used in the analysis of the features from a set of data, especially the analysis of the pattern of the image. CNN is excellent in image recognition, detect image objects, and is widely used in automatic image classification systems (Jong-Hwan Kim, Weimin Yang, Jun Jo, Peter Sincak, & Hyun Myung, 2015). To apply CNN classifier in analyzing the features of network traffic is unprecedented.

They input the serialized pre-processed data into LSTM and CNN neural networks. While the LSTM predicts the temporal characteristics of the traffic sequence, and the CNN learns the spatial characteristics of the network traffic sequence. This three-layer neural network model structure has shown in Figure 1. Each layer of the neural network using the dropout discard part features, to prevent overfitting, on the fourth floor, LSTM is combined with CNN through the flatten layer for dimension reduction, and the output was sorted through the SoftMax layer.

According to this research article, after the two kinds of neural networks were combined, the overall accuracy of the CNN-LSTM neural network model reached 96.6%. the accuracy of detection rate about the DDOS can achieve a 100% successful rate. Table 2 shows this method's performance in anomaly network traffic detection.

Types	Precision	Recall	F1
BENIGN	1.00	0.98	0.99
DoS	0.99	1.00	0.99
Port Scan	0.95	1.00	0.99
DDoS	1.00	1.00	1.00
Other Attacks	0.89	0.94	0.92
Total	0.966	0.984	0.976

Table 1: Classification report about multiple network malicious activity. Source from (Lan Liu, Jun Lin, Pengcheng Wang, Langzhou Liu, & Rongfu Zhou, 2020,)

In conclusion, this LSTM-CNN model has an excellent result in detecting anomaly network traffic activities. However, this research article does not mention the false positives rate. Therefore, objectively it still needs to study another kind of data training model for comparing. Then demonstrate the effectiveness and accuracy of these methods.

Moreover, In the PCAP heading structure, there has an attribute called 'label'. When the packet is labeled as a botnet, it means that that packet is a kind of malware-compromised computer under the control of a hacker. It is important to identify the botnet Internet Protocol (IP) packet because 'Botnet comprises 80% of the attacks on the internet in the modern world' p 1 (Vishwakarma , 2020).

Many researchers are aiming to achieve a high accuracy rate of the deep learning model. However, the model that had a high accuracy also resulted in the highest number of false positives. (Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, & Mansoor Alam, 2015).

After finishing a comprehensive related work search, apparently, the CNN-LSTM classifier provides higher accuracy, but we need to assess other performance, such as the speed and dependability, to determine a trustable intrusion detection system. To ensure the training models system has high trustworthy and reliable.

III. Dataset of network intrusion detection

The Dataset of this network intrusion detection system is from the Canadian Institute for Cybersecurity. This project used the latest version of the dataset, which is KDD99 (the KDD Cup 1999 Data). The dataset mainly used for solving the inherent problem included cybersecurity purposes. (M. Tavallaee E. B., 2009)

The dataset type of KDD99 is KDDTrain+.TXT. It has all the NSL-KDD train sets including attack-type labels and difficulty level in CSV format.

Since a huge number of redundant records is one of the most important factors that cause the deep learning classifier to be biased to the high frequent records. Therefore, it is important to choose a no redundant or duplicate record of the dataset.

The following tables show the redundant records of the KDD99 train and test sets. They show the KDD99 has fewer redundant or duplicate records, which is a perfect dataset for deep learning training.

Table 2: Statistics of redundant records in the KDD'99 train set

	Original records	Distinct records	Reduction rate
Attacks	3,925,650	262,178	93.32%
Normal	972,781	812,814	16.44%
Total	4,898,431	1,074,992	78.05%

Table 3: Statistics of redundant records in the KDD'99 test set

	Original records	Distinct records	Reduction rate
Attacks	250,436	29,378	88.26%
Normal	60,591	47,911	20.92%
Total	311,027	77,289	75.15%

A. Advantages of using KDD'99 train set

The KDD99 training dataset has the following advantages over another dataset.

1) No redundant records

The classifier of deep learning models will have less biased to high frequent records. It is because the dataset did not include the redundant records.

2) No duplicate records

The performance of the deep learning models will have a less biased and better detection rate to the frequent records. It is because the dataset did not include duplicate records.

3) Dataset Labeled difficulty level

Each record of the KDD data set labeled the related difficulty level group, which is inversely proportional to the percentage of the record of the dataset. After applied distinct deep learning algorithms, comparing the training result with the difficult level group. It makes the classification rate more efficient and has a higher accurate evaluation.

Consequently, the evaluation training result can be consistent and comparable.

B. Details of the KDD99 dataset

The KDD99 dataset of the Canadian Institute of Cybersecurity includes 125974 network records. It widely covered several protocol types of network packets such as TCP and UDP. (M. Tavallaee E. B., 2012)

1) Categories of the KDD99 dataset.

There are 22 types of attacks in the KDD99, the attack label included normal, neptune, satan, ipsweep, portsweep, smurf, nmap, back, teardrop, warezclient, pod, guess_passwd, buffer_overflow, warezmaster, land, imap, rootkit, loadmodule, ftp_write, multihop, phf, perl, spy.

These 22 types of attack will be categorized into four different types of attack classes, which are DoS (Denial of Service), R2L (Root to Local), U2R (User to Root), Probe (Probing). (Cosimo Ieracitano, 2018)

a) DoS (Denial of Service)

It is a cyber-attack to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting the services of a host connected to the Internet. It includes attacks such as the back, and, Neptune, pod, smurf, teardrop, etc. (Denial-of-service attack, 2021)

b) R2L (Root to Local)

Root to local provides illegal local access to a computer system by sending remote malicious packets to the target system. It includes attacks such as ftp_write, guess_paswed, imap, multihop, phf, spy, warezclient, warezmaster, etc.

c) U2R (User to Root)

U2R allows the hacker to use the system as a normal user or root permission after hacking a vulnerability system. It includes attacks such as buffer_overflow, loadmodule, perl, rootkit, etc.

d) Probe (Probing)

Probe able attacker to gather information about the target network, such as port or system service scanning, to understand how a target responds to intrusions. It includes attacks such as ipsweep, nmap, portsweep, satan, etc.

Table 4: Distribution of Attack Labels before grouping

Attack Label	Count
normal	67343
neptune	41214
satan	3633
ipsweep	3599
portsweep	2931
smurf	2646
nmap	1493
back	956
teardrop	892
warezclient	890
pod	201
guess_passwd	53
buffer_overflow	30
warezmaster	20
land	18
imap	11
rootkit	10
loadmodule	9
ftp_write	8
multihop	7
phf	4
perl	3
spy	2

2) Features of the KDD99 dataset.

The features of the KDD99 dataset have 39 continuous and 3 symbolics.

Table 5: Feature details of KDD99 dataset.

No.	Features	Types
1	protocol type	symbolic
2	service	symbolic
3	fla	symbolic
4	duration	continuous
5	source bytes	continuous
6	destination bytes	continuous
7	land	continuous
8	wrong fragment	continuous
9	urgent	continuous
10	hot	continuous
11	num failed logins	continuous
12	logged in	continuous
13	num compromised	continuous
14	root shell	continuous
15	su attempted	continuous
16	num root	continuous
17	num file creations	continuous
18	num shells	continuous
19	num access files	continuous
20	num outbound cmds	continuous
21	is host login	continuous
22	serror rate	continuous
23	srv serror rate	continuous
24	rerror rate	continuous
25	srv rerror rate	continuous
26	same srv rate	continuous
27	diff srv rate	continuous
28	srv diff host rate	continuous
29	dst host count	continuous
30	dst host srv count	continuous
31	dst host same srv rate	continuous
32	dst host diff srv rate	continuous
33	dst host same src port rate	continuous
34	dst host srv diff host rate	continuous
35	dst host serror rate	continuous
36	dst host srv serror rate	continuous
37	dst host rerror rate	continuous
38	dst-host srv rerror rate	continuous
39	is guest login	continuous
40	count	continuous
41	srv count	continuous

IV. Methodology

The methodology of the network intrusion detection system in this project mainly applied 5 different machine learning classifiers with 2 kinds of networks.

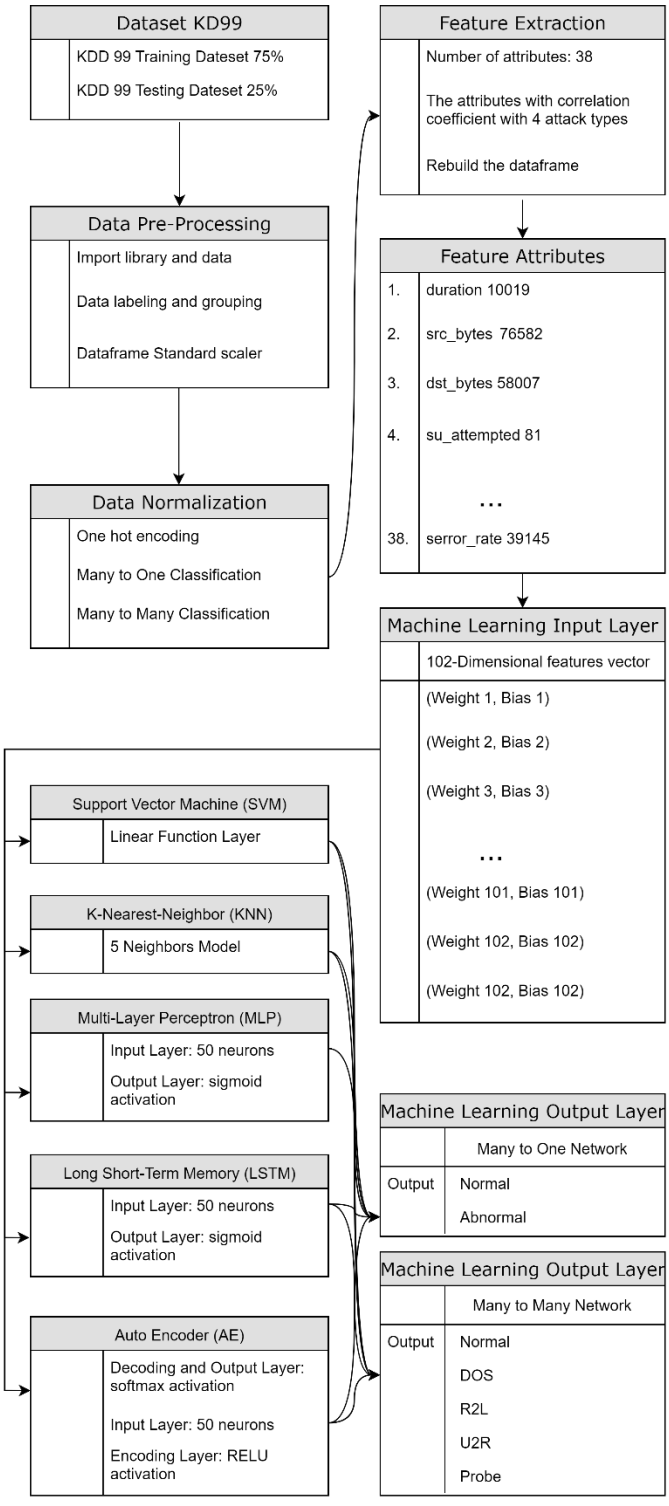


Figure 1: Proposed AI-assisted IDS methodology framework

A. Machine Learning Models used

1) Support Vector Machine (LSVM)

The Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection.

2) K-Nearest-Neighbour (KNN)

The K-nearest Neighbor (KNN) is a supervised machine learning algorithm that can be used to solve both classification and regression problems.

3) Multi-Layer Perceptron (MLP)

MLP is a supervised machine learning technique called backpropagation for training which consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. (Ros61)

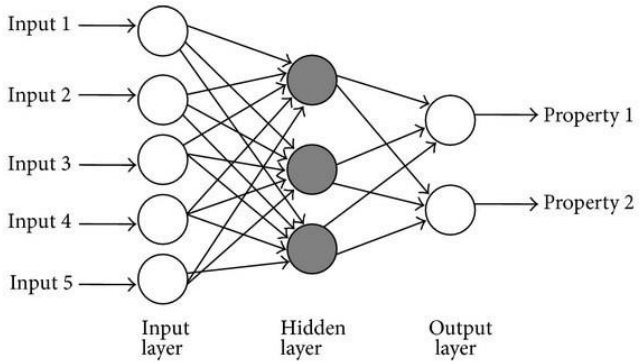


Figure 2 A hypothetical example of Multilayer Perceptron Network. Image source from (Mathur, 2016)

4) Long Short-Term Memory (LSTM)

LSTM is an unsupervised model which is suitable to use in classifying, processing, and making predictions based on time series data. It is an ideal choice to model sequential data and is hence used to learn complex dynamics of human activity.

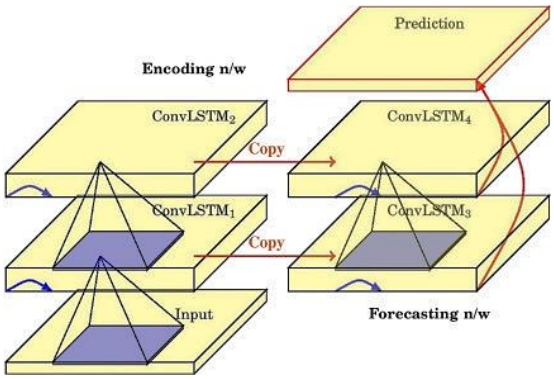


Figure 3 Structure of a convolutional long short-term memory. Image source from (SRo20)

5) Auto Encoder (AE)

Autoencoder is unsupervised learning which used to learn efficient coding of the unlabelled data model. To be more precise, they are self-supervised because they generate their labels from the training data. ((Wikipedia AE, n.d.))

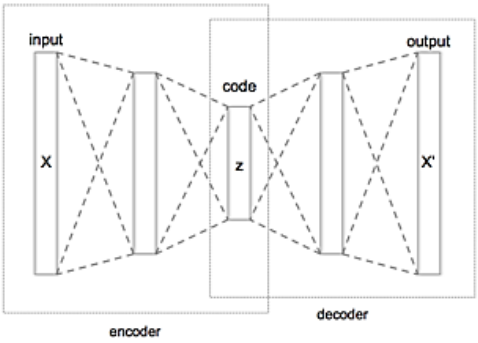


Figure 4 Schematic structure of an autoencoder with 3 fully connected hidden layers where the code i.e. z, or his the most internal layer. Image Source (Wik19, a,r,).

Autoencoders will be trained with a single layer encoder and a single layer decoder but using many-layered (deep19) encoders and decoders which exponentially decrease the amount of training data needed to learn some functions. (Goo16)

B. Machine Learning Network used

The sequences of the machine learning network can influence the input layer and output layer. Below fig.5 showed the sequences of the machine learning network. Each rectangle is a vector, and the arrows represent a function, such as a matrix multiply, etc.

1) Many to One network

More than one input neuron and the output layer have to predict a single output. Only one desired output can be triggered.

2) Many to Many networks

More than one input neuron and more than one desired output can be triggered. Since more than one output can be triggered, the vectors of the hidden layer may hold more neural network states.

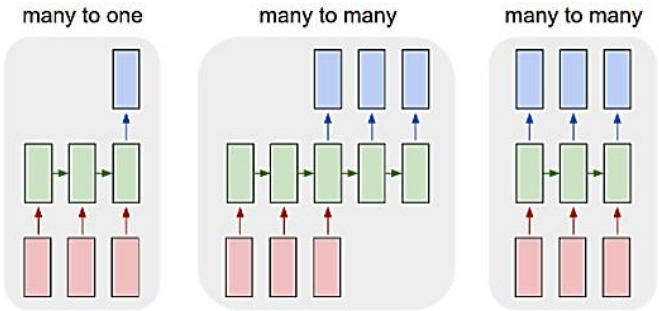


Figure 5: The Sequences of Machine Learning Network. Image source from Ayush Thakur. (Ayu15)

V. System environment

A. System operating environment

The IDS wrote on a notebook web-based IDE - Google Colab, by using the backend of Google Compute Engine. The requirement of RAM is 12 GB and the requirement of Disk is 100 GB.

B. Library and framework of IDSs notebook

The IDS wrote by using Python 3.7.12. It applied an open-source library - Tensor flow and Keras, for high-performance numerical computation, train and run deep neural networks, and deep learning, such as Multi-Layer Perceptron (MLP) Classifier, Long Short-Term Memory (LSTM) Classifier, and Auto Encoder (AR) Classifier.

While Tensor and Keras handle the deep neural network and deep learning classifier, Sklearn in this project used to train and run classical machine learning algorithms, both supervised and unsupervised, such as Support Vector Machine (SVM), K-Nearest-Neighbor (KNN). Moreover, it has been used for data mining and analysis, calculating the accuracy of the model, generating a classification report of the model,

NumPy in this project was used for normalizing and reshaping data frames, handling large multi-dimensional arrays, matrix processing, linear algebra, and some scientific computations.

Pandas in this project are used for data preprocessing, extraction, analysis, and grouping or filtering data.

Matplotlib in this project was used for data visualization, creating and plotting 2D graphs, such as histogram, error chart, bar chart, etc.

Drive in this project used for import data and notebook for google colab from cloud file storage - google drive.

VI. Data preprocessing

Data preprocessing can increase efficiency and reduce the size of data frames. Improve the speed and time consuming while training data. Moreover, the dataset is more readable after the preprocessing. (Abh17)

A. Import dataset

To import the dataset, the dataset of this project was uploaded to a cloud file storage service - google drive. Permission for accessing google drive is needed while using IDS.

The default location of the google drive disk is mounted at '/content/gdrive'.

```
1.  from google.colab import drive

2.  drive.mount('/content/gdrive')

3.  # importing dataset

4.  data = pd.read_csv('/con-
    tent/gdrive/MyDrive/Colab Notebooks/1-
    IDS/datasets/KDDTrain+.txt',header=None, names
    =col_names)
```

B. Labeling and refining

Since the dataset does not have attributes attached to the dataset. It is needed to provide the labels to those column names. To increase the readability of data, and the classifiers are easier to identify and handle the date after labelling.

There have three parts of the dataset needed to label or refine.

1) Column name

```
1.  col_names = ["duration","protocol_type","ser-
    vice","flag","src_bytes",

2.      "dst_bytes","land","wrong_fragment","ur-
    gent","hot","num_failed_logins",

3.      "logged_in","num_compro-
    mised","root_shell","su_attempted","num_root",

4.      "num_file_creations","num_shells","num_ac-
    cess_files","num_outbound_cmds",

5.      "is_host_login","is_guest_login","count","
    srv_count","serror_rate",

6.      "srv_serror_rate","rerror_rate","srv_rer-
    ror_rate","same_srv_rate",

7.      "diff_srv_rate","srv_diff_host_rate","dst_
    host_count","dst_host_srv_count",

8.      "dst_host_same_srv_rate","dst_host_diff_sr
    v_rate","dst_host_same_src_port_rate",

9.      "dst_host_srv_diff_host_rate","dst_host_se
    rror_rate","dst_host_srv_serror_rate",

10.     "dst_host_rerror_rate","dst_host_srv_rer-
    ror_rate","label","difficulty_level"]
```

2) Label the data set

In order to extract the feature, those 22 attack types classified into 4 attack class, which is DoS (Denial of Service), R2L (Root to Local), U2R (User to Root), Probe (Probing).

```
1.  def change_label(df):

2.    df.label.re-
    place(['apache2','back','land','nep-
    tune','mailbomb','pod','processta-
    ble','smurf','teardrop','ud-
    pstorm','worm'],'Dos',inplace=True)

3.    df.label.replace(['ftp_write','guess_pass-
    wd','httptunnel','imap','multi-
    hop','named','phf','sendmail',

4.          'snmpgetattack','snmpguess','spy','ware
    zclient','warezmas-
    ter','xlock','xsnoop'],'R2L',inplace=True)
```

Ka Leung, CHEUNG
Computer Science Department, Swansea University

```
5.    df.label.re-
    place(['ipsweep','mscan','nmap','portsweep','s
    aint','satan'],'Probe',inplace=True)

6.    df.label.replace(['buffer_overflow','load-
    module','perl','ps','root-
    kit','sqlattack','xterm'],'U2R',inplace=True)

7.
```

Table 6: Distribution of attack classes after grouping.

	Count
Normal	67,343
DOS	45,927
Probe	11,656
R2L	995
U2R	52
Total	125,973

3) Refine the data set

Since the difficulty level is useless in training data, it removed the attribute 'difficulty_level'. It reduces the biased toward the high frequent records.

```
1.  data.drop(['difficulty_lev-
    el'],axis=1,inplace=True)
```

VII. Data Normalization

In order to keep data in unity and keep value consistent, it needs to normalize before training the data. (abh21)

A. Data Standardization and Scaling

This project used Standard Scaler scaled 38 numeric attributes of the data frame after removed the 'difficulty level' attribute.

Since variables are measured at different scales do not contribute equally to the model fitting, it might create a bias. it is needed to standardize the data before fitting a machine learning model by using Standard Scaler. To transform the distribution of the data. After standardization, the distribution has a mean value of 0 and a standard deviation of 1. (μ=0, σ=1). (Ser20)

Equation 1: the equation of standardization

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

Equation 2: the equation of mean

$$\mu = \frac{1}{n}\sum_{i=1}^{n}(x_i)$$

Equation 3: the equation of standard deviation

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

```
1.  std_scaler = StandardScaler()

2.  def normalization(df,col):

3.    for i in col:

4.      arr = df[i]

5.      arr = np.array(arr)

6.      df[i] = std_scaler.fit_transform(arr.re-
    shape(len(arr),1))

7.    return df
```

'df ' in line 7 represents the x, which is the standard score. The range of normalized feature values is between 0 and 1.

After that, StandardScaler() will normalize the features and each column of x individually.

B. One-Hot-Encoding

Since machine learning algorithms have the higher performance of finding patterns from numbers. It is needed to transform the non-numeric attributes into numerical values by using the one-hot-encoding function. There are three attributes that are non-numeric, which are protocol_type, service, flag respectively. These three attributes are represented in the binary value. For example, the protocol type feature has three attributes, which are TCP, UDP, and ICMP. These three attributes will be represented as 001, 010, 011 respectively. After these three features are converted into on-hot-encoding, 84 additional attributes will be added. The total size of the data frame will be increased from 41 to 122-dimensional features, as it included 38 numeric and 84 binary values.

```
1.  categorical = data['protocol_type','ser-
    vice','flag']

2.  categorical = pd.get_dummies(categorical,col-
    umns=cat_col)
```

the disadvantage of using one-hot encoding is high attributes demand since additional columns will create for each attribute and respective the attributes by binary value, 0/1.

VIII. Machine-Learning Network

A. many to one network

In this network, the classifier will only predict a single output. Only one desired output can be triggered. The attack labels have been classified into two categories as an output option, which are 'normal' and 'abnormal' respectively. The attack label using LabelEncoder() and a one-

hot-encoding technique. The encoded label is saved in the 'intrusion' attribute. Label Encoder required no additional attributes.

Figure 6: The distribution of data frame in many to one network

Copy a set of data frame for many to one network labeling for further prediction usage. Named m2o_data.

```
1.  label_many2one = preprocessing.LabelEncoder()

2.  m2o_label = m2o_label.apply(la-
    bel_many2one.fit_transform)

3.  m2o_data['intrusion'] = m2o_label

4.  m2o_data = pd.get_dummies(m2o_data,col-
    umns=['label'],prefix="",prefix_sep="")

5.  m2o_data['label'] = m2o_label
```

B. *many to many network*

In this network, the classifier will predict multiple outputs. The attack label in this network is classified into 5 attributes, which are 'normal', 'U2R', 'R2L', 'Probe', 'DOS' respectively. The labeling used LabelEncoder() and a one-hot-encoding technique. The encoded label is saved in the 'intrusion' attribute.

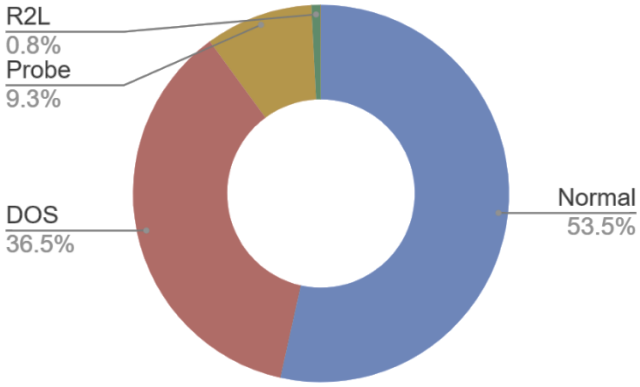


Figure 7: The distribution of data frame in many to many network

Copy a set of data frame for many2many network labeling for further multiple prediction usage. Named m2m_data.

```
1.  label_many2many = preprocessing.LabelEncoder()

2.  m2m_label = m2m_label.apply(la-
    bel_many2many.fit_transform)

3.  m2m_data['intrusion'] = m2m_label

4.  m2m _data = pd.get_dummies(m2m_data,col-
    umns=['label'],prefix="",prefix_sep="")

5.  m2m _data['label'] = m2m_label
```

IX. Feature Extraction

To extract the most correlated features, help machine learning algorithm finds out the pattern of the data.

The total number of attributes for the dataset of 'many to one network' is 45. And the total number of attributes for the dataset of 'many to many networks' is 48, since it requires more attack labels.

A. *Pearson correlation coefficient*

Pearson correlation coefficient is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations. (Pea21) It is used to measure how strong a relationship is between two variables, then to find out the features and attributes of the data frame.

Equation 4: the Pearson correlation coefficient formula definition

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

$$r = \text{correlation coefficient}$$
$$x_i = \text{values of the } x - \text{variable in a simple}$$
$$\bar{x} = \text{mean of the values of the } x - \text{variable}$$
$$y_i = \text{values of the } y - \text{variable in a sample}$$
$$\bar{y} = \text{mean of the values of the } y - \text{variable}$$

After the applied the Pearson correlation coefficient, it can plot a correlation matrix, then get the highest correlated features. (Pea211)

The attributes of 'm2m_data' and 'm2o_data' used Pearson Correlation Coefficient. Only select the attributes that more than 0.5 correlation coefficient with the 'intrusion' attribute. (pan21)

```
1.  numeric_df = df[numeric_col]

2.  numeric_df['intrusion'] = df['intrusion']

3.  corr= numeric_df.corr()

4.  corr_y = abs(corr['intrusion'])

5.  highest_corr = corr_y[corr_y >0.5]

6.  highest_corr.sort_values(ascending=True)
```

After comparing the data frame to the encoded intrusion label attribute. It showed the most correlated features attributes, which showed in table 9.

Table 7: The attributes which have more than 0.5 correlation when comparing the m2o_data frame with encoded attack label attribute.

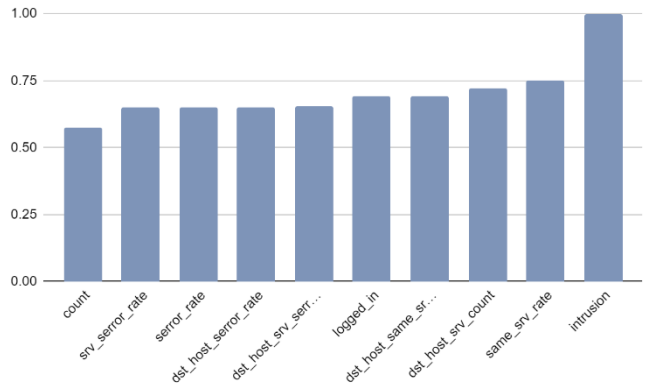
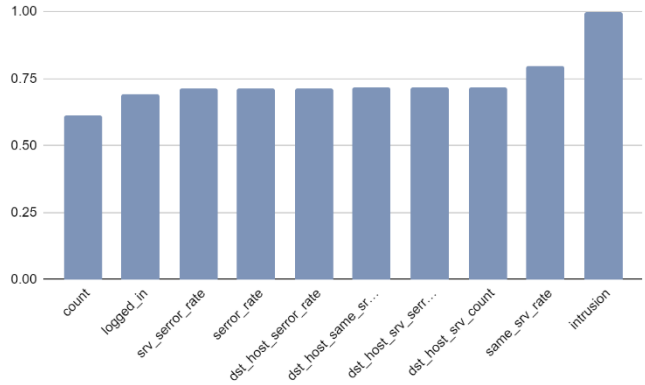


Table 8: The attributes which have more than 0.5 correlation when comparing the m2m_data frame with encoded attack label attribute.



After found desired attributes by Pearson correlation coefficient. Those attributes will join with the one-hot encoded data frame of intrusion, then be saved to the disk for future machine learning data training usage.

X. Machine Learning Classification (Many to One Network)

The dataset split into 75% for training and 25% for testing. The machine learning classification require two set of data frames, which are X and Y.

X is dataset that excluded encoded intrusion attribute for supervised algorithms usage. Y is dataset, which is target attribute, it labelled attack type for each record, for supervising the result. (Cos20)

In many to one network dataset, there has 93 feature attributes.

```
1.  X = df_data.iloc[:,0:93].to_numpy() # dataset
    excluded the target attribute

2.  Y = df_data['intrusion'] # target attribute

3.  X_train, X_test, y_train, y_test =
    train_test_split(X,Y, test_size=0.25, ran-
    dom_state=42)
```

A. *Support Vector Machine Classifier*

1) *Classifier model structure*

The Support Vector Machine Classifier used kernel as linear. Since it is a supervised model, it needs to fit both datasets X and Y.

```
1.  lsvm = SVC(kernel='linear',gamma='auto')

2.  lsvm.fit(X_train,y_train)

3.  SVC(C=1.0, break_ties=False, cache_size=200,
    class_weight=None, coef0=0.0, decision_func-
    tion_shape='ovr', degree=3, gamma='auto', ker-
    nel='linear', max_iter=-1, probability=False,
    random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

2) *Model accuracy and report*

After use x_train dataset for data training, it used x_test for testing the model, and the predicted result saved in y_pred. The accuracy of support vector machine classifier is 96.69%.

```
1.  y_pred = lsvm.predict(X_test)

2.  ac = accuracy_score(y_test, y_pred)*100

3.  print(classification_re-
    port(y_test, y_pred,target_names=le1.clas-
    ses_))
```

Table 9: The classification report of support vector machine model

	precision	recall	f1-score	support
abnormal	0.97	0.96	0.97	14720
normal	0.96	0.97	0.97	16774
accuracy			0.97	31494
macro avg	0.97	0.97	0.97	31494
weighted avg	0.97	0.97	0.97	31494

B. *K-nearest-neighbor Classifier*

1) *Classifier model structure*

Set 5 neighbors for the K-nearest-neighbor model with uniform weights. Since it is a supervised model, it needs to fit both datasets X and Y.

```
1.  knn=KNeighborsClassifier(n_neighbors=5)

2.  knn.fit(X_train,y_train)

3.  KNeighborsClassifier(algorithm='auto',
    leaf_size=30, metric='minkowski',
```

```
    metric_params=None, n_jobs=None, n_neigh-
    bors=5, p=2, weights='uniform')
```

2) Model accuracy and report

The accuracy of support vector machine classifier is 98.55%.

```
1.  y_pred=knn.predict(X_test)

2.  ac=accuracy_score(y_test, y_pred)*100

3.  print(classification_re-
    port(y_test, y_pred,target_names=le1.clas-
    ses_))
```

Table 10: The classification report of KNN model

	precision	recall	f1-score	support
abnormal	0.99	0.98	0.98	14720
normal	0.99	0.99	0.99	16774
accuracy			0.99	31494
macro avg	0.99	0.99	0.99	31494
weighted avg	0.99	0.99	0.99	31494

C. *Multi-Layer Perception Classifier*

1) Classifier model structure

Since the Multi-Layer Perception model is a linear stack of the layer where each layer has just one input tensor and one output tensor. It needs to create a sequential model by using Sequential().

The first input layer was created with 93 input dimensions.

The second hidden layer was created with 50 neurons, RELU activation.

The last output layer is created with the Sigmoid function. Comparing the sigmoid function to the RELU function, RELU is sensitive to 0 and 1, it output 0 if it receives any negative input, otherwise, it will output x. Given that: $y = max(0, x)$.

This structure model has a total of 4751 trainable params.

The model used the x_train dataset for data training with epochs 100 times, the optimizer is adam and 5000 batch size. And compile the model with binary cross-entropy of loss function since the output is binary.

Table 11: Model structure of MLP

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 50)	4700
dense_4 (Dense)	(None, 1)	51

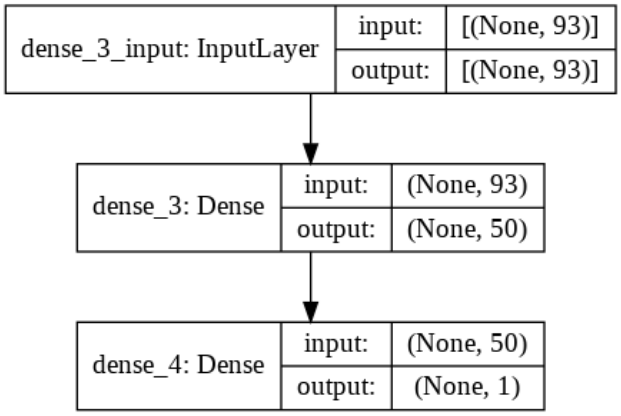


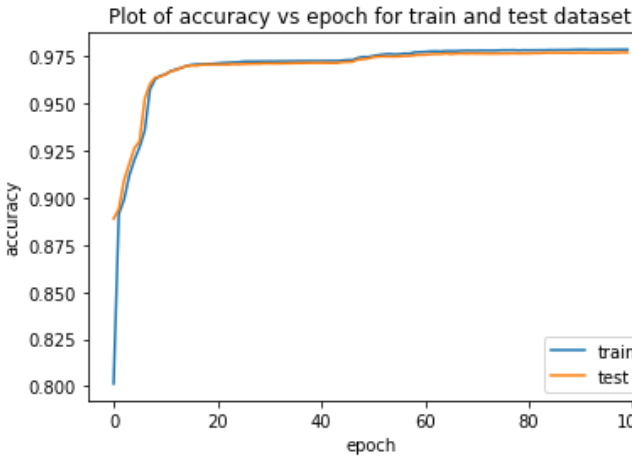
Figure 8: Model Layer of MLP

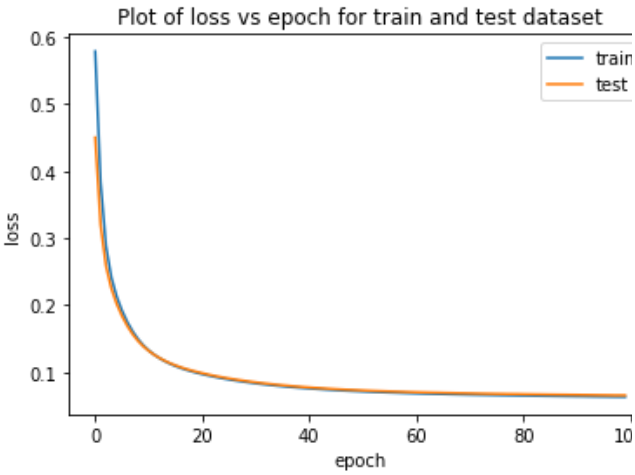
```
1.  Classifier model structure

2.  mlp = Sequential() # create model

3.  #input layer with 50 neurons

4.  mlp.add(Dense(units=50, in-
    put_dim=X_train.shape[1], activation='relu'))

5.  # output layer with sigmoid function

6.  mlp.add(Dense(units=1,activation='sigmoid'))

7.  # set loss function, optimizer, metrics and
    then compile model

8.  mlp.compile(loss='binary_crossentropy', opti-
    mizer='adam', metrics=['accuracy'])

9.  # training the model with training dataset

10. history = mlp.fit(X_train, y_train,
    epochs=100, batch_size=5000,valida-
    tion_split=0.2)

11. mlp.compile(loss='binary_crossentropy', opti-
    mizer='adam', metrics=['accuracy'])
```

2) Model accuracy and report

The loss rate of the model is 0.06432 and the accuracy is 97.81% after 100 epochs.

There has a positive relationship between the accuracy rate and the number of epochs.






```
1.  test_results = mlp.evaluate(X_test, y_test,
    verbose=1)

2.  print(f'Test results - Loss: {test_results[0]}
    - Accuracy: {test_results[1]*100}')
```

D. Long Short-Term Memory Classifier

1) Classifier model structure

The first input layer was created with 93 input dimensions.

LSTM layer with 50 neurons cells.

The last output layer was created with Sigmoid function with 1 neuron.

This structure model has a total of 10451 trainable params.

The model used the x_train dataset for data training with epochs 100 times and 5000 batch size. And compile the model with adam optimizer, binary cross-entropy of loss function since the output is binary.

Table 12: Model structure of LSTM

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 93, 50)	10400
dense_5 (Dense)	(None, 93, 1)	51

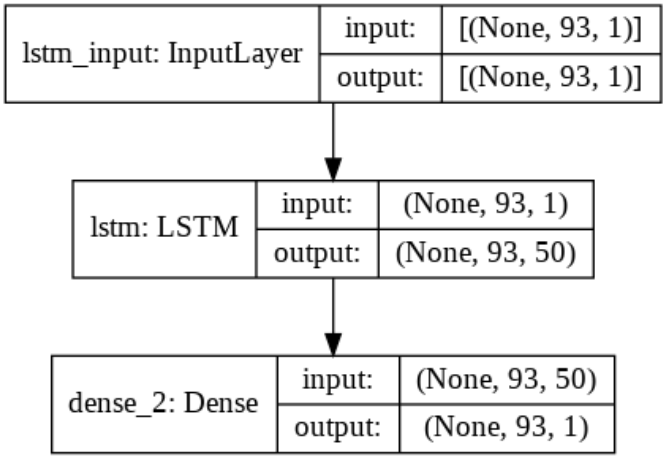


Figure 9: Model layers of LSTM

```
1.  lstm = Sequential()

2.  # first layer with 50 neurons

3.  lstm.add(LSTM(units=50, return_sequences=True,
    input_shape=(x_train.shape[1],1)))

4.  # last layer with sigmoid activation

5.  lstm.add(Dense(1, activation='sigmoid'))

6.  # training the model on training dataset

7.  his-
    tory = lstm.fit(x_train, y_train, epochs=1, ba
    tch_size=5000,validation_split=0.2)

8.   # defining loss function, optimizer, metrics
    and then compiling model

9.  lstm.compile(loss='binary_crossentropy',opti-
    mizer='adam',metrics=['accuracy'])
```

2) Model accuracy and report

The loss rate of the model is 0.0689 and the accuracy is 56.7 % after 100 epochs.

```
1.  test_results = lst.evalu-
    ate(x_test, y_test, verbose=1)

2.  print(f'Test results - Loss: {test_re-
    sults[0]} - Accuracy: {test_re-
    sults[1]*100}%')
```

E. Auto Encoder Classifier

1) Classifier model structure

The first input layer was created with 50 neurons with RELU function.

The last decoding (output) layer is created with the softmax function. Since the output type is more than one, the sigmoid activation function is not preferred in multi-class classification. Instead of using the sigmoid function,

Ka Leung, CHEUNG
Computer Science Department, Swansea University

softmax calculates the relative probabilities and has the ability to solve the multi-class classification problem.

This structure model has a total of 9443 trainable params.

The model used the x_train dataset for data training with epochs 100 times and 500 batch size. And compile the model with mean_squared_error of the loss function.

Table 13: Model structure of autoencoder

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 93)]	0
dense_3 (Dense)	(None, 50)	4700
dense_4 (Dense)	(None, 93)	4743

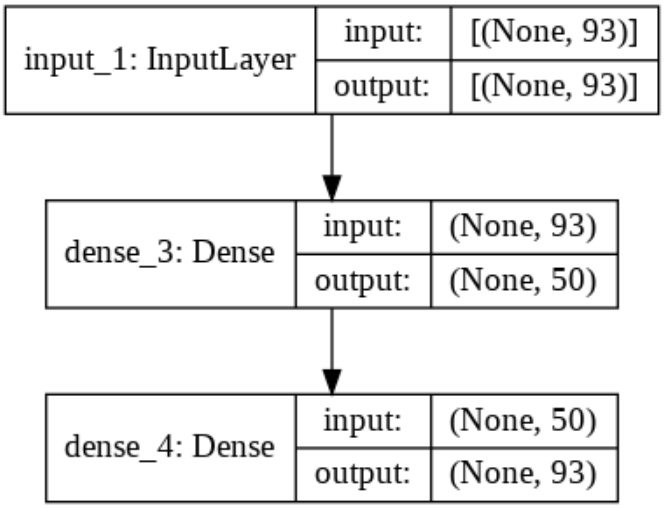


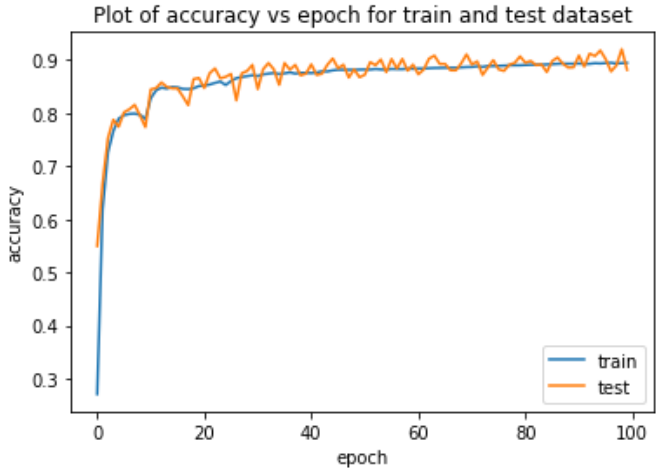
Figure 10: : Model layers of autoencoder

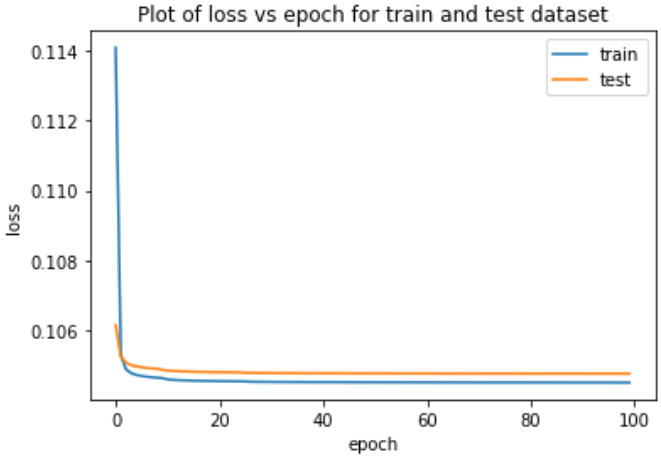
```
#input layer

input_layer = Input(shape=(input_dim, ))

#encoding layer with 50 neurons

encoder = Dense(encoding_dim, activation="relu")(input_layer)

#decoding and output layer

output_layer = Dense(input_dim, activation='softmax')(encoder)

 # creating model with input, encoding, decoding, output layers

autoencoder = Model(inputs=input_layer, outputs=output_layer)

 # defining loss function, optimizer, metrics and then compiling model

autoencoder.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])

 # training the model on training dataset

history = autoencoder.fit(X_train, X_train, epochs=100,batch_size=500,validation_data=(X_test, X_test)).history

# defining loss function, optimizer, metrics and then compiling model

autoencoder.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
```

2) Model accuracy and report

The loss rate of the model is 0.1047 and the accuracy is 88.18 % after 100 epochs. And there has reconstruction_error total count is 31494 and the mean is 0.1047. the total count of the true class is 31494 and the mean is 0.5326.

There has a positive relationship between accuracy and the number of epochs.






```
# predicting target attribute on testing dataset

test_results = autoencoder.evaluate(X_test, X_test, verbose=1)

print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%')

# calculating reconstruction error
```

```
predictions = autoencoder.predict(X_test)

mse = np.mean(np.power(X_test - predictions, 2), axis=1)

error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class': y_test})
```

XI. Machine Learning Classification (Many to Many Networks)

Similar to the above network, the dataset split into 75% for training and 25% for testing. The machine learning classification requires two sets of data frames, which are X and Y.

X is the dataset that excluded encoded intrusion attribute for supervised algorithms usage. Y is the dataset, which is the target attribute, it labeled the attack type for each record, for supervising the result.

In many to one network datasets, there have 93 feature attributes.

```
X = multi_data.iloc[:,0:93].to_numpy()

Y = multi_data['intrusion']

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25, random_state=42)
```

A. Support Vector Machine Classifier

1) Classifier model structure

The Support Vector Machine Classifier used kernel as linear. Since it is a supervised model, it needs to fit both datasets X and Y.

```
svm=SVC(kernel='linear',gamma='auto')

svm.fit(X_train,y_train) # training model on training dataset

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)
```

2) Model accuracy and report

The accuracy of the Support Vector Machine Classifier is 95.24%.

Table 14: The classification report of support vector machine model

	precision	recall	f1-score	support
DOS	0.95	0.96	0.96	11484
Probe	0.86	0.76	0.82	2947
R2L	0.61	0.60	0.60	247
U2R	0	0	0	15
Normal	0.97	0.98	0.98	16774
accuracy			0.95	31494
macro avg	0.687	0.67	0.67	31494
weighted avg	0.95	0.95	0.95	31494

```
y_pred=lsvm.predict(X_test) # predicting target attribute on testing dataset

ac=accuracy_score(y_test, y_pred)*100  # calculating accuracy of predicted data

print("LSVM-Classifier Multi-class Set-Accuracy is ", ac)

print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

B. K-nearest-neighbor Classifier

1) Classifier model structure

The K-nearest-neighbor Classifier is built with 5 neighbors with uniform weights. Since it is a supervised model, it needs to fit both datasets X and Y.

```
knn=KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train,y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=5, p=2, weights='uniform')
```

2) Model accuracy and report

The accuracy of the Support Vector Machine Classifier is 98.29%.

Table 15: The classification report of the KNN model

	precision	recall	f1-score	support
DOS	0.99	0.99	0.99	11484
Probe	0.96	0.97	0.96	2947
R2L	0.62	0.87	0.89	247
U2R	0.40	0.13	0.20	15
Normal	0.99	0.99	0.99	16774
accuracy			0.98	31494
macro avg	0.85	0.79	0.81	31494
weighted avg	0.98	0.98	0.98	31494

```
y_pred=knn.predict(X_test)

ac=accuracy_score(y_test, y_pred)*100
```

Ka Leung, CHEUNG
Computer Science Department, Swansea University

```
print("KNN-Classifier Multi-class Set-Accuracy is ", ac)

print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

C. Multi-Layer Perception Classifier

1) Classifier model structure

The first input layer was created with 93 input dimensions.

The second hidden layer was created with 50 neurons, RELU activation.

The last output layer was created with a softmax function with 5 neurons.

This structure model has a total of 4955 trainable params.

The model used the x_train dataset for data training with epochs 100 times, and the optimizer is adam and 5000 batch size. And compile the model with categorical_crossentropy of the loss function.

Table 16: the model of Multi-Layer Perception Classifier

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 50)	4700
dense_6 (Dense)	(None, 5)	255

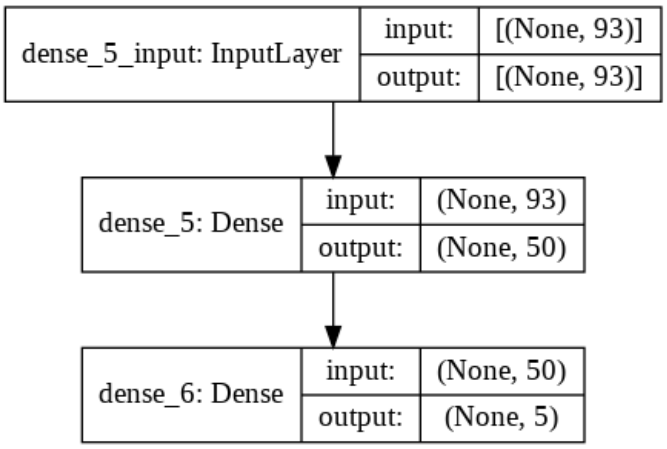


Figure 11: the layer of Multi-Layer Perception Classifier

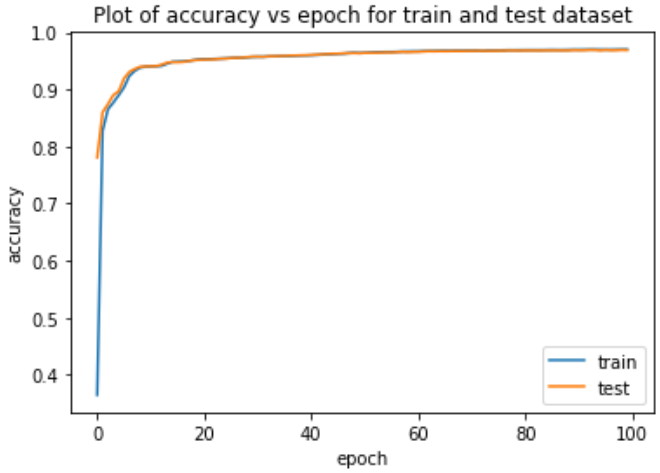
```
mlp = Sequential() # initializing model

mlp.add(Dense(units=50, input_dim=X_train.shape[1], activation='relu'))

mlp.add(Dense(units=5,activation='softmax'))

mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

 history = mlp.fit(X_train, y_train, epochs=100, batch_size=5000,validation_split=0.2)
```

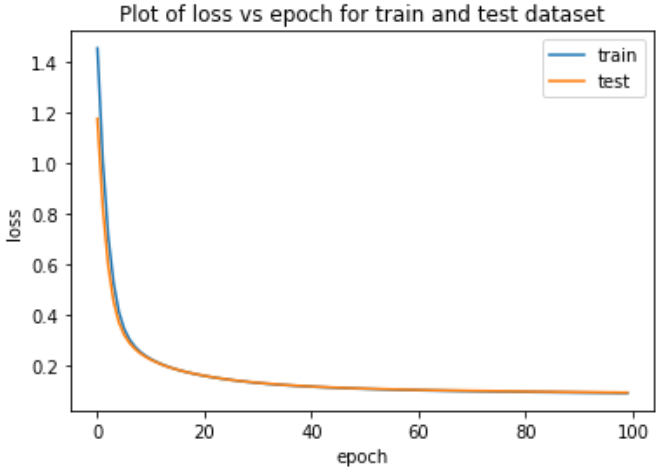
```
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

2) Model accuracy and report

The loss rate of the model is 0.0910 and the accuracy is 96.85 % after 100 epochs.

There has a positive relationship between accuracy and the number of epochs.






```
# predicting target attribute on testing dataset

test_results = mlp.evaluate(X_test, y_test, verbose=1)

print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]*100}%')

```

D. Auto Encoder Classifier

1) Classifier model structure

The first input layer was created with 50 neurons with RELU function.

The last output layer is created with the softmax function.

This structure model has a total of 9443 trainable params.

The model used the x_train dataset for data training with epochs 100 times and 500 batch size. And compile the model with mean_squared_error of the loss function.

Table 17: Model structure of Auto Encoder Classifier

Layer (type)	Output Shape	Param #
input_2 (Input-Layer)	[(None, 93)]	0
dense_7 (Dense)	(None, 50)	4700
dense_8 (Dense)	(None, 93)	4743

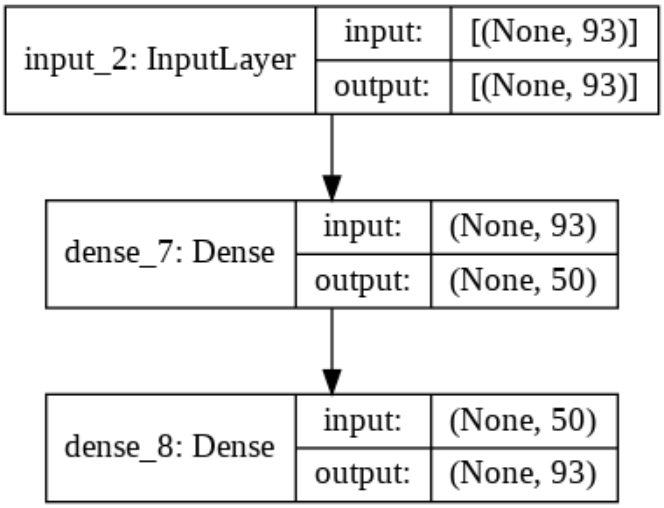


Figure 12: the layers of Auto Encoder Classifier

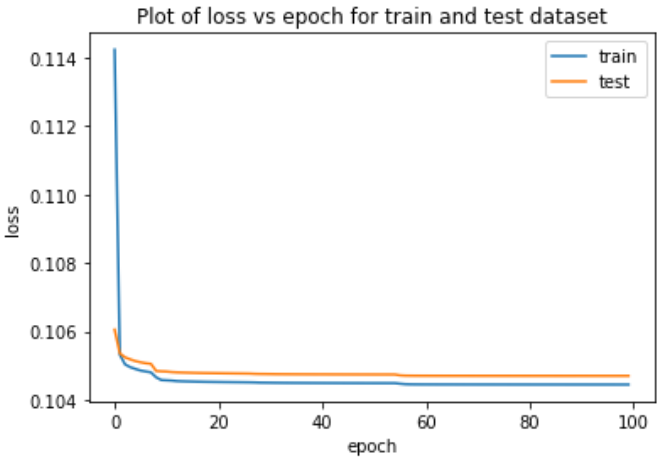
```
#input layer

input_layer = Input(shape=(input_dim, ))

#encoding layer with 50 neurons

encoder = Dense(encoding_dim, activation="relu")(input_layer)

#decoding and output layer

output_layer = Dense(input_dim, activation='softmax')(encoder)


# creating model with input, encoding, decoding, output layers

autoencoder = Model(inputs=input_layer, outputs=output_layer)

```

```
# defining loss function, optimizer, metrics and then compiling model

autoencoder.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])

# training the model on training dataset

history = autoencoder.fit(X_train, X_train, epochs=100,batch_size=500,validation_data=(X_test, X_test)).history

# defining loss function, optimizer, metrics and then compiling model

autoencoder.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
```

2) Model accuracy and report

The loss rate of the model is 0.1047 and the accuracy is 91.91 % after 100 epochs. And there has reconstruction_error total count is 31494 and the mean is 0.1047. the total count of the true class is 31494 and the mean is 2.22428

There has a positive relationship between accuracy and the number of epochs.




```
1.  # predicting target attribute on testing da-
    taset

2.  test_results = autoencoder.evalu-
    ate(X_test, X_test, verbose=1)

3.  print(f'Test results - Loss: {test_re-
    sults[0]} - Accuracy: {test_results[1]*100}%')

4.   # calculating reconstruction error

5.  predictions = autoencoder.predict(X_test)

6.  mse = np.mean(np.power(X_test - predic-
    tions, 2), axis=1)

7.  error_df = pd.DataFrame({'reconstruction_er-
    ror': mse, 'true_class': y_test})
```

XII. Experimental Results

This project used the traditional metrics to find out the most stable, accurate, and trustable classifier in IDSs.

For the definition of the above metrics (Jas20):

$$prceision = \frac{TP}{TP + FP} - (12)$$

$$Recall = \frac{TP}{TP + FN} - (13)$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} - (14)$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} - (15)$$

Given that: TP is True Positive, TN is True Negative, FP is False Positive, FN is False Negative,

A. *many to one network*

By comparing different classifiers under many to one network with the same data set. KNN classifier has the highest accuracy rate, which reached 98.55%, overall kind of metrics, the reason mainly because it is a kind of supervised algorithms. For those unsupervised deep learning algorithms, the MLP classifier gave 97.79% overall performance.

	AE	LSTM	MLP	SVM	KNN
Attack Class					
Normal	81.09	79	78.49	77.69	99
Abnormal	92.91	91.25	91.57	90.96	99
Average Precision	87	85.13	85.03	84.32	99
Attack Class					
Normal	96.96	96.47	96.69	96.55	98
Abnormal	63.79	58.92	527.57	55.56	98
Average Recall	80.37	77.7	77.13	76.06	98
Attack Class					
Normal	88.32	86.86	86.65	86.09	99
Abnormal	75364	71.61	70.69	68.99	98
Average F1 score	81.98	79.24	78.67	77.54	98.5
Overall Performance Accuracy Rate	92.26	83.05	97.79	96.69	98.55

B. *Many to Many networks*

Under the many to many networks, after measuring the metrics of precision, recall, and F1 score. For supervised algorithms, KNN has the highest performance, which is 98.29%. For unsupervised algorithms, MLP performance is the best, which is 96.92%.

	AE	MLP	SVM	KNN
Attack Class				
Normal	85.03	79.46	78.37	99
DOS	97.05	97.21	96.52	99
Probe	69.82	64.33	77.03	96
R2L	99.49	99.19	95.15	92

Ka Leung, CHEUNG
Computer Science Department, Swansea University

Average Precision	87.85	85.05	86.77	96.5
Attack Class				
Normal	96.19	96.58	96.64	99
DOS	98.18	97.42	96.86	99
Probe	94.03	95.16	97.9	97
R2L	39.78	0.81	4.97	87
Average Recall	82.04	72.49	74.09	95.5
Attack Class				
Normal	90.27	87.10	86.55	99
DOS	97.61	97.08	96.69	99
Probe	80.14	77.13	77.13	96
R2L	56.83	11.74	11.74	89
Average F1 score	81.21	68.26	68.26	95.75
Overall Performance	91.22	96.92	95.24	98.29

XIII. Discussion

When designing a machine learning model, there are a few key factors that can affect the performance and result of the model seriously.

A. *The selection of the Loss function*

A loss function is a measure of error between the value of the model predicts and the value is. It is used to calculate the gradients. (Moh20)

This project used three types of the loss function, which is binary cross-entropy, mean_squared_error, categorical_crossentrop. Comparing the sigmoid function to the RELU function, RELU is sensitive to 0 and 1, it output 0 if it receives any negative input, otherwise, it will output x.

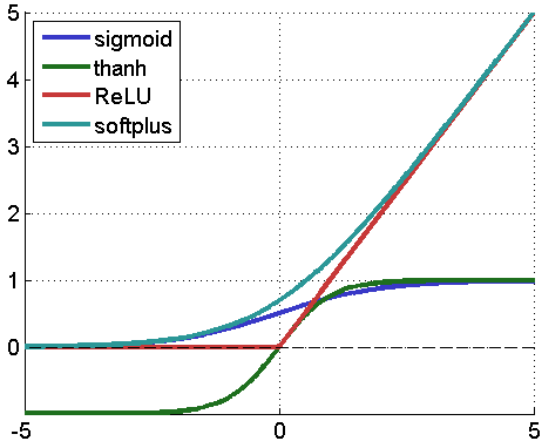


Figure 13: the function of the relu, sigmoid, and softmax

The activation function is different when the algorithms are different. For example, if the output type is more than one, the sigmoid activation function will not be preferred in multi-class classification. Instead of using the sigmoid function, softmax calculates the relative probabilities and has the ability to solve the multi-class classification problem. (Val19)

B. *Gradient Descent*

Gradients are used to update the weights of the Neural Net. It usually uses this optimization algorithm for finding a local minimum of a differentiable function. Since it needs to loop several of iterative, propagation backward can increase the performance of the Gradient descent. However, back-propagation usually causes a gradient vanishing problem since the weights of the model changed (Van21)

C. *Overfitting*

Overfitting problems usually happened after the parameters are getting large, such as millions or billions while the training data or the complexity of the network is too large. (Shi19)

D. *Explanatitrustabilityden layer*

To understand the model better and increase the trustability of the deep learning system. It can be achieved by explaining a deep learning model using DeepExplainer. and visualize the first prediction. (Ker21)

XIV. Conclusion

This paper presented the process of a trustworthy AI-Assisted Network Intrusion Detection System (IDS) built with several machine learning algorithms. It shows this IDS can be trustable, which achieved over 90% in the recall, f1-score, and accuracy rate with KNN, MLP, AE classifiers. This proved this IDS has the ability to detect and analyse different types of attacks such as DOS, R2L, Probe, U2R.

In the future, the system can apply a deep learning explainer, to increase the trust of humans. In addition, it can be developed as a real-time IDS to detect and analyze the real-time IP packet, to reduce the existing network threats.

It is believed that Artificial intelligence is trustable in ana-lysing network majority of abnormal IP packets, which can reduce the workload for humans, and maybe more reliable than the majority of humans, in making decisions.

XV. Bibliography

(n.d.).

abhinav-bhardwaj. (2021). *Network-Intrusion-Detection-Using-Deep-Learning.* Retrieved from github: https://github.com/abhinav-bhardwaj/Network-Intrusion-Detection-Using-Machine-Learning

AL-Maksousy, L. H. (2018). Applying Machine Learning to Advance Cyber. *Old Dominion University*, 53.

Alto, V. (2019). *Neural Networks: parameters, hyperparameters and optimization strategies*. Retrieved from towardsdatascience: https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5

Artificial intelligence. (2021, March 16). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Artificial_intelligence

Brownlee, J. (2020, 08 2). *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/

Cosimo Ieracitano, A. A. (2018). Statistical Analysis Driven Optimized Deep Learning System for Intrusion Detection. *researchgate*.

Cosimo Ieracitanoa, Ahsan Adeelb, Francesco Carlo, Morabitoa, Amir Hussainc. (2020, 4 28). *A novel statistical analysis and autoencoder driven intelligent intrusion detection approach.* Retrieved from Volume 387, Neurocomputing: A novel statistical analysis and autoencoder driven intelligent intrusion detection approach

Denial-of-service attack. (2021). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Denial-of-service_attack

Dubey, A. (2017). *Network Intrusion Detection using Deep Learning.* Retrieved from medium: https://medium.com/geekculture/network-intrusion-detection-using-deep-learning-bcc91e9b999d

Georgy Evgen'evich SHILOV, & Richard A SILVERMAN. (1971). Linear Algebra. Revised English Ed. Trs. RA Silverman. *Prentice-Hall*.

Gonzalo Mar, Pedro Casas, & German Capdehourat. (2020). DeepMAL - Deep Learning Models for Malware. *Montevideo, Uruguay*, 7.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.*

H. Sundmaeker, P. G. (2010). Vision and challenges for realising the internet of things.

Intrusion detection system. (2021, March 16). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Intrusion_detection_system

J. F., D., & S., C. (2010, July). What is smart manufacturing. Retrieved from https://scholar.google.com/scholar_lookup?hl=en&publication_year=2010&author=S.+Chand&author=J.+F.+Davis&title=What+is+smart+manufacturing%3F

Jong-Hwan Kim, Weimin Yang, Jun Jo, Peter Sincak, & Hyun Myung. (2015). *Robot Intelligence Technology and Applications 3* (Vol. 345). Springer.

Keras LSTM for IMDB Sentiment Classification¶. (2021). Retrieved from shap: https://shap.readthedocs.io/en/latest/example_notebooks/text_examples/sentiment_analysis/Keras%20LSTM%20for%20IMDB%20Sentiment%20Classification.html

Kostas, K. (2018). Anomaly Detection in Networks Using Machine Learning. *University of Essex*.

Lan Liu, Jun Lin, Pengcheng Wang, Langzhou Liu, & Rongfu Zhou. (2020,). Deep Learning-Based Network Security Data Sampling and. *Discrete Dynamics in Nature and Society Article ID 4163825, 2020*, 9. Retrieved from https://doi.org/10.1155/2020/4163825

Loukas, S. (2020). *How and why to Standardize your data: A python tutorial*. Retrieved from towardsdatascience: https://towardsdatascience.com/how-and-why-to-standardize-your-data-996926c2c832

M. Tavallaee, E. B. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*.

M. Tavallaee, E. B. (2012). *Nsl-kdd dataset.* Retrieved from Google Scholar: http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html

Mathur, P. (2016). Retrieved from https://medium.com/pankajmathur/a-simple-multilayer-perceptron-with-tensorflow-3effe7bf3466

Mulla, M. Z. (2020). *Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?* Retrieved from medium: https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de

pandas.DataFrame.corr¶. (2021). Retrieved from pandas: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

Pearson correlation coefficient. (2021). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Pearson Product-Moment Correlation. (2021). Retrieved from statistics: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

Pintelas1, I. E. (2019). *A CNN–LSTM model for gold price time-series forecasting.* Neural Computing and Applications (.

Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, & Mansoor Alam. (2015). A Deep Learning Approach for Network Intrusion Detection. *The University of Toledo*.

Ram B. Basnet1∗, Riad Shash, Clayton Johnson, Lucas Walgren, & Tenzin Doleck. (n.d.). Towards

Detecting and Classifying Network Intrusion. *Colorado Mesa University*, 2.

Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington DC: Spartan Books,.

S. B. Kotsiantis, Zaharakis, & P. Pintelas. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 3-24.

S. Goel, K. W. (2017). Got phished? internet security and human vulnerability.

S.RoshanG.SrivathsanK.DeepakS.Chandrakala. (2020). Violence Detection in Automated Video Surveillance: Recent Trends and Comparative Studies.

T. Le, J. Kim, & H. Kim. (2017). An effective intrusion detection classifier using long shortterm memory with gradient descent optimization. *IEEE*, 1–6.

Thakur, A. (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. Retrieved from Andrej Karpathy blog: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Vanishing gradient problem. (2021). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Vanishing_gradient_problem#:~:text=In%20machine%20learning%2C%20the%20vanishing,based%20learning%20methods%20and%20backpropagation.&text=The%20problem%20is%20that%20in,weight%20from%20changing%20its%20value.

Verma, S. (2019). *Understanding different Loss Functions for Neural Networks*. Retrieved from towardsdatascience: https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718

Vishwakarma , A. R. (2020). Network Traffic Based Botnet Detection Using Machine Learning. *San Jose State University*, 19.

Wenjie Lu, Jiazheng Li, Yifan Li, & Aijun Sun. (2020). *A CNN-LSTM-Based Model to Forecast Stock Prices.* Hindawi Complexity. Retrieved from https://downloads.hindawi.com/journals/complexity/2020/6622927.pdf

Wikipedia AE. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Autoencoder

Wikipedia SVM. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Support-vector_machine

Xiaoling Tao, Deyan Kong, Yi Wei, & Yong Wang. (2016). A big network traffic data fusion approach based on fisher and deep auto-encoder. *Information*, 7(2):20.

XIE1, H., & LI ZHANG. (2020). *Evolving CNN-LSTM Models for Time Series.* IEEE.