Vietnam National University

University of Information Technology

REPORT

HoneyBear

Mentor	Phạm Văn Hậu
Students: Group 6	Đặng Minh Trí - 12520980 Phạm Minh Sang - 12520352

Table of Contents

I. Abstract

With the quick development and expansion of computer networks during the past decade, security has become a vital issue for computer systems. Intrusion detection is the process to identify instances of network attacks by comparing current activity against the expected actions of an intruder. Most current approaches to this involve the use of rule-based systems to identify indications of known attacks, for example Snort and Suricata. However, these methods are less successful in identifying attacks which vary from expected patterns.

Artificial neural networks provide the potential to identify and classify network activity based on limited, incomplete, and nonlinear data sources. This report presents a framework which takes advantages of neural network approach using a Multi-Layer Perceptron (MLP) for intrusion detection. Different neural network structures are studied to find the optimal neural network with regards to the number of hidden layers. NSL-KDD dataset, which is an upgraded version of KDDcup 99 dataset is chosen because it not only contains only selected record, but also provides a good analysis on various machine learning techniques for intrusion detection.

II. Introduction

Every day, criminals are invading countless homes and offices - not by breaking down windows and doors, but by breaking into laptops, personal computers, and network devices via hacks and bits of malicious code. Billions of dollars are lost every year repairing systems hit by such attacks. Some take down vital systems, disrupting and sometimes disabling the work of hospitals, banks, business companies, etc.

The costs of temporary or permanent damages caused by unauthorized access to computer systems have urged different organizations to increasingly implement various systems to collect and monitor data flow in their networks. These systems are generally referred to as Intrusion Detection Systems (IDSs).

There are two general categories of attacks which intrusion detection technologies attempt to identify - anomaly detection and intrusion detection. In a intrusion detection based IDS, intrusions are detected by looking for activities that correspond to known signatures of intrusions or vulnerabilities.

On the other hand, an anomaly detection based IDS detects intrusions by searching for abnormal network traffic. The abnormal traffic pattern can be defined either as the violation of accepted thresholds for frequency of events in a connection or as a user's violation of the legitimate profile developed for his/her normal behavior. Anomaly detection typically involves the creation of knowledge bases that contain the profiles of the monitored activities.

Most current approaches to the process of detecting intrusions utilize some form of rule-based analysis. Rule-based IDS includes of a set of rules that encode the knowledge of a human "expert". These rules are used by the system to make conclusions about the security-related data.

Unfortunately, rules-based IDSs require frequent updates to remain current. While it offer an enhanced ability to review audit data, the required updates may be ignored or performed infrequently by the administrator. At a minimum, this leads to an IDS with reduced capabilities. At worst, this lack of maintenance will degrade the security of the entire system by causing the system's users to be misled into believing that the system is secure, even as one of the key components becomes increasingly ineffective over time.

Rule-based IDS also suffers from an inability to detect attacks scenarios that may occur over an extended period of time. While the individual instances of suspicious activity may be detected by the system, they may not be reported if they appear to occur in isolation.

Rule-based IDS is also lack of flexibility in the rule-to-audit record representation. Variations in an attack sequence can affect the activity-rule comparison to a degree that the intrusion is not detected by the intrusion detection mechanism. While increasing the level of abstraction of the rule-base does provide a partial solution to this weakness, it also reduces the granularity of the intrusion detection device.

The problem may lie in the fact that the intruder is an intelligent and flexible agent while the rule-based IDS obey fixed rules. This problem can be tackled by the application of soft computing techniques in IDS.

Soft computing is a general term for describing a set of optimization and processing techniques that are tolerant of imprecision and uncertainty. The principal constituents of soft computing techniques are Fuzzy Logic (FL), Artificial Neural Networks (ANNs), Probabilistic Reasoning (PR), and Genetic Algorithms (GAs).

The idea behind the application of soft computing techniques and particularly ANNs in implementing IDS is to include an intelligent agent in the system that is capable of disclosing the latent patterns in abnormal and normal connection audit records, and to generalize the patterns to new connection records of the same class.

Unlike rule-based systems, which can provide the user with a definitive answer if the characteristics which are reviewed exactly match those which have been coded in the rule base, a neural network conducts an analysis of the information and provides a probability estimate that the data matches the characteristics which it has been trained to recognize. While the probability of a match determined by a neural network can be 100%, the accuracy of its decisions relies totally on the experience the system gains in analyzing examples of the stated problem.

The neural network gains the experience initially by training the system to correctly identify preselected examples of the problem. The response of the neural network is reviewed and the configuration of the system is refined until the neural network's analysis of the training data reaches a satisfactory level.

In this report, an off-line intrusion detection system is implemented using Multi-Layer Perceptron (MLP) artificial neural network. Different structures of MLP are examined to find a minimal architecture that is reasonably capable of classification of network connection records. We also take advantages of NSL-KDD dataset to improve system accuracy.

Section I has introduced the basic ideas in intrusion detection and the motivations for this study. Section II reviews IDS, then focuses on Artificial Intelligent, Machine Learning and some basic ideas in neural network theory including Back Propagation Algorithm. Section III deals with the dataset, attack types, and the features used for classifying network connection records. Section IV describes the implementation procedure and training-validation method. Section V presents a new framework with a discussion of the results and possibilities for future work.

III. IDS

An intrusion detection systems is used to monitor and protect networks or systems for malicious activities. They accomplish this by collecting information from a variety of systems and network sources, and then analyzing the information for possible security problems.

- Intrusion detection provides the following:
- Monitoring and analysis of user and system activity
- Auditing of system configurations and vulnerabilities
- Assessing the integrity of critical system and data files
- Statistical analysis of activity patterns based on the matching to known attacks
- Abnormal activity analysis
- Operating system audit

There are three main components to the Intrusion detection system

- Network Intrusion Detection system (NIDS) performs an analysis for a passing traffic on the entire subnet. Works in a promiscuous mode, and matches the traffic that is passed on the subnets to the library of knows attacks. Once the attack is identified, or abnormal behavior is sensed, the alert can be send to the administrator.
- Network Node Intrusion detection system (NNIDS) performs the analysis of the traffic that is passed from the network to a specific host. The difference between NIDS and NNIDS is that the traffic is monitored on the single host only and not for the entire subnet.

- Host Intrusion Detection System (HIDS) takes a snap shot of existing system files and matches it to the previous snap shot. If the critical system files were modified or deleted, the alert is sent to the administrator to investigate.

The IDS however is not an answer to all Security related problems. The IDS CAN provide the following:

- CAN add a greater degree of integrity to the rest of infrastructure
- CAN trace user activity from point of entry to point of impact
- CAN recognize and report alterations to data
- CAN automate a task of monitoring the Internet searching for the latest attacks
- CAN detect when system is under attack
- CAN detect errors in system configuration
- CAN guide system administrator in the vital step of establishing a policy for computing assets
- CAN make the security management of system possible by non-expert staff

The IDS CAN NOT provide the following:

- CAN NOT compensate for a weak identification and authentication mechanisms
- CAN NOT conduct investigations of attacks without human intervention
- CAN NOT compensate for weaknesses in network protocols
- CAN NOT compensate for problems in the quality or integrity of information the system provides
- CAN NOT analyze all the traffic on a busy network
- CAN NOT always deal with problems involving packet-level attacks
- CAN NOT deal with some of the modern network hardware and features

Generally, Intrusion detection systems fall into three basic categories:

- Signature-based intrusion detection systems: Like computer viruses, intruders have signatures that can be detected using software. Based upon a set of signatures and rules, these detection systems are able to find and log suspicious activities and generate alerts using a pattern matching technique.
- Anomaly-based intrusion detection systems: It is also known as profile-based detection since the systems purpose is to examine the statistical and characteristic behavior. It usually depends on packet anomalies present in protocol header parts. In some cases, these methods produce better results compared to signature-based IDS.
- Target Monitoring: In the third type of intrusion detection systems, a standing check after changes is done. Several objects as file modifications and program-logon are of special interest to this kind of control. The control is done through a crypto algorithm and a crypto-check sum is calculated from the objects of interest. The comparison between new and old check sums is made, and system is compromised, if there is mismatch

Depending upon the network topology, the type of intrusion activity (i.e. internal, external or both), and security policy, IDSs can be positioned at one or more places in the network. For example:

- Between network and Extranet.
- In the DMZ before the Firewall to identify the attacks on servers in DMZ.
- Between the firewall and network, to identify a threat in case of the firewall penetration.
- In the Remote access environment.
- If possible between servers and user community, to identify the attacks from the inside.
- On the intranet, ftp, and database environment.

IV. AI – Machine learning

AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. AI currently encompasses a huge variety of subfields, the central problems (or goals) of AI research include reasoning, knowledge, planning, learning, natural language processing (communication), perception and the ability to move and manipulate objects.

Machine learning evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal. Another example is learning to play a game by playing against an opponent

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one (or multi-label classification) or more of these classes. This is typically tackled in a supervised way.
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand -> this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.
- Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space.

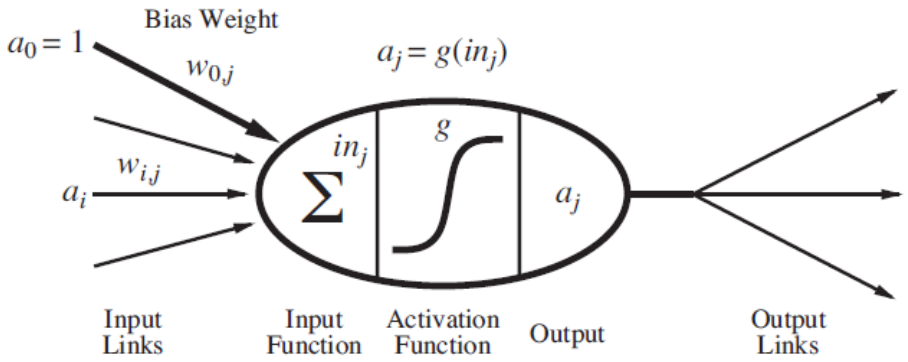
Overtraining is a problem that arises when too many training examples are provided, and the system becomes incapable of useful generalization. This can also occur when there are too many neurons in the network and the capacity for computation exceeds the dimensionality of the input space. During training, care must be taken not to provide too many input examples and different numbers of training examples could produce very different results in the quality and robustness of the network.

V. Neural Network

As mentioned above, artificial neural networks (ANNs) are a family of models inspired by biological neural networks and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown.

Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

Figure below shows a simple mathematical model of the neuron devised by McCulloch and Pitts (1943).



This is a simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

Neural networks are composed of nodes or units connected by directed links. A link from unit i to unit j serves to propagate the activation a_i from i to j. Each link also has a numeric weight $w_{i,j}$ associated with it, which determines the strength and sign of the connection. Just as in linear regression models, each unit has a dummy input $a_0 = 1$ with an associated weight $w_{0,j}$. Each unit j first computes a weight sum of its input:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

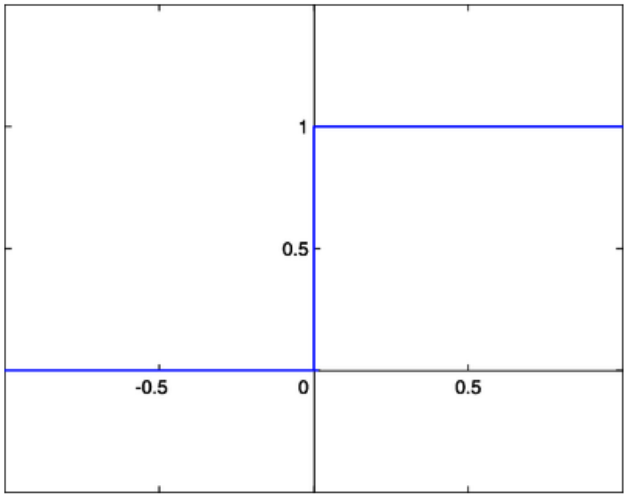
Then it applies an activation function g to this sum to derive the output:

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i)$$

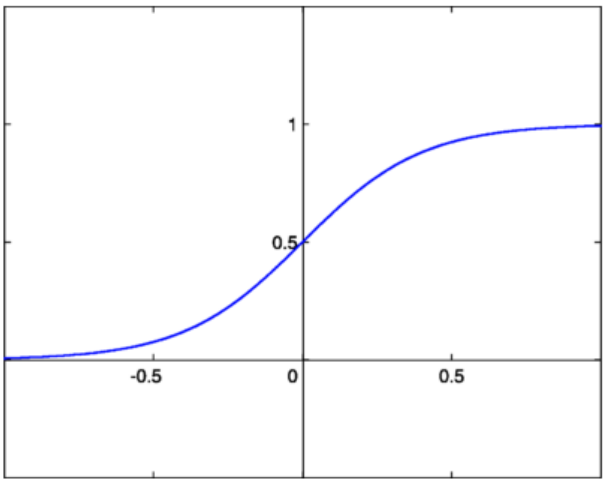
The activation function g is typically either a hard threshold, in which case the unit is called a perceptron, or a logistic function, in which case the term sigmoid perceptron is sometimes used. Both of these nonlinear activation function ensure the important property that the entire network of units can represent a nonlinear function.

Moreover, there are a number of common activation functions in use with neural networks:

- Step Function: is a function is likely used by the original Perceptron. The output is a certain value, A_1, if the input sum is above a certain threshold and A_0 if the input sum is below a certain threshold. The values used by the Perceptron were A_1 = 1 and A_0 = 0.



- Continuous Log-Sigmoid Function: log-sigmoid function, also known as a logistic function, is given by the relationship: $\sigma(t) = \frac{1}{1+e^{-\beta t}}$ where β is a slope parameter. This is called the log-sigmoid because a sigmoid can also be constructed using the hyperbolic tangent function instead of this relation, in which case it would be called a tan-sigmoid. The sigmoid has the property of being similar to the step function, but with the addition of a region of uncertainty. Sigmoid functions in this respect are very similar to the input-output relationships of biological neurons, although not exactly the same. Below is the graph of a sigmoid function.



Having decided on the mathematical model for individual "neurons," the next task is to connect them together to form a network. There are two fundamentally methods to do this.

A feed-forward network has connections only in one direction – that means it forms a directed acrylic graph. Every node receives input from "upstream" nodes and delivers output to "downstream" nodes; there are no loops. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves.

A recurrent network, on the other hand, feeds its outputs back into its own inputs. This means that the activation levels of the network form a dynamical system that may reach a stable state or exhibit oscillations or even chaotic behavior. Moreover, the response of the network to a given input depends on its initial state, which may depend on previous inputs. Hence, recurrent networks can support short-term memory. This makes them more interesting as models of the brain, but also more difficult to understand.

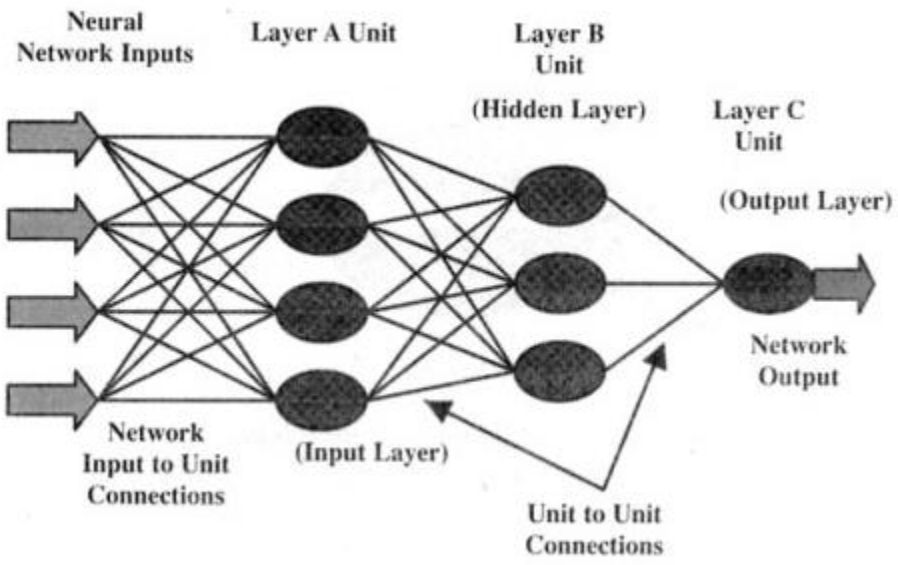
In this report, we will focus only on feed-forward network. Generally, there are two architectures of feed-forward network:

A. Single-layer feed-forward neural networks

A network with all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.

B. Multilayer feed-forward (MLF) neural networks

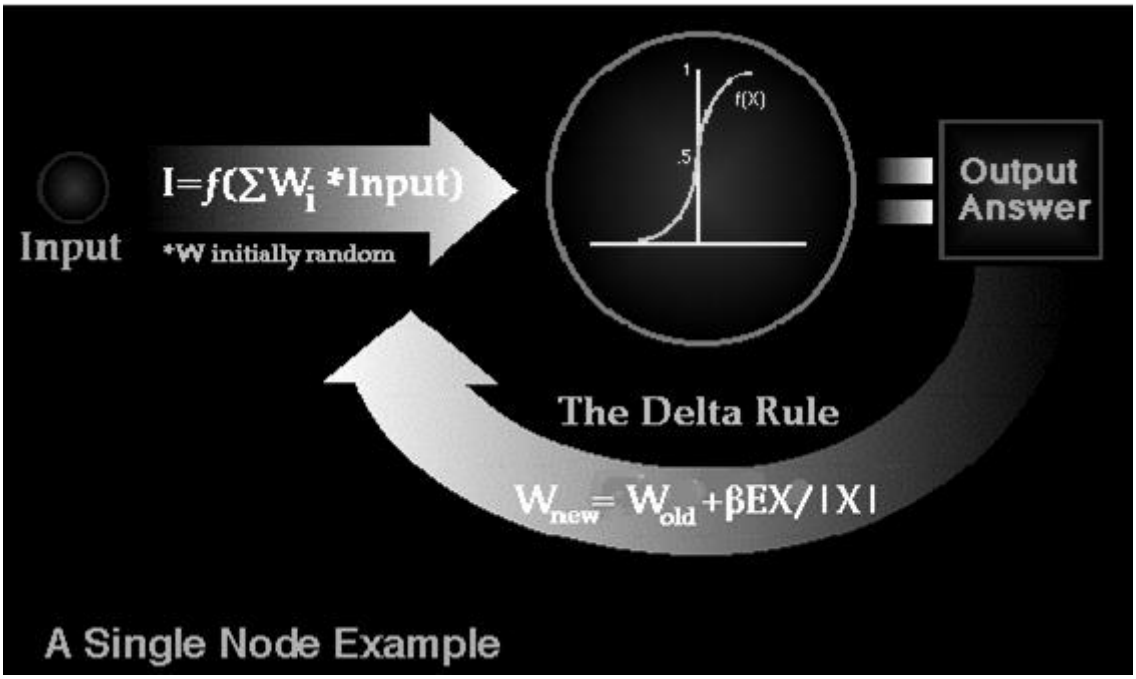
MLF neural networks, trained with a back-propagation learning algorithm, are the most popular neural networks. A MLF neural network consists of neurons that are ordered into layers. The first layer is called the input layer, the last layer is called the output layer and the layers between are hidden layers. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.



Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts; a child learns to recognize dogs from examples of dogs.

Although there are many different kinds of learning rules used by neural networks, this demonstration is concerned only with one; the delta rule. The delta rule is often utilized by the most common class of ANNs called 'backpropagational neural networks' (BPNNs). Backpropagation is an abbreviation for the backwards propagation of error.

With the delta rule, as with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. More graphically, the process looks something like this:

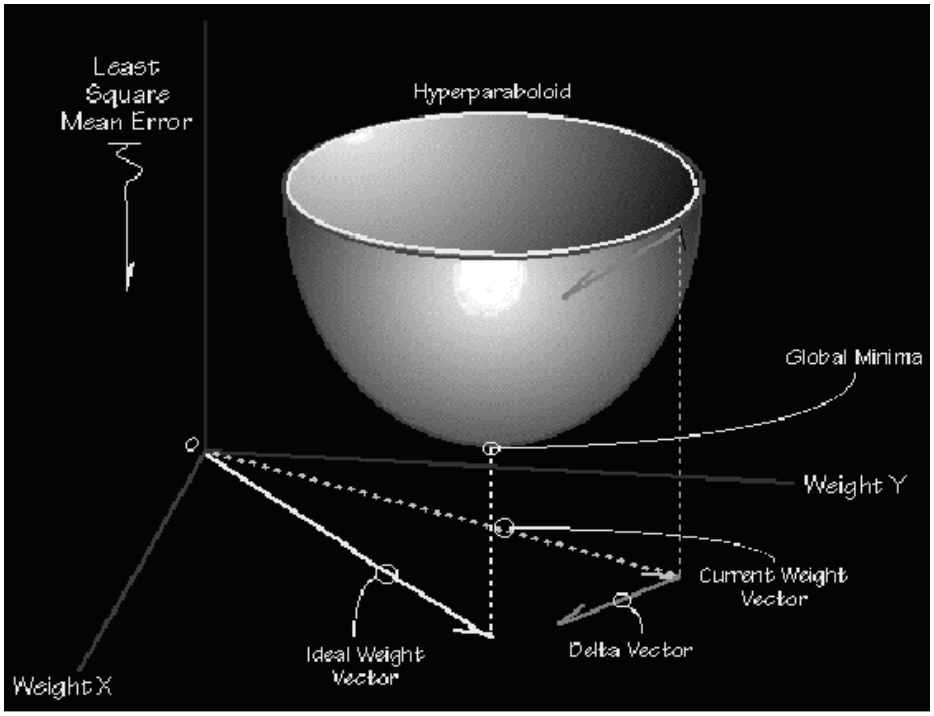


As mentioned above, that within each hidden layer node is a sigmoidal activation function which polarizes network activity and helps it to stabilize.

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\ a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\ \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

Backpropagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. The error surface itself is a hyper paraboloid but is seldom 'smooth' as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minimum' which is not the best overall solution.



Since the nature of the error space cannot be known a prior, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no backpropagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

It is also possible to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "grand mothered" in neural network jargon. In real-world applications this situation is not very useful since one would need a separate “grand mothered” network for each new kind of input.

There are many advantages and limitations to neural network analysis and to discuss this subject properly we would have to look at each individual type of network, which isn't necessary for this general discussion. In reference to backpropagational networks however, there are some specific issues potential users should be aware of:

- Backpropagational neural networks (and many other types of networks) are in a sense the ultimate 'black boxes'. Apart from defining the general architecture of a network and perhaps initially seeding it with a random numbers, the user has no other role than to feed it input and watch it train and await the output. In fact, it has been said that with backpropagation, "you almost don't know what you're doing". Some software freely available software packages do allow the user to sample the networks progress at regular time intervals, but the learning itself progresses on its own. The final product of this activity is a trained network that provides no equations or

coefficients defining a relationship (as in regression) beyond its own internal mathematics. The network is the final equation of the relationship.

- Backpropagational networks also tend to be slower to train than other types of networks and sometimes require thousands of epochs. If run on a truly parallel computer system this issue is not really a problem, but if the BPNN is being simulated on a standard serial machine (i.e. a single SPARC, Mac or PC) training can take some time. This is because the machines CPU must compute the function of each node and connection separately, which can be problematic in very large networks with a large amount of data. However, the speed of most current machines is such that this is typically not much of an issue

ANNs Parameters:

- Number of neurons in the hidden layer: number of neurons in the hidden layer (additional layer to the input and output layers, not connected externally). Data type: Integer Domain: [1,∞). Typical value: 8.
- Learning Rate: training parameter that controls the size of weight and bias changes in learning of the training algorithm. Data type: Real Domain: [0, 1]. Typical value: 0.3
- Momentum: Momentum simply adds a fraction m of the previous weight update to the current one. The momentum parameter is used to prevent the system from converging to a local minimum or saddle point. A high momentum parameter can also help to increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local minima, and can also slow down the training of the system. Real Domain: [0, 1]. Typical value: 0.9
- Training type: 0 = train by epoch, 1 = train by minimum error. Data type: Integer Domain: [0, 1]. Typical value: 1.
- Epoch: determines when training will stop once the number of iterations exceeds epochs. When training by minimum error, this represents the maximum number of iterations. Integer Domain: [1,∞). Typical value: 5000000.
- Minimum Error: minimum mean square error of the epoch. Square root of the sum of squared differences between the network targets and actual outputs divided by number of patterns (only for training by minimum error). Data type: Real Domain: [0, 0.5]. Typical value: 0.01

VI. NSL-KDD Dataset

Since 1999, KDD'99 has been the most wildly used dataset for the evaluation of anomaly detection methods. This dataset is prepared by Stolfo and is built based on the data captured in DARPA'98 IDS evaluation program. DARPA'98 is about 4 gigabytes of compressed raw (binary) tcpdump data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes. The two weeks of test data have around 2 million connection records. KDD training dataset consists of approximately 4,900,000 single connection vectors each of which contains 41 features and is labeled as either normal or an attack, with exactly one specific attack type. The simulated attacks fall in one of the following four categories:

- Denial of Service Attack (DoS): is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine.
- User to Root Attack (U2R): is a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.
- Remote to Local Attack (R2L): occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine.
- Probing Attack: is an attempt to gather information about a network of computers for the apparent purpose of circumventing its security controls.

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data which make the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the signature of known attacks can be sufficient to catch novel variants. The datasets contain a total number of 24 training attack types, with an additional 14 types in the test data only.

KDD'99 features can be classified into three groups:

Basic features: this category encapsulates all the attributes that can be extracted from a TCP/IP connection. Most of these features leading to an implicit delay in detection.

Traffic features: this category includes features that are computed with respect to a window interval and is divided into two groups:

- "Same host" features: examine only the connections in the past 2 seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.
- "Same service" features: examine only the connections in the past 2 seconds that have the same service as the current connection.

Content features: unlike most of the DoS and Probing attacks, the R2L and U2R attacks don't have any intrusion frequent sequential patterns. This is because the DoS and Probing attacks involve many connections to some host(s) in a very short period of time; however the R2L and U2R attacks are embedded in the data portions of the packets, and normally involves only a single connection.

However, there are some inherent problems in the KDDCUP'99 dataset which highly affects the performance of evaluated systems, and results in a very poor evaluation of anomaly detection approaches.

The first important deficiency in the KDD dataset is the huge number of redundant records. Analyzing KDD train and test sets, researchers found that about 78% and 75% of the records are duplicated in the train and test set, respectively. This large amount of redundant records in the train set will cause learning algorithms to be biased towards the more frequent records, and thus prevent it from learning infrequent records which are usually more harmful to networks such as U2R attacks. The existence of these repeated records in the test set, on the other hand, will cause the evaluation results to be biased by the methods which have better detection rates on the frequent records.

The second crucial problem in KDD dataset is that random parts of the KDD train set are used as test sets. As a result, learned machine achieve about 98% classification rate applying very simple machine learning methods. Even applying the KDD test set will result in having a minimum classification rate of 86%, which makes the comparison of IDSs quite difficult since they all vary in the range of 86% to 100%.

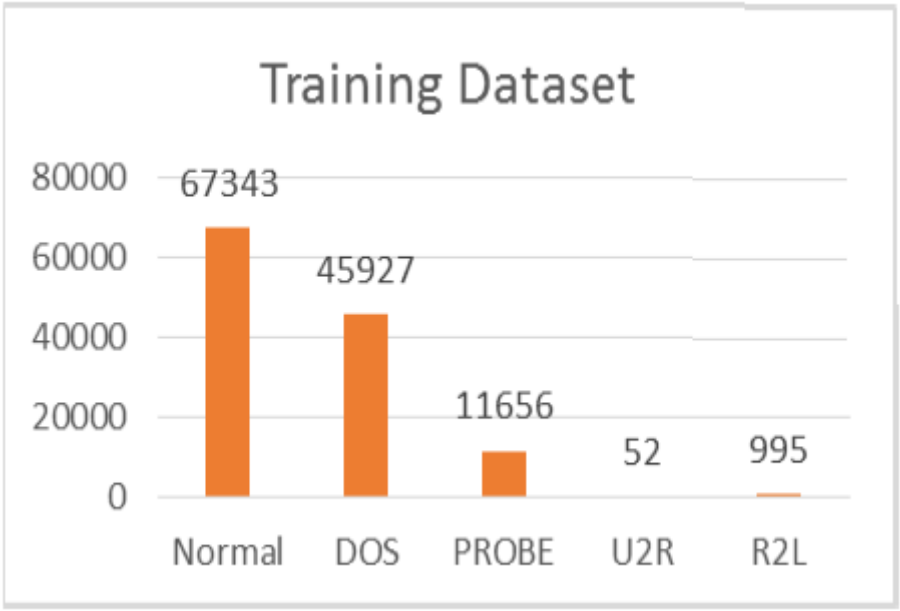
In this report, we used a new version of KDD dataset: NSL-KDD dataset, which provided a solution to solve the two mentioned issues, resulting in new train and test sets which consist of selected records of the complete KDD dataset. NSL-KDD dataset does not suffer from any of the mentioned problems. Furthermore, the number of records in the train and test sets are reasonable. This advantage makes it affordable to run the experiments on the complete set without the need to randomly select a small portion.

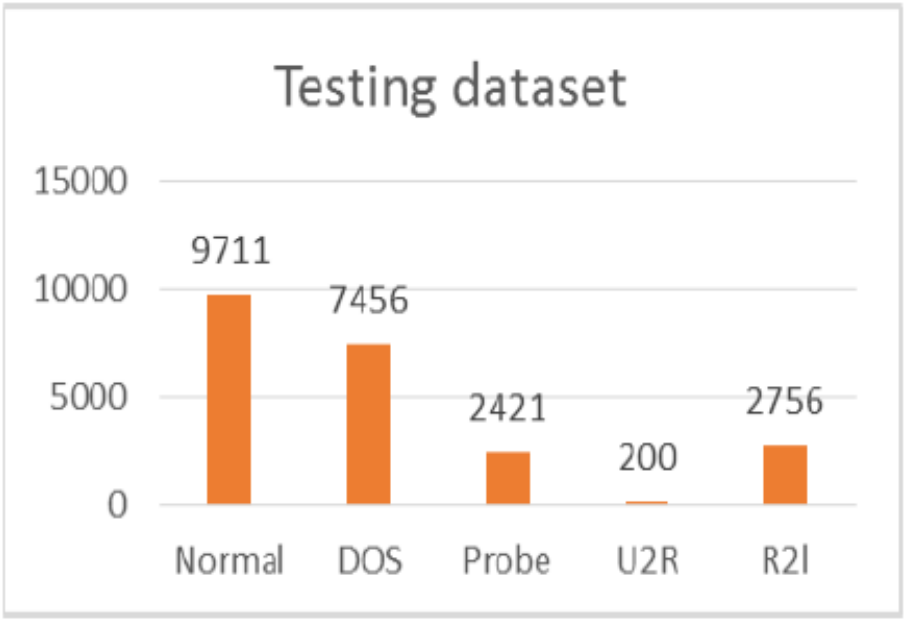
The advantage of NSL KDD dataset are

- No redundant records in the train set, so the classifier will not produce any biased result
- No duplicate record in the test set which have better reduction rates.
- The number of selected records from each difficult level group is inversely proportional to the percentage of records in the original KDD data set.

The training dataset is made up of 21 different attacks out of the 37 present in the test dataset. The known attack types are those present in the training dataset while the novel attacks are the additional attacks in the test dataset i.e. not available in the training datasets. The attack types are grouped into four categories: DoS, Probe, U2R and R2L.

Attacks in Dataset	Attack Type
DOS	Back, Land, Neptune, Pod, Smurf, Teardrop, Mailbomb, Processtable, Udpstorm, Apache2,Worm
Probe	Satan, IPsweep, Nmap, Portsweep, Mscan, Saint
R2L	Guess_password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps





NSL-KDD dataset input field's name:

duration	continuous
protocol_type	symbolic
service	symbolic
flag	symbolic
src_bytes	continuous
dst_bytes	continuous
land	continuous
wrong_fragment	continuous
urgent	continuous
hot	continuous
num_failed_logins	continuous
logged_in	continuous
num_compromised	continuous
root_shell	continuous
su_attempted	continuous
num_root	continuous
num_file_creations	continuous
num_shells	continuous
num_access_files	continuous
num_outbound_cmds	continuous
is_host_login	continuous
is_guest_login	continuous
count	continuous
srv_count	continuous
serror_rate	continuous
srv_serror_rate	continuous
rerror_rate	continuous
srv_rerror_rate	continuous
same_srv_rate	continuous
diff_srv_rate	continuous
srv_diff_host_rate	continuous
dst_host_count	continuous
dst_host_srv_count	continuous
dst_host_same_srv_rate	continuous
dst_host_diff_srv_rate	continuous
dst_host_same_src_port_rate	continuous
dst_host_srv_diff_host_rate	continuous
dst_host_serror_rate	continuous
dst_host_srv_serror_rate	continuous
dst_host_rerror_rate	continuous
dst_host_srv_rerror_rate	continuous

VII. Suggested IDS Architecture

A. ANN for intrusion detection system

An amount of research has been conducted on the application of neural networks to detecting computer intrusions. Artificial neural networks offer the potential to resolve a number of the problems encountered by the other current approaches to intrusion detection. Artificial neural networks have been proposed as alternatives to the statistical analysis component of anomaly detection systems.

Statistical Analysis involves statistical comparison of current events to a predetermined set of baseline criteria. The technique is most often employed in the detection of deviations from typical behavior and determination of the similarly of events to those which are indicative of an attack.

Neural networks were specifically proposed to identify the typical characteristics of system users and identify statistically significant variations from the user's established behavior.

Artificial neural networks have also been proposed for use in the detection of computer viruses.

B. Application of Neural Networks in Intrusion Detection

While there is an increasing need for a system capable of accurately identifying instances of intrusion on a network there is currently no applied alternative to rule-based intrusion detection systems. This method has been demonstrated to be relatively effective if the exact characteristics of the attack are known.

However, network intrusions are constantly changing because of individual approaches taken by the attackers and regular changes in the software and hardware of the targeted systems. Because of the infinite variety of attacks and attackers even a dedicated effort to constantly update the rule-base of an expert system can never hope to accurately identify the variety of intrusions.

The constantly changing nature of network attacks requires a flexible defensive system that is capable of analyzing the enormous amount of network traffic in a manner which is less structured than rule-based systems. A neural network-based intrusion detection system could potentially address many of the problems that are found in rule-based systems.

C. Advantages of Neural Network-based Intrusion Detection Systems

The first advantage in the utilization of a neural network in the detection of instances of intrusion would be the flexibility that the network would provide. A neural network would be capable of analyzing the data from the network, even if the data is incomplete or distorted. Similarly, the network would possess the ability to conduct an analysis with data in a non-linear fashion. Both of these characteristics is important in a networked environment where the information which is received is subject to the random failings of the system. Further, because some attacks may be conducted against the network in a coordinated assault by multiple attackers, the ability to process data from a number of sources in a nonlinear fashion is especially important.

The inherent speed of neural networks is another benefit of this approach. Because the protection of computing resources requires the timely identification of attacks, the processing speed of the neural network could enable intrusion responses to be conducted before irreparable damage occurs to the system.

Because the output of a neural network is expressed in the form of a probability the neural network provides a predictive capability to the detection of instances of intrusion. A neural network-based intrusion detection system would identify the probability that a particular event, or series of events, was indicative of an attack against the system. As the neural network gains experience it will improve its ability to determine where these events are likely to occur in the attack process. This information could then be used to generate a series of events that should occur if this is in fact an intrusion attempt. By tracking the subsequent occurrence of these events the system would be capable of improving the analysis of the events and possibly conducting defensive measures before the attack is successful.

However, the most important advantage of neural networks in intrusion detection is the ability of the neural network to "learn" the characteristics of intrusion attacks and identify instances that are unlike any which have been observed before

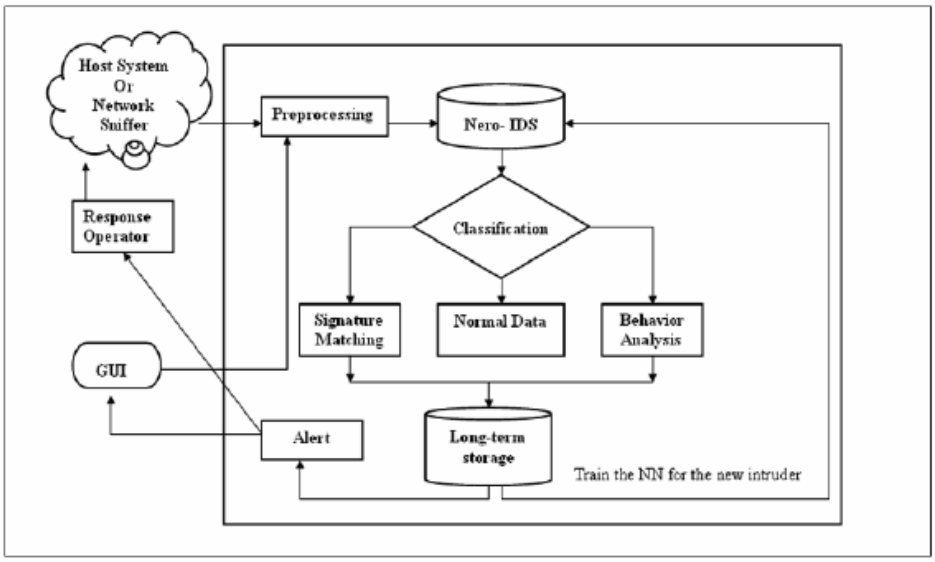
by the network. A neural network might be trained to recognize known suspicious events with a high degree of accuracy. While this would be a very valuable ability, since attackers often emulate the "successes" of others, the network would also gain the ability to apply this knowledge to identify instances of attacks which did not match the exact characteristics of previous intrusions. The probability of an attack against the system may be estimated and a potential threat flagged whenever the probability exceeds a specified threshold.

D. Disadvantages of Neural Network-based Misuse Detection Systems

There appear to be two primary reasons why neural networks have not been applied to the problem of misuse detection in the past. The first reason relates to the training requirements of the neural network. Because the ability of the artificial neural network to identify indications of an intrusion is completely dependent on the accurate training of the system, the training data and the training methods that are used are critical. The training routine requires a very large amount of data to ensure that the results are statistically accurate. The training of a neural network for misuse detection purposes may require thousands of individual attacks sequences, and this quantity of sensitive information is difficult to obtain.

However, as mentioned above, the most significant disadvantage of applying neural networks to intrusion detection is the "black box" nature of the neural network. Unlike expert systems which have hard-coded rules for the analysis of events, neural networks adapt their analysis of data in response to the training which is conducted on the network. The connection weights and transfer functions of the various network nodes are usually frozen after the network has achieved an acceptable level of success in the identification of events. While the network analysis is achieving a sufficient probability of success, the basis for this level of accuracy is not often known.

E. Proposed Architecture



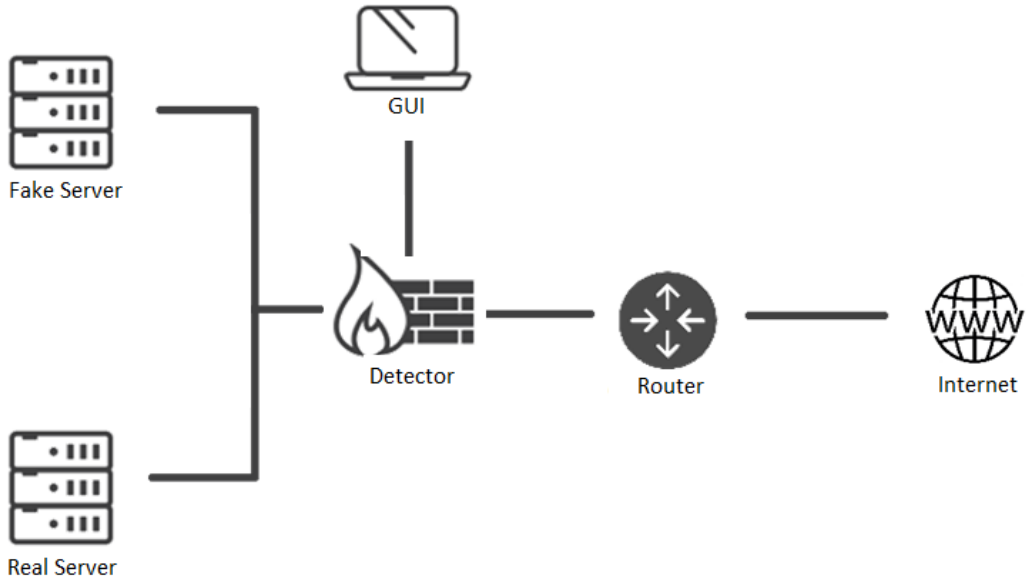
1. Design an interface for packet sniffer
2. Three levels of preprocessing of the data were conducted to prepare the data for use in the training and testing of the neural network:
    a. Select data elements from the event record from available set
    b. Convert data elements to a standardized numeric representation. The process involved the creation of relational tables for each of the data types and assigning sequential numbers to each unique type of element
    c. Convert the results into an ASCII comma delimited format that could be used by the neural network
3. Build a MLP neural network for classifying Network traffic
4. Build a signature-based IDS (Snort or Suricata)
5. Build a honeypot to do behavior analysis
6. Build a database to store result and new training sets for MLP neural networks
7. Build a GUI for better user experience.

VIII. HoneyBear

A. HoneyBear Framework

We realized that above design is not flexible enough. Therefore, from above researches, we are developing HoneyBear framework, which is a combination of honeypot and IDS, also has modularity in design. Packet flows are monitored from routing step in router to detector modules which perform behavior analysis on each stream of packets. Results of analytical process are then used for directing packet flows to appropriate path.

Below is the structure of HoneyBear framework:



HoneyBear framework will include:

- Hidden level:
  - Packet Sniffer: intercept and log packets that passes over router. We will try to collect all packet information in here.
  - Detector: is heart of the system, this is not a usual detector which perform detective process, instead, it is a combination of sub detectors modules, which have been already developed or can be designed by users. Main function of detector is not only to manage, collect analysis results from sub detectors but also to control packet flows into each sub detector.
  - Redirector: After having analysis results, this part has responsibility for direct packet flows in internal network. Now, network structure has 2 nodes: Fake and Real.
  - Logger: log outgoing analyzed packets for further study.
- Interact level:
  - Web GUI: user interface used for displays relevant information including alerts, log, etc.
  - Configuration: configure parameters, priorities, manage sub detector modules, etc.

B. Packet Label

Incoming packets are logged by Packet Sniffer, then analyzed by sub detectors modules in Detector. After that, packets would be labelled as one of these three states:

1) Normal
2) Critical
3) Unknown

With each state, framework will record and mark packets based on their IP address, Source Port and Destination Port. Redirector then directs packet flows to suitable path in internal network. There are 2 nodes right now, so this is what will happen:

- Normal packets will be directed to Real node
- Unknown and critical packets will be directed to Fake node. Here, they will be analyzed by behavior monitoring, which identify attack signatures. These signatures are used by sub detectors later.

C. Sub Detector Modules

HoneyBear framework has modularity in design, which means each part of HoneyBear will work separately as a module, but they have a unified management. Detector is the heart of the system. As mentioned above, function of detector is not only to manage, collect analysis results from sub detectors but also to control packet flows into each sub detector.

Each sub detector module will have packet input, separated database and signatures. Each of them will be responsible for a different type of attacks. For example:

- SQL Injection Detector module.
- DDoS Detector module.
- Shellcode Detector module.

Thus, they should be independently developed and will be integrated to framework when they are ready.

D. Machine Learning in Sub Detector Module

The most important part that our framework inherited from above research is machine learning. As mentioned above, compared to signature-based systems, ANN has much more advantages such as:

- A neural network is more flexible.
- A neural network has better speed in detection.
- A neural network provides a predictive capability to the detection of instances of intrusion.
- A neural network has ability to learn the characteristics of intrusion attacks.

Each sub detector module will be a neural network which is specifically designed for a different type of attacks. Some elements should be discussed in neural network design are:

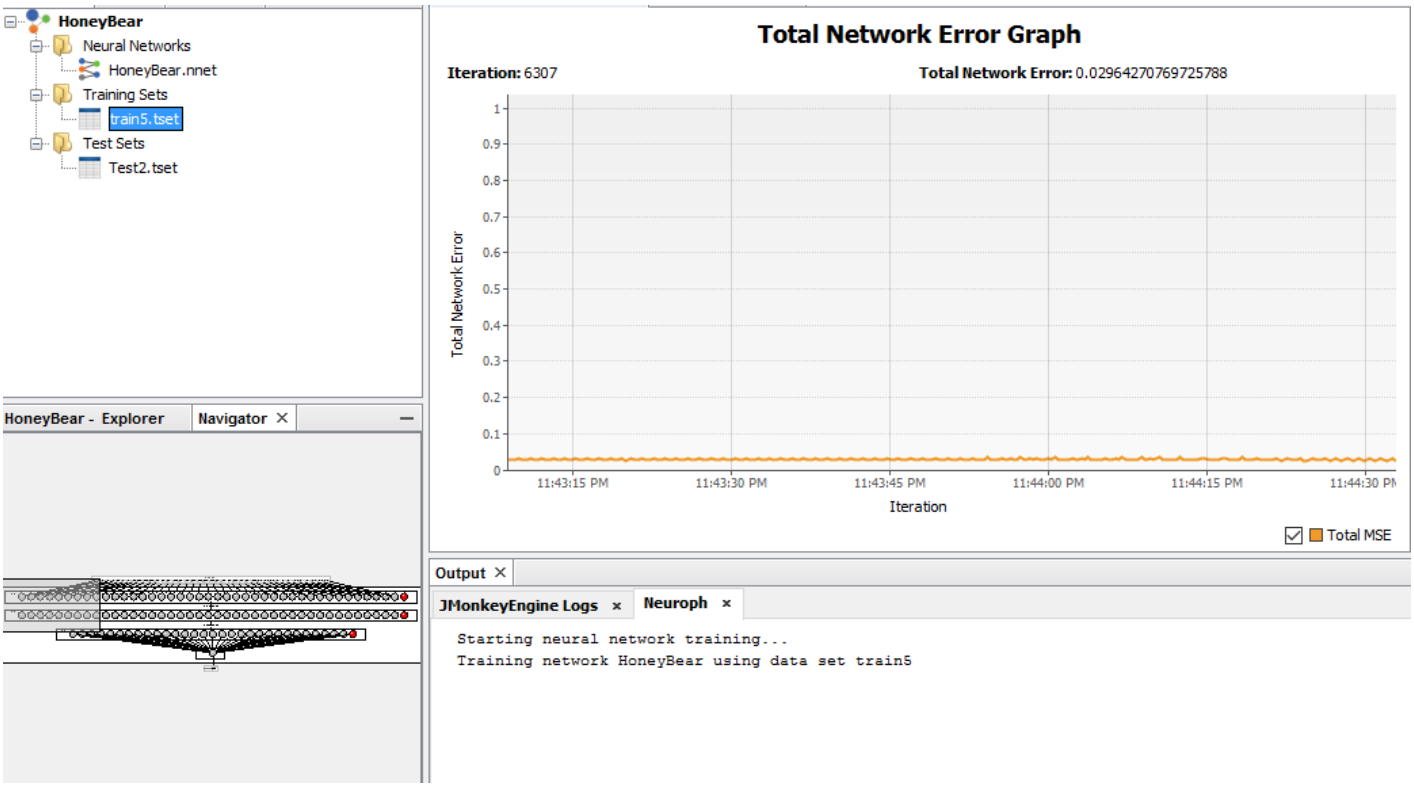
- Develop a parser (to preprocess) collected information to input data.
- Numbers of Input nodes.
- Number of hidden layers.
- What parameter should be used for input?
- Create train set, test set, etc.

Note that the most important feature of an ANN is ability to learn. However, with only created train set and test set, this neural network cannot detect new methods in this type of attacks. Therefore, after analytical process, packets will be directed to a Fake server, which perform further analysis including behavior analysis. As a result, there should be new signatures which can be converted to a train set for ANN, thus help ANN to be more intelligent.

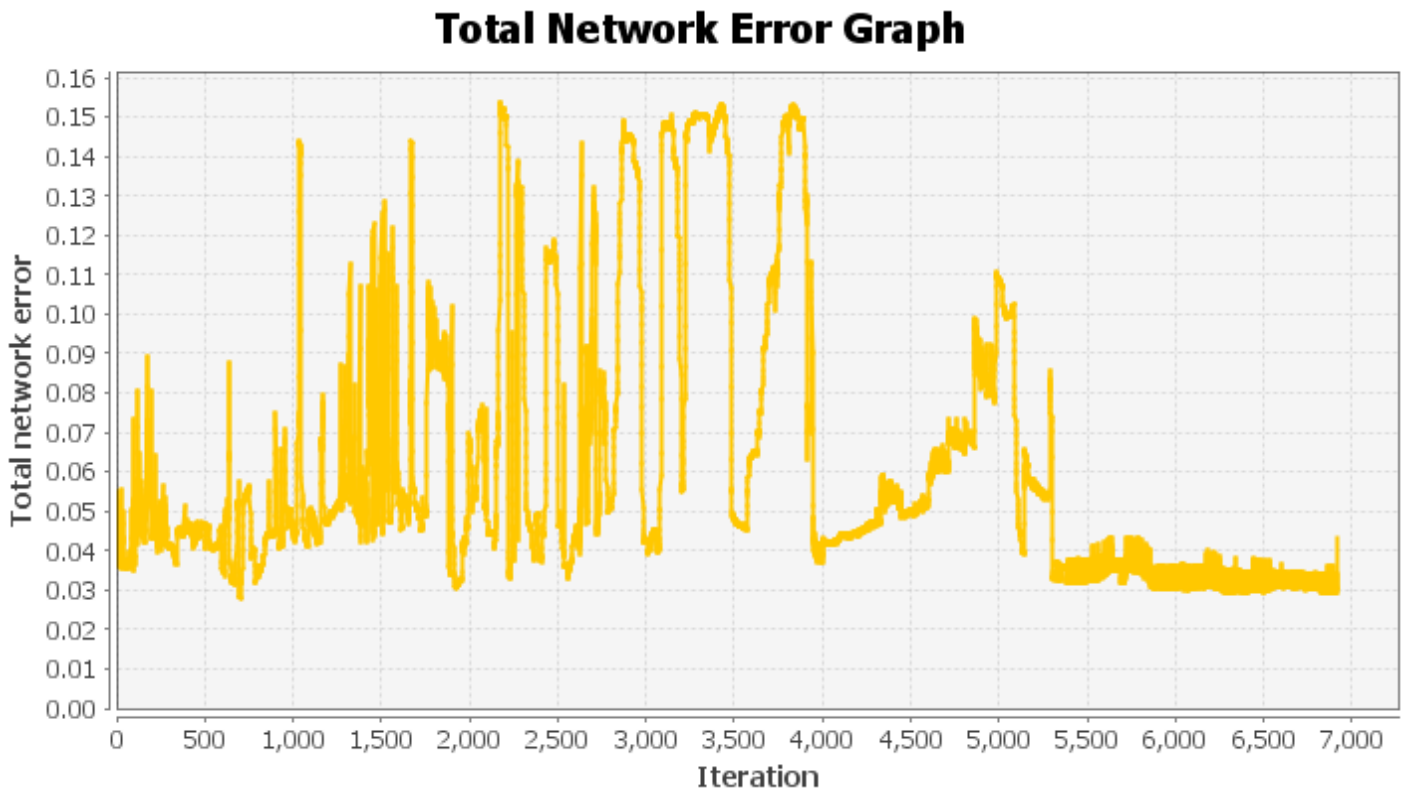
IX. Results

Firstly, we simulated an ANN using NSL-KDD dataset for training and testing. This helps to strengthen our studied theory, also verify our neural network design which includes 41 input nodes, 2 hidden layers (41 nodes – 31 nodes) and one output node.

Below image is training process.



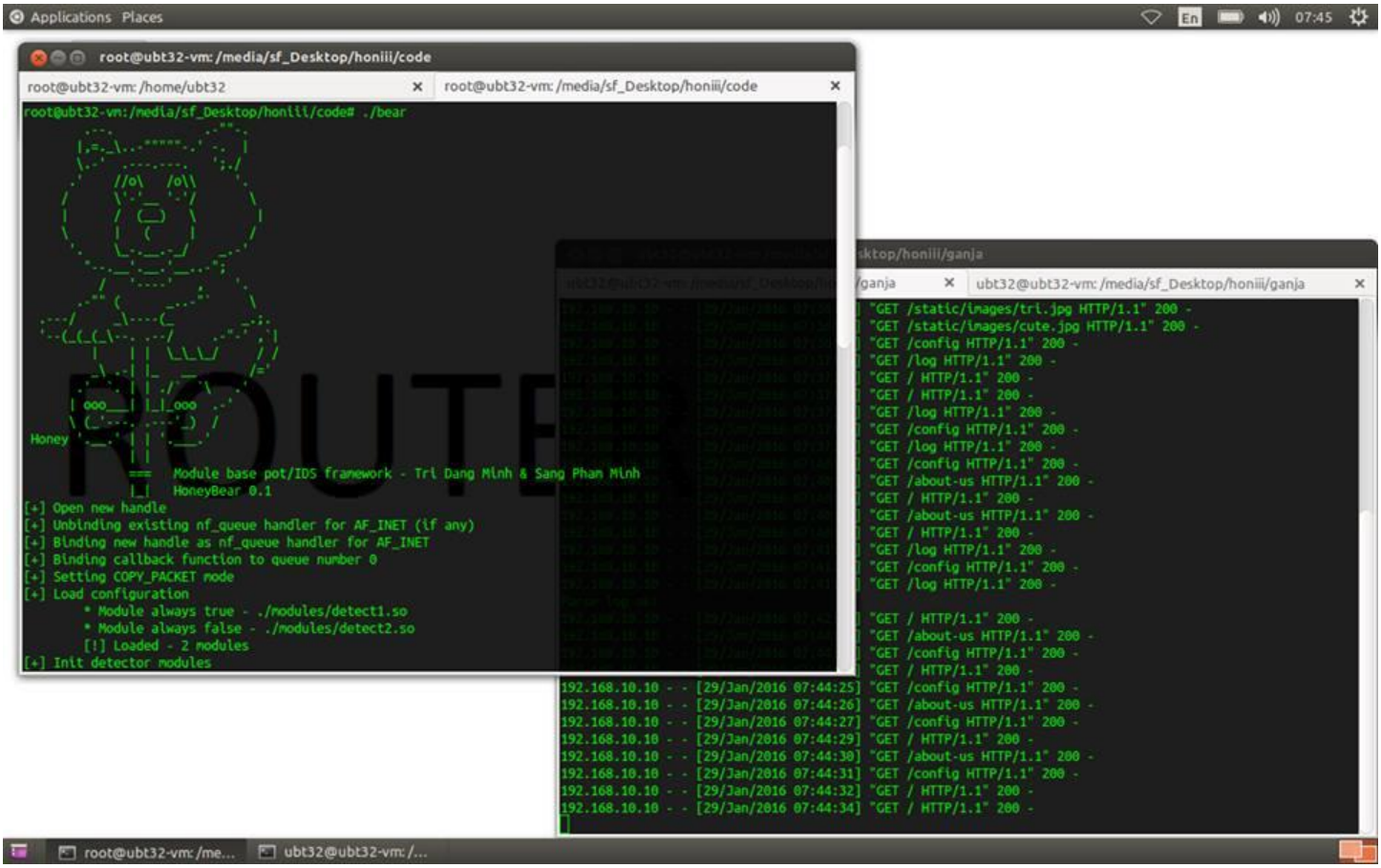
After about 7000 iterations, error of network remains stable.

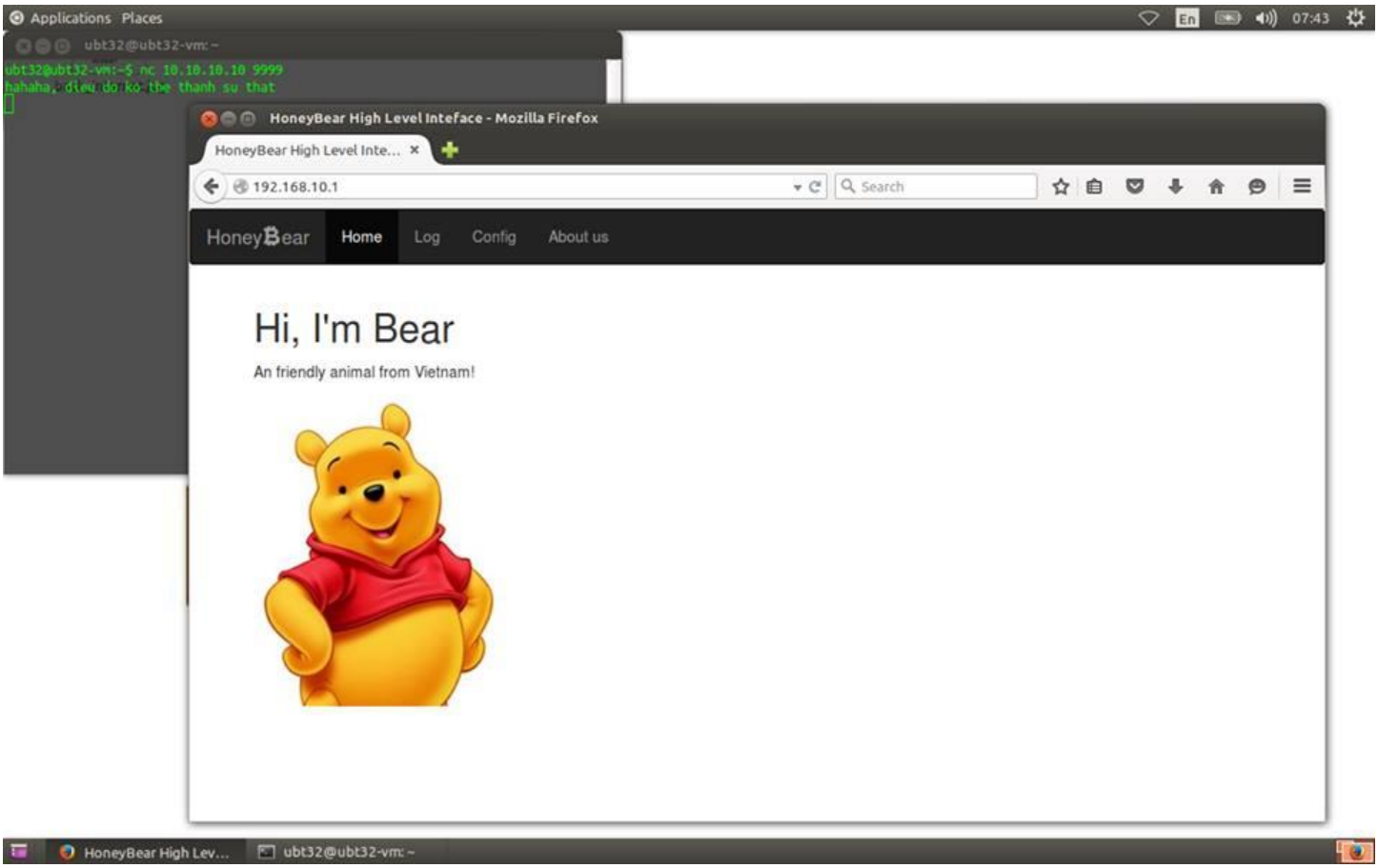


Then, we use test set to verify our neural network. However, this test set includes many new attacks, therefore the output of out trained neural network has lots of false positive (about 50.51%).

```
Input: 0; 1; 22; 9; 306; 2,239; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 6; 6; 0; 0; 0; 0; 1; 0; 0; 255; 233; 0.91; 0.02; 0; 0; 0; 0; 0.02; 0;
Input: 0; 1; 57; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 136; 16; 0; 0; 1; 1; 0.12; 0.06; 0; 255; 16; 0.06; 0.06; 0; 0; 0; 0; 1; 1; O
Input: 0; 1; 43; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 1; 1; 1; 0; 0; 255; 15; 0.06; 0.01; 0; 0; 0; 0; 0.05; 0.87; Outp
Input: 282; 1; 17; 9; 160; 599; 0; 0; 0; 2; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 255; 104; 0.41; 0.03; 0; 0; 0; 0; 0; 0; Ou
Input: 0; 2; 48; 9; 45; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 2; 0; 0; 0; 0; 1; 0; 0; 255; 254; 1; 0.01; 0.01; 0; 0; 0; 0; 0; Output: 0
Input: 280; 1; 18; 9; 283,618; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 2; 0; 0; 0; 0; 1; 0; 0; 148; 43; 0.19; 0.04; 0.19; 0.05; 0; 0; 0;
Input: 0; 1; 22; 9; 192; 1,490; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 0; 0; 255; 229; 0.9; 0.02; 0; 0; 0; 0; 0.06; 0; O
Input: 0; 1; 22; 5; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 49; 18; 1; 1; 0; 0; 0.37; 0.06; 0; 255; 63; 0.25; 0.02; 0.01; 0; 0.96; 1; 0;
Input: 0; 1; 43; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 1; 1; 1; 0; 0; 255; 12; 0.05; 0.02; 0; 0; 0; 0; 0.06; 1; Output:
Input: 0; 1; 22; 5; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 48; 48; 1; 1; 0; 0; 1; 0; 0; 255; 255; 1; 0; 0; 0; 0.26; 0.26; 0.7; 0.7; Outp
Input: 0; 2; 10; 9; 46; 78; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 158; 158; 0; 0; 0; 0; 1; 0; 0; 255; 255; 1; 0; 0.01; 0; 0; 0; 0; 0; Outpu
Input: 0; 2; 10; 9; 45; 45; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 112; 112; 0; 0; 0; 0; 1; 0; 0; 255; 254; 1; 0.01; 0; 0; 0; 0; 0; 0; Outpu
Input: 11,675; 1; 59; 9; 103; 2,504; 0; 0; 0; 0; 0; 1; 4; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 0; 0; 255; 1; 0; 0.06; 0; 0; 0.99; 0; 0; 0;
Input: 282; 1; 17; 9; 160; 597; 0; 0; 0; 2; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 255; 6; 0.02; 0.3; 0; 0; 0.28; 0; 0.3; 0; O
Input: 0; 3; 12; 9; 20; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 65; 0; 0; 0; 0; 1; 0; 1; 3; 60; 1; 0; 1; 0.27; 0; 0; 0; 0; Output: 0.0248
Input: 0; 1; 18; 9; 12; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 0; 0; 167; 31; 0.19; 0.04; 0.19; 0; 0; 0; 0; 0; Output
Input: 0; 2; 48; 9; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 72; 8; 0; 0; 0; 0; 0.11; 0.85; 0; 255; 8; 0.03; 0.74; 0.99; 0; 0; 0; 0.01; 0;
Input: 2,088; 1; 22; 11; 72,564; 0; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 13; 13; 0.08; 0.08; 0.92; 0.92; 1; 0; 0; 255; 237; 0.93; 0.01;
Input: 0; 1; 26; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 18; 0; 0; 1; 1; 0.5; 1; 1; 58; 59; 0.28; 0.09; 0.02; 0.03; 0.03; 0.02; 0.8
Input: 812; 1; 22; 11; 57,184; 0; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 35; 35; 0.03; 0.03; 0.97; 0.97; 1; 0; 0; 255; 253; 0.99; 0.01; 0;
Input: 0; 1; 22; 6; 54,540; 8,314; 0; 0; 0; 2; 0; 1; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 3; 3; 0.33; 0.33; 0; 0; 1; 0; 0; 255; 255; 1; 0; 0; 0; 0; 0; 0.05;
Input: 0; 1; 22; 9; 54,540; 8,314; 0; 0; 0; 2; 0; 1; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 5; 10; 0; 0; 0; 0; 1; 0; 0.2; 255; 255; 1; 0; 0; 0; 0; 0; 0.07; 0.0
Input: 0; 2; 48; 9; 54; 50; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 511; 511; 0; 0; 0; 0; 1; 0; 0; 255; 255; 1; 0; 0.88; 0; 0; 0; 0; 0; Output
Total Mean Square Error: 0.5051507234449208
```

Below images show low-level interaction and high-level interaction of our framework respectively:





X. Future Work

Our sub detector modules are in theory right now. Therefore, in the future, we planned to develop many sub detector modules for different kind of attacks. We also planned to apply a new training algorithm for neural network called Extreme Learning Machines.

The analytical module is also not presented. We need to focus on developing it, because it is the most important part which help the neural network to be more intelligent.

XI. References

[1] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani. A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. Retrieved December 24, 2015.

[2] S. Revathi, A. Malathi. A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. International Journal of Engineering Research & Technology (IJERT). Retrieved December 24, 2015.

[3] James Cannady. Artificial Neural Networks for Misuse Detection. Retrieved on: 19 December 2015.

[4] Mehdi Moradi, Mohammad Zulkernine. A Neural Network Based System for Intrusion Detection and Classification of Attacks. IEEE International Conference on Advances in Intelligent Systems – 2004. Retrieved on: 19 December 2015.