ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ:

«Ανάπτυξη Intrusion Detection System με τη χρήση Μπεϋζιανών Δικτύων»

“Intrusion Detection System development via Bayesian Neural Networks”

ΝΙΚΟΛΑΟΣ-ΟΡΕΣΤΗΣ ΓΑΒΡΙΗΛΙΔΗΣ

Επιβλέπων καθηγητής – Supervisor professor
ΚΑΤΣΑΡΟΣ ΔΗΜΗΤΡΙΟΣ – KATSAROS DIMITRIOS
Συνεπιβλέποντες καθηγητές – Assistant Supervisor professors
ΤΣΟΥΚΑΛΑΣ ΕΛΕΥΘΕΡΙΟΣ – TSOUKALAS ELEFTHERIOS
ΚΟΡΑΚΗΣ ΑΘΑΝΑΣΙΟΣ – KORAKIS ATHANASIOS

ΒΟΛΟΣ 2020

Στην οικογένεια και τους φίλους μου

ΕΥΧΑΡΙΣΤΙΕΣ

Με την περάτωση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες της διπλωματικής εργασίας κ. Κατσαρό Δημήτριο, κ. Τσουκαλά Ελευθέριο και κ. Κοράκη Αθανάσιο για την εμπιστοσύνη που επέδειξαν στο πρόσωπό μου, την άριστη συνεργασία, την συνεχή καθοδήγηση και τις ουσιώδεις υποδείξεις και παρεμβάσεις τους. Επιπλέον, θα ήθελα να εκφράσω την εκτίμησή μου σε όλους τους συναδέλφους με τους οποίους συνεργάστηκα στη διάρκεια αυτών των σπουδών. Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένεια και τους φίλους μου για την αμέριστη υποστήριξη, την αγάπη και την ανεκτίμητη βοήθεια που μου παρείχαν τόσο κατά την διάρκεια των σπουδών μου όσο και κατά την εκπόνηση της διπλωματικής εργασίας.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation and sincere gratitude to the leading supervisor professor Katsaros Dimitrios and the assistant supervisors of the committee professors Tsoukalas Eleftherios and Korakis Athanasios for their trust and useful guidance without which this dissertation would not have been realized. I would also like to express my thanks and gratitude to all my fellow colleagues with whom I collaborated during my latest academic endeavor. Finally, my deepest love and appreciation is extended to my family and friends for their unending support and confidence in me which helped me achieve my goals.

ΠΕΡΙΛΗΨΗ

Στην παρούσα εργασία εξετάζεται η δημιουργία ενός Intrusion Detection System, με τη χρήση Μπεϋζιανών νευρωνικών δικτύων. Αρχικά παρουσιάζονται ορισμένες από τις βασικές θεωρίες σχετικά με τα Intrusion Detection Systems και τη Μπεϋζιανή στατιστική. Το δεύτερο κεφάλαιο αναφέρεται σε σχετικές με το θέμα εργασίες. Στη συνέχεια ακολουθεί το κυρίως μέρος της εργασίας αυτής το οποίο περιγράφει τη δημιουργία, την εκπαίδευση και τον έλεγχο της μεθόδου που χρησιμοποιήθηκε. Στο τέταρτο κεφάλαιο αναλύεται η αρχιτεκτονική και τα αποτελέσματα που προέκυψαν σε σύγκριση και με άλλες μεθόδους. Τέλος, στο πέμπτο κεφάλαιο, γίνεται μια αποτίμηση των αποτελεσμάτων και παρουσιάζονται προτάσεις για τις πιθανές προοπτικές εξέλιξης αυτής της μεθόδου.

ABSTRACT

In this thesis an Intrusion Detection System construction using Bayesian neural networks is presented. Initially some of the basic theories of Intrusion Detection Systems and Bayesian statistics are discussed. In the second chapter related work to this thesis is provided. The construction, the training and testing of the architectures used in this thesis are presented in chapter three. Evaluation of the architectures used and the results they provided in relation also to other approaches, are displayed in chapter four. In conclusion, an overall assessment is attempted along with proposals for further research on this subject.

Table of Contents

1. Introduction

The rapid development of new technologies in the field of informatics has made clear that the use of computers and networks is invaluable. The abundant use of this technologies has led to an increase of data flow in the network which consequently increased the number and the type of cyber-attacks. With such a fast and everchanging environment cyber-security requires complementary techniques and technologies in order to be able to defend successfully against newly formed attacks [1][2][3]. Thus, the need and use of Intrusion Detection Systems (IDS) proves to be irrefutable and valuable.

1.1 Intrusion Detection System

An Intrusion Detection System is a mechanism designed to monitor a network (NIDS) or a system-host (HIDS), detect anomalies from malicious activities and report the incident to the administrator [4]. The intrusion detection can be performed in various ways. Signature based detection is used to identify already known threats by comparing signatures or patterns between new data and rules that are pre-installed in the intrusion detection device. The intrusion detection can be also achieved via anomaly detection. Every packet that deviates from the normal traffic pattern is classified as an intrusion. This method is more effective in unknown and new threats. Signature based detection produces high detection accuracy on known attacks but performs weakly on unknown ones. On the other hand, anomaly detection suffers from high false-positives but it is widely used due to its ability to recognize new and unknown threats [5]. Intrusion Detection Systems may also rely on

reputation-based detection. Misbehavior leads in reputation detraction. A packet is categorized as normal traffic or threat according to the reputation of their source [6].

In the development of anomaly detection-based Intrusion Detection Systems, many machine learning algorithms such as K-Means Clustering [7], Support Vector Machines (SVM) [8], Bayes Network [9], Self-Organized Maps (SOM) [10], Random Forest Classifier [11], Neural Networks [12], etc. have been used. Machine learning algorithms can be classified into four categories. Supervised machine learning, unsupervised machine learning, semi-supervised machine learning and reinforcement machine learning algorithms.

Supervised machine learning is used for classification and regression problem solving. A function which maps inputs to outputs will be inferred from labeled training data which are forced into the algorithm. When the algorithm achieves a predetermined performance, training stops. New and unknown data for the model can be predicted with a certain accuracy [13].

Unsupervised machine learning also known as self-organization, on the other hand is used for clustering, extraction of association rules, density estimation, or to project data from a high-dimensional space down to a number we can visualize and understand. Data forced into the algorithm are unlabeled and the model works on its own to discover information and patterns [14].

Semi-supervised learning combines labeled and unlabeled training data. However unlabeled data are much more than the labeled ones. It is a combination of supervised and unsupervised learning. It is mostly used when labelled data are very difficult to find or costly to create. Adding unlabeled data and converting a supervised to semi-supervised model, sometimes may lead to accuracy improvement, such as in Alexa’s case [15].

Reinforcement learning aims to lead the software agent to solve a complex game-like situation by making a sequence of decisions. The game provided to the machine has specific rules of obtaining and losing points. By trial and error, agents end up with sophisticated solutions that maximize the notion of aggregated reward [16].

1.2 Artificial Neural Networks

As mentioned above, one of the methods often used to develop intrusion detection systems are artificial neural networks. These networks simulate the processes of the brain. An artificial neuron is a mathematical function conceived as a model of a biological neuron. Biological neural networks are formed by billions of neurons that are interconnected through synapses. Electrochemical signals are responsible for the information transmitted over the network [17]. In contrast, a large artificial neural network may consist of hundreds or thousands of processor units, resulting to a decrease of their overall interaction and emergent behavior [18]. The definition of artificial neural networks as given by Dr. Robert Hecht-Nielsen is:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." [19].

There are two fundamental types of artificial neurons, those are Perceptron and Sigmoid neurons. The first artificial neuron, classical perceptron model, was proposed by Frank Rosenblatt, an American psychologist, in 1958. Marvin Minsky and Seymour Papert further polished the precedent model in 1969, resulting to what is referred to as the perceptron model.

Let $\Phi = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$ be a family of predicates. Then ψ is linear with respect to Φ if a set of numbers $\{a_{\varphi 1}, \alpha_{\varphi 2}, \dots, \alpha_{\varphi n}\}$ and a number θ exist, such that ψ(X) = 1 if and only if $\alpha_{\varphi 1}\varphi_1(X) + \cdots + \alpha_{\varphi n}\varphi_n(X) > \theta$.
The a's are called weights or coefficients and θ is called threshold. Threshold can be replaced with bias b = - θ.

$$\psi(X) = \begin{cases} 0 \; if \; \sum_{\varphi \in \Phi} \alpha_\varphi \varphi(X) \leq \theta \\ 1 \; if \; \sum_{\varphi \in \Phi} \alpha_\varphi \varphi(X) > \theta \end{cases}$$

or,

$$\psi(X) = \begin{cases} 0 \; if \; a \cdot \varphi(X) + b \leq 0 \\ 1 \; if \; a \cdot \varphi(X) + b > 0 \end{cases} \quad , \quad b = -\theta$$

Marvin Minsky and Seymour Papert define perceptron as:
"a device capable of computing all predicates which are linear in some given set Φ of partial predicates." [20]

Perceptrons are binary classifiers which make decisions by weighing up evidence. For the output to get calculated, unit step activation function is used:

$$f(x) = \begin{cases} 0 \; if \; x \leq 0 \\ 1 \; if \; x > 0 \end{cases}$$

Changes in weights and biases produce different models. Weights and biases can be selected freely or be automatically tuned by devise learning algorithms.

The weight of each input represents its importance whereas bias value is used to shift activation function's curve up or down.
Perceptrons are divided into families, some of which are [20]:

i. Diameter-limited perceptrons
ii. Order-restricted perceptrons
iii. Gamba perceptrons
iv. Random perceptrons
v. Bounded perceptrons

In order for a model to be tuned, small changes in weights and bias are needed. When those changes take place, an irrelevantly big change in output is not desired. Perceptrons suffer from such an unwanted behavior due to the fact that their activation function is discontinuous.

The sigmoid neuron is a slightly more sophisticated and smoothed-out perceptron. Perceptron's unit step activation function was replaced by the sigmoid function:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

In this model, small changes in weights and biases achieve small changes in output, which means that it is more numerically stable than the perceptron. Also, it is easier to figure out the output changes for small parameter changes. The outputs are no longer binary, they are infinite in [0,1]. Provided a binary

classification is needed, an appropriate threshold can be set. Other activation functions used in neurons are:

tanh (Figure 1)

$\tanh(x)$

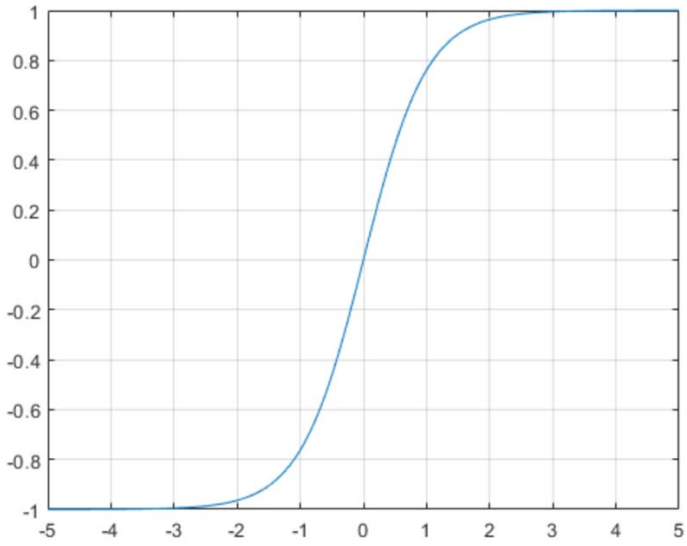


Fig. 1: Graph of hyperbolic tangent function [21].

ReLU (Figure 2)

$\max(0, x)$

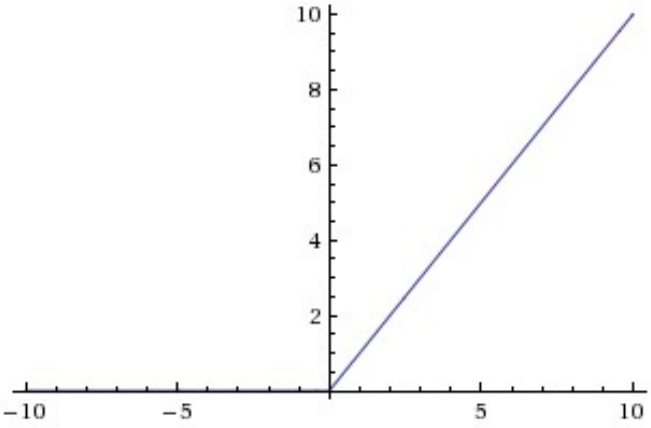


Fig. 2: Graph of rectified linear unit [22].

Leaky ReLU (Figure 3)

$$\max(0.1x, x)$$

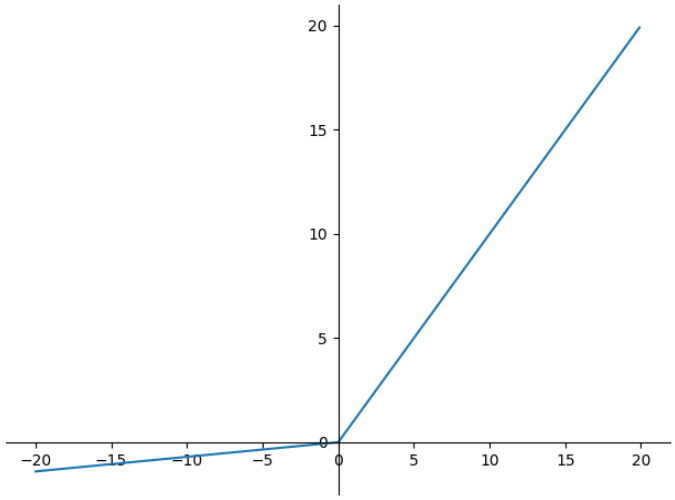


Fig. 3: Graph of leaky rectified linear unit [23].

Maxout (Figure 4)

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

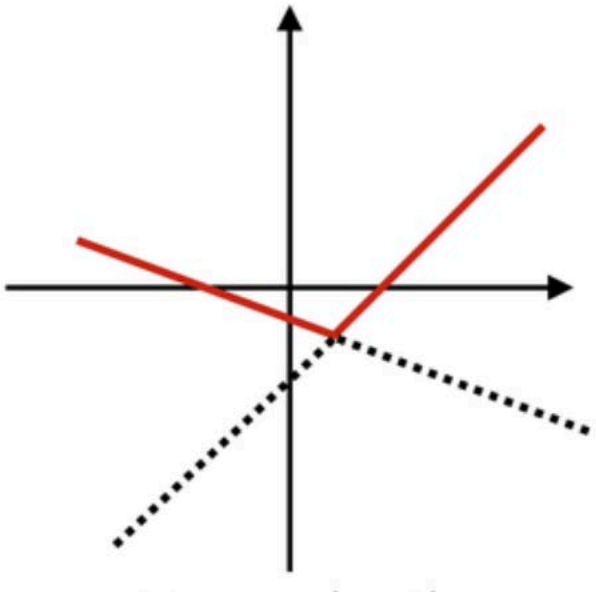


Fig. 4: Graph of maxout (n=2) [24].

ELU (Figure 5)

$$\begin{cases} x & x \geq 0 \\ a(e^x - 1) & x < 0 \end{cases}$$

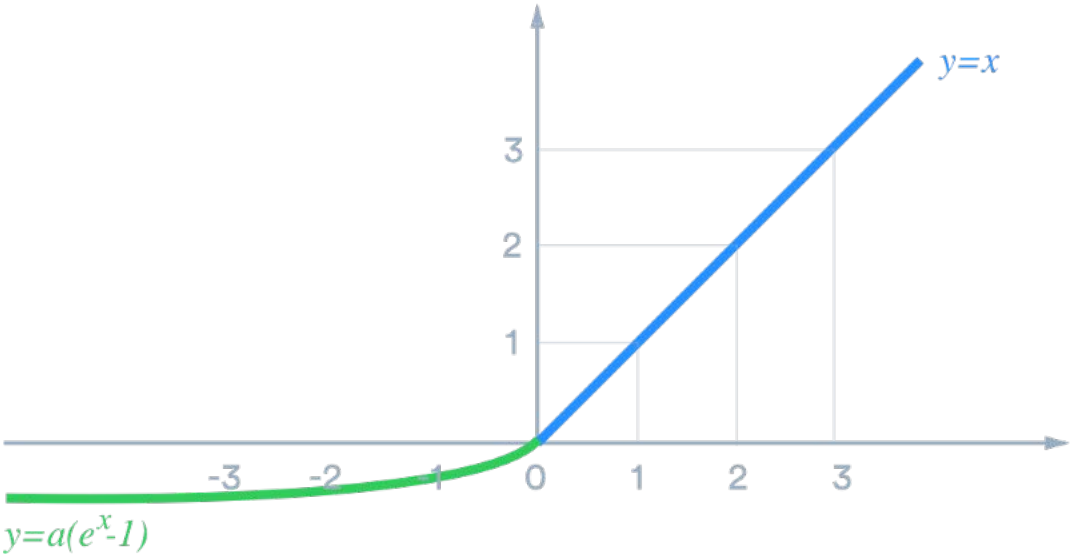


Fig. 5: Graph of exponential linear unit [25].

Artificial neural networks consist of multiple artificial neurons. Artificial neurons may be referred to also as nodes. Neurons are typically organized in layers which are interconnected [18]. In a layered architecture the set of computing units N is subdivided into λ subnets $N_1, N_2, \dots, N_\lambda$. Connections can only go from units in N_1 to units in N_2, from N_2 to N_3, etc. [26]. Multiple variations of the layered architecture exist and are used depending on the features of the problem the model is designed for. There are three types of layers:

i. Input Layer - N_1
ii. Hidden Layer - $N_2, \dots, N_{\lambda-1}$
iii. Output Layer - N_λ

Data are fed in the neural network through the input layer, also known as the first layer of the network. In this stage simple decisions are made. The processed data are then forwarded to the hidden layer. Hidden layer may be consisted of several layers depending on the problem and the architecture of the model. Those layers process data through a system of weighted connections. As a result, more complicated decisions are made at a more involved and abstract level. Because of their obscure decision-making, neural networks are commonly viewed as black box [27]. The final layer of a neural network is formed by computing units from which results are read off [26].

In order for a neural network to be trained, in a supervised manner, many learning algorithms have been devised. The algorithm which led neural computing back to the mainstream of computer science was the backpropagation algorithm which was rediscovered in the 1980s [26]. Backpropagation can be decomposed in four steps:

i. Feed-forward computation
ii. Backpropagation to the output layer
iii. Backpropagation to the hidden layer
iv. Weight updates

The termination of the algorithm is signaled when the value of the error function has become sufficiently small [26].

1.3 Bayesian Neural Networks

Neural networks can be treated in a Bayesian manner. Algorithms who do so, are referred to as Bayesian Neural Networks (BNNs) [28]. Originally,

Bayesian neural network model was studied as a one-layer recurrent artificial neural network. Keystone of this model was the idea of a naive Bayesian classifier. In this model, training is achieved via the Bayesian learning rule. All units in the network are considered as stochastic events and the weight calculation is based on the correlation between these events. The activation rate of a unit represents the probability of that event occurring, given the corresponding events [29].

In a classical neural network in order to optimize the loss of the loss function, it is common practice to seek a parameter optimum, say θ*- a point estimate of the weights. On the other hand, in the Bayesian approach, the inherent uncertainty in the estimates is taken in to account. For this to be done, priors are introduced on the network weights. Learning is attained by approximating the posterior, i.e., $p(\theta|data)$. Some of the methods used to achieve the posteriors approximation are:

i. Probabilistic backpropagation
ii. Variational inference
iii. Expectation propagation

The approximation of the posterior, in most cases, is challenging. Additionally, complexity in a Bayesian neural network architecture may rise further, as the selection of the prior might prove to be challenging. Bayesian neural networks are well suited when uncertainty estimates are important [28]. These networks are based on the famous Bayes theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In this thesis the challenging problem of the Bayesian inference will be approached by Variational Bayes, which is an approximation of Bayesian inference. It is a method which approaches large scale, multidimensional modern problems rapidly. Some of the problems this method is used for are:

i. Text analysis
ii. Structure discovery
iii. Gene expression analysis
iv. Neuroscience

As mentioned above Bayesian inference makes use of the Bayes rule:

$$p(\theta|y_{1:N}) \propto_{\theta} p(y_{1:N}|\theta)p(\theta)$$

Or

$$p(\theta|y_{1:N}) = p(y_{1:N}|\theta)p(\theta)/p(y_{1:N})$$

Were θ is the parameter.

$y_{1:N}$ are the data.

$p(\theta|y_{1:N})$ is the posterior.

$p(y_{1:N}|\theta)$ is the likelihood.

$p(\theta)$ is the prior distribution over the parameter.

$p(y_{1:N})$ is the evidence. It can also be written as: $\int p(y_{1:N}, \theta)d\theta$

The steps for Bayesian inference are [30]:

i. Build a model: choose prior and likelihood
ii. Compute the posterior
iii. Report a summary, e.g. posterior means (point estimate) and (co)variances (uncertainty)

Steps **ii** and **iii** present a computational challenge. That is the reason why approximate Bayesian inference is used. Markov Chain Monte Carlo (MCMC), which is a Bayesian inference approximation algorithm, has been called one of the ten most influential algorithms of the 20th century and it is widely used because it is accurate. The problem with this algorithm is that it is extremely slow. Instead of Markov Chain Monte Carlo, an optimization approach can be used. The posterior will be approximated with q^*.

$$q^* = argmin_{q \in Q} f(q(\cdot), p(\cdot\,|y))$$

Where f represents the farness from the posterior $p(\theta|y)$. f is not a distance, because that would mean that it is positive definite and symmetric.
Using Variational Bayes (VB) f is Kullback-Leibler divergence in a particular direction:

$$\mathrm{KL}(q(\cdot)||p(\cdot\,|y)) := \int q(\theta) log \frac{q(\theta)}{p(\theta|y)} d\theta$$

The reasons why Kullback-Leibler divergence has been chosen are the fact that it is extensively used, it is fast and point estimates and predictions seem

to be pretty accurate. The main problem presented is that we don't know the exact posterior. So, by analyzing Kullback-Leibler divergence we get:

$$\int q(\theta) log \frac{q(\theta)}{p(\theta|y)} d\theta = \int q(\theta) log \frac{q(\theta)p(y)}{p(\theta,y)} d\theta = \int q(\theta) \log q(\theta)\, d\theta +$$
$$+ \int q(\theta)p(y) d\theta - \int q(\theta) \log\big(q(\theta)p(y)\big)\, d\theta =$$
$$= \log p(y) - \int q(\theta) log \frac{p(\theta,y)}{q(\theta)} d\theta$$

We are trying to find the q that minimizes the Kullback-Leibler divergence, so the term $\log p(y)$ of the result above can be ignored because it affects nothing. On the other hand, the second term $\int q(\theta) log \frac{p(\theta,y)}{q(\theta)} d\theta$ contains only thing that we already know [30]. Because:

$$p(\theta, y) = p(y|\theta)p(\theta)$$

Which are the inputs of the problem. So, the optimization takes place in the $\int q(\theta) log \frac{p(\theta,y)}{q(\theta)} d\theta$ term which is named "Evidence Lower Bound" or (ELBO). It is known that:

$$KL \geq 0 \Rightarrow \log p(y) \geq ELBO$$

Minimizing KL, namely finding q* to be the minimizer of the KL is exactly equivalent of q* being the maximizer of the ELBO:

$$q^* = argmax_{q \in Q} ELBO(q)$$

That way the closest of the “nice” distributions is found. Nice distributions are [29][30]:

i. Low dimensional distributions.
ii. Exponential families.

2. Related Work

Intrusion Detection Systems are a vital component of the security infrastructure of a commercial or private network. In recent years, safety of data such as sensitive personal information, financial records, patents etc have proven to be a cornerstone for maintaining the trust between corporations and consumers. In this respect, many efforts have been made to develop intrusion detection systems.

2.1 K-means

One of the methods used to create an intrusion detection system was the combination of K-means and Random Forest algorithms [7]. The methodology followed in this approach was:

i. Clustering via K-means algorithm
ii. Classification via Random Forest algorithm

A distance-based clustering was performed on objects in order to classify the invasions into clusters. The reason why clustering was used is that no information about the labels of the data existed, so unsupervised learning was unavoidable. K-means algorithm was used to cluster dataset connections. This algorithm is widely used when clustering is needed and groups the data into a user-specified number of K clusters. K-mean's algorithm steps are:

i. The primary group centroids are denoted by K points which are placed into the space.
ii. Data are assigned into groups according to their adjacency to the centroid.
iii. When all data have been assigned to groups, the position of the K centroids is recalculated.
iv. Steps ii and iii are repeated until the centroids positioning remains unchanged.

By applying K-means algorithm to the dataset, data are partitioned into 5 groups. That is because the number assigned to K was 5.

Following the clustering, classification is achieved via the Random Forest algorithm, which is a supervised classification algorithm. The forest created by this algorithm consists of trees. As the number of trees grows, the result of the classification gets more accurate. The way each classification tree grows in the method used is:

i. Given that the number of cases in a training set is N, a random sample of N cases is selected from the original data. That way the training set for growing the tree is formed.
ii. Let M is the number of the input variables, a number mM is specified where in each node m represents the variables selected at random out of M. A procedure where the node gets split, follows. The number m remains immutable during the forest growing procedure.
iii. Growing of the tree continues until no further extension can be made. No pruning is applied.

The dataset used for the evaluation of the IDS proposed was the KDDCup'99 dataset [7].

2.2 Support Vector Machines

Another approach in the construction of an intrusion detection system was based on Rough Set Theory (RST) and Support Vector Machines (SVM) [8]. The methodology followed was:

i. After data preprocessing feature selection was accomplished by the use of Johnson's and genetic algorithm of Rough Set Theory. That way valuable features were uncovered.
ii. The classification of data was achieved by feeding the reduced set of data to the Support Vector Machine.

Rough set theory is intended to handle data that are indistinguishable, imperfect or incompatible. Relationships are formed between every object of significance and a piece of information which represents relative relationship. That information is used to derive data classification. Summarizing, we get that Rough set theory is:

i. A data-mining technique which, given a large number of data will decrease their features.
ii. By conducting an upper and a lower approximation to set membership, it deals with unpredictability, vagueness and incompleteness.

iii. It is a method which has effectively been used to eliminate redundant information and determine data dependencies.

After the feature selection process, the method used to classify intrusions from normal data was Support Vector Machines. Those algorithms are capable of achieving high accuracy rates and perform great in classification problems. Support Vector Machines are:

i. A statistical learning theory which foundations are machine learning methods.
ii. It is an algorithm which enjoys great acceptance and usage in data mining, image recognition, bioinformatics and text recognition.
iii. A relatively quick algorithm.
iv. Scalable.

The input data and the output areas are given by:

$$(x_i, y_i), \dots, (x_n, y_n), x \in R^m, y \in \{+1, -1\}$$

Where $(x_i, y_i), \dots, (x_n, y_n)$ are the training data.
n is the number of samples.
m is the inputs vector.
y fits respectively into +1 or -1.

The dataset used for the evaluation of the IDS proposed was the KDDCup'99 dataset [8].

2.3 Bayesian Networks and Hidden Markov Model

Construction of an intrusion detection system has been also achieved by combining Bayesian Networks and Hidden Markov Model (HMM) [9]. While Hidden Markov Models are still in primary stages in framing intrusion detection systems construction, they are used in:

i. Speech recognition.
ii. Speech synthesis.
iii. Gene prediction.
iv. Crypt analysis. Etc.

A Hidden Markov Model consists of a finite set of states. A probability distribution is associated with each state. A set of probabilities, which are called state transition probabilities, determine the transitions among these states. An outcome or an observation, can be generated in a specific state through the associated probability distribution. The term “Hidden” is due to the fact that an external observer can’t see the state but only the outcome. A Hidden Markov Model can be defined by:

i. The number of states N that the model has.
ii. The number of observation symbols in the alphabet, M, which is infinite when the observations are continuous.
iii. A set of state transition probabilities.
iv. A probability distribution on each state.
v. The initial state distribution.

A Bayesian network can be also referred to as a belief network. It is an amplified acyclic graph which is represented by G(V, E).
V is a set of vertices.
E is a set of directed edges joining vertices.

In a belief network loops are prohibited. A name of a random variable and a probability distribution is contained in all vertices of V. The Bayesian network used was built as follows:

i. A set of variables was chosen from the training dataset.
ii. An ordering for the previously selected variables is chosen.
iii. Let $X_1, X_2, \dots, X_m$ the previously chosen order then:

For $i = 1\ to\ m$:

a. X_i node is added to the network.
b. Parents (X_i) are set to be a subset of $\{X_1 \dots X_{i-1}\}$ such that there is a conditional independence of X_i and all other members of the subset $\{X_1 \dots X_{i-1}\}$ given Parents(X_i).
c. The probability table of $P(X_i = \text{k} \mid \text{Assignments of Parents}(X_i))$ is defined.

In this approach of intrusion detection system construction after the preprocess of the data a separate Bayesian Network for normal and attack type records respectively is created via a Hidden Markov Model architecture.

The initialization of the Hidden Markov Model follows and its parameters are based on the previous Bayesian Networks.
The dataset used for the evaluation of the IDS proposed was the KDDCup'99 dataset [9].

2.4 Self-organizing map

An additional approach on intrusion detection systems construction was by the use of Self-organizing map (SOM) [10]. This is an algorithm which has the following characteristics:

i. Unsupervised.
ii. Has fast conversion.
iii. An automatic clustering algorithm.

Self-organizing map is a neural network model which aims in building a topology in which the points in the dataset preserve their neighborhood relation. This algorithm is used in:

i. Image processing.
ii. Voice recognition.
iii. Speech recognition.
iv. Spatial data mapping.
v. Data compression.
vi. Pattern recognition. Etc.

It is a feedback network which can turn high-dimensional data into low-dimensional ones. The grid of neurons in Self-organizing map is predefined. The way of learning that it uses is competitive learning. The neuron whose vector is more similar to the input vector is considered as the winning neuron. This neuron is called Best Matching Unit (BMU). After Best Matching Units have been determined, weights from neurons surrounding those units are adjusted, resulting to a distance diminution over time. Self-organized map based intrusion detection systems are already been used commercially because they can achieve high speed and fast conversion. After testing the algorithm in KDDCup99' dataset, there were two major concerns:

i. Poor detection rate of some attacks.
ii. Computational time.

In order to overcome those problems two variations of the initial algorithm were tested:

i. Hierarchical Self Organizing Map (HSOM)
ii. Growing Hierarchical Self Organizing Map (GHSOM)

The Hierarchical Self Organizing Map algorithm was found to be even more time demanding than the initial method used. In addition, in order for this algorithm to perform well, a great deal of effort was required to select the best combination of layers.

On the other hand, by using Growing Hierarchical Self Organizing Map the computational time problem was eliminated. This algorithm has an adaptive nature, which means that the topology grows as the inputs flow in.

Its initial grid is 2x2 and can expand according to thresholds, horizontally and vertically. As a result, all neurons of the resulting topology are beneficial for the detection [10].

2.5 Random Forest

Random Forest is a common classification algorithm which has also been used for the construction of intrusion detection systems [11]. A collection of decision trees, form the random forest. For nodes to split, the number of trees, the minimum node size and the number of features, are used. The advantages of the random forest algorithm are:

i. Forests that have been generated can be saved and be used for future reference.
ii. Over fitting problem is avoided.
iii. Accuracy and variable importance are generated automatically.

Before the data are fed to the random forest algorithm feature selection (FSS) is applied. By applying feature selection, the following are achieved:

i. Dimensionality reduction.
ii. Irrelevant features removal.
iii. Improved accuracy of the classification.

Additional preprocessing techniques used are:

i. Replacement of missing values.
ii. Discretization.

The dataset used for the evaluation of the IDS proposed was the KDDCup'99 dataset and the evaluation was performed by 10-fold cross validation [11].

2.6 Artificial neural networks

Intrusion detection systems construction has also been accomplished via neural networks as described in the following approach [12]. There are several architectures and types of neural networks. Five different neural networks were tested:

i. Feed Forward Neural Network (FFNN).
ii. Elman Neural Network (ENN).
iii. Generalized Regression Neural Network (GRNN).
iv. Probabilistic Neural Networks (PNN).
v. Radial Basis Neural Network (RBNN).

Feed forward neural networks can be either single-layer or multi-layer networks. Single layer neural networks consist of an input and an output layer while multi-layer neural networks have an additional hidden layer. The hidden layer may be formed by more than one interconnected layers. Signals in a feed forward neural network can only travel from input to output layer.

By adding layer recurrent connections with tap delays in a feed forward neural network, an Elman neural network is created. This type of network can perform sequence prediction, which is something that multilayer perceptrons can't.

Generalized regression neural networks have four layers:

i. Input layer.
ii. Hidden layer.
iii. Pattern layer / Summation layer.
iv. Decision layer.

Their advantages compared to a multilayer perceptron are:

i. Faster training.
ii. Better accuracy.
iii. Relative insensitivity to outliers.

While their disadvantages are:

i. Slower in the classification of new cases.
ii. More memory demanding.

Probabilistic neural networks in essence are functions which approximate the distribution's probability density.

They consist of three layers:

i. Pattern Layer.
ii. Summation Layer.
iii. Output Layer.

Lastly, radial basis neural networks also consist of three layers:

i. Input layer.
ii. Hidden layer.
iii. Output layer.

Hidden layer neurons contain Gaussian transfer functions. Those networks are considered to be curve-fitting problems in high-dimensional space.
The dataset used for the evaluation of the IDS proposed was the KDDCup’99 dataset [12].

3. Intrusion Detection System Proposed

In this thesis the construction of an intrusion detection system has been attained through a Bayesian neural network. Those networks were chosen due to the following three reasons:

i. Their unique ability to say “I don’t know”.
ii. Fast convergence of data training phase.
iii. Their ability to be trained by small training datasets.

The facts that internet is an environment where new or variations of already known attacks are a frequent phenomenon and datasets of network traffic are limited, it would be a game changing tool, for a network administrator, an intrusion detection system which in addition to the extra level of security that it provides, can also distinguish traffic which is unknown. That way new datasets could be formed, better training on intrusion detection systems achieved and system or network vulnerabilities avoided.

3.1 Programming environment – Tools

The intrusion detection system was implemented by using Python programming language in PyCharm CE IDE. The libraries used are:

i. Pytorch.
ii. Pyro.
iii. Numpy.
iv. Matpotlib.

Numpy and Matprotlib are widely known libraries and are covered by an extensive number of references, therefor, in this thesis, more details will be provided for Pytorch and Pyro.

Pytorch is a computing package which is based on Python. It is a platform which provides flexibility, speed and ease of use in deep learning research. In addition, GPUs computing power can be utilized. The reasons why Pytorch was preferred are [31]:

i. Ease of use.
ii. It has Python support because it integrates with the python data science stack.
iii. Dynamic computational graphs can be built by a framework provided.

Probabilistic modeling of the Bayesian neural network created, was achieved by Pyro. This is a universal probabilistic programming language (PPL) which is supported by PyTorch and it is also written in Python. Key benefits of Pyro are [32]:

i. Universality. Any computable probability distribution can be represented.
ii. Scalability. Little overhead is required to scale to large datasets.
iii. Because of the fact that it is implemented with a small core of abstractions, Pyro is minimal.
iv. Flexibility.

3.2 Datasets – Preprocessing

A very common dataset used for training and testing of intrusion detection systems is the KDDCup'99 dataset [33]. That is the dataset which is also used for the testing of the intrusion detection system of this thesis. The exact datasets used are:

For training: kddcup.data_10_percent.gz

For testing: corrected.gz

The training set consists of 494,021 data while the test set of 311,029 data. Each data of the datasets has 41 features and a label. A total of 42 columns. The list of the feature is:

1. duration: continuous.
2. protocol_type: symbolic.
3. service: symbolic.
4. flag: symbolic.
5. src_bytes: continuous.
6. dst_bytes: continuous.
7. land: symbolic.
8. wrong_fragment: continuous.
9. urgent: continuous.
10. hot: continuous.
11. num_failed_logins: continuous.
12. logged_in: symbolic.
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: symbolic.
22. is_guest_login: symbolic.
23. count: continuous.
24. srv_count: continuous.
25. serror_rate: continuous.
26. srv_serror_rate: continuous.
27. rerror_rate: continuous.
28. srv_rerror_rate: continuous.
29. same_srv_rate: continuous.
30. diff_srv_rate: continuous.
31. srv_diff_host_rate: continuous.
32. dst_host_count: continuous.
33. dst_host_srv_count: continuous.
34. dst_host_same_srv_rate: continuous.
35. dst_host_diff_srv_rate: continuous.
36. dst_host_same_src_port_rate: continuous.
37. dst_host_srv_diff_host_rate: continuous.
38. dst_host_serror_rate: continuous.
39. dst_host_srv_serror_rate: continuous.
40. dst_host_rerror_rate: continuous.
41. dst_host_srv_rerror_rate: continuous.

Training and test sets have 5 different classes of data:

i. Normal data
ii. Denial of Service (DoS) attacks.
iii. Remote to Local (R2L) attacks.
iv. Probing attacks.
v. User to Root (U2R) attacks.

Each of the attack classes consists of several attack types. In the training set there are 22 different attack types which are classified as shown in table 1:

Table 1: Training set analysis

Class	Attack type	Total number of data
DoS	back, land, neptune, pod, smurf, teardrop	391,458
R2L	ftp-write, guess-password, imap, multihop, phf, spy, warezclient, warezmaster	1126
Probing	Ipsweep, nmap, portsweep, satan	4107
U2R	Buffer-ocerflow, loadmodule, perl, rootkit	52

In the test set there are 17 additional attack types which classify as shown in table 2:

Table 2: Test set analysis

Class	Attack type	Total number of data
DoS	processtable, apache2, mailbomb, udpstorm	229,853
R2L	named, snmpguess, snmpgetattack, sendmail, xlock, httptunnel, worm, xsnoop	16,347
Probing	saint, mscan	4166
U2R	xterm, ps, sqlattack	70

From the aforementioned 41 features, three of them, namely protocol_type, service and flag, were categorical. Those values were mapped into numerical input data as follows:

Say A is a categorical feature which has 3 values {a, b, c}.

Then:

{a} is represented as [1,0,0]
{b} is represented as [0,1,0]

{c} is represented as [0,0,1]

Due to the large deviation of the feature values, normalization and standardization are the two frequently used methods to resolve this issue. In this thesis a standardization method was applied for data preprocessing, with the use of standard score (z-score):

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

Where X is the input value that needs to be transformed.

μ is the mean.

σ is the standard deviation.

By performing this transformation, we express each value X into a z-score which has mean equal to 0 and a standard deviation equal to 1. Z-score transformation was selected because its sensitivity to data outliers is rather small.

One of the codes used for preprocessing the data was as follows:

```
import pandas as pd
import numpy as np
import json

from sklearn.preprocessing import StandardScaler



def header_creation(df):
    header_list = []
    for i in range(0, len(df.columns)):
        col_name = str(i)
        header_list.append(col_name)

    df.columns = header_list
    return df


def inner_header_creation(df,col_name):
    header_list = []
    col_num = ''
    for i in range(int(col_name), int(col_name)+len(df.columns)):
        col_num = str(i)
        header_list.append(col_num)

    df.columns = header_list
    return df


def rename_header(df,col_name,length):

    header_list = []
    for i in range(0, len(df.columns)):
        if i < int(col_name):
            col_num = str(i)
            header_list.append(col_num)
        else:
            col_num = str(i + length)
            header_list.append(col_num)

    df.columns = header_list
    return df


def string_to_code(df,col_name):
    table_size = len(df[col_name].unique())
    dicts1 = {}
    a = np.zeros((table_size, table_size), int)
    np.fill_diagonal(a, 1)
    a_str = ''
```

```
    a_list = []

    for j in range(0, table_size):
        for i in range(0, table_size):
            if i == 0:
                a_str = str(a[j][i])

            else:
                a_str = a_str + ',' + str(a[j][i])
        a_list.append(a_str)

    labels1 = list(df[col_name].unique())

    for i in range(0, table_size):
        dicts1[labels1[i]] = a_list[i]

    filename = 'labels' + col_name +'.json'
    with open(filename, 'w') as json_file:
        json.dump(dicts1, json_file)

    df = df.replace({col_name: dicts1})
    return df

def string_to_code_test_set(df,col_name):
    filename = 'labels' + col_name +'.json'
    with open(filename) as json_file:
        dicts1 = json.load(json_file)

    df = df.replace({col_name: dicts1})

    return df



def spilt_cols(df,col_name):

    df2 = df[col_name].str.split(",", expand=True)
    df2 = inner_header_creation(df2, col_name)
    try:
        df2 = df2.apply(pd.to_numeric)
    except:
        df2.fillna('0', inplace=True)
        df2.loc[((df2[col_name] != '1') & (df2[col_name] != '0')), col_name] = '0'
        df2 = df2.apply(pd.to_numeric)


    length = len(df2.columns)
    df = df.drop(col_name, axis=1)

    df = rename_header(df, col_name, length)

    df3 = pd.concat([df, df2], axis=1)
    labels = list(df3)
    labels.sort(key=int)
```

```
    df3 = df3.reindex(columns=labels)

    return df3


def column_number_to_change(df,col_name,set_type):

    if set_type == "train":
        df = string_to_code(df, col_name)
    elif set_type == "test":
        df = string_to_code_test_set(df, col_name)
    else:
        print("set correct set_type option")
        exit()
    df = spilt_cols(df, col_name)

    return df


def file_processing(df,set_type):
    df = header_creation(df)
    last_column = str(len(df.columns) - 1)
    df = group_values(df, last_column)

    while True:
        flag = 0
        for i in range(0, len(df.columns) - 1):
            col_name = str(i)
            if df[col_name].dtype == 'object':
                print("get in column: ", i)
                df = column_number_to_change(df, col_name,set_type)
                flag = 1
                break
        if flag == 0:
            break

    return df

def group_values(df,col_name):
    dicts = {'normal.':'normal', 'buffer_overflow.':'u2r', 'loadmodule.':'u2r', 'perl.': 'u2r',
'neptune.':'dos', 'smurf.':'dos', 'guess_passwd.':'r2l', 'pod.':'dos', 'teardrop.':'dos',
'portsweep.':'probe', 'ipsweep.':'probe', 'land.':'dos', 'ftp_write.': 'r2l','back.':'dos', 'imap.':'r2l',
'satan.':'probe', 'phf.':'r2l', 'nmap.':'probe', 'multihop.':'r2l','warezmaster.':'r2l',
'warezclient.':'r2l', 'spy.':'r2l', 'rootkit.':'u2r','named.':'r2l', 'xterm.':'u2r', 'ps.':'u2r',
'saint.':'probe', 'snmpguess.':'r2l', 'snmpgetattack.':'r2l','sendmail.':'r2l', 'xlock.':'r2l',
'httptunnel.':'r2l', 'processtable.':'dos', 'sqlattack.':'u2r', 'apache2.':'dos', 'worm.':'r2l',
'xsnoop.':'r2l', 'mailbomb.':'dos', 'mscan.':'probe', 'udpstorm.':'dos'}

    df = df.replace({col_name: dicts})
    return df


def normalize_values(df, df1):
    scaler = StandardScaler()
```

```
    col_list = []
    for i in range (0, len(df.columns) - 1):
        col_name = str(i)
        if df[col_name].dtype != 'object':
            x = list(df[str(i)].unique())
            if (x[0] != 0 and x[0] != 1) or len(x) > 2:
                col_list.append(col_name)
    df[col_list] = scaler.fit_transform(df[col_list])
    df1[col_list] = scaler.transform(df1[col_list])


    return df, df1

def separate(df, df1, filename, filename1):

    df_labels = df[['118']].copy()
    df1_labels = df1[['118']].copy()

    df = df.drop('118',axis=1)
    df1 = df1.drop('118',axis=1)


    filename = filename + "_preprocessed.csv"
    df.to_csv(filename, index=False)
    filename1 = filename1 + "_preprocessed.csv"
    df1.to_csv(filename1, index=False)
    df_labels.to_csv("Labels_"+filename, index=False)
    df1_labels.to_csv("Labels_" + filename1, index=False)


def main():


    # training set
    print("training set")
    filename = "train_set"
    df = pd.read_csv("/Users/Orestis/Desktop/Diplwmatikh Met/Data/Train/train_set",
header=None)
    set_type = "train"

    df = file_processing(df, set_type)

    # test set
    print("test set")
    filename1 = "test_set"
    df1 = pd.read_csv("/Users/Orestis/Desktop/Diplwmatikh Met/Data/Test/test_set",
header=None)
    set_type = "test"

    df1 = file_processing(df1, set_type)
    df, df1 = normalize_values(df, df1)
    separate(df, df1, filename, filename1)

main()
```

From the initial training set, three smaller ones were created as shown in table 3.

Table 3: Analysis of the training datasets used

Dataset	Normal	DoS	R2L	Probe	U2R	Sum.
Initial Dataset	97,278	391,458	1,126	4,107	52	494,021
Mini dataset_1	130	126	133	120	52	561
Mini dataset_2	750	851	706	800	52	3,159
Mini dataset_3	350	601	406	450	52	1,859

Those sets where created in order to investigate whether Bayesian neural network trained by the smaller training dataset could perform at a similar or even enhanced accuracy with the initial dataset. The three smaller datasets tested were < 0.64% of the original size. Tests were conducted with the same test dataset consisted of 311,029 data.

3.3 Architecture

The preprocessing described above leads to 118 features which will act as the input and 5 classes from which the classification of the data will be determined. Therefore, the model's architecture will consist of 118 input-layer neurons and 5 output-layer neurons. The decision left to be made is regarding the size and depth of the hidden layer.

There are no solid rules to construct a perfect hidden layer. This is more of a trial and error process that with the help of heuristic methods, as the ones that follow, a kind of guideline is provided that helps to determine the size therefore the number of neurons.

1. Heuristic method:

$$\frac{number\ of\ input\ neurons + number\ of\ output\ neurons}{2}$$

Application:

$$\frac{118 + 5}{2} \approx 62$$

2. Heuristic method:

$$\frac{(number\ of\ input\ neurons + number\ of\ output\ neurons) \cdot 2}{3}$$

Application:

$$\frac{(118 + 5) \cdot 2}{3} = 82$$

3. Heuristic method: $x < 2 \cdot number\ of\ input\ neurons$

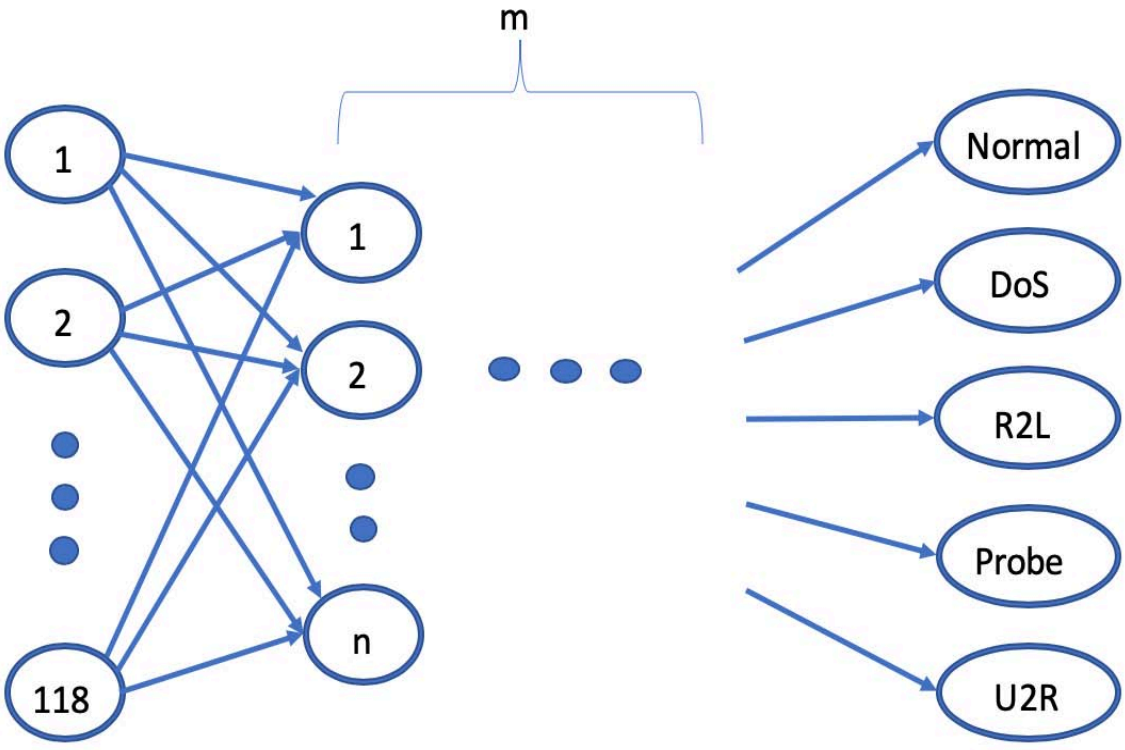
Application: $x < 236$, the x chosen was 210

Apart from the aforementioned three architectures, two more were tested, namely 30 and 40.

As for the depth of the hidden layer, two approaches were considered. The first one having one hidden layer whereas the second one having two.

The following chart displays the architectures described above (Chart 1**)**.

Chart 1: Graphic display of Bayesian neural network architecture.



In the above chart
n is the number of the selected size of neurons on the hidden layer (30,40,62,82,210).
m is the number of hidden layers (1,2).

In total ten different architectures were tested. Each architecture was trained by the four datasets mentioned above and tested with the same test set.

3.4 Training

In the training process weights and biases must be adjusted so that the loss function is minimized. The formation of weights and biases follows the same procedure. The way that the weights/biases are tuned is determined by the optimizer. In Bayesian neural networks, the goal of training is the approximation of the posterior distribution of the weights/biases. In order for this to be achieved standard conjugate analysis has been used with hierarchical normal priors and normal likelihood, leading to normal posterior distributions. Scale and location parameters, of these distributions, are approximated with variational Bayes method.

For better comprehension of the method followed, a detailed paradigm will be presented of how the distribution of a weight is deduced. The same applies for biases as mentioned above.

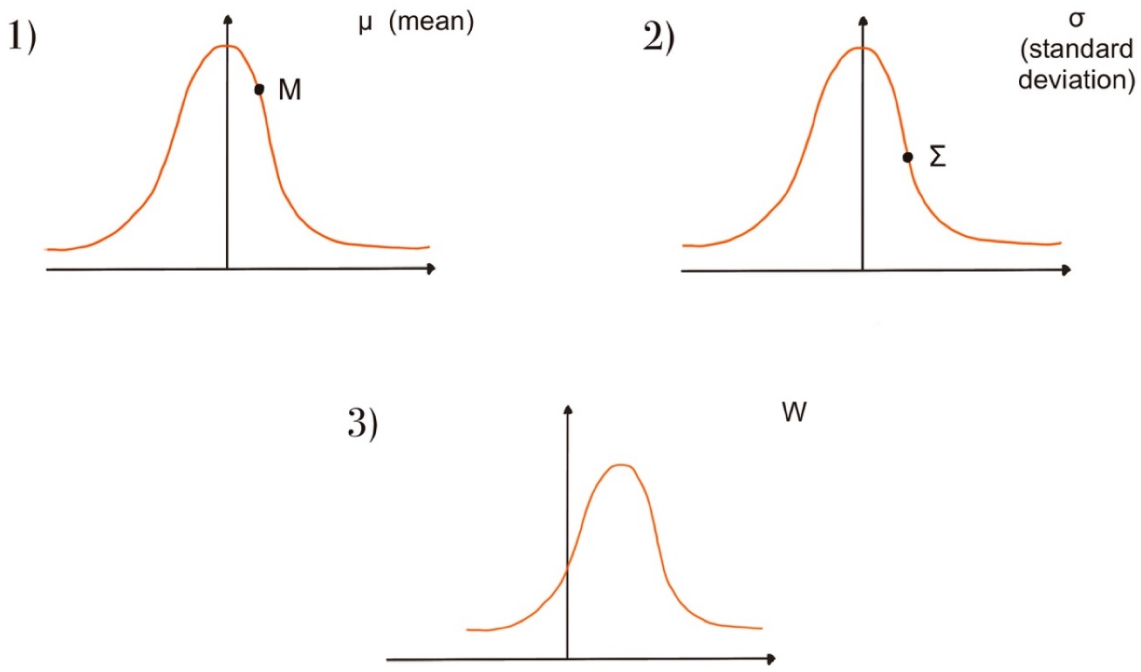


Fig. 6: Graphic representation of how weights derive.

As shown in figure 6:

Step 1. A random point – number **M** is selected from graph 1, which represents the normal distribution of mean (μ).

Step 2. Similarly, a random point – number **Σ** is selected from graph 2, which represents the normal distribution of standard deviation (σ).

Step 3. From the above two point – numbers M, Σ, a normal distribution derives for weight (w), as shown in graph 3.

Therefor in the new graph (3):

$$mean = M \qquad \text{and} \qquad standard\ deviation = \Sigma$$

The Σ chosen is transformed by softplus function (Figure 7):

$$f(x) = \ln(1 + e^{x}) \ , \ f(x) \in (0, +\infty)$$

Step 4. Through variational inference new mean and standard deviation normal distributions are inferred. The procedure is repeated from Step1 until loss converges.

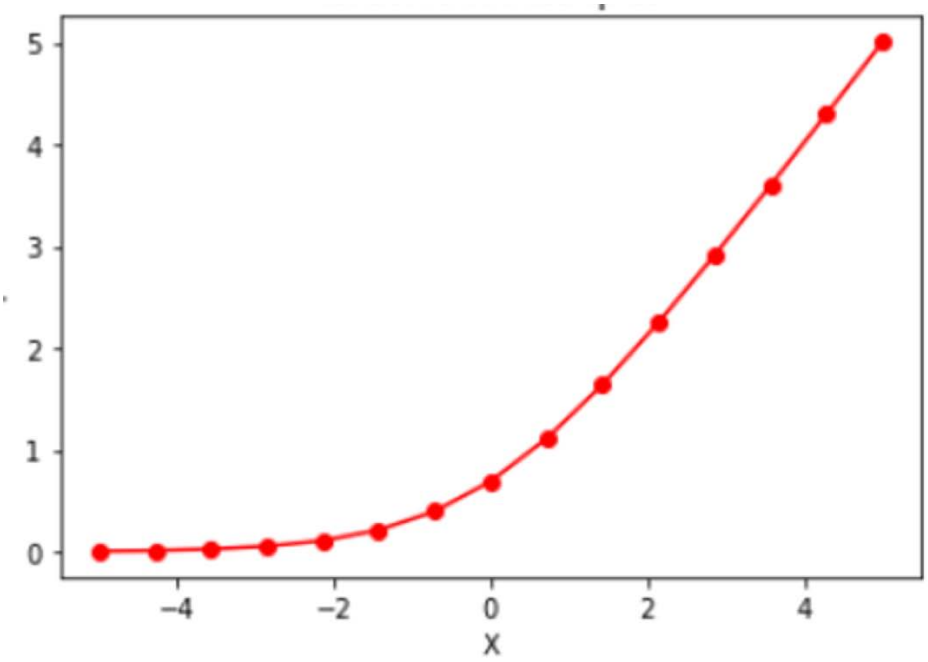


Fig. 7: Graph of softplus function [34].

The tools needed for stochastic variational inference are the loss function and the optimization algorithm. Loss function measures how wrong the predictions of the model are. In addition, it shows how the optimization performed on weights, affected the training procedure. Meaning that if current loss is higher than the previous one, optimization performed was in the wrong "direction". Essentially, loss function, is the guide that determines how the optimization algorithm must proceed. When the loss is stabilized, the model has finished its training, or as it is most commonly known, the model has *converged*.

The optimization algorithm determines, how and when, changes on weights will be performed.

The loss function and optimizer used in the training of the data are:

Loss function: ELBO.

Optimizer: ADAM optimizer.

3.5 Testing

When the algorithm is sufficiently trained, testing must be performed in order for the intrusion detection system to be evaluated. For a prediction to be made, ten different sets of weights and biases are sampled from the distributions that emerged from the training process. A prediction is made for each of the sets and the final decision for the classification is made by averaging those predictions and selecting the class with the highest activation value. In other words, the final result (the prediction) occurs from the

combination of ten different neural networks. The number of weights and biases that are sampled may defer without any restrictions.

The above-mentioned complexity of how the prediction derives explains the reason why there may be a denial of an answer, which is an integrated characteristic of Bayesian neural networks.

In the testing phase of this thesis the results came from two approaches. In the first one the algorithm was obligated to provide an answer whereas in the second one the algorithm could deny an answer.

In the second approach a threshold is set which indicates the certainty with which the algorithm provides an answer. Bellow this threshold no answer is given.

Due to the vast number of results, a sample of them will be distributed in the following pictures showing the accuracy achieved by the algorithm. From each of the four training sets the result of two architectures will be provided. One of them from the “one hidden layer” and the other for the “two hidden layers” approach.

In the following pictures classes either mentioned by name or numerically.

0 → Normal

1 → DoS

2 → R2L

3 → Probe

4 →U2R

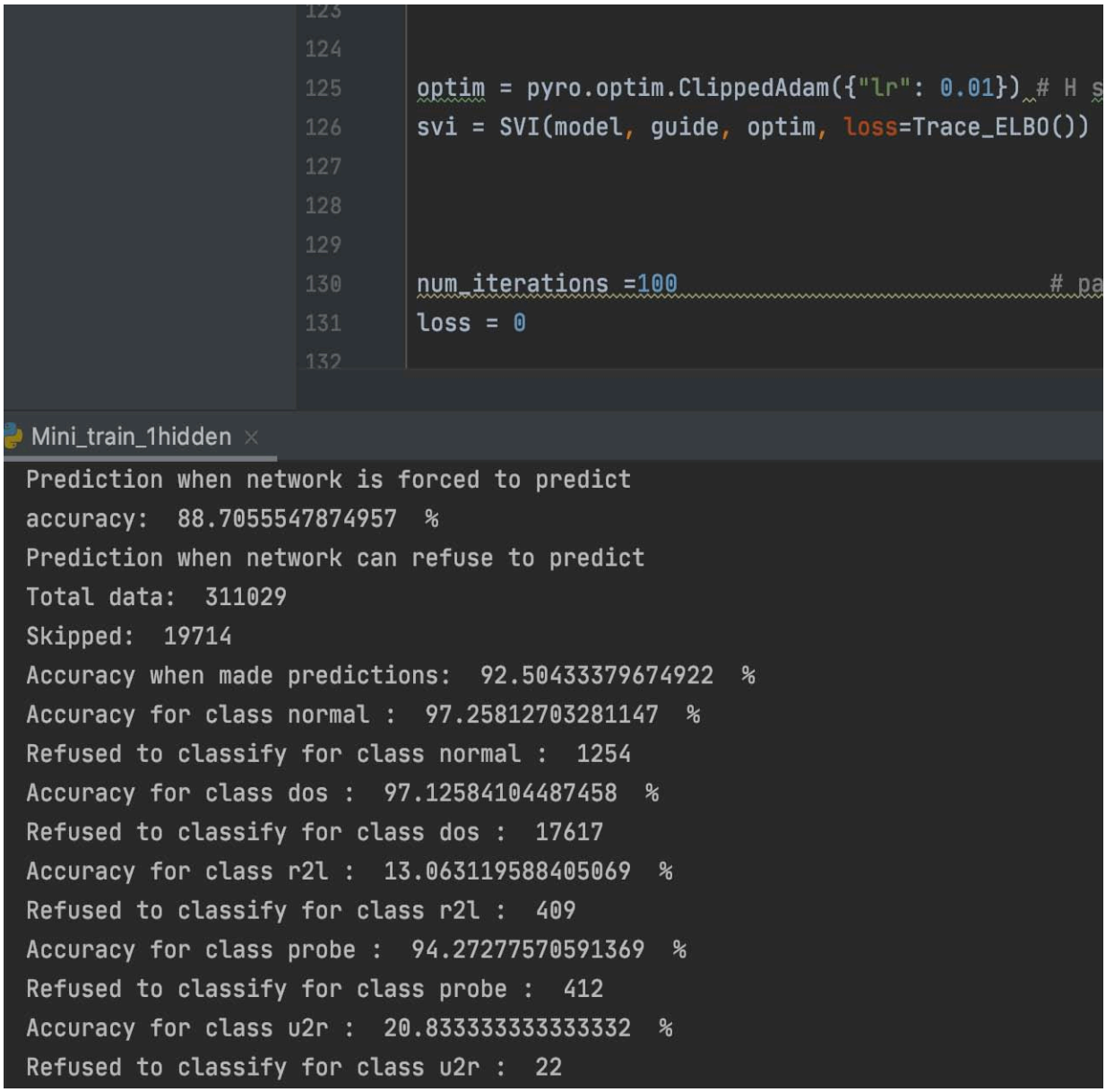


Fig. 8: Mini dataset_3, 1-hidden layer, 40 neurons

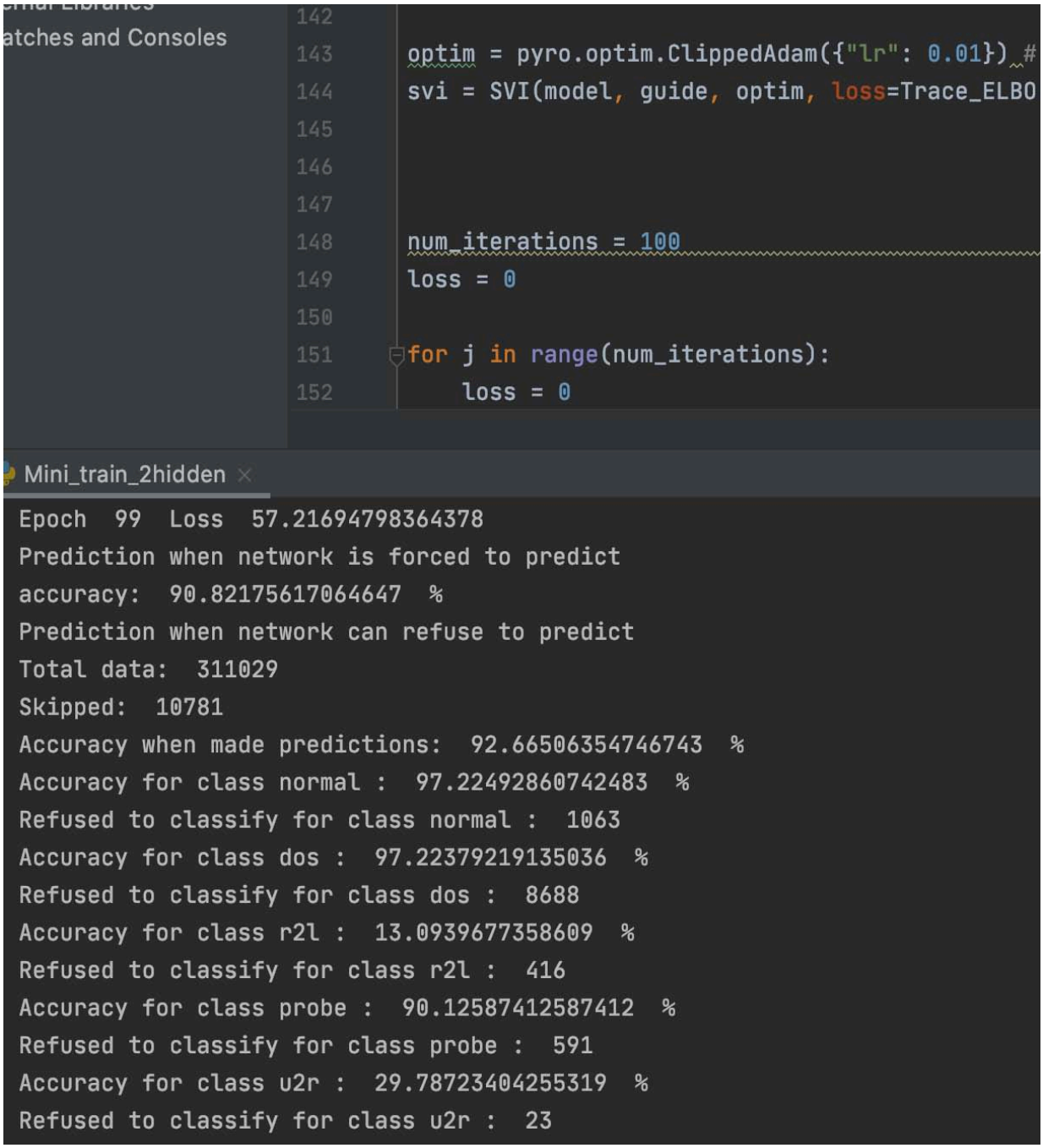


Fig. 9: Mini dataset_3, 2-hidden layers, 62 neurons each.

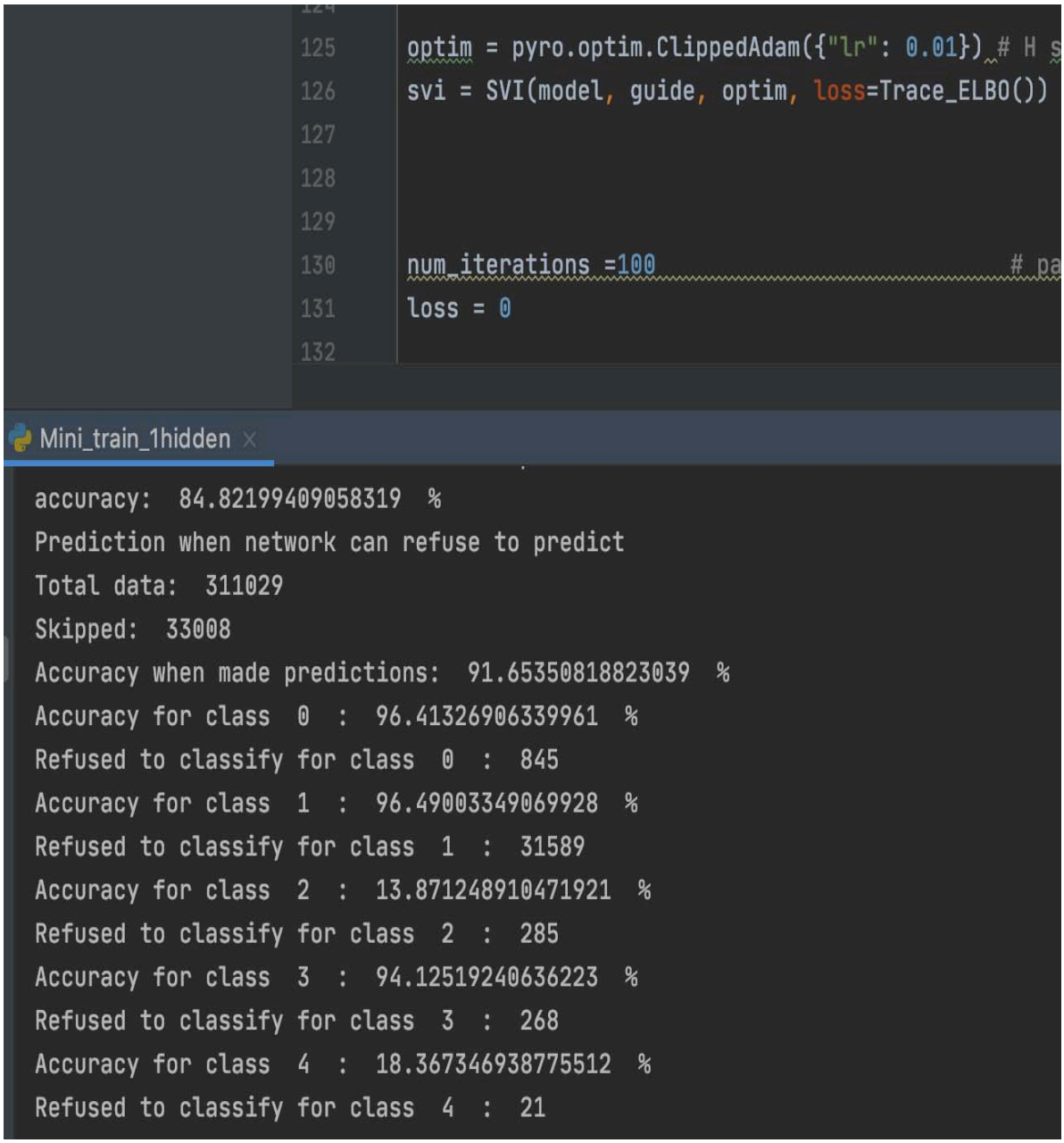


Fig. 10: Mini dataset_2, 1-hidden layer, 82 neurons.

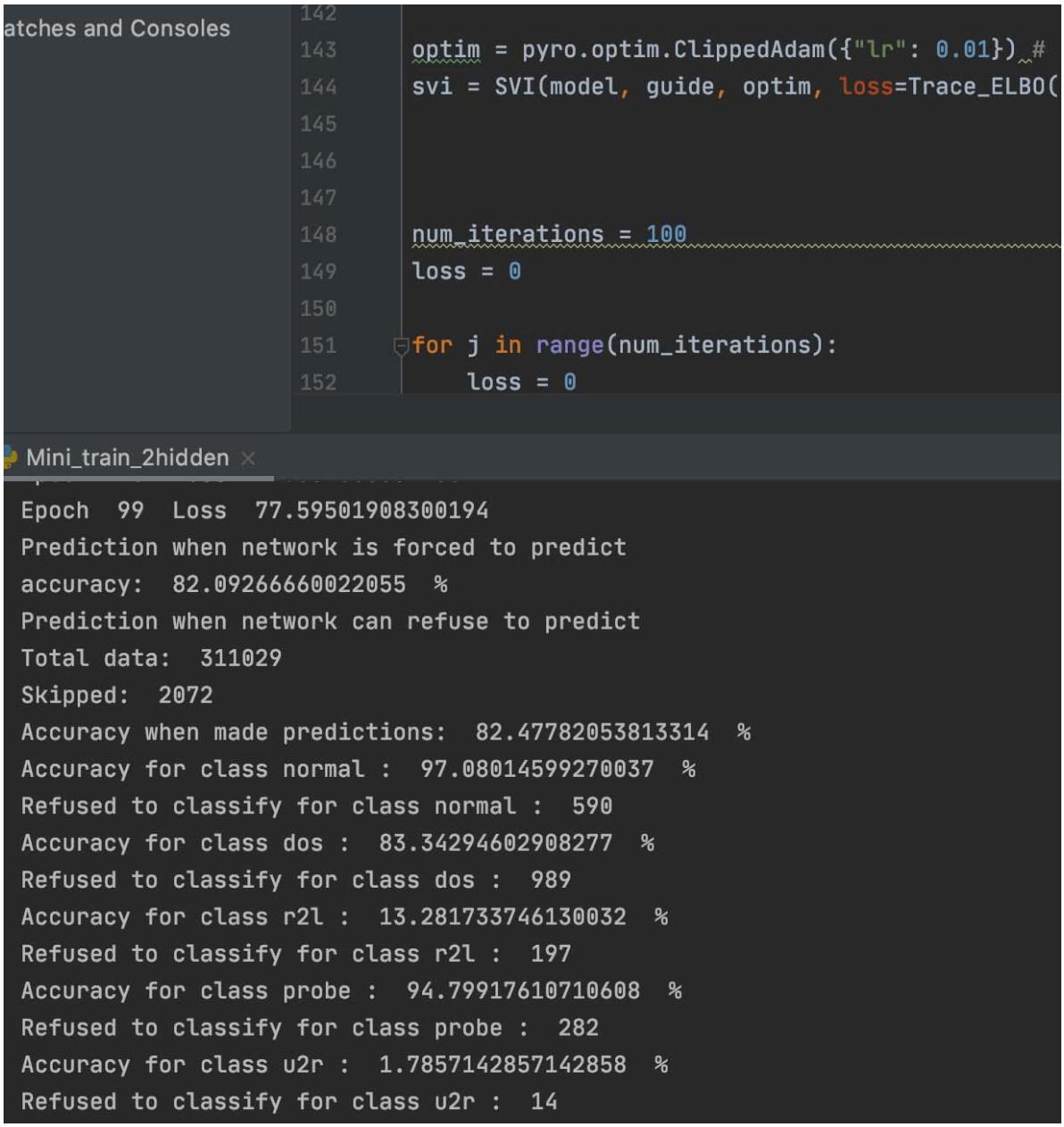


Fig. 11: Mini dataset_2, 2-hidden layers, 82 neurons each.

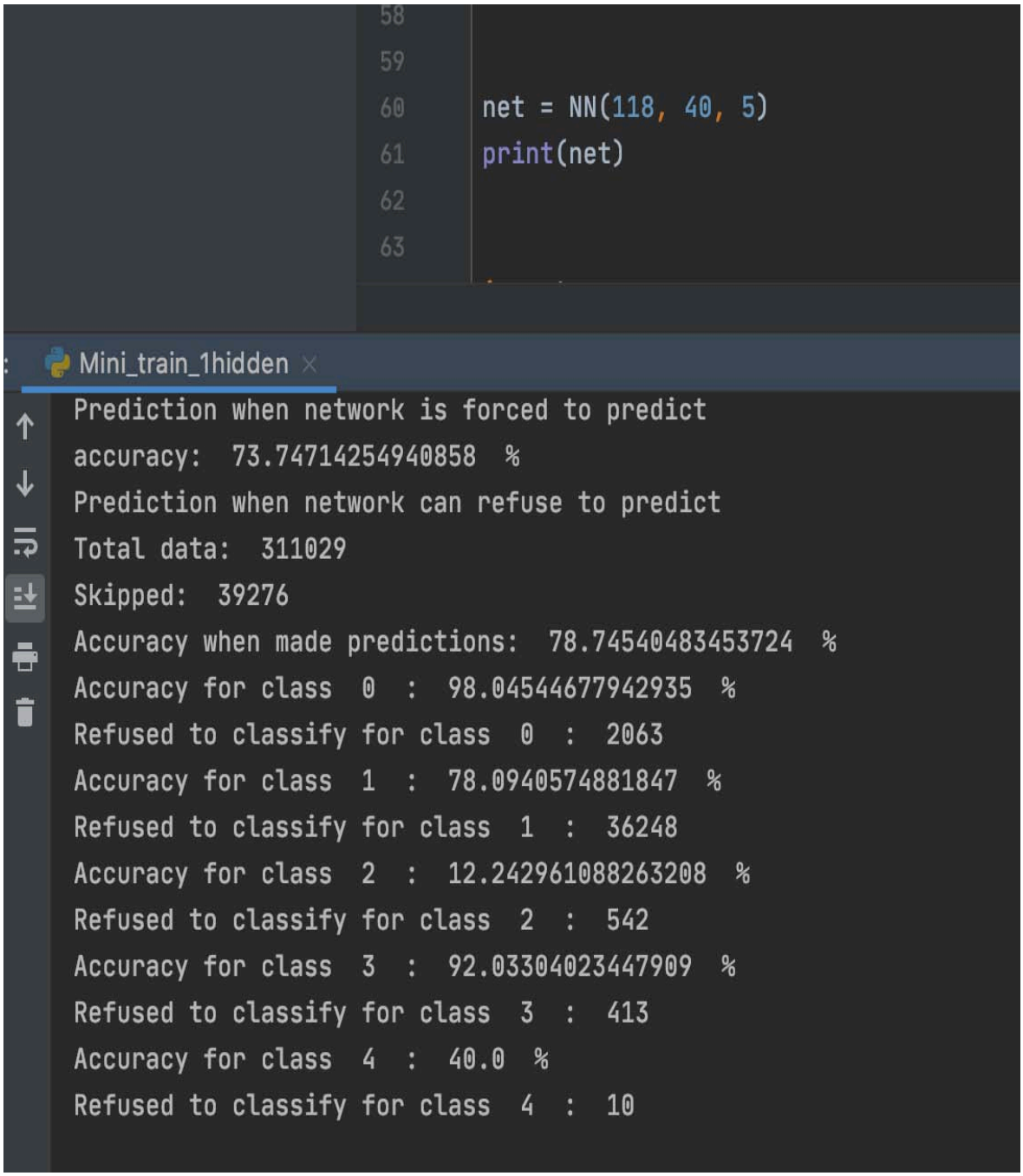


Fig. 12: Mini dataset_1, 1-hidden layer, 40 neurons.

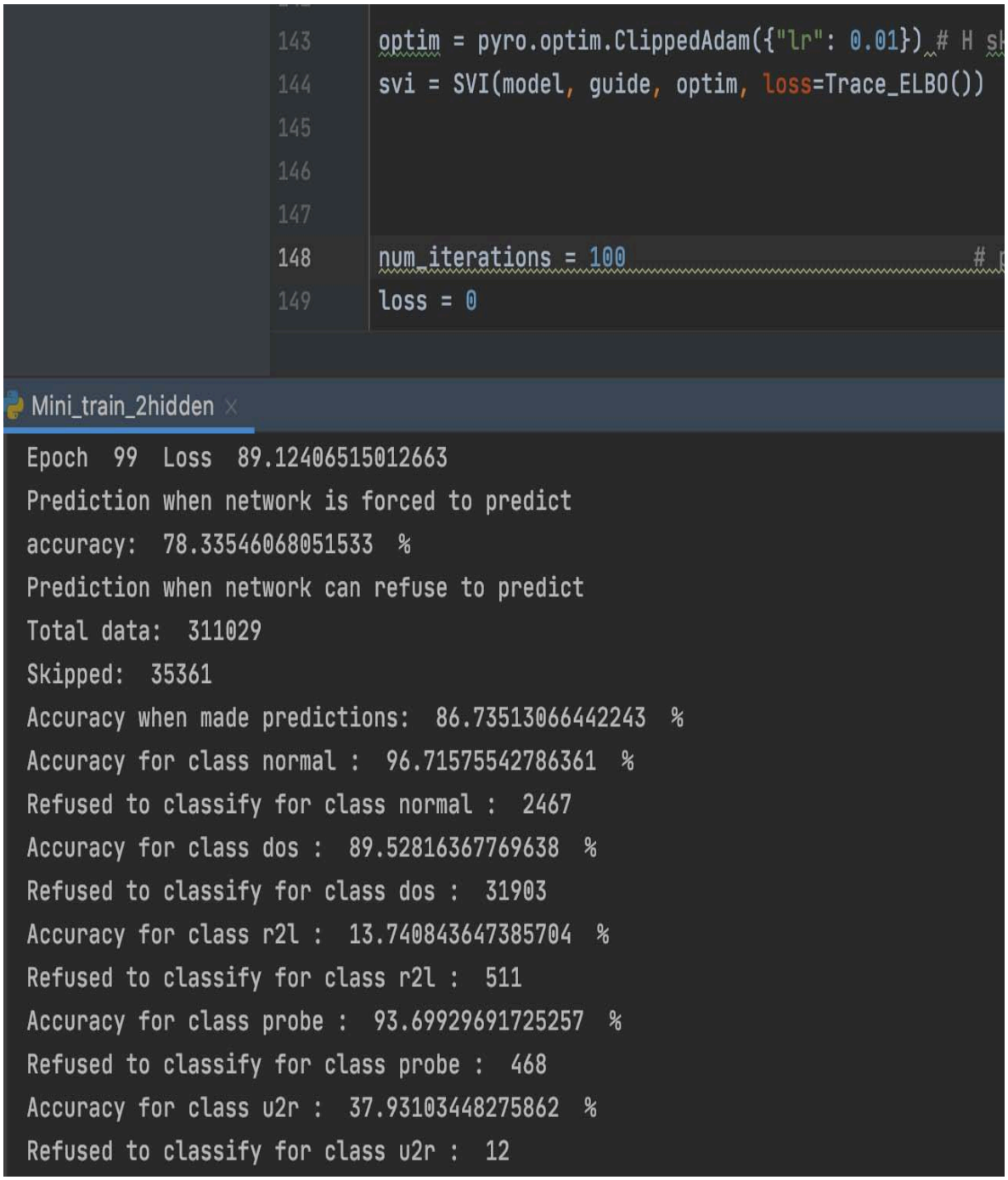


Fig. 13: Mini dataset_1, 2-hidden layers, 30 neurons each.

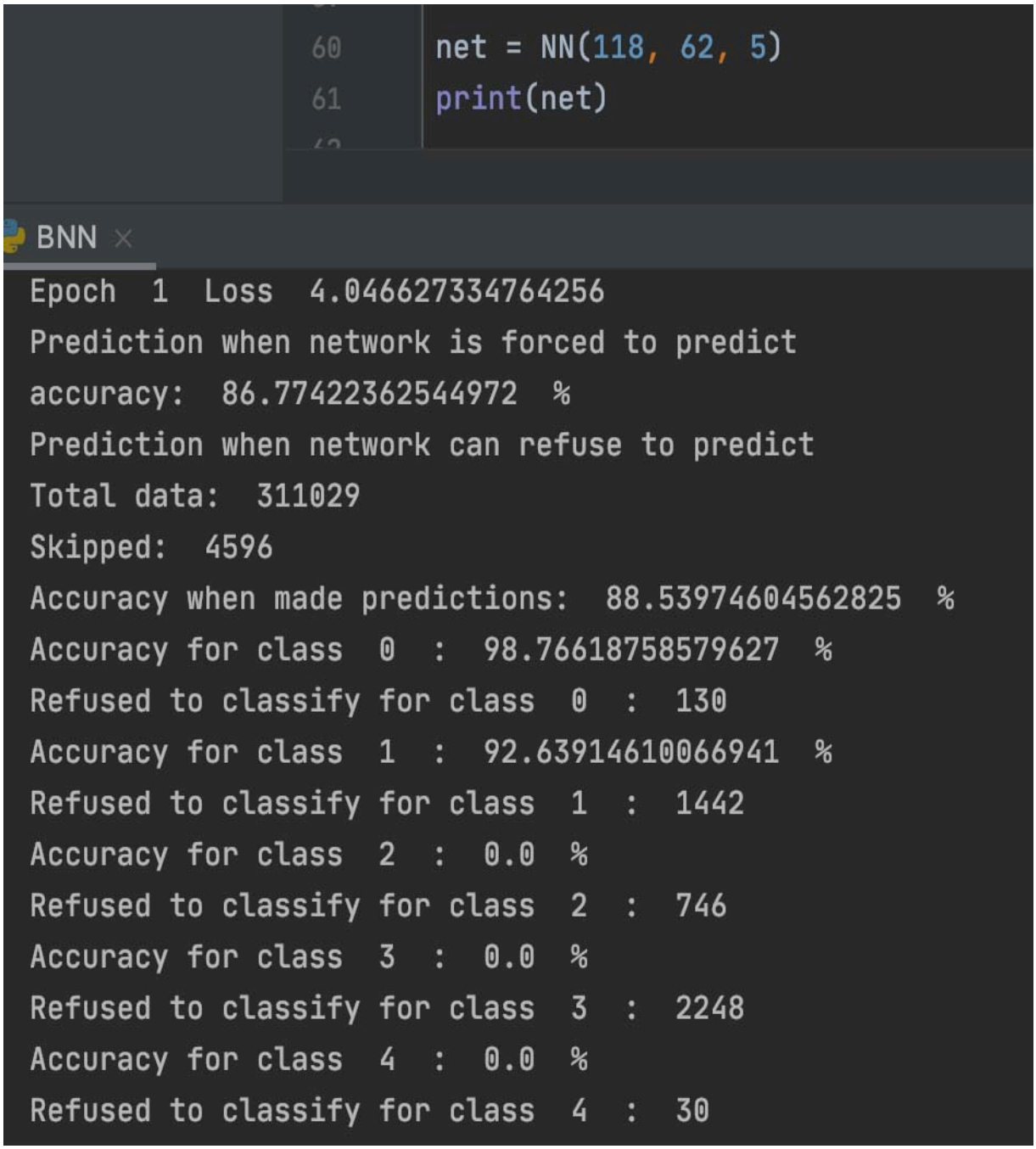


Fig. 14: Initial dataset, 1-hidden layer, 62 neurons.

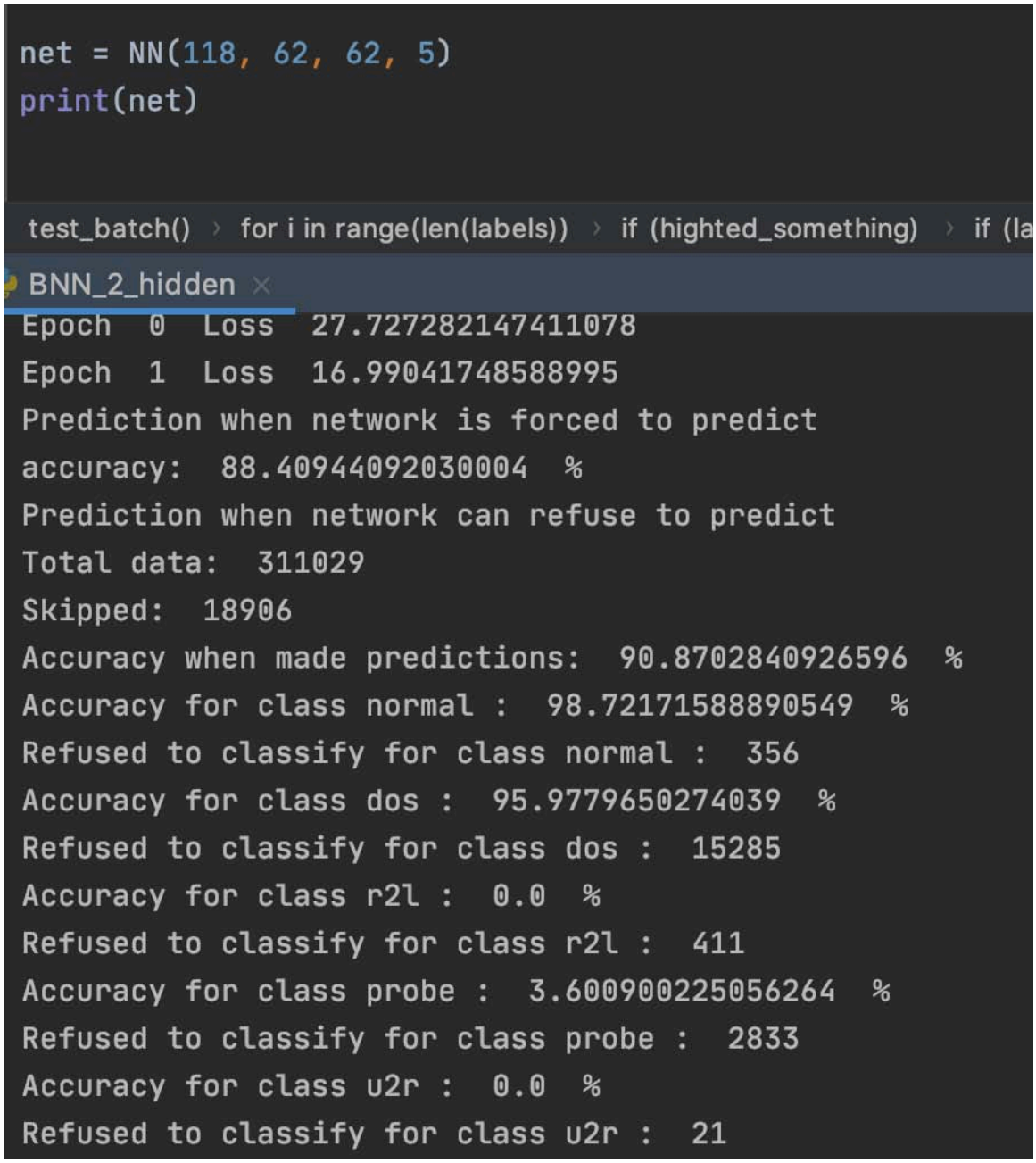


Fig. 15: Initial dataset, 2-hidden layers, 62 neurons each.

4. Discussion

The algorithm used in this thesis is not often applied in the development of intrusion detection systems, thus triggering the idea of this experiment. Additional purpose for conducting this experiment was to assess how testing results would be affected when the training set volume is decreased to an extreme.

As described in detail in the above chapters, the irregularity of this approach is based on the fact that many different small datasets have been used for training but tested in the initial large test set. The outcome was positive since the results from the small datasets manage to approach the ones from the large dataset and in some cases to overpass them as shown in charts 2-11.

Chart 2: Max accuracy achieved by the four different datasets.

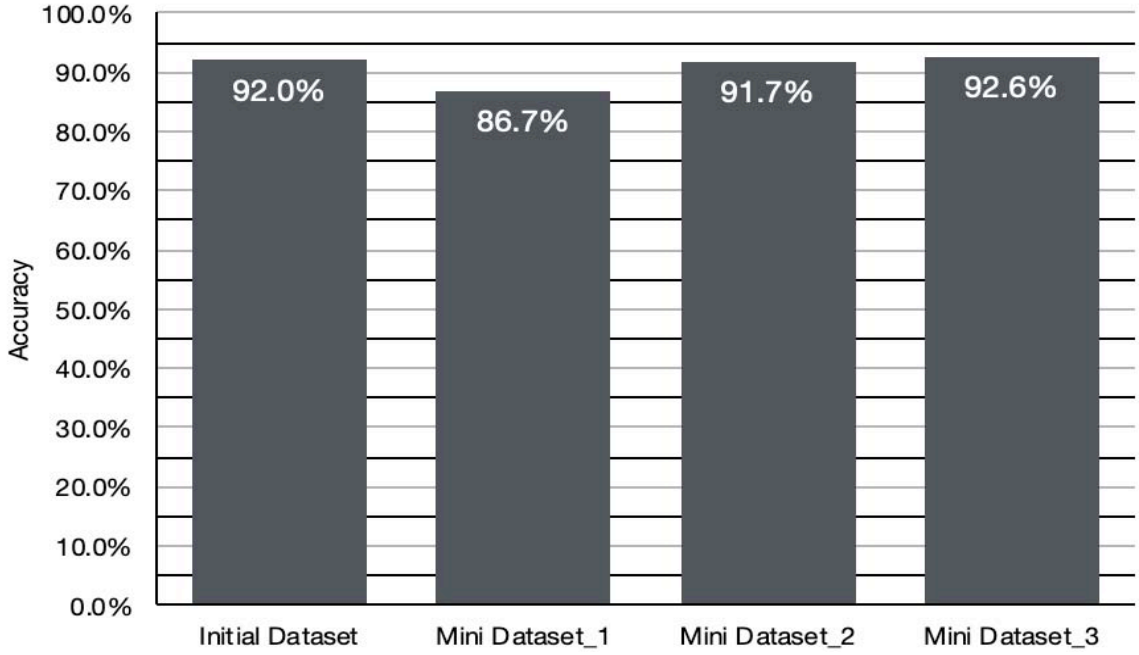


Chart 3: Accuracy of initial dataset with one hidden layer.

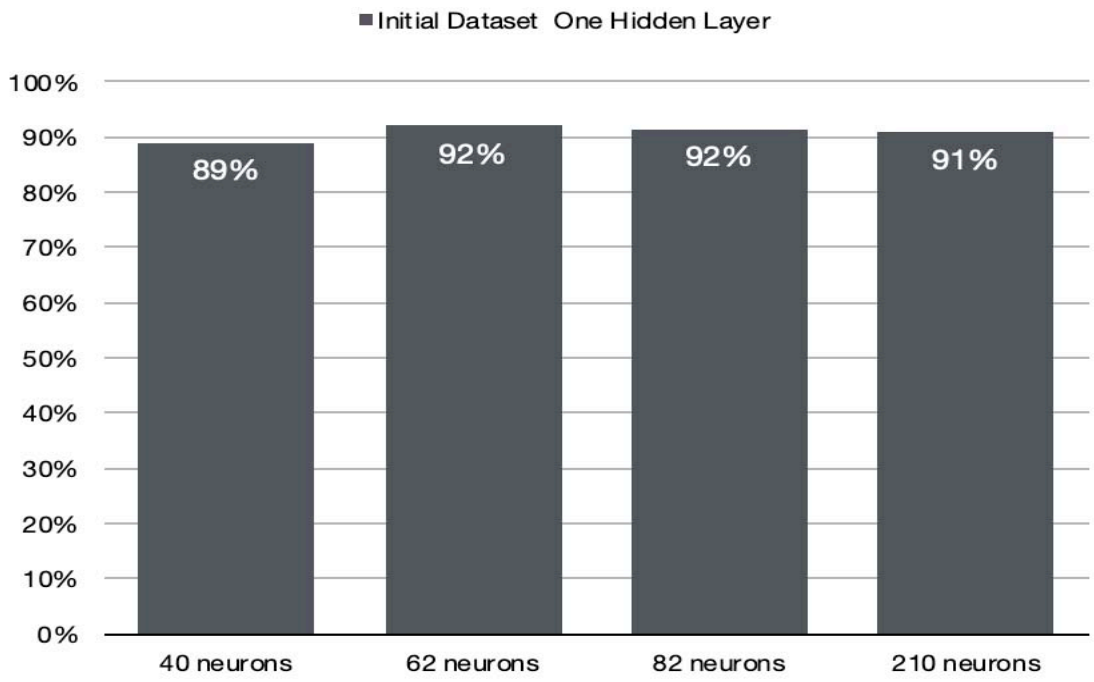


Chart 4: Accuracy of initial dataset with two hidden layers.

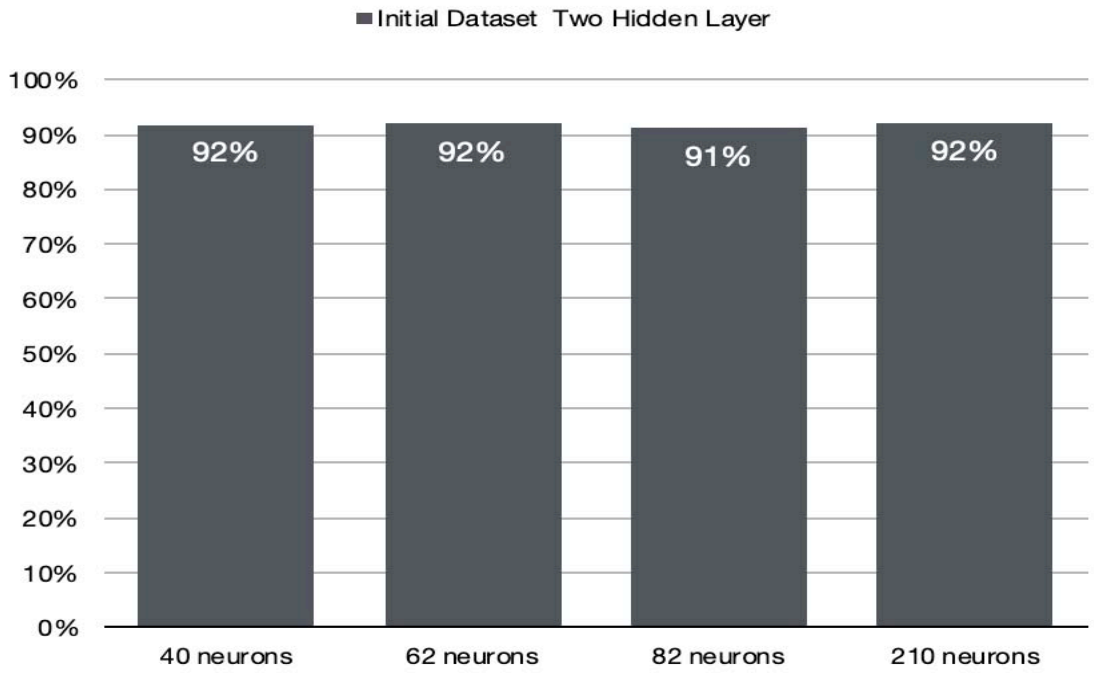


Chart 5: Accuracy of mini dataset 1 with one hidden layer.

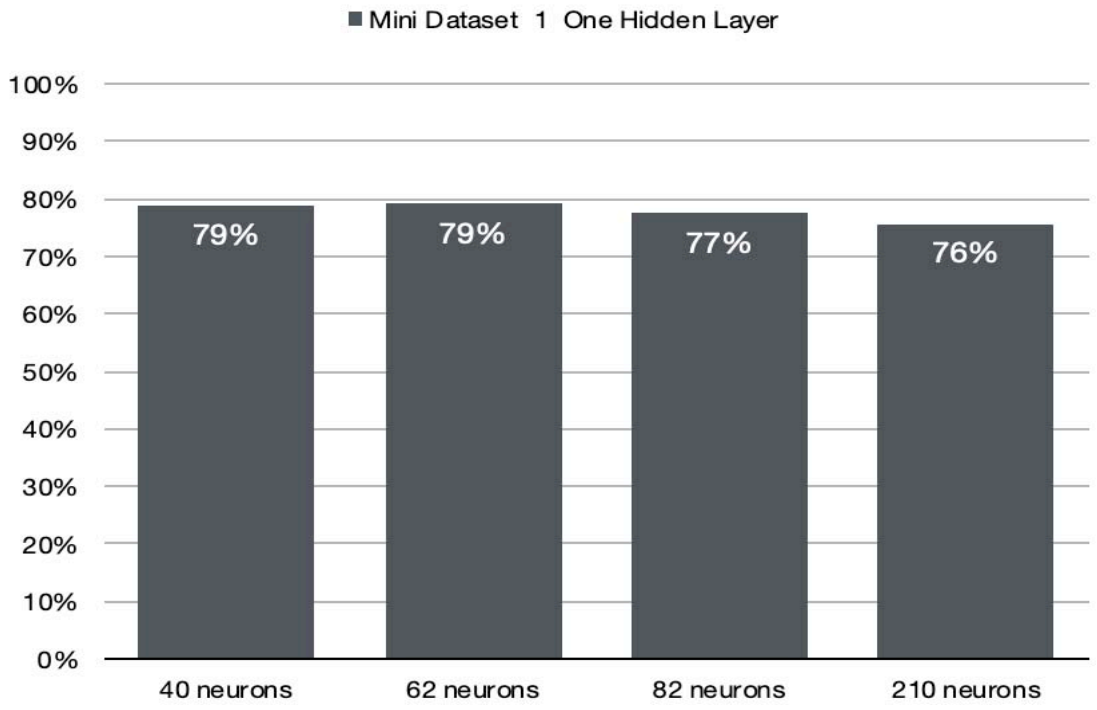


Chart 6: Accuracy of mini dataset 1 with two hidden layers.

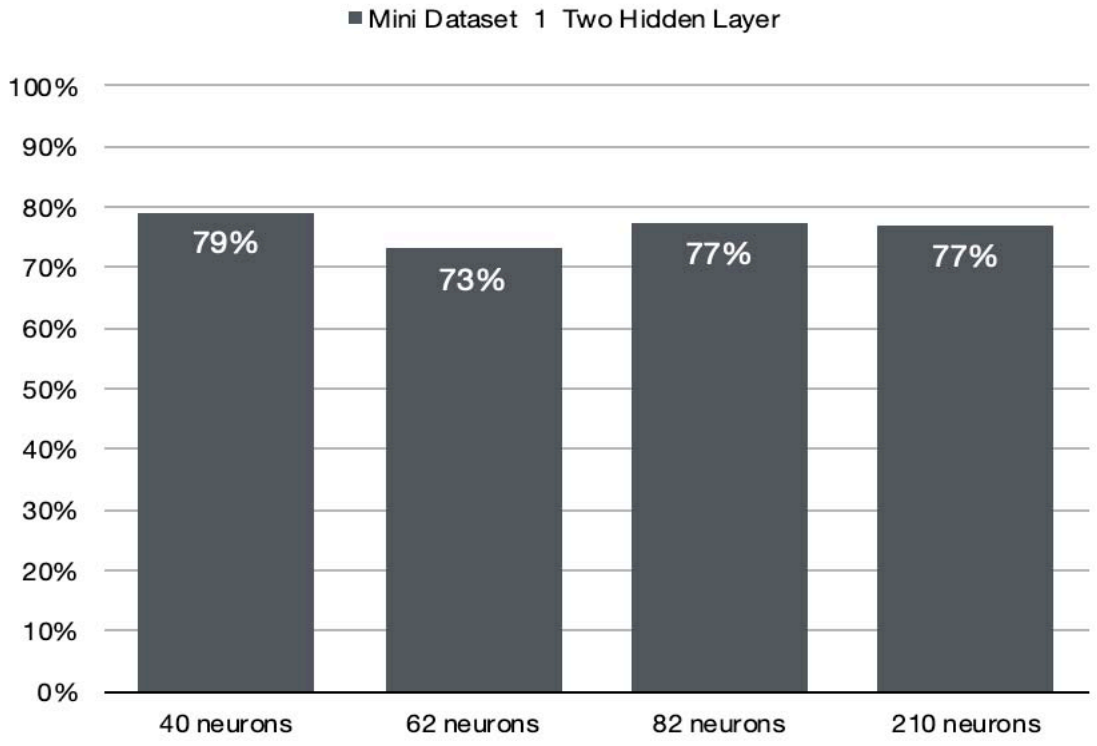


Chart 7: Accuracy of mini dataset 2 with one hidden layer.

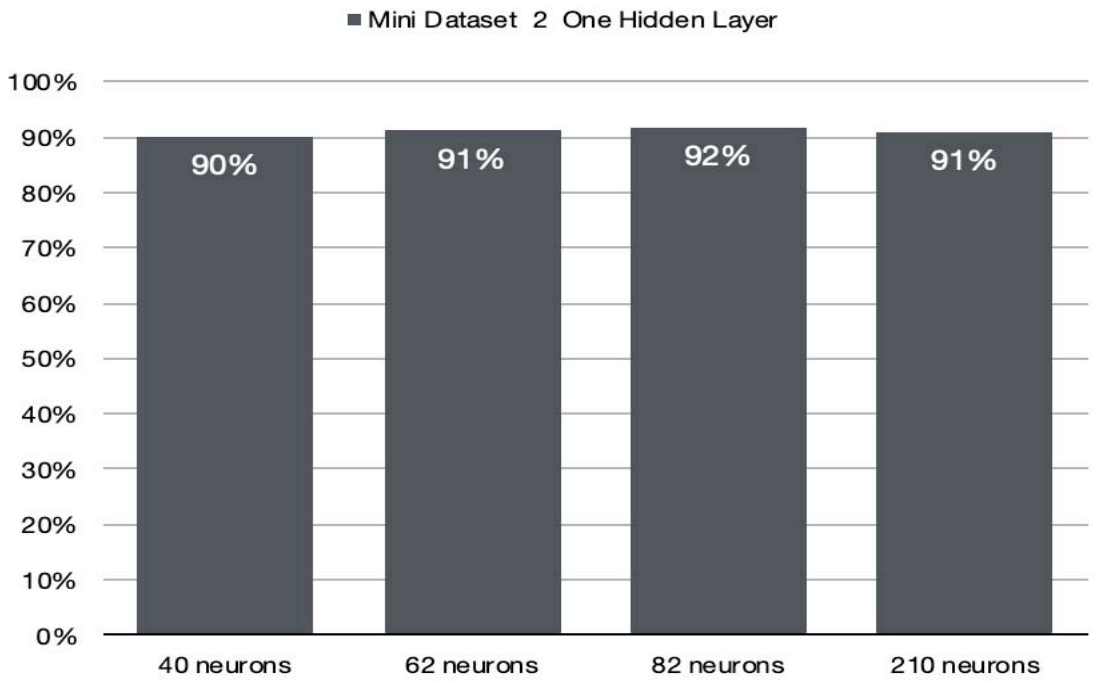


Chart 8: Accuracy of mini dataset 2 with two hidden layers.

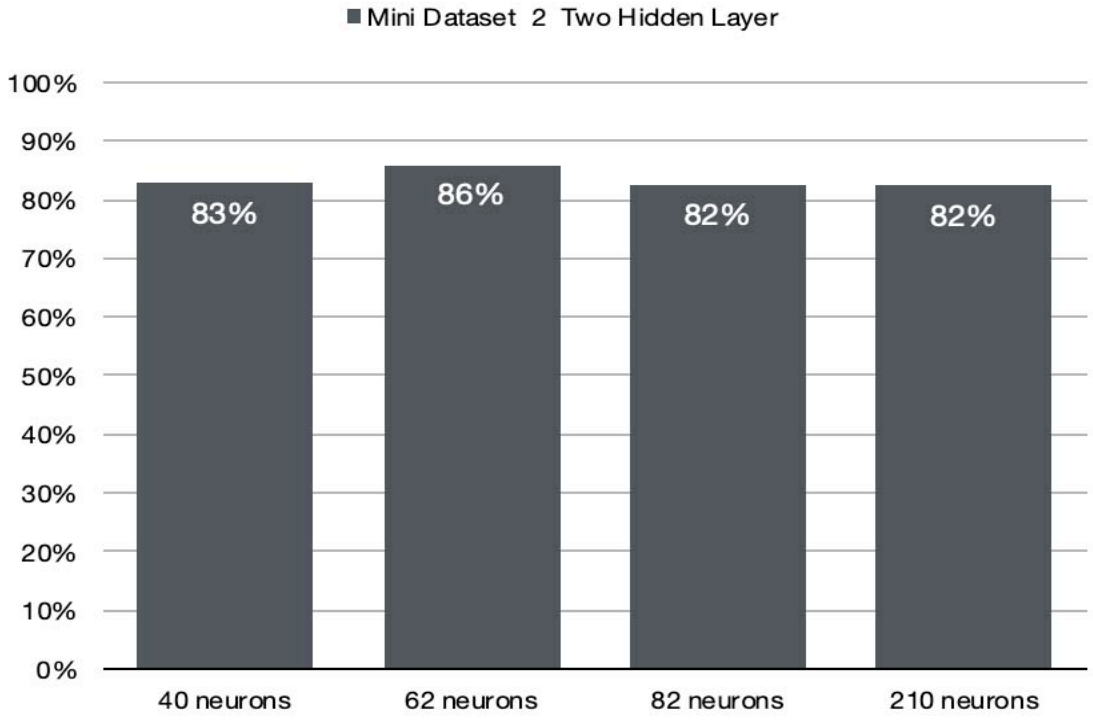


Chart 9: Accuracy of mini dataset 3 with one hidden layer.

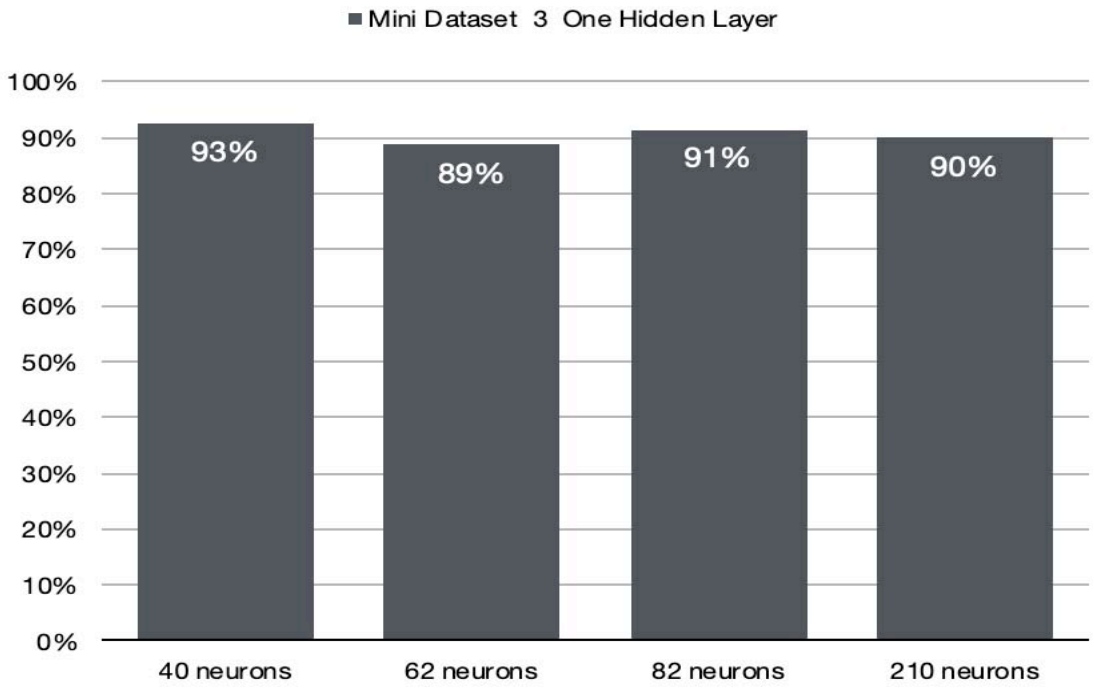


Chart 10: Accuracy of mini dataset 3 with two hidden layers.

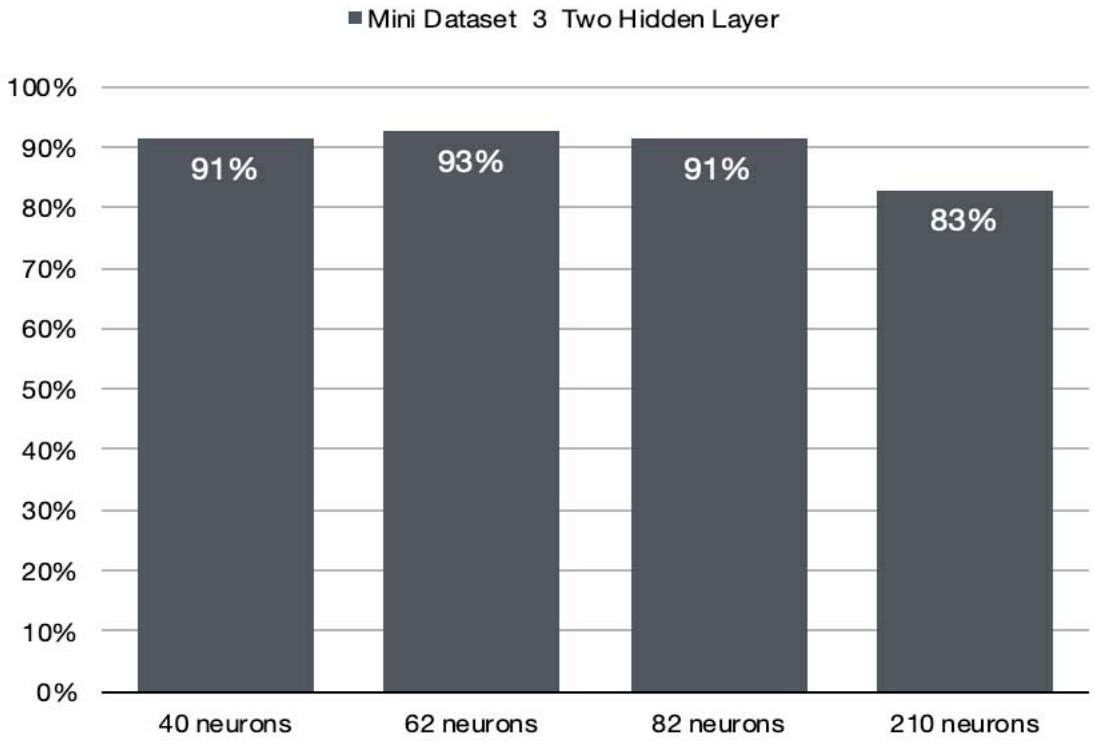
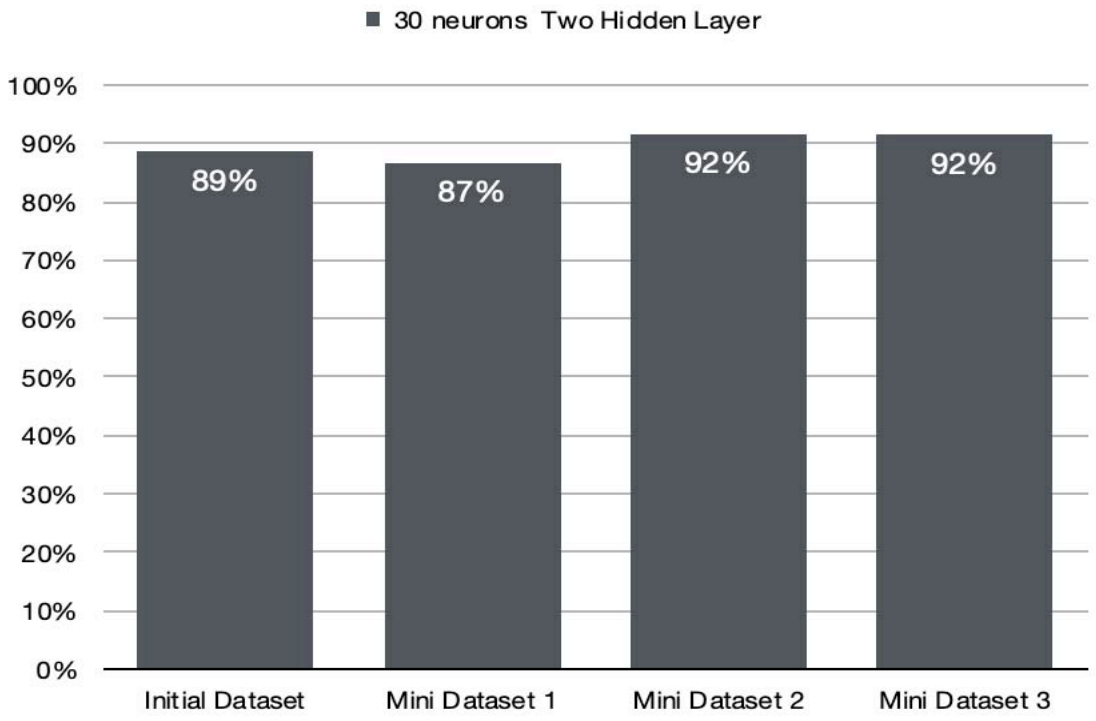


Chart 11: Accuracy of all datasets with two hidden layers of 30 neurons each.



A significant observation was that in the large dataset, classes that were under represented such as R2L, Probe and U2R could either not be classified correctly, or they were classified to an extremely low percentage. In contrast to the small datasets, where the same classes were at some extend correctly classified.

An additional point of interest was the fact that results of enhanced accuracy were achieved by exploiting the integrated ability of Bayesian neural networks to deny an answer.

Furthermore, an advantage was observed in that the convergence of the algorithm was rapid. In the large dataset a total of three epochs were sufficient to complete the training. In the small datasets, completion was achieved after ten to fifteen epochs.

Due to the structure of this experiment, comparison of results with other references can be done only for the large dataset. In this respect the Intrusion Detection System as constructed in this thesis while achieving good results, couldn’t overpass results provided from other approaches. Indicatively, some of the highest accuracies achieved as presented by Kruti Choksi et al. [10] are close to a 100% whereas in this thesis it was 92%.

5. Conclusion

In overall an Intrusion Detection System constructed via Bayesian neural networks, shows promising results in the containment and better defence from cyber-attacks. However, there is still ample space for further improvement to be made in order to achieve the said level. Some additions to be considered could be *data feature selection*, *different optimization algorithms and architectures.*

As discussed in the above chapter several advantages occur from the method used in this thesis. However, the one that stands out is the fact that significantly small datasets produce better results than the initial one. In this respect, further adjustment of the Intrusion Detection System construction procedure could lead to results comparable to the highest ones achieved by other models up to now.

Finally, the proposed system cannot be applied as is for intrusion detection. However, it could prove to be a valuable tool in the creation of new datasets, where rare or unknown attacks could be gathered. In this way training of new Intrusion Detection Systems could be enhanced.

References

[1] Federal Bureau of Investigation, "2019 Internet Crime Report," Federal Bureau of Investigation, 2019.

[2] Hackmageddon, "Information Security Timelines and Statistics," [Online]. Available: https://www.hackmageddon.com/2020/04/14/q1-2020-cyber-attacks-statistics/. [Accessed 10 May 2020].

[3] Davey Winder, "Forbes," 18 January 2020. [Online]. Available: https://www.forbes.com/sites/daveywinder/2020/01/18/us-government-confirms-critical-zero-day-security-warning-for-windows-users/#47ce4c1a3212. [Accessed 3 May 2020].

[4] Techopedia team, "Techopedia," 22 September 2011. [Online]. Available: https://www.techopedia.com/definition/3988/intrusion-detection-system-ids. [Accessed 20 April 2020].

[5] Q. Niyaz, W. Sun, A. Javaid, and M. Alam, A Deep Learning Approach for Network Intrusion Detection System, *College of Engineering*, University of Toledo, 2015.

[6] Wikipedia, “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Intrusion_detection_system. [Accessed 3 May 2020].

[7] Y. &. M. M. Aung, “An Analysis of K-means Algorithm Based Network Intrusion Detection System,” *Advances in Science, Technology and Engineering Systems Journal*, vol. 3, pp. 496-501, 2018.

[8] Shailendra Kumar Shrivastava & Preeti Jain, “Effective Anomaly based Intrusion Detection using Rough Set Theory and Support Vector Machine ,“ *International Journal of Computer Applications*, (0975 – 8887), vol. 18– No.3, March 2011

[9] Nagaraju Devarakonda, Srinivasulu Pamidi, V. Valli Kumari, A. Govardhan, "Intrusion Detection System using Bayesian Network and Hidden Markov Model," *Elsevier BV*, vol. 4, pp. 506-514, 2012.

[10] Kruti Choksi, Prof. Bhavin Shah, Asst. Prof. Ompriya and Kale, "Intrusion Detection System using Self Organizing Map: A Survey," *Journal of Engineering Research and Applications*, vol. 4, no. 12, pp. 11-16, 2014.

[11] Nabila Farnaaz & M. A. Jabbar, “Random Forest Modeling for Network Intrusion Detection System,” Published by Elsevier B.V. , Procedia Computer Science 89, pp. 213 – 217, 2016.

[12] S. Devaraju & S. Ramakrishnan, “DETECTION OF ACCURACY FOR INTRUSION DETECTION SYSTEM USING NEURAL NETWORK CLASSIFIER,” *International Journal of Emerging Technology and Advanced Engineering*, Vol. 3, Special Issue 1, January 2013.

[13] Supervised Learning, “Supervised Learning,” [Online] Available: https://deepai.org/machine-learning-glossary-and-terms/supervised-learning [Accessed 21 April 2020].

[14] Christopher M. Bishop, “Pattern Recognition and Machine Learning,” Springer Science and Business Media, LLC, 2006.

[15] Martin Heller, “Semi-supervised learning explained,” InfoWorld, [Online] Available: https://www.infoworld.com/article/3434618/semi-supervised-learning-explained.html [Accessed 21 April 2020].

[16] Błażej Osiński and Konrad Budek, “What is reinforcement learning? The complete guide,” [Online] Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ [Accessed 22 April 2020].

[17] Neural Networks Fraud Solutions, “Neural Network Primer,” Issue 2.0 [Online] Available: ftp://ftp.dca.fee.unicamp.br/pub/docs/gudwin/referencias/neuralprimer.pdf [Accessed 20 April 2020]

[18] Duke University, “A Basic Introduction to Neural Networks,” [Online] Available: https://users.cs.duke.edu/~brd/Teaching/Previous/AI/Lectures/NN/neural.html [Accessed 20 April 2020].

[19] Maureen Caudill, “Neural Network Primer: Part I,” AI Expert, Feb. 1989.

[20] Marvin Minsky and Seymour Papert, “Perceptrons, An Introduction to Computational Geometry,” Published by The MIT Press, 1969.

[21] Mathworks, [Online] Available: www.mathworks.com [Accessed 3 June 2020].

[22] Leonardo Araujo dos Santos, [Online] Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/relu_layer.html [Accessed 3 June 2020].

[23] Clay-Technology World, [Online] Available: https://clay-atlas.com/us/blog/2020/02/03/machine-learning-english-note-relu-function/ [Accessed 3 June 2020].

[24] Research Gate, [Online] Available: https://www.researchgate.net/figure/ReLU-PReLU-and-Maxout-activation-functions_fig1_307889463 [Accessed 3 June 2020].

[25] Stack Exchange Data Science, [Online] Available: https://datascience.stackexchange.com/questions/53487/exponential-linear-units-elu-vs-log1ex-as-the-activation-functions-of-de [Accessed 3 June 2020].

[26] Raul Rojas, "Neural Networks, A systematic Introduction," Published by Springer-Verlag, Berlin, 1996.

[27] Lu, Yingjing & Yang, Runde. (2019). Not All Features Are Equal: Feature Leveling Deep Neural Networks for Better Interpretation. [Online] Available: https://arxiv.org/pdf/1905.10009v2.pdf [Accessed 6 May 2020].

[28] Hao Henry Zhou, Yunyang Xiong and Vikas Singh, "Building Bayesian Neural Networks with Blocks: On Structure, Interpretability and Uncertainty," University of Wisconsin-Madison, 2018.

[29] Anders Holst, "The Use of a Bayesian Neural Network Model for Classification Tasks," Stockholm, 1997.

[30] Variational Bayes and Beyond: Bayesian Inference for Big Data, [Online] Available: https://www.youtube.com/watch?v=DYRK0-_K2UU&feature=emb_logo [Accessed 8 May 2020].

[31] PyTorch, [Online] Available: https://pytorch.org [Accessed 12 May 2020].

[32] Pyro, [Online] Available: http://pyro.ai [Accessed 11 May 2020].

[33] KDD Cup 1999 Data, [Online] Available: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [Accessed 2 April 2020]

[34] GeeksforGeeks, [Online] Available: https://www.geeksforgeeks.org/python-tensorflow-nn-softplus/ [Accessed 3 June 2020].