Neural networks learning improvement using the K-means clustering algorithm to detect network intrusions

K. M. Faraoun[1], A. Boukelif[2]
Département d'informatique, Djillali Liabès University.
[1] Evolutionary Engineering and Distributed Information
Systems Laboratory, EEDIS
Sidi Bel Abbès - Algeria
Kamel_mh@yahoo.fr
Département d'électronique, Djillali Liabès University.
. [2]Communication Networks ,Architectures and Multimedia laboratory
University of S.B.A. Algeria
aboukelif@yahoo.fr

Abstract. In the present work, we propose a new technique to enhance the learning capabilities and reduce the computation intensity of a competitive learning multi-layered neural network using the K-means clustering algorithm. The proposed model use multi-layered network architecture with a backpropagation learning mechanism. The K-means algorithm is first applied to the training dataset to reduce the amount of samples to be presented to the neural network, by automatically selecting an optimal set of samples. The obtained results demonstrate that the proposed technique performs exceptionally in terms of both accuracy and computation time when applied to the KDD99 dataset compared to a standard learning schema that use the full dataset.

Keywords : Neural networks, Intrusion detection, learning enhancement, K-means clustering

(Received December 29, 2005 / Accepted April 17, 2006)

1 Introduction

Intrusion detection is a critical process in network security. Traditional methods of network intrusion detection are based on the saved patterns of known attacks. They detect intrusion by comparing the network connection features to the attack patterns that are provided by human experts. The main drawback of the traditional methods is that they cannot detect unknown intrusions. Even if a new pattern of the attacks were discovered, this new pattern would have to be manually updated into the system. On the other hand, as the speed and complexity of networks develop rapidly, especially when these networks are open to the public Web, the number and types of the intrusions increase dramatically. Hence, with the changing technology and the exponential growth of Internet traffic, it is becoming difficult for any existing intrusion detection system to offer a reliable service. From earlier research, we have found that there exists a behavioural pattern in the attacks that can be learned. That is why an artificial neural network is so successful in detecting network intrusions; it is also capable of identifying new attacks to some degree of resemblance to the learned ones. The neural networks are widely considered as an efficient approach to adaptively classify patterns, but their high computation intensity and the long training cycles greatly hinder their applications, especially for the intrusion detection problem, where the amount of treated data is very important.

Neural networks have been identified since the beginning as a very promising technique of addressing the intrusion detection problem. Many researches have been performed to this end, and the results varied from inconclusive to extremely promising. The primary premise of neural networks that initially made it attractive was its generalization property, which makes it suitable to detect day-0 attacks. In addition neural networks also posses the ability to classify patterns, and this property can be used in other aspects of intrusion detection systems such as attack classification, and alert validation. In this work, an attempt is made to improve the learning capabilities of a multi-layered neural network and reduce the amount of time and resource required by the learning process by sampling the input dataset to be learnt using the K-means algorithm. This paper is organized as follow: section 1 gives some theoretic background about the use of neural networks for intrusion detection and the k-means clustering technique, then describe the proposed technique of samples reduction. The section 2 presents the architecture of the used neural networks with the different used parameters. Section 3 summarizes the obtained results with comparison and discussions. The paper is finally concluded with the most essential points and possible future works.

2 Theory

2.1 Neural network models for IDS

A neural network contains no domain knowledge in the beginning, but it can be trained to make decisions by mapping exemplar pairs of input data into exemplar output vectors, and adjusting its weights so that it maps each input exemplar vector into the corresponding output exemplar vector approximately [1]. A knowledge base pertaining to the internal representations (i.e. the weight values) is automatically constructed from the data presented to train the network. Well-trained neural networks represent a knowledge base in which knowledge is distributed in the form of weighted interconnections where a learning algorithm is used to modify the knowledge base from a set of given representative cases. Neural networks might be better suited for unstructured problems pertaining to complex relationships among variables rather than problem domains requiring value-based human reasoning through complex issues. Any functional form relating the independent variables (i.e. input variables) to the dependent variables (i.e. output variables) need not be imposed in the neural network model. Neural networks are thought to better capture the complex pattern of relationships among variables than statistical models because of their capability to capture non-linear relationships in data.

The rules with logical conditions need not be built by developers as neural networks investigate the empirical distribution among the variables and determine the weight values of a trained network. A neural network is an appropriate method when it is difficult to define the rules clearly as is the case in the misuse detection or anomaly detection.

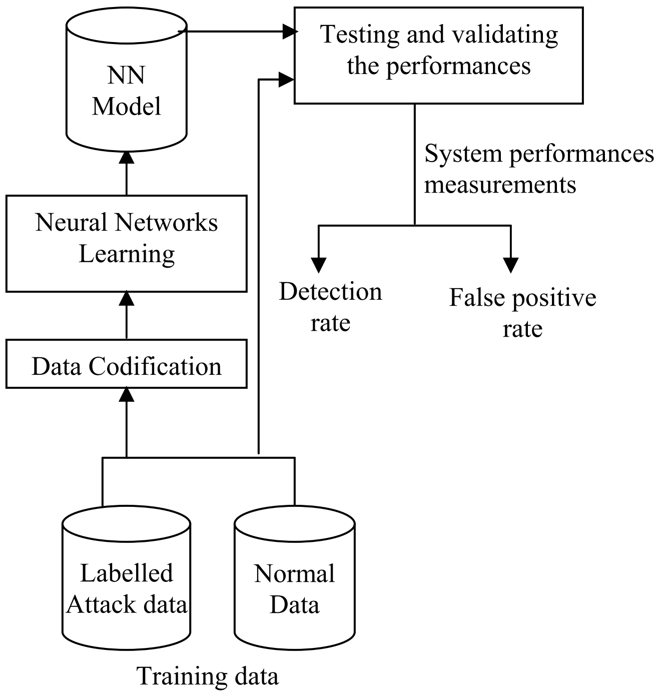

Figure 1: A generic form of a NN-base intrusion detection system

In order to measure the performance of an intrusion detection system, two types of rates are identified, false positive rate and true positive rate (detection rate) according to the threshold value of the neural network. The system reaches its best performance for height value of detection rate and low value of false positive rate. A good detection system must establish a compromise between the two situations.

A generic form of a neural network intrusion detector is presented in the Figure.1. The system use the input labelled data (normal and attack samples) to train a neural network model. The resulting model is then applied to the new samples of the testing data to determine the corresponding class of each one, and so to detect the existing attacks. Using the label information of the testing data, the system can compute the detection performances measures given by the false alarms rate, and the detection rate. A classification rate can also be computed if the system is deigned to perform attacks multi-classification

2.2 Data Clustering and k-means algorithm

2.2.1 Data clustering

Clustering of data is a method by which large sets of data are grouped into clusters of smaller sets of similar data. A clustering algorithm attempts to find natural groups of components (or data) based on some similarities. The clustering algorithm also finds the centroid of a group of data sets. To determine cluster membership, most algorithms evaluate the distance between a point and the cluster centroids. The output from a clustering algorithm is basically a statistical description of the cluster centroids with the number of components in each cluster. The centroid of a cluster is a point whose parameter values are the mean of the parameter values of all the points in the clusters. The k-means algorithm used in this work is one of the most non-hierarchical methods used for data clustering.

2.2.2 Algorithm description

The K-means [2] is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early grouping is done. At this point we need to re-calculate k new centroids as barycentre of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated, as a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more.

Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^j - c_j \right\|^2 \qquad (1)$$

Where $\left\| x_i^j - c_j \right\|^2$ is a chosen distance measure between a data point x_i^j and the cluster centre c_j, is an indicator of the distance of the n data points from their respective cluster centres. The general algorithm is composed of the following steps:

- Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
- Assign each object to the group that has the closest centroid.
- When all objects have been assigned, recalculate the positions of the K centroids.
- Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect. K-means is a simple algorithm that has been adapted to many problem domains. The proposed procedure is a simple version of the k-means clustering. Unfortunately there is no general theoretical solution to find the optimal number of clusters for any given data set. A simple approach is to compare the results of multiple runs with different k classes and choose the best one according to a given criterion, but we need to be careful because increasing k results in smaller error function values by definition, but also an increasing risk of over-fitting.

3 The proposed method

In the present work, the role of the k-means algorithm is to reduce the computation intensity of the neural network, by reducing the input set of samples to be learned. This can be achieved by clustering the input dataset using the k-means algorithm, and then take only discriminant samples from the resulting clustering schema to perform the learning process. By doing so, we are trying to select a set of samples that cover at maximum the region of each class in the N-dimensional space (N is the size of the training vectors). The input classes are clustered separately in such a way to produce a new dataset composed with the centroid of each cluster, and a set of boundary samples selected according to their distance from the centroid. Reducing the number of used samples will enhance significantly the learning performances, and reduce the training time and space requirement, without great loss of the information handled by the resulting set, due to its specific distribution. The Figure.2 illustrates an example of the application of this selection schema to a 2-dimentional dataset.

The number of fixed clusters (the k parameter) can be varied to specify the coverage repartition of the samples. The number of selected samples for each class is also a parameter of the selection algorithm. Then, for each class, we specify the number of samples to be selected according to the class size. When the clustering is achieved, samples are taken from the different obtained clusters according to their relative intra-class variance and their density (the percentage of samples belonging to the cluster). The two measurements are combined to compute a coverage factor for each cluster. The number of samples taken from a given cluster is proportional to the computed coverage factor. Let

A be a given class, to witch we want to apply the proposed approach to extract S sample. Let k be the number of cluster fixed to be used during the k-means clustering phase. For each generated cluster cl_i (i:1..k), the relative variance is computed using the following expression:

$$Vr(cl_i) = \frac{\frac{1}{Card(cl_i)} * \sum_{x \in cl_i} dist(x, c_i)}{\sum_{j=1}^{k} \left(\frac{1}{Card(cl_j)} * \sum_{x \in A} dist(x, c_j) \right)} \qquad (2)$$

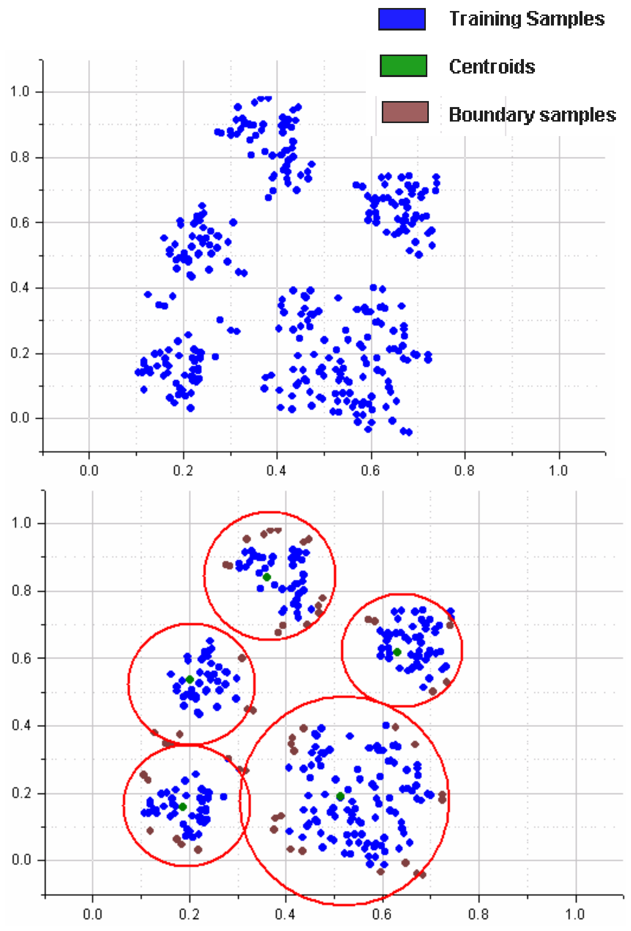

Figure 2: An illustrative example on the application of the proposed method to a 2-dimentional training set.

When Card(X) give the cardinality of a given set X, and dist(x,y) give the distance between the two points x and y. Generally, the distance between two points is taken as a common metric to assess the similarity among the components of a samples set. The most commonly used distance measure is the Euclidean metric which defines the distance between two points $x=(p_1,\ldots,p_N)$ and $y=(q_1,\ldots,q_N)$ from R^N as:

$$dist(x, y) = \sqrt{\sum_{i=1}^{N} (p_i - q_i)^2} \qquad (3)$$

The density value corresponding to the same cluster cl_i is computed like the following:

$$Den(cl_i) = \frac{Card(cl_i)}{Card(A)} \qquad (4)$$

The coverage factor is then computed by:

$$Cov(cl_i) = \frac{(Vr(cl_i) + Den(cl_i))}{2} \qquad (5)$$

We can clearly see that: $0 \leq Vr(cl_i) \leq 1$ and $0 \leq Den(cl_i) \leq 1$ for any cluster cl_i. So the coverage factor $Cov(cl_i)$ belong also to the [0,1] interval. Furthermore, it is clear that:

$$\sum_{i=1}^{k} Vr(cl_i) = 1 \quad \text{and} \quad \sum_{i=1}^{k} Den(cl_i) = 1 \qquad (6)$$

We can so deduce easily that:

$$\sum_{i=1}^{k} Cov(cl_i) = 1 \qquad (7)$$

Hence, the number of samples selected from each cluster is determined using the expression:

$$Num\_samples(cl_i) = Round(S*Cov(cl_i)) \qquad (8)$$

Using (8), the algorithm presented in the figure.3 will permit to select S sample from a class A clustered with the k-means algorithm into k cluster. The parameter ε serve to ensure that the selected samples are placed in separated regions, and are not duplicated. The choice of ε's value depend on the size of the cluster. We have proposed the following heuristic expression to compute an approximate value of ε:

$$\varepsilon = \frac{\underset{x \in cl_i}{Max}(dist(x, c_i))}{10} \qquad (9)$$

This expression is only an approximate heuristic. No theoretic background was used to determine the value of ε. The performances of the expression were evaluated experimentally. Finally, the resulting set of samples is then used to train the neural network.

When dealing with the intrusion detection problem, the proposed technique is applied only to the large classes. With the KDD99 dataset used in our experiments, the technique is applied to the class: normal, Dos, Probe and R2l. The U2R class is very small according to the other classes mentioned, so the totality of its samples is used during the learning process. The figure.4 illustrates the general operation schema of the proposed approach.

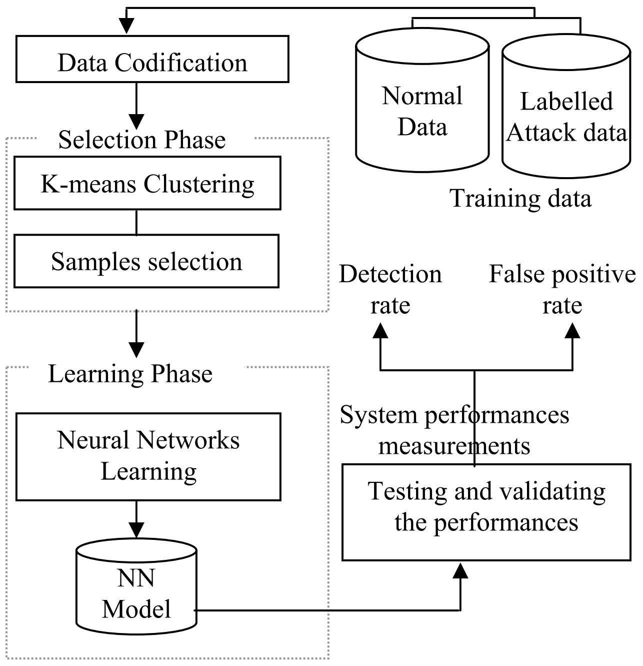

Figure 4: The general operating mechanism of the proposed method.

4 Datasets and experiments

Because the goal of this work is to study and enhance the learning capabilities of the neural network techniques for intrusions detection, the proposed method is compared to a classic neural networks implementation that use the full set of samples sampled from the KDD99 dataset [4], and witch contain 24788 sample. The use of the full '10% KDD' dataset containing 972780 samples is impracticable using the neural networks on any machine configuration. Even with the used subset, the experiments show that the learning process is very hard and take hours and hours to converge. The Table.1 lists the class's distributions of our used sets.

	Training Set		Testing Set	
Normal	11673	47.09 %	60593	19.48 %
DOS	7829	31.58 %	229853	73.90 %
PBR	4107	16.56 %	4166	1.34 %
R2L	1119	4.51 %	16347	5.25 %
U2R	52	0.24 %	70	0.02 %

Table 1: Distribution of the normal and attack records in the used training and testing sets.

At the first time, we tried to implement an intrusion classification system, to classify each intrusion to one of the learned attack classes (Dos, Prob, U2R, R2L), but the result demonstrate that a poor classification rate is obtained in this case. This can be interpreted by the fact that the power of the neural networks approach reside in their ability to discriminate the normal comportment from the intrusive one, and the discrimination between attack classes remain a hard task and give limited performances, especially for the classes U2R and R2L. The presented result demonstrates that considering the attacks classes as a single one improve significantly the detection rate with respect to the multi-classification approach. The new proposed technique was also implemented using the same principle, and the attack classes were merged in a single intrusion class, regrouping attacks categories with a relatively equivalent distribution.

```
Let A be the input class
k: the number of cluster
S: the number of samples to be selected (S ≥ k)
Sam(i): the resulting selected set of samples for the cluster i
Out_sam: the output set of samples selected from the class A
Candidates: a temporary array that contain the cluster points
and their respective distance from the centroid
i,j,min,x: intermediates variables
ε: Neiberhood parameter

1-Cluster the class A using the k-means algorithm into k
    cluster.
2-For each cluster cl_i (i:1..k) do
      { Sam(i) :={centroid(cl_i)};
        j:=1;
        For each x from cl_i do
              { Candidates [j].point :=x;
                Candidates [j].location :=dist(x, centroid(cl_i)) ;
                j:=j+1 ;
              };
        Sort the array Candidates in descending order with
         respect to the values of location field;
        j:=1;
        While((card(Sam(i)))<Num_samples(cl_i))
              and (j<card(cl_i)) do
                        {min:=100000;
                          For each x from Sam(i) do
                   {if dist(Candidates[j].point,x)<min
                      then min:= dist(Candidates[j].point,x) ;
                    }
                     if (min > ε) then
                     Sam(i):=Sam(i) ∪{Candidates[j].point};
                     j:=j+1;
                    }
        if card(Sam(i)) < Num_samples(cl_i) then
    repeat  {Sam(i):=Sam(i) ∪ Candidates[random].point
            }until (card(Sam(i)) = Num_samples(cl_i));
3-For i=1 to k do Out_sam:=Out_sam ∪ Sam(i);
```

Figure 3: The proposed samples selection algorithm

In the following, we describe the architecture of the neural networks used in the experiments with the relevant parameters. The next section details the obtained results for each implementation, and compares the performances achieved by each detection system.

Attributes in the KDD datasets had all forms :continuous, discrete, and symbolic, with significantly varying resolution and ranges. Most pattern classification methods are not able to process data in such a format. Hence, pre-processing was required before pattern classification models could be built. Pre-processing consisted of two steps: first step involved mapping symbolic-valued attributes to numeric-valued attributes and second step implemented scaling. In the present work, we have used the data codification and scaling used in [10]. All the resulting scaled fields belong to the interval [0, 1]

4.1 Network architecture used with the standard method

As indicated above, the first experiments were performed using a multi-layered neural network to classify the input data samples of the training set presented in the Table.1, to one of 5 classes :Normal, Dos, Prob ,U2r, R2l corresponding to the normal and intrusive possible situations. The used network is composed of 3 hidden layers containing 30, 15 and 30 neuron respectively, and has 41 input and 5 outputs. The neural network was designed to produce a value of 1.0 in the output node corresponding to the class of the current sample and the value of 0.0 for the other output nodes. When testing new samples with the network, the outputs can be any value from [0, 1] due to the approximate nature of the learning, so we consider the nearest output value to 1.0 as the activated output node.

In the case of two-category learning (normal and attack), the network has only one output neuron corresponding to the involved classes. The outputs activation is handled in the following way: during the learning phase, the output value is set to 0 for normal samples, and 1.0 for the attack samples. During the test phase, the output value is rounded to the nearest value 0 or 1.0.

We have used the feed-forward backpropagation [3] as learning algorithm, the Table.2 show the set of parameters used during the learning process used for all the implementations presented in this work.

4.2 Network architecture used with the proposed method

Since the proposed schema use a reduced set of samples, the network architecture can be more trivial. We use only two hidden layers with 18 and 5 neurones respectively, an input layer of 41 neurones, and the output layer contain 1 neurone for normal and intrusive classes. The same parameters of learning are used as illustrated in the Table.2.

The described experiments were implemented using the MATLAB 7 environment, on a Pentium4 2.88 GHz, with 256 Mb of memory.

Parameter	**Name**
Network type	Feed-forward backpropagation
Number of inputs	41
Number of outputs	1 or 5
Hidden layers	3
Hidden layers size	15 or 30
Input and output ranges	[0,1]
Training function	TRAINGDX (training function that updates weight and bias values according to gradient descent momentum and adaptive learning rate)
Adaptation learning function	LEARNGDM (the gradient descent with momentum weight/bias learning function)
Performances function	MSEREG
Transfer function	TANGSIG
Training epochs	1000

Table 2: Set of parameters used to train the proposed neural networks

4.3 Clustering and selection parameters

As described above, the sampling algorithm has two parameters to be defined as inputs: the cluster number k of each class, and the number of samples to be extracted S. Different possible values were tested during the experiments to find a good compromise between the size of the resulting dataset and its coverage of the input classes' space. The Table.3 list the final chosen parameters for each class. The class U2R was totally selected, because it present a very small portion of the initial dataset (0.02% only).

Class	**Initial class size**	**Number of clusters k**	**Total selected samples S**	**Classe percentage**
Normal	11673	8	258	34.95 %
Dos	7829	7	195	26.42 %
Prob	41077	5	121	16.39 %
R2L	1119	6	112	15.17 %

Table 3. The selected clustering parameters used with the select samples from the initial dataset

The selected parameters were determined heuristically. For the intrusion classes, we have chosen the number of cluster according the number of attack types included in each class. We have tried also to choose an equivalent distribution of the total number of selected samples over the different classes to avoid that one class dominate the learning process.

5 Results and comparison

In the following, we present the different obtained results for the implemented approaches. The performances of each method are measured according to the detection rate and false positive rate calculated using the following expressions:

Detection rate

$$DR = 1 - \frac{\text{False negatives number}}{\text{Total Number of Attaks}} \qquad (9)$$

False Positive Rate

$$FP = \frac{\text{False Positives}}{\text{Total Number of normal connections}}$$

The classification rate computed for the first approach was calculated for each class using the following formula:

$$CR = \frac{\text{Number of samples classified correctly}}{\text{Number of samples used for training}} * 100 \qquad (10)$$

5.1 Result of the standard NN-classification method: multi-classification approach

As mentioned in the section 2.1, the proposed presented neural network architecture is trained using the dataset presented in the Table.1. When learning is achieved, the resulting neural network is benchmarked using the 'Corrected (Test)' containing 14 additional (unseen) attacks, and used by almost all the classification systems developed for the KDD99 dataset.

The Table.4 illustrate the obtained classification matrix. Figure.5 show the training error evolution during the learning epochs. Obtained performances are summarized in the Table.5. We can see clearly from the comparative table (Table.6) that the classification results are relatively poor with respect to the other mentioned approaches that gives better performances with less computation and time requirements.

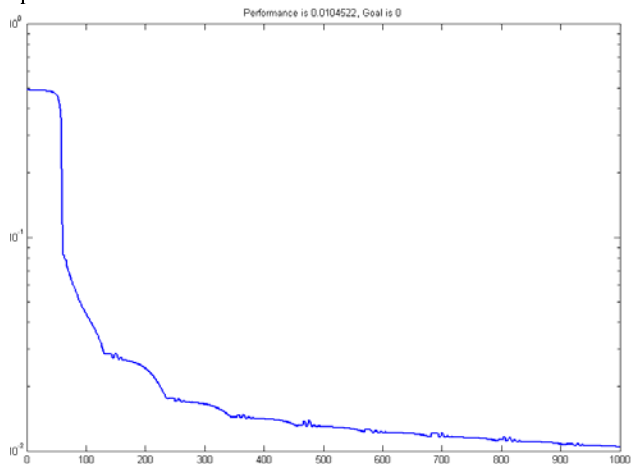

Figure 5: Training error evolution during the learning process

	Normal	**Prob**	**Dos**	**U2R**	**R2L**	**%**
Normal	58557	1348	467	14	207	**96.64 %**
Probe	235	3651	127	56	97	**87.65 %**
Dos	9387	938	220662	55	8008	**95.00 %**
U2R	36	9	7	6	12	**08.52 %**
R2L	10613	3956	8	159	1611	**0.9.85 %**
% Correct	**74.28 %**	**36.87 %**	**99.72 %**	**2.06 %**	**6.21 %**	

Table 4: Classification matrix obtained using the standard learning schema

Parameter	**Value**
Detection rate	91.90 %
False alarm rate	3.36 %
Execution run time	29 hour 51 minute
Classification rate	91 %

Table 5: Performances results for the multi-classification approach

Classification method	**Dos**		**Prob**	
	DR	**FP**	**DR**	**FP**
KDD cup Winner [5]	0.971	0.003	0.833	0.006
SOM map [6]	0.951	-	0.643	-
Linear GP [7]	0.967	-	0.857	-
Multi-classifier NNet	0.950	0.001	0.876	0.020
Gaussian classifier [8]	0.824	0.009	0.902	0.113
K-means clustering [8]	0.973	0.004	0.876	0.026
Nearest cluster algo. [8]	0.971	0.003	0.888	0.005
Radial basis [8]	0.730	0.002	0.932	0.188
C4.5	0.970	0.003	0.808	0.007

Classification method	**R2L**		**U2R**	
	DR	**FP**	**DR**	**FP**
KDD cup Winner [5]	0.084	5E-5	0.123	3E-5
SOM map [6]	0.113	-	0.229	-
Linear GP [7]	0.093	-	0.013	-
Multi-classifier NNet	0.085	0.026	0.098	9E-4
Gaussian classifier [8]	0.096	0.001	0.228	0.005
K-means clustering [8]	0.064	0.001	0.298	0.004
Nearest cluster algo. [8]	0.034	1E-4	0.022	6E-6
Radial basis [8]	0.059	0.003	0.061	4E-4
C4.5 decision tree [8]	0.046	5E-5	0.018	2E-5

Table 6: A comparative summary of the detection rates for each attack class

5.2 Result of the standard NN-classification method: 2-category classification approach

When we consider all the attacks as one category, the intrusion detection problem can be handled with the network proposed above (in the section 2.1) with one output handling the normal and intrusive classes. The learning is sill very slow and convergence is difficult, but the detection rate is significantly enhanced compared to the multi-category learning approach. The Table.7 summarizes the obtained performances results.

Parameter	Value
Detection rate	93.02 %
False alarm rate	1.5 %
Execution run time	22 hour 38 minute

Table 7: Performances results for the NNet 2-category approach

5.3 Result of the proposed classification method

Using the output set of samples obtained form the clustering phase, we construct a new training set composed by the normal samples, and the grouped attacks samples labelled as intrusive. The resulting set is presented to the neural network described above (section 2.2). The Table.8 summarizes the obtained performances results. Table.9 give the detailed description of the detection rate for each attack class from the used dataset. Figure.6 show the ROC curve [9] of the detection rate according to different vale of detection threshold δ. This parameter is used to control the output of the network, and determine from witch value we consider it as intrusion.

Parameter	Value
Detection rate	92 %
False alarm rate	6.21 %
Execution run time	28m 21s

Table 8: Performances results for the k-means based NNet approach

	Samples	Attacks detected
Normal	60593	6.21 % (False Alarm)
Dos	229853	97.23 %
Pbr	4166	96.63 %
R2L	16347	30.97 %
U2R	70	87.71 %

Table 9: Detailed detection rate for the learned classes

The obtained results demonstrate that we can achieve relatively the same detection performances with very less computation resources and time. The Table.10 compare the obtained performance with the full learning method and the clustering-based proposed one. It is clear that our goal to reduce the computation requirement is achieved.

	Standard NNet detection	K-means learning based detection
Detection rate	93.02 %	92 %
False alarm rate	1.5 %	6.21 %
Execution run time	22 h 8 m	28m 21 s
Training samples	24788 sample	738 sample

Table 10: Comparison of the obtained performances between the proposed methods

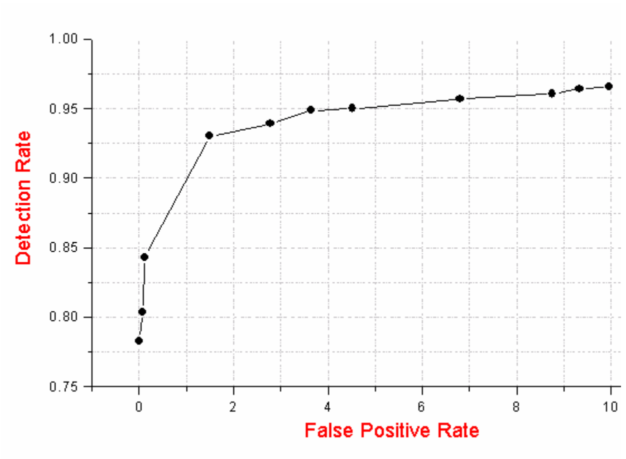

Figure 6: ROC curve for different value of the δ threshold parameter

6 Conclusion and Future Work

In this work, we study the possible use of the neural networks learning capabilities to classify and detect network intrusions from a collected dataset of network traffic trace. A multi-layered neural network was used with a backpropagation feed-forward learning algorithm. The intrusion detection problem is considered as a pattern recognition one, the neural network must learn to discriminate between the attack and the normal patterns. The experiments show that the neural networks are more suitable for 2-category classification problem, the discrimination between attacks classes remain a hard task. Since the high computation intensity and the long training cycles are the main obstacle to any neural networks IDS, we propose a new learning schema to reduce the amount of used samples using a k-means clustering algorithm. The input data are automatically clustered to a fixed number of clusters and the new samples set is constructed with the centroids of the obtained clusters and their relative boundaries, this will permit to give a maximum coverage of the initial space region occupied by the class data. The technique is independent of the dataset and structures employed, and can be used with any real values training dataset.

The proposed system is shown to be capable of learning attack and normal behaviour from the training data and make accurate predictions on the test data, in very less runtime, and with reasonable computation requirements. According to the obtained results, it can be asserted that substantial improvements of the NN-IDS performance are feasible, even if other classification methods can perform better.

In terms of future work, more work must be performed to find an optimal way to determine the number of used clusters and selected samples of each class. This work use only heuristics and trays to determine these parameters. A statistical sturdy of the information distribution of the information in each class seem to be a good appropriate approach.

References

[1] Hecht-Nielsen, R. (1988). Applications of counter propagation networks. Neural Networks, 1, 131–139.

[2] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297

[3] E. M. Johansson, F. U. Dowla and D. M. Goodman, “Backpropagation Learning for Multilayer Feed-forward Neural Networks using the Conjugate Gradient Method", Int. J. Neur. Syst. 2, 291 (1992).

[4] KDD data set, 1999; http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, cited April 2003

[5] Levin I.: KDD-99 Classifier Learning Contest LLSoft’s Results Overview. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 67- 75

[6] Kayacik G., Zincir-Heywood N., and Heywood M. On the Capability of an SOM based Intrusion Detection System. In Proceedings of International Joint Conference on Neural Networks, 2003.

[7] Dong Song, Malcolm I. Heywood, and A. Nur Zincir-Heywood. "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection", IEEE Transactions on Evolutionary Computation, 9(3), pp 225-240, 2005

[8] Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context, Maheshkumar Sabhnani, Gursel Serpen, Proceedings of the International Conference on Machine Learning, Models, Technologies and Applications (MLMTA 2003), Las Vegas, NV, June 2003, pages 209-215.

[9] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In Proceedings Of 15th International Conference On Machine Learning, pages 445-453, San Francisco, Ca, 1998. Morgan Kaufmann.

[10] C. Elkan, “Results of the KDD’99 Classifier Learning”, SIGKDD Explorations, ACM SIGKDD, Jan 2000.