Computers & Security, 14 (1995) 435-448

0167-4048(95)00008-9

Refereed paper

Design of a neural network for recognition and classification of computer viruses

Anastasia Doumas[1], Konstantinos Mavroudakis[1], Dimitris Gritzalis[1,2] and Sokratis Katsikas[1]

[1]*Department of Mathematics, University of the Aegean, Samos GR-83200, Greece*
[2]*Department of Informatics, Technological Education Institute of Athens, Athens GR-12210, Greece*

A sufficient number of experiments has been conducted to ascertain that a neural network can be used as a component of a computer security system for the recognition and classification of computer virus attack. A set of attributes that describe the system activity and the behaviour of computer viruses has been identified. The error back propagation training algorithm and the self-organizing feature map have been studied. Several experiments were conducted using both algorithms, different learning parameters, and two different training sets. For each architecture, the size of the network with the best performance was estimated experimentally. Results indicate that both neural networks can discriminate input patterns, at almost the same level of accuracy. The number of neurons required for the solution of the specific problem using a multilayer perceptron network was smaller than the respective number for a self-organizing feature map network. Therefore, using back propagation, the training and the recall process were faster. In conclusion, neural networks were proved to be efficient and practical devices for computer virus recognition and classification, in certain environments.

Keywords: Computer security, Information systems security, Computer virus, Worm, Neural network, Intrusion detection, Virus recognition, Virus classification.

1. Introduction

The number of computer viruses developed to infect contemporary computer systems increases rapidly [1–3]. Users usually have limited knowledge about viruses. Thus, in the case of an intrusion, they cannot easily identify a virus without help; even less can they take appropriate countermeasures to deal with the problem [4]. It is therefore necessary to build systems that will be able to monitor the actions performed in a computer system, to detect which of these actions are malicious, and to decide if these malicious actions are due to the activity of a computer virus that has intruded the system. In this sense, a computer virus can be considered as another type of computer user.

Several models of intrusion detection systems have already been designed [5–9]. Most of today's intrusion detection systems are based on the idea of analysing the audit trails. The audit trails are sets of audit records gathered through a specific auditing mechanism. This auditing mechanism must act below the insecure parts of the system under monitoring.

Denning [6, 7] proposed a model for a real-time intrusion detection expert system. This model is independent of any particular system, application environment, system vulnerability or type of intrusion, and provides detection of penetrations and security violations. The audit records for this model are generated by the target system in response to actions performed or attempted by subjects (initiators of activity) on objects (resources managed by the system, e.g. files, programs, etc.). This type of audit record contains information about the subject of a specific action, the object, the action, information about an exception condition that is raised by the system on return and information about the usage of resources. This model includes profiles for representing the behaviour of subjects with respect to objects in terms of metrics and statistical models, and rules for acquiring knowledge about this behaviour from audit records and for detecting anomalous behaviour.

At the Computer Science Laboratory of SRI a similar expert system, the Intrusion Detection Expert System (IDES) [10, 11], was designed. The purpose of this model was to observe behaviour on a monitored system and to learn adaptively what is normal for an individual as well as a group of users and remote hosts. The main components of this model are subjects, objects, audit records, profiles, anomaly records and activity rules. The model tries to monitor the standard operations on a target system, without dealing with specific subjects or objects. Intrusion detection is based only on deviations from an expected behaviour, and security violations can be detected from abnormal patterns of system usage that can be represented by audit records. These audit records can describe, as in Denning's intrusion detection model, the actions performed or attempted by subjects on objects. IDES maintains a database of statistical profiles corresponding to the description of subjects' behaviour in terms of metrics and statistical models. Such an experimental anomaly detector based on statistical tests for abnormality, considering deviations from an expected behaviour, could be applied in the case of a virus attack.

In the approach discussed here, it is deemed that there is a mechanism that monitors the activity of a computer system (e.g. a computer network) and creates a set of audit records. An expert system module or a neural network module can analyse the audit trail to detect irregular activity on the system. If irregular activity is detected, a neural network that is responsible for computer virus classification is triggered, to specify which virus has infected the system.

Furthermore, this neural network module could be independently used as a computer virus classification and recognition module by a system that can help users to detect and classify viruses in case of attack [9]. Artificial Neural Networks

(ANNs)—after training by introducing specific attributes [1, 3] of virus behaviour—can recognize a wide variety of viruses and determine whether a set of actions constitutes virus behaviour.

This paper deals with the implementation of an ANN that is able to recognize whether a set of attributes of system activity implies the presence of a specific virus. Two algorithms are studied to find the solution for this kind of problem: the first is the Error Back Propagation training algorithm (BP) of Multilayer Perceptrons [12–15] and the other is the Self-organizing Feature Maps (SOFM) training algorithm [16–18]. In detail, the classification accuracy and the computational efficiency of a layered neural network trained with the BP training algorithm is compared to that of a classifier trained on the same data with the SOFM.

2. General ANN architectures

ANNs are an old field of research for computer scientists, but because of their inability to overcome theoretical barriers they were not used for several years. Many learning algorithms for neural networks have recently appeared [12, 13] and have stimulated the application of ANNs to new problem areas [9, 19–21].

An artificial neural network (ANN) can be defined [15, 22] as an interconnection of neurons, such that neuron outputs are connected through weights to all others, including sometimes themselves.

A neuron is a linear automaton which realizes a weighted sum of several inputs according to a set of weights, and then computes an activation or transfer function to obtain an output value, called activation of the neuron. Weights measure the degree of correlation between activity levels of the neurons they connect [23]. Neural networks are often arranged in layers of neurons.

Neural network models can be classified by their synaptic connection topologies and by how learning modifies their connection topologies [22]. A synaptic connection topology and the corresponding neural network is feedforward if it contains no closed synaptic loops. The neural network is a feedback or recurrent network, if its topology contains closed synaptic loops or feedback pathways.

In a feedforward network the output is an explicit function of the input. The input is propagated through the network and produces the output right away. The mapping of an input pattern x into an output pattern o involves no time delay between the input and the output.

In a contrast, feedback networks always need some kind of relaxation to reach their equilibrium state or attractor. In a third category of neural network architectures, neighbouring cells in a neural network compete [16, 17, 24–26] in their activities by means of mutual lateral interactions, and develop adaptively into specific detectors of different signal patterns. In this category, learning is called competitive or self organizing.

Neural networks learn or adapt when their synaptic topologies change. A system learns or adapts or 'self-organizes' when sample data change system parameters. In neural networks, learning means any change in any weight.

An unknown probability density function $p(x)$ describes the continuous distribution of input patterns x in the pattern space R^n, a few of which the neural system samples [22]. Learning seeks only to accurately estimate $p(x)$. There is no need for learning if $p(x)$ is already known. Sometimes a set of input patterns is divided into a number of decision classes or categories $D_1, D_2, \ldots, D_k$. In response to an input pattern in the set, a classifier is supposed to recall the information regarding class membership of the input pattern [15, 27, 28]. This process is called classification. If the network's desired response is the class number but the input pattern does not exactly correspond to any of the patterns in the set, the processing is

called recognition. When a class membership for one of the patterns in the set is recalled, recognition becomes identical to classification. Neural networks can operate as effective pattern classifiers. The decision classes may correspond to high probability regions of the probability density function $p(x)$. Class boundaries then correspond to low probability regions on the probability hypersurface.

3. BP learning in multilayer perceptrons

Multilayer perceptrons [12, 13, 15, 29–31] are feedforward networks with one or more layers of nodes between the input and the output layer. These layers, that do not have direct connection to the outside world, are called hidden layers.

The most important attribute of a multilayer feedforward network is that it can learn a mapping of any complexity [15, 32]. It can implement arbitrary complex input/output mappings or decision surfaces separating pattern classes. The network learning is based on repeated presentations of training samples. The trained network often produces impressive generalizations in applications where explicit derivation of mappings and discovery of relationships is almost impossible. This kind of network is usually trained with the error back propagation training algorithm.

The BP training algorithm is an iterative process in which an output error signal is propagated back through the network and is used to modify weight values so that the current least mean-square classification error is reduced. Because BP learning is so important, it is summarized below in terms of a step-by-step procedure.

Given are P training pairs $\{x_1, d_1, x_1, d_2, \ldots, x_p, d_p\}$, where x_p is an $(I \times 1)$ input pattern, d_p is a $(K \times 1)$ desired output vector, $p = 1, 2, \ldots, P$ and P is the total number of input patterns. The number of hidden neurons is $J-1$. The outputs y of the neurons of the hidden layer and the outputs o of the output layer are of dimension $(J \times 1)$ and $(K \times 1)$, respectively.

Step 1: Learning rate $n > 0$, momentum constant $a > 0$ and a maximum error value $E_{\max}$ are chosen. Weights of the output layer W and of the hidden layer V are initialized at small random values. W is $(K \times J)$ and V is $(J \times I)$. Also, $q \leftarrow 1$, $p \leftarrow 1$, $E \leftarrow 0$.

Step 2: The training steps start here. Input is presented and the layers' outputs computed: $x \leftarrow x_p$, $d \leftarrow d_p$, $y_j = f(v'_j x)$, for $j = 1, \ldots, J$, where v_j (a column vector) is the jth row of V and $o_k = f(w'_k y)$, for $k = 1, \ldots, K$, where w_k (a column vector) is the kth row of W.

Step 3: Error value is computed:

$$E = (1/2) \sum_{k=1}^{K} (d_k - o_k)^2 + E.$$

Step 4: Error signal vectors e_o and e_v of both layers are computed. Vector e_o is $(K \times 1)$ and e_v is $(J \times 1)$. The error signal terms of the output layer in this step are: $e_{ok} = (d_k - o_k) f_k(h_k)$, for $k = 1, \ldots, K$. The error signal terms of the hidden layer in this step are:

$$e_{yj} = f_j(h_j) \sum e_{ok} w_{kj},$$

for $j = 1, \ldots, J$.

Step 5: Output layer weights are adjusted: $\Delta w_{kj}(q) \leftarrow n e_{ok} x_i + a \cdot \Delta v_{ji} \cdot (q-1)$, for $k = 1, \ldots, K$ and $j = 1, \ldots, J$.

Step 6: Hidden layer weights are adjusted: $\Delta v_{ji}(q) \leftarrow n e_{yj} x_i + a \cdot \Delta v_{ji}(q-1)$, for $j = 1, \ldots, J$ and $i = 1, \ldots, I$.

Step 7: If $p < P$, then $p \leftarrow p+1$, $q \leftarrow q+1$ and go to Step 2; otherwise go to Step 8.

Step 8: The training cycle is completed. If $E < E_{\max}$, then terminate the training session. Output

weights are W, V, q and E. If $E > E_{max}$, then $E \leftarrow 0$, $p \leftarrow 1$ and initiate the new training cycle by going to Step 2.

4. Self-organizing feature mapping

Self-organizing feature mapping (SOFM) [16–18, 33] is based on a neural network that consists of two layers, an input layer and an output layer. The input layer is a dummy layer that feeds the output layer with the input patterns. Each neuron of the output layer is connected to every neuron of the input layer via a set of variable reference or codebook vectors $w \in R^n$.

The basic principle underlying SOFM is competitive learning; especially, Vector Quantization (VQ) [34] which is a classical method that produces an approximation to a continuous probability density function $p(x)$ of the vectorial input variable x, using a finite number of codebook vectors w_i, for $i = 1, 2, \ldots, k$ (where k is the number of neurons in the output layer for the SOFM case). Once the codebook is chosen, the approximation of x involves finding the reference vector w_c (winner) closest to x.

In the above principle, the spatial relationships of the resulting 'feature-sensitive cells' are not considered. All the cells act independently. Therefore the order in which they are assigned to the different domains of input signals is more or less haphazard, most strongly depending on the initial values of the w_i. Such systems are often called zero-order topology networks [18]. In the models called maps, on the other hand, the localized feature-sensitive cells respond to the input signals in an orderly fashion, as if some meaningful coordinate system for different input features, reflecting some topological order of events in the input signal space, were drawn over the ANN. Depending on the connectivity, such networks can then be named n-order topology networks. A second-order topology network is a planar, with a two-dimensional coordinate system defined over it.

Kohonen [16] has introduced a self-organizing algorithm that effectively produces a global ordering over the network, the latter often reflecting rather abstract relationships of the data. It is crucial to the formation of ordered maps that the cells doing the learning are not affected independently of each other, but as topologically related subsets, on each of which a similar kind of correction is imposed. Such selected subsets or blocks are defined as neighbourhood sets N_c around a neuron c.

The self-organizing feature map algorithm is explained below for a planar array of neurons $(K \times J)$. Given are P input patterns $\{x_1, x_2, \ldots, x_p\}$, where x_p is an $(n \times 1)$ input pattern, $p = 1, \ldots, P$ and P is the total number of input patterns. The weight vector of neuron i is denoted by $w_i \in R^n$. The Euclidean distance is used here as a more convenient measure of similarity between x and w_i.

Step 1: $t \leftarrow 0$. Learning rate $a(t) = 0$, neighbourhood radius $N_c(t) > 0$ and a maximum error value E_{max} are chosen. The weights W are initialized at small random values. W is $(n \times K \times I)$, $p \leftarrow 1$, $E \leftarrow 0$.

Step 2: Training steps start here. Input is presented. Find the winning neuron c for which the best match with input x is calculated according to:

$$||x - w_c|| = \min\{||x - w_i||\}.$$

Step 3: Error value is computed:

$$E = (1/2) \sum_{i=1}^{n} (w_i - w_{ei})^2 + E.$$

Step 4: Weights are adjusted according to (1st variant):

$$w_i(t+1) = \begin{cases} w_i(t) + a(t)[x(t) - w_i(t)], & \text{if } i \in N_c(t), \\ w_i(t), & \text{if } i \notin N_c(t), \end{cases}$$

for $i = 1, \ldots, n$.

A proper form for $a(N_i, t)$ might be $a(N_i, t) = a(t) \exp[-||r_i - r_c||^2/(2N_c^2(t))]$, where r_c and r_i are the position vectors of the winning cell and of the winning neighbourhood nodes, respectively, and $a(t)$ and $N_c^2(t)$ are suitably decreasing functions of learning time t. In the above equations $0 < a < 1$.

Step 5: If $p < P$, then $p \leftarrow p+1$, $t \leftarrow t+1$, and go to Step 2; otherwise go to Step 6.

Step 6: The training cycle is completed. If $E < E_{max}$, then terminate the training session. Output weights are W and E. If $E > E_{max}$, then $E \leftarrow 0$, $p \leftarrow 1$ and initiate the new training cycle by going to Step 2.

5. Computer virus coding

A computer virus can be defined [35] as a piece of code (program) containing a self-reproducing mechanism riding on other programs and which cannot exist by itself.

A neural network can be trained to recognize the behaviour of a virus. This behaviour can be described by a set of attributes that correspond with or are the same as the attributes that describe activity on a computer system in audit records.

In the recall phase, when a set of attributes of system activity is presented in the input of the neural network, the latter should be able to recognize whether the behaviour depicted in this set is the same or similar to the behaviour of a specific virus that the neural network has been trained with. Thus, such a neural network can be used to analyse (probably in association with an expert system) the audit trails that a mechanism which monitors the activities on a computer system running a DOS operating environment generates, and to decide about the existence of computer viruses. The Threat Description Language for viruses from the Computer Virus Catalogue from Virus Test Centre (VTC) [3] and the Virus Information Summary List [2] could provide a first coding of the input for a neural network.

The set of attributes, specific values for which are used as input to the neural network in order to describe the behaviour of a specific virus, is depicted in Table 1. Most of the attributes can take the symbolic values 'Yes', 'No', 'Unknown', that correspond to the numeric values 1, −1, 0 (or 1, 0, 0.5) respectively, excluding the attributes for which these values are defined explicitly.

There are two different training sets which were used in the training and the recall phase for both types of network. The first training set contains 49 computer viruses (Appendix A) described by 38 virus characteristics that are almost similar to the list of characteristics arranged in Tables 1, 2 and 3, that correspond to 48 different classes. The second set contains 301 viruses corresponding to 301 different classes described with the 40 virus characteristics. The 300 patterns represent 300 computer viruses (Appendix B) and the remaining pattern represents the information that no virus exists. Each pattern is coded according to the above attribute values.

6. Virus classification via back propagation

A commercially available software product (Neural Works) was used for the implementation of the error back propagation training algorithm. A sufficient number of training experiments has been made with both training sets.

The training experiments with the second training set showed the best and most interesting results.

TABLE 1. Numeric values of length size attribute

Length size	
Number of bytes	n
No	0
Undefined	−1
Varying	−3

TABLE 2. Virus attributes list

Attribute	Description
Program virus	The virus is a program virus.
COM files	The virus infects COM files.
EXE files	The virus infects EXE files.
COMMAND.COM file	The virus infects the COMMAND.COM file.
Extending	The virus extends the size of an infected program file.
Stealth techniques	The virus uses stealth techniques.
Self encryption	The virus uses self-encryption techniques.
Boot sector	The virus infects the partition table of the hard disk or the floppy disk boot sector.
Hard disk boot sector	The virus infects the DOS boot sector of a hard disk.
Floppy disk boot sector	The virus infects the boot sector of a floppy diskette.
Floppy only	The virus infects only the boot sector of a floppy diskette.
Boot corruption	The virus corrupts/overwrites the boot sector or the partition table.
Disk corruption	The virus corrupts all or part of disk.
Program/Overlay corruption	The virus corrupts program or overlay files.
Data corruption	The virus corrupts data files.
System hangs and crashes	The virus hangs up or crashes the system.
System reboots	The virus creates unexpected system reboots.
Disk problems	The infected system will experience disk problems.
Printer problems	The infected system will experience printer problems.
Ports problems	The infected system will experience ports problems.
Keyboard problems	The infected system will experience keyboard problems.
Resident	The virus installs itself in memory.
Overwrites	The virus overwrites the beginning of a file.
Spawning	The virus creates a companion file with the viral code.
File linkage	Directly or indirectly corrupts the file linkage.
Runtime slowdown	The virus affects system run-time operations.
Bad or lost sectors	The virus creates bad or lost sectors.
Format	The virus formats or overwrites all or part of disk.
Created files	Existence of associated files.
Hidden files	The virus creates hidden files.
Change size	The virus changes the size of an infected program file.
Length size	The number of bytes by which the size of the infected file will be increased.
Date/time change	The virus changes the date or the time of the infected file.
Visual display effects	The virus has some visual display effects.
Visus screen message	The virus displays a message on the screen.
Unexpected errors	The virus displays some unexpected errors.
Beep noise	The virus creates some beep.
Melody	The virus plays a melody.
Infection trigger	Events which trigger the viruses to infect.
Damage trigger	Events which trigger the viruses to begin their damage.

In the first training experiments the second training set contained 298 viruses instead of 301 viruses. Nine different multilayer perceptron networks were developed and tested with different values for the learning rate and momentum parameters (Table 4).

The multilayer perceptron with the best performance was that with one hidden layer and 27 hidden neurons. This network was tested with different values for the parameters of the learning rate, the momentum and the noise. These parameters are given in Table 5. After 250000 training

steps, the network correctly created 301 different classes.

7. Virus classification via SOFM

The best solution for the implementation of the SOFM was considered to be the development of a software tool in the C programming language. This program implements the two different variants of the self-organizing feature map training algorithm described above and this implementation is independent of the specific virus classification problem.

In the first variant of the self-organizing feature map algorithm, the learning rate α and the neighbourhood radius are decreased exponentially every time a predefined number of training steps (sweep) has been accomplished. For the adaptation of the weights w_{ij} a Mexican hat lateral inhibition type function (Gaussian), of the form of the function $a(N_i, t)$ defined in Section 4, has been used. The training phase lasts a specific number of epochs. The epoch (learning cycle) is defined as a complete presentation of the training set.

TABLE 3. Numeric values of infection and damage triggers

Infection trigger		Damage trigger	
Load/execute	−1·0	Load/execute	−1·00
Random	−0·7	Random	−0·75
Counter	−0·4	Counter	−0·50
No trigger	0·0	Infection time	−0·25
Other	0·4	Other	0·00
Boot process	0·7	No trigger	0·25
Any time	1·0	CTRL-ALT-DEL	0·50
		Boot process	0·75

According to the second variant of the self-organizing feature map, the training process is composed of two training phases. In the first training phase the initial (or coarse) ordering of the network activity is attained. Spatial concentration of the network activity on the cell (or its neighbourhood), best tuned to each input, is achieved. In the second phase (final ordering), further tuning of the best-matching cell and its topological neighbours to the present input is done. In both phases (initial and final), the learning rate α and the neighbourhood radius are decreased linearly every time a predefined number of training steps (sweep) has been accomplished. The adaptation of weights is based on the first formula which is defined in Step 4 of the SOFM algorithm. According to this formula, all weights of the output neurons that belong to the neighbourhood are defined according to the neighbourhood topology, that can be either circular or rectangular, and the neighbourhood radius $N_c(t)$ around the winning neuron c.

TABLE 4. Results of training experiments with back propagation

Network	Number of viruses	Hidden units	Final error	Training steps	Noise
1st	298	70	0·000020	245064	
2nd	298	50	0·000027	200001	
3rd	298	30	0·000030	225074	
4th	301	50	0·000006	184945	
5th	301	30	0·000010	157220	
6th	301	20	0·000170	234905	
7th	301	25	0·000090	250000	
8th	301	30	0·000009	195597	√
9th	301	27	0·000012	233211	√

TABLE 5. Learning parameters of the training experiment

Training steps	1–60000	60001–120000	120001–160000	160001–200000	200001–250000
Noise	10	0	10	0	0
Learning rate	0·9	0·9	0·4	0·4	0·2
Momentum	0·6	0·6	0·4	0·4	0·3

Three different diagrams are used to depict the state of the network for each training step. The first one is a Hinton-type diagram which represents the distances between the input patterns and the weight vectors of the output nodes. The smaller the distance, the larger the corresponding rectangle of the Hinton diagram and the closer the input pattern to the corresponding weight vector of the output node. The second diagram is a bar chart, representing each input pattern the accuracy with which a specific element of the weight vector of the winning node for this pattern approximates the corresponding element of this pattern. Each bar of this diagram represents the accuracy of a specific element. With this diagram we can examine how good a winning node is for a specific input pattern. The third diagram represents the following error

$$E = \int || x - w_c ||' p(x)\, dx$$

approximated by the RMS error

$$E_{\max} = (np \cdot n)^{-1} \sqrt{|| x_i - w_c ||^2},$$

where x is the current input pattern, w_c is the weight vector of the winning node that corresponds to this input pattern, np is the size of the training set and n is the dimension of the input patterns.

In the following subsections, the results of some training experiments for the computer virus classification problem, using the above variants of the self-organizing feature map algorithm, are given. These experiments were performed using only the first training set with the 49 computer viruses.

A training experiment using the second training set of 301 computer viruses has been developed on a self-organizing feature map neural network with a planar array of 40×40 output nodes. The second variant of the algorithm has been applied. The training was continued for 300 000 steps. At the end of the training process the network could not discriminate between the classes. It is estimated that a self-organizing feature map with at least 55×55 output nodes can create the appropriate clustering for the 301 viruses.

7.1 Experiments using the first variant of the SOFM

A sufficient number of training experiments (6) using the first variant of the self-organizing feature map has been done on a 386SX machine with co-processor. The experiment with the best results is presented here. A network with a planar array of 30×30 output nodes was created.

The initial learning rate and neighbourhood radius were 0.5 and 20, respectively, each with a reduction factor of 0.98. The training process was continued for 4500 epochs. The sweep size was 900 training steps.

The final learning rate and neighbourhood radius had values very close to zero. The neural network had created 47 different clusters. The expected number of clusters is 48. The network had erroneously created one cluster for two input patterns that belong to two different classes. The RMS error value at the end of the training was 0.0019.

7.2 Experiments using the second variant of the SOFM

A sufficient number of training experiments (4) using the second variant of the self-organizing feature map has been done on a 486DX machine. The experiment described in this subsection is the one with the best results. A network with a planar array of 30×30 output nodes was created.

In the initial ordering, the initial learning rate and neighbourhood radius were 0.5 and 17, respectively, and the final learning rate and neighbourhood radius were 0.04 and 1, respectively. The initial ordering process was continued for 3000 epochs. The sweep size in this phase was 1 training step.

In the final ordering, the initial learning rate and neighbourhood radius were 0.04 and 1, respec-

TABLE 6. Learning parameters of the experiments with the SOFM training algorithm

1st variant		2nd variant		
			Initial ordering	Final ordering
Dimension	30 × 30	Dimension	30 × 30	
Learning rate	0·50	Initial learning rate	0·50	0·04
Learning rate reduction	0·98	Final learning rate	0·04	0
Neighb. size	20	Initial neighb. size	17	0
Neighb. size reduction	0·98	Final neighb. size	0	0
Epochs	4500	Epochs	3000	4811
Neighb. topology	Mexican hat	Neighb. topology	circular	circular

tively, and the final learning rate and neighbourhood radius were 0 and 1, respectively. The final ordering process was continued for 4811 epochs. The sweep size in this phase was 1 training step. The neighbourhood topology that was selected for both initial and final ordering was 'circular'.

The neural network had created 48 different clusters. These clusters correspond to the 48 classes of the input patterns (Appendix A). This experiment showed the best results. The RMS error value at the end of the training was 0.0000021.

The training parameters for the best training experiments described in the above paragraphs are given in Table 6. Finally, the results of these experiments are summarized in Table 7.

TABLE 7. Results of the SOFM training experiments

Variant	Dimension	Training steps	Final error
1st	30 × 30	220500	0·0019
2nd	30 × 30	382739	0·0000021

8. Practicality issues

The use of a neural network for computer virus recognition and classification may require a substantial level of computing overhead, thus decreasing the performance of a system in use. In particular, if its use is combined with an expert system, then the overhead is more serious.

However, whether the above observation is indeed valid or not depends heavily on a number of factors. In detail, it depends—*inter alia*—on:

- The assets to be protected. If the value of some of them is high enough, then the suggested architecture can often be considered as practicable, even if the performance of the system will be considerably degraded. In any case, the final decision on this issue must be based upon the findings of an appropriate risk analysis review.

- The risks and vulnerabilities of the system in use. In the case that the system in use provides input or operates in connection with a dependable system (e.g. nuclear plant control information system, medical diagnosis information system, etc.) the suggested solution can be very well adopted as practicable, even if it leads to considerably lower performance of the system in use.

- The connectivity of the system in use. In the case of a network or an internetwork topology, the risk analysis review may very well lead to the conclusion that the use of a neural network for virus recognition and classification is indeed practicable, since the damaging potential of malicious software structures (viruses, worms etc.) in such environments is very high. It is common knowledge that today's trend in connectivity is strongly towards these topologies.

- The computing power of the system in use. It can be expected that the performance of the

PCs of the forthcoming generations will be considerably higher than that of the machines available today. It is already a fact that the (performance) distance between 'PC' and 'Workstation' machines is no longer very great, if it exists at all.

- The way the neural network is realized. Hardware-based neural networks provide an effective means for high performance virus recognition and classification mechanisms.

As a result, it can evidently been concluded that even if the proposed solution may look—or be—impracticable in certain environments, it is probably practicable in several other cases. The definite answer on its practicability is dependent on context and can be positively ascertained after a thorough risk analysis review.

9. Conclusions

Several neural network architectures have been studied. The most appropriate of them, namely multilayer perceptrons and self-organizing feature map networks, have been selected for the problem of computer virus recognition and classification.

The behaviour of (DOS) computer viruses has been investigated and this behaviour has been described by simple parameters. These parameters were used for coding input data for several neural networks that were used in the training experiments presented in this paper.

A series of experiments using two different training sets, one with 49 computer viruses and 38 virus parameters and the other with 301 viruses and 40 characteristics, was performed. The results from the experiments with both the SOFM and multilayer perceptrons are promising.

The total number of neurons required for the construction of a multilayer perceptron network is smaller than the respective number for the SOFM. The number of neurons is one of the most significant factors for the training and the response (recall) time of a neural network. This is mainly why the training and the recall process were faster in the error back propagation training of the multilayer network.

The accuracy of the results is good with both training algorithms, but training with the error back propagation showed very good results after only a few training experiments.

In Table 8, a number of comparison results are given for the neural networks training on the generated training sets.

Although the user does not provide any class membership information to the SOFM, this neural network can derive common attributes from the input patterns and create clusters of 'similar' patterns. Consequently, using this neural network architecture, one can acquire knowledge about which class (cluster) an input pattern (computer virus information) belongs to, as well as knowledge about which are the nearest topological classes, i.e. classes with similar attributes. In contrast, using the error back propagation training algorithm, one knows only the class an input pattern belongs to.

In conclusion, we consider that the most efficient solution to the problem of computer virus classification and recognition with neural networks is the utilization of a multilayer perceptron neural network, trained with the error back propagation algorithm, because of its comparatively smaller training and recall (response) time. Evidently, if the neural network is going to the developed and

TABLE 8. Comparison of back propagation against SOFM training algorithm

	Back propagation	SOFM
Number of nodes	$27+301$	55×55
Training steps	233211	$\alpha400000$
Training time/step	t	$3{\cdot}6t$
Response time/step	t	$2{\cdot}7t$

used with a considerably faster machine (faster than 100 MHz), both architectures can be applied.

A drawback of the suggested architectures is that it is not difficult to be attacked if the mechanism is detected already in place. A good answer to this is to consider the mechanism as part of a well protected (trusted) system-software component (trusted kernel), which in turn leads to the need to design such a kernel. Furthermore, this consideration can be combined with a hardware-based realization of the proposed solution.

Another drawback of the neural network architectures is that, if we want to extend the neural network classifier ability by adding more computer virus categories (since it can fight most new viruses that fall into one of the existing categories), we have to start the training procedure from scratch and must sometimes add neurons to the network. However, again, neural networks are a good choice for implementation in hardware, where both learning and recognition would be considerably fast and secure.

References

[1] Audit Commission, *Opportunity Makes a Thief: An Analysis of Computer Abuse*, UK, 1994.

[2] P. Hoffman, *VSUM—Virus Information List*, USA, 1994.

[3] K. Brunnstein, et al., *Computer Virus Catalogue*, Hamburg Virus Test Centre, University of Hamburg, Feb. 1992.

[4] D. Guinier, Prophylaxis for 'virus' propagation and general computer security policy, *ACM SIGSAC Rev.*, 9(2) (1991) 1–10.

[5] K. Brunnstein, S. Fischer-Hubner and M. Swimmer, *Concepts of an Expert System for Virus Detection*, Hamburg Virus Test Centre, University of Hamburg, 1991.

[6] D. Denning, An intrusion-detection model, in *Proc. 1986 IEEE Symposium on Security and Privacy*, 1986, pp. 118–131.

[7] D. Denning, An intrusion-detection model, in *IEEE Trans. Softw. Eng.*, 13(2) (1987) 222–226.

[8] MORI, ICL and DTI, *Attitudes towards Information Security in top UK Companies*, Research study, UK, Oct. 1993.

[9] D. Guinier, Computer 'virus' identification by neural networks, *ACM SIGSAC Rev.*, 9(4) (1991) 49–59.

[10] D.E. Denning and P. Neumann, Requirements and model for IDES: a real-time intrusion detection expert system, *SRI Int.* (Aug. 1985).

[11] D. Denning, et al., A prototype IDES: a real-time intrusion-detection expert system, *SRI Int.* (Aug. 1987).

[12] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by back-propagating errors, *Nature*, 323 (Oct. 1986) 533–536.

[13] D. Rumelhart and J. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, 1988.

[14] F. Rosenblatt, *Principles of Neuro-Dynamics*, Spartan, Washington, DC, 1962.

[15] J.M. Zurada, *Introduction to Artificial Neural Systems*, West-Publishing Co., 1992.

[16] T. Kohonen, *Self-Organization and Associative Memory*, 3rd edition, Springer-Verlag, 1989.

[17] T. Kohonen, The self-organizing map, *IEEE Trans. Neural Networks*, 28(9) (Sept. 1990) 1464–1480.

[18] J. Kangas, T. Kohonen and J. Laaksonen, Variants of self-organizing maps, *IEEE Trans. Neural Networks*, 1(1) (1990) 93–99.

[19] D.J. Burr, Experiments on neural net recognition of spoken and written text, *IEEE Trans. Acoustics, Speech and Signal Processing*, 36(7) (July 1988) 1162–1168.

[20] E. Ifeachor, *Neural Networks and their Applications to Health Care*, Medical Electronics Systems & Centre for Intelligent Systems, University of Plymouth, UK, 1993.

[21] R. Poli, S. Cagnoni, R. Livi, G. Coppini and G. Valli, A neural network expert system for diagnosing and treating hypertension, *Computer* (1991).

[22] B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice-Hall, 1992.

[23] D.O. Hebb, *The Organisation of Behaviour*, J. Wiley & Sons, USA, 1949.

[24] S. Grossberg, Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors, *Biol. Cybernetics*, 23 (1976) 121–134.

[25] S. Grossberg, Do all neural models really look alike?, *Psychol. Rev.*, 85(6) (1978) 592–596.

[26] S. Grossberg, How does a brain build a cognitive code?, *Psychol. Rev.*, 87 (1980) 1–51.

[27] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, J. Wiley & Sons, New York, 1973.

[28] N. Nilsson, *Learning Machines*, McGraw-Hill, New York, 1965.

[29] M.L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, 2nd edition, MIT Press, USA, 1988.

[30] R.P. Lippmann, An introduction to computing with neural nets, *IEEE ASSP Mag.* (Apr. 1987) 4–22.

[31] B. Widrow and M. Lehr, 30 years of adaptive neural networks: perceptron, madaline, and back-propagation, *Proc. IEEE*, 78(9) (Sept. 1990) 1415–1442.

[32] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, 1991.

[33] J. Hertz, A. Krogh and R. Palmer, *Introduction to the*

Theory of Neural Computation, Addison-Wesley, 1991.
[34] M.R. Gray, Vector quantitization, *ASSP Mag.* (Apr. 1984).
[35] D. Guinier, Biological versus computer viruses: a better understanding for a better defence, *ACM SIGSAC Rev.*, 7(2) (1989) 1–15.

APPENDIX A

Table A1 shows the coordinates of the winning node of the output planar array for each cluster, the virus patterns that belong to this cluster, and the Euclidean distance between each pattern and the winning node of its cluster.

TABLE A1. Clusters of computer viruses

Cluster	Name	Euclidean distance	Cluster	Name	Euclidean distance
(11, 22)	1971/8 Times	0·0004883	(25, 07)	Murphy	0·0004883
(24, 28)	4096	0·0009766	(13, 28)	Oropax	0·0009766
(05, 10)	512	0·0000014	(04, 04)	Ping Pong-B	0·0000018
(28, 28)	5120	0·0019531	(19, 00)	Saratoga	0·0002441
(12, 09)	Amstrad	0·0002441	(04, 00)	Stoned	0·0000017
(07, 16)	Cancer	0·0004883	(23, 00)	Suriv A	0·0002441
(19, 16)	Cascade/170x	0·0003662	(04, 02)	Swap Boot	0·0000026
	Cascade-B/1701	0·0003662	(10, 14)	Sylvia	0·0004883
(00, 25)	Dark Avenger	0·0004883	(24, 24)	Syslock/3551	0·0007324
(28, 02)	Datacrime/1168	0·0004883	(18, 29)	Traceback	0·0009766
(07, 29)	DBASE	0·0004883	(10, 04)	V299	0·0001221
(07, 04)	Den Zuk	0·0000018	(19, 10)	Vacsina	0·0004883
(22, 05)	Devil's Dance	0·0002441	(04, 13)	Vienna/648	0·0002441
(00, 00)	Disk Killer	0·0000021	(08, 08)	Yale/Alameda	0·0000015
(08, 11)	Do Nothing	0·0002441	(04, 19)	Zero Bug/1536	0·0004883
(01, 02)	Form	0·0000026	(28, 24)	Machosoft	0·0007324
(14, 19)	Fu Manchu	0·0009766	(03, 43)	Shoe-B	0·0000020
(23, 15)	Ghost COM	0·0009766	(16, 25)	Advent Virus	0·0009766
(14, 00)	Icelandic	0·0002441	(01, 08)	Hello-1A	0·0000016
(14, 04)	Icelandic II	0·0002441	(00, 18)	Murphy II	0·0004883
(00, 29)	Jerusalem	0·0004883	(03, 27)	Suriv III	0·0004883
(03, 07)	Lehigh	0·0000014	(00, 05)	V277	0·0001221
(00, 14)	Lisbon	0·0002441	(10, 00)	V345	0·0001221
(26, 13)	MIX1	0·0004883	(07, 00)	12-Tricks Trojan	0·0000016
(00, 11)	Munich	0·0000022			

APPENDIX B

The training set of the 300 viruses used in the experiments is depicted in Table B1.

TABLE B1. Computer viruses training set

12-Tricks	Blood	Westwood	Multi-Face	Sad	Cascade-B
Scrambler	Crew-2480	Mannequin	Something	Anto	Datacrime
Fish Boot	621	982	Armagedon	Chad	Label
Guillon	Polimer	Fu Manchu	Troi	Green Joker	Twin Peaks
LZRQ	Swiss 143	Smack	Parasite	Europe-92	Itti
Cannabis	Violator	Frere Jacques	Albania	646	Amstrad
Boot Killer	Meditation	ABC	66A	Bryansk	Malmsey
HiDos	Marauder	Slayer Family	Grapje	Davis	I-B
Keydrop	Civil War	Saturday 14TH	Lazy	MPS 4.01	Explode
EDV	Kalah	Plastique-B	JoJo2	Icelandic	Lehigh
SF Virus	Tumen	Krivmous	AT144	Icelandic-III	Mindless
Swap	MGTU	CD	F-Word Virus	Got-You	1554
Den Zuk	Polish 217	Maltese Amoeba	Saddam	Cossiga	Datacrime IIB
Golden Gate	PhoenixD	Invader	Guppy	Pa	Prime
Ping Pong-B	Polish Tiny	Define	Mutant family	MIX/1	Casper
Generic Boot	Parity	Blaze	AIDS II	557	Emmie
Ashar	Manta	4870	Little Brother	Invol	Dir Virus
FORM-Virus	LiveChild	Nowhere Man	The Plague	NCU LI	Bomber
Disk Killer	Necro Fear	Burger	AIDS	Animus	Scroll
Chaos	Bebe	Viper	981	Dark Avenger	Globe
Evil Empire-B	Cerburus	Leprosy	Groen Links	Little Girl	DisDev
Bloody!	MG	Harakiri	Joker 2	Black Monday	Busted
Filler	Medical	Psychogenius	Traceback II	V2000	Clonewar
Exebug	VirDem-1542	Pascal-5220	Eight Tunes	Paris	Hacktic
Anti-Tel	Iraqui Warrior	Small-38	Sunday	Nomenklatura	Dutch Tiny
Deicide	Vienna	382 Recovery	RNA	M.I.R	PC Flu-2
Typo Boot	Hell	Silver Dollar	Little Pieces	Anthrax.	Caz
Brazilian Bug	DOSHunter	Shhs	Yankee 2	V2100	Sunday-2
Attack	Revenge attacker	Gnose	Thursday-12	Crazy Eddie	Spanish
Dad	Frogs	Italian 803	Taiwan 4	Liberty-2	Silly Willy
G&H	Beeper	Ear	Bios	Liberty	DataLock
Holland Girl	Devil's Dance	Funeral	Tack	Plumbum	Damage
Incom	OMT	All Sys 9	1963	CV4	Green Caterpillar
Kennedy	Taiwan	Necro Shadow	Lycee	Kuku-448	Keypress
JoJo	Best Wishes	Sistor	VMem	StarDot 600	5120
Cancer Virus	Anti-Pascal II	RSP-1876	4096	USSR	Victor
Ice 9	Hybryd	Bow	Gremlin	Witcode	Dima
Argentina	Ah	1575	1024 SBC	Flip	WordSwap
Phantom	Jeff	ZK900	Haifa	Happy new year	Groove
Solano 2000	Violator B4	Possessed	Dir-2	Ha	Kemerovo
Lisbon	1253	RAM Virus	Whale	Arf	Father Christmas
Ant	Japan christmas	1392	Fish	595	Headcrash
dBASE	VCS 1.0	Jerk	Mayak	ARCV Friends	Hitchcock
834	Wisconsin	Akuku	SVC 5.0	Terminator	CSL
Ghostballs	Attention!	Rybka	Virus-101	Null-178	Saratoga
1226	Evil	V483	VP	Print Screen	Itavir
1704 Format	Phoenix	Zaragosa	Hydra Family	Warning	99%
Murphy	Icelandic-II	BFD	Worm-16850	10 Past 3	Tequila
Sentinel	Close	Internal	Horror	Kthulhu	Crusher
923	Siskin	RefRef	Green Peace	Naughty hacker	Shield