Computer "virus" identification by neural networks

An artificial intelligence connectionist implementation naturally made to work with fuzzy information

by **Dr. Daniel GUINIER**

ACM, IEEE, IACR, EDPAA, IIA member

President OSIA, Inc.

and

Information Systems Security & Quality Department Chairman
Regional Institute for Promotion of Applied Research (IREPA)

Abstract

Computer viruses are more and more numerous : around 400 in the year 1990 and this number is estimated to reach 1,000 for 1994-95. Users are not experts and need help in identifying the virus and carrying out the most appropriate cure in case of attack.

Knowledge of viruses is necessary but public information offered by virus databases or catalogs gives a powerful advantage to virus makers. On the other hand, not enough or no information to users is also a problem because then they use the product they have which does not necessarily provide the appropriate solution in case of virus attack. We propose **an alternative solution to the dilemma** found in a **neural network**, an artificial intelligence connectionist model which is fault tolerant, self adaptative to learn automatically, retaining experience to solve the problem of virus identification regarding **fuzzy information** on concerns and effects.

Principles of the formal neuron and the neural network using hidden nodes is examined as well as the theoretical and practical apects of the **gradient *back propagation* algorithm**. **An implementation** of the algorithm is applied to virus identification with data referring to virus concerns and their obvious effects. First results have shown **a correct identification of viruses** while using fuzzy knowledge of end users **introducing uncertaincy on answers** or, even, **forcing erroneous data**. Such a system can be employed by ordinary users, system or computer security managers, as well as consultants as a complementary tool for virus warfare.

Further work needs to be conducted to **validate methodologically** such an approach and to **optimize** input data coding, the **choice for parameters** andthe **learning strategy**.

Keywords : *Gradient back propagation, Fuzzy logic,Identification, Knowledge, Neural network, Neuron, TDL/V, Virus*

Mailing address : BP 86
F 67034 STRASBOURG Cedex, (France)

1. Introduction

Computer security losses due to virus attacks are often catastrophic for a single organization when it cannot solve the problem very quickly. The number of viruses in the year 1990 was about 400 : about 250 on PC, 70 on Amiga, 25 on Atari and 25 on Macintosh. Their number should be superior to 1000 in 1994-95 !

2. Knowledge, objectives and needs

Knowledge of viruses using databases for **investigation** (D. Guinier (1989b)) *(i.e. dedicated to criminality and external expension virus studies)* or **classification** (K. Brunnstein (1989)), **virus description languages** such as the Threat Description Language for Viruses (TDL/V) (M.G. Swimmer (1990)) from the VTC *(Virus Test Center from Hamburg)* contribute to give enough detail **to identify** possible virus attacks. Objectives and needs of different entities are :

Computer security organizations and virus emergency centers	
Objectives :	To collect, centralize and manage properly information to generate knowledge. To offer prevention and protection measures and policy recommendations. To develop methods to help users in case of crisis, etc.
Needs :	**All valuable information available** *(virus, system and organization lacks, etc.).* Understanding virus makers *(goals, motivations, education, behavior, etc.).* Early information from users in case of virus attack, etc.

Normal end users	
Objectives :	To use properly computer systems.
Needs :	To have an organizational virus policy for prevention and protection. To have knowledge about an available virus emergency center. **To identify the virus** in case of attack for **choosing the most appropriate countermeasures** without being a specialist on viruses. Early help from virus an emergency center.

Virus makers	
Objectives :	To produce an efficient virus *(e.g. difficult to detect, giving significant damage).*
Needs :	**Maximum information on virus**, worms, etc. Early information on systems and procedure lacks. Knowledge about organizational deficiencies To infiltrate virus emergency centers, virus research labs, organizations, etc.

3. Proposal for an alternative solution to the dilemma

Giving too much information on viruses can offer a powerful advantage to virus makers but, in the other hand, not enough information to users is also a problem. There is evidence of a **bottleneck** of unclassified information on viruses and a **dilemma** of the general necessity of academic knowledge for men and the general need of protection against agressive agents : viruses and their makers. Our purpose is to solve the dilemma by introduction of **an alternative solution** which can provide sufficient information to identify the virus and to help any normal user with appropriate countermeasures.

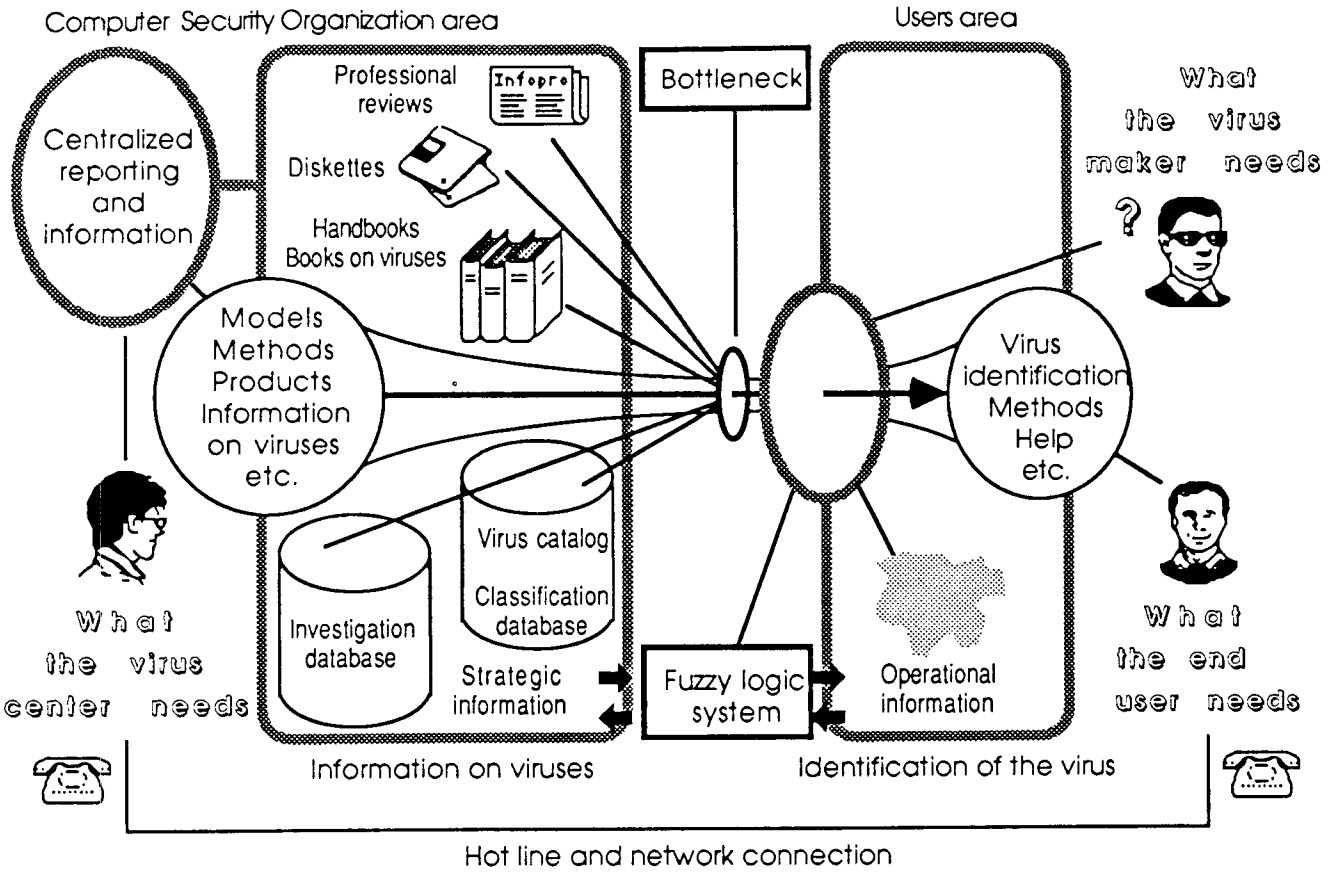

Fig 1 : A possible model for distributed knowledge regarding the different objectives.

Usually computer virus strategies use **emergency procedures** in case of virus invasion **after identification of the virus**. Virus warfare directly conducted without identification with anti-virus products can be **more or less well adapted**, depending on the virus.

Identification by fingerprinting does not work every time and not at all for a new virus. It is necessary to have a system able to investigate with fuzzy, incomplete or even erronous data. Such a method should work as a tolerent system and for new viruses having some similarity *(i.e. a clone)* with an older one. It must be easy to employ and at no risk for the user to inform a virus center which will possess information on virus erradication and cure. Also, the solution will be to use a **fuzzy logic** based system.

4. The use of a connectionist model of AI

Artificial intelligence should help and there are two possible approaches : the **symbolic** or the **connectionist** models. The first is rationalist rule-based and will be successful for clear goals and rules. The second is empiriscist brain decision based on and better in case of uncertaincy. We have been attracted by neural networks, corresponding to connectionist models.

Brain is **fault tolerant**. Even in case of neuron losses, it continues its function with the same level of efficiency by compensation, and even in case of massive incapacitation, the loss of the performance is partial and does not tend to a catastrophe. It is **self adaptative** by learning automatically without the need for new algorithms, because it synthesizes pattern classifications from its experience and is a **good pattern matcher to perform perceptual informal problems**.

Also, neural networks have their advantage in that they simulate the human brain and can easily solve a problem, especially in case of **classification**, **diagnostic** or help for **decision-making** and when it is not needed to explain why but when it just required to make a choice **based on experience after learning**.

4.1. From biological neuron to formal neuron

Nerve cells, also called **neurons**, are the structural and functional units of the nervous system. They are basically also the main elements of the central nervous system. There are approximately **100 billions** of such cells in the human brain. Each is constituted of dendrites, a cellular body and an axon. **The function of a neuron is to receive, to integrate signals coming from other neurons and in turn to transmit the nervous influx** to the next.

The **dendrites** have as a role to capture influx signals. They are the passive parts of the nerve cell. The **cellular body** with its nucleus in the middle can be considered as the biochemical center. The **axon** is a nervous fiber used for transportation of the signal from one neuron to the next, it is the electrically active part of the nerve cell. Finally, **connections of neurons** are made at particular places called **synaps**.

A **neuron** makes the sum of electrical influxes coming from its neighbourhood and reacts with an electrical current transported by the axon when the sum of input signals is superior to a given threshold. At this moment, it is said to be an active neuron. A representation of such a biological neuron can be modelized in the form of the formal neuron. The behavior of the neuron will be described by inputs, weight parameters, some functions *(input, activation and output functions)* and finally, outputs.

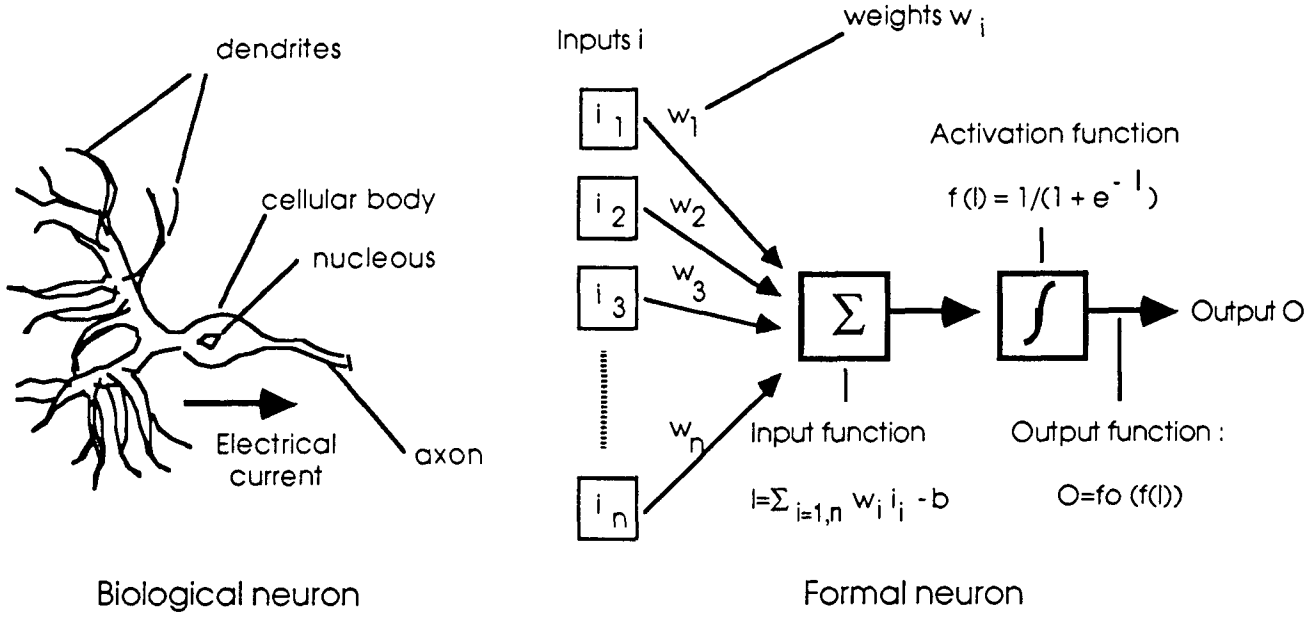

Fig 2 : Biological and formalized neurons

4.2. Modelisation of a neuron : the formal neuron

Input node content depends on the nature of the inputs. It can be binary or analog. The total **input function fi 0** defines a treatment applied to the inputs. The **activation function f (I)** determines the internal state of the neuron in relationship to its total input function I. The **output function fo 0**

computes the output of the neuron (J.L. McClelland, D.E. Rumelhart (1987)).

4.3. Hidden node multi-layer neural networks

In the brain, there is a considerable number of connections (nodes) between the neurons and the cerebral cortex is divided into numerous layers themselves with many connections. These layers are linked with others to form a complex network. Considering the formal neurons, total connections are possible, but it is considered local step by step connections to form which is called **a multi-layer neural network**. In this case, nodes from the same layer are non interconnected, each layer treats the signals from the previous and transmits the final results to the next, and so on. The middle layers and associated nodes are called **hidden**.

Weighted parameters (w) are used to give more or less importance to each connection and to represent connection strengths, playing the role of inhibitor or excitator corresponding to the biological synaps. The connections are dynamical because of the evolution in the value of the weights. This is **the learning process** which can be mathematically considered as a transfer function for which it is necessary to have an implementation in the form of an **algorithm**.

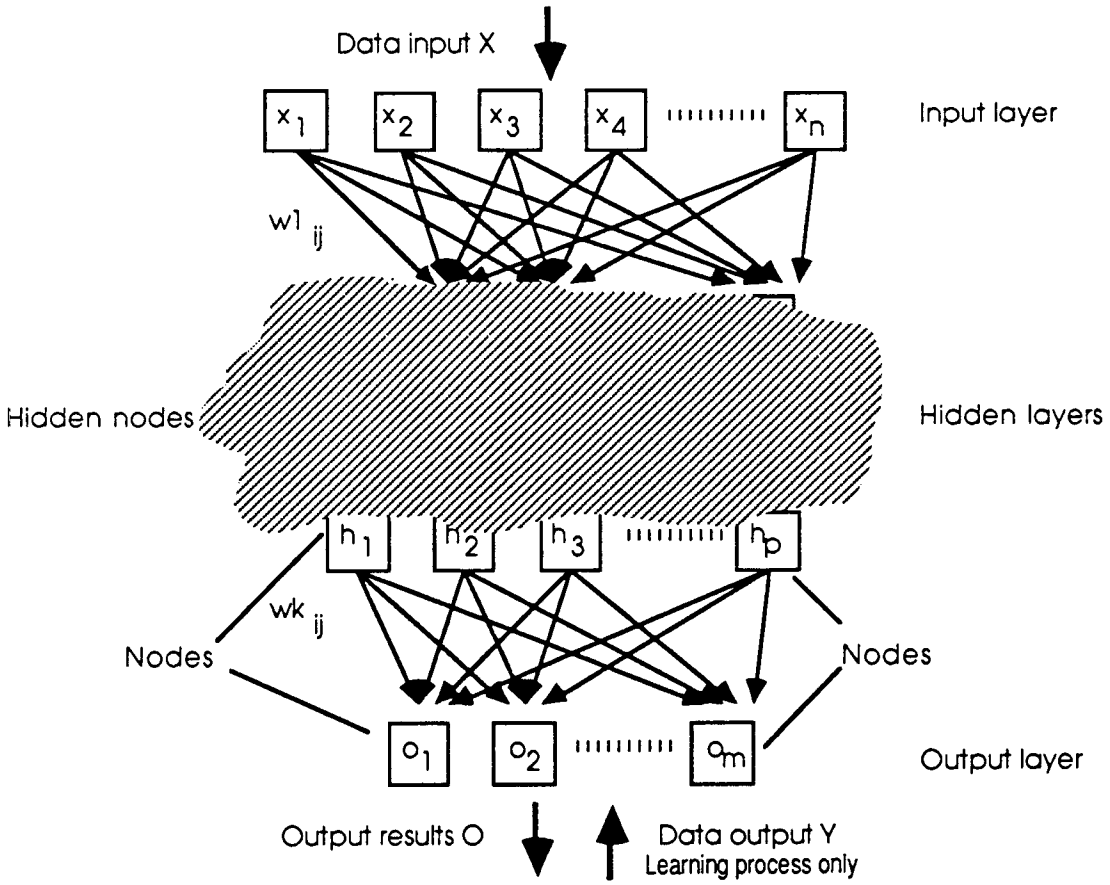

Fig. 3 : Hidden multi-layer neural network architecture

4.4. The gradient back propagation algorithm

Such an algorithm has been described by D.E. Rumelhart, G.E. Hinton, R.J. Williams (1986) under the name *gradient back propagation* algorithm ***(GBP algorithm)***. The signal crosses several connections and is non linearily transformed and presents a measurable error at the output layer.

The goal of the algorithm is to distribute this error at each node. This is possible if we can propagate the signal from the input to the output as well as to retropropagate the signal from the output to the input layer. Computation of the gradient in the learning process begins at the last layer and progresses layer after layer in the opposite direction of the outputs. Hence the name : **back propagation**.

The **error signal following the inverse path of the intput signals** in the network is measured directly from the output neurons by the weighted sums from the following layer for the other neurons. It is **a local algorithm** with a minimum of overall control due to learning computations realized independently at the level of each neuron. The procedure allows the possibility of **parallelization for such an algorithm** with respect of the biological plausibility.

4.5. Learning process with GBP

The neural network *(or the humain brain)* must be trained by exposure to a large set of cases which consist of defined input characteristics coupled with their corresponding output. During the learning, it is necessary to **submit a sample of input data X and their corresponding known output Y**. From each input, the neural network computes from near to near an output result O using the actual values of the weight parameters. Then, the error Δ represented by the the quadratic sum of the errors obtained for each output node is then back propagated in the network **to modify each weight (w)**. This process is repeated up to a submitted error threshold. Finally, learning consists of minimizing the quadratic error Δ which is a function of the weight parameters using a gradient descent. The activation function will be chosen as a derivable function in sigmoïdic form.

The GBP algorithm offers a progressive self-adaptative answer to the input-output association patterns. Several strategies are possible. Patterns can be chosen sequentially or randomized or the learning process can be done using just one part of the list to reduce learning time. Some patterns could also be duplicated and disposed of at different places in the learning list to force their influence.

Data input vector $X = \{ x_1, x_2, x_3, ..., x_n \}$

Data output vector $Y = \{ y_1, y_2, y_3, ..., y_m \}$

Output result $O = \{ o_1, o_2, o_3, ..., o_m \}$

The quadratic error is $(y_i - o_i)^2$

The total error for all the s stages from the samples is $\Delta = \sum_s \Delta^s = \sum_{t=1,s} \Delta^t$

where Δ^t represents the error on the t^{th} example $\Delta^t = (y^t - o^t)^2 = \sum_{i=1,m} (y_i - o_i)^2$

Gradient for the output layer	$\partial_j = 2.f'(i_i).(o_i - y_i)$
Gradient for the hidden layers	$\partial_i = \sum_h \partial_h .w_{hi}.f'(i_i)$
Weight modification rule for each **stage s** is	$w_{ij}(s) = w_{ij}(s-1) - g(s).N_j.\partial_j$

$f(i_i) = 1 / (1 + \exp(-i_i))$ a sigmoïdic function

$f'(i_i)$ = derivative of the function : $f(i_i)$

N_j is the output value for neuron j

$i_i = \sum_j w_{ij}.N_j$ is the intput value for neuron i

g(s) is the step of the gradient at stage s

h considers the neurons which have a connection from i

4.6. Hidden nodes, learning rate and *momentum*

The learning time necessary has been demonstrated by M. Minsky and S. Papert (1969) to follow an exponential law of the complexity. But a measure of complexity has not been

established for the moment. The convergence of the algorithm has not been proven and the working efficiency depends on the adjustment of some parameters under real working conditions.

The **number of hidden nodes** in the input and in the output layers are well specified by the problem to be solved under user guidance to choose these characteristics, but the number of hidden nodes in the middle cannot be clearly defined. It has been shown that the number of middle layers improves the quality of learning to a point but is time costly. There is no formal law to determine the adequation between learning improvement and learning time. However, if only a few hidden nodes are in use, the network may not be able to formulate good answers.

The **learning rate** is the constant of proportionality. It regulates the weight changes as a function of the error. The larger the learning rate, the larger the weight changes, and also the faster the learning ! The **learning threshold** is the parameter set to stop the learning process when the error falls below the value given by the user.

Sometimes, the **error surface contains a long and smooth descent** in space and it is necessary to increase the value of the change along the direction of the gradient or else **large learning leads to an oscillation** in the weight changes. The process never completes and tends to a **chaotic** situation. There is one solution however which allows to change weight as a function of a previous change and to provide a smoothing effect. This allows a faster learning without an oscillation. The ***momentum*** factor gives this possibility setting the proportion of the last weight change to add to the new change :

New weight change = { Last weight change }.{ **Momentum** }+{ Learning rate }.{ Error function }

5. A GBP neural network implementation

5.1. Choice for a neural network program

Rather than develop a complete new system, we have fixed our choice on an existing and **commercially available package**. The system has to apply to a specific but **real problem** and get an **adapted solution**. The program has to build a neural network of **sufficient size** to solve the problem in terms of experience and if possible, to offer a **hardware solution** to reduce the learning time process. After computing the weights, the software has to give the possibility **to integrate** them automatically in a run time source code. This last run time functionality makes the development easier.

Several packages are presenting available fulfilling these preliminary conditions. We tum our choice to **Neuroshell** *(release 4.0)* from Ward Systems Group, Inc. as an economical product to implement on a PC and because they offer NeuroBoard as a potential hardware solution.

5.2. Virus identification modules

Identification modules consist of three programs : the **extraction** program, the **learning** program *(NeuroShell)* and the virus **identification** program.

The **extraction program** is in charge of the input data generation for the learning process. The questions file for operational process of identification, and the file for virus names, countermeasures and general virus information.

The **learning program** *(NeuroShell)* is furnished by Ward Systems Group, Inc.

and is in charge of the learning process to generate the parameters implemented in the identification program.

The **virus identification program** seeks answers to oriented questions. It is interruptible and restartable, saving the results of questions on a file. Answers can be "Yes" (coded 1.0), "Perhaps" or "Not sure" (coded 0.5) and "No" (coded 0.0). It gives pointer for corresponding virus name, countermeasures and general information on virus in extracted virus names and measures file.

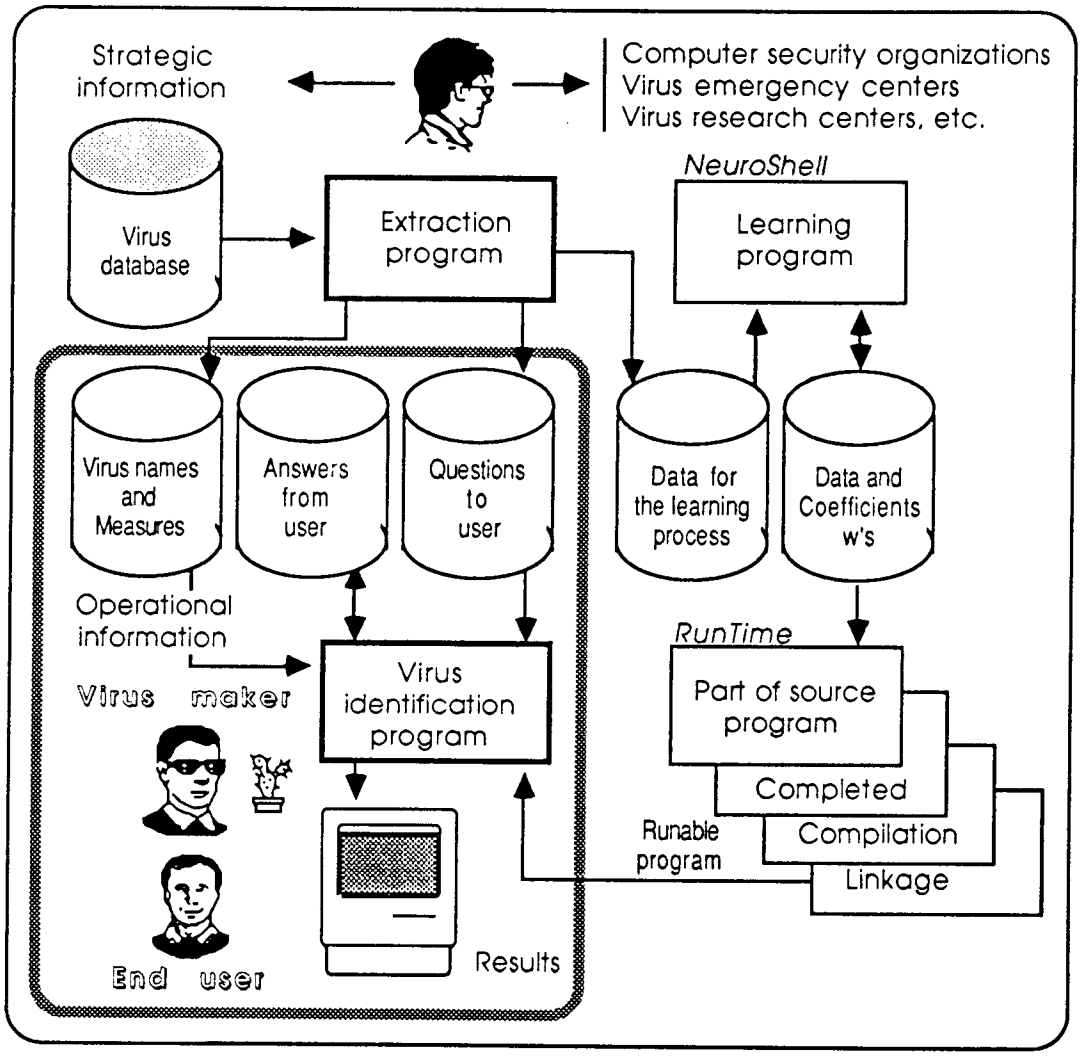

Fig. 4 : Transforming strategic to operational information and satisfying the needs of end users

5.3. Data coding

The Threat Description Language for Viruses (TDL/V) (M.G. Swimmer (1990)) from the CVC *(Computer Virus Catalog)* from VTC *(Virus Test Center (Hamburg))* is sufficient in itself to provide an easy coding of the input for a neural network.

Input data considers the obvious viral effects and concerns, **coding is made by a combination** of the strict information contained in the CVC and regarding the potentiality of answers made by novice users. **Output data** is the presence or not of the virus *(a single 1.0 and*

others 0.0).

Virus characteristics	Virus attributes
Virus type	{ program virus, boot sector virus, RAM memory resident }
File infected	{.COM, .EXE, both, .E*, COMMAND.COM, etc. }
System infected	{MS-DOS, MacOS, Amiga-DOS, Atari-TOS, UNIX, VMS, etc. }
Computer model	{ROM BIOS version}
Version release	{Version x.x}
Message on screen	{ your PC is stoned, Merry Christmas, April April, etc.}
Message contains in virus	{ @aids, today is Sunday, IBMBIO.COM, Access denied, etc.}
Sound effects	{ beeps before message, Oh Tannenbaum melody, etc.}
Size change	{ com +1704 O, exe + 1618 -1634 O, command.com + 555 O, etc.}
Existence of associated file	{ hidden file c:\bugs.dat, hidden file \ibmnetio.sys, etc. }
Damage	{hang, numlock on, slow down, destroy file on Friday 13th, etc.}
Trigger	{infection trigger, damage trigger}
Date	{date change, seconds are put at 62}
Special	{ read only files are lost, .COM >64 KO unrunable, etc.} , etc.

5.4. Learning process

At each step, the learning process is under control of the error distribution histogram to follow the quality and the velocity and to adjust the values of the learning parameters at any time. The program has been run on an AT-386 at 20 MHz. As an example :

Parameters adjustments for a learning process involving **210 Inputs** and **54 outputs**			
Hidden nodes	**32**	Learning threshold	0.0001
Learning rate	0.6	Momentum	0.9
Learning time	8 hours		
Error distribution	92.6%	0.0051 to 0.01	(50)
	3.7%	0.0001 to 0.005	(2)
	3.7%	0.0000 to 0.0001	(2)

A **contribution factor** is also used to give a rough measure of the importance of each input data relative to the others, and finally suggest to maintain or remove such a data to simplify the network and by consequence to permit a faster convergence to the network. The contribution factor is the sum of the unsigned values of the weights playing a leading part from a given input data.

The **NeuroBoard** should permit to **run the learning process about 100 times faster** than when using a single software solution on an AT-386 machine at 20 MHz equipped with a 80387 arithmetic coprocessor. That is less than 5 minutes comparing to 8 hours ! This permits to gain more experience on the neural network behavior in relationship with the parameters *(learning rate, momentum, number of hidden nodes, ...)* on a single set or to try to test results regarding the possibility of using subsets of input data.

6. Operational process and results

During the **learning process** 210 inputs corresponding to characteristics and attributes taken from the Computer Virus Catalog from the VTC (K. Brunnstein (1989), M.G.Swimmer (1990)) are coded {0.0, 1.0}.They correspond exactly to the 54 different outputs, that is the experimental sample used. The total sample should be around 500 (the actual total number of virus). During the **operational process** the 210 inputs are coded {0.0, 0.5, 1.0} by the program in relationship with the

questions and the answers furnished by the user. Results are offered in the form of the **name of the three most probable viruses** and their **associated curative measures**.

Input

Questions asked to the user relative to different characteristics of the virus
Answers are coded by the virus identification program

ex. : Question : Do you have change for program size *(Yes, ?, No)*.
Answer : Yes *(Virus identification program coding is : 1.0)*

Results

Virus name and **alias**	{syslock, swap,fumanchu, marijuana, cascade, zero bug, etc.}
Countermesures	{measures and standard means to apply to eliminate virus}
General information	{Lacation, classification and documentation authors, date}

7. System validation and further work

Forcing unknown answers or ambiguity by the way of a novice user or possible input errors, random or voluntary errors **give correct answer up to 25% of wrong or indecidable answers** on this sample of 54 different viruses. Experimentation needs to be done on the **full catalog**. Various **optimizations** in the questions as well as in neural networking, data coding, parameter choice, operating system virus compartmentation, etc. will probably increase the power of the neural network answer as well as the conviviality of the answers.

8. Conclusion and perspectives

Such a system can be considered as a **complementary tool for the virus warfare**. It is dedicated to different users : ordinary users, system or computer security managers as well as consultants. For virus makers, information obtained has a reduced effect on their knowledge.Data refer to virus concerns and their obvious effects. Also, **in case of a new virus**, the results furnished by the neural network are in relationship with **similarities with a known virus**.

The changing nature of viruses, **men** and **environment** is well adapted to neural networks. It should be interesting to consider this technique inside the new generation of behavioral real time intrusion detection systems to monitor systems, men, environment and interactions in terms of behavior rather than isolated single facts and classification using segregation by value (Guinier D. (1991)).

9. Bibliography

Brunnstein K. (1989) : Zur Klassifikation von Computer-Viren : Der "Computer Virus Catalog", Proceed of the 19th. German Computer Science Association, Ann Conf.

Guinier D. (1989 a) : Biological *versus* Computer Viruses, A better understanding for a better defense. ACM SIGSAC *REVIEW*, Special Interest Group on Security Audit and Control. Vol.7, No.2, pp.1-15.

Guinier D. (1989 b) : Proposal for a "C-Virus" database dedicated to SRP's, Worms, Trojan horses,... ACM SIGSAC *REVIEW*, Special Interest Group on Security Audit and Control. Vol.7, No.3, pp.13-16.

Guinier D. (1991) : Prophylaxis for "virus" propagation and general computer security policy. ACM

SIGSAC *REVIEW*, Special Interest Group on Security Audit and Control *(submitted for publication)*.

Highland H.J. (1990) : Computer virus handbook. Elsevier Advanced Technology. pp.1-375.

McClelland J.L., Rumelhart D.E. (1987) : Parralel distributed processing, Explorations in the microstructure of cognition. MIT Press,Vol. 1.

Minsky M., Papert S. (1969) : Perceptrons. MIT Press.

Rumelhart D.E., Hinton G.E., Williams R.J. (1986) : Learning internal representations by back-propagating errors. Nature, Vol. 323, pp.533-536.

Swimmer M.G. (1990) : Response to the Proposal for a "C-Virus" database.ACM SIGSAC *REVIEW*, Special Interest Group on Security Audit and Control. Vol.8, No.1, pp.1-5.

Ward Systems Group, Inc. (1989) : NeuroShell manual. pp.1-98.

Ward Systems Group, Inc. (1989) : NeuroShell run time manual. pp.1-26.

Ward Systems Group, Inc. (1990) : NeuroShell utility manual. pp.1-46.

Ward Systems Group, Inc. (1991) : NeuroShell tips, techniques and update information manual. pp.1-46.

Winston P.H. (1984) : Artificial Intelligence. Massachusetts, Addison-Wesley Publishing Company, 2nd. edition.