The Radar Concept using Neural Networks

Thomas Alexandre

Recherche et Développement Dossier Portable (RD2P - LIFL)
1,rue du Professeur Jules Leclerc
59037 LILLE Cedex - FRANCE
Tel (01133) 20.44.60.46 / Fax (01133) 20.44.60.45 / E-mail: thomas@ rd2p.lifl.fr

Most of researches in smart card security have been oriented towards improvement of lockers. The goal is to prevent unauthorized people from getting accesses to the ressources of the card. Whatever can be the level of a security system, it can fail, especially because of human factors such as negligence or violence.

In houses or buildings, the best lockers are often considered as not sufficient and people use radars. The role of a radar is not to prevent intrusions but to detect abnormal ones. Thus a radar does not replace keys and secret codes. It just completes the overall security scheme.

We have advanced a research to identify a possible implementation of this concept in smart cards. The basic idea is to recognize the behaviour of the so called "visitor" and to possibly refuse the required service because that behaviour does not seem normal. The locker is independent of the application. On the contrary, the radar is closely related to the application.

In a prior definition of «the Radar Concept» *[ALEX94]*, we have introduced an expert system using the Prolog Language to implement the idea of Radar through a credit card application. The goal was to identify somebody's behavior doing transactions with his card.

The purpose of this paper is to give both a complementary and different approach of the radar concept introducing a neural network for regulating the security levels in accordance to their importance in the whole process of behavior recognition.

In addition to the Prolog-based radar implementation, we will try in a first step to use a small neural network for classifying the security levels given by the expert system.

Then we will turn towards a complete neural network implementation of radar on the basis of the four items <Nature, Amount, Date, Location> defined for the credit card application. We will discuss the results of both the Expert System and the Neural Network approaches in terms of efficiency and performances.

1. The Radar based upon Expert Systems

The first approach of the «Radar» concept we have experimented was related to expert system technology, in which the special knowledge of the «experts» in intrusion detection was codified as rules to analyze the audit data for suspicious activity *[ALEX94]*.

The obvious drawback to this approach is that the system has obtained a limited expertise because of the small amount of rules available inside the smartcard. This restricted number of rules is due to the reduced memory space and processing power to perform the checks in a reasonable time.

The small expert system kernel we have constructed, written as a Prolog interpreter in a smartcard, had only the minimum functionalities of the rule-based Prolog language and therefore was limited in the complexity of the rules. For this main reason, the expert system was not taking into account several aspects, such as:

- There was no difference in diagnosis between somebody who would for example do a transaction with an *Amount* overflow of only 1FF, and an *Amount* overflow of 10000FF.

- A knowledge base was defined at the begining with static values representing someone's habits in the form of thresholds among which the transaction was considered as suspicious. We know that people habits seem to vary considering a long term period. This last point was not implemented through the expert system and would require to set up a dynamic knowledge base that would be updated according to the user's past activities over the last few months only and not from the card issue.

- Suspicious transaction elements detected by the expert system were considered at the same level (an unknown *Location* and an *Amount* overflow had about the same weight in the decision). A distinction should be made between them so that for example an *Amount* overflow has a bigger influence in the final decision than an unknown *Location.*

Those lacks regarding an expert system solution of Radar have led to other investigations relative to competitive approaches in artificial intelligence. One of them that is increasingly used to solve recognition problems is the neural network technology. This emerging technology seems relevant for our behavior recognition problem because of the following points, as underlined in the NIDES project (Next generation Intrusion Detection Expert System) *[LUNT93]* dedicated to the set up of a system for detecting intruders in computer systems *[ANDE93] [JAGA93]*:

- The set of intrusion-detection measures collected for analyzing behaviors is mainly selected on the basis of experience and intuition criteria. It is difficult to evaluate the efficiency of any given set of intrusion-detection measures for characterizing user behavior. A neural network can help in this evaluation.
- In terms of algorithm development, adapting neural network algorithms when defining new behavior characteristics is easier than reconstructing new statistical algorithms and rebuilding the software implementing them.

2. A Radar implementation through Neural Networks

2.1 Introduction to Neural Networks

Neural networks technology is based on the modelization of the neural structure of the brain on which scientists have been doing intensive research to understand its biological structure and behavior. Basically, the human brain is composed of more than 100 billion highly interconnected neurons delivering electric pulses of various intensity depending on their activity due to the operations to be achieved.

From biological experiments have resulted different artificial computer modelizations of various complexity *[GELE91][DAVA93]*. Here we are going to briefly detail the most famous algorithms.

2.1.1 The Perceptron

The first approach known as «The Perceptron» and invented by F.Rosenblatt in 1957 *[ROSE57]* provided an architecture family as below:

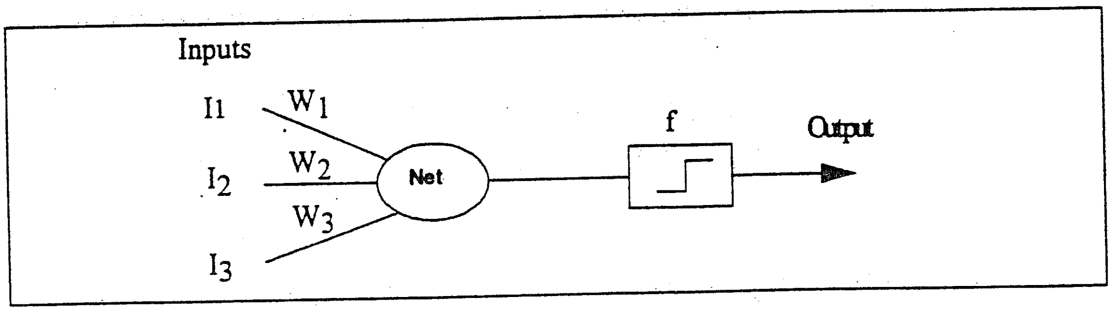

FIGURE 1. A single neuron with 3 inputs.

A network can be seen as a black box with inputs and outputs performing a special function or task. For any given set of inputs ($I_1, I_2, \ldots, I_N$) applied to a single processing unit (neuron) the ouput is calculated by summing these inputs, weighted by elements ($W_1, W_2, \ldots, W_N$) of the weight matrix W which represent the contribution of each input. The resulting value is then compared to a threshold. If the threshold is bigger than the net input then a zero value is given at the output, otherwise a one is given. The summarized process is:

$$Output = f\left(\sum_{i=1}^{n} Wi \times Ii + b\right)$$

where (W1,....,Wn) are the weights
(I1,..., In) the inputs of the network
b the bias
and f a threshold function of the following shape:

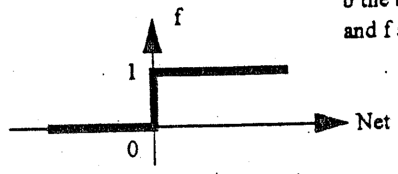

FIGURE 2. Mathematical formulation of neuron processing

The biases are helpful for they provide an additional free variable that can be adjusted to obtain the desired network performance. They correspond to a neuron with a constant weight of 1.

Neural Networks main interest resides in a learning phase which is the procedure where the weights are calculated to solve a particular problem. The perceptron network is trained to respond to each input vector with a corresponding target output vector whose elements are eiher 0 or 1 *[BLOCK]*.

Vectors from a training set are presented to the network one after another. If the network's output is corresponding to the target output no change is made. Otherwise the weights are updated using the perceptron learning rule *[ROSE61]*.When an entire pass of the training set (an epoch) has ocurred without error, training is complete. At this stage any input training vector may be presented to the network and it will respond with the correct output target vector. If a new vector P_i not in the training set is applied, the network will tend to exhibit generalization by responding with an output similar to target vectors for input vectors close to the previously unseen input vector P_i.

The perceptron learning rule has been proven to converge on a solution in finite time if a solution exists. Unfortunately, perceptrons have inherent weaknesses that completely stop them from solving some problems. Because perceptrons learn to output 1's and 0's for classes of input vectors by dividing the input vector space with a line, they can properly classify only sets of vectors which are linearly separable. An example of 2D classification problem is shown below:

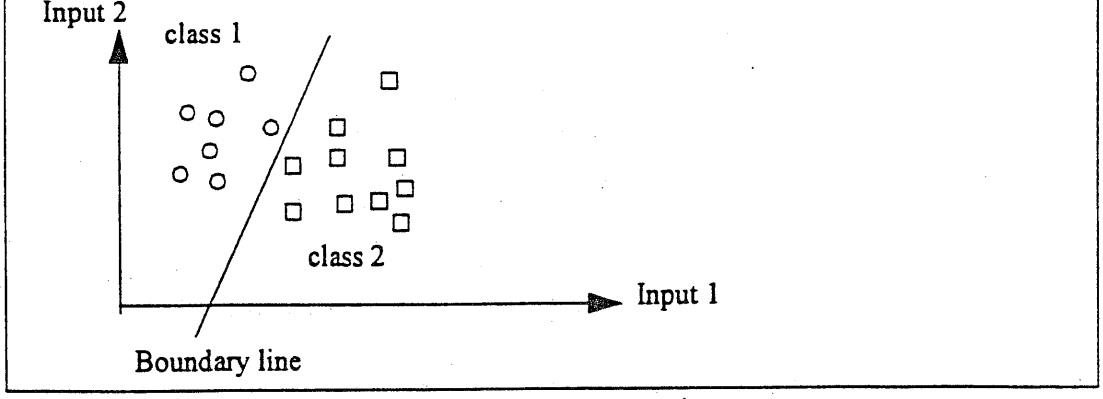

FIGURE 3. 2D linear separable problem

The most famous example of the perceptron's inability to solve problems with linearly nonseparable vectors is the XOR problem, demonstrated by Minsky and Papert *[MINS69]*.

From these difficulties encountered with the Perceptron have emerged many other learning criteria and rules. The most famous, introduced by Rumelhart *[RUME86]* and called «Backpropagation» has become a reference and is used in most of today's practical applications. The coming section is describing this algorithm. We are going to use it for the Radar because of its efficiency to solve non linear problems.

2.1.2 Backpropagation

Multilayered networks are capable of achieving just about linear or non linear computation and can approximate any reasonable function arbitrarily well. Such networks don't suffer from the problems associated with the perceptron and linear networks.

Being a supervised learning algorithm, the Backpropagation learning rule relies on a teacher which is a set of example pairs of patterns. The basic idea of this learning process is to minimize the error commited at the output (which is the difference between the output given by the network and the target output) by retropropagating the error backward the network for adjusting the connection weights.

A typical «feed forward network» required for executing this type of algorithm is shown below:

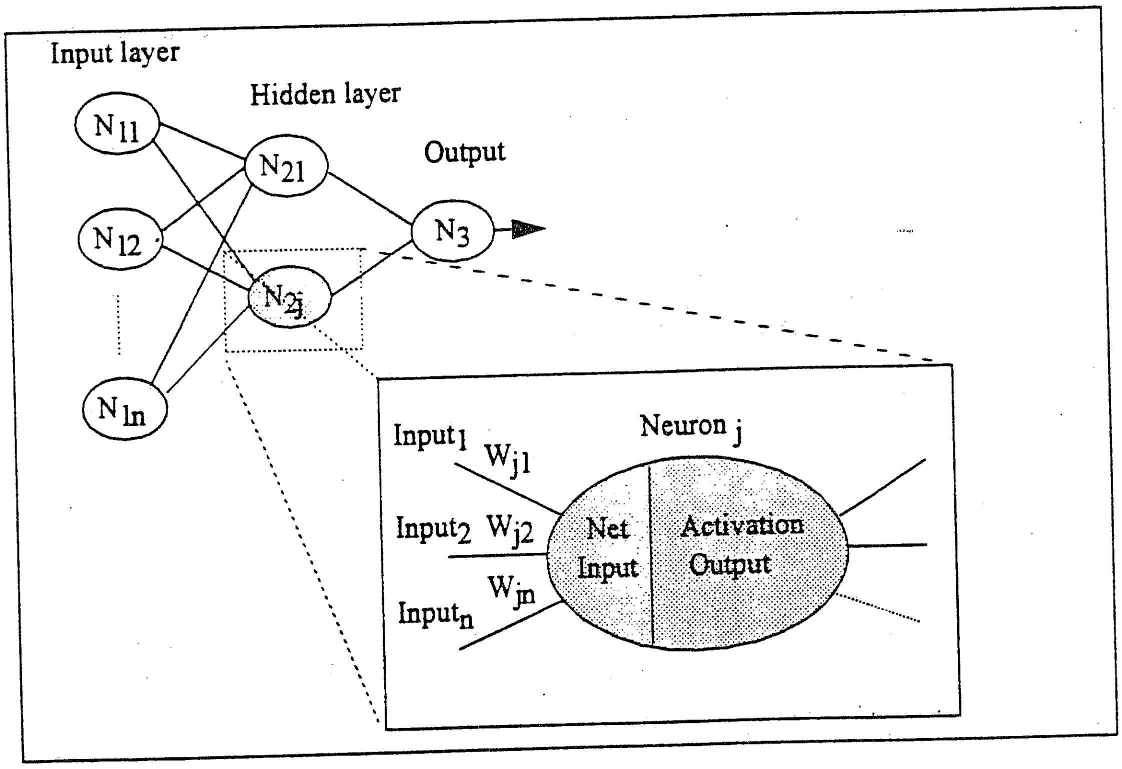

FIGURE 4. Feed forward network

For each example extracted at random from the training set, activation values are calculated for each neuron of the network in order to compute the resulting general output which is the response through the network to an input vector. Mathematically, the feed forward process is done for each neuron by the two steps:

$$NetInput = \sum_{i=1}^{n} Input_i \times W_{ji}$$

$$Activation = F(NetInput) = \frac{1}{1+e^{-NetInput}}$$

FIGURE 5. Feed forward process

W_{ji} represents the weight for each connection and the threshold or activation function F is a *sigmoïd* function of the type below:

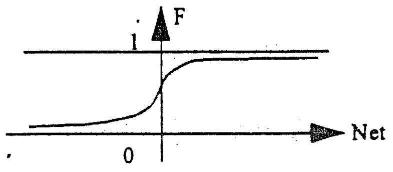

FIGURE 6. Example of sigmoïd function

Once the activation values of all neurons have been computed, the network is calculating an error value E_p that corresponds to example pair p:

$E_p = \frac{1}{2} \times \sum (O_{pi} - T_{pi})^2$ Where O_{pi} is the output value of neuron i on the output layer of the network
T_{pi} is the i'th signal value on the target output pattern of example pair p

FIGURE 7. Error value between actual output and desired ouput

The goal of the training process is to minimize this error so that the output closely corresponds to the target output. This process involves sweeping the error backwards through the network and calculating at each layer the relevant changes in the connection weights W_{ij}, using a learning rule known as the **Delta Rule** *[RUME86]*, which is:

$$\Delta W_{ij} = \alpha\left(\frac{\partial E_p}{\partial W_{ij}}\right) = \eta \times \delta Lpi \times O_{Lpi}$$

E_p is the error value described previously

ΔW_{ij} is the amount by which the weight Wij should change correspondingly to training pattern pair p

η is the learning rate

δLpi is the error on the output of unit i on layer L for pattern pair p

The learning rate specifies the changes that are made in the weights at each epoch. The actual objective of backpropagation is to complete a gradient descent along the error surface in search of a global minima which is a state of the network where the error output is minimum.

Although back-propagation has proven its efficiency considering a large set of practical applications, a few difficulties are often encountered during the set up of such an algorithm, because of the following points:

•An error surface of a neural network can have several minimas. The training process may be trapped into a local minima which is not the best solution to a problem, because a global minima would give a better minimization of the error, as shown on the next figure:

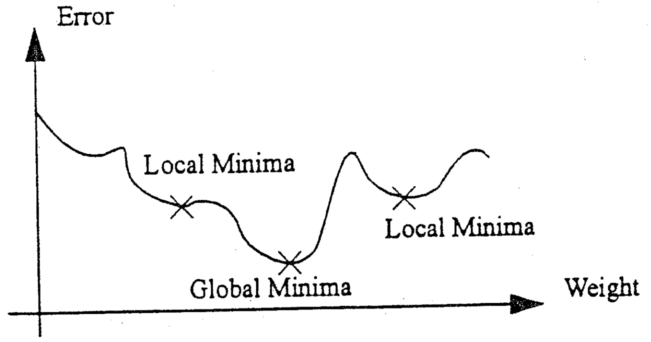

FIGURE 8. Typical example of an error surface of neural network

•Picking the appropriate learning rate for a non linear network remains difficult. On the one hand, large values can lead to an oscillating learning because in that particular case, even if the local minimas can be avoided by jumping over them, the system can jump forward and backward a global minima without ever reaching it. On the other, a two small learning rate results in incredibly long training times.

• Networks are also sensitive to the number of neurons in their hidden layers. Too few neurons can lead to underfitting, which appears when there is no set of weights such that the network can produce outputs reasonably close to the targets. Conversely, when overfitting occurs, each data point is fit perfectly, but the fitting curve varies wildly between points. This is due to an overtrained network with noise existing in the training data.

• Training time can largely vary depending on the initial conditions. Choosing the relevant weights instead of random numbers is not very easy but can significantly reduce the training time *[NGUY90]*.

A few additional methods applied to the basic backpropagation algorithm can solve at least part of the problems previously detailed *[VOGL88]*. A method called momentum decreases backpropagation's sensitivity to small details in the error surface and helps avoiding to be trapped in local minimas.The use of an adaptive learning rate which attempts to keep the learning step size as large as possible while keeping learning stable can also decrease the training time.

We have introduced in the last two parts two families of learning methods based on what is called supervised learning, i.e. learning with a teacher. These networks require to have available example patterns that include targeted outputs. Unsupervised learning (or Self-organization in Networks) doesn't need to rely on a teacher which tells who to classify the training data. Such networks can learn to detect regularities and correlations in their input and adapt their future responses to that input accordingly. Kohonen's work on this matter has led to other considerations than supervised learning methods and based on biological experiments *[KOHO87] [KOHO91]*.

2.2 Prototype of card

Integrating neural networks in smartcard applications has to take into account two major parameters:

• The small memory space available for the network storage in the EEPROM of the card.

• The limited processing power due to today's 8-bit processors embedded into smartcards.

Several options are offered for integrating the neural network technology in smartcards. We have seen in the previous section that a neural network application was divided into two stages:

• A learning phase where the weights are updated to respond to a training set of patterns.

• An interrogation phase that corresponds to the use of the neural network.

The feed forward process needed in the second stage has only to be done once for each query and therefore can easily be achieved by the card. Only a few multiplications and additions are needed. The complexity of integration mainly concerns the learning phase because many epochs of the training data are required for adjusting the weight values. The resulting time needed for these computations can be a significant problem for a small environment like a smartcard.

Neural Networks is a recent technology. Because associated basic mathematics and design procedures haven't been firmly established and applied for many years, the set up of neural network applications is still mainly based on experiments. We have decided to detail in this paper two possible implementations of Radars, resulting from our experience and experiments. Although many different network configurations and input criteria can be applied, we will try to justify our intuitions.

2.2.1 Perceptron algorithm on the expert system

The first implementation of Radar we are proposing is using the alarms generated by an expert system and is synthesizing these data for making the final decision.

In a prior study of the Radar concept *[ALEX94]*, we have set up an expert system in the Prolog language *[BRAT90][SPEN91]*. The goal was to generate alarms as soon as a transaction was suspicious or unusual. The total of these alarms was giving a score for making the final decision.

The alarms can be of different natures and therefore shouldn't be considered at the same level. That's the reason why we have thought of constructing an additional small neural network for evaluating the final decision. As the expert system was already time consumer we needed to implement a simple architecture capable of classifying the alarms.

We first experimented a single layer Perceptron because of its simplicity. The chosen network is described below:

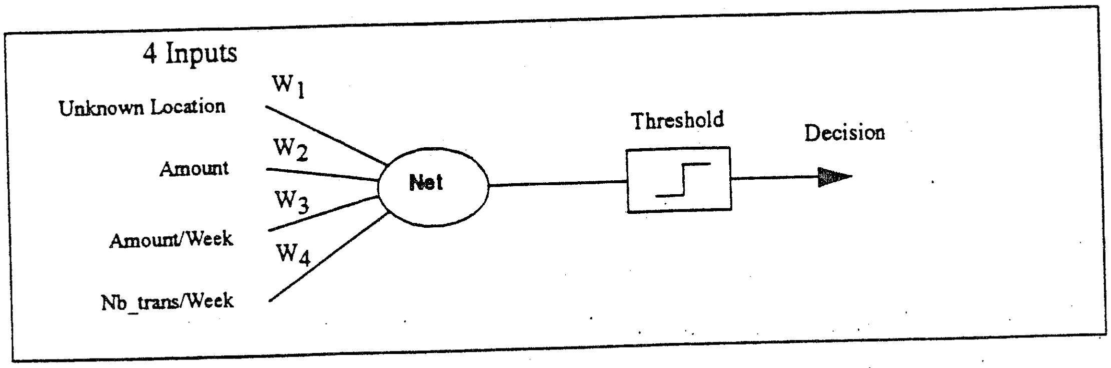

FIGURE 9. Single neuron with 4 inputs

Each of the 4 inputs of the network corresponds to a particular alarm type and has a value of 1 when activated, 0 otherwise. These alarms include:

•Unknown Location: activated when the *Location* parameter of the transaction is not in the Location Table, i.e. the list of the most common places where the card owner uses to achieve transactions.

•Amount Overflow: activated when the *Amount* of the transaction is higher that the defined maximum for a particular *Nature*.

•Amount/week Overflow: activated when the total amount of all transactions of *Nature* type exceeds the declared threshold for the week.

•Nb_trans/week Overflow: activated when the number of transactions of *Nature* type the same week is too high.

We first trained a network of this type with a set of data representing a particular boolean function using the Matlab environment *[DEMU92]*. The network learnt to classify the input data into two classes, a one on the output muting the card. The chosen boolean function is detailed below:

I_1	I_2	I_3	I_4	Out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Inputs

$I_1 \rightarrow$ Unknown Location

$I_2 \rightarrow$ Amount

$I_3 \rightarrow$ Amount/Week

$I_4 \rightarrow$ Nb_trans/Week

FIGURE 10. Example of training function

Giving random weights and bias for starting has led to the following results after convergence (i.e. when all the training examples are classified correctly):

W = [1,3586 5,8694 5,7670 8,0388] and bias = [-12,33] after 14 epochs

Depending on the initial weight values, the training time will vary and the final weights will differ from one simulation to another because there exists several solutions. The preceding network is now equivalent to our boolean function.

Implementing this network on a smartcard remains very simple. We only have to store the four weights obtained by the training phase and execute in the card the feed forward procedure. Integers may replace the real values in the calculation of the weights to simplify the process (W=[2 6 6 9] bias=-12 is also a solution). Because we have attributed fixed output values to the training set, the role of the neural network is limited with this solution because only one neural network is defined for everybody.

A second step is to take into account the owner's habits in the training process. This can be done by fixing the outputs of the boolean function so that the card is never muted considering the last 6-month-transactions of the card's owner.

We have completed a simulation on a single layer Perceptron similar to the preceding but with a datapreprocessing on the past transactions figures for determining a boolean function adapted to the owner. For a given knowledge base defining the owner's habits *[ALEX94]*, the expert system can estimate which alarms are activated when scanning the last 6-months transactions. The outputs of the boolean function below have been calculated so that none of these transactions can result in muting the card.

Experiments have given the following figures:

I_1	I_2	I_3	I_4	Out
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Inputs

$I_1 \rightarrow$	Unknown Location
$I_2 \rightarrow$	Amount
$I_3 \rightarrow$	Amount/Week
$I_4 \rightarrow$	Nb_trans/Week

Training results:

W = [2,4379 2,0941 2,3577 12,3586]

Bias = -5,1306

Number of epochs: 17

FIGURE 11. Boolean function established with an existing set of bank transactions

Although this neural network process has a very simple implementation in a smartcard (the card has only to achieve one single feed forward computation, the training being performed outside the card), this solution has limitations in terms of compactness and efficiency due to the following reasons:

• If the data corresponding to the output of the boolean function are not linearly separable, the network will never classify correctly all of the training data. This is a limitation of single layer networks.

• Adjusting the behavior recognition process for each owner can be done by changing parameters of different natures:

- the threshold values included in the knowledge base describing the owner's habits.
- the outputs of the network training data.

These two parameters are related to each other. Trying to find a compromise between them remains difficult.

• Such a Radar requires to implement both the expert system and the small neural network and therefore is a heavy memory space consumer.

We have advanced other investigations concerning Radar implementations not related to boolean functions set up but more oriented toward the major interests of neural network technology which are the ability to classify data intended to pattern recognition, identification, and a learning faculty to resolve complex non linear problems.

2.2.2 A Radar based on the Backpropagation algorithm

In this section we have abandoned the expert system technology to build a full neural network solution of Radar. We have chosen the Backpropagation learning rule because this efficient algorithm is used in perhaps 80 to 90% of today's practical applications.

Inputs of the network are no longer alarms but the transaction figures themselves defined by the four items <Nature, Amount, Date, Location>. In order to take into account a significant parameter related to the *Date* and not the *Date* itself which means nothing for the network we have computed as a datapreprocessing the *Amount/Week* and the *Number_of_transactions/Week*.

Amount and *Number_of_transactions* are figures and therefore can be compared. We can easily put in order (for example from the highest value to the lowest) the 3 different amounts (1000, 300, 1400) because we have a classification criterium. Here the obvious result is 1400>1000>300.

Conversely, we can't put in order *Natures* or *Locations* of transactions by saying Transportation>Cash or Paris>Lille. We need to involve in the input process a binary encoding technique, i.e. one input for each *Nature*. For a particular transaction only one of these inputs will be activated at the same time as below:

Nature	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
Cash	1	0	0	0	0	0	0	0
Transportation	0	1	0	0	0	0	0	0
Gas	0	0	1	0	0	0	0	0
Food	0	0	0	1	0	0	0	0
Housing	0	0	0	0	1	0	0	0
Furniture	0	0	0	0	0	1	0	0
Hobby	0	0	0	0	0	0	1	0
Other	0	0	0	0	0	0	0	1

Location	I_{12}	I_{13}
in Location Table	1	0
Not in Location Table	0	1

FIGURE 12. Binary coding of *Nature* Inputs and *Location* Inputs

The training set of data has been made of a mix of existing bank transactions achieved during a 6-month-period and «fraudulent» transactions modelized by asking to a few people what kind of transactions they would do if they had a stolen card with the appropriate PIN code.

The network, consisting of 13 inputs fed to a two-layer network, is trained with back-propagation on a 28 pattern problem.

Layer 1 consists of 5 tan-sigmoid neurons, while layer 2 is made of one linear neuron. The network is initialized with the Nguyen-Widrow method [NGUY90]. An adaptive learning rate which adjusts to be as large as possible without taking too large of steps speeds up the network while avoiding large increases in error. The momentum technique to avoid being trapped in local minimas is also implemented.

We have written all the network description and the training algorithm using the Matlab programming language *[DEMU92]*.

We first established that the training process was over when the total sum squarred error of the network was inferior to some particular value (0.02 for example).

We then realized that the end of the training process could occur when the error of each training vector was inferior to 0.5, which corresponds to the difference between a Target (0 or 1) and an actual network output. This last condition is good enough to classify correctly the pattern (a network output > 0.5 will be interpreted as a target of 1 while a network output < 0.5 will be seen as a target of 0).

The simulation results are summarized in the following diagram:

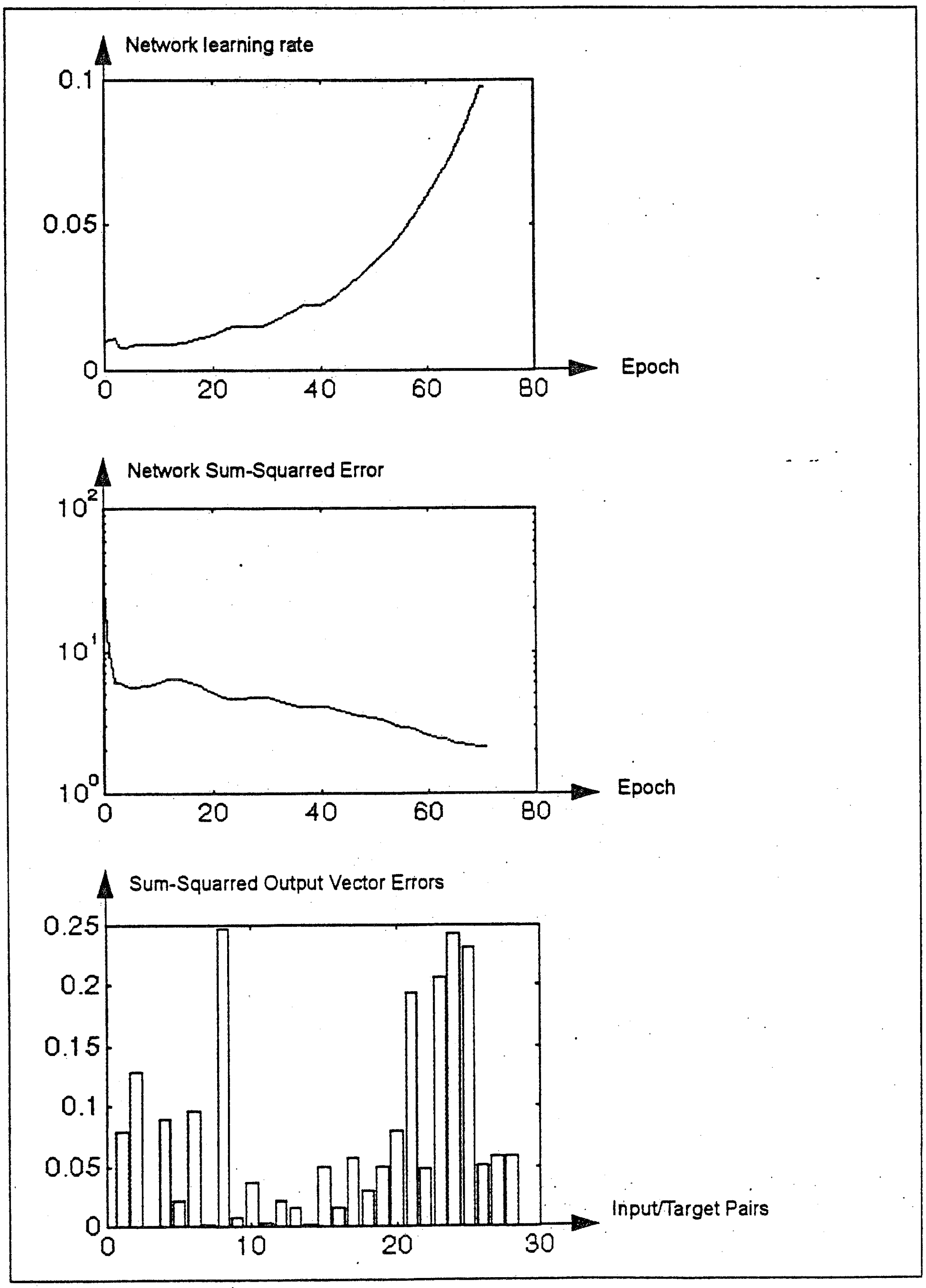

FIGURE 13. Simulation results for a Radar based on backpropagation

Training took 71 epochs to correctly classify the data, representing about 3 seconds on a 486 microprocessor running 50Mhz.

We have tested this newly trained network on new figures both coming from new transactions achieved by the card owner and transactions completed by intruders, leading to an approximate 90% of success. This evaluation has only concerned a few transactions to give an idea of the usefulness of the Radar but should be further experimented on additional intrusion strategies to get a more precise performance estimation.

The memory space required for the network storage consists of 13 x 5 weight values + 5 biases for the first layer and 5 weights + 1 bias for the second layer. Assuming weights and biases are stored as real values (4 bytes), the total network size to be recorded on the EEPROM of a smartcard represents 304 bytes.

To evaluate the performance of the feed forward process completed on a smartcard, we have examined an existing smartcard algorithm written for the CASCADE project, which is the definition of a smartcard 32-bit RISC processor featuring biometric recognition based on neural networks *[PEYR94]*. We have estimated that the Radar algorithm (feed forward process) applied on such a processor would only consume less than 300 bytes of ROM and less than 30 bytes of RAM, one feed forward process time being about 5000 clock cycles.

Conclusion

There still remains a significant amount of research to be done in determining exactly which aspects of behavior are most indicative of intrusions. To obtain meaningful indicators of intrusion behavior, such research needs to have available many examples of actual intrusions. A library of such examples does not currently exist and is needed.

In order to effectively address the various intrusion threats, a system should combine several intrusion-detection approaches, as emphasized in the NIDES project *[LUNT93]*.

Because of the limited capability of the smartcard in terms of storage ability and processing power, the only algorithm that seems to be easy to implement in the card is the feed forward process. The next step would be to set up a relevant training algorithm targeted to run into a smartcard. Thus the network could continue to learn after each transaction. The obvious drawback to this approach is the amount of time required for each training session, unless to have available an internal clock for continuing the training even when the transaction is over and the card disconnected.

References

[ALEX94] Thomas Alexandre, Vincent Cordonnier, «The Radar Concept», in *Proceedings of the CardTech/SecurTech'94 International Conference*, Arlington,Virginia, USA, April 10-13, 1994, p87-98.

[ANDE93]	D.Anderson, T.F.Lunt,H.S.Javitz, A.Tamaru, A.Valdes, «Detecting Unusual Program Behavior Using the NIDES Statistical Component», Safeguard Final Report, SRI International, Menlo Park, California, December 1993.
[BLOCK]	H.D.Block, «The Perceptron: a model for brain functionning», *Reviews ofModern Physics 34:* 123-135.
[BRAT90]	Ivan Bratko, *Prolog Programming for Artificial Intelligence,* second edition, Addison-Wesley, 1990.
[DAVA93]	Eric Davalo, Patrick Naïm, *Des Réseaux de Neurones,* Eyrolles, Paris, 2nd edition, 1993, 232 pages.
[DEMU92]	H Demuth, M.Beale, Neural Network Toolbox for Use with Matlab, The MathWorks Inc., 1992.
[GELE91]	E.Gelenbe, *Neural Networks: Advances and Applications,* Elsevier Science Publishers, 1991.
[JAGA93]	R. Jagannathan, A.Tamaru, F.Gildham, D.Anderson, C.Jalali, C.Dodd, H.S.Javitz, A.Valdes, T.F.Lunt and P.G.Neumann, *«Next Generation Intrusion-Detection Expert System (NIDES): Software Design Specifications»,* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, March 1993.
[KOHO87]	Teuvo Kohonen, *Self-Organization and Associative Memory,* 2nd edition, Berlin: Springer-Verlag, 1987.
[KOHO91]	T.Kohonen, K.Mäkisara, O.Simula, J.Kangas, «Artificial Neural Networks», in *Proceedings of the 1991 International Conference on Artificial Neural Networks,* Espoo, Finland, 24-28 June, 1991.
[LUNT93]	Teresa F. Lunt, «Detecting Intruders in Computer Systems», in *Proceedings of the 1993 Conference on Auditing and Computer Technology.*
[MINS69]	M.Minsky, S.Papert, *«Perceptrons»,* The MIT Press, 1969.
[NGUY90]	D.Nguyen, B.Widrow, «Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights», *Internal Joint Conference of Neural Networks,* vol3, pp.21-26, July 1990.
[PEYR94]	P.Peyret, «RISC-based, Next-generation Smart Card Microcontroller Chips», in *Proceedings of the CardTech/SecurTech'94 International Conference,* Arlington,Virginia, USA, April 10-13, 1994.
[ROSE57]	F.Rosenblatt, «The Perceptron: a Perceiving and Recognizing Automaton», Project PARA, Cornell Aeronautical Lab. Report 85-460-1, 1957.
[ROSE61]	F.Rosenblatt, *Principles of Neurodynamics,* Washington D.C.: *Spartan Press,* 1961.
[RUME86]	D.E.Rumelhart, G.E.Hinton, R.J.Williams, «Learning internal representation by error propagation», D.Rumelhart and J.McClelland editors, *Parallel Distributed Processing: exploring the Microstructure of Cognition,* Vol.1, MIT Press, Cambridge, MA, 1986.

[SPEN91] R.Spencer-Smith, *Logic and Prolog,* AI Group, Middlesex Polytechnic, A Harvester Wheatsheaf Publication, 1991.

[VOGL88] T.P.Vogl, J.K.Mangis, A.K.Rigler, W.T.Zink, D.L.Alkon, «Accelerating the convergence of the backpropagation method», *Biological Cybernetics*, Vol.59, pp.257-263, 1988.