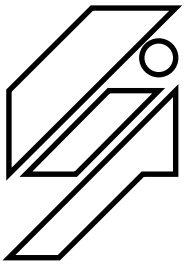
Knowledge Extraction From Neural Networks : A Survey

Richard Baron



Laboratoire de l'Informatique du Parallélisme

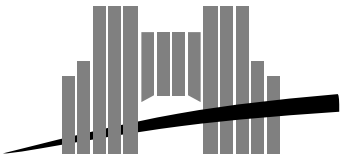
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Knowledge Extraction From Neural Networks : A Survey

R. Baron

March 1994

Research Report N^{O} 94-17



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Knowledge Extraction From Neural Networks : A Survey

R. Baron

March 1994

Abstract

Artificial neural networks may learn to solve arbitrary complex problems. But knowledge acquired is hard to exhibit. Thus neural networks appear as "black boxes", the decisions of which can't be explained. In this survey, different techniques for knowledge extraction from neural networks are presented. Early works have shown the interest of the study of internal representations, but these studies were domain specific. Thus, authors tried to extract a more general form of knowledge, like rules of an expert system. In a more restricted field, it is also possible to extract automata from neural networks, likely to recognize a formal language. Finally, numerical information may be obtained in process modelling, and this may be of interest in industrial applications.

Keywords: Artificial neural networks, knowledge extraction, expert systems

Résumé

Les réseaux de neurones artificiels peuvent apprendre à résoudre des problèmes complexes. Mais les connaissances qu'ils peuvent acquérir sont difficilement utilisables. Aussi apparaissent-ils comme des "boîtes noires" dont les décisions ne peuvent être expliquées. Dans ce rapport, plusieurs techniques permettant l'extraction de connaissances sont présentées. Les premiers travaux dans ce domaine ont montré l'interêt de l'étude des représentations internes, mais ces travaux sont très dépendants du domaine d'étude. Aussi, certains auteurs ont essayé d'extraire des connaissances sous une forme plus générale, comme des règles d'un système expert. Dans un autre domaine, il est également possible d'extraire, à partir de réseaux de neurones, des automates reconnaissant un langage formel. Enfin, des informations d'ordre numérique peuvent être obtenues dans la modélisation des systèmes, ce qui peut présenter un interêt pour des applications industrielles.

Mots-clés: Réseaux de neurones artificiels, extraction de connaissances, systèmes experts

Knowledge Extraction From Neural Networks : A Survey

R. Baron
Laboratoire de l'Informatique du Parallélisme
ENS-Lyon CNRS
46 Allée d'Italie
69364 Lyon Cedex 07
France.
e-mail: rbaron@lip.ens-lyon.fr

May 27, 1994

1 Introduction

Multilayer feedforward networks are universal approximators [HSW89]. Thus, they may be used to realize a mapping between an input and an output. Several works have been done which use artificial neural networks in real world applications. But one of their main drawbacks lies in the fact that they appear as "black-boxes". They are unable to explain their decisions, that is, how a given output is associated to a given input, which is a drawback for real world applications. Moreover, since learning algorithms modify cells in the hidden layer, this may constitute interesting internal representations.

For these two reasons, a few authors studied internal representations, and tried to extract knowledge from artificial neural networks. In this study, we will show different ways which have been used. The first section describes the first works which showed the interest of studying internal representations built during the learning phase. The second section presents different ways for extracting rules from neural networks, which is the kind of knowledge used in traditional Artificial Intelligence expert systems. In the third section, we describe works about the extraction of Finite State Automata, that recognize regular languages. Finally, in the fourth section, we present works on sensitivity analysis, which is a way of extracting information about the function implemented by a neural network.

2 Study of Internal Representation

Research which focused on internal representations in artificial neural networks began with the first multilayer perceptrons. For that kind of architecture, weights in the hidden layers are modified during the learning phase to comply with the data to be learned. Thus a specific internal representation is built, which depends on the problem to be solved. It was assumed that this representation may be interesting: it is a kind of knowledge, acquired by the neural network. Moreover, acquiring knowledge on real word applications can still be a difficult task for human experts, when faced with a great amount of data. This is especially true for knowledge engineers who are supposed to extract knowledge from human beings.

2.1 Interest of Internal Representations

Rumelhart et al. [RHW86] first took interest in the study of internal representations while developing the back-propagation algorithm. In their paper, they use a multilayer feedforward network to store family ties between persons of a same family. The neural network stores the ties for two families, an English one and an Italian one. The two family trees are isomorphic (fig. 1). The network learns relations by learning triples of the form $< person1 >< relation >< person2 >$. The relations are learned once the neural network can find the third argument when given the first two.

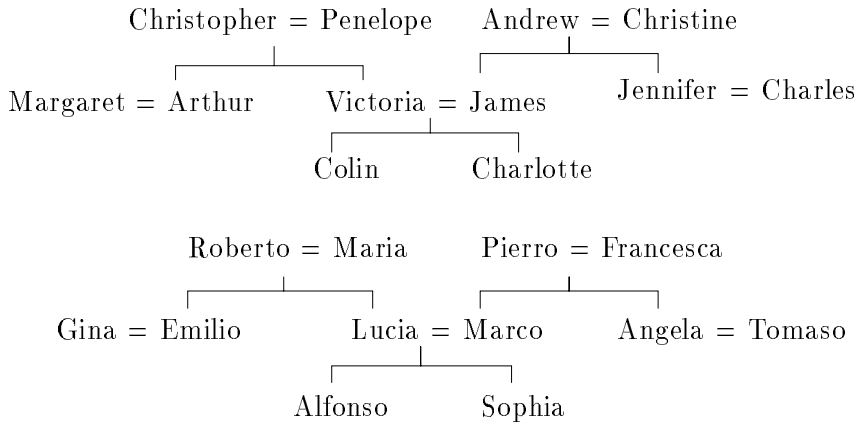


Figure 1: Two isomorphic family trees

The network architecture is composed of one input layer, three hidden layers, and one output layer (fig. 2). The input layer is divided in two parts: 24 cells represent the first person, 12 cells represent the relation. The output layer is composed of 24 cells representing the second person. The learning phase uses the backpropagation algorithm, with 100 relations as training examples.

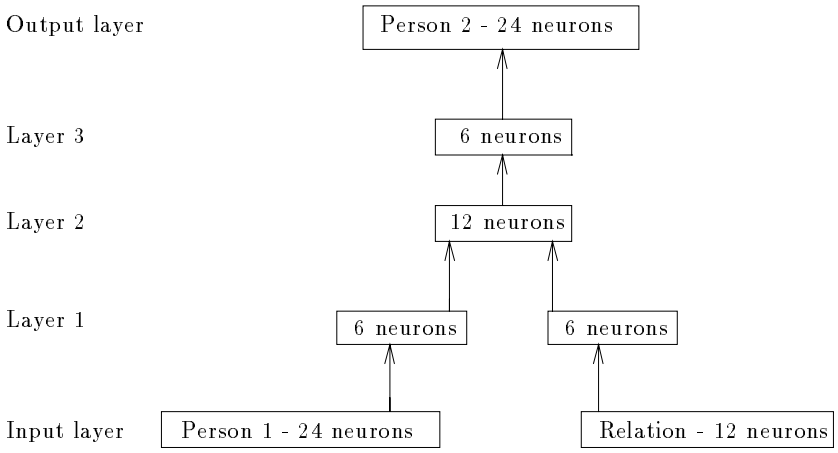


Figure 2: Neural network for storing family ties

Then, the authors analyse the weights between the input layer coding the person and the first hidden layer. This analysis shows that it is possible to extract semantic information from the neural

network. For example, the first hidden cell can be interpreted as representing the nationality of the input person: the weights connecting this cell to two input cells coding two persons at the same place in the both trees are opposite. In the same way, the second hidden unit encodes the generation, and the sixth encodes the branch of the family.

2.2 Learning Signal Characteristics

The same kind of observations have been made by Gorman & Sejnowski [GS88], in a very different field. They use a neural network for classifying sonar returns from metal cylinder and cylindrically shaped rock (fig. 3). After preprocessing, the spectral envelope of the signal gives the power spectral density for each frequency. This constitutes the input signal for the neural network.

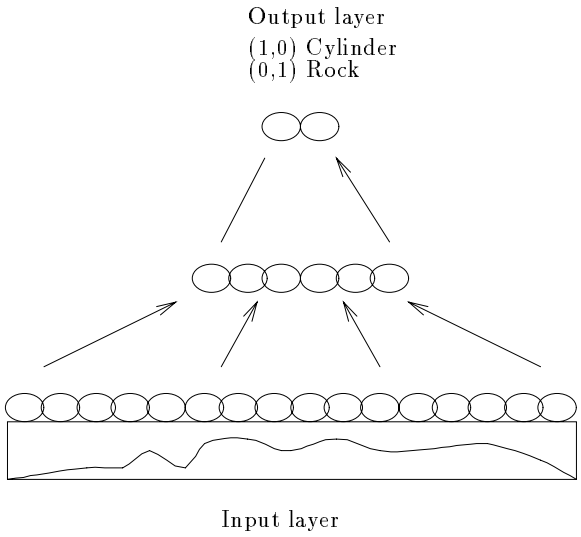


Figure 3: Neural network architecture for sonar signal

This network is composed of 60 input units, several hidden units, and two output units coding the class of the input signal. The backpropagation algorithm is used during the learning phase, with several numbers of hidden cells. A network with three hidden cells is finally chosen because of its performance : 100 % correct classification on the training set and 93.7 % on the test set.

One of the purposes of this work is to study the internal representation of the neural network, and its link with the ability to classify correctly. This study is made through the observation of the weight state vector, which is the product of the input vector by the weights connecting the input to an hidden unit (See figure 4).

It shows that the weights at the extremes of the input layer (corresponding to higher and lower frequencies) are inhibitory. Thus they turn off the hidden units for wide band signals. Moreover there are alternating bands of positive and negative weights. If the onset or the decay of the signal is gradual, positive and negative values cancel one another, thus shutting down hidden units. If the decay rating is rapid, the activation is not balanced and results in a positive response. For wide band signals, corresponding to cylinder returns, hidden units are poorly activated, while they respond to narrow band signals, corresponding to rock returns. If no hidden unit is activated, the bias drives the output layer to code for a cylinder. A weak response from any of the hidden units

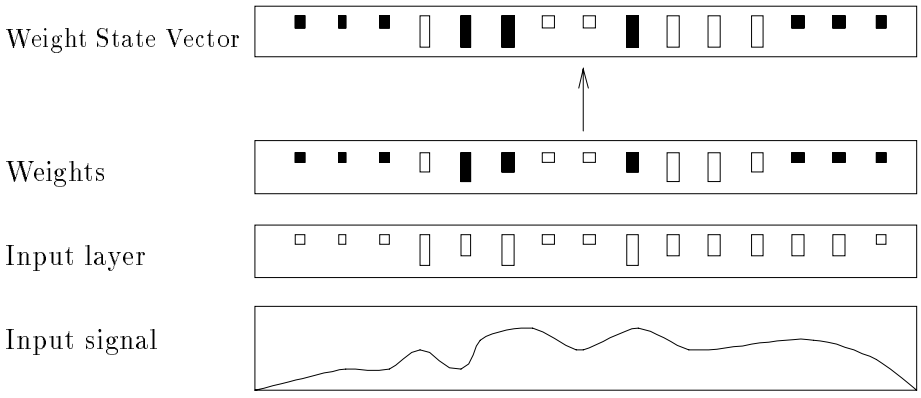


Figure 4: A weight-state vector. White and black rectangles represent positive and negative weights, respectively. The area of a rectangle is proportional to the absolute value of the weight.

is sufficient to code for a rock.

Notice that hidden units are tuned to various central frequencies, such that the network can recognize signals centered on different peak frequencies. Even wide band rock signals are recognized, though they do not conform to general descriptions. This correct classification is achieved by coding for the precise location of peaks and nulls in the signal by positive and negative weights between input and hidden layers. Thus internal representation built by the neural network is a kind of filter adapted to different features during learning. But, according to the authors, this kind of pattern coding is more suggestive of a model based approach rather than simple feature extraction.

2.3 Extracting Semantic Information in Bridge Game

In the same way of studying internal representations, Bochereau & Bourgine [BB89] take interest in a neural network which learns to play bridge game. Precisely, given a "hand" of playing cards, the network is supposed to give the corresponding first call. It is composed of an input layer of 52 units, representing the cards, a hidden layer, and an output layer of 11 units, corresponding to the call to be made (fig. 5). After the learning phase, using the backpropagation algorithm, the authors found that the best results were obtained with 5 hidden units.

It also appears that these units get "specialized". First, four of these units are specifically adapted to each color : for each one, the weights connecting it to cells of the input layer of one color are stronger (for example the second hidden unit is strongly connected to input cells representing cards of the diamond color). Second, the authors distinguish two kinds of hidden units, "hard neurons" and "soft neurons". For hard neurons, the majority of the input values lies in the interval [-2,+2]. For soft neurons, the majority of the input values lies outside this interval (See figures 6 and 7).

This remark allows the authors to extract semantic information from the neural network, after a few manipulations on the weights. First, weights are clustered : weights which have nearly equivalent values are replaced by their average value. If the output of a neuron j is given by the equation $s_j = f(w_0 + \sum_j w_{ij} \cdot x_j)$, it will be replaced by $s_j = f(w_0 + \sum_k w_{ik} \cdot \sum_{j_k} x_{j_k})$ (f is a standard sigmoidal function. x_{j_k} is the output of neuron j, which belongs to kth cluster of cells). Second, if x_{j_k} is the output of a hard neuron, as is the case for output neurons, x_{j_k} may have two possible

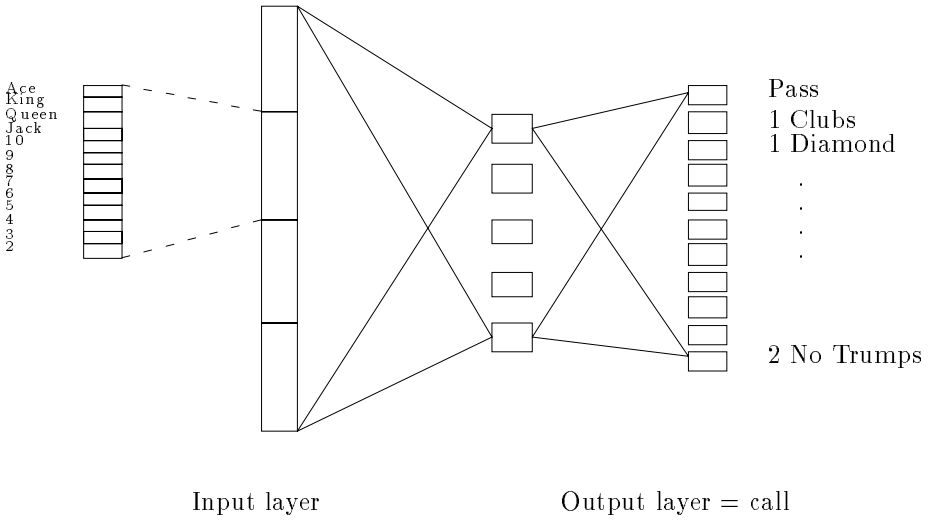


Figure 5: A neural network for playing bridge

values, 0 or 1. If a cell has to be activated at the value s, the relation $f(w_0 + \sum_k w_{ik} \cdot \sum_{j_k} x_{j_k}) \geq s$ must be verified, which gives $\sum_k w_{ik} \cdot \sum_{j_k} x_{j_k} \geq f^{-1}(s) - w_0$. Because x_{j_k} may have two values, a simple algorithm can be used to enumerate all sets $\{x_{j_k}\}$ satisfying the relation.
Using these two heuristics, the authors extract semantic information from the neural network (See table 1). For example, the input from the first hidden unit is linearly dependent on the number of points in a hand (see table 1), and the other hidden cells count the number of cards in each color. There are relations between inputs of the hidden cells, and consequently between their output values. By constraint propagation, authors can thus extract rules concerning the first call to be compared with rules for an expert system.

Neuron	Type	Interpretation of input	Condition (Treshold = 0.5)
N1	Soft	$-0.2 * P_H + 3.2$	$P_H < 16$
N2	Hard	$-2 * P_H + 5.4 * N_K$	$N_K > 0.37 * P_H$

Table 1: Example of information extracted at a threshold of 0.5. P_H stands for the number of points in a hand, and N_K the number of kings.

These experiments show the interest of the study of internal representations of neural networks which solve a specific problem. It also appeared that extracting knowledge may be possible, as in the example of the bridge game. But no general solution is provided for extracting knowledge. Extracted knowledge is domain specific. It would thus be of interest to have some general method of extraction, available for different problems. One way would be to extract rules from neural networks, in order to build Expert Systems.

3 Rules Extraction

One advantage of expert systems is that they provide a general way of working out a problem. A knowledge base may be built for each domain, but the inference mechanism, which uses these

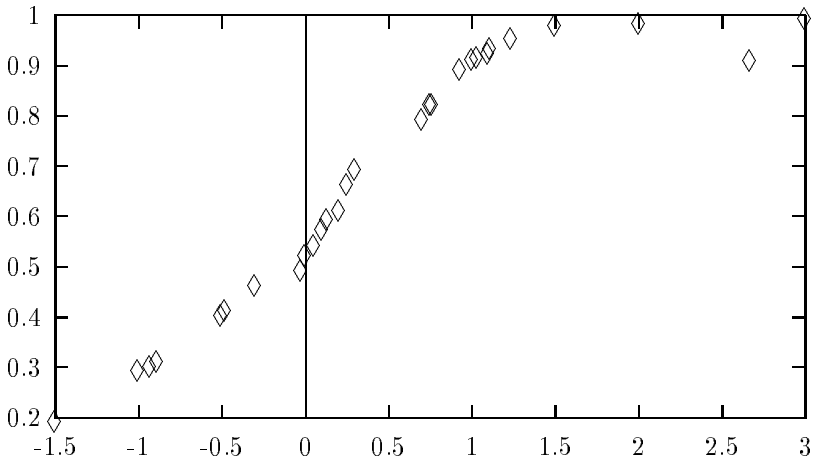


Figure 6: Output of a soft neuron, as a function of its input value. Each diamond corresponds to one value of the output when one input is presented to the neural network.

knowledge, remains the same. It is usually admitted that the most difficult task is the knowledge base construction, which is entirely domain specific. Knowledge engineers encounter problems when acquiring knowledge from human experts, for both technical and psychological reasons. Thus, authors try to ease the knowledge-acquisition bottleneck, especially with the help of a connectionist model.

Gallant [Gal88] uses a neural network as a connectionist expert system in medical diagnosis. This network is a three-layer network. Cells in the input, hidden and output layers respectively represent symptoms, diseases and treatments. There are connections between each successive layer and between input and output layers. A few cells are added for increasing the discriminating capacity of the network. The output of the cells are three-valued : +1, -1 and 0 corresponding respectively to logical values $True, False$ and $Unknown$. The learning phase is performed using Pocket Algorithm [Gal86]. Then, inferencing can be made given several input values : it is possible to deduce the activation for a cell u_i without knowing the values of all its inputs. Let u_j be the inputs to cell i, then, if

$$\sum_{j:u_j known} w_{ij} \cdot u_j > \sum_{j:u_j unknown} |w_{ij}|$$

then the conclusion is as follows:

$$u_i = \begin{cases} +1 & if \quad \sum_{j:u_j \ known} w_{ij} \cdot u_j > 0 \\ -1 & if \quad \sum_{j:u_j \ known} w_{ij} \cdot u_j < 0 \end{cases}$$

Backward chaining is also possible. It allows the network to ask for unknown values by questioning the user. The values to be asked for are selected using a heuristic. One interesting point is that the system can partly "explain" its decision, by producing **if-then** rules when a cell has a known value. For example, if u_7 has the value +1:

1. List all units which contribute to positivity

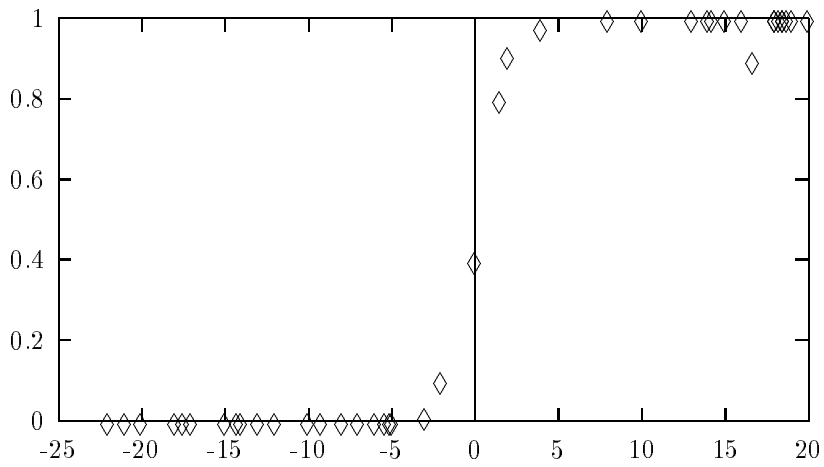


Figure 7: Output of a hard neuron, as a function of its input value

2. Arrange the list by decreasing order of weights
3. Generate **if-then** rules until $\{\sum_{u_i \ used \ in \ clause} |w_i|\} > \{\sum_{u_i \ unused} |w_i|\}$

This produces rules as follows :

```
IF u2 is False
   and u5 is True
THEN Conclude that u7 is True
```

In Gallant's article, this method is used to induce rules in medical diagnosis domain. These rules were judged "reasonable" by doctors. Other systems have been built in different domains, such as management decision, but no other results are reported. According to the author, the connectionist expert system described may be used as a tool for knowledge engineers, not as a replacement: a lot of rules can be extracted, but they are not always pertinent. Thus, the rule base may not be of interest for human experts. In fact, such a technique is supposed to make the knowledge base construction easier, and would be of use in fields requiring great amount of rules.

A non real world problem was treated in the Connectionist Scientist Game [MMS91]. It deals with rules that map strings of symbols. The rules are composed of a *condition part* and an *action part*. The condition is a feature or a combination of features to be satisfied. The action part describes the mapping to be performed on the input string. Three types of conditions are allowed :

- presence of a symbol s at a specific place (or *slot*)
- a conjunction of two conditions ($\wedge$)
- a disjunction of two conditions ($\vee$)

For example, the rule $[\wedge A\_G \rightarrow 32B]$ concerns the strings with a symbol A at the first place, *and* a symbol G at the third one. If the rule is applied, the *action* part is used. Numbers in the *action* part denote the place of the symbol of the input string. The 3 at the first slot of $32B$, indicates that the first symbol of the output string is the *third* of the input string (i.e. a G), and the 2 at the second slot indicates that the second symbol of the output string is the *second* of the input string. Finally, B is a constant symbol of the output string. Using this rule, the string AEG would be replaced by GEB.

In the article, strings have a length of $n = 4$ symbols, and the alphabet is composed of 8 symbols: {A,B,...,H}. The input layer of the neural network, called *Rule Net*, codes the input string : it is made of n bit-strings of length k. In these bit-strings, one bit is 1, indicating which symbol of the alphabet is present, while the others are 0. The output layer uses the same code. Two subnets are between these layers: the *condition subnet*, and the *action subnet*.

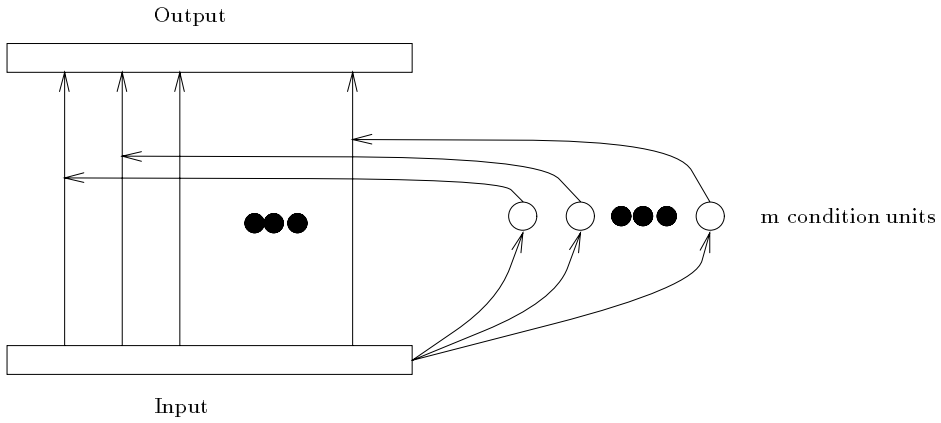


Figure 8: The *RuleNet* architecture

In the *condition subnet*, the net input to each *condition unit* i is computed

$$net_i = \frac{1}{1 + e^{-\boldsymbol{c_i^T}.\boldsymbol{x}}}$$

where $\boldsymbol{x}$ is the input vector and $\boldsymbol{c_i}$ is the incoming weight to *condition unit* i. The activity p_i is then determined by normalization:

$$p_i = \frac{net_i}{\sum_j net_j}$$

The *action subnet* is composed of m weight matrices $\boldsymbol{A_i}$. A set of multiplicative connections between each *condition unit* i and $\boldsymbol{A_i}$ determines to what extent $\boldsymbol{A_i}$ will contribute to the output vector $\boldsymbol{y}$, calculated as follows:

$$\boldsymbol{y} = \sum_{i=1}^{m} p_i \boldsymbol{A_i} \boldsymbol{x}$$

Ideally, only one condition unit is fully activated by a given input. Weights are adapted using the backpropagation algorithm, with constraints on $\boldsymbol{c_i}$ and $\boldsymbol{A_i}$ to keep a valid semantic. Only one bit of $\boldsymbol{c_i}$ can be active in bit substrings of length k. $\boldsymbol{A_i}$ is formed of $k \times k$ submatrices, which must be either the identity or the zero matrix. During the learning phase, $\boldsymbol{c_i}$ and $\boldsymbol{A_i}$ are modified using

a process called *projection*, which ensures that the network can be interpreted as a set of symbolic rules.

Simulations compare results obtained with different learning techniques, depending on whether the *projection* process is used or not, and on the number of rules in the rule base. The performance takes into account the percentage of patterns which are correctly classified, and the number of valid rules extracted. It shows that the use of the *projection* process gives the better results. The number of rules extracted is exactly the same as this of the rule base used to generate the strings. Compared with a multilayer network with 15 hidden units, *Rule Net* has better generalization results.

Another way of extracting knowledge has been developed by Saïto & Nakano [SN90] called the RN Method. It deals with rules of the form $u_1 \in [.2, .8]\ \&\ u_3 \in [.3, .7] \cdots$. The idea of the algorithm described below is to grow a region containing a training example correctly classified as +1, by changing one dimension at a time, until the region bumps into a +1/-1 network boundary for the dimension. Then, the algorithm considers the hyperrectangle which has been built, and substracts out misclassified negative examples by the same method (see figure 9).

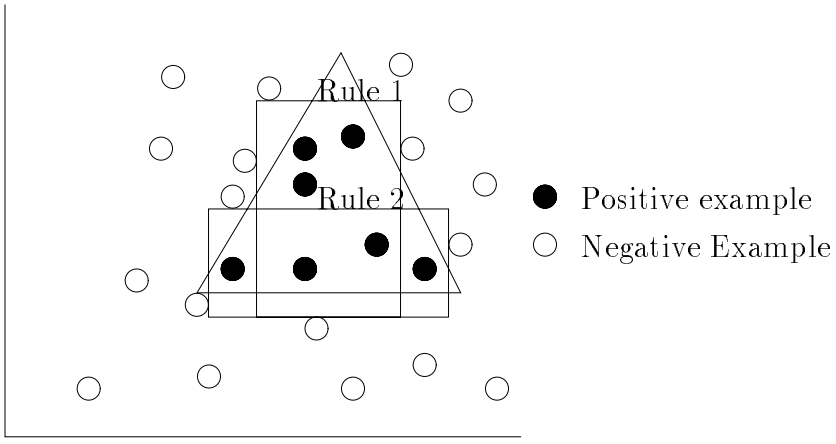


Figure 9: Example of rule generated by the RN Algorithm for a 2-D problem. Positive examples are picked in the triangle. Rectangles indicate domains generated by extracted rules.

RN Algorithm: Given a neural network $\mathcal{N}$ and a set of positive examples $\{E^k\}$

1. Pick a positive training example E^k not yet covered by any term of the rules
2. For each input variable j :
   - Find a range around E^k that $\mathcal{N}$ classifies as positive (other variables unchanged)
3. Intersect all such ranges (with *And* connector, as in the example of rule given above). This gives a new term $\mathcal{T}$
4. For each negative example E^l misclassified by $\mathcal{T}$
   - Find range for E^l as in step 2, and substract ranges from $\mathcal{T}$ (thus modifying $\mathcal{T}$)
5. If some positive example remain, goto 1
6. Join all terms by *OR*

These works point out the possibility of extracting explicit rules from neural networks. Some rule bases produced were judged "reasonable" by experts of the domain, although measuring the quality of a rule base remains a difficult task. Recent works are studying the possibility of inferring fuzzy rules, likely to treat uncertainty, as neural techniques do [Glo91, HG92].

4 Automaton extraction

Another original way of extracting knowledge from neural networks lies in the extraction of finite state automata, which recognize a language $\mathcal{L}$. The problem is *grammatical inference*: find a procedure to infer the syntactic rules of an unknown grammar G based on a finite set of strings $\mathcal{I}$ from L(G), the language generated by G, and possibly, on a finite set of strings from the complement of L(G). More precisely, inferring a Deterministic Finite Automaton (DFA) , is finding a DFA which accepts the positive strings and rejects the negative ones. Chen et al. [GMC+92] use a neural network to perform this task. First the network learns to classify the strings, and second, a procedure extracts a Deterministic Finite Automaton (DFA), likely to recognize positive and negative strings. The network is a second order recurrent neural network (see figure 10). The authors use the grammars on the alphabet {0,1}, and more specifically, on the language of all strings not containing "000" as a substring. The neural network is composed of N recurrent hidden units S_j, L nonrecurrent input neurons I_k, and weights W_{ijk}. At a time t, one character of a string is presented to the input neurons. Activity of the hidden neurons is then computed following the equation:

$$S_i^{(t+1)} = g(\sum_{jk} W_{ijk} \cdot S_j^{(t)} \cdot I_k^{(t)})$$

where g is a sigmoidal function.

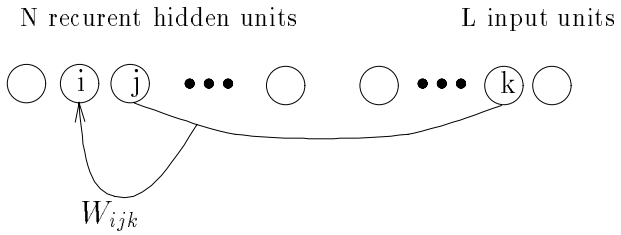


Figure 10: Neural second order recurrent architecture. Weight W_{ijk} connects cells j and k to cell i.

The output neuron S_0 is either set on ($S_0 > 1 - \epsilon$) if an input string is accepted, or set off ($S_0 < \epsilon$) if the string is rejected. During the training phase, weights are updated using a gradient descent technique, after each string presentation. The network is said to converge when all training samples are well classified. The extraction process may occur during or after training. It includes the following steps:

1. Clustering of the DFA states
2. Construction of the transition diagram
3. Construction of the full di-graph
4. Reduction to minimal representation

The idea is that the neural network partitions its state space in different regions, corresponding to states in some finite state automaton. Consequently, each neuron's range [0,1] is first divided into q partitions. Since there are N hidden neurons this gives q^N possible states. Then, the DFA is generated by giving the symbol which makes the network changing from one state to another one. Finally, the extracted DFA is reduced, using a standard algorithm. It appears that second order reccurent neural networks are capable of learning small regular grammars, and of well generalizing on unseen grammatical strings (*see also* [WK92]). It is even possible to extract the exact minimal DFA of the language. This DFA outperforms some of the trained neural networks in correct classification of unseen strings. Thus, neural networks may be considered as tools for extracting a DFA that would be representative of an unknown grammar. This would be a possible way for solving the problem of *grammatical inference*.

5 Sensitivity Analysis

Artificial neural networks may be used for determining relationships between input and output variables. A successfully trained neural network maps input vectors $\vec{X}$ from a n-dimensional space to output vectors $\vec{Y}$ in a m-dimensional space. It can be expressed as:

$$\vec{Y} = f(\vec{X})$$

where $\vec{Y} = (y_1 y_2 \ldots y_m)^T$ and $\vec{X} = (x_1 x_2 \ldots x_n)^T$. Then, $\frac{\partial y_i}{\partial x_j}$ measures the change in y_j when x_i is changing. Thus, it is representative of how sensitive y_i is with respect to x_j. This is a kind of knowledge which may be of interest in complex systems. As an example, Uhrig & Guo [GU92] use a neural network to determine important measures in a nuclear power plant system. They want to control the variation of the plant thermal performance. One measure, *heat rate*, frequently changes, and may cause a loss of energy.

The authors use sensitivity analysis method applied to a neural network, for determining the influence of several variables on *heat rate*. The network is a combination of self-organizing and backpropagation neural networks. There are 24 inputs and 2 outputs. The self-organizing network works as an organizer, and rearranges the original training patterns in clusters. Then, the centroids of these clusters are used as inputs for the multilayer perceptron. This network has 24 input units, 10 hidden units, and 2 outputs. Once a reasonable error rate is reached, the derivatives are computed. It appears, that for that kind of network, these derivatives are functions of the weights in the network, and of the input pattern. Thus, it must be averaged over all input patterns. These values may be ranked in the order of sensitivity. The greater the derivative, the more important the input variable. This implies that a small change in this input variable is likely to affect the output variable. The authors applied the method to the *heat rate*, and tried to apply the method further to secure information. There is no comparison with classical methods such as Principal Component Analysis.

Another study of Hashem used higher order derivatives, with a very simple sinusoidal function [Has92]. It showed that the first and second order derivatives approximations are less precise, than the function approximation itself. But according to the author, such a method would be of interest in process modeling.

6 Conclusion

In this survey, we relate different techniques aiming at extracting knowledges from artificial neural networks. Initial research in this field have been initiated after the backpropagation algorithm has been developed, which allows neural networks to build internal representations. Studying these internal representations showed the interest of knowledge extraction. But such a method is entirely domain specific. Thus, authors studied the way to extract rules, as rules used by expert systems. It has been proven to be possible. Still the capacity of a neural network to treat numerical information is lost in a traditional expert system: modelling uncertainty or using incomplete information is difficult in classical expert systems. Consequently, research are now focusing on the extraction of fuzzy rules. Numerical informations may also be extracted, using sensitivity analysis. This method can be used in complex process modeling, when knowledge extraction is difficult because of the amount of data. Finally, the problem of grammatical inference could be solved, using connectionist techniques. Artificial neural networks can learn to recognize positive and negative strings of a formal language. Then, an automaton can be extracted from this network. All these ways of extracting informations from neural networks seem to be promising. Successful experiments have been reported. Thus, artificial neural networks may not be "black boxes" anymore. And they can provide a way to learn informations about complex problems.

References

[BB89] L. Bochereau and P. Bourgine. Implémentation et extraction de traits sémantiques sur un réseau neuro-mimétique : Exemple de la première annonce au bridge. In *Neuro'Nimes 89*, pages 125–141, Nimes, France, November 1989.

[Gal86] S. I. Gallant. Optimal linear discriminants. In IEEE Press, editor, *Proceedings of the 8th International Conference on Pattern Recognition*, pages 849–852, Paris, France, October 1986.

[Gal88] S. I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, February 1988.

[Glo91] P.Y. Glorennec. Un reseau "neuro-flou" evolutif. *Neuro'Nimes 91*, pages 301–314, November 1991.

[GMC+92] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, May 1992.

[GS88] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[GU92] Z. Guo and R. E. Uhrig. Sensitivity analysis and applications to nuclear power plant. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 453–458, Baltimore, June 1992.

[Has92] S. Hashem. Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions. In *Proceedings of the International Joint Conference on Neural Networks*, volume I, pages 419–424, Baltimore, June 1992.

[HG92] S.K. Halgamuge and M. Glesner. A fuzzy-neural approach for pattern classification with the generation of rules based on supervised learning. *Neuro'Nimes 92*, pages 165–173, November 1992.

[HSW89] K. Hornik, S. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[MMS91] C. McMillan, M. I. Mozer, and P. Smolensky. The connectionist scientist game: Rule extraction and refinement in a neural network. Technical report, Department of Computer Science and Institute of Cognitive Science, University of Colorado, 1991.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[SN90] K. Saito and R. Nakano. Rule extraction from facts and neural networks. In *INNC-90 Paris: Proceedings of the International Neural Netwok Conference*, volume 1, pages 379–382, July 1990.