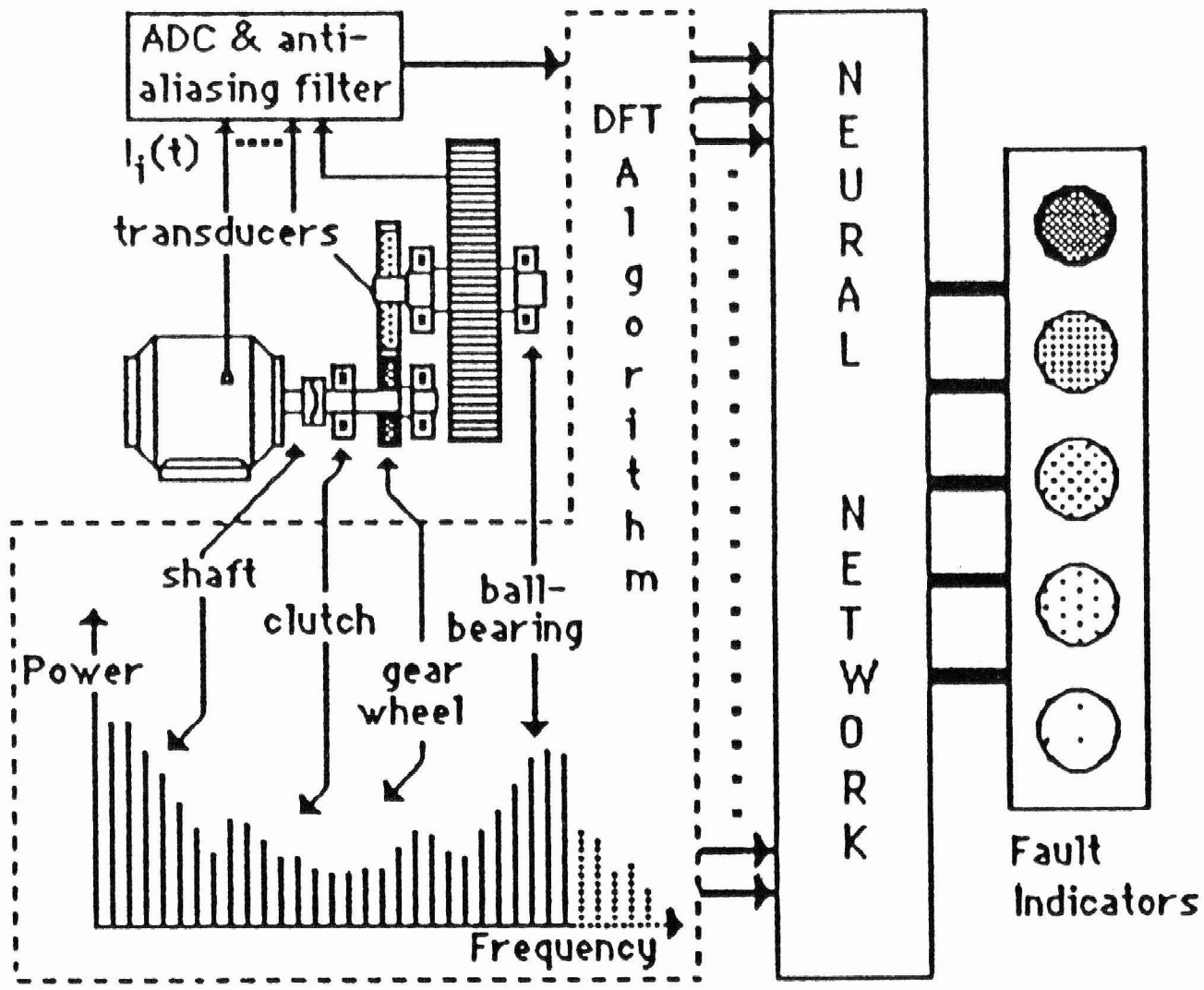
IDG

NEURAL NETWORK WORLD

International Journal on Neural and Mass-Parallel Computing and Information Systems

VOLUME 1 1991 NUMBER 3



Kerckhoffs E. J. H.: A View on Problem-Solving Paradigms Including Neurocomputing

Cimagalli V., Balsi M., De Carolis A.: Information Storage in Neurocomputing

Jiřina M.: Binary Model of Neural Net

Vítková G., Míček J.: Knowledge Processing by Neural Networks

Hořejš J.: A View on Neural Network Paradigms Development (Part 3)

NEURAL NETWORK WORLD is published in 6 issues per annum by the Computer World Company, Czechoslovakia, 120 00 Prague, Blanická 16, Czechoslovakia, the member of the IDG Communications, USA.

Editor-in-Chief: Dr. Mirko Novák

Associate Editors: Prof. Dr. V. Hamata,
Dr. M. Jiřina,
Dr. O. Kufudaki

Institute of Computer and Information Science, Czechoslovak Academy of Sciences, 182 07 Prague, Pod vodárenskou věží 2, Czechoslovakia.

Phone: (00422) 82 16 39, (00422) 815 20 80, (00422) 815 31 00

Fax: (00422) 85 85 789,

E-Mail: CVS15@CSPGCS11. BITNET

Price Information:
Subscription rate 399 US $ per annum.
One issue price: 66.50 US $.
Subscription adress: ComputerWorld Co., Czechoslovakia,
120 00 Prague 2, Blanická 16, Czechoslovakia.

Advertisement: Ms. M. Váňová, Ms. Ing. H. Vančurová,
ComputerWorld Co., Czechoslovakia, 120 00 Prague 16,
Blanická 16
Phone: (00422) 25 80 23, Fax: (00422) 25 73 59.

Scanning the Issue

Editional 129

Kerckhoffs E.J.H.: **A View on Problem-Solving Paradigms Including Neurocomputing** 129

The spectrum of the complex problem-solving simulation and knowledge-based expert systems is surveyed with respect to methodological, functional and application aspects. The role of parallel processing in these is discussed. Some neural-network oriented projects running at the Delft University of Technology are mentioned

Cimagalli V., Balsi M., De Carolis A.: **Information Storage in Neurocomputing** 155

The concept of relational information stored in neurocomputer is presented and the method for its measurement both in dynamic systems and in arbitrary neural network is proposed

Jiřina M.: **Binary Neural Net** 163

In this contribution a special case of binary neural net is introduced. The inputs and outputs have values 0 or 1 and also the weights have values just −1,0 or 1. The thresholds are set so that the overall function of neuron is either logical sum or logical product of direct (the weight +1) or inverted (the weight -1) inputs. Several variants of adaptive dynamics are described. This kind of network represents a simplest purely digital kind of neural net. It is interesting from the point of view of realization by digital technology.

Vítková G., Míček J.: **Knowledge Processing by Neural Networks** 171

The basic properties of neural associative memory and the main abilities of fuzzy cognitive maps are described. The associative memory extrapolation based knowledge system for diagnostics is presented.

Hořejš J.: **A View on Neural Network Paradims Development (Part 3)** 185

Literature Survey 162, 170, 183, 184, 192

ISSN 46999

Editorial

Being interested to bring to the readers a good selection of interesting contributions arising from the East and the West in if possible equal range, we have chosen from the increasing amount of manuscripts which were send to us for the third issue of our Journal: two papers from the Netherlands, one from Italy and two from Czechoslovakia.

We also took care to the balance among theoretically oriented papers, the papers presenting the overall general views and the papers oriented more to applications.

From the last point of view the paper prepared by Vítková and Míček is hoped to be interesting. The paper written by Kerckhoffs opens very inspiring insight on the development of complex user oriented simulation and knowledge systems supporting the human experts activity in several significant areas, like banking, transportation control and engineering systems faults prediction, detection and protection.

This represents the main part if this issue content. Beside this, we insert in this issue some information on papers and books which appeared in the last time. Of course, the preference is given to the original contributions and the above mentioned information is therefore presented in limited extend only.

M.Novák

A VIEW on PROBLEM-SOLVING PARADIGMS INCLUDING NEUROCOMPUTING

Eugene J. H. Kerckhoffs

Connectionism is considered as a problem-solving paradigm among other methodologies such as (numeric) simulation and (symbolic) reasoning. In order to create still more powerful and useful problem-solving tools, simulation systems, knowledge-based expert systems and connestionist systems can, at least in principle, be coupled. The spectrum of these so-called "coupled systems" (or "hybrid systems") is surveyed with respect to methodological aspects, functionalities and practical applications. The emerging role of parallel processing when dealing with the more complex systems in either domain is discussed. Finally, some neural-network application projects currently running at Delft University of Technology (the Netherlands) are briefly dealt with; they might illustrate some of the issues considered.

1. Introduction

Engineers are assumed to solve (technical) problems in engineering, such as physical, chemical, mechanical, and civil engineering. At the universities students are instructed on how to solve problems in various disciplines. "Problem solving" is a major issue in daily life. There exist several paradigms to solve problems. Problems can be solved a.o. by calculation, by reasoning, by learning and subsequent generalization, or by a combination of these. It depends on the particular problem concerned and the circumstances what is most efficient.

Let us consider a simple, even trivial, example to set the scene. Suppose we have to organize a number of matches, each with two players; the loser of a match is out, the winner continues the competition. The total number N of players is 16, and the problem is how many matches are to be organized. In the very essence, this is a computing problem. The straightforward way to solve the problem is by numeric calculation. The total number of matches is: (16/2) + (8/2) + (4/2) + (2/2) = 15. However, the straightforward or obvious way to solve a problem does not necessarily need to be the most efficient. For example, the problem considered here could be solved by reasoning. Instead of on the match winners we focus on the losers. At the very end there is only one (fi-

*) Delft University of Technology
The Netherlands

nal) winner, hence we have M = N-1 losers in total. Since each match delivers one loser, we clearly have to organize M, hence 15, matches. With respect to the value of N (the number of players), obviously the reasoning method is the most efficient, i.e. the effort to solve the problem is entirely independent of this value. (Note: if the problem was to find the number of matches in each round, only the numeric computing method is feasible).

Under certain conditions there is another way to solve the above problem. Suppose, we have "learning examples": for N = 3, 9, and 11 it is known that the number of matches to be organized is respectively 2, 8 and 10. "Generalization" of this immediately shows that with 16 players 15 matches should be organized. This is a really efficient way to solve the problem, because given the learning examples we immediately see the solution without consciously thinking as in the above cases of computing and reasoning. The latter paradigms can, of course, be used to verify the solution we "feel" automatically.

The computer era, in which we live today, allows to make extensive use of computers and computer-based techniques to solve the real-world problems we are confronted with. Also for computer-based problem solving the above-mentioned paradigms and their combinations are feasible: we distnguish *numeric simulation systems* for computer-based calculation, *knowledge-base (expert) systems* for computer-based reasoning, *artificial neural networks (ANNs)* for computer-based learning and generalization, coupled simulation / expert systems, coupled expert systems / ANNs, and the like. It depends on the problems concerned and the circumstances what (combination of) tools and techniques are most efficient. One of the key issues for future problem solving is: *integration*; various tools and techniques will be integrated in order to enhance their capabilities and compare alternative solutions with respect to, for instance, efficiency. Integrated environments to run simulations, expert systems, artificial neural networks, and combinations of these (compare this with the existing integrated packages for word processing, spreadsheets and databases) could perhaps provide — along with intelligent front- and back-end systems — the ideal toolboxes for problem solving in the future.

2. Some Problem-Solving Approaches in a Nutshell

In this paper we focus on model-based approaches to problem solving. As shown in Fig. 1 we may distinguish (continuous and/or discrete) numeric models, (rule-based and/or frame-based) symbolic models, connectionist models and all possible combinations of these, covered by the collective term "coupled models" (or "hybrid models"). In contrast with numeric and symbolic systems, the are designed to model part of the real-world problem domain we are interested in, connectionist models are inspired by the functionality of the brain. They reflect modelling at a different level. Their network topologies may well depend on the kind of application concerned (such as optimization, classification, organization, and adaptive control).

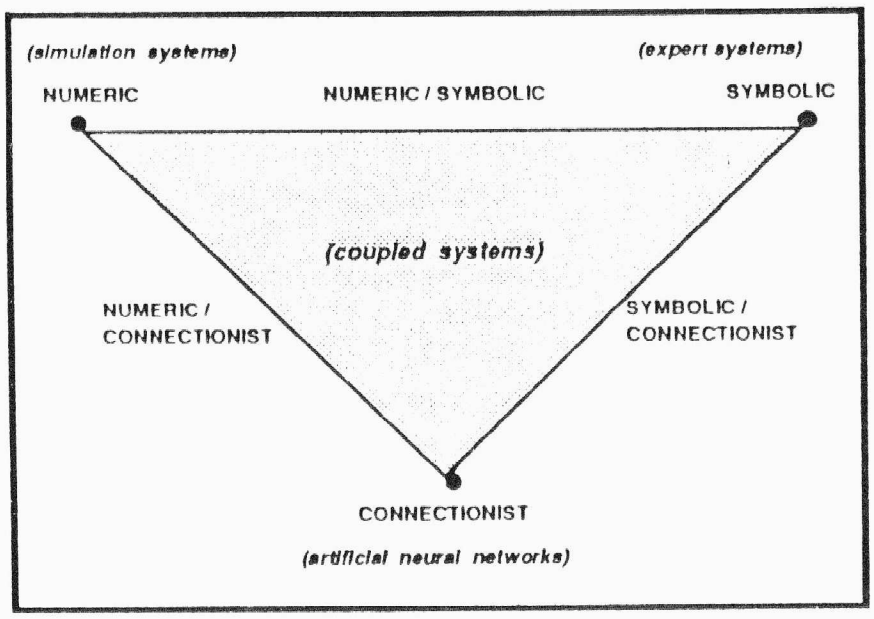


Figure 1: The spectrum of numeric, symbolic, connectionist and coupled models

In principle the numeric, symbolic and connectionist paradigms can be coupled in order to create still more powerful, more useful and more user-friendly problem-solving tools. The interdisciplinary use of knowledgebased, numeric and connectionist methods is still largely prescientific; the research community has hardly begun to establish a rigorous methodology for developing coupled systems. The current state-of-the-art in coupled systems, although evolving especially in the coupled numeric/symbolic field [Kowalik and Kitzmiller, 1988], [Widman et al., 1989], are not particularly sophisticated. Most existing coupled systems employ relatively simple coupling schemes and combine the simpler methods of each discipline.

In general, it is useful to think of three methodological levels of coupling [Widman and Loparo, 1989]: shallow, deep and "very deep". (It should be noted that the usage of the terms "shallow" and "deep" differs from the usage of these in describing expert-system knowledge bases; see section 2.2). In shallow coupling, a system treats another one as a black box to be called as needed. In deeply coupled systems, in a way a system has additional "knowledge" about another system that is coupled to it. In "very deep" coupling there is no real distinction between the separate systems; they are fully integrated.

In the technical sense, it is more common to speak of loosely coupled, tightly coupled and fully integrated systems. In loosely coupled systems, the communication relies mainly on simple file interfaces; variations may include pre- and postprocessing by one system of data provided by another system. In tightly coupled systems, the communication is established by data passing. Variations may include blackboard-system, cooperating-system and embedded-system techniques. In fully integrated systems, it is hard to dis-

cern any separate modules and communication is accomplished through a dual nature of the structure.

All of these have their benefits and limitations. As for the loosely coupled systems the benefits include model simplicity, ease of development, usability of commercially available software, and reduced maintenance time, whereas limitations are operation speed, communication overhead, overlapping data gathering and redundancy in the development process.

Benefits of tight coupling include reduced communication times, increased run-time performance, retained modularity and being more robust and sophisticated than loose coupling. Limitations are the increased development and maintenance complexity, limited possibilities in using available software and more difficult testing.

The benefits of fully integrated systems are operation speed and resource utilization, elegance and robustness of the model, flexibility in development and no redundancy in data gathering. Limitations, however, are the complexity in conceptual and design issues, increased development time and resource requrements, the lack of commercially available tools, and increased complexity in testing and maintenance.

2.1 Numeric simulation systems

Computer simulation is the problem-solving process of predicting the future state of a real-world system by studying a (more or less idealized) computer model of this. Simulation experiments are usually performed to achieve predictive information that would be costly or impractical to obtain with real devices. Typical applications would include determining the most energy-efficient design for a reactor, the best mix of reactants to maximize production of a certain product, and the optimum capacity and lay-out of a factory. Ultimately, information gained from simulation experiments should contribute to decision making with respect to real-world systems modelled by the simulations. Other purposes of simulation could be the accessibility and documentation of knowledge about specific real-world systems, and education and training.

There are two major types of simulation: continuous and discrete. Continuous simulation predicts the behavior of systems that can be described by (ordinary and partial) differential equations, such as electrical, mechanical, thermal, and fluid devices. Discrete simulation predicts the behavior of event-driven systems, such as manufacturing plants, purposeful movements of people such as in bank queues, and message traffic on networks. Typically, these event-driven systems use stochastic processes to model unknown influences on the system.

Building and using a simulation model is a skilled process requiring expertise in a number of theoretical fields including statistics, systems analysis, and numerical analysis. Also, practical rules of thumb and experience are needed to use simulation as an effective tool. Simulation studies normally follow some well-defined subsequent steps with possible feedbacks (see Fig. 2):

- Problem specification (result: detailed abstract problem description)
- Selection of modelling method, conceptual model description (result: tool-independent mode description)
- Selection of solution techniques and tools, realization of an executable model (result: tool/computer-dependent model)
- Model validation (result: validated model)
- Experiment planning, performing model experiments (result: simulation results)
- Analysis and interpretation of results.

The programming languages available for simulation include the general-purpose languages such as FORTRAN, PL/1, C and Pascal. Specialized languages have evolved for certain types of simulation. Continuous simulation is supported by DYNAMO, CSMP, ACSL, CSSL IV, and many others. Discrete-event simulation is supported by process-oriented languages such as GPSS and SIMSCRIPT II.5, and object-oriented languages such as SIMULA and SMALL-TALK-80. As the limitations of these languages have become more apparent, hybrid languages combining features of several types of simulation have become available (e.g. PROSIM, COSMOS, SLAM II, SIMAN).

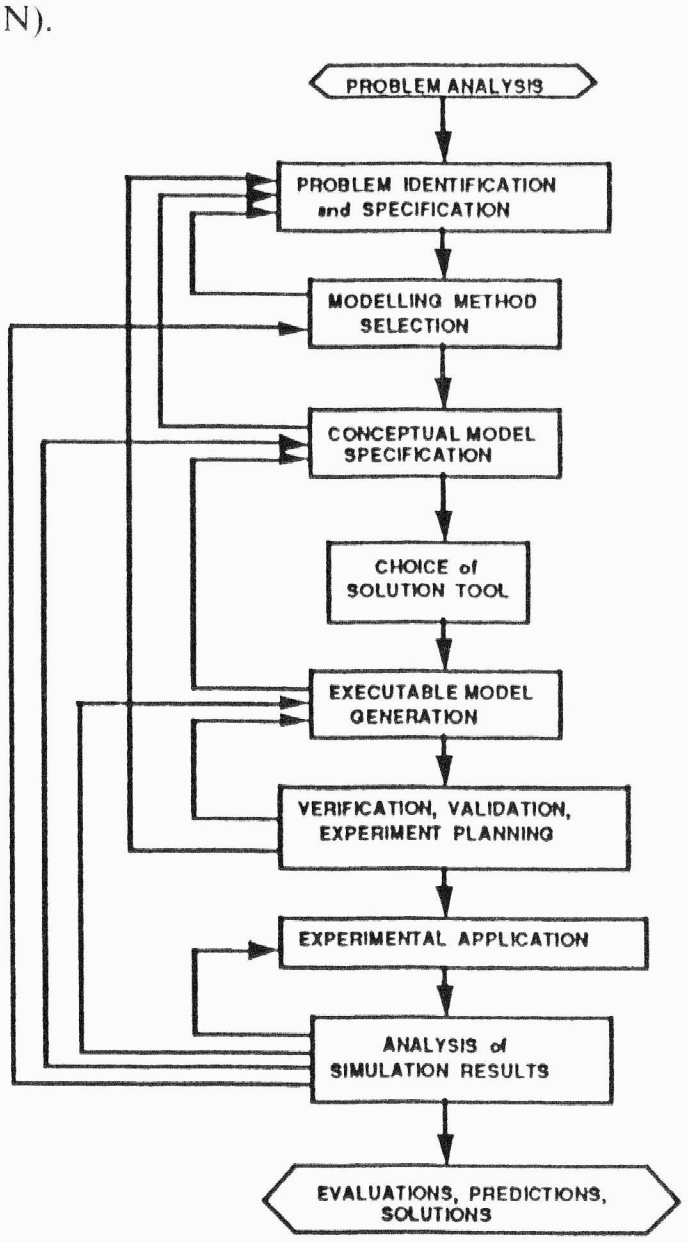


Figure 2: Stages in a typical modelling and simulation process

There exist many good introductory textbooks in the simulation field covering various aspects such as simulation methodology [Zeigler, 1976 & 1984], mathematical modelling techniques (Spriet and Vansteenkiste, 1982], continuous simulation (Roberts et al., 1983] and discrete simulation [Neelamkavil, 1987].

2.2 Knowledge-based (expert) systems

Artificial intelligence (AI) programs are assumed to perform tasks that, if performed by a human being, would be considered intelligent. AI includes many fields, such as robotics, natural-language understanding, machine vision, character and speech recognition, and machine learning. The area that currently has the largest impact outside the research setting, however, is that of knowledge-based (expert) computer systems. The terms "knowledge-based systems" and "expert systems" are commonly used to indicate the subfield of AI that deals with reproducing human experts' behavior given a problem within their specialisms. A knowledge-based system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution; if the human expertise in a specific narrow domain is emulated, we speak of a (knowledge-based) expert system [Harmon and King, 1985], [Johnson, 1984].

Expert systems typically consist of a knowledge base, which includes the expert knowledge available, and an inference engine, which contains the control structures that enable the program to use the knowledge base.

Modern expert systems contain one or more methods of knowledge representation, and one or more control algorithms.

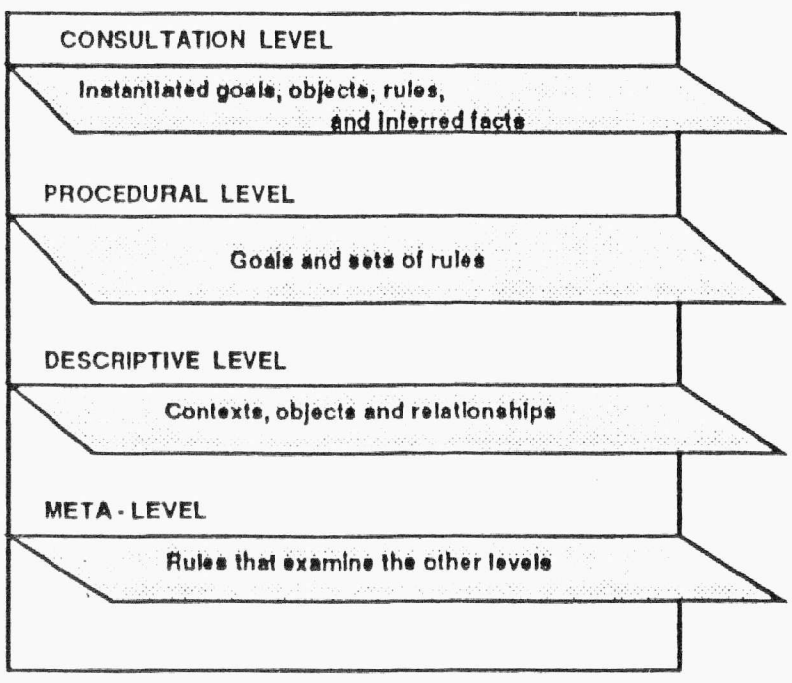


Figure 3: Knowledge at different conceptual levels

Knowledge can be represented at different conceptual levels (see Fig. 3): consultation, procedural, descriptive, and meta-level. Depending on the knowledge-representation technique used, knowledge-based systems can be distinguished between:

- rule-based systems (knowledge at the procedural and — if the system is being used — consultation level), embodying independent chunks of knowledge (production rules), and
- object-oriented or frame-based systems (knowledge at the descriptive, procedural and — if being used-consultation level).

In the latter case, the developer starts by describing objects and their relationships; some of the objects might have rules associated with them, and these rules would create a procedural level. Structured rule-based systems and context trees are between the above-mentioned extrema.

For both above types of knowledge representation the standard control algorithms are forward and backward "chaining". Backward chaining is a reasoning method starting with the desired goal (goal driven); in forward chaining the reasoning proceeds from input data (data driven). A major strength of these algorithms is their ability to deal with uncertainty. Uncertainty can arise from noisy, unavailable or incorrect data and incomplete or self-contradictory expert knowledge. Uncertainty is quantified by "confidence factors" with each datum and each rule, yielding in the reasoning process to confidence factors for the conclusions and recommended actions.

Most expert systems that nowadays are in widespread use are of the "first generation" (i.e. *shallow* reasoning systems based on empiric rules of thumb). They have shown limitations with respect to representing time-varying phenomena, performance outside the narrow range of expertise, ensuring consistency in the knowledge base, and the ability to learn form errors. Some of these are addressed in the "second-generation" and even "third-generation" expert programs. In second-generation expert programs the knowledge is primarily captured in (*deep*) models of the expert domain rather than in empiric rules of thumb. In contrast to production rules, that capture the empiric mapping from causes to effects without asserting causality, these models capture explicitly causal relationships in the system being modelled. As these deep models have become more complex, their application has become progressively more difficult. Third-generation expert systems focus on learning or parameter tuning from examples; they still are rare.

AI programming has traditionally been done in the LISP and in the PROLOG language. These are however not unique in supporting AI programming; some large and complex expert computer programs have been written in conventional languages such as C, Pascal, Modula and even FORTRAN. LISP-to-C converters are becoming more popular because C is faster and more widely available.

In addition to the above languages, there exist dedicated software packages that support the development and application of expert systems: "on-the-shelf" sys-

tems and expert system shells (or tools). Shells provide the user more general support than (AI-) languages do; languages are however more flexible. Three well-known hybrid tools with multiple knowledge representations and inference techniques) are: ART, KEE and Knowledge Craft. These tools are LISP-based and run on LISP-machines (such as Symbolics and TI-Explorer) or advanced workstations. KEE has been ported to PC, which is an interesting development. Meanwhile, a collection of PC-based hybrid tools has appeared on the market, which exploit the enhanced capacity of PCs (like 386-based PCs) to offer a representational power comparable to the larger systems at moderate prices. Examples are: Gold Hill's GoldWorks, Neuron Date's Nexpert Object, TI's Personal Consultant Plus, and Intellicorp's 386-based version of KEE.

For a general introduction to AI concepts, the reader is referred to [Tanimoto, 1987]. Expert system technology is reviewed in [Hayes-Roth et al., 1983], [Buchanan and Shorlife, 1984] and [Szolovits, 1987].

2.3 Coupled simulation / expert systems

In the sections 2.1 and 2.2 there was talk of two historically distinct approaches to model reality: numeric (quantitative) and symbolic (qualitative) modelling. Since the mid 1980s the gap separating these two schools has narrowed. Both the numerical modelling community and the AI community have found that AI can contribute to simulation and simulation to AI. The growing cross-fertilization of ideas between the fields of AI and simulation is evidenced by the inclusion of special sessions at the major simulation conferences since 1985. Also in 1985, the first (stand alone) Working Conference on "AI and Simulation" in Europe was held at the University of Ghent, Belgium [Kerckhoffs et al., 1986]. In the AI community, the US National Conference on Artificial Inteligence held its first Workshop on AI and Simulation in 1986.

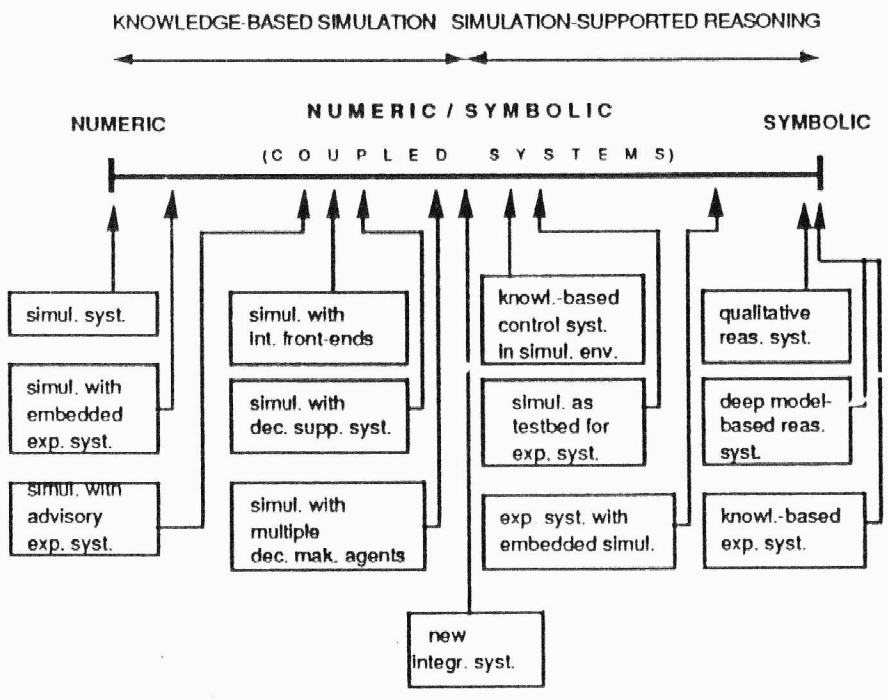


Figure 4. The spectrum of numeric/symbolic systems

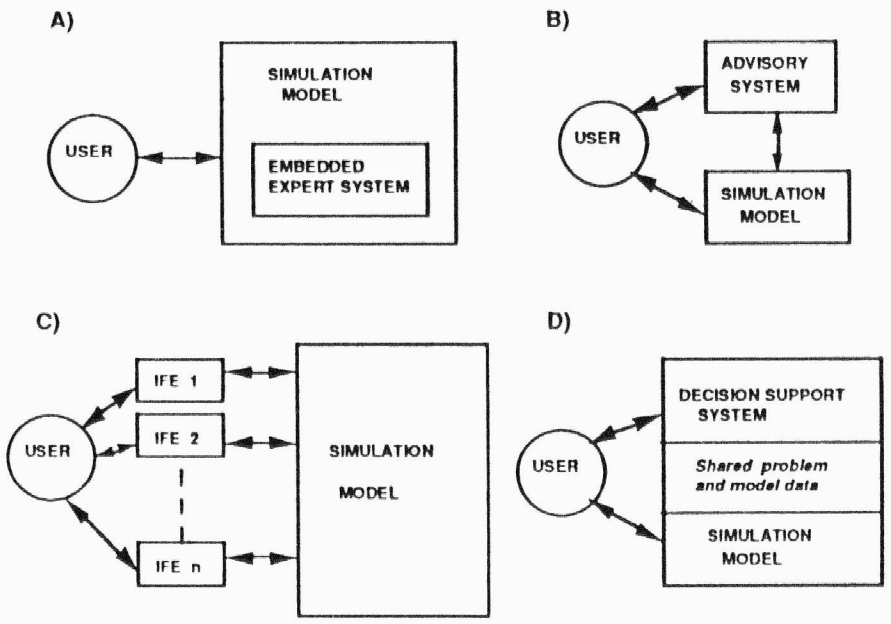


Figure 5: Expert systems in simulation: A) embedded, B) advisory, C) intelligent front-ends (IFEs), and D) decision support; as distinct from embedded systems and IFEs, the expert systems in B) and D) are normally also denoted as "cooperating systems"

The spectrum of coupled numeric/symbolic systems can globally be subdivided in "knowledge-based simulation systems" (AI included in simulation) and "simulation-supported reasoning systems" (simulation included in AI). Fig. 4 shows examples in each of these categories, seen from a functional point of view. Because of the diversity in practical realizations, they may well overlap and shift along the spectrum of coupled numeric/symbolic systems. Knowledge-based simulation systems can be distinguished in (see Fig. 4 and 5):

1. *(Numeric) simulation systems.* See section 2.1.

2. *Simulation systems with embedded expert systems.* For example, a simulation system may obtain knowledge about queue priority from an embedded expert system according to the states of the modelled objects [Castillo et al., 1988].

3. *Simulation systems and advisory expert systems.* An advisory system or advice giving system is an expert system that gives coherent useful advice on an particular topic following a short consultation. Simulation system and expert system are both accesible to the user; they may well cooperate and share some data (see Fig. 5B). Advisory systems for simulation are developed for application in the various stages of the modelling and simulation process as shown in Fig. 2. Most of the first available expert systems are applicable for only one isolated problem area, such as the selection of a model-adapted and computer-adapted simulation language [Elmaghraby and Jagannathan, 1985] or analysis of results for presentation to the user. Also conceptual modelling is a particular area of the simulation process where users may benefit from advice. An example of an expert system in this vein is given in [Doukidis and Paul, 1985], which allows natural-language input of model definitions and prompts the user to extend these definitions, pointing out inaccuracies and inconsistences. More recent research

projects typically address more than on of these phases in the simulation process.

4. *Simulation systems with intelligent front-ends (IFEs).* An intelligent front-end is a user-friendly interface to a simulation software-package that would otherwise be technically incomprehensible and/or too complex to be accessible to many potential users. Front-ends are directly accessible to the user, whereas the access to the simulation system is exclusively through these front-ends (see Fig. 5C). This class of expert systems is used to bridge over the gap between problem domains and a certain modelling tool. The meaning and description of items of a specific application domain are internally mapped into terms used and needed by the particular modelling tool. Examples of IFEs are described in [Fjellheim, 1986] and [Muetzelfeldt et al., 1986].

5. *Simulation-based decision support systems.* Frequently, simulation studies are aimed to provide predictive information which ultimately should contribute to decisions. Expert systems can select (especially in data overload situations) pertinent data and help in the interpretation of trends, provide perspectives from similar cases in its database, and help define further questions and new simulation experiments addressing unresolved issues [Mellichamp and Wahab, 1987]. Similar to the advisory systems mentioned above, decision support systems and their related simulation systems are separately accessible to the user (see Fig. 5D). Decision supporting expert systems take obviously more part in the proper problem-solving process than the expert systems included in the knowledge-based simulation systems considered in the foregoing points 2—4.

6. *Simulation systems with multiple decision-making agents.* Here, the real-world system is modelled partly by a traditional simulation model (e.g., physical processes) and partly by an expert system to model the decision making processes.

7. *New integrated systems.* In these systems the simulation paradigm is changed essentially using knowledge-based technology; we speak of new simulation development environments. In fact, the simulation system and expert system are fully integrated. Well-known examples are Knowledge-Based Simulation System (KBS) developed at Carnegie-Melon [Fox et al., 1989], and Rule-Oriented Simulation System (ROSS) developed by the Rand Cooperation.

The systems considered in point 3, 5 and 6 are sometimes termed as "cooperating systems" [Merkuryeva et al., 1990]. Programming paradigms for knowledge-based simulation systems include:

— *Object-oriented programming (eventually with embedded rules).* Examples are: LASER/SIM (Intelli-Sys. Corp.), KBS (Carnegie-Mellon), ROSS (Rand Corp.).
— *Rule-based programming.* Examples are given in [Merkuryeva et al., 1990].
— *Logic programming.* The knowledge-based simulation systems may be based on different derivations from the Prolog family such as T-CP, T-Prolog and TS-Prolog, adapted particularly for simulation. Examples are given in [Merkuryeva et al., 1990].
— *Multiple programming.* Different programming styles are used in one simulation environment. For example, available hybrid expert system tools, integrating several programming styles, are used to build knowledge-based simulation systems (e.g., KBS using SRL [Fox et al., 1989], SIMKIT using KEE [Nielsen, 1987]).

Simulation-supported reasoning systems may be functionally distinguished in (see Fig. 4):

1. *Knowledge-based control systems in simulation environment.* Using knowledge-based expert systems in (real-world) supervisory and also direct process control is becoming an increasingly important application domain. Simulation of the process to control would replace the often complex, expensive and fault-prone input and output interfaces and related programs between the controller and the process. It can also serve as a testbed within which precise, well-designed experiments might be run in order to check the knowledge-based controller with respect to its perfect working. Hence, in the simulation domain we are confronted with a knowledge-based expert system controlling a simulation system. The knowledge base may contain process knowledge (such as the order of the process, the parameters and their values and variations, non-linearities), technical control knowledge (e.g., rules to control damping and overshoot) and heuristic knowledge based on past experiences (for example, with respect to similar processes).

2. *Simulation systems as a testbed for expert systems.* Simulation may be used to verify expert system reasoning as a refinement of the expert knowledge base. For example, in a design problem the expert system would select reasonable bounds for each of the parameters concerned; simulation with optimization routines could then be used to select the best parameter values. Simulation can also be used as a double-check on the expert program in tasks such as fault identification or planning, in which the system concludes that a given problem exists or a given sequence of actions will lead to the desired goal.

3. *Expert systems with embedded simulation systems.* During its operation the expert systems may use data which are produced by an embedded simulation system. For example, the expert system may need a simulation to obtain some results for the user [O'Keefe, 1986]. An expert system may use one or more time-dependent variables and a simulation is needed for updating their values; in [Russel, 1989], applications are

considered where the expert system needs to know the position of a military aircraft.

4. *Qualitative simulation systems.* Qualitative simulation, also termed "envisionment", is the most common application of qualitative reasoning; it is used to describe changes in quantities and their propagation [Forbus, 1984]. These changes are characterized by qualitative values, such as "increasing" and "decreasing". Along with describing changes in quantities, the qualitative simulation identifies possible changes in the state of behavior of the device. For instance, if water is being heated, at some time it may begin to boil, representing a change of state from liquid to gaseous. While there are many advantages to qualitative simulation, it does have its limitations. A primary drawback is the frequency of ambiguities in determining how quantities change. To resolve these, several methods have been explored such as combining qualitative reasoning with quantitative techniques to choose among the different possibilities generated by the envisionment.

5. *Knowledge-based expert systems: 2nd-generation (deep-model based) and 1st-generation systems.* See section 2.2.

For more details about coupled numeric/symbolic systems including knowledge-based simulation systems and simulation-supported reasoning systems, the reader is referred to the literature [Fishwick and Modjeski, 1991], [Widman et al., 1989], [Merkuryeva et al., 1990], [Kowalik and Kitzmiller, 1988].

2.4 Artificial Neural Networks

An artificial neural network (ANN) is an adaptive information-processing system that develops its own algorithms (in response to its environment) to solve the problems concerned, without especially having been programmed for that. A neural network is adaptive or self-organizing: it learns on the basis of training. These characteristics reveal ANNs truly a new computing paradigm.

An ANN consists of a number of identical, fairly simple processing elements called neurons, that are densely interconnected. The particular fashion in which the neurons are connected is called a network paradigm, of which there are currently some 15 in common use. Just like a human neuron, each artificial neuron can have any number of inputs but only one output, which may branch out to become the input for many other neurons (see Fig. 6). Some neurons in an ANN receive their input from the outside world (input neurons). The signals handled by an ANN may be analog or digital. In either case, during processing a neuron performs a weighted sum of its inputs, and if that sum exceeds a given threshold, the neuron outputs a signal: it "fires" or is "stimulated". (Often, the neuron's "activation function" is not a hard limiter, but a continuous function such as the Sigmoid-function considered below). During "learning" the weights, which affect the relative strength of each input of any neuron, are modified according to both a "learning rule" and the data being presented to the ANN. Information or knowledge in an ANN is represented by the complex patterns of neuron stimulations and the adjusted weights associated with each interconnection. The knowledge associated with an ANN is therefore distributed throughout the network, and not located in any one location as it is in other computing systems.

The diffuse, highly parallel structure of information in an ANN offers interesting advantages. First, it may make an ANN inherently more resistant to damage than a traditional computing system. Second, it may eventually allow ANNs to be implemented successfully in very large VLSI chips, since they could tolerate many more defects. Third, in a parallel processing environment it would enable an ANN to solve many problems very quickly, even though its individual processors are quite slow. Despite their superficial resemblance, ANNs exhibit a surprising number of the brain's characteristics. For example, they can *learn* from experience (i.e. modify their behavior in response to their environment), *generalize* from previous examples to new ones (note the example in section 1) and *abstract* essential characteristcs from inputs containing irrelevant data.

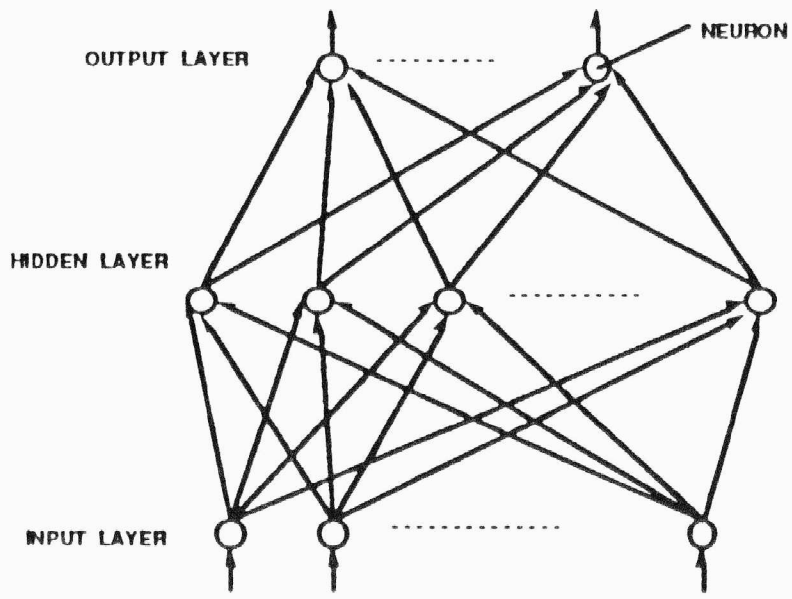


Figure 6: An example of an ANN

ANNs are currently implemented with several different training algorithms. The most popular network is the multilayer backpropagation neural network (BNN) which has a structure similar to the one in Fig. 6, however in general with multiple hidden layers; the training is supervised (i.e. on the basis of the differences between actual and desired outputs), and during the training phase the errors that appear at the output neurons are "backpropagated" through the net [Rumelhart and McClelland, 1987]. In the BNN of Fig. 7 the variables $u_i^{[s-1]}$, $w_{ji}^{[s]}$ and $I_j^{[s]}$ respectively represent the current output state of the i-th neuron in layer s-1 to a j-th neuron in layer s and the weighted

summation of inputs to a j-th neuron in layer s. Training the BNN implies the computation of the synaptic weights (w's) such that corresponding desired or target outputs (d's) will appear on the output layer neurons when a set of data is presented to the neurons of the input layer. Once training is completed, correct outputs are available, also if previously unknown data samples are presented (generalization).

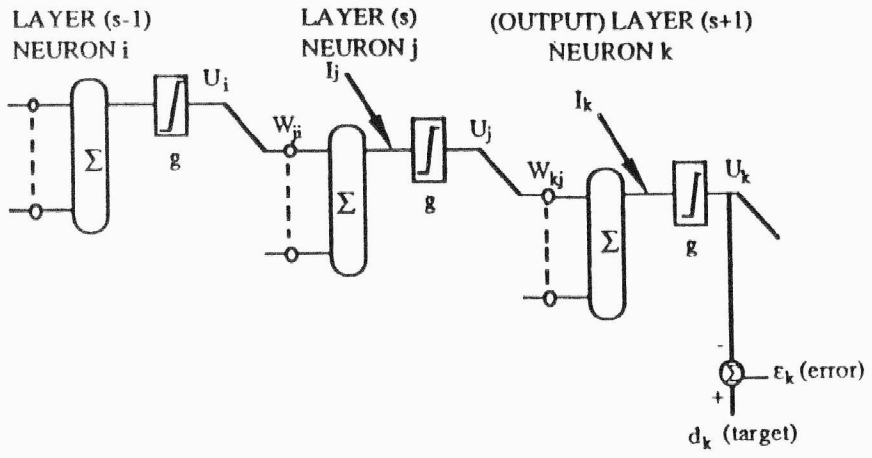


Figure 7: The backpropagation neural network (BNN)

The process in the BNN-neuron j in layer s results in an activation $u_j^{\{s\}}$ which is mathematically characterized

$$u_j^{\{s\}} = g\left(\sum_i u_i^{\{s-1\}} w_{ji}^{\{s\}}\right) = g\left(I_j^{\{s\}}\right) \qquad (1)$$

where

$$g(I) = \frac{1}{1 + e^{-I}}$$

is the so-called Sigmoid-function. In the forward pass these activations are propagated from predecessor layer to successor layer.

During the backward pass (in the training phase) weights are modified according to

$$\Delta w_{ji}^{\{s\}}(t+1) = \eta \delta_j^{\{s\}} u_i^{\{s-1\}} + \alpha \Delta w_{ji}^{\{s\}}(t) \qquad (2)$$

where η and α respectively denote the "learning rate" and "momentum", and t characterizes the backward cycle concerned. The so-called error signal δ in Eq. (2) is defined for neuron k in the (output) layer s + 1 as (see also Fig. 7):

$$\delta_k^{\{s+1\}} = \left(d_k - u_k^{\{s+1\}}\right) g'\left(I_k^{\{s+1\}}\right);\ d_k - u_k = \varepsilon_k\ , \qquad (3)$$

where

$$g'(I) = \frac{dg}{dI} = g(1-g).$$

For neuron j in a hidden layer s this error signal δ is given by:

$$\delta_j^{\{s\}} = g'\left(I_j^{\{s\}}\right) \sum_k \delta_k^{\{s+1\}} w_{kj}^{\{s+1\}} \qquad (4)$$

For more details on BNNs the reader is referred to the literature on neural networks [Soucek, 1989], [Wasserman, 1989], [Dayhoff, 1990].

ANNs can be simulated on traditional sequential computers. There exist many commercially available neural network development systems, e.g. NeuralWorks from NeuralWare Inc., Nestor Development System (NDS), Cognitron from Cognitive Software Inc., ANSim from Science Application International Corp., DESIRE/NEUNET [Korn, 1989], Genesis from Neural Systems Inc., and many others. It is evident that there is an increasing interest to implement ANNs on general-purpose parallel computers in order to try reducing especially the often long training times; an example is presented in section 4.4.

2.5 Coupled systems including ANNs

Recently, we have seen a beginning appearance of articles and papers dealling with the combined usage of expert systems and ANNs as well as simulation systems and ANNs, and even all three approaches combined in one application. It would be useful to arrange those coupled systems functionally along the two edges concerned of the triangle in Fig. 1, and in the interior of this triangle as well, just as we did for coupled numeric/symbolic systems (see section 2.3 and Fig. 4). It is the author's current study and research to do so in the near future. Therefore, rather than presenting a functionally systematic overview on coupled symbolic/connectionist and numeric/connectionist systems, in this section we shall restrict ourselves to some preliminary notes and comments. One remark in advance is, that up to now coupled symbolic/connectionist and numeric/connectionist systems did not mature to such wide-spread applications as we currently see with coupled numeric/symbolic systems, and even for the latter we are just in the very beginning [Kowalik and Kitzmiller, 1988], [Widman et al., 1989].

a) Combining neural and symbolic processing

Historically, AI has been separated into symbolic and nonsymbolic approaches to simulate intelligence. For a considerable time the symbolic approach has dominated, but the interest in parallel (distributed) processing in recent years has given the nonsymbolic approach new momentum. The symbolic approach (or more specifically knowledge-based systems and expert systems), however, have yielded systems with capabilities different from those of nonsymbolic systems (such as ANNs or connectionist systems). ANNs have shown to be adept in tasks such as image processing and pattern recognition, whereas expert systems have had more success in problem solving and game playing. To provide systems that demonstrate both

characteristics (referred to as "low- and high-level cognitive processing" [Hendler, 1989] or "behavior-based and knowledge-based problem solving" [Steels, 1989], several approaches are possible:

— Designing a methodology for getting expert systems to handle image processing, pattern recognition and other perceptual processes
— Designing a methodology for getting ANNs to handle high-level symbol-processing tasks in applied domains, involving for example manipulation of data structures and handling variable bindings
— Linking the current connectionist systems and the current symbolic systems, and produce coupled systems exploiting the strengths of both.

Currently, the focus is mainly on the last-mentioned possibility. The most important reason for combining neural and symbolic processing are their complementary characteristics. Most of the limitations of expert systems, such as

— unadaptiveness,
— no generalization capabilities,
— difficulty with complex pattern matching,
— the knowledge-acquisition bottleneck, and
— relatively slow performance in normal operating environments

may be overcome by the benefits of ANNs. Vice versa, the limitations of ANNs, such as

— non-transparency
— lack of explanation facilities, and
— extensive initial training needs

may be overcome by the benefits of expert systems.

The three models for coupled (or hybrid) systems that we described in the heading of section 2, namely loosely coupled, tightly coupled, and fully integrated, do obviously apply to coupled connectionist/symbolic systems [Serra, 1989],[Amy et al., 1990]. Loose coupling may include coprocessing as well as pre- and postprocessing by ANNs functioning as a user-interface. As shown in Fig. 9, tightly coupled systems may embody blackboard systems, cooperating systems and embedded systems; an example of the latter can be found in [Hendler, 1989], where a local connectionist-like network is embedded into a symbolic network. Fully integrated systems are sometimes referred to as *connectionist expert systems* [Gallant, 1988]. In the spectrum of coupled connectionist/symbolic systems, these connectionist expert systems have the same location (namely, in the middle) as the new integrated knowledge-based simulation systems have in the spectrum of numeric/symbolic systems (see Fig. 4).

For practical applications of coupled connectionist/symbolic systems the reader is referred to the literature, a.o. [Bigus and Goolsbey, 1990], [Schreinemakers and Touretzky, 1990], [Rabelo and Alptekin, 1989], and [Vercauteren et al., 1989].

b) Combining neural and numeric processing

In principle, again the above three models of coupled systems do apply. Loose coupling may include

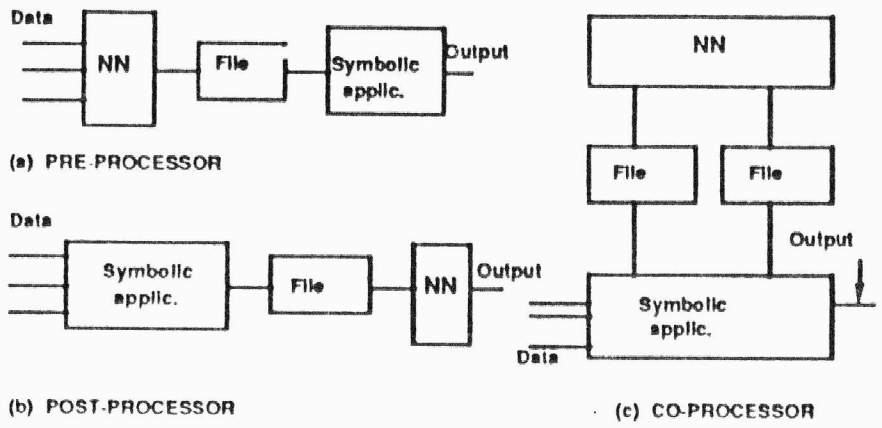


Figure 8: Loosely-coupled connectionist/symbolic systems (NN: neural net)

preprocessing by ANNs of data needed as input for numeric simulation systems (an example of this is given in section 5.2) or preprocessing by numeric systems of input data for ANNs (see the example discussed in section 5.3). Tightly coupled connectionist/numeric systems may embody cooperating systems and ANNs embedded in numeric systems. Systems in which neural and numeric processing are fully integrated may be possible, since analytical functions can be learned and generalized by ANNs (note: in contrast with "connectionist expert systems", the author never met the term "connectionist (numeric) simulation system").

Since ANNs do well in pattern-recognition tasks and simulation modelling can often be formulated as a pattern-recognition problem, ANNs could at least in principle be employed in the modelling process. As such, the use of them is comparable with the employment of expert advisory systems in numeric simulation (see section 2.3). Let us consider an example concerned with the selection of an appropriate numeric model for a given input-output data stream of a so-called ill-defined system (input-output measurements subject to noise and other inaccuracies). Each candidate model (on the basis of a priori knowledge) can be characterized by a point in an n-dimensional feature-space. If all the features are independent, it is possible

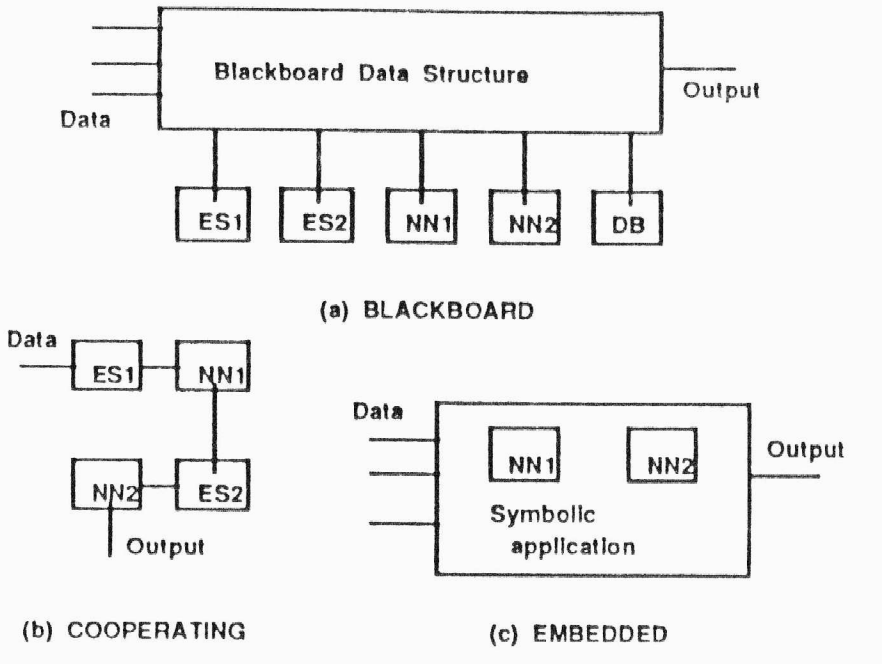


Figure 9: Tightly-coupled connectionist/symbolic systems (NN: neural net, ES: expert system)

to select a model on the basis of feature selection. Then, by investigating the variability of each feature, one can decide which model best corresponds with the real-world system. ANNs are proposed for classification of the least variable feature [Vermeersch et al., 1990].

There is an increasing use of ANNs in nonlinear process control. A reason may be that the control tasks are extremely knowledge-intensive and there is a need to integrate various decision-making modules into one effecively functioning mechanism. The ANN's training process reflects that complexity, superimposing different pieces of knowledge onto the distributed memory of the ANN [Pao and Sobajic, 1990]. Moreover, the benefit of adaptiveness of ANNs may be very useful in process control. The development of a neural-net control system strategy normally proceeds in two steps: first, we deal with a simulated environment (see Fig. 10) and second, the simulated system is replaced by the real-world system. So, in the simulation domain we have to cope with a coupled connectionist/numeric system: neural-net control system. In the spectrum of coupled connectionist/numeric systems, neural-net control systems are functionally similar to the knowledge-based control systems in the spectrum of coupled numeric/symbolic systems (see section 2.3).

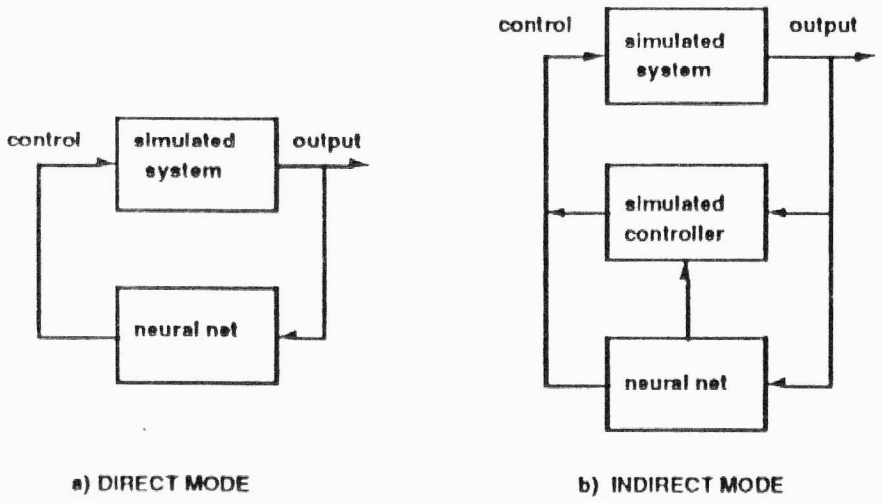


Figure 10: Direct and indirect mode neural-net control in the simulation domain

c) Integrating numeric, symbolic and neural computing

In [Ballard, 1990], an ongoing research effort is described whose goal is to develop a single, unified computational paradigm for conjoint computing which integrates concepts from symbolic processing, numeric processing, and neural-network technologies. The result should be a novel methodology for synthesizing intelligent systems. By combining these technologies, it may be possible to build systems that really behave intelligently, i.e. operate in real time, exhibit adaptive, goal-oriented problem-solving skills, tolerate errors, exploit large amounts of knowledge, use symbols and abstractions, and learn from the environment.

3. Problem-Solving Capabilities

The term"problem solving" is frequently met in the simulation and AI literature. In simulation, problem solving is based on model formalism and model use to tackle e.g. "what-if" and optimization questions, whereas in AI problem-solving may include methods such as generate-and-test, heuristic search, inferences, and many others (in this paper, we concentrate on inference techniques). In the ANN-literature we also meet the term "problem solving", especially as far as connectionist reasoning and connectionist expert systems are concerned (see section 2.5a). For interesting general considerations on problem solving the reader is referred to [Newell and Simon, 1972].

In this paper we are using a fairly broad definition of (computer-based) problem solving. Problem solving is simply the process of finding relevant answers to questions about real-world situations by employing problem-domain related knowledge formerly stored in the computer. This, irrespective of the particular knowledge representation (e.g., implicit in a differential model, explicit in a rule- or frame-base, diffuse and distributed in a neural net), and irrespective of the particular knowledge acquisition (e.g., a priori knowledge and measurements for deductive and inductive modelling, extraction from experts, training ANN as a form of problem solving, namely the answering of the question: "to which class does this pattern belong".

a) Numeric simulation

In numeric simulation the quality of the problem solving depends on the validity of the model used. Let us restrict ourselves to mathematical models. Sources for mathematical modelling are: a priori knowledge, measurements (a posteriori knowledge) and goals. For "well-defined systems", the modelling methodology encompasses deductive analysis with additional parameter estimation and validation. For so-called "ill-defined systems', appropriate techniques are necessary to combine the (often small) a priori knowledge with data information and goal considerations. Here, advanced methods are needed for frame definition, inductive structure-characterization, inductive parameter-estimation, experimental design and goal incorporation, and in addition, more than normal attention must be paid to model validation [Spriet and Vansteenkiste, 1982].

In Table 1, systems and goals for problem solving by numeric simulation are listed. The ordering is based on the well-known "Arc of Karplus" [Karplus, 1976]. From top to bottom, systems are gradually changing from "hard" to "soft" disciplines, from "white" or well-defined to "black" or ill-defined, and from mathematical models of high validity to those of low validity. In fact, the table reveals the concept of "ill-definition": a property of systems that makes their mathematical models be less valid so that their pro-

blem-solving capability by simulation is reduced. For well-defined systems we have high-quality a priori knowledge, i.e. general mathematical laws and principles of broad generality and large validity, allowing a deductive way of modelling. For ill-defined systems, a priori knowledge is of low quality or missing at all, forcing us to inductive modelling techniques on the basis of experimental fits and data interpretations of often narrow generality and small validity.

As can be seen from Table 1, for physical and engineering systems the problem-solving goals can be set on a higher level than for systems from other disciplines. For ill-defined systems the goals have to be more modest so that a proper (goal-dependent) model can be assembled.

SYSTEMS	GOALS
Mechanical Systems	Design
Electrical Systems	Prediction
Aero-hydrodynamical Systems	Control
Heat Systems	
Chemical Systems	Test of strategies
Hydrological Systems	
Biochemical Systems	Test of hypotheses
Microbial Systems	
Physiological Systems	Increase insight
Ecological Systems	
Economic Systems	Help thinking
Psychologic Systems	Analyse data
Sociologic Systems	
Political Systems	Arouse public opinion

Table 1: Systems and goals in numeric simulation

For satisfactory handling of ill-defined systems, advanced information processing environments are generally felt to be necessary. Such environments would comprise number crunching, advanced on-line control, database management, and symbolic and neural processing in addition to purely numeric.

Just like the other approaches to problem solving, numeric simulation has its benefits and limitations. While quantitative solutions to problems can be very accurate and provide a consistent set of values for the parameters involved, traditional numeric simulation systems are difficult to build and modify (since they require extensive programming effort), expensive in their use of resources, generally inflexible and not extendable in expressing model structures, insufficient in handling incompleteness and impreciseness, lacking of automatic examination of consistency, and are not able to distinguish causes from effects; moreover, explanation facilities are missing and there are no possibilities of heuristic solutions. These drawbacks can be eliminated by combining numeric processing with symbolic and/or neural processing in coupled (or hybrid) systems (see sections 2.3 and 2.5).

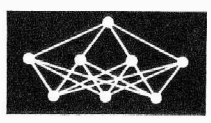
b) Knowledge-based reasoning

In contrast with the procedural algorithms for problem solving in numeric simulation, in symbolic computing they are more declarative: the program specifies how to find the sequence of steps needed to solve the given problem. For example, a numeric (procedural) program for selecting and optimal drug prescription might calculate the quantity of drug and the interval between doses using a formula adjusted to the patient's age, weight, liver and kidney function, and severity of illness. In contrast, an expert-system (declarative) program for selecting the best of several antibiotics to administer when the exact infecting organism is not known, might assemble a list of possible organisms using a knowledge base of frequent and/or serious organisms in the involved part of the body, the particular part of the hospital, and the specific underlying condition of the patient.

PROBLEM DOMAIN	PROBLEM DESCRIPTION
CONTROL	Performing real-world interventions to achieve desired goals
DESIGN	The making of specifications to create objects satisfying particular requirements
INSTRUCTION	Teaching concepts and information to non-experts
INTERPRETATION	Analysis of data to determine their meaning
REPAIR	Prescription of real-world interventions to resolve problems
PREDICTION	Forecasting the future from a model of the past and present
PLANNING	Creating programs of actions that can be carried out to achieve goals
MONITORING	Continuous interpretation of signals and the setting of alarms when intervention is required
DEBUGGING DIAGNOSIS	Finding faults in a system based on interpretation of potentially noisy and incomplete data

Table 2: Some types of problems to which expert systems have been applied

Table 2 lists some problems to which knowledge-based expert systems have been applied. Some of the



criteria to decide whether a particular area of knowledge is suitable for problem solving by expert-system reasoning are [Widman and Loparo, 1989], [Walters and Nielsen, 1988]:

- The knowledge required is well circumscribed
- There are acknowledged experts in the field
- Experts can find high-quality solutions to a typical problem in a reasonable time
- Nonexperts require much more time to achieve solutions of generally lower quality
- A timely solution to the problem is worthwhile
- The knowledge base is stable: once the knowledge is extracted, it can be used with only little modifications for a substantial period of time.

Also the employment of knowledge-based expert systems for problem solving has its benefits and limitations. While expert systems can reason under uncertainty, provide explanations about their solutions, and are generally explicit and transparent in the capturing of knowledge, they suffer from the well-known knowledge-acquistion bottleneck [Bounds, 1989], are inherently very domain-specific (non-graceful degradation of results), are difficult to ensure consistency in the knowledge base, and are generally nonadaptive; moreover, they are not specifically good in image processing, pattern recognition and other perceptual processes, and they are difficult to parallelize due to their inherently sequential nature (see section 4.3). To overcome these drawbacks, dependent on the problem area concerned one might prefer another problem-solving approach (neural net or coupled system approach).

c) Neural computing

ANNs are not a panacea. They are not specificly suited to such tasks as, for example, calculating the payroll. It appears that they will, however, become the preferred technique for a large class of pattern-recognition, classification and image-processing tasks that conventional techniques do poorly, if at all. There have been many impressive demonstrations of ANN capabilities: a network has been trained to convert text to phonetic representations, which were then converted to speech by other means [Sejnowsky and Rosenberg, 1987]; another network can recognize handwritten characters [Burr, 1987], and a neural-network based image compression system has been devised [Cottrell et al., 1987]. These all used the backpropagation network (see section 2.4).

In Table 3, some ANN applications currently under development and investigation are listed. It shows that the spectrum of practical ANN applications covers much more than the ones mentioned-above. New, still unexpected application possibilities might be discovered in the (near) future. With a view to the above broad definition of problem solving, we may conclude that very many of the today's ANN applications show problem-solving aspects, especially those concerned with optimization, adaptive control, prediction, scheduling, and the like.

•• SALES PREDICTION
•• AIRLINE SEAT SCHEDULING
•• STOCK FORECASTING
•• MEDICAL EXPERT SYSTEMS
•• PHONETIC TYPEWRITER
•• SENSOR FUSION
•• HEARING AID CHIP
•• PATTERN RECOGNITION
•• TOYS
•• SHARE PORTFOLIO MANAGEMENT
•• INDUSTRIAL DEFECT DETECTION
•• OPTIMIZATION
•• ADAPTIVE CONTROL
•• SIGNATURE RECOGNITION
•• FACE RECOGNITION
•• RADAR ECHO IDENTIFICATION
•• SONAR CLASSIFIER
•• VISUAL TEXTURE ANALYSIS
•• ROBOTS
•• HANDWRITTEN CHARACTER RECOGNITION
•• MULTISENSOR ANALYSER
•• EVALUATING INSURANCE RISKS
•• ACOUSTIC EMISSION ANALYSIS
•• CREDIT WORTHINESS EVALUATION

Table 3: Some ANN applications

Just as numeric and symbolic cumputing, neural computing has its benefits (such as adaptiveness, generalization, robustness, graceful degradation, learning and abstraction capabilities) and drawbacks; in coupled systems (see section 2.3 and 2.5), the first might be strengthened and the latter be reduced. Some major limitations of the ANN approach are that the network cannot be "told" facts, as can conventional expert systems, and that knowledge in the network is not easily available to the user. Unlike a symbolic AI system, the ANN elements cannot "explain" their numeric weighting factors.

d) Hybrid computing

In hybrid systems we are confronted with four possible combinations: numeric/symbolic, numeric/connectionist, symbolic/connectionist, and numeric/symbolic/connectionist. From the problemsolving capabilities of numeric, symbolic and neural computing each (see section 3a, b, c) and the motivations and possibilities to couple them (see sections 2.3 and 2.5), it is easy to conclude the problem-solving capabilities of hybrid systems.

Returning to expert systems and ANNs, some may that ANNs are going to replace expert systems, but there are many indications (see section 2.5a) that the two will coexist and be combined into systems in which each technique performs the tasks for which it is best suited. This viewpoint is supported by the way how humans operate in the world. Activities requiring rapid responses are governed by pattern cognition (without conscious effort). When our pattern-recognition system fails to produce an unambigous interpretation (and when time permits), the matter is referred to higher mental functions. These may require more infor-

mation and certainly more time, but the quality of the resulting decisions can be superior. One can envision a acoupled connectionist/symbolic system as an artificial system that mimics this division of labor [Wasserman, 1989].

4. The Emerging Role of Parallel Processing

The design motivations and application areas of parallel processing systems cover a broad spectrum, and there is increasing overlap. Along with parallel Office Systems (such as parallel UNIX systems, e.g. Sequent SYMMETRY and Encore MULTIMAX, parallel database systems supporting large relational databases, and parallel Communication Systems for simultaneously routing a number of messages, e.g. BNN BUTTERFLY), three notable application areas do reflect exactly the angular points of the triangle in Fig. 1:

- (Mini)supercomputers for numeric computation and simulation
- Artificial Intelligence systems for symbolic computation, also denoted as 5th-generation computers
- Neurocomputers, specifically designed to implement artificial neural networks, also denoted as 6th-generation computers.

Users of digital computers for numeric computation and simulation are strongly motivated to demand high processing speeds for two distinct reasons [Kerckhoffs and Vansteenkiste, 1990]: the increasingly detailed representation required for more and more complex distributed parameter systems, characterized by partial differential equations (PDEs), and the real-time (or faster than real-time) computation of complex and large-scale lumped parameter systems, described by ordinary differential equations (ODEs).

Also the AI community is fully aware of the crucial importance of robust, flexible, extremely fast (hence highly parallel) systems. Such systems are necessary to handle the complexity and stringent real-time demands of the real-world problems we are confronted with. Without computers that are enormously large and highly parallel, true artificial intelligence is impossible, and we are doomed to remaining tackle the rigid toylike problems for which AI systems are too often developed. In order to be able to develop AI systems (such as e.g. intelligent robots with vision) that concurrently recognize, move to, gain control over and interact with moving and changing real-world objects, AI researchers are increasingly turning to parallel processes [Uhr, 1987].

The use of artificial neural (ANNs) to perform such tasks as artificial vision, speech recognition, signal processing, and the like, is promising, but especially the training algorithms (such as backpropagation, see section 2.4) tend to be very time-consuming. It is therefore natural to try to capitalize on the intrinsic parallelism of these systems in order to speed up the computations. Also the increasing demands for larger and larger networks as well as the beginning real-time use of ANNs require their implementation on neurocomputers or on general-purpose (massively) parallel computers [Treleaven, 1989], [Soucek, 1989].

In section 4.1, we examine the above-mentioned three classes of parallel computers in some more detail. In the subsequent sections 4.2—4.5, we deal with implementation aspects of respectively simulations, knowledge-based systems, ANNs and coupled systems on *general-purpose* parallel computers.

4.1 Parallel computing framework

One of the earliest and most common taxonomies for parallel computers is that of Flynn [Flynn, 1972]. He differentiated between computer structures in two dimensions, namely flow of control and flow of data:

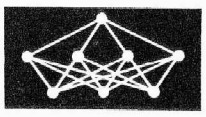
- SISD (single instruction stream, single data stream), which covers all conventional uniprocessor systems (such as IBM 370, DEC VAX, Sun)
- SIMD (single instruction stream, multiple data stream), which includes both vector processors (such as Cray-1) and array processors (such as FPS and the classic ILLIAC IV)
- MIMD (multiple instruction stream, multiple data stream), which covers multiprocessor systems with both shared memory (such as Sequent SYMMETRY) and distributed local memories (such as Intel iPSC and NCube hypercubes).

For the MISD-architecture, no obvious example is forthcoming.

a) Numeric supercomputers and minisupercomputers

Numeric simulation has been particularly influential in the evolution of SIMD and MIMD systems, which resulted in a wide range of supercomputers and minisupercomputers. Although these systems have been specificly developed for complex numeric computation and simulation, they may well be used (and indeed often are being used) for AI purposes or neurocomputing.

The first supercomputer to win wide-spread acceptance was the Cray-1, which was installed in 1976. In this computer the pipelining (vector processing) concept was incorporated in the architecture. The machine had a peak performance of 160 Mflop/s (i.e. 160 million floating point operations per second). This Cray-1 was the trend-setter for many later supercomputers with a similar approach to high-speed computation. Control Data introduced in 1980 the Cyber 205 with a peak performance of 400 Mflop/s. These two (American) machines have dominated the market until the early 1980s, when three Japanese vendors (Hitachi, Fujitsu, and NEC) introduced a new generation of supercomputers with peak performances of 710—11 300 Mflop/s. Along with a better performance, these machines had better vectorizing compilers. Cray and Control Data responded to this challenge in



the mid and late 1980s with new models (Cray X-MP, Cray-2, Cray Y-MP, Eta-10); these MIMD-structured systems were no longer uniprocessors and had considerably improved software. After then, new supercomputers emerged and surely will emerge (Hitac S820, Fujitsu VP2000, Cray-3).

In the above systems the individual processor is a vector-processing unit. Currently, there is an increasing interest to develop massively parallel computers, in which the individual processors are scalar CPUs. These machines are expected to be able to deliver the same or even more computing power than the "traditional" vector-oriented supercomputers. Recently, Cray announced to start developments in the direction of massively parallel computers. In this respect, NCube has starting experiences and expertise for many years. They recently announced the fastest supercomputer of today. Ncube-2 system (model 80) with 8192 processors arranged in a 13-dimensional hypercube structure (peak performance: 27 000 Mflop/s).

In the mid 1980s, some vendors started to fill the gap between supercomputers and mainframes by introducing computers with peak performances of "only" several tens of Mflop/s for prices comparable to those of large minicomputers or small mainframes. These became known as "minisupercomputers". Their architectures contain concepts used in supercomputers (both vector-processing units and multiple scalar-CPUs), but they are built mainly with of-the-shelf hardware resulting in good price/performance ratios. Currently, market leaders are Convex, Alliant and SCS, which produce computers with peak performances in the range 20—200 Mflop/s.

For more details on supercomputers and minisupercomputers the reader is referred to the literature, e.g. [Hockney and Jesshope, 1988].

b) Parallel Artificial Intelligence Systems (5th-generation systems)

The term "5th-generation computer" was introduced in the mid 1980s by the Japanese authorities to announce an extensive research program by Japanese universities and computer manufacturers with respect to a new computer paradigm. This project, with 1990 as its original goal, has had enormous amounts of funding, and of publicity and imitation both in the USA and Europe. The project focusses on developing a gigantic Prolog machine, capable of executing 1 000 Mlips (i.e. one billion logical inferences per second, where each logic inference needs the equivalence of 100 to 1 000 ordinary instructions).

The Japanese initiative has had as a result, that also elsewhere in the world extensive research projects have been started with respect to 5th-generation systems. At present, it is not fully clear what exactly is meant with "5th-generation computers". The term is generally reserved for parallel AI systems, i.e. parallel computers designed to efficiently support symbolic processing. These parallel computers might also be called "high-level language computers", because each is typically optimized to support a specific class of high-level programming languages. The four major approaches are [Treleaven, 1990]:

- *Object-oriented* (e.g. Philips DOOM [Bronnenberg et al., 1987])
- *Functional* (e.g. ICL FLAGSHIP [Watson et al., 1987])
- *Logic* (e.g. Bull DDC [Bergsten et al., 1988])
- *Knowledge-based* (e.g. the rule-based computers NON-VON and DADO [Treleaven et al., 1987], and the cellular array computer Thinking Machine's Connection Machine [Hillis, 1985]).

For more details about 5th-generation systems, the reader is referred dto the literature, e.g. [Uhr, 1987], [Treleaven et al., 1987] and [Treleaven, 1990]; in the latter reference, 5th-generation system developments within the framework of ESPRIT (the well-known **E**uropean **S**tragegic **P**rogramme for **R**esearch and **D**evelopment in **I**nformation **T**echnology) are reported.

c) Neurocomputers (6th-generation systems)

A neurocomputer is an information-processing system specifically designed to implement ANNs [Treleaven, 1989]. Neurocomputers are said to be of the *sixth generation*, since they reflect a really new computing paradigm. Current implementations involve one or more of three basic technologies: electronic, optical and electro-optical, although electronic implementations of neurocomputers are presently dominant (the other two technologies are beyond the scope of this paper). Each of these implementations can be "fully implemented" or virtual. In a fully implemented system, each neuron is an individual processor with a fixed interconnection geometry. A virtual neurocomputer implements most of the neurons sequentially and uses standard memory to maintain their states and interconnection weights. For many current applications, virtual neurocomputers are more poupular. They are slower than hard-wired systems, but have the advantage of being more general and flexible. Many of the current implementations use high-speed digital signal-processing VLSI chips with advanced CMOS memory and virtual network strategies; examples of this class of (virtual) neurocomputers are HNC ANZA and SAIC Sigma. The next level of perfomance can be reached by using multiprocessor techniques, such as in the TRW Mark III (still virtual) neurocomputer. The logical ultimate extension of this last kind of neurocomputers is a fully-implemented configuration, in which each processing element consists of a unique piece of silicon (or gallium arsenide). These systems could be realized with digital or analog circuits. Work on analog neurochips is being performed by AT&T and Synaptics.

4.2 Parallel implementation of numeric simulation systems

The definition of a supercomputer, i.e. top-of-the-line computer, given by N. Lincoln: "a system that is only one generation behind the computing requirements of leading edge efforts in science and engineering" is not only a flexible one, but it also emphasizes the relationship between (mini)supercomputers and computational science, which is essentially simulation-based. Realistic problems may lead to very complex models requiring extensive computational resources.

For many often encountered numerical problems, stable and robust algorithms have been devised for SISD-type computers. It is usually clear which variants of those algorithms will lead to the most efficient implementations. This has stimulated the birth of program libraries: collections of standard portable software for problems of very different nature. Some of these have matured to widely accepted collections of subroutines at a distinguished high level, preventing the user from large programming efforts and testing problems. At present, many scientists have access to vector computers, such as Cray-1, Cray X-MP, Cyber 205, ETA 10-P, Convex, Alliant, and the like. Early experiences indicated that the efficient use of these computers requires a considerable modification of many existing codes and many popular algorithms had to be replaced by more suitable ones. As a consequence, existing libraries are gradually being adapted to other than SISD-architectures; some general software packages that are, or will be made, suitable for a wide variety of vector computers as well include IMSL, NAG and LAPACK.

While the use of vecor processors is quite well understood by now, this is far from being true for parallel computers that embody shared-memory systems with a modest number of vector processors (such as Cray X-MP/4, Cray-2, Convex C-240, Alliant FX/80) and local memory systems with either (a large number of) scalar processors or (a modest number of) vector processors, such as NCube's hypercubes, Intel's hypercubes, ETA-10E/8, and transputer-based systems like MEIKO. At present, a major problem with those systems is that there are many different programming styles and different algorithms, and therefore it seems impossible to construct general efficient transportable software for the various computers concerned.

Rather than performance measures in, for instance, Mflop/s provided by the vendors, realistic performance comparisons and predictions are to be based more reliably upon benchmark problems. A general additional remark is that, based on the special characteristics of the parallel or vector computer and the size of the simulation system (e.g. the number of state equations, the number of finite difference or finite element points, the types of nonlinearities to be included), the user must be enabled to provide various directives to allow the compilers to recognize parallelizable or vectorizable code in order to take optimal advantage of the computers' potentialities.

For a thorough treatment of vector-oriented and parallel numeric computing methods, the reader is referred to the literature, e.g. [Ortega, 1988] and [Schendel, 1984].

In addition to the motivations of using parallel computers in (real-time or faster than real-time) simulations of complex distributed or lumped parameter systems as mentioned in the heading of section 4, there may be other motivations to apply parallel simulation. In methodology-based interactive simulation, the specific architectures of such parallel computers might well be exploited. Focussing the attention on MIMD arrays of scalar processors, examples of this in model implementation and experimentation are [Kerckhoffs and Brok, 1985]:

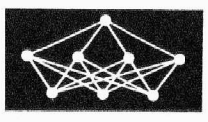
- the exploitation of a one-to-one analogy between a model's structure and its physical implementation on the computer,
- model composition by assembling submodels running on different processing elements (configuring excitable units), and
- interactive experimentation on model bases (for instance, multimodel output analysis after one single run).

4.3 Parallel implementation of knowledge-based (expert) systems

As said before (heading of section 4), the AI community is increasingly turning to parallel processing. Examples are: the use of multicomputers in perception (such as image processing, pattern recognition and computer vision); parallel processing in the structuring and accessing of symbolic, linguistic and perceptual information; the use of multicomputers in speech recognition and language analysis; the use of multicomputers in robot motor control; parallel learning systems. A good survey can be found in [Uhr, 1987], where is examined how well the multicomputer architectures meet the demands from the various subfields of AI. In this section, we restrict ourselves to parallelism in reasoning and problem-solving systems, and more particularly to parallelism in expert systems based on production rules.

In their original pure form, expert production systems use
a) sets of if-then production rules pertaining to conditions to be searched for in:
b) a single common memory.

Pure production systems have no control structures (such as the ordering of rules and procedure calls) that specify how to move between productions. However, most systems are given a number of additional control capabilities. In the following, we examine the various possibilities of parallelism in a rule-based expert system. At the highest level we have to distingu-



ish *knowledge-representation (rule-base)* parallelism and *knowledge-manipulation* parallelism.

Rule-base parallelism can be subdivided in the following distinct types:

1. *Context parallelism.* In context parallelism several disjunct contexts (groups of rules) are distributed among different processors. Since it is plausible that data dependency is high within a context, but low between different contexts, theoretically speed-ups can be obtained. The speed-up attainable is determined by the ratio of internal and inter-context data dependencies. Therefore, context parallelism is domain dependent since this ratio varies among applications. A slightly different form of context parallelism can be obtained if processing redundancy is allowed (overlapping instead of disjunct contexts). The introduction of redundant processing of rules in a rule-based system can reduce the data dependency and might well increase the speed-ups attainable.

2. *Production parallelism.* Production parallelism explits the possibility of distributing all productions (rules) among different processors. Besides the large number of processors needed, the major disadvantage of production parallelism is the small fraction of processors that can run concurrently. This is caused by the implicit sequential coding of production rules.

3. *Action parallelism.* Action parallelism exploits the possibility of executing the actions in the consequence (then-part) of a rule concurrently. Since variable bindings occur in the antecedent (if-part) of the rule and the actions often are independent from each other, true parallelism might be possible. The major disadvantage of action parallelism is the small number of actions normally occurring in the consequence of a rule and the relatively short time needed to execute these actions, even if they are executed sequentially [Gupta, 1987]. Therefore, the possible speed-up resulting from action parallelism is rather limited.

4. *Clause parallelism.* Clause parallelism can be exploited to obtain further speed-ups. In clause parallelism different clauses in a production are executed in parallel. Clause parallelism can be subdivided in two subclasses [Conery, 1987]:

4a. *Or-parallelism.* Consider the following clauses in a production:

p(X) or q(X) or r(X) → ....., where X is an unbound variable.

Three processes can be executed concurrently to evaluate p, q and r respectively. If p binds X to a constant a, q binds X to a constant b and r binds X to a constant c, all three bindings are legitimate and we have found three solutions simultaneously. For example, "p(b) or q(b) or r(b)" is true, since q(b) is true. Therefore, or-parallelism can be exploited to speed-up the reasoning process.

4b. *And-parallelism.* Consider the following clauses in a production:

p(X) and q(X) and r(X) → ....., where X is an unbound variable.

If three processes are executed concurrently to evaluate p, q and r respectively, and p binds X to a constant a, q binds X to a constant b and r binds X to a constant c, we do not have a solution of the antecedent of the rule. For example, "p(b) and q(b) and r(b)" needs further evaluation, since the truth values of p(b) and r(b) are unknown. Therefore, and-parallelism cannot be exploited directly to speed-up the reasoning process. If p(X) is evaluated first, the resulting binding can be used to evaluate q and r concurrently. For example, if p binds X to a, q(a) and r(a) can be evaluated in parallel since they are independent. Note that this scheme does not work for antecedents of the following type:

p(X) and q(X,Y) and r(X,Y) → ....., where X and Y are unbound variables.

On the other hand, if in this case r(X,Y) is evaluated first, resulting in a binding of X to a and Y to b, p(a) and q(a,b) can be evaluated in parallel. This type of and-parallelism is only possible if the ordering in which p, q and r are evaluated is not important. In many production systems, however, the ordering in which the clauses are evaluated is important and, consequently, and-parallelism cannot be exploited.

According to [Gupta, 1987], the average number of clauses in a production (although dependent on the kind of application) is about 2. Hence, the possibility of speeding-up the reasoning process by and-parallelism is rather limited.

Knowledge-manipulation parallelism can be subdivided in two types of conceptual parallelism (inference parallelism and case parallelism) and functional parallelism:

1. *Inference parallelism.* Inference parallelism exploits the stategy to use several different inference (reasoning) techniques to attack a problem (note: this strategy is based on ideas from [Newell and Simon, 1972] regarding "strategy shifts" of human problem solvers) and to execute these different techniques concurrently. As soon as one technique succeeds in solving the problem, the other processes can be halted. In comparison with a sequential approach (where the techniques have to be performed on after each other) relevant speed-ups can be obtained if the number of techniques is relatively large.

2. *Case parallelism.* Case parallelism can be used whenever a variable can take on a limited number of different values. For every possible value of the variable a process can be run in parallel (independent of the others) to infer the validity of a certain conclusion. Proofs by exhaustion are quite common in mathematical problems. Case parallelism can also be used in those situations, where the validity of a certain state-

ment b can be proven by proving "a→b" and "not a→b". These two cases can be evaluated in parallel. For example, induction proofs cosist of two parts. The first part of the proof is based on an assumption such as "i = 0". The second part of the proof is based on the assumption "i< >0" (actually, most inductions are based on "i>0"). Both partial proofs can be performed in parallel. It is allowed to have a part of a proof dependent on another part of the proof, as long as such proo-dependencies are not cyclic.

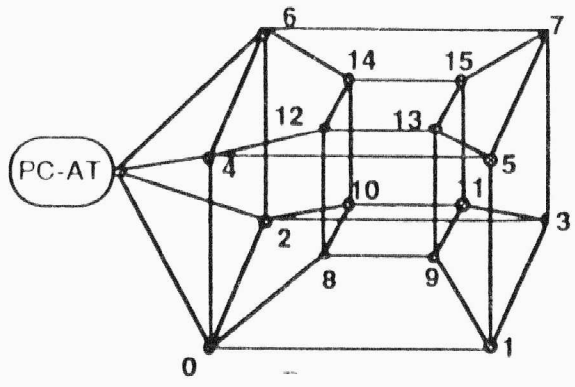
3. *Functional parallelism.* Functional parallelism exploits the distinct concurrent functions within an expert system and the possibility to implement these on different processors. Some functions can be implemented in manifold to increase the speed-up of the system further. For instance, functional parallelism was used in the HYDRA-2 system [Kerckhoffs et al., 1989]. Here, a User Interface, Rule-Agenda Manager, Question-Agenda Manager, Data-Base Manager, and Rule Processor (in manifold) were implemented to run in parallel. Research in the HYDRA-project has shown that functional parallelism suffers too much from inherent sequentialism (contention problems) and that speed-ups are limited to a factor considerably below 10.

It has turned out that parallelizing standard production expert systems normally yields disappointingly little speed-up when using most of the above-mentioned rule-base parallelism techniques or the functional-parallelism approach [Gupta, 1987], [Uhr, 1987], [Kerckhoffs et al., 1989]. Although this is not systematicly examined, to the author's opinion more substantial speed-up might be attained in combining a number of the various approaches, or in expert systems that are more complex than the standard ones (for instance, functional parallelism could do well in expert systems with embedded numeric simulations [Kerckhoffs et al., 1989]), or — dependent on the specific rule base concerned — by the context-parallelism approach with overlapping contexts, or by ultimately exploiting the inference-parallelism and case-parallelism approaches. Nevertheless, a general feeling is that production systems with entirely different formulations and algorithms than the traditional ones, and hence much greater inherent parallelism, should be explored in today's and future research.

4.4 Parallel implementation of ANNs

The regular architecture of ANNs suggests their simulations on general-purpose parallel computers to be implemented with simple placement rules. However, the processors which are available on many parallel computers are much more powerful, and have a much smaller number of input-output ports, than an individual neuron. Hence, the placement problem is not trivial and deserves attention since the communication problem is obviously crucial.

Many ANN-implementations on general-purpose parallel computers have been reported in literature [Petrowski et al., 1989], [Bourrely, 1989], [Wang et al, 1989], [Beynon and Dodd, 1987]. As an illustrative example, in this section we consider briefly our own implementation of backpropagation neural networks (BNNs), see section 2.4, on an NCube/4+ parallel computer [Kerckhoffs et al., 1991]. This NCube/4+ is a 1st-generation hypercube computer, that consists of 16 processing elements with 512 kbyte memory each, arranged in a 4-dimensional hypercube architecture; see Fig. 11.



Node No	Binary	Node No	Binary
0	0000	10	1010
1	0001	11	1011
2	0010	12	1100
3	0011	13	1101
4	0100	14	1110
5	0101	15	1111
6	0110		
7	0111		
8	1000		
9	1001		

Figure 11: The NCube/4+ architecture (neighbouring nodes only differ in one single bit of their binary addresses)

The BNN-implementation, considered here, partitions the units (neurons) of any layer among all the available nodes of the requested hypercube (subcube), so exploiting parallelism per layer. The subsequent layers have to be handled sequentially. This approach allows large networks to be trained and recalled according to the pure unchanged backpropagation algorithm.

For each unit the relevant data (such as incoming weights, bias, activation, backpropagated error) are stored locally on the node that holds the unit concerned. For the purpose of propagating the activations and backpropagating the errors (differences between actual and target outputs), both during the forward and backward pass (in the training phase) data from the neurons in the current layer have to be transferred to all neurons in the next layer to be handled. Consequently, after the parallel computations for a certain layer any node of the hypercube has to transfer data to each other node. This is realized by shifting the information several times along the nodes placed in a ring structure. It is easy to map the hypercube structure on such a ring structure; see Fig. 12. By a technique of shifting partial sums along the ring, while building up the required total sums representing the backpropagated errors, the efficiency of the computation has been optimized [Kerckhoffs et al., 1991].

The speed-up of a parallel BNN-implementation is defined as the "sequential time" divided by the "parallel time" necessary to complete a fixed number of learning cycles. The time, it takes to execute this on one single node, is defined as the "sequential time". Efficiency is defined as the speed-up divided by the needed number of nodes. To get an indication of the

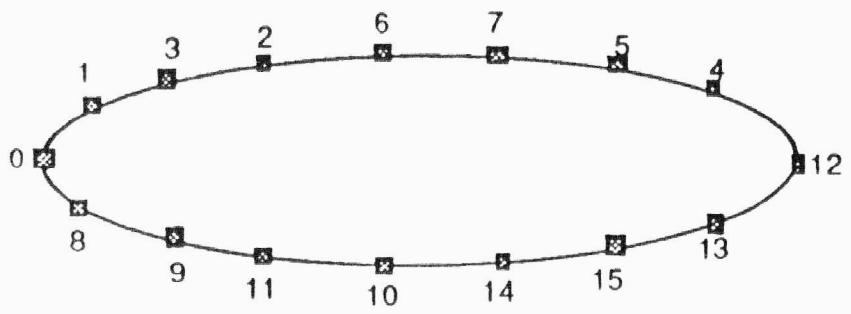


Figure 12: Ring structure (node numbers correspond with those of Fig. 11)

speed-up dependent upon the network size some different net configurations have been executed on the Ncube/4+. The results of varying the sizes of respectively the input, hidden and output layer in a three-layer network with a fixed number of 64 units in the remaining layers are depicted in Fig. 13. It shows that varying the size of the input or output layer has less influence on the results than varying the size of the hidden layer. This can be explained by the fact that the processing of a hidden unit amounts to more computation than that of an input or output unit.

In Table 4, peak performances are given for two-, three- and four-layer networks with equal layers, both in training and recall phases, measured over ten patterns in one sweep. The input and target patterns are stored on the nodes, so a greater number of examples will lead to smaller maximum network sizes and lower peak performance rates. Performance rates are given in KLips, meaning the number of thousands of links that are updated per second in the training phase, or the number of thousands of links that are passed per second in the recall phase.

Net	Training	Recall	Weights
896-896	187	635	803
624-624-624	137	609	779
512-512-512-512	124	594	786

Table 4: Peak performances for the "unit distribution per layer" procedure on an NCube/4+ computer (with 10 training examples): network size, performance in KLips for training and recall phases, and number of weights in Kweights.

4.5 DUTIES: A parallel environment for coupled simulation / expert systems

At present, parallel implementations of coupled systems are rare. In this section, as an illustrative example we describe an environment for parallel coupled numeric/symbolic systems currently being developed at Delft University of Technology: DUTIES (**D**elft **U**niversity of **T**echnology **I**ntelligent **E**nvironment for **S**imulation). It is meant to support running concurrently simulations and coupled expert systems on a distributed and/or parallel hardware platform. Both the simulation and expert systems themselves can be parallelized too (see sections 4.2 and 4.3). Starting points such as accessibility, flexibility, openness and maintainability have led to a design philosophy based on:
— Automatic code generation and program mapping
— Abstraction of concepts
— Modularity of software
— Machine-independence.

a) Global system set-up: code generation and program mapping

The DUTIES environment provides programming tools to implement simulation systems, expert systems and combinations of both on distributed hardware environments and/or MIMD-structured parallel computers. Roughly speaking, the programming tools can be subdivided in three groups:
— Code generator for (numeric) simulation systems
— Code generator for (symbolic) expert systems
— Program mapper.

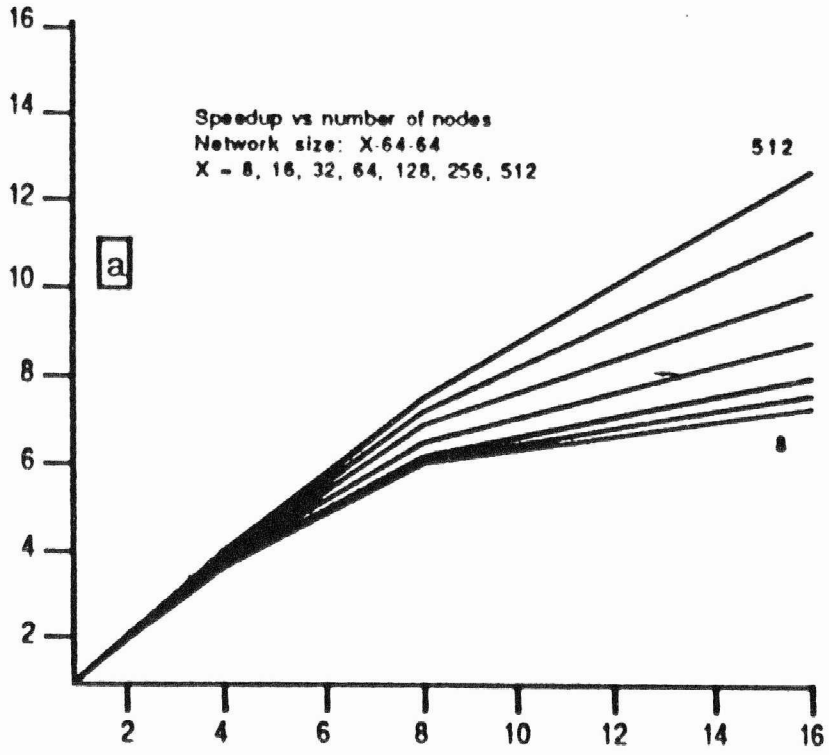
Code generator for numeric systems:

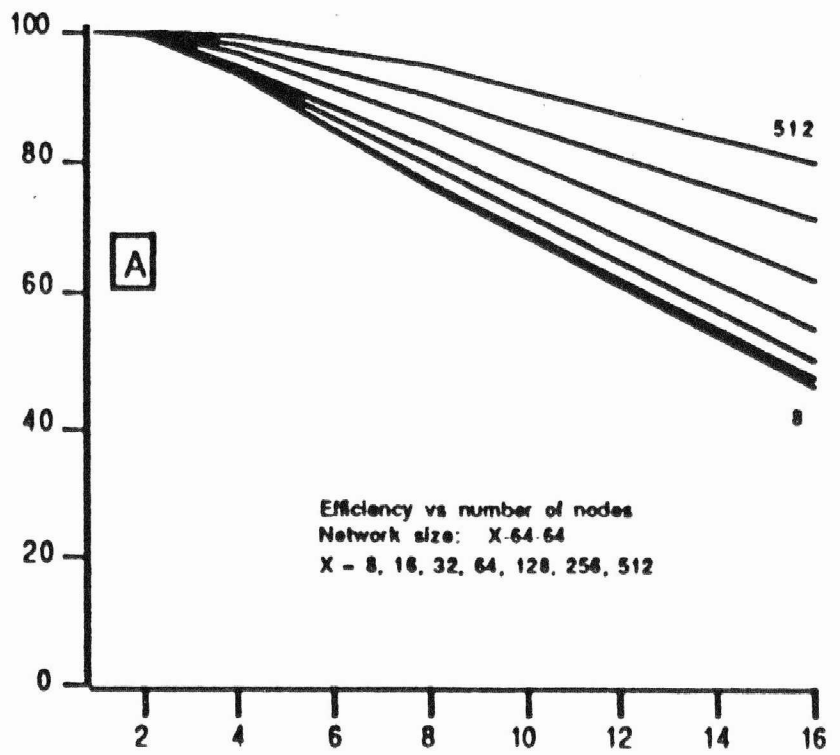
For the time being, DUTIES only supports the use of (first-order) ordinary differential equations (ODEs) to model the numeric system. The ODEs must be formulated as a set of systems of ODEs. The code generator implements every system of differential equations as a separate subprogram to be placed on a separate processor. The differential equations are solved by means of a selected numerical integration method. Furthermore, the code generator generates data (to be processed by the program mapper) regarding the information needed (input) and generated (output) by each of the subprograms. The final generated programs (C-source code) include the communication procedures generated by the program mapper. They are linked to the above-mentioned subprograms. Finally, these subprograms are linked to a special library containing specific machine-dependent procedures.

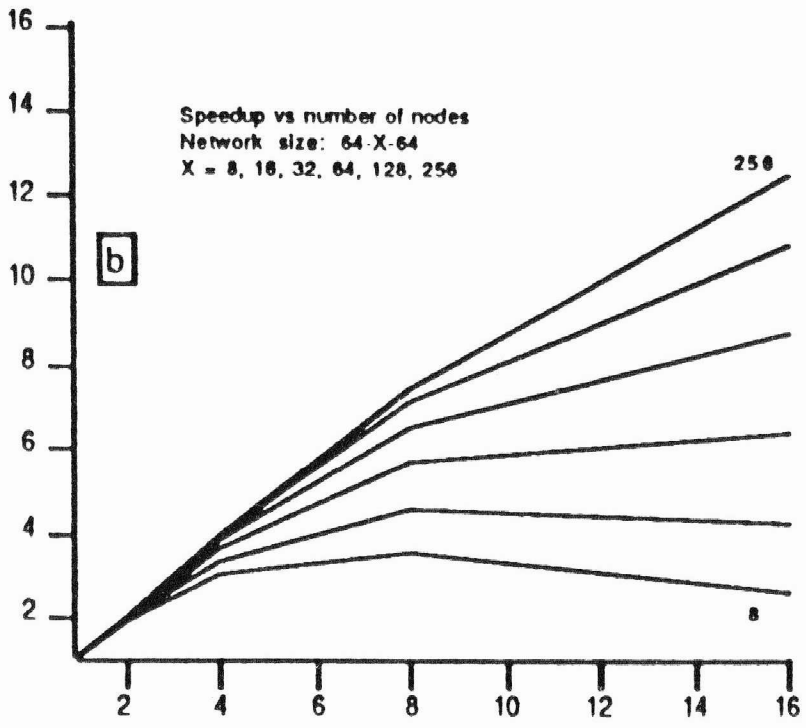
In principle, the final generated programs can be ported to any distributed environment and/or multiprocessor machine, provided that a C-compiler and linker are available. The special library, containing the machine-dependent procedures, should be ported separately. If the processor-interconnection structure differs from the original one, the program mapper must be re-invoked.

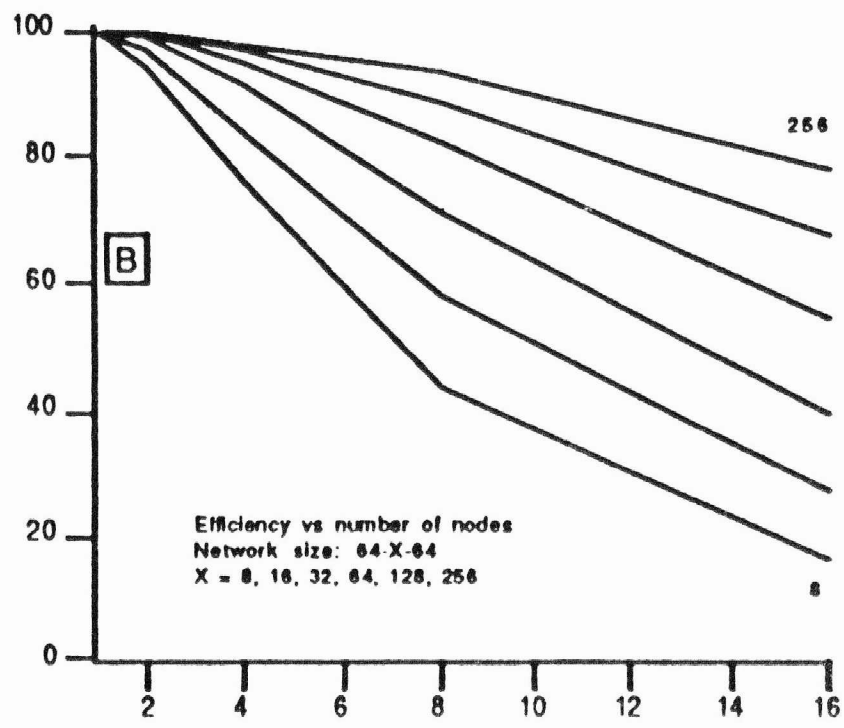
Code generator for expert systems:

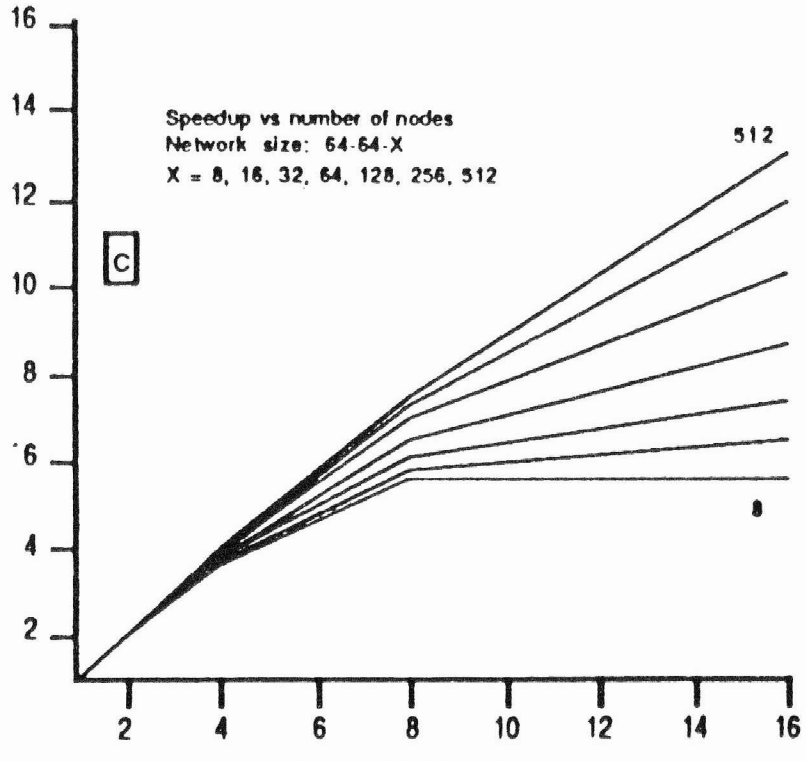
The structure of the code generator for (symbolic) expert systems is more elaborate. Basically, a rule base is translated into an intermediate code which is interpreted by an expert system shell (inference engine). Different rule bases are translated into different parts of intermediate code, interpreted by different (concurrently running) expert system shells. The rule base compiler and the expert system shell are constructed from several modules. These modules support the use











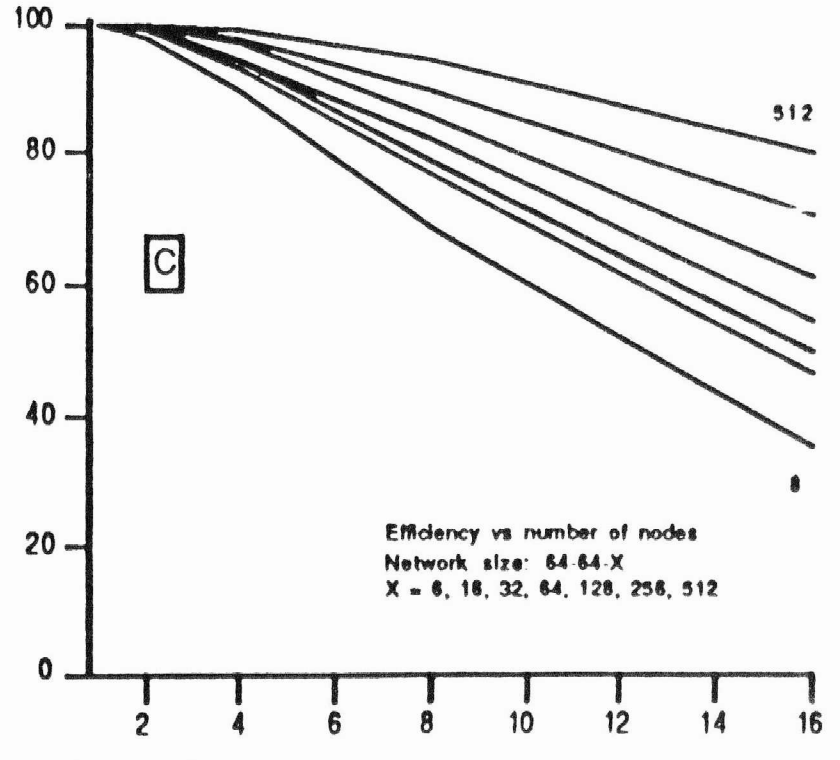


Figure 13: Speed-up (a, b, c) and efficiency (A, B, C) of three-layer back propagation neural networks

of specific data structures, functions or inference mechanisms.
In addition to creating intermediate code, the rule-base compiler also generates data to be processed by the program mapper to map efficiently the different expert systems on different processors (taking into account the communications between the symbolic subprograms mutually and between the numeric and symbolic subprograms).

Program mapper:

The program mapper maps the generated (numeric and symbolic) subprograms on the available processors and generates the communication procedures to be used by these programs. In order to be able to perform these tasks, the program mapper needs information about the intercommunication between the various subprograms. This information is generated by the code generators for the symbolic and numeric systems. For mapping the programs on the processors the program mapper uses several heuristics to achieve a reasonably optimal distribution. It tries to minimize the communication overhead by placing the programs as efficiently as possible. After the programs have been mapped on the various processors, the implemented system routes the data through the interconnection structure. Since, in general, there are several paths between two processors in a network, the data can be routed along various different routes. Several switches are introduced to enable the user to influence the mapping and routing.

b) Abstraction of concepts

The meaning of abstraction is to generalize, i.e. to extract the essential elements while ignoring the irrelevant details. Abstractions play an important role in problem solving by reducing the problem to the fundamental issues which underlie the problem. Software development uses abstractions in order to achieve uniformity and clarity in the code. An important form of abstraction in software development is procedural abstraction, which is available in advanced high-level computer languages, such as Pascal, ADA, etc. In these languages, one can declare procedures to perform a certain task. The programmer is only interested in "what" kind of a task a procedure will perform, not "how" it is done.

Although more concepts have been generalized, the following abstractions play a key role in DUTIES:

- *Data abstraction:* This mechanism allows to access of each data element available in DUTIES, irrespecive of its type, form or location.
- *Communication abstraction:* This mechanism facilitates a uniform way to address any type of communication device, whether it is serial, parallel, a LAN or modem. Implementation details concerning routing, protocol and multiplexing are hidden from the user.
- *Medium abstraction:* A medium is defined as any IO device or memory device, which accepts or produces a stream of bytes. Typical examples of such devices are files, windows, keyboards, cache memory, etc. The medium abstraction is a powerful mechanism allowing to access each medium in a uniform way.
- *Administration abstraction:* A variety of strategies to organize information (such as, sequential, indexed, and sorted) is accessed through the administration mechanism in an identical way.
- *Lazy evaluation abstraction:* This mechanism is used to create a "truth-maintenance facility" for the expert systems in the DUTIES environment.

The main idea behind the generalization of a concept (such as the above "data", "communication", "medium", "administration", "lazy evaluation" concept) is to create a uniform way to deal with that concept, and at the same time hide implementation details. Every concept is sustained by a specific set of functions. "Abstraction" implies creating a new set of generalized (abstract) functions and generalized (abstract) elements. The abstract functions operate on the abstract elements to cover the actual functions with a conceptually equivalent working.

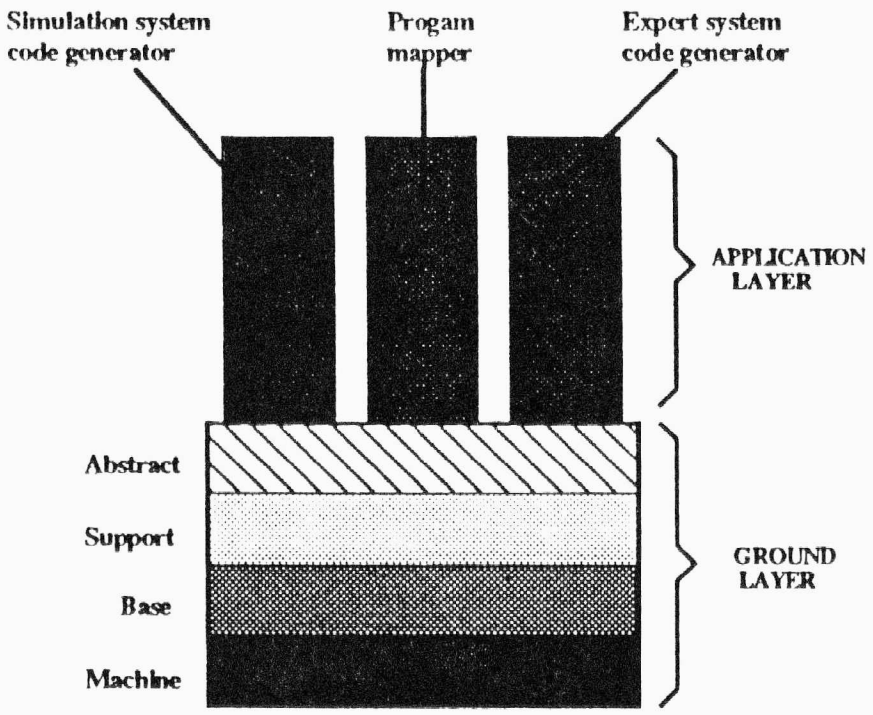


Figure 14: Software modularity in DUTIES

c) Modularity of software

The DUTIES software is based on two main layers (see Fig. 14):

- the *"application layer"*, containing the afore-mentioned code generators for the simulation systems and expert systems as well as the program mapper, and
- the *"ground layer"*, forming the basis of the DUTIES environment on which the applications are built. It is constructed as a set of increasingly complex sublayers:

— the *"machine layer"*, which is the interface to the operating system of a particular computer system. It is this layer that renders DUTIES its machine independence (see section 3.5d). Every time the software of DUTIES needs to access a piece of hardware, the calls are routed through the machine layer, which hides specific implementation details from the operating system. To port DUTIES to another hardware platform, only the machine layer must be rewritten; the other layers only have to be recompiled.
— the *"base layer"*, which contains a number of typeless (i.e. solely byte-oriented) mechanisms, such as memory management routines and primitive IO mechanisms. Some base-layer functions are meant to improve the quality of the machine-layer functions, for instance by adding error handling. Other functions provide utilities that can be used throughout the system, such as "stack mechanism", "pool mechanism" and "semaphore mechanism".
— the *"support layer"*, which imposes structure and limitations on the base layer and introduces the types of data, medium and other concepts on top of this base layer. For instance, on the base level one can print information anywhere on the screen while on the support level printing is only allowed within the borders of a predefined window. Strings, integers and lists with their supporting manipulating functions are examples of data types that are introduced in the support layer.
— the *"abstract layer"*, which utilizes the similarities of the data, medium and other concept types introduced in the support layer to cover each class through a set of abstract functions (see section 3.5b). In this layer several kinds of abstractions are created for the various concepts implemented in the support layer, such as data, medium, etc.

d) Machine independence

Machine indepence, as defined by the designers of DUTIES, exists when the software does hardly relies on the structure or functionality of a particular computer or operating system. Therefore, in principle DUTIES can be ported to any computer system different from the target computer NCube/4+ (see section 4.4) used in this research. Our objective was to minimize the efforts needed to do so.

e) Applicability of DUTIES

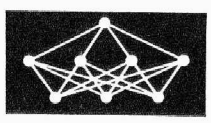
At the moment of writing this paper the actual implementation of the DUTIES environment on the NCube/4+ computer is in its final stage. After completion the system will be used to run practical applications of coupled numeric/symbolic systems. Examples are:
Knowledge-based direct or supervisory control of (simulated) continuous processes [Meijer and Kerckhoffs, 1990]: A control (expert) system is assumed to run concurrently with the simulation and to reason on the basis of (continuously changing) information monitored from the simulation system. Simulation system and expert system run on separate (clusters of) nodes of the hypercube machine and they regularly exchange data.
— *Multiple expert systems acting on a simulated dynamic object:* For example, in a moving robot with vision multiple intelligent processes have to be performed concurrently.
— Parallel treatment of "method bases": For instance, several dynamic parameter optimization algorithms can be run concurrently on separate (clusters of) nodes. An expert system, running in parallel on one or more additional nodes, might be used to select the best method(s) on the basis of intermediate dynamic system results.
— *Parallel treatment of "model bases":* For instance, inductive mathematical modelling leads to nonunique models which might result in more or less extensive model bases. The various mathematical models with related additional equations (such as e.g. parameter sensitivity equations) can be run simultaneously on different (clusters of) nodes, and again an expert system may be used for decision making 'on the fly' (i.e. in parallel with and based on the model simulations).
— *Parallel implementation of progressive reasoning techniques:* In real-time applications expert systems have often to respond within a certain time-frame. Running multiple inference engines in parallel on different rule bases with knowledge on different levels of abstraction allows to select "on the fly" the best advice (on the possibly deepest level) given the available time; the more time available, the more accurate and detailed the expert system's advice. The rules bases must obviously be constructed in such a way that this kind of explicit parallelism can be exploited.

5. Some ANN-Aplication Research Projects

In this section, we describe shortly three current ANN-application research projects of the Group KBS (Knowledge-based Systems) at Delft University of Technology. The chosen example projects may illustrate some of the issues dealt with earlier in this paper.

In the first project, the problem is how to have more grip on the cost related to large software development projects, especially in the beginning phase when uncertainties and incomplete information are to be faced. This essentially budgetting problem is tackled with the use of two backpropagation neural networks (see section 2.4) in cascade.

The second project concerns the problem of finding the optimal (in the sense of minimum cost) air-route among a limited number of (5—10) possible routes connecting two cities. The cost as a function of the de-



lay (i.e. difference between actual and scheduled departure-time) is actually calculated in a numeric simulation model; an ANN-based preprocessor is proposed to provide some of the needed input data to this simulation system.

In the third project, the problem of predicting in an early stage the remaining life of mechanical assembly on the basis of vibration responses is dealt with. Here, time-signals are preprocessed numerically (Discrete Fast Fourier Transform (DFT) processing) in order to provide training data and input data for a classifying ANN.

The systems in the second and third project are actual examples of practical applications of (loosely) coupled connectionist/numeric systems as considered in section 2.5a.

5.1 Budgetting large software projects

Frequently, the budgetting of big software development projects suffers from serial problems. A number of causes that may constitute these budgetting problems are, for instance, the lack of empiric data from finished projects, the instability of the system specifications during the software development, the specific features of the software development process, the large number of factors that influence the development effort, the unknown influence of factors on the development effort, etc. There exists a general need for techniques which support the budgetting process. A technique that has proven its usefulness is the so-called "Function Point Analysis" (FPA) developed by IBM Data Processing Services. This technique was originally meant to measure objectively and in a uniform way the productivity of software developers, but it has also shown its value in budgetting software development projects. The FPA technique estimates the functionality of the desired software system. This functionality, expressed in function points, is transformed into effort, i.e. manhours, by using productivity curves. The productivity curves are derived from empiric data gathered from previous finished projects.

Unfortunately, FPA is only applicable when the functionality of the system can be measured (System Implementation Phase), i.e. when the greater part of the development process has already taken place. It is highly desirable to have also insight in the effort needed in an earlier stage of the software development process (Information Analysis Phase). Techniques that support this do not yet exist, mainly due to the specific features of the budgetting problems in this earlier stage of the development process.

In this research, it is tried to tackle the budgetting problem (for the earlier stages of the development process) in a way similar to the FPA technique. Just as is the case with FPA, the technique proposed actually consists of two submodels (see Fig. 15). The first submodel, the *Complexity Point Analysis* (CPA) model, transforms the problem, identified through problem characteristics, into complexity points. Hence, complexity points (being an as yet non-existing measure) are an environment-independent indicator of the scope of the problem. These complexity points, added with environment characteristics, are then transformed into effort by the second submodel, the *Productivity* model. Because of the constraints of lack of rules and empiric data, as well as the needed adaptation to changing environments, in this research both submodels are realized by (backpropagation) neural networks (see section 2.4). The topology of the networks and their training have been subject to examination.

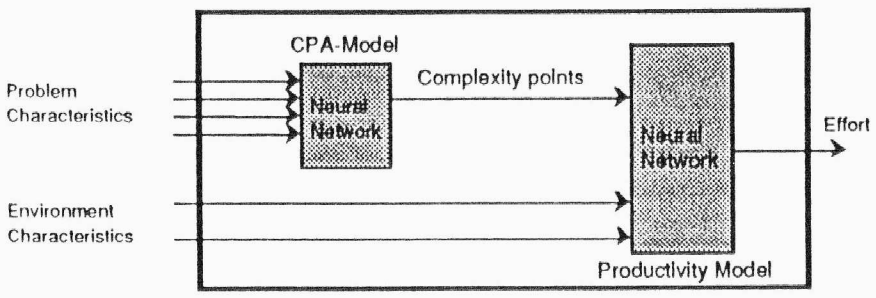


Figure 15: Neural networks in a budgetting model

The research considered is a common project of the NMB Bank (the Netherlands) and the Delft University of Technology. The results achieved are satisfactory [Pellikaan and in 't Veld, 1990]; these will be published in the near future.

5.2 Optimal air-route selection

Recent years showed an increasing trend in the number of delayed European flights with a delay of more than 15 minutes (see Fig 16). A substantial part of these delays was due to a growing congestion in the airspace. Causes for this congestion are among others: shortage of air-traffic controllers, lack of radar capacity, incompatible air-traffic control systems and the (in time and space compressed) number of flights to be dealt with. One of the options to avoid or reduce the delay caused by congestion is to fly over less popular, hence less congested, routes. It is not known in advance, however, which of the possible alternative routes has the least delay. It is also unknown whether the additional operational cost of these, often longer and more expensive, routes counterbalance the gains of less delay.

The problem of choosing the (near) optimal route in a particular situation focusses in the very essence on determining the (for the moment unknown) quantities:

— actual departure-times of each of the possible routes
— cost of delay as a function of delay-time.

The problems that arise in determining these quantities are due to the fact that there are no data about the consequences of delays and about the cost of these consequences. Furthermore, the actual time of departure for a route is unknown until the authorities (in

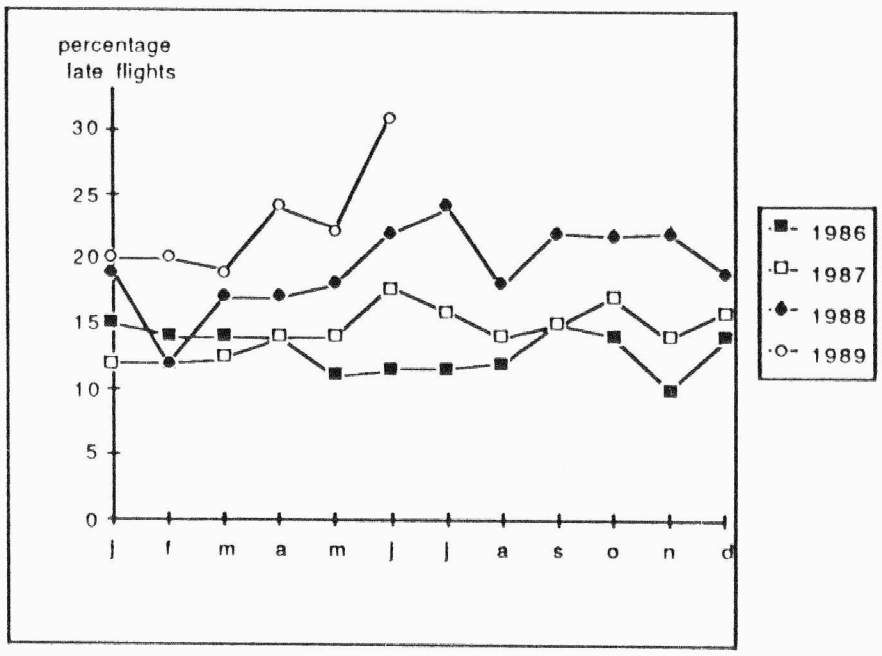


Figure 16: Flight delays from January 1986 until June 1989

the Dutch case, the RLD: Dutch Airspace Authority) have determined a time-slot for departure, while only one slot at a time can be requested from them.

A numeric simulation model to calculate the cost of a delay as a function of delay-time has been built with the use of the simulation-package STELLA (a graphical simulation tool, originally designed to model financial and economical processes). The model is based on extensive research with respect to the causalities and values of the parameters of a particular scheduled service (KL501) Amsterdam-Athens). Fig. 17 shows a typical graphical representation of a part of the model, that is mainly meant to calculate the influence of the delay on passenger yield. This submodel also deals with some elements, that are hard to quantify, such as cost of non-quality experiences by passengers as a result of a delay. The spherical shapes in Fig. 17 represent the parameters, defined as mathematical functions of other parameters or as constant values; the arrows represent relations between these parameters.

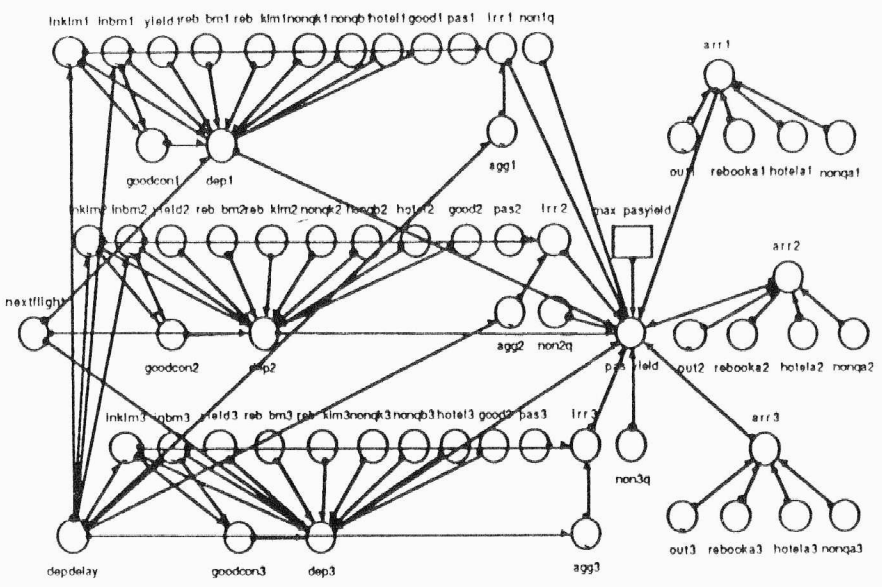


Figure 17: Submodel for calculating the influence of a delay on the passenger yield

With the aid of this model an empirical sensitivity analysis has been perfomed to determine the influence of some 55 relevant parameters on the model's output. Parameter sensitivities are determined in three different setpoints, then averaged, and the results here of are compared with their average (see Fig. 18). The sensitivity analysis shows that there is a limited number of more sensitive parameters. After further investigation it became evident that a few of these can be quantitatively evaluated without any problem, but a seven parameters cannot, since the information needed to this is incomplete for the parameters cannot directly be expressed in numeric values (e.g., passenger satisfaction). So, further research had to be concentrated on a "quantification" of these seven parameters. With them known, and given the actual departure-time per route (the main input variable for the model), the model can provide the final cost per route. The route with the minimum cost is the wanted optimal route. Unfortunately, also the actual routes' departure-times are unknown in advance and should in a way be predicted on the basis of past (incomplete) data.

The above problem might be solved by coupling to the numerical simulation system additional modules based on other techniques, such as knowledge-based or ANN approaches (or perhaps constraint propagation techniques). At present, it is tried to estimate the unknown departure-times by using an ANN approach. A (backpropagation) network architecture has been designed (and is currently being tested) to predict delays (i.e. differences between actual and scheduled departure-times) for multiple routes in a given situation on the basis of available historical data.

The project considered is a common project between KLM (Royal Dutch Airlines) and Delft University of Technology.

5.3 Guarding of a mechanical "fingerprint"

All rotating machinery generate their own vibration patterns, the analysis of which renders valuable data about the condition of the machines. This so-called "fingerprint", represented by a repetitive waveform, is composed of multiple time-varying real-time responses $I_i(t)$ (such as acceleration, velocity, pressure and temperature), detected by superior surface mounted transducers, which are mounted practically near or even within the Mechanical Assembly (MA) in order to be able to produce optimum responses. Diagnostics performed in an early stage not only permits estimation of the remaining life of each MA, but also is of particular engineering interest in order to be able to non-destructively evaluate the overall conditions and in situ determine the amount of degradation of the various subparts with respect to preconditioned tolerances.

A more efficient use of all the information embedded in the vibration signal can be achieved by characterizing the differences (due to disturbances and wear

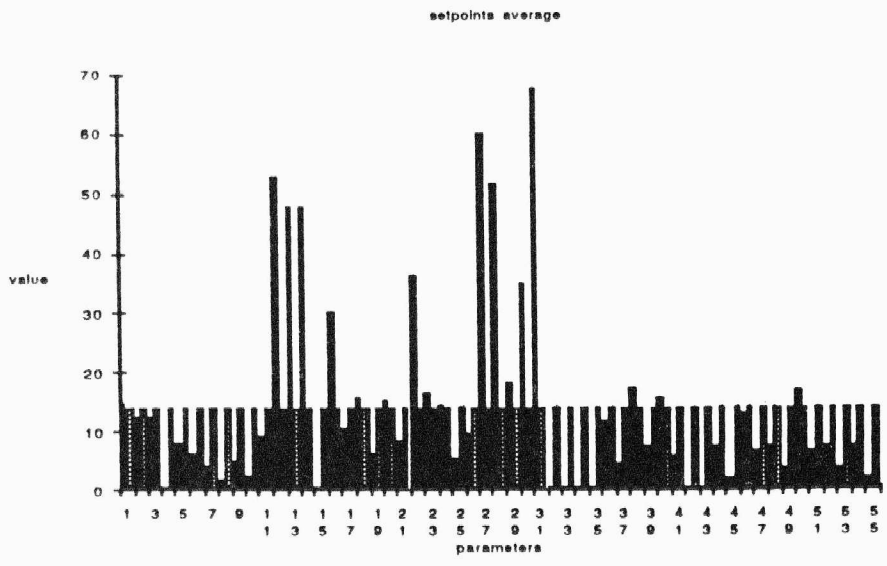


Figure 18: Parameter sensitivities

of the mechanical components within a MA) with respect to the "fingerprint" by analyzing either the Power/Frequency or Amplitude/Frequency spectrum. In order to be able to exploit the real-time situation to the utmost, the processed result of the previous training-set has to determine the moment of occurrence of the next training-set. So, superior occurring defects of increasing gravity will decrease the time with respect to the succeeding training-set. This can be achieved by developing a mechanism in which an actual trespassing of a predetermined "error" threshold appoints the moment of occurrence of the next training-set.

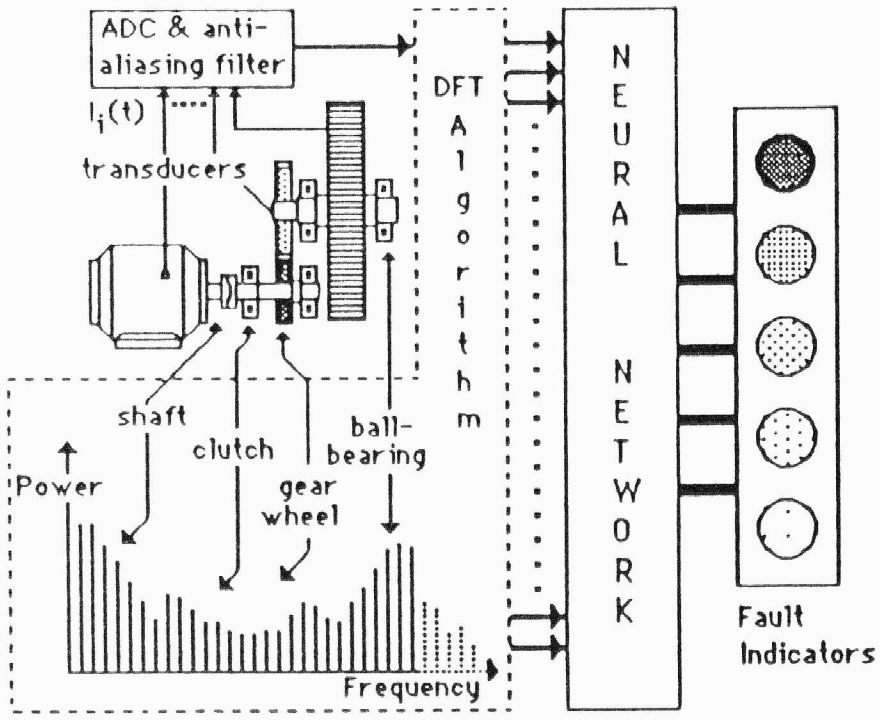


Figure 19: A block-schematic overview of the "fingerprint" analysis and guarding

After the time signals have been transformed to a discrete frequency spectrum by way of a numerical DFT algorithm, an array of numbers, representing the amplitudes of the various frequency components, is subject to a connectionist analysis with the help of an ANN algorithm (see Fig. 19). ANN techniques, supporting massively parallel networks of simple structured neurons, offer an approach to recognize differences in patterns based on automatic learning procedures. The application is attractive, not only because it provides faster responses, but also because of its capability to automatically discover irregularities in a pattern not seen or detected before. It even enables the discovery of regularities in the training signal itself as a consequence of the learning process.

A backpropagation algorithm (see section 2.4) is probably the most applied method to perform a supervised learning task, which in this context means the adaptation of an ANN in that actual outputs $O_k(t)$ approach a set of target outputs $T_k(t)$, given a training-set containing P learning patterns. But before variables or parameters can ever be adapted, first a training database of actual information and correct classification have to be processed. An overall bandwidth of at least 2 KHz combined with a frequency interleave of 2 Hz composes a database, filled up with 1 000 discrete frequency components. Classification of a certain vibration guarding pattern could easily require 100 of such learning patterns, which not only represents the actual "fingerprint" but also the alterations caused by the influences of mechanical failures.

During execution, the "interconnection topology" will first define the influence of the inputs $I_i(t)$, on the (in our case five-digits wide) actual outputs $O_k(t)$, taking into account the information of the disired results $T_k(t)$. This relationship then forms a set of weights W_{ij}, representing the variables or parameters, which will be adjusted concurrently during the learning process. The procedure passes into a so-called "learning rule", during which the weights W_{ij} are adjusted as to force the actual outputs $O_k(t)$ to approximate the desired results $T_k(t)$.

The final goal is to adapt the parameters of the ANN in such a way that it also adapts to those patterns, which were not generated before in the original training-set. A feature of essential importance when processing data with varying contents due to changes in the location of the sensors and possible deformations in the material.

The project considered has been started at Delf University of Technology and will be continued in cooperation with the Academy of Sciences in Prague (Czechoslovakia).

6. Final Note

In this paper we surveyed the usage of numeric, symbolic, connectionist and coupled systems to "problem solving". The emerging role of parallel processing in all of these has been stressed.

We have made the suggestion that integrated environments to run simulations, expert systems, artificial neural networks, and combinations of these (coupled systems) could perhaps provide — along with intelligent front- and back-ends — the ideal toolboxes for "problem solving". It is the author's opinion that the ideal hardware platform for such integrated environ-

ments is a distributed network of dedicated processors or a heterogeneous parallel computer, in which the above different techniques could run (councurrently and with mutual data exchange) on different dedicated (clusters of) processors. Such heterogeneous parallel machines do not exist. The best alternatives are provided by (massively) parallel computers that are really "generalpurpose", i.e. with powerful nodes, fast and sufficiently dense interconnection structures, extensive local memories and appropirate general-purpose software facilities. We think that, for example, NCube's 2nd-generation hypercube parallel computers, launched as (massively) parallel systems for both scientific and business purposes which also underlines their general-purpose character, are very suited for research as considered above.

There will come a time that it is feasible to implement, on an appropriate multicomputer, several large-scale intelligent systems, large-scale ANNs and complex simulation systems, that — if necessary — concurrently operate and interact with each other in a truly real-time mode. The obstacles and problems to solve on the way to such advanced "coupled-systems environment" will be numerous and so complex that today we only can start attempting to handle them. This paper intends to stimulate thinking and discussion about further research needed.

Acknowledgements

The author gratefully acknowledges that the NCube/4+ parallel computer, that was used in some of the projects described in this paper (see sections 4.4 and 4.5), has been donated by SHELL Netherlands to support research in parallel AI. He also would proudly like to announce the availability of an NCube-2 system (the smallest model: 32 nodes with 4 Mbyte local memory each; see also sections 4.1a and 6) at Delft University of Technology in the quite future; this is financially made possible by the Dutch Foundation KBS (Knowledge-based Systems).

Further, the author wants to thank Edward Frietman and (particularly) professor Henk Koppelaar (both from Delft University of Technology), and professor Ghislain Vansteenkiste (University of Ghent, Belgium) as well, for the many stimulating discussions concerned with some of the issues in this paper.

Literature

D. R. I. Ballard (1990): Conjoint computing: integrating numeric, symbolic and neural computing. In: B. Zeigler, J. I. Rozenblit (Eds.): AI, Simulation and Planning in High Autonomy Systems. IEEE Comput. Soc. Press, Los Alamitos, Ca.; pp. 194—201.

B. Bergsten et al. (1988): An advanced database accellerator. IEEE Micro, Vol. 8 No. 5, 47—63.

T. Beynon, N. Dodd (1987): The Implementation of Multi-Layer Perceptrons on Transputer Networks. Proc. of the 7th Occam Users Group.

J. P. Bigus, K. Goolsbey (1990): Integrating neural networks and knowledge-based systems in a commercial environment. Proc. of IJCNN-90 Washington D. C., Hillsdale, N. Y., Lawrence Elbraum; pp. II-463 to II-466.

D. Bounds (1989): Expert systems and connectionist models. In: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.): Connectionism in Perspective. Elsevier Science Publishers (North Holland), Amsterdam; pp. 277—282.

J. Bourrely (1989): Parallelization of a neural learning algorithm on a hypercube. In: F. André, J. P. Verjus (Eds.): Hypercube and Distributed Computers. Elsevier Science Publishers (North-Holland), Amsterdam.

W. J. H. J. Bronnenberg et al. (1987): DOOM: A decentralised object-oriented machine. IEEE Micro, Vol. 7 No. 5, 52—69.

B. G. Buchanan, E. H. Shortliffe (1984): Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA.

D. J. Burr (1987): Experiments with a connectionist text reader. In: M. Caudill, C. Butler (Eds): Proc. of the 1st Int. Conf. on Neural Networks. SOS Printing, San Diego, Ca; pp. 717—724.

D. Castillo, M. McRoberts, B. Sieck (1988): Embedded expert systems improve model intelligence in simulation experiments. In: C. C. Barnet, W. M. Holmes (Eds.): Proc. of the Summer Computer Simulation Conference (Seattle, Washington). Society for Computer Simulation Int., San Diego, Ca., pp. 591—597.

J. S. Conery (1987): Parallel Execution of Logic Programs. Kluwer Academic Publishers, Amsterdam.

G. W. Cottrell, P. Munro, D. Zipser (1987): Image compression by backpropagation: an example of extensional programming. Advances in Cognitive Science, Vol. 3; Ablex, Norwood, NJ.

J. Dayhoff (1990): Neural Network Architectures / An Introduction. Van Nostrand Reinhold, New York.

G. I. Doukidis, R. J. Paul (1985): Research into expert systems to aid simulation model formulation. J. Op. Res. Soc. 36, 319—325.

A. S. Elmaghraby, V. Jagannathan (1985): An expert system for simulationists. In: B. Birtwistle (Ed.): AI, Graphics and Simulation. Society for Computer Simulation Int., San Diego, Ca., pp. 106—109.

R. A. Fellheim (1986): A knowledge-based interface to process simulation. In: E. J. H. Kerckhoffs, G. C. Vansteenkiste, B. P. Zeigler (Eds.): AI Applied to Simulation. Society for Computer Simulation Int., San Diego, Ca.; pp. 97—102.

P. A. Fishwick, R. B. Modjeski (Eds.), 1991: Knowledge-based Simulation / Methodology and Application. Springer Verlag, New York.

M. J. Flynn (1972): Some computer organisations and their effectiveness. IEEE Trans. on Comp., Vol. 21, 948—960.

K. Forbus (1984): Qualitative process theory. In: D. G. Bobrow (Ed.): Qualitative Reasoning about Physical Systems. North-Holland, Amsterdam; pp. 85—168.

M. S. Fox, N. Husain, M. McRoberts, Y. V. Reddy (1989): Knowledge-based simulation: An artificial intelligence approach to system modelling and automating the simulation life cycle. In: L. E. Widman, K. A. Loparo, N. R. Nielsen (Eds.): Artificial Intelligence, Simulation and Modelling. Wiley; pp. 447—486.

S. I. Gallant (1988): Connectionist expert systems. In: Communications of the ACM, Vol. 31 No. 2.

A. Gupta (1987): Parallelism in Production Systems. Pitman Publishing, London.

P. Harmon, D. King (1985): Expert Systems — Artificial Intelligence in Business. Wiley, New York.

F. Hayes-Roth, D. . Waterman, D. B. Lenat (1983): Building Expert Systems. Addison-Wesley, Reading, MA.

J. A. Hendler (1989): Problem solving and reasoning: a connectionist perspective. In: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.): Connectionism in Perspective. Elsevier Science Publishers (North Holland), Amsterdam; pp. 229—244.

W. D. Hillis (1985): The Connection Machine. MIT Press.

R. W. Hockney, C. R. Jesshope (1988): Parallel Computers 2 — Architecture, Programming and Algorithms. Adam Hilger, Boston.

T. Johnson (1984): The Commercial Application of Expert Systems Technology. Ovum Ltd., London.

W. J. Karplus (1976): The spectrum of mathematical modelling and systems simulation. In: L. Dekker (Ed.): Simulation of Systems /

Proc. of 8th IMACS (formerly AICA) Congress. North Holland, Amsterdam; pp. 5—13.
E. J. H. Kerckhoffs, H. Koppelaar, H. J. van den Herik (1989): Toward parallel intelligent simulation. In: L. E. Widman, K. A. Loparo, N. R. Nielsen (Eds.): Artifical Intelligence, Simulation and Modelling. Wiley, New York; pp. 207—230.
E. J. H. Kerckhoffs, G. C. Vansteenkiste (1990): Parallel processing in biological systems. In: D. P. F. Moeller (Ed.): Advanced Simulation in Biomedicine. Springer-Verlag, New York; pp. 1—9.
E. J. H. Kerckhoffs, G. C. Vansteenkiste, B. P. Zeigler (Eds.), 1986: AI Applied to Simulation. Society for Computer Simulation Int., San Diego, CA.
E. J. H. Kerckhoffs, F. W. Wedman, E. E. E. Frietman (1991): Speeding up backpropagation training on a hypercube computer. In: M. Novák, E. Pelikán (Eds.): Neural Network Applications (IMACS Int. Symp. NEURONET '90). Elsevier Science Publishers (North-Holland), Amsterdam 1991.
G. A. Korn (1989): A new environment for interactive neural network experiments. Neural Networks, Vol. 2, 229—237.
J. S. Kowalik, C. T. Kitzmiller (Eds.), 1988: Coupling Symbolic and Numerical Computing in Expert Systems, II. North-Holland, Amsterdam.
R. R. Maijer, E. J. H. Kerckhoffs (1990): Towards the modelling of knowledge-based control systems in DUTIES (a parallel environment for coupled numeric/symbolic systems). In: G. C. Vansteenkiste, E. J. H. Kerckhoffs, H. Muller (-Malek), F. Broeckx (Eds.): Proc. of ESS90 (European Simulation Symposium on Intelligent Process Control and Scheduling & Discrete Event Systems). Society for Computer Simulation Int., San Diego, Ca; pp. 73—79.
J. M. Mellichamp, A. F. Wahab (1987): An expert system for FMS design. Simulation 48, 201—208.
G. V. Merkuryeva, Y. A. Merkuryev, H. T. Toivonen (1990): Knowledge-based simulations systems — A survey. Report 90—12. Abo Academy, Department of Chemical Engineering, Process Control Laboratory. Abo, Finland.
R. Muetzelfeldt, A. Bundy, M. Uschold, D. Robertson (1986): EC—An intelligent front-end for ecological modelling. In: E. J. H. Kerckhoffs, G. C. Vansteenkiste, B. P. Zeigler (Eds.): AI Applied to Simulation. Society for Computer Simulation Int., San Diego, Ca.; pp. 67—70.
F. Neelamkavil (1987): Computer Simulation and Modelling. Wiley, New York.
A. Newell, H. A. Simon (1972): Human Problem Solving. Prentice-Hall.
N. Nielsen (1987): Applicability of AI techniques to simulation models. In: B. T. Fairchild (Ed.): Simulator IV. Society for Computer Simulation Int., San Diego, Ca.; pp. 121—122.
R. M. O'Keefe (1986): Simulation and expert systems — A taxonomy and some examples. Simulation 46, 10—16.
J. M. Ortega (1988): Introduction to Parallel and Vector Solution of Linear Systems. Plenum Press, New York.
Y. H. Pao, D. J. Sobajic (1990): Nonlinear process control with neural nets. Neurocomputing 2, 51—59.
R. Pellikaan, L. in 't Veld (1990): Neural Modelling for Budgetting. MSc-thesis. Internal Report Delft University of Technology (Faculty of Technical Mathematics & Informatics), Delft, The Netherlands.
A. Petrowski, L. Personnaz, G. Dreyfus, C. Girault (1989): Parallel implementations of neural network simulations. In: F. André, J. P. Verjus (Eds.): Hypercube and Distributed Computers. Elsevier Science Publishers (North-Holland), Amsterdam.
L. C. Rabelo, S. Alptekin (1989): Integrating scheduling and control functions in Computer Integrated Manufacturing using artificial intelligence. Computers and Industrial Engineering, Vol. 17 No. 1—4, 101—106.
N. Roberts, D. F. Anderson, R. M. Deal, M. S. Garet, W. A. Shaffer (1983): Introduction to Computer Simulation: The Systems Dynamics Approach. Addison-Wesley, Reading, MA.
D. E. Rumelhart, J. L. McClelland (1987): Parallel Distributed Processing. MIT Press, Cambridge, MA.
J. Russel (1989): A knowledge-based assistant for a flight simulator instructor. In: J. K. Clema (Ed.): Proc. of the 1989 Summer Computer Simulation Conference (Austin, Texas). Society for Computer Simulation Int., San Diego, Ca.; pp. 606—612.
U. Schendel (1984): Introduction to Numerical Methods for Parallel Computers. Halsted Press.
J. F. Schreinemakers, D. S. Touretzky (1990): Interfacing a neural network with a rule-based reasoner for diagnosing Mastitis. Proc. of IJCNN-90 Washinton D. C., Hillsdale, N. Y., Lawrence Elbraum; pp. II-487 to II-490.
T. J. Sejnowsky, C. R. Rosenberg (1987): Parallel networks that learn to pronounce English text. Complex Systems 3, 145—168.
R. Serra (1989): Dynamical systems and expert systems. In: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.): Connectionism in Perspective. Elsevier Science Publishers (North Holland), Amsterdam; pp. 265—275.
B. Soucek (1989): Neural and Concurrent Real-Time Systems / The Sixth Generation. Wiley, New York.
J. A. Spriet, G. C. Vansteenkiste (1982): Computer-aided Modelling and Simulation. Academic Press, New York.
L. Steels (1989): Connectionist problem solving — an AI perspective. In: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.): Connectionism in Perspective. Elsevier Science Publishers (North Holland), Amsterdam; pp. 215—228.
P. Szolovits (1987): Expert systems tools and techniques: past, present and future. In: W. E. L. Grimson, R. S. Patil (Eds.): AI in the 1980s and Beyond / An MIT Survey. MIT Press, Cambridge, MA; pp. 43—74.
S. L. Tanimoto (1987): The Elements of Artificial Intelligence (an introduction using LISP). Computer Science Press, Rockville, Maryland.
P. C. Treleaven (1989): Neurocomputers. Int. J. of Neural Computing. Vol. I, 89/1, 4—31.
P. C. Treleaven (Ed.), 1990: Parallel Computers / Object-oriented, Functional, Logic. Wiley, New York.
P. C. Treleaven et al. (1987): Computer Architectures for Artificial Intelligence in Future Parallel Computers. Lecture notes in Computer Science, Vol. 272. Springer-Verlag.
L. Uhr (1987): Multi-computer Architectures for Artificial Intelligence / Toward fast, robust, parallel systems. Wiley, New York.
L. A. A. Vercauteren, R. A. Vingerhoeds, M. Lee, L. Boulart (1989): Pattern directed real-world interface. In: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.): Connectionism in Perspective. Elsevier Science Publishers (North Holland), Amsterdam; pp. 455—461.
L. K. Vermeersch, G. C. Vansteenkiste, E. J. H. Kerckhoffs (1990): Introducing neural networks for feature extraction in system modelling. In: B. Svrcek, J. McRae (Eds.): The Proceedings of the 1990 Summer Computer Simulation Conference. The Society for Computer Simulation Int., San Diego, Ca; pp. 776—780.
J. Walters, N. R. Nielsen (1988): Crafting Knowledge-based Systems. Wiley, New York.
C. J. Wang, C. H. Wu, S. Sivasundaram (1989): Neural Network Simulation on Shared-Memory Vector Multiprocessors. Proc. of Supercomputing (Reno, Nevada). ACM Press, New York.
P. D. Wasserman (1989): Neural Computing / Theory and Practice. Van Nostrand Reinhold, New York.
I. Watson et al. (1987): Flagship computational models and machine architecture. ICL Tech. Journal, Vol. 5, 555—574.
L. E. Widman, K. A. Loparo (1989): Artificial intelligence, simulation and modelling: A critical survey. In: L. E. Widman, K. A. Loparo, N. R. Nielsen (Eds.): Artificial Intelligence, Simulation and Modelling. Wiley, New York; pp. 1—44.
L. E. Widman, K. A. Loparo, N. R. Nielsen (Eds.), 1989: Artificial Intelligence, Simulation and Modelling. Wiley, New York.
B. P. Zeigler (1976): Theory of Modelling and Simulation. Wiley, New York.
B. P. Zeigler (1984): Multifacetted Modelling and Discrete Event Simulation. Academic Press, New York.

INFORMATION STORAGE IN NEUROCOMPUTING

V. Cimagalli, M. Balsi), A. De Carolis**)*

Abstract:

In this paper we introduce the concept of "relational informations" as a peculiar property of neurocomputers. In fact, due to the distributed way of storig and processing information, the organized structure of a neurocomputer adds significant relations to data fed to it and this is the reason why it is able to generalize from a limited number of training inputs. After having summarized the most significant results related to our problem, available in the literature, we suggest methods for measuring the said relational information both in a dynamic system as a chaotic map and in a neural network of any kind.

1. Introduction

Regardless of the particular kind of neural network under consideration, we may look at it as to black box with an input and an output. Its purpose is to process the information contained in the input signal in order to obtain some specific result as, e.g., to recognize patterns, to classify data, to detect moving objects, to solve a problem of minimum, etc. This means that the net maps the space of all the admissible inputs into the space of its outputs according to some well defined rule. The peculiar characteristic of neural networks is that such a rule (i.e. the rule or the algorithm that allows the desired result to be obtained) does not need to be explicitly formalized as in the case of AI. Nevertheless such a rule exists and is an amount of information stored into the network. A deep understanding of the mechanism by which such an information is loaded and stored in the network should be paramount for using the best way each type of neural network, for choosing the particular network most suitable for solving a given problem and for devising new and more powerful architectures.

Knowledge, a systematic comprehension of information processing by means of neural networks is still an open problem far from being solved. In our opinion this is due to the twofold way through which the research in such a field has developed. On one hand we note the works of neurophysiologists and physicists, the aim of which has been to develop models of physical and chemical processes occurring in the learning of living beings: they did not pay too much attention to questions related to information or made a misleading use of concepts proper to information theory. On the other hand, engineers, who should have been more familiar with information theory, generally have preferred to overcome deep theoretical problems and work hard to solve actual problems by means of heuristic methods.

Among the more significant problems related to the study of the behavior of information in neural networks, we may quote:

1. Is the classical definition of information, as stated by Shannon in 1948 for the purpose of its use in communication theory, still valid in our case or does it need to be revisited? And the algorithmic theory of information too, how much turns out to be useful in a computing structure where no sequential algorithm is present?

2. As the "content" of a message (i.e. of the input to the neural network) and the "association" of the meanings of different inputs play an important role in neurocomputing, how, where and in what quantity is information stored in the network for accomplishing semantic and relational tasks?

3. Independent of the fact that learning is previous to, or simultaneous with the processing phase, do different sets of inputs having the same amount of bits, or the same set stored in different architectures, or in the same architecture in different places with different strategies, give rise to the same amount of stored information?

4. If the answer to the last question is negative, (a) is it convenient to speak of a "relation information" as the information related to the structure of the mathematical relations and/or to the architecture of the network that process it and (b) is it possible to devise some method for measuring it?

In part 2 of our paper we will briefly review the main trends and the main results available in the literature related to our previous questions. In part 3 we will introduce the concept of relational information as applied to a mathematical relationship and to the ability of generalization of a neural network.

*) Prof. V. Cimagalli, Dr. M. Balsi Facolta' d' Ingegneria — University of Rome „La Sapienza", via Eudossiana 18-00-184 Roma, Italy

**) Dr. A. De Carolls is now with Ericsson Sielte S.p.A., via Campo Romano, 71-00173 Roma, Italy

2. Classical approaches to informational question

2.1 Semantics in neurocomputing

A general neural network can be depicted as in *Fig. 1*.

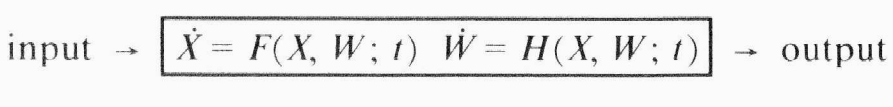


Fig. 1

where X is the vector of state, W is the vector defining the dynamics (in the more current cases it is the vector of weights) and F and H are nonlinear functionals or functions.

It would be advisable that a neural network be able to perform the following tasks:

1. To recognize the inner coherency of its input data;

2. To classify them by detecting some of their invariants (e.g. with respect to translation, rotation, added noise, etc.) and taking into account its previous experience.

3. To take some appropriate action in the case of meaningless input data.

From the standpoint of information theory, such a set of operations may be viewed as a twofold coding function: a dynamic or static coding definition and an encoding of the input signal. However, as the concept of "meaning" is involved, the question arises as to the quality and the amount of information that need to have been stored into the network in order to render it amble to distinguish between a "meaningful" and "meaningless" input signal.

In the late fifties D. M. Mac Kay, in a series of broadcasting talks and papers, faced the problem of semantics in communication and in representation of some physical or non-physical (mental or ideational) aspects of experience. He said [1] that "information may be defined in the most general sense as that *which ads to a representation*" Hence a criterion for defining *false or true* a set of input data is strictly related to the diminution or to the increase of the extent of correspondence between this set and the original represented by it. Applying this concept to neurocomputing, we can deduce that: 1. it is convenient to relate the amount of information entering a network with the *dynamic* operation of changing the state of it, and 2. it is *necessary* to refer to an *original*. Such an original may be a thing (e.g. a learned pattern in a usual neural network) or a rule. This is well explained again by Mac Kay [2] when he examines the sentence "This message is meaningless" and writes that it means "This message lacks a selective function . . . it has no selective relationship to . . ." and adds that "now it is possible for something to lack a selective function for two reasons: (*a*) one or more of its component terms may be undefined — may have no selective function — so that the total selective operation is undefined; (*b*) two or more of the component selective functions may be incompatible, so that the total selective operation cannot be completed." Once again we are led back to the problem of storing in the computing architecture either definitions or rules for judging of compatibility.

More recently Haken [3] observed that it is possible to attribute a meaning to a message only if the response of the receiver is taken into account. Therefore he deals with system interacting with surroundings, i.e. with open systems from the point of view of thermodynamics, and models the receiver by the Langevin equation. Under this hypothesis it is possible to judge if a given message is useful or not; the usefulness coinciding with the jump of the receiving system from one attractor (fixed point, limit cycle or strange attractor (fixed point, limit cycle or strange attractor) to another. An input is useless if it leaves the system in the previous state and therefore it is "meaningless".

If, in addition, we attribute to the attractors a relative weight $0 \leq p'_j \leq 1$, normalized in such a way that $\Sigma_j p'_j = 1$, it is possible to classify messages according to their utility, and hence according to their content of "meaning". Let M_{jk} be the normalized ($\Sigma_k M_{jk} = 1$) probability for the attractor k reached after input j has been received; we define as the relative usefulness p_j of the input j as the following:

$$p_j = \sum_k L_{jk} p'_k = \sum_k \frac{M_{jk}}{\sum_j M_{jk} + \varepsilon} p'_k \qquad (1)$$

where ε is an arbitrary small quantity that prevents the numerator and denominator from going to zero simultaneously.

We note that such an approach (*a*) requires an a *priori* classification of the attractors (i.e. it needs some information related to the "meaning" of admissible inputs to be stored into the receiver — in our case into the neural network) and (*b*) gives a criterion for choosing the "most meaningful" messages. In a certain sense we get an answer to question 3 of our Introduction, as according to Haken's classification not all the sets of inputs having the same amount of bits are equally useful and we may find the existence of some "relational information".

It should be pointed out explicitly that such a "relational information" as introduced in question 4 of our Introduction is different from Mac Kay's "structural information". In fact the latter is defined as *the basal multiplicity of distinguishable groups or clusters* [1], while our "relational information" stresses not a merely dimensional property, but the proper choice among equidimensional sets that are not equivalent because of *different relations* between their elements. As a trivial example we may note that the set of the numbers 10 and 2 has different meanings (and the resulting number needs a different number of bits to be represented) if we choose one or another of the following expressions:

$$10+2, \log_2 10, 2^{10}, 10^2. \quad (2)$$

In Haken's work the necessity of storing in the net an *a priori classification* of the relative importance of the attractors is still present. This renders the net scarcely useful in dealing with time varying inputs, as the structure of its state space is fixed in advance and moreover it needs a rigid classification of the things that are meaningful. Such a way of operating corresponds to verifying the meaning according to the rule (*a*) of Mac Kay that we quoted above in this paragraph (i.e. the selective function is based on definition of terms). On the contrary it would seem of more general utility and more similar to the behavior of living beings to use the rule (*b*) of Mac Kay (i.e. to use logical rules for judging of the inner coherency of the input).

Some efforts in this direction have been made by our research group. We are pursuing research on a architecture with weight dynamics granting a continuous redefinition of its state space [4] [5] and on an architecture able to recognize moving objects [6].

2.2 Information and organization

Obviously the behavior of a neural network depends on its architecture and on the actual values of its parameters, i.e. on the way its data is organized. It is well known that information is stored in a distributed fashion and therefore we may argue that the architecture itself is able to store information in a more or less efficient way.

Complex systems were considered by Atlan [7] who studied a system composed by several subsystems connected to each other by noisy communication channels and deduced equations describing the information supplied to an observer as a function of time, noise and interaction with environment. Although his analysis is very clever, its hypothesis seems at present to be too far from being immediately applicable to neurocomputing.

Other authors studied problems specifically related to neural networks, but the influence of organization on information is considered from the point of view of how dynamics evolves rather than from the point of view of how information is stored. So Parisi [8] found that asymmetry in the synaptic strengths may be crucial for the process of learning, while Tsuda et al. [9] proposed a mechanism with positive and negative feedback that gives rise to a type of chaos that can be an effective gadget for memory traces.

If we take again into consideration the Langevin equation in the case of a system made up of several subsystems and limit ourselves to considering only deterministic stationary solutions (i.e. fixed points), it is possible to arrive at some interesting conclusions. This has been done by Haken [10] and the results may be summarized as follows: (*a*) the amplitudes of stable modes (modes corresponding to eigenvalues $\lambda_i < 0$) can be computed as a function of the amplitudes of the istable modes (i.e. with $\lambda_i < 0$) and (*b*) as a consequence, it may happen that even important changes in the macro-structure of the system require only a little change of the amount of information necessary for describing it and this corresponds to a strong compression of information. Unfortunately such interesting results have been obtained at the expense of strong simplifying hypotheses, the most important being linearization. Therefore the criticism expressed with regard to the work of Atlan applies once again.

Nevertheless the quoted works may be considered as the first useful stones for building up a systematic theory of relational information.

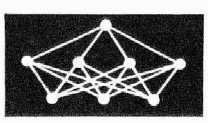
2.3 Memory capacity of a neural network

As neural networks are mainly used as classifiers, it is evident that one of the most important problems is to know how many different patterns a given network is able to recognize correctly. In the case where the network acts as a dynamic system that evolves from an initial point or state (the input pattern) to a stable attractor (the corresponding reference pattern), a very simple and intuitive formulation of the problem seems to be that it consists in finding the number of attractors, each attractor corresponding to a memorized pattern. Nevertheless such a problem is more subtle than how it may appear at first glance. First of all, it is obvious that the capacity of a network depends on the architecture of the network itself and on its size, but there are two other important factors: the learning algorithm and an exact definition of what the words "to recognize correctly" mean. Therefore, besides being of great importance in the field of neurocomputing, the estimate of the capacity of a neural network has been a challenging topic in information theory. Despite this fact, some results have been obtained only in very particular cases and a general theory is not yet available.

Only the Hopfield network with one and three layers has been investigated and some sufficient condition is known for cellular neural networks. Huh and Dickinson [11], extending results obtained some years earlier by McEliece et al. [12], computed values of capacity for a Hopfield network with only one layer, using the outer product algorithm for learning, under different conditions of operation, which differ from each other because of the use of the idea of probability in defining what we called "proper recognition". From there, in the asynchronous case three different kinds of convergence follow which we summarize in this way:

— convergence *a*: at every step the point in state space representing the actual state of the network moves along a straight line toward a fixed point;

— convergence *b*: movements of the state point occur *with* the *amount of probability great but less than one* toward a fixed point that ultimately is realched as in case *a*;



— convergence *c*: with a great probability, the state point reaches a *new* fixed point that is much nearer to the correct point than the starting point was.

The said values of capacity $C(n)$, where n is the number of nodes of the network, are:

A) Convergence *a* is required for *almost all* the memorized patterns:

$$C(n) = \frac{(1-2\alpha)^2}{2} - \frac{n}{\ln(n)} \tag{3}$$

where $0 \leq \alpha \leq 1/2$ is a parameter (called radius of convergence of the associative memory) associated with the dimension of the basin of attraction.

B) Convergence *a* is required for *all* the memorized patterns:

$$C(n) = \frac{(1-\alpha)^2}{4} - \frac{n}{\ln(n)} \tag{4}$$

C) Convergence *b* is required for *all* the memorized patterns:

$$C(n)_c = \frac{n}{4 \cdot \ln(n)} \tag{5}$$

D) Convergence *b* is required for *almost all* the memorized patterns:

$$C(n)_D = 2 \cdot C(n)_c \tag{6}$$

E) Convergence *c* is required and an amount of error of 10^{-4} is admitted:

$$C(n)_E \cong 0.0723\, n \tag{7}$$

In the case that learning is made by means of the spectral algorithm, Venkatesh and Psaltis [13] found that:

$$C(n)_s = n \tag{8}$$

A greater value of capacity was found by Mitchinson and Durbin [14] in the case of a three layer Hopfield network, with n input nodes, h hidden nodes and s output nodes ($s \leq h \leq n$). However it should be noted than in this case they assumed as capacity C the number of input-output pairs that the network can store while the error probability does not exceed 0.5. Under these assumptions they arrived at the following inequalities:

$$2n \leq C \leq nt \cdot \log(t) \tag{9}$$

where $t = 1 + h/s$.

In a recent paper [15] Tan, Hao and Vandewalle found a sufficient condition for k sub-patterns to be stored in a cellular neural network using the Hebb rule. This rule is founded on some relations concerning the Hamming distance between the patterns to be stored as well as betveen them and the patterns to be recognized. Probability is not at all involved in the definition of capacity. Their studies are still in progress, as the said condition seems to be too restrictive and far from the necessary one.

All of the studies noticed, focused on capacity, considered only one aspect of information storage in neurocomputing, i.e. what is equivalent to memory size in sequential machines. An attempt to encompass such a limited point of view has been made by introducing in some way the concept of probability, but the problem of evaluating how much information, useful for processing the incoming information, is stored *both in the memory and in the architecture of the network*, has not yet been faced.

3. The relational information

We think that a neurocomputer stores some data but also, even more importantly, an algorithm for using these data. Moreover the structure of the set of stored data can itself contain more or less information according to the task to be performed, even if the amount of stored bits is invariant. Having such an idea in mind, we started to investigate how to suggest some methods for evaluating such particular information, hidden within the relations implied by a formula or by the architecture of a network.

In the present paper we limit our attention to neural networks that act as dynamic system evolving to attractors that are fixed points only. This is due to the fact that only this case was taken into account in the current literature on the topic of our interest and we summarized. However it is also well known that other kinds of attractors can be used in neurocomputing and in particular our research group is investigating the use of chaotic dynamics as possible devices for storing information [16].

For this reason we present now in the next sub-paragraph an original study on the measure of information stored in chaotic dynamics [17]. Such a research also led to rather interesting results in the field of communication as a means for reducing transmitted information.

The second sub-paragraph will deal more directly with neurocomputers and will report on research on the ability of a neural network to acquire knowledge by induction [18].

3.1 Information properties of chaotic dynamic

In this section we discuss how information is produced, processed and stored by a chaotic dynamic system: we will show that such a system has a behavior which is quite peculiar as compared both to ordinary deterministic and to stochastic systems. A series of data obtained from observations of the evolution of

a chaotic system displays a hidden structure that can be recognized and exploited.

The most striking feature of chaotic dynamics is a strong local instability: orbits diverge locally exponentially (at least in one direction) and are eventually kneaded. Supposing that we can divide the state space into a finite partition (because of limits in the precision of measuring or computing), it happens that the points contained in one of the subsets spread over many subsets and eventually fill the whole attractor under the action of the dynamics. This menas that the entropy of the set is varied, and information is not conserved; as a consequence we lose ability to forecast the state into the future. This is explained in physics by observing that the system is non-conservative, and described by means of instability parameters such as Lyapunov exponets. From a mathematical point of view, topology and the theory of measure are applied, and scaling parameters computed.

Both approaches to the problem are found to be insufficient in engineering because the dynamics is described globally, but no knowledge is available about the semantics, that is how information flows through and is processed by the system. K. Matsumoto and I. Tsuda [19] dealt with the problem of tracking the information about initial conditions, but in the quoted paper they kept themselves on a qualitative level.

We claim that to understand the informational properties of chaos a study is needed of the structure of the data produced by the evolution of the system. The key consideration for our reasoning is that chaotic dynamics lives on the border between determinism and change: even if the system is totally determined, the knowledge of it does not allow for long-term prediction, so that the behavior appears eventually random, and our inability to follow the orbits is inherent in the structure of the system. This structure should be recognized and respected when attempting to measure information.

Looking at a stream of data produced by a chaotic system (we have studied maps on the unit interval that are time-discrete), it is possible to extract from it an approximate representation of their generation law. However, even if we were able to reconstruct it exactly, as we have argued above, this is not enough to predict future data. So if we want to compute the same data by letting the reconstructed dynamics evolve, we need additional information to keep the precision of the computation to a given standard; for one-dimensional systems the average information change amonts to the Lyapunov exponent for each time step. We divide in this way the stream of data considered into two streams: the first, which we call "prediction flow", conveys information about the dynamics, e.g. parameters of a reconstruction, the second, called "unpredictable flow", completes the information about the actual data. We note here that the amount of information stored in the prediction flow is much higher than one could expect by merely computing the number of transmitted bits. This is due to the fact that, as already stated, it conveys information about the dynamics, or, in other words, "relational information".

We have developed a model of a communication system that implements this splitting of information and tested it by computer simulation. As was conjectured, this system does information compression, which is stronger when the system is less chaotic, This is obtained by processing the data to extract a polynomial reconstruction of the chaotic function, using an algorithm developed in our research group [21], and transmitting the coefficients of the polynomial as prediction flow; the unpredictable flow is made of the symbolic dynamics associated with the system. Operating like that, the information flow can be dramatically reduced, because the amount of information required for transmitting the prediction flow is almost negligible even with a limited amount of data processed, while the unpredictable flow conveys one symbol per data (data are real numbers!).

The following computations will show how a storage of information is produced by attaching to a set of numbers the meaning of being coefficients of a polynomial approximating the dynamical function. Let $n_c(\sigma, N)$ be the number of bits per data needed to represent a data flow of length N within a mean square error of σ, by applying the optimized chaotic coding sketched above. Let $n_u(\sigma)$ and $n_{opt}(\sigma)$ be the same quantities in the case where we apply a traditional uniform or statistically optimized quantization (we consider here also non-integer values of n_u and n_{opt}, obtained by reversing the formulas for $\sigma(n)$ that can be found in the literature [22]).

We define "statistical information" as the difference

$$I_s = n_u(\sigma) - n_{opt}(\sigma). \tag{10}$$

This quantity is non-negative, and has an explicit expression which does not depend on σ, but just on the shape of the distribution of the data. Statistical information measures the amount of information contained in the statistical structure of the data.

Additionally, we denote the "deterministic information relative to the N data" as

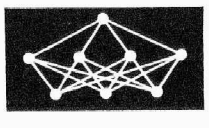
$$I_d(\sigma, N) = n_{opt}(\sigma) - n_c(\sigma, N) \tag{11}$$

and the "absolute deterministic information of the system" (or just "deterministic information")

$$I_d(\sigma) = \lim_{N \to \infty} I_d(\sigma, N). \tag{12}$$

Deterministic information measures the amount of information contained in the deterministic structure of the data. The behavior of $I_d(\sigma)$ must obey the conditions:

$$\lim_{\sigma \to 0} I_d(\sigma) = +\infty \tag{14}$$



$$\lim_{\sigma \to +\infty} I_d(\sum) = 0^+ \qquad (14)$$

which correspond to the limit situations of exact or no knowledge of the dynamics.

As an example we the case of a very simple chaotic system described by the relation

$$x_{n+1} = 0.95 \sin(\pi x_n) \qquad (15)$$
$$x_i \in [0, 1].$$

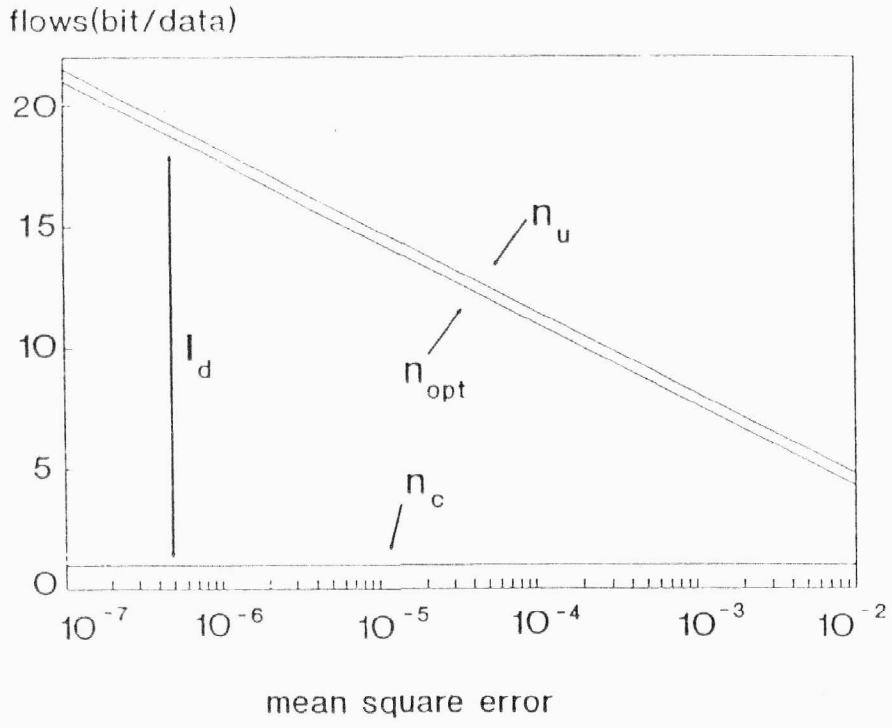


Fig. 2 Information flows as a function of the accepted error, for function 0,95 sin(pi. x)

Figure 2 shows $n_u(\sigma)$, $n_{OPT}(\sigma)$ and $n_c(\sigma)$ on a broad range of σ; n_c is not shown as depending on N because it settles on a value very close to 1 independet of σ already for N in the range of thousands, due to the fact that the prediction flow is in this case very low, so that the total flow is almost completely the flow of symbols that are here binary.

It is apparent that I_d has and exponetial behavior, which is consistent with the conditions stated above. It also evident how large the deterministic information is, as compared to the total.

The definition of deterministic information can give us a clearer view of how information is stored in a mathematical expression. At first consideration, we should say that a known dynamic function stores infinite information and produces nothing neew (i.e. unpredictable) in its evolution. Actually, as in no practical case may we suppose an exact knowledge or representation of the function, expecially when dealing with chaos, deterministic information may be exploited to measure the amount of information linked to the approximate knowledge of a dynamic function, separating it from the information linked to the pseudorandom behavior peculiar of chaos.

The study of such quantities could provide valuable hints for the design and operation of neural networks exhibiting chaotic evolution.

3.2 Information involved in the generalization process of a neural network

We consider below the deductive and inductive characteristics of the neural processing of information and give a measure of the generalization capability of a neural network.

PROPOSITION 1: *A digital neural network is able to implement any boolean logic function: for any given algorithm there is an appropriate neural network that implements it.*

The dimension of the network (number of nodes and number of synaptic connections) will depend on the particular algorithm and on the chosen architecture, but the focal point is the capability of a neural network to implement a logical function anyway: a connectionist system is a universal deductive automaton [23].

DEFINITION 1: *Generalization by induction in the neural information processing is the capability to learn an algorithm without using the entire amount of information about it.*

If we select a subset of possible inputs to the neural network (an appropriate training set) and, after the learning phase, the network is able to make an exact mapping of the inputs into the outputs, according to the algorithm, we can say that the network performs a total generalization. On the contrary, if the number of the correct outputs of the system is random we say that the generalization process did not happen.

Now we consider a neural network with n_{in} and n_{out} binary inputs and outputs. Moreover we define an algorithm p^* as an operator such that for all $2^{n_{in}}$ input binary arrays the $2^{n_{out}}$ output arrays are univocally defined (*Fig. 3*)

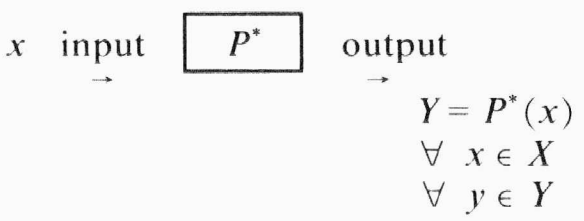


Fig. 3. Pertaining to the definition of "algorithm"

PROPOSITION 2: *The amount of information necessary for a system with n_{in} and n_{out} binary inputs and outputs to implement and algorithm P^* is expresed by*

$$I_{net}(P^*) = \log_2 \frac{1}{P} \text{ bit} \qquad (16)$$

where P is the probability that the network, *untrained*, is able to respond correctly to *all* the inputs:

$P = Pr(Y = P^*(x), \ \forall x \in X$ and $\forall y \in Y$ (untrained netvork)

Intuitively this probability P is smal and then the quantity $I_{net}(P^*)$ will be a large amount of information. To evaluate the said probability P we assume that:

A) The n_{out} events are statistically independent and equiprobable, i.e.:

$$q_j = Pr(y_j = P^*(x) \mid \forall x \in X \text{ (untrained network)} \tag{17}$$

$$q_j = q \quad \forall j = 1 \ldots n_{out}$$

B: In the situation of an untrained neural network, synaptic connections have random values and theorefore

$$q = 0.5$$

C: The events "exact outputs $y \in Y$ of the *untrained* network, according to the algorithm", are statistically independet.

Under hypotheses A and C we obtain the following expression for the probability P:

$$P = Pr\left(\{Y^{(1)} = P^*(x^{(1)})\}, \ldots, \{Y^{(2^{n_{in}})} = P^*(x^{(2^{n_{in}})})\} \,/\, / \text{ untr. net.}\right) = \left(q^{n_{out}}\right)^{2^{n_{in}}} \tag{18}$$

Now, from (18), hypothesis B and Proposition 2, we are able to calculate the following expression for the information amount $I_{net}(P^*)$:

$$I_{net}(P^*) = 2^{n_{in}} \cdot n_{out} \quad \text{bits} \tag{19}$$

This is the amount of information that should be necessary to supply to the network in the case of using for the learning all the possible inputs $x \in X$.

On the contrary, if in the learning phase a set of only m « $2^{n_{in}}$ inputs were used, the amount of information supplied to the network should be:

$$I_{net}(m) = m \,.\, n_{out} \log_2(l/q) = m \,.\, n_{out} \quad \text{bit} \tag{20}$$

Actually the true amount of information supplied to the network is greater than the quantity computed by (20), because it is assumed that the used set is *an appropriate set*, i.e. its elements are supposed to have been chosen in such a way to put into evidence as much as possible the rule to be learned by the network. So, e.g., if we have to implement an algorithm for separating two clases of elements, one should not choose a training set containing only elements of one class. The evaluation of such a supplementary information I_c is still a matter of investigation. Nevertheless we may suppose that its relevance diminishes as m increases and we will neglect it in the following.

If the network is able to perform a *complete generalization* after having been trained by means of m inputs, we may state that:

(a) the behavior of the network under consideration is the same as that of a network having an amount of information $I_{net}(P^*)$;

(b) the amount of information actually supplied to the network is $i_{net}(m) + I_c$.

As pointed out in the Introduction, in our case we are not dealing with an *amorphous* amount of information, like that stored in the memory of a digital computer or traveling in a communication channel. Then it becomes evident that the organized structure of the network plays an essential role in the total balance of information. In the trivial example by (2) we showed that different results are obtained if number 2 is next to the number 10 as an addend or as an exponent: the numbers are the same, but their *relation* is different. Analogously we claim that the architecture of the neurocomputer introduces *relations between* the different items of information supplied to it and that also this fact turns out to be an amount of information. Moreover we are now able to quantify it and give the following

DEFINITION 2: *We call* "relational information" *stored in a neurocomputer the following quantity:*

$$\Delta I_{net}(m) = I_{net}(P^*) - I_{net}(m) - I_c \cong n_{out} \,.\, (2^{n_{in}} - m) \tag{21}$$

As an example we may consider the simple case of classifying N elements in M classes. As

$$n_{in} = [log_2 N] \quad and \quad n_{out} = [log_2 M] \tag{22}$$

if we suppose that a complete generalization is obtained training the network with only the most significant element of each class, the relational information stored in the network is:

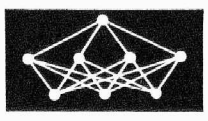
$$I_{net}(M) = [\log_2 M] \cdot (2^N - M) \tag{23}$$

The amount of sucha *relational information* can be not at all negligible. E.g. we consider the particular case of the experimental results obtained by Paternello and Carnevali [24]. They implemented the 8-bit sum by a network of 80 nodes and obtained a complete generalization with a training set of $m = 22$ inputs out of the 64,000 possible. Applying (21) we obtain:

$$\Delta I_{net}(224) = 8 \,.\, (2^{16} - 224) = 522496 \quad \text{bits} \tag{24}$$

4. Conclusions

We have examined the problems connected with information storage in neurocomputing putting into evidence (a) that many current ideas need to be revisited and probed further and (b) that the architecture of the neurocomputer is itself able to store information independently of the actual data fed to it. We have called this type of information "relational information" and have shown how it is possible to measure it by carrying out the difference between known quantities.



References

[1] D. M. Mac Kay, "The Nomenclature of Information Theory", *Proc. of First London Symp. on Inf. Theory*, 1950 — Reproduced in *Proc. 8th Conf on Cybernetics* (H. von Foerster, ed.), Josiah Macy Jr. Foundation, New York, 1951.

[2] D. M. Mac Kay, "The Place of 'Meaning' in the Theory of Information", *Information Theory* (E. C. Cherry, ed.), Butterworths, 215—255, 1956.

[3] H. Haken, "Information and Self-Organization: a Macroscopic Approach to Complex Systems", *Springer Verlag*, Berlin, Heidelberg, 1988.

[4] V. Cimagalli, M. Giona, G. Basti, A. Perrone, E. Pasero, "An Asymmetric Spin-Class Model of Long-Term Memory in a Dynamic Network Architecture", *Proc. IEEE-INNS Int. Joint Conf. on Neural Networks*, Washington D. C., Jan. 15—19, 1990.

[5] G. Basti. A. Perrone, A. Ballarin, V. Cimagalli, G. Morgavi, "Second-Order Statistic in a Non-Stationary Neural Network", *to be published.*

[6] V. Cimagalli, "A Neural Network Architecture for Detecting Moving Objects", in *Parallel Architecture and Neural Networks* (E. R. Caianiello ed.), World Scientific, Singapore, 225—230, 1990

[7] Atlan, "Rôle positif du Bruit en Théorie de l'Information Appliquée à une Définition de l'Organisation Biologique", *Ann. Phis. Biol. e Med.* 1, pp. 15—33, 1970

[8] G. Parisi, "Asymmetric Neural Networks and the Process of Learning", *J. Phys. A*, pp. L675—L680, 1986.

[9] I. Tsuda, E. Koerner, H. Shimizu, "Memory Dynamics in Asynchronous Neural Networks", *Progr. of Th. Phys.*, pp. 51—71, 1987.

[10] H. Haken, "Information Compression in Biological Systems", *Biol. Cybern.*, pp. 11—17, 1987.

[11] A. Kuh and B. W. Dickinson, "Information Capacity of Associative Memories", *IEEE Trans. on Inf. Theory*, pp. 59—68, 1989.

[12] R. J. Mc Eliece et al, "The Capacity of the Hopfield Associative Memory", *IEEE Trans. on Inf. Theory*, pp. 461—82, 1987.

[13] S. S. Venkatesh and D. Psaltis, "Linear and Logaritmic Capaticies in Associative Neural Networks", *IEEE Trans. on Inf. Theory*, pp. 558—568, 1989.

[14] G. J. Mitchinson and R. M. Durbin, "Bounds on the Learning Capacity of Some Multi-Layer Networks", *Biol. Cybern.*, pp. 345—356.

[15] S. Tan, J. Hao, J. Vandewalle, "Cellular Neural Networks as a Model of Associative Memories", *Proc. Int. Workshop on Cellular Neural Networks and their Applications*, Budapest, pp. 26—35, 1990.

[16] G. Basti, A. Perrone, V. Cimagalli, M. Giona, E. Pasero, G. Morgavi, "Informational versus Bifurcative Use of Chaotic Dynamics in Neural Networks", *Proc. Int. Neural Network Conf.*, Paris, pp. 941—944, 1990.

[17] M. Balsi, „Informational Properties of Chaotic Dynamics", (in italian), *EE. Dr. Dissertation*, Univ. of Roma "La Sapienza", 1991.

[18] A. De Carolis, "Information in Neural Networks. A Contribution to a Systematic Approach", (in italian), *EE. Dr. Dissertation*, Univ. of Roma "la Sapienza", 1990.

[19] K. Matsumoto, I. Tsuda: "Extended Information in One-Dimensional Maps" — Physica 26D, 347—357, 1987.

[20] R. Shaw: "Strange Attractors, Chaotic Behavior and Information Flow" — *Zeitschrift für Naturforschung* 36a, 80—112, 1980.

[21] M. Giona, F. Lentini, V. Cimagalli: „Functional Reconstruction of Chaotic Time Series" — *submitted to Phys. Rev. A*

[22] K. W. Cattermole: "Principles of Pulse Coding Modulation" — Iliffe, London (1969).

[23] MacCulloch and Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bull. Math. Biophys., pp. 115—133, 1949.*

[24] Patarnello and Carnevali, "Learning Networks of Neurons with Boolean Logic", *Europhys. Letters*, pp. 503—508, 1987.

Literature Survey

In this section of our Journal we continue in presenting of a survey on the selection of the last records concerning the neuroscience and the related fields which appeared in the Scientific Information System of the Institute of Computer and Information Science of the Czechoslovak Academy of Sciences,Prague.

We shell be grateful to the readers to inform the Editors or the Institute about any publication, which they recommend to insert in this literature survey.

Hopfield J. J.: The Effectiveness of Analogue "Neural Network" Hardware
NETWORK, Vol. 1, 1990, No. 1, pp. 27—40

Key words: models of nets; technical electronic realization.

Abstract: One always encounters imprecision in decision making. The facts that we base our decisions on may be fuzzy. The rules that we use to do logic inference are not precise. Even the words "AND" and "OR" used in describing the rules do not have precise meaning. In expert system that are built to help in making decisions, various ways are used to reflect this imprecis. A commonly used method is the certainty factor formalism. This has three disadvantages: 1/ The certainty factors of the "AND" and "OR" operat. are taken to be the maximum and minim. of the certainty factors of premises respectively. This means that that some informat. is completely ignored in the process. 2/ The uncertainties in the facts, rules and logical operat. are not treated in a unified way. In this paper, we define a new logic that weighs all available information and implement it using an emulated neural networks. This not only removes the difficulties stated above, but also allows the resulting expert system to be able to learn through examples.

Hsu L.-S., Teh, H.-H., Chan S.-C., Loe K. F.: Imprecise Reasoning Using Neural Networks
IEEE, 1990, pp. 363-368

Key words: neural network; expert system; certainty factor.

Huang J. N., Vlontzos J. A., Kung S. Y.: A systolic neural network architecture for hidden Markov's models
IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 32, 1990, No. 12, pp. 1967—1979

Key words: models of nets.

BINARY NEURAL NET: A LOGICAL NETWORK MODELING SOME FEATURES OF NEURAL NETS

M. Jiřina)*

Summary

The paper deals with possibility of using the principle of ordinary digital logical elements for design of a model of neural net. The two-layer structure of AND and OR logical gates similar to minimal disjoint form is introduced. Active as well as adaptive dynamics is described and it is shown that the net can serve as adaptive classifier, decoder and it can recognize „blurred" patterns, but it is not noise resistant.

1. Introduction

When classifying the neural nets two kinds of nets are usually considered [1]:

— the „continuous" or „analog" nets. In these nets the input and output signals may be of arbitrary value from some interval of real numbers. The weights of synapses are in essence arbitrary real numbers,

— „binary" nets having the input and output signals of two values only, preferably 0 and 1, or −1 and +1. The weights may be, as in the preceding case, in fact arbitrary real numbers. The function of the neuron in the net of this kind is based in evaluation whether the weighted sum of input signals is greater than some threshold, or not. According to this the output becomes in one of two possible states [5].

In this contribution a special case of binary neural net is introduced. Not only inputs and outputs, but also the weights may be in two or three states only. The weights have values −1, 0 or 1, the inputs have values 0 or 1. Two possibilities in setting the threshold are used. In the first case the threshold is set so low, that a single positive or inverted negative input signal is sufficient to overcome it. Than the overall function of neuron is logical sum of direct (the weight +1) or inverted (the weight −1) inputs. In the second case the threshold is set so high, that all positive and inverted negative inputs are necessary to overcome it. Thus logical product of direct or inverted inputs is accomplished. This kind of network represents a simplest purely digital kind of neural net. It is interesting from the point of view of realization by digital technology.

The neuron with binary inputs and outputs is in no relation to standard logical networks. It may be modeled as an analog resistive network, where the conductances of resistors represent the synaptic weights [3]. The neuron is then a threshold element and generates the threshold function

$$f(x_1, \ldots x_n) = 1 \quad when \quad \sum_{i=1}^{n} (2v_i - 1)\, x_i > \vartheta,$$

$$= 0 \quad (\text{or} -1) \quad \text{otherwise.}$$

In this formula $x_1, \ldots x_n$ are the inputs, $v_1, \ldots v_n$ are the corresponding weights and ϑ is a threshold.

Let a parallel n-bit input signal, a pattern, be given. To detect it a single n-inputs logical AND gate is sufficient. The 1's in the pattern are directly connected to the inputs of the AND gate, the 0's in the pattern are connected via inverters. If only $(n-1)$ — inputs AND gate is used then one of signal bits (the i—th) remains unconnected. The gate then detects two signals — patterns — as equivalent, just those having i—th input 0 or 1. From lessening of number of bits connected to the AND gate follows a broader class of signals detected as equivalent.

2. Theory of AND-OR neural net

2.1. Synapses of the AND-OR neuron

Definition 1. The **pattern** (or n—inputs pattern) is a vector of dimension n, having elements 0 or 1.

Definition 2. If the net generates some output combination in response to a given pattern, it generates a feature of the pattern. The disjunction (or logical sum) of all undistinguishable patterns which generate the same feature form the **maximal pattern** M_v of this feature. The conjunction (or logical product) of all undistinguishable patterns which generate the same feature form the **minimal pattern** mv of this feature.

Definition 3. The information is fed to the neuron by **synapse**. Each neuron has arbitrary but fixed number of synapses. Each synapse is any time in one of three states:

1. direct (noninverting),
2. inverting,
3. disconnected (abolished, dead).

We denote the direct and inverting synapses as **active**.

Definition 4. The **stimulus** of the synapse is a binary signal given by one element (bit) of the pattern.

*) Ing. Marcel Jiřina, DrSc., Institute of Computer and Information Science, Prague, Czechoslovakia.

To the direct synapse corresponds the weight $w = +1$, to the inverting synapse the weight $w = -1$. (More exactly: the inverting synapse for stimulus equal to 1 gives signal 0 to neuron's body (soma), for stimulus 0 it transfers signal 1.) The disconnected synapse corresponds to the weight 0. The disconnected synapse transfers no signal, but it doesn't matter to the function. The neuron with disconnected synapse behaves as the neuron having one synapse less. For simplification of formal descriptions the states of synapses are denoted simply by weights 1, 0, -1.

2.2. AND-OR neural net

The AND neuron as well as OR neuron are arranged as usual neurons are. We make difference between the stimulus and the input to the body of neuron:

Definition 5. The **stimulus** of the neuron or of the net is one pattern, whose individual elements form (as binary signals) the stimuli of individual synapses.

In the net one element of the pattern is an input of several synapses belonging to different neurons.

Definition 6. The **input to the body** of neuron is a binary signal coming from the output of active synapse (direct or inverting) to the body of neuron.

Definition 7. The **AND** neuron generates the feature just when all inputs to the body of neuron from active synapses are equal to 1.

Let us denote stimuli of direct synapses by $x_i, \ldots x_k$ and stimuli of inverting synapses by $y_i, \ldots y_l$. Then the AND neuron generates the logical function as follows

$$F = x_1 x_2 \ldots x_k \bar{y}_1 \bar{y}_2 \ldots \bar{y}_l; \qquad (1)$$

The body of the neuron then generates the logical product.

Definition 8. The **OR** neuron generates the feature just when at least one input to the body of neuron from active synapses is equal to 1.

Using the same notation as above, the OR neuron generates the logical function

$$G = x_1 + x_2 + \ldots + x_k + y_1 + y_2 + \ldots y_l. \qquad (2)$$

Definition 9. AND-OR neural net is a two-layer net with hidden layer formed by AND neurons and output layer formed by OR neurons.

Usually we shall consider AND-OR neural net with simple OR neurons (i. e. neurons which have synaptic weights 0 or 1 only) in the output layer. Formally, i. e. without considering the kind of the neurons and kind of synapses, the structure of the AND-OR net is the same as of two layer perceptron.

The net of n neurons in hidden layer can be learned n different patterns in the sense of metrics according to next chapter.

Let two features P_1, P_2 be given. Let maximal and minimal patterns M_1, M_2, m_1, m_2 correspond to these features. If for some pattern v it holds

$$(v \subset (M_1 \cap M_2)) \wedge ((m_i \cup m_2) \subset v), \qquad (3)$$

then the net generates the feature $P(v) = P_1 \cup P_2$. It can be useful property (the net generates more features than it was learned and therefore it distinguishes more classes of patterns, than it was learned). It may be an unacceptable property as well — to the pattern v the feature P_1 corresponds correctly, and P_2 is a surplus. In this second case the condition

$$M_i \cap M_j \cap (m_i \cup m_j) = 0 \qquad A_i, j = 1, 2, \ldots n,$$

where n is number of neurons in hidden layer, is the condition for good behavior. The output will never be a conjunction of features to which the net was learned.

2.3. Metrics of the AND-OR net

Let the i-th neuron generate a feature for r patterns $v_{i1}, \ldots v_{ir}$. Then the state of the neuron i for some pattern ω is

$$s_i(v) = \left(v \subset \bigcup_{j=1}^{r_i} v_{ij} \right) \wedge \left(\bigcup_{j=1}^{r_i} v_{ij} \subset v \right), \qquad (4)$$

or

$$s_i(v) = (v \subset M_i) \wedge (m_i \subset v), \qquad (4a)$$

where v is the stimulus, M_i is the maximal pattern and m_i is the minimal pattern.

If the k-th output neuron is a simple OR neuron, then its state is

$$S_k(v) = \bigcup_{i=1}^{r_i} w_{ki} \cdot s_i(v), \qquad (5)$$

where w_{ki} is the weight (0 or 1) of i-th synapse and s_i is the state of i-th neuron of the hidden layer. The AND-OR net generates then the feature

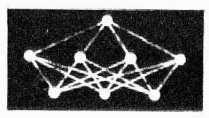
$$P(v) = (S_1(v), S_2(v), \ldots S_n(v)),$$

where for $S_k(N)$, $k = 1, 2, \ldots n$ it holds (5) and for $s_i(N)$ it holds (4).

Next let us consider a nontrivial net, i. e. the net, where for each neuron in hidden layer there exists at least one pattern, for which it generates the feature. Let us choose in (5) such a numbering of neurons in hidden layer (for each output neuron different), that for first n_1 neurons the weights w_{kj} are equal to 1 and the others are zero. Then

$$S_k(v) = \bigcup_{i=1}^{n_1} s_i(v) \qquad (6)$$

and then



$$S_k(v) = \bigcup_{i=1}^{n_i} [(\subset M_i) \wedge (m_i \subset v)].$$

Nonempty feature $P(v)$ is generated, when

$$p = \bigcup_{k=1}^{n} S_k(v) = 1, \qquad i.e$$

$$\bigcup_{k=1}^{r} \bigcup_{i=1(k)}^{n_i(k)} [v \subset M_i) \wedge (m_i \subset v)] = 1, \qquad (7)$$

where the symbol (k) denotes the numbering valid for k-th output neuron. It is easily seen that generally nothing guaranties the fulfillment of the eq. (7). Then it can exist a pattern v_0 giving the empty feature, i. e. the outputs of all output neurons are zero.

The **metrics** in the space of patterns generated by AND-OR net is then

$$\rho(v_i, v_j) = \begin{cases} 0 & \text{when } P(v_i) = P(v_j), \\ 1 & \text{otherwise.} \end{cases}$$

From it follows, that $\rho(v_{0i}, v_{0j}) = 0$ for all patterns v_0, for which the net generates the empty feature. The condition for distinguishing of the patterns v_i a v_j is clearly

$$\rho(v_i, v_j) = 1,$$

In practice it means, that the pattern v_i by at least one stimulus (bit) does not go in the M_i, or that in the v_j does not go the m_i. M_i and m_i are the maximal and minimal patterns for the feature $P(v_i)$.

2.4. Adaptive dynamics of AND or OR neuron

Each synapse may be in one of three states. The adaptive dynamics of AND-OR net is given by the conditions under which the states of individual synapses are changed. It may depend on initial states of the synapses and on the sequence the states can follow each other. Beside this, it is also important, how the active dynamics of individual neurons is controlled in the whole net.

From this point of view in the ideal net the active dynamics of each neuron follows from states of synapses, stimuli of the synapses, the demand to the state of the neuron, and globally given (to all neurons) commands for learning, initializing etc. The realization of such a net is in principle simplest, because only the neurons directly connected each to other are influenced. For hardware realization it is more complicated, when the learning is influenced by momentarily valid state of other neurons in the net. It means to build more connections than it is given by basic interconnection between layers.

From the point of view of active dynamics influenced by the hardware realization we then consider three kinds of nets:

1. The nets with fixed weights. In AND-OR net it means fixed connected logical network,

2. externally learned nets, where the weights (states) of synapses are controlled by external signals according to algorithm residing outside the net. For AND-OR net it means to set externally each synapse into one of three states. The memorizing of the state can be made in the net as well as outside of it. The reading of just valid state of neurons for informing the adaptive algorithm is not excluded.

3. The learning (self learning) nets, which have build in means for changing the weights like inherent component of the synapses or neurons. We can distinguish two main kinds:

a) The control from outside is limited to the change adaptive dynamics — active dynamics and setting the net into the initial state,

b) the control from outside influences in addition to ad a) the individual layers, or individual neurons, even individual synapses. Note, that by deepening of the control and lessening the independence of individual neurons we can get an externally controlled net.

In this part we will concentrate to the nets of the second and third kind. Several kinds of self learning AND-OR nets will be designed and their behavior will be shown, including a net without additional relations among neurons, but using randomly generated initial states of synapses.

2.4.1. The control of the learning process of one synapse

Let us suppose, that each synapse is controllable by two signals. One of these two signals U controls, whether the synapse learns or not — the signal „learning". By the other signal N it is possible to set the synapse into an initial state, independently on the preceding state — signal „initialize". The signals U and N are usually common to all synapses of the neuron. These signals are two extra inputs of the neuron.

A realization is possible having three controlling signals as follows:

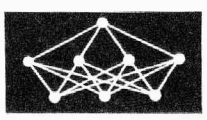
— initializing N,

— primary learning P (as separate signal),

— additional learning U (in preceding case only „learning").

Note, here we distinguish two kinds of learning. The primary learning is the first learning of the neuron since it was set to the initial state. The additional learning means, that the neuron is relearned to other pattern usually without forgetting the pattern it „knows" already. The additional learning arises when there is no neuron in the initial state or in other special circumstances.

Let $W = (w_1, w_2, \ldots w_n)$ be the vector of synaptic weights of a neuron. Each weight w_i, $i = 1, 2, \ldots n$ has one of values 1, -1 or 0. Let the formula



$$p_i = w_i * x_i,$$

where x_i is a stimulus of a synapse (two-valued logical variable) and p_i is the input to neuron's body, have the meaning given by a table:

$x_i =$	0	1
1	0	1
$w_i = -1$	1	0
0	#	#

In the table # denotes such an input to neuron's body, that it does not influence its function (disconnected synapse). In the case of AND neuron it is 1, in the case of OR neuron it is 0. It is appropriate to introduce also the notation

$$P = W*x,$$

where $P = (p_1, p_2, \ldots p_n)$ is the vector of inputs to body of the neuron.

From the point of view of adaptive dynamics we shall consider two kinds of learning:

A) primary learning, i. e. such a learning, that the synapses is set to one of two active states, direct or inverting. The primary learning arises as the first learning after the initializing signal N,

B) additional learning, i. e. the learning, when either

a/ the state of synapse is the same as the requested state, or the synapse is in the disconnected state. In this case the state of the synapse is not changed, or

b/ the state of synapse is opposed to the requested state. Then the synapse changes its state to a disconnected state.

The manner of control of the learning may be then as follows:

a) the primary learning of a neuron may arise only once and it is its first learning after the initial state. Any other learning is an additional learning,

b) the primary learning may arise under defined condition not only as the first learning after the initial state.

2.4.2. Adaptive dynamics of the AND neuron

Let to the synapses of the AND neuron considered be applied the pattern. For this pattern the neuron generates a feature.

Definition 10. Primary learning: Each synapse is set to direct state, if its stimulus i is equal to 1. The synapse is set to inverting state, if its stimulus is equal to 0.

It is possible to write the primary learning in form:

$$w_i = 2x_i - 1,$$

where the multiplication and subtraction are operations of real number arithmetics. We write also

$$W = 2x - 1.$$

Definition 11. The **additional learning**: To the neuron is applied a new pattern. Each synapse, which is in direct state and its stimulus is 0, or which is in inverting state and its stimulus is 1 is set to disconnected state. Otherwise nothing is changed.

The additional learning is possible to write in form

$$w_i' = (2x_i - 1 + w_i)/2,$$

where w_i denotes the preceding state of the synapse and w_i', its new state. We write also

$$W' = (2x - 1 + W)/2.$$

2.4.3. Adaptive dynamics of a single layer of AND neurons

Let us consider the net according to chap. 2.2. When learning we apply a pattern to the inputs of this net. A feature requested P is applied to the output. The individual neurons can be divided according to this feature P to those, which should generate the feature (and therefore they should have the output equal to 1, i. e. $p_i = 1$), and the others, which should have the output 0. The neurons of the second group mentioned need not any learning. The synapses of the neurons of the first group have to be set in a proper way. If controlling the i-th neuron by two signals N_i and U_i only, the signals N_i may be common to all neurons and synapses. The net then has a single initializing signal N. The signals of learning U_i must be conditioned by the value of p_i. If the control signal of learning for the net is U, then $U_i = U.p_i$.

In the system of two control signals the primary learning is the first learning following the initializing signal. Due to this fact the primary learning arise for neuron having $p_{i1} = 1$ only. The index 1 denotes the learning to the feature. If learning the another pattern with another feature, the primary learning arises for the neurons having $p_{i2} = 1$ and if it was $p_{i1} = 0$. For neurons having $p_{i2} = 1$ and $p_{i1} = 1$ the additional learning arises. The neurons having $p_{i2} = 0$ and also $p_{i1} = 0$ remain in the initial state. For these neurons the primary learning may arise later.

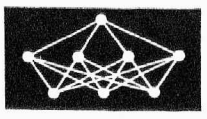
3. Adaptive dynamics of the AND-OR net

First, we introduce an important notion.

Definition 12. Let the output of the hidden layer neuron be connected to active synapses of simple output neurons. Let all these output neurons should generate a feature (i. e. their outputs should be 1). Then this hidden neuron has **uniform demand to the output.**

3.1. The simplest model

Let us consider an AND-OR net, where the control signals of the hidden layer are common. The hidden



layer is then controlled by the signals N_s, U_s, or N_s, P_s, U_s. Similarly, the output layer is controlled by signals N_v, U_v, or N_v, U_v, P_v.

During learning a pattern x is applied to the inputs and feature requested p to the output. It is sufficient to take into our consideration the fact of undistinguisability of the neurons in the hidden layer as well as in the output layer (and impossibility to control them individually). From it follows impossibility to state, which neuron of the hidden layer should generate the feature and which not. It is only possible to learn all neurons the same or nothing. From it an important conclusion follows, that it is necessary to „choose" a neuron and only this neuron is learned. The main problem of the adaptive dynamics of the AND-OR net becomes a **selection** of the neuron to be learned, i. e. establishment of the rules for this selection and algorithmic and hardware realization of these rules.

3.2. AND-OR net with random initial states of synapses

In the same way as in the preceding chapter we assume, that all neurons in the hidden layer are of the same kind and similarly connected. The same is true for output layer. In the same way as in chap. 2.4.3 the signals N_i let be common. The net has then a single initializing signal N. The learning signals U_i must be conditioned by the value of the feature to be generated by the i-th neuron. Then if the command signal of learning of the hidden layer is U_s, it is $U_i = U_s . p_i$. The control signals for output layer are common and the output layer is controlled by signals N and U_v. The signals P_s and P_v we do not consider because it will be seen, that the primary learning does not happen for all neurons at the same time.

The adaptive dynamics algorithm:

Initial state: In the beginning the states of synapses of hidden layer neurons are random: disconnected, direct or inverting. The neuron of the output layer has the synapses in disconnected state. (We proceed from the fact the synapses of the hidden layer are already set in some random way).

1) Apply a pattern to the inputs of the net. The active dynamics of the hidden layer takes place. The states of hidden layer neurons are set.

2) Learning of the output layer: If the output neuron should generate a feature (its output should be 1) and its synapse is connected to the hidden layer neuron generating a feature, the synapse is set to the direct state (+1).

3) Learning of hidden neurons — there are two possibilities:

3a) If the neuron generates a feature, then the primary learning arises.

3b) If the neuron does not generate a feature (its output is 0) and it has the uniform demand to the output, it is additionally learned.

4) end.

For output layer there is no difference between primary learning and the additional learning. The learning begins from the state of all synapses disconnected. If any synapse becomes direct, it remains in such a state forever. It results in following behavior of the net: If for the same pattern a feature is given and later an another one, the net then generates a feature which is the disjunction of both of these features.

If the i-th hidden neuron was learned different patterns $v_{i1}, \ldots v_{ip}$ it generates then a feature to each pattern for which the formula (4) or (4a) holds. If this neuron is learned some pattern for which (4) or (4a) holds again, a primary learning arises. Since then the neuron „knows" the latter learned pattern only. Then if a neuron is learned something it „knows", it forgets all other what it was learned formerly.

The same way behaves the net: When it is learned a double pattern + feature it „knows" already, it forgets all other patterns corresponding to the same feature.

The primary learning is necessary, because the additional learning only disconnects the synapses to which contradictory requests are applied. On the other hand there is no mechanism for preventing to the accidental repeating of primary learning. It is possible to include this mechanism by modification of steps 3a) and 3b) so, that a condition the neuron was never primarily learned, is added. It means to check this fact, and it is a little complication.

3.3. AND-OR net with controlled order

We proceed from the net according to the chap. 3.1. The same way as in chap. 3.2 the net has a single initializing signal N.

The learning control signals of individual synapses in the output layer are conditioned by their order. In the hidden layer the learning control signals are conditioned by the order of neurons, which were not yet primarily learned.

Algorithm of adaptive dynamics:

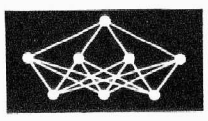
Initial state: The synapses of all neurons are in disconnected state.

1) Apply a pattern to the inputs of the net. The active dynamics of the hidden layer takes place, the states of hidden layer neurons are set.

2) Learning of the output layer: If the output neuron generates a feature, then it is found the first synapse connected to a hidden neuron which should generate the feature. This synapse is set to the direct state (+1).

3) The learning of hidden neurons:

3a) The first hidden neuron which was never lear-



ned since initialization is found. Then primary learning of this neuron takes place.

3b) Otherwise (i. e. all hidden neurons was already learned) and the neuron does not generate the feature and it has the uniform demand to the output, it is additionally learned.

4) End.

As the initial state of synapses of hidden layer is a disconnected state, all hidden neurons generate the feature (output signal is 1).

The output layer is additionally learned the same way as according to the algorithm in chap. 3.2.

The hidden layer is primarily learned only once. It is additionally learned any pattern, for which the corresponding feature is already known. Additional learning arises also if the feature applied includes as a subset any feature already known. Otherwise the feature is not known and the output layer learns nothing. It is possible to condition the learning of the output layer by the fact whether in the hidden layer arises a primary learning or not. Then it is possible to modify the algorithm as follows:

Algorithm of adaptive dynamics:

Initial state: The synapses of all neurons are in the disconnected state.

1) Apply a pattern to inputs of the net. The active dynamics of the hidden layer takes place, the states of hidden layer neurons are set.

2) It is looked for the first never learned hidden neuron. When it is found, its number is K and go to 3), otherwise go to 5).

3) Learning of the output layer: If the output neuron generates a feature, then its synapse No. K is set to the direct state (+1).

4) The K-th hidden neuron is primarily learned. End.

5) All hidden neurons having the uniform demand to the output, are additionally learned.

6) End.

There is impossible additionally learn the output layer, but otherwise the behavior of the net is the same as in the preceding case. In this algorithm a very simple deterministic condition for learning is used.

3.4. AND-OR net — model „find the first, which . . . "

The condition for searching the neuron which should be learned can be substituted by another condition, which uses no exactly given order. It is looked for the first neuron (in any order) of the hidden layer generating the feature. The first algorithm in the preceding chapter is changed so, that the step 3a) is a little modified as follows:

3a) find the first neuron of the hidden layer, **which generates the feature**. Then the primary learning arises.

In the same way the step 2) of the second algorithm from preceding chapter is changed:

2) Find the first neuron of the hidden layer, **which generates the feature**. If it is found, its number is N and go to 3), otherwise go to 5).

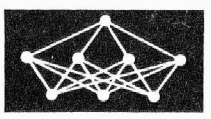
In contradiction to similarity of algorithms from this and the preceding chapter, the behavior of them differs heavily. It is due to the fact, that if the initial state of synapses of hidden neurons are disconnected (0), all hidden neurons generate the feature. Than all neurons of the output layer are primarily learned to the first pattern. During next learning only additional learning of output layer arises (the hidden layer learns in a proper way). To eliminate this fact, it is necessary to learn the net to pairs empty pattern — real feature at first. Then after excerpting of all hidden neurons to learn the net again with nonempty patterns. The order of learning is not essential.

4. Example of an AND-OR net and its function

We choose the net for transformation of different types of characters to reference characters. The characters are formed in a grid of 8x8 pixels, bits. The AND-OR net has 64 inputs, three hidden neurons and 64 output neurons. The net generates to the different forms of letters A, B, C the features (characters) E, F, G respectively. The net uses the algorithm from chap. 3.2.

In *Figs. 1 — 4* in the left upper corner there is the character applied, or the feature (character) as the response of the net. In the lower half in the left, there are the states of synapses of three hidden neurons. To it correspond in the right hand side three arrays of 8x8 points. The first array represents the synapses of the 64 output neurons connected to the first hidden neuron. The same for second and for the third set of synapses. The „." denotes a disconnected synapse, „*" the direct synapse, and „-" the inverting synapse.

In *Fig. 1* there is shown the state after learning the first pattern and the corresponding feature. The second hidden neuron and the synapses from it to the output neurons was learned. For the first and the third hidden neuron there is seen an initial state. In *Fig. 2* the state after learning of all hidden neurons is shown. In *Fig. 3* there is shown a state after additional learning of bold characters A, B, C. Now the thin letters A, B, C represent the minimal patterns and the bold ones the maximal patterns. These are samples of normal behavior when learning. The net in active state then gives the the feature (output) in form which it was learned for any form of letter A, B, or C which include the corresponding minimal pattern and is included in the corresponding maximal pattern. Now, let the net be learned the bold „A" again with the prescri-



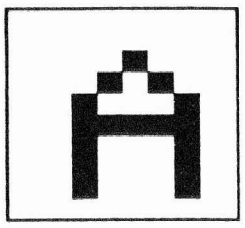
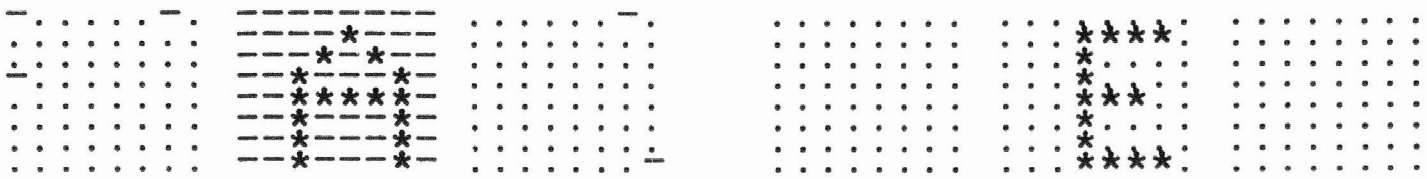


Fig. 1. The state after learning the first pattern and corresponding feature. In the first and the second hidden layer neuron the initial state of the net is seen.



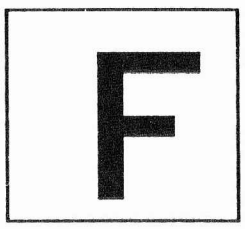


Fig. 2. The state after the primary learning of all hidden layer neurons.

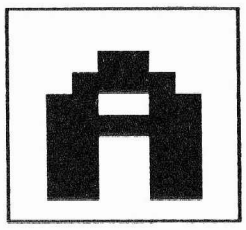
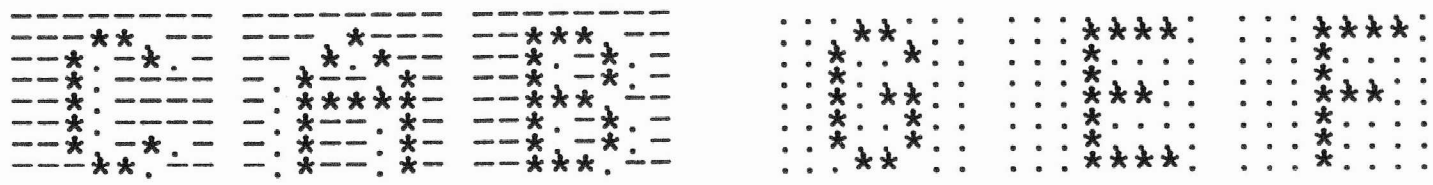


Fig. 3. The state after additional learning of „bold" characters A, B, C. Now the „thin" letters A, B, C represent the minimal patterns, the bold ones the maximal patterns.



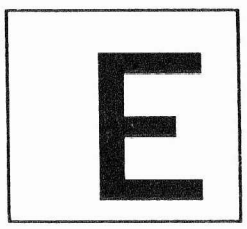
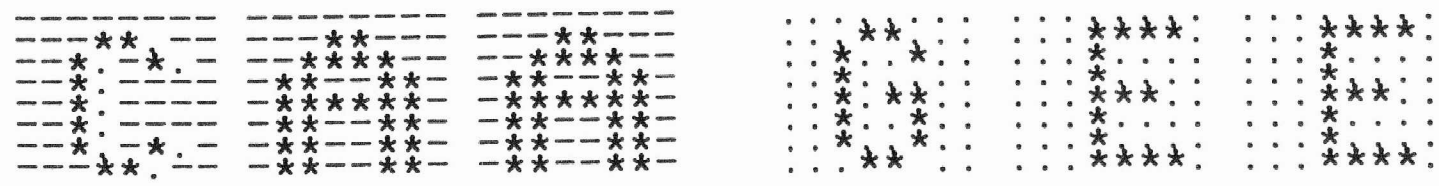
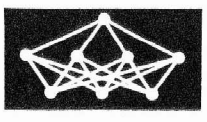


Fig. 4. The state after repeated learning of the pattern bold A with assigned feature E. The primary learning took place not of the second, but also of the third hidden neuron and „B" is completely forgotten.





bed feature „E". In *Fig. 4* there is seen that the learning has arisen not only for the second neuron, but also for the third hidden neuron. Both of these neurons „know" the bold „A" only. For the third neuron it happened, because the feature „F" is a part of the feature „E". The response of the net for thin „A" is empty feature now. The net is able to learn it additionally again, but „B" is forgotten forever. It is impossible to reach the state according to *Fig. 3* again.

6. Conclusions

It is easily seen, that the binary neuron in the active dynamics is a combinatorial logical network only. In the adaptive dynamics the memories of synaptic states are put into the effect. The net of binary neurons in more complicated adaptive state is in the end a finite automaton only. Despite of this an AND-OR net shows a nontrivial behavior.

The class of tasks processible by the AND-OR net is derived in essence from the classification to $n + 1$ classes, if it has been given n classes. For this it would suffice a single layer net. The second, output layer allows to get the response in a requested form. The AND-OR net can serve as an adaptive classifier, decoder, error correcting network and it can recognize „blurred" characters. The behavior of control structures according to chap. 4 was tested just for simple pattern recognition. From Chaps. 3.5 and 3.6 it is seen that the net is good for recognition or regeneration of „blurred" patterns, but not for patterns attacked by noise.

The possibilities of realization by VLSI technology, especially CMOS, are easily seen. A little problem remains, how complicated the net is in fact. Let us consider a system for regeneration of up to 128 kinds of „blurred" patterns in an array of 8x8 pixels. The AND-OR net has 64 inputs, 128 hidden neurons with 8292 synapses and 64 output neurons with 8292 synapses. Let approximately 1/3 of synapses of hidden neurons be disconnected, 1/3 inverting and 1/3 direct. For output neurons let us suppose 1/2 disconnected and 1/2 direct synapses.

The fixed learned net then represents 128 approx. 45 input AND logical gates, 64 approx. 35 input OR gates and approx. 2800 (1/3 of 8292) inverters. This is complicity of order of 10 000 transistors.

The fully adaptive net with minimal external control is approx. 50 times more complicated. It is necessary to build all synapses, not only the inverting ones, and the synapses of the output layer. In all of the synapses of hidden layer neurons must be a three state memory, in all of the synapses of output layer neurons must be two-state (one bit) memory.

The speed is rather high. The fixed net represents at most three logical steps in series. The fully adaptive net represents approx. ten times more logical steps in series, but in form of internal gates. The learning of this net needs two clock intervals needed for setting of the internal memories for each pattern. Using minimal and maximal patterns for additional learning of the net it is necessary to apply two times more patterns then there are features.

References

[1] Lippmann R.P.: An Introduction to Computing with Neural Nets. IEEE ASSP Magazine April 1987, pp. 4 — 22.

[2] Kung S.Y.— Hwang J.N.: Parallel Architectures for Artificial Neural Nets. Proc. Conf. San Diego 1988, pp. II-165 — II-172.

[3] US patents No. 4802103, 4773024, 4737929, 4752906, 4660166.

[4] Foo, S. Y. — Anderson, L. R. — Takefuji, Y.: Analog Components for the VLSI of Neural Networks. Circuits and Devices July 1990, pp. 18 — 26.

[5] Yanai, H. — Sawada, Y.: Associative Memory Network Composed of Neurons with Hysteretic Property. Neural Networks Vol. 3, 1990, pp. 223 — 228.

Literature Surwey

Heileman G. L., Papadourakis G. M., Georgiopoulos M.: A Neural Net Associative Memory for Real-Time Applications

Neural Computation, Vol. 2, 1990, No. 1, pp. 107—115

Abstract: A parallel hardware implementation of the associative memory neural network introduced by Hopfield is descibed. The design utilizes the Geometric Arithmetic Parallel Processor (GAPP), a commercially available single-chip VLSI general-purpose array processor consisting of 72 processing elements. The ability to cascade these chips allows large arrays of processors to be easily constructed and used to implement the Hopfield network. The memory requirements and processing time of such arrays are analyzed based on the number of nodes in the network and the number of exemplar patterns. Compared with other digital implementations, this design yields significant improvements in runtime performance and offers the capability of using large neural network associative memories in real-time applications.

Newquist H. P. I.: The AI Eighties. (developments in artificial intelligence between 1975 and 1989) (In practice) (column)

AI-Expert, Vol. 5, 1990, No. 1, pp. 61—66

Key words: artificial intelligence; new technique; review of past year ; neural networks.

Abstract: A review of developments in artificial intelligence that took place during the years 1975—1989 is presented. AICorp was founded in 1975 and developed one of the first natural-language products; Nestor Inc was formed to create neural-network applications. Compagnies founded in 1979 included Advanced Information and Decision Systems and Cognitive Systems. Speech-recognition and knowledge-based systems began to be explored in the early 1980s. ACT Ltd introduced a portable voice-recognition system in 1984. Excalibur announced its Savvy Retriever query system in 1985. Borland International legetimized the PROLOG language with its introduction of Turbo Prolog in 1986. The Japanese entered the AI field in the mit-1980s. Artificial Intelligence went through a slump in 1987; Symbolics laid off 160 employees. Apple and Texas Instruments introduced a Mac with a LISP coprocessor in 1988.

KNOWLEDGE PROCESSING BY NEURAL NETWORKS

G. Vítková, J. Míček)*

Abstract:

The paper outlines the basic properties of a neural network associative memory extrapolation model and describes the main abilities of fuzzy cognitive maps. A revised equation for associative memory model performance is proposed. Modeling the logical function „and" for inferring knowledge by a neural network is discussed. The results of an investigation of an associative memory extrapolation based knowledge system used for diagnostics are presented. Outlined is the frame of the QUEST system for knowledge processing. The system functions and its other facilities are also presented.

Keywords:

Neural network. Associative memory models. Category formation. Single layer auto-associative memory. Associative memory extrapolation model. Nonprocedural knowledge processing. Unsupervised learning.

1. Introduction

In the past four years, neural network models were used to build knowledge — based systems [1,2,4,5]. Traditional knowledge-based systems face the problems connected with effective knowledge acquisition and representation, storage of experts' knowledge, and appropriate knowledge — handling facilities. Knowledge acquisition, i.e. moving domain knowledge into a software system by whatever means, extends over the complete lifetime of a knowledge-based system. Continuous expansion and modification of a system's knowledge base is driven by two factors: changing domain knowledge, and broadening the scope of application. The iterative and uncertain nature of knowledge only complicates the situation. Knowledge representation involves a study of how we can represent particular semantic notions such as time, causality, beliefs, intentions, and data consistency [13].

Neural networks offer one of many possibilities for coping with these problems and for confronting difficulties related to implementation and maintenance of knowledge-based systems. An inference mechanism in neural networks provides necessary knowledge-handling facilities. Adaptive or learning mechanisms in neural networks help to solve the knowledge acquisition problem and at the same time enables a knowledge base to improve throughout the whole of its lifetime.

The advantages of a neural network approach to knowledge base construction over other existing approaches are as follows:

- the knowledge base response time is independent of the number of knowledge base components, which enables its real-time behavior [5,7].
- the approach enables the gaining of knowledge by learning using measured data or data acquired in other ways even should the data be incomplete and inaccurate or inconsistent and noisy [1,2,5,12],
- the approach provides a possibility of combining learned knowledge and the knowledge of individual experts into representative knowledge bases [5,7].

Using the main properties of the neural network associative memory model [3,9,10,11,20] and fuzzy cognitive maps for representation of expert knowledge [5-8,21] the QUEST system was developed at the Institute of Computer Sciences of the Czechoslovak Academy of Sciences [15,17]. Its inference mechanism provides necessary knowledge-handling facilities . Its learning mechanism helps to solve a knowledge acquisition problem. The system consists of three basic parts: the first one checks the consistency of related concepts introduced by different experts, the second part enables the maintenance of a knowledge base, and the third one processes queries to a knowledge base. The QUEST knowledge base is represented by a weights matrix of an associative memory model. The system supports the creation of a knowledge base i) by experts, ii) by learning, and iii) by a composition of existing knowledge bases. As learning algorithms, the Hebbian law [9] and the pseudo-Hebbian law [9,11] are used. The system is able to process matrices of weights which contain both positive and negative values. An input vector (a query) is considered to involve only zeros and ones.

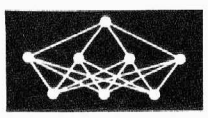
Below, after introducing the basic theoretical background, we present our approach to using a neural network associative memory model for knowledge processing.

2. Theoretical background

2.1. Memory extrapolation network

A neural Hopfield-like network capable of restoring continuous level library vectors from memory is introduced in [11]. The interconnections in this network are

*) Ing. Galina Vítková, CSc., Dr. Jiří Míček,
Institute of Computer and Information Science of the Czechoslovak Academy of Sciences, Prague



determined analogously to the Hopfield model, i.e. they are formed by a set of library or training vectors. The network can operate synchronously as well as asynchronously and is fault tolerant. It differs from the Hopfield model in that the initially known neural states are imposed on the network's last iteration. The nodes with the known states act as the network stimuli and the remaining nodes behave as the response. An analogy to human memory is our ability to recall complete information given only a portion of it.

Consider a neural network consisting of L nodes. The interconnection strength between the j-th and i-th node is denoted w_{ji}. It is assumed moreover that the network is symmetric (i.e. $w_{ji} = w_{ji}$). The state of activity of the j-th mode is a function of the sum of its inputs (without loss of generality, we assume that threshold $v_j = 0$), i.e.

$$\xi(t) = S\bar{x}(t)(1), \tag{1}$$

where $\xi(t)$ is the vector of L input sums at time t, W is the matrix of interconnection synaptic weights, $\bar{x}(t)$ is the vector of the L neural states at time t. Let S denote the node operator that determines the next set of states from the input sum:

$$\bar{x}(t+1) = S\xi(t). \tag{2}$$

Substituting Eq.(1) into Eq.(2) gives the state iteration equation :

$$\bar{x}(t+1) = SW\bar{x}(t). \tag{3}$$

Now consider a set I of N continuous level linearly independent training vectors of length $L \geq N$, $I = \{f_n \mid 1 \leq n \leq N\}$ and the corresponding matrix $F = [f_1 : f_2 : \ldots : f_n]$. Using training vectors, the following algorithm referred to as the pseudo-Hebbian learning law is used to compute W:

$$W = F(E^t F)^{-1} F^t. \tag{4}$$

Let us say more about creating the matrix of interconnections W [9,11]. Consider an auxiliary vector:

$$\bar{z} = \bar{x}(t+1) - W\bar{x}(t).$$

Then the pseudo-Hebbian learning law is expressed by the equation:

$$W = W + \bar{z} \cdot \bar{z}^T / |\bar{z}|, \tag{5}$$

$$W = F \cdot F^+, \quad F^+ = (F^T \cdot F)^{-1} F^T, \tag{6}$$

F^+ is a so-called pseudo-inversion of a matrix F.

Given a portion of one of the vectors which belongs to a certain object described by matrix F, a memory extrapolation network will reconstruct the remainder of that vector. Let us divide all L nodes into two sets : one, in which states are known, and the remainder, in which states are unknown. Without loss of generality, assume that states through the first $P < L$ nodes are known for a given application. Sometimes these nodes are called a key. It means that for $1 \leq k \leq P$ the node state is kept without change. The P known nodes (or key) thus act as the input or the stimulus to the network and the states of the remaining $Q = L - P$ nodes represent the output or the response.

In summary the algorithm describing the process is:
(1) Initialize all unknown states by setting to zero. The state of the remaining nodes are equated to the known portion of the input vector.
(2) Multiply the state vector by W.
(3) Replace states of the first P nodes with their known (input) values.
(4) Go to step 2 and repeat the whole process until the state of the Q nodes does not change.

In this case, an active process (or recall) may be described as follows:

$$\bar{x}(0) = \xi(0),$$

$$\bar{x}(1) = W\bar{x}(0), \quad \bar{x}(2) = K(x(1)),$$

$$x(3) = W\bar{x}(2), \quad \bar{x}(4) = \ldots,$$

where K is such a function that

$$K(\xi_1, \ldots, \xi_P, \xi_{P+1}, \ldots, \xi_L) = (\xi_1 \ldots, \xi_{P+1}, L).$$

Thus in general, the dynamics of the iteration process is given by the following equations:

$$\bar{x}(2t+1) = W\bar{x}(2t) \qquad t = 0,1, \ldots$$
$$\bar{x}(2t) = K(\bar{x}(2t+1)) \qquad t = 1,2, \ldots$$

Generally, a memory extrapolation network doesn't give the exact correct answer; it only gets iteratively closer and closer to the answer. Now let us make some short notes about convergence properties of the memory extrapolation network and the effects of the input uncertainty on the network's performance.

The problem is whether a net iteration $\bar{x}(t+1) = SW\bar{x}(t)$ will converge. A sufficient condition for unique convergence is that $P \geq N$ and the matrix $F_P(f_1 : \ldots : f_P)$ is full-rank (the proof is given in [11]). If $P < N$ there exists a continuum of solutions.

2.2. Fuzzy cognitive maps

Uncertain causal knowledge can be stored in fuzzy cognitive maps (FCM) [6-8]. A FCM is a fuzzy signed digraph with feedback which represents uncertain causal relationships. A simple FCM has a causal edge in {1,0,1}, i.e. shows a maximal degree of causality. In

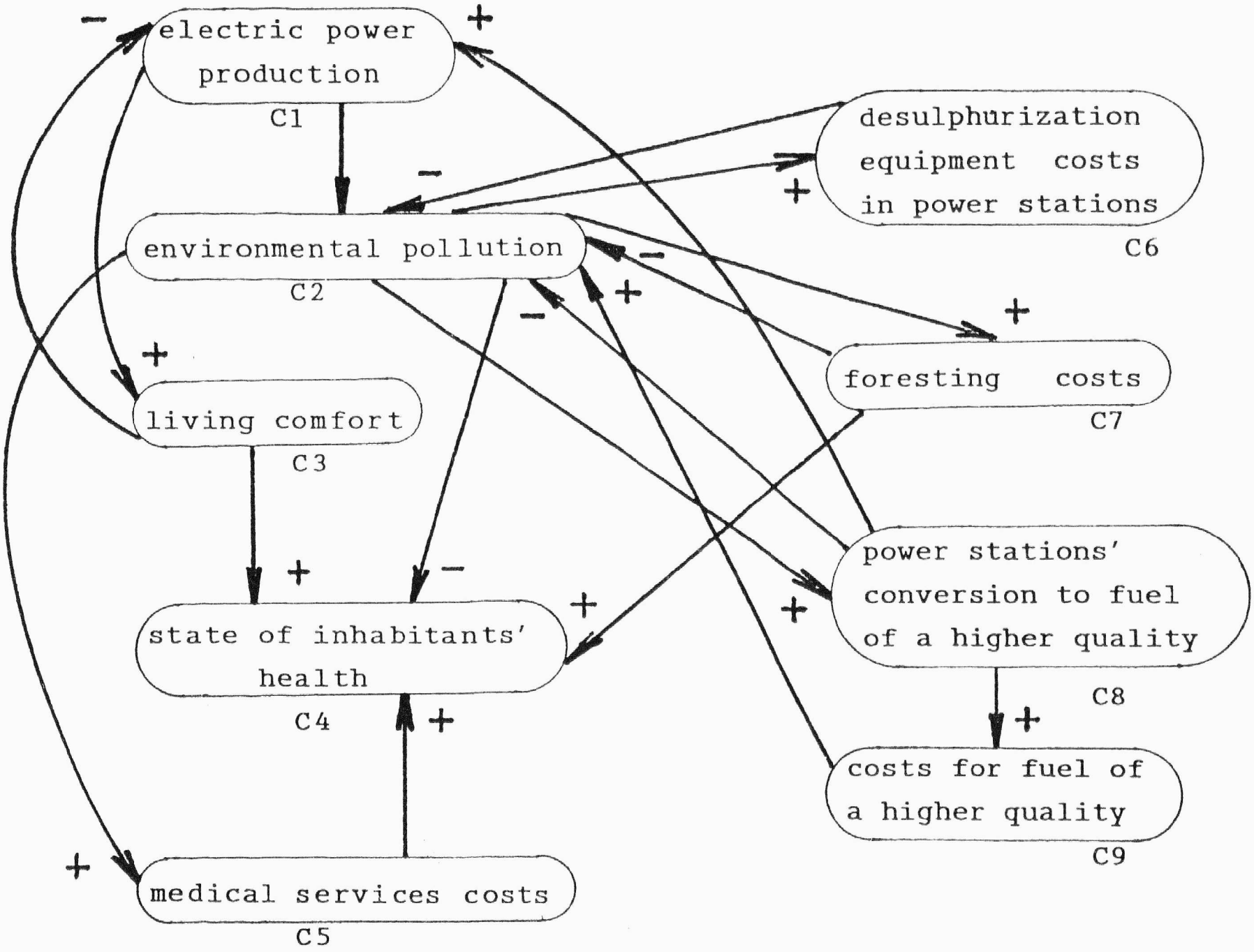


Fig. 1. The simple FCM expressing the 1st expert's opinion (FCM1)

general FCM causal edge weights are numbers in [-1,1], allowing one to express different degrees of causality. An example of a simple FCM is depicted in the graph below (see *Fig. 1*).

This non-fuzzy signed digraph with feedback is equivalent to the connection matrix in *Fig. 2*.

In the matrix, the i-th row expresses the connection strength of the edges w_{ij} directed out from C_i, and the i-th column describes the connection strength of the edges w_{ij} directed in towards C_i ($w_{ij} > 1$ means a positive causal relationship between C_i and C_j, $w_{ij} < 1$ means negative causality).

Simple FCM are easier to get from experts. They are usually more reliable because as a rule experts are more likely to agree on causal signs than on causal values. The simple FCM matrices drawn by individual experts can be combined into a non-simple FCM that represents causal relationships by a more representative scale. No restrictions on the number of experts or the number of concepts exist. The more experts, the more reliable the combined FCM; moreover each expert can be assigned a credibility weight of his opinion in [0,1]. Combined weighted FCMs reflect the different level of experts' knowledge. *Figures 2, 4, 5* illustrate a combination of two experts' opinions into one representative matrix.

An FCM matrix representation enables an FCM to be viewed as a feedback associative memory model that allows causal inferences to be processed in a feedback associative memory fashion. Similarly to a neural network, each causal node C_i is considered to be a nonlinear function which transforms the paths weighted activation into an output signal. Again similarly to neural network models, this function is in general a bounded monotone increasing transformation, such as the sigmoid or the S — shape function. The simplest nonlinear operation is thresholding, which in the case of a synchronous state-transition looks like:

$$C_i(t+i) = \begin{cases} 1 \text{ if } C(t)\, W^i > 0, \\ 0 \text{ otherwise} \end{cases}$$

where $C(t) = (C_1(t), \ldots, C_N(t))$ is the state vector of

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	1	1	0	0	0	0	0	0
C2	0	0	0	−1	1	1	1	1	0
C3	−1	0	0	1	0	0	0	0	0
C4	0	0	0	0	0	0	0	0	0
C5	0	0	0	1	0	0	0	0	0
C6	0	−1	0	0	0	0	0	0	0
C7	0	−1	0	0	0	0	0	0	0
C8	1	−1	0	0	0	0	0	0	1
C9	0	−1	0	0	0	0	0	0	0

Fig. 2. The connection matrix of FCM1

a causal activation at discrete time t (compare with the state vector of a neuron), and W^i is the i-th column of the FCM connection matrix W.

Causal flow in an FCM is easily maintained with a vector- matrix operation and thresholding. In general it is described by the expression $C(t+1) = T[C(t)\,W]$ where T is the vector threshold operation. For simple FCMs and most nonlinear FCMs, the causal flow will quickly stabilize to a limit cycle. In the case of simple FCMs, convergence is always achieved because thresholding is a deterministic operation, and every FCM converges after at most 2^n iterations [5]. But complex (non-simple) FCMs with time-varying edges can resonate on chaotic attractors.

The resonant limit cycle of an FCM is a hidden pattern in the causal edges W. The hidden patterns in an expert's FCM correspond to the expert's answers to What-if questions. As with an expert's answer, the resonant hidden pattern can be tested against the available facts and the appropriate FCM can be changed accordingly as needed.

Consider, with respect to the matrix in *Fig. 5*, the input vector V = (1 0 0 0 0 0 0 1 0 0 0 0 0 0 0). It is equivalent to the question „What happens if electric power production increases and reconstruction of power stations is realized". The inferring iteration process is described by sequences of states as follows:

$$
\begin{aligned}
&V = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^1 W = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^1 = (1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^2 W = (0\ -1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ -1\ 1\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^2 = (1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^3 W = (0\ -1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ -1\ 1\ 0\ 0\ 0) \longrightarrow\\
\longrightarrow\ &V^3 = (1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0)\ .
\end{aligned}
$$

The limit cycle, which indicates the end of iterations, is: C3 C4 C9 C12.

It means that in this case living comfort will improve, the state of inhabitants' health will not worsen , and costs for higher quality fuel will increase.

3. Development of a memory extrapolation model

As was mentioned, we have used the concept of an FCM (section 2.2) and the memory extrapolation model of a neural network (section 2.1) as the theoretical background for our approach to neural networks' application in the development of knowledge systems. Below we present and argue our contribution to this topic concerning *i*) modification of the equation for associative memory model performance (or recall) and for performing the logical function „and" by associative memory; *ii*) examination and analysis of neural network (more exactly its memory extrapolation model) behavior used as a diagnostic system.

3.1. Associative memory model performance

Consider without loss of generality a neural net-

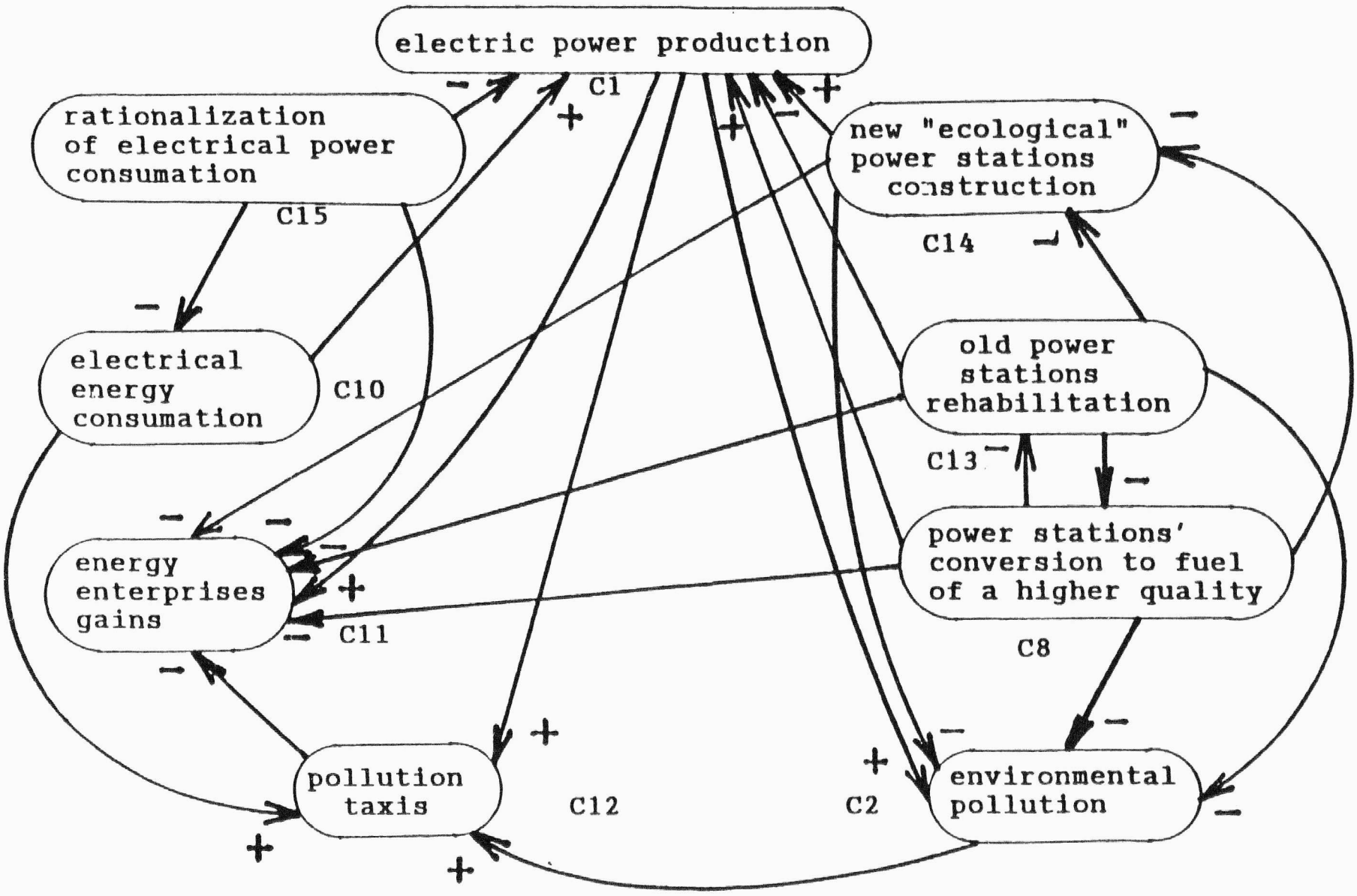


Fig. 3. The simple FCM depicting the 2 nd expert's opinion (FCM2)

	C1	C2	C8	C10	C11	C12	C13	C14	C15
C1	0	1	0	0	1	1	0	0	0
C2	0	0	0	0	0	1	0	0	0
C8	1	−1	0	0	−1	0	−1	−1	0
C10	1	0	0	0	0	1	0	0	0
C11	0	0	0	0	0	0	0	0	0
C12	0	0	0	0	1	0	0	0	0
C13	−1	−1	−1	0	−1	0	0	−1	0
C14	1	−1	0	0	−1	0	0	0	0
C15	−1	0	0	−1	−1	0	0	0	0

Fig. 4. The connection matrix of FMC2

work consisting of 6 nodes and its synaptic weights matrix. (See *Fig. 6, 7, 8.*)

Consider a binary input vector x' and transform it into bipartite form x according to the recommendation in [20], i.e.: $2x' - 1$

$$(1\ 0\ 0\ 0\ 1\ 0) \Longrightarrow (1\ -1\ -1\ -1\ 1\ -1).$$

Now if we use for performance evaluations the equation

$$x_i = S\left(\sum_{j=1}^{n} x_j w_{ij} - \upsilon_i\right),$$

(where x_i, x_j *denote a state of the* i-th and j-th neurons, w_{ij} is a synaptic strength (weight) connecting neuron i and j, υ_i is a ithreshold, $S(x) = 1$ if $x > 0$, and $S(x) = 0$ if $x = < 0$) we get the result vector (0 1 0 0 0 1). Testing this result against the evidence, we found that it is not adequate to the reality in some cases. Ba-

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C1
C1	0	2	1	0	0	0	0	0	0	0	1	1	0	0	0
C2	0	0	0	−1	1	1	1	1	0	0	0	1	0	0	0
C3	−1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C6	0	−1	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	0	−1	0	1	0	0	0	0	0	0	0	0	0	0	0
C8	1	−2	0	0	0	0	0	0	1	0	−1	0	0	0	0
C9	0	−1	0	0	0	0	0	0	0	0	0	0	0	0	0
C10	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
C11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C12	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	0
C13	−1	−1	0	0	0	0	0	−1	0	0	−1	0	0	−1	0
C14	1	−1	0	0	0	0	0	0	0	0	−1	0	0	0	0
C15	−1	0	0	0	0	0	0	0	0	−1	−1	0	0	0	0

Fig. 5. The combined matrix representing the opinion of both experts

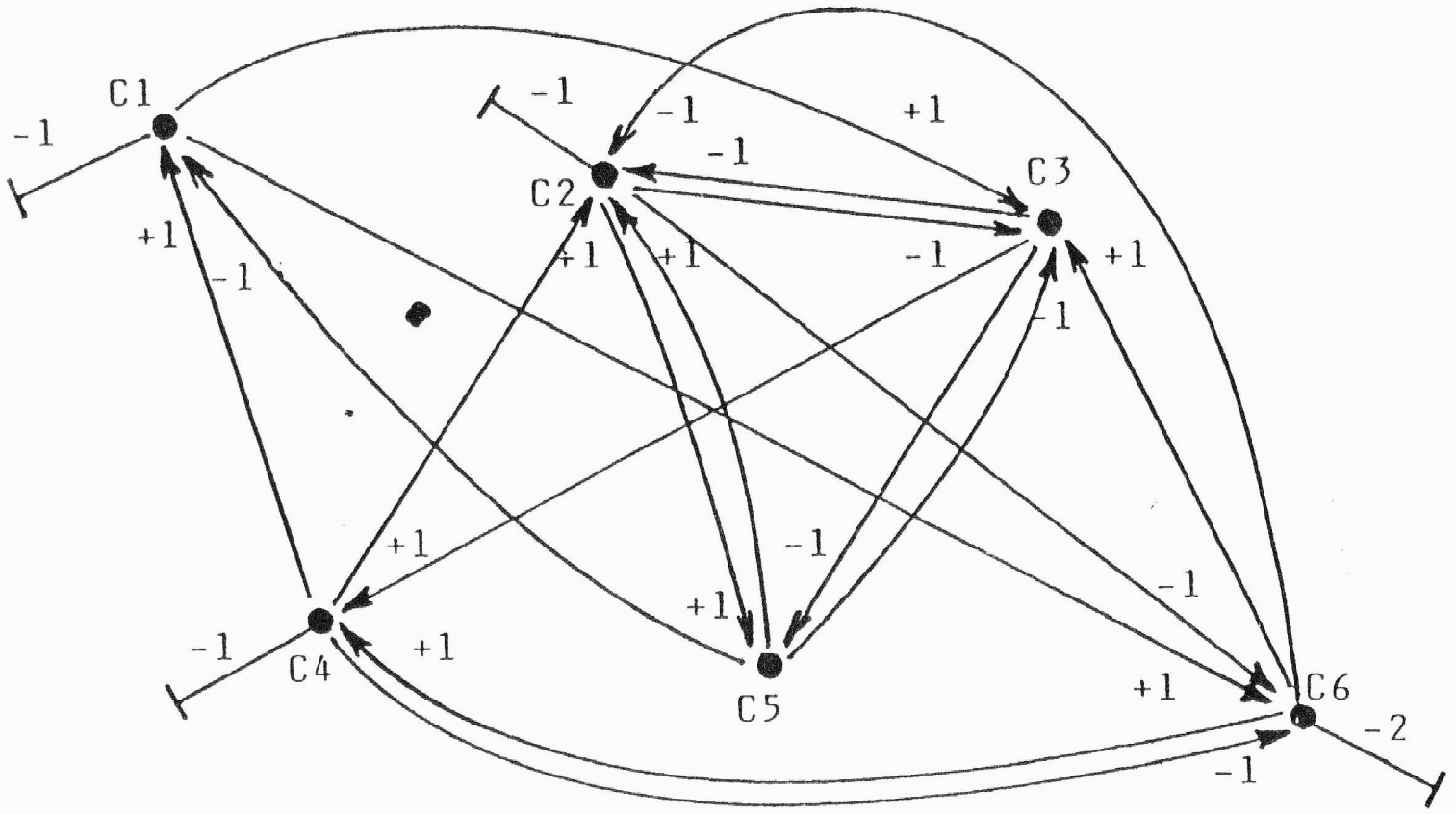


Fig. 6. Illustrations of thresholding role (thresholds are here depicted as ⊣)

sed on our experience with psychological questionnaires processing, and taking into consideration the fact that the thresholds magnitudes may be only positive numbers we proposed thresholding once more immediately after multiplying $x_j \,.\, w_{ij}$. In this case the performance equation is:

$$x_{ij} = S\left(\sum_{j=1}^{n} S x_j w_{ij} - \upsilon_i\right).$$

Using this equation, the result is quite distinct from the first one, i.e. the output vector equals (0 1 1 0 1 1). This result is the same as in the case of the conventi-

Input vector I	Weights matrix W							I x W					
1	0	0	1	0	0	1		0	0	1	0	0	1
−1	0	0	−1	0	1	−1		0	0	1	0	−1	1
−1	0	−1	0	1	−1	0	=	0	1	0	−1	1	0
−1	1	1	0	0	0	−1		−1	−1	0	0	0	1
1	−1	1	−1	0	0	0		−1	1	−1	0	0	0
−1	0	−1	1	1	0	0		0	1	−1	−1	0	0
	State of individual nodes							−2	2	0	−2	0	3
						threshold ϑ_i		1	1	0	1	0	2
				sum		$\sum_j x_j w_{ij} - \vartheta_i$		−3	1	0	−3	0	1
output:	sums		$S\left(\sum_j x_j w_{ij} - \vartheta_i\right)$					0	1	0	0	0	1

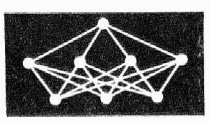
Fig. 7. Recall performance with single thresholding

Input vector I	Weights matrix W							S(I x W)					
1	0	0	1	0	0	1		0	0	1	0	0	1
−1	0	0	−1	0	1	−1		0	0	1	0	0	1
−1	0	−1	0	1	−1	0	=	0	1	0	0	1	0
−1	1	1	0	0	0	−1		0	0	0	0	0	1
1	−1	1	−1	0	0	0		0	1	0	0	0	0
−1	0	−1	1	1	0	0		0	1	0	0	0	0
	State of individual nodes							0	3	2	0	1	3
						thresholds ϑ_i		1	1	0	1	0	2
				sum		$\sum_j S\, x_j w_{ij} - \vartheta_i$		−1	2	2	−1	1	1
output:	sum		$S\left(\sum_j S\, x_j w_{ij} - \vartheta_i\right)$					0	1	1	0	1	1

Fig. 8. Recall performance with double thresholding

onal evaluation of the questionnaire. In *Fig. 7, 8* an attempt to illustrate this phenomena is depicted.

The next improvement which we have made in using neural network inference ability for knowledge processing deals with the necessity of capturing the simultaneous influence of some concepts on others. This function, similar to the logical „and", can be realized by creating new auxiliary nodes in the neural



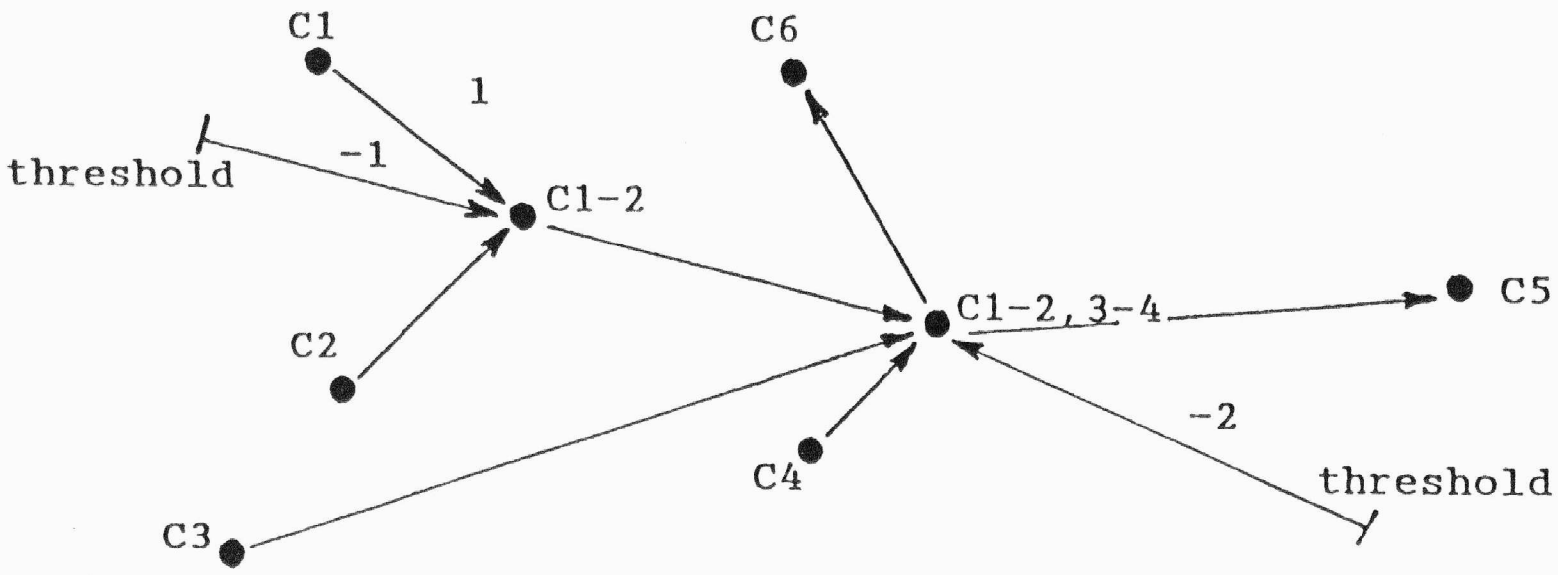


Fig. 9. Realization of "and" function with the help of thresholds

network graph representation. Thresholds of constant (for individual nodes) value are connected to the auxiliary nodes. In the case where weights magnitudes equal 1, the threshold value is given by the equation: $v_i = m - 1$, where m is a inumber of active nodes directed to the node i. Then new nodes are activated only if all links directed to the i-th node are activated (see *Fig. 9*).

In general it is possible with the help of thresholds to regulate the strength of the mutual interconnection between nodes which are here considered as concepts.

3.2. Memory extrapolation networks as a diagnostic system

Consider again a neural network which consists of L nodes where every node is connected to another node. A state of P nodes is known, while a state of the remaining $Q = L - P$ nodes is unknown. If the same P nodes are always used as input (this usually happens in diagnostic systems), the number of interconnections can be reduced. The state of these P nodes is not dependent on their input V. Thus the interconnections to these nodes can be excluded and the network can be reconfigured to $Q < L$ nodes.

Assume again that the first P elements of a query vector is our input. Using equation (1) we can write for sums of inputs to the individual nodes (i.e. for an internal potential of proper neurons):

$$\xi_1(t) = x_2(t)\, w_{12} + x_3(t)\, w_{13} + \ldots + x_L(t)\, w_{1,L},$$
$$\xi_2(t) = x_1(t)\, w_{21} + x_3(t)\, w_{23} + \ldots + x_L(t)\, w_{2,L},$$
$$\vdots$$
$$\xi_P(t) = x_1(t)\, w_{P,1} + x_2(t)\, w_{P,2} + \ldots + x_L(t)\, w_{P,L},$$
$$\xi_{P+1}(t) = x_1(t)\, w_{P+1,2} + x_L(t)\, w_{P+1,2} + \ldots + x_L(t)\, w_{P+1,L},$$
$$\vdots$$
$$\xi_L(t) = x_1(t)\, w_{L,1} + x_2(t)\, w_{L,2} + \ldots + x_{L-1}(t)\, w_{L,L-1}.$$

Since the state of P neurons does not change during network recall, it's useful to partition the above equations into parts as follows:

$$\xi(t) = \xi_P(t) + \xi_Q(t),$$
$$\xi_P(t) = \bar{x}_P W_1 + \bar{x}_Q W_3,$$
$$\xi_Q(t) = \bar{x}_P W_2 + \bar{x}_Q W_4,$$

where

$$\bar{x}_P W_1 = \begin{vmatrix} x_2(t)\, w_{12} + \ldots + x_P(t)\, w_{1,P} \\ \vdots \\ x_1(t)\, w_{P,1} + \ldots + x_{P-1}(t)\, w_{P,P-1} \end{vmatrix}$$

$$\bar{x}_Q W_2 = \begin{vmatrix} x_1(t)\, w_{P+1,1} + \ldots + x_P(t)\, w_{P+1,P} \\ \vdots \\ x_1(t)\, w_{L,1} + \ldots + x_P(t)\, w_{L,P-1} \end{vmatrix}$$

$$\bar{x}_P W_3 = \begin{vmatrix} x_{P+1}(t)\, w_{1,P+1} + \ldots + x_L(t)\, w_{1,L} \\ \vdots \\ x_{P+1}(t)\, w_{P,P+1} + \ldots + x_L(t)\, w_{P,L} \end{vmatrix}$$

$$\bar{x}_Q W_4 = \begin{vmatrix} x_{P+2}(t)\, W_{P+1,P+2} + \ldots + x_L(t)\, W_{P+1,L} \\ \vdots \\ x_{P+1}(t)\, W_{L,P+1} + \ldots + x_{L-1}(t)\, w_{L,L-1} \end{vmatrix}$$

It means that matrix W is partitioned in such a way:

	P elem-s	Q elem-s
P elem-s	W_1	W_3
Q elem-s	W_2	W_4

Now we see that it is not necessary to care about the state of P nodes because it remains the same. Thus W_1, W_2 have no contribution to the final result. Setting $\bar{x}_Q(t+1) = \xi_Q(t)$ the informational part of the equation is given by:

$$\bar{x}_Q(t+1) = \bar{x}_P(t)\, W_3 + \bar{x}_Q(t)\, W_4.$$

Since $x_p(t)\, W_3$ has a constant value during recall, the proper weight matrix is W_4.

In the case where no positive or negative interconnections exist between individual Q nodes, a matrix W_4 contains only zeros. This means that in this case a neural network functions as a diagnostic system which evaluates queries immediately without iterations.

4. The QUEST system and its functions

The QUEST system has been developed using the theoretical background which was discussed in the previous sections. Corresponding to the two main working modes of neural networks (i.e. recall and learning), QUEST consists of two main parts:

i) creation and maintenance of knowledge bases;

ii) query processing.

Further QUEST has a third part which is designed to check knowledge base consistency in a sense that the same concepts have the same meaning. The QUEST functions are depicted in *Fig. 10.*

A knowledge base created with the help of QUEST begins with the definition of so called concepts which can by introduced by different experts (for theoretical background see section 2.2). After checking the consistency of concepts main and users concept files are stored in the system. Now it is possible to define the interconnections between concepts and create in this way a weight matrix, i.e. our knowledge base. It is supposed that interconnections may be defined by an expert or by learning using appropriate training files or by a combination of existing knowledge bases stored in the system. This enables us to develop and extend knowledge bases throughout the lifetime of a system.

The system supports learning according to:

a) the Hebbian rule

$$w_{ij}(t+1) = w_{ij}(t) + x_i(t+1)\, x_j(t+1),$$

b) and the pseudo-Hebbian rule

$w_{ij}(t+1) = w_{ij}(t) + z_i(t)\, z_j(t) \,/\, \sqrt{z_i(t)\, z_j(t)}$, where

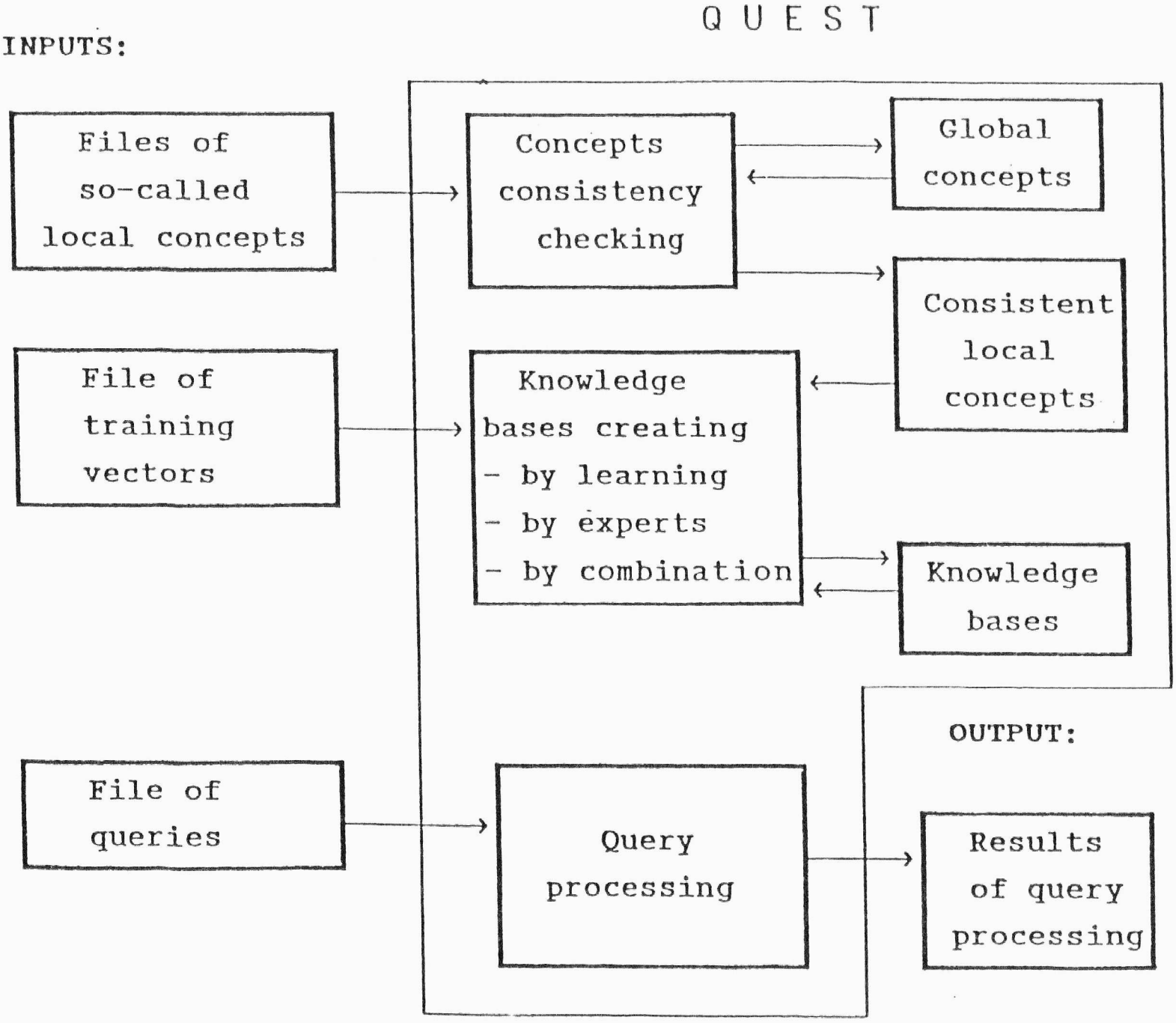
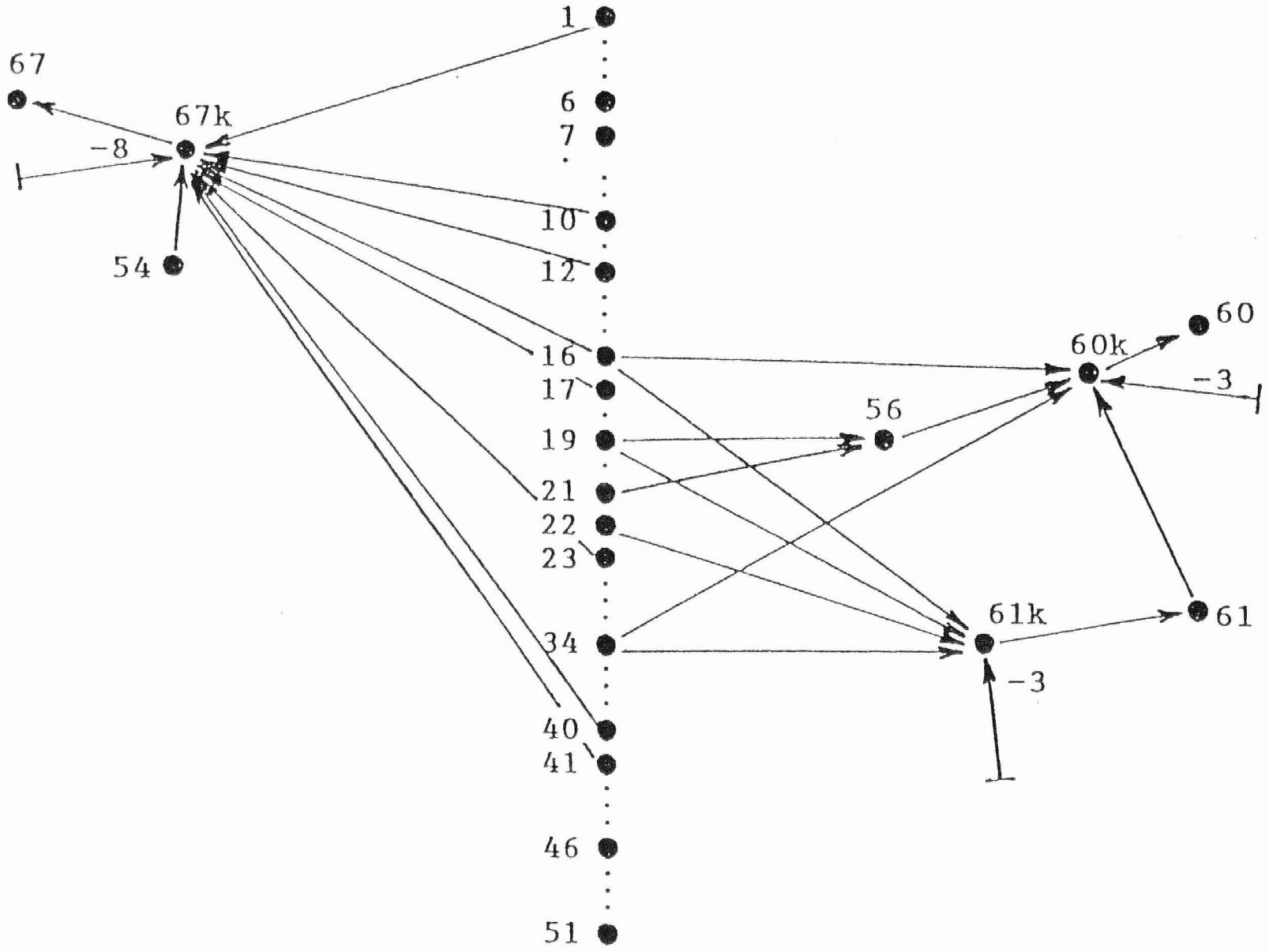


Fig. 10. The functional layout of QUEST



1 - quantity of nuclear activity in the hermetic space of a steam-generator (inside/outside limits)
6 - amount of moisture in the hermetic space of a steam-generator
7 - amount of moisture in the machine-room
10 - level of the steam-generator PG1
12 - amount of active power produced by the generator TG1
16 - value of pressure on a demi-water pump delivery
17 - value of pressure on a delivery of ENC
19 - value of pressure in the hermetic space
21 - value of average pressure in the hermetic space
22 - value of pressure v IO
23 - value of oil pressure in the high pressure valve.of the steam-generator PG1
34 - status of the valve PS-A
40 - status of the valve for the generator TG1
41 - status of the valve for the generator TG2
46 - temperature in the hermetic space
51 - state of regulating valve for generator TG2
54 - steam coming out
56 - difference between values of pressure in the hermetic space
60 - Diagnosis:"One of the valves PS-A is not closed, pressure is constant", followed by the list of instructions recommending what to do,
61 - Diagnosis: "One of valves of PS-A is not closed, pressure is decreasing" with the appropriate instructions what to do,
67 - Diagnosis: "The pipeline of feeder water is ruptured in front of the swing" - check valve of PG1.

Fig. 11. The FCM of alarm handling

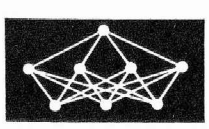
	1	...	46	...	54	.	56	...	60	61	...	67	60k	61k	...	67k
1	0		0		0		0		0	0		0	0	0		1
6	0		0		1		0		0	0		0	0	0		0
7	0		0		1		0		0	0		0	0	0		0
10	0		0		0		0		0	0		0	0	0		1
12	0		0		0		0		0	0		0	0	0		1
16	0		0		0		0		0	0		0	1	1		1
17	0		0		0		0		0	0		0	0	0		1
19	0		0		0		1		0	0		0	0	1		0
21	0		0		0		1		0	0		0	0	0		0
22	0		0		0		0		0	0		0	0	1		0
23	0		0		0		0		0	0		0	0	0		1
34	0		0		0		0		0	0		0	1	1		0
40	0		0		0		0		0	0		0	0	0		1
41	0		0		0		0		0	0		0	0	0		1
46	0		0		1		0		0	0		0	0	0		0
54	0		0		0		0		0	0		0	0	0		1
56	0		0		0		0		0	0		0	1	0		0
60	0		0		0		0		0	0		0	0	0		0
61	0		0		0		0		0	0		0	1	0		0
67	0		0		0		0		0	0		0	0	0		0
60k	0		0		0		0		1	0		0	0	0		0
61k	0		0		0		0		0	1		0	0	0		0
67k	0		0		0		0		0	0		1	0	0		0
Thresholds	0		0		0		0		0	0		0	−3	−3		−8

Fig. 12. The connection matrix of alarm handling

$$z_i(t)\ x_i(t+1)\ w_{ij}(t)\ .\ x_i(t+1).$$

With respect to the fact that a vector-stimulus obtains only zeros and ones (binary form) whereas a weight matrix contains in general numbers from <-1, 1> in the learning algorithms used in QUEST, the term $x(t+1)$ is replaced by $2x(t+1) - 1$.

The system is able to process matrices of weights which contain both positive and negative values of weights. The ability of an associative memory model



to reconstruct an entire vector from a partial input is used here for inference. An input vector (a query to a knowledge base) is considered to contain only zeros and ones.

During recall, the system synchronously updates elements of a network with the equation:

$$x_i(t+1) = S\left(\sum_{j=1}^{n} S w_{ij}(2x_j(t)-1) - \upsilon_i\right).$$

The end of the iteration process is detected by a minimum of the so-called harmonian function which is calculated according to the equation:

$$H(x_1, \ldots, x_n) = \sum_{j=1}^{n}\sum_{i=1}^{n} x_{ij}(2x_i - 1)(2x_j - 1).$$

5. Application

Neural network based knowledge systems may be especially useful in real-time control and information systems. Alarm handling in power stations and the diagnosis of the emergency states is an example of a task that must be solved as fast as possible. In modern energy control centers, a large amount of detailed information is measured and displayed. Especially in the case of severe faults in the power system, the operator is flooded with a shower of messages, so that it is often a problem to determine which are the really important messages and to decide on the quickest way to return to a normal situation. We have chosen this area of application for testing our approach with the help of QUEST before all because we had an opportunity to deal with the set of experimental and real data due the courtesy of our colleagues from respective research institutes [22].

Let us examine in more detail how a diagnostic system based on an associative memory model (see section 3.2) can help in this case. Assume that fifty one primary indicators are placed in the steam generator environment for monitoring its auxiliary equipment, e. g. temperature and pressure of oil or water, switch gear status, vents status, etc. [22]. Using these indicators (numbered from 1 to 51) and knowing their influence on the state of a steam-generator, the corresponding FCM is drawn (see *Fig. 11*). In this FCM the concepts (or nodes) numbered 1 to 51 are always input nodes. The nodes numbered 52 to 67 are always output. The nodes with the letter „k" in their number are auxiliary and by using them, we are able to express the fact that only the conjunction of certain concepts can influence another concept(s).

Now consider for example that indicators 16 (pressure in the demi-water pump delivery dangerously increased), 19 (pressure in the hermetic space also dangerously increased), 22 (pressure in IO dangerously increased too), and 34 (the valve PS A is not closed) are activated. In that case, we have the input (query) vector in which the input nodes 16, 19, 22, 34 are equal to 1 while all others are equal to 0. After inference process is finished the value of the output node 61 will be equal to 1. If any of the nodes 16, 19, 22, 34 is not activated, node 61 remains equal to 0. Notice node 56, which may be activated in case of an active state at node 19 or node 21 or both 19 and 21. This means that for node 56 to contribute to the activation of the node 60, it is enough for either node 19 or node 21 to be active. We have tested this example for whole alarm handling for steam generator according to [22] and the results tallied with expected ones.

Needless to say, such partial knowledge bases monitoring different equipment in a power station can be gradually developed. After testing they can be combined (step by step too) into one representative knowledge base for alarm handling in the whole power station.

6. Conclusion

The first experience gained during the building and testing of the QUEST system showed that an associative memory model of neural networks offers a powerful tool for inference in knowledge processing; meanwhile the learning abilities provided by QUEST based on this model did not appear as successful. Therefore the system now supports adaptive inference in the sense of expert knowledge and the combination of weight matrices.

Needless to say, the problem of conditional probabilities expressed by real numbers in a weight matrix was not addressed nov solved The problem is especially important in the case of a combination of such matrices. Nonetheless, by building QUEST we have developed assumptions for the serious testing of knowledge processing by neural networks in real conditions.

References

[1] Bradshaw G., Fozzard R., Ceci L.: A connectionist expert system that actually works. 1989, pp. 248-255.

[2] Cruz C.A., Hanson W.A., and Tam J. Y.: Knowledge processing through flow-of-activation. In: Proc. IEEE 2nd Int. Conf. on Neural Networks, San Diego, 1987, pp. II/343-350.

[3] Husek D.: HOPNET 2X5. Experimental model of Hopfield—like neural network. Technical report No. V-418, the Institute of Computer Science of the Czechoslovak Academy of Sciences, 1989.

[4] Irani I.A., Slagle J.R., Long J.M., Mattis J.P. and the Posch group: Formulating an approach to develop a system for the temporal analysis of clinical trial data — The Posch AI Project. In: Proc. of the 2nd Int.Conference on Statistics and AI, Florida, 1989, pp.24-1 to 24-9.

[5] Kosko B.: Adaptive reference in fuzzy knowledge networks. In: Proc. IEEE 1st Int. Conference on Neural Networks, Diego, 1987, pp. II/168-261.

[6] Kosko B.: Fuzzy cognitive maps. Int. J. Man-Machine Studies, 1986, vol. 24, pp. 65-75.

[7] Kosko B.: Fuzzy knowledge combination. Int. Journal on Intelligent Systems, San Diego, 1986, vol. 1, pp. 293-320
[8] Kosko B.: Feedback stability and unsupervised learning. In: Proc. IEEE 2nd Int. Conference on Neural Networks, San Diego, 1988, pp. I/141-152.
[9] Kufudaki O., Horejs J.: Computers and brain (in Czech). In: Proc. Conference SOFSEM'88, Tatras Czechoslovakia, Dec. 1988, pp.119-156 .
[10] Lippman R.P.: An introduction to computing with neural nets. IEEE ASSP Magazine, 1987, pp. 4-22.
[11] Marks R.S.: Class of continuous level associative memory neural nets. Applied optics, May 1987, vol.26, pp.2005-2010.
[12] Saito K., Nakano R.: Medical diagnostic expert system based on PDP model. In: Proc. IEEE 2nd Int. Conference on Neural Networks, San Diego, 1988, pp.I/255-262.
[13] Schoen E., Smith R. G., Buchanan B..G.: Design of knowledge-based systems with knowledge-based assistant. IEEE Transactions on Software Engineering, vol. 14, No. 12, 1988, pp. 1771-1781.
[14] Vítková G., Húsek D.: Integration of knowledge bases and databases (in Russian). In: Proc. Conference KNVVT on Personal Computers, Budapest, May 1986, Tanulmanyok 193/1986, pp. 39 — 52.
[15] Vítková G., Miček J.: Evaluation of binary input vectors by help of neural network inference mechanism (in Czech). Technical report No.V-416, the Institute of Computer Science of Czechoslovak Academy of Sciences, Prague, 1989.
[16] Vítková G.: Problems related to integration of knowledge bases and databases (in Russian). In: Proc. WG-25 KNVVT on Problems and Tools of Information Systems Integration , INFORMA Sofia, Bulgaria,1989, pp. 69-82..
[17] Vítková G., Míček J., Ianson N.: Neural networks and knowledge processing (in Russian). Technical report No.V-438, the Institute of Computer Science of Czechoslovak Academy of Sciences, Prague, 1989.
[18] Vítková G., Miček J.: The neural network based system for knowledge processing. In: Proc. Int. Symposium on Neural Networks and Neural Computing NEURONET '90, Prague, September 1990, pp.345-347.
[19] Vítková G.: Neural networks associative memory models for knowledge base building (in Russian). In: Proc. WG — 25 KNVVT on Problems and Tools of Information Systems Integration, Breitenbrunner, Germany, May 1990, (to appear).
[20] Weisbuch G., D' Humieres D. Determining the dynamic landscape of Hopfield networks. NATO ASI Series, vol. F30' Disordered Systems and Biological Organization, Springer Verlag, Berlin, Heidelberg, 1986, pp.187-191.
[21] Zhang W.-R., Chen S.S.: A logical architecture for cognitive maps. In: Proc. IEEE 2nd Int. Conference on Neural Network, San Diego, 1988, pp. I/231-238.
[22] Trojánek Z.: Using the methods of artificial intelligence for electrical power systems control (in Czech). Technical report No. DU III-8-4/03, the Faculty of Electrical Engineering of Technical University in Prague, September 1990.

Literature Surwey

Caudill M.: Using Neural Nets: Fuzzy Decisions (fuzzy logic) (part 2) (includes related article on air conditioner controlled by fuzzy logic)
AI-Expert, Vol. 5, 1990, No. 4, pp. 59—63

Key words: fuzzy sets; logical organization; set oriented languages; artificial intelligence; neural networks; Iona College.

Abstract: The use of fuzzy logic and decision tools in neural networks and expert systems is discussed. Fuzzy set theory is designed to mimic human thought and solve problems in which membership in a set is a continuous "gray" scale rather than a defined boundary. A fuzzy set lets an object have "partial" membership, assigning it a membership value between 0 and 1. Fuzzy logic deals properly with the combinations of vague or uncertain factors found in many expert system applications. The Yager decision making procedure, published by Ronald R. Yager of Iona College, uses fuzzy sets to decide between several possibilities. It uses simple subtraction and "min" and "max" functions to provide effective, elegant solutions.

Cosset J. C., Roy J.: Forecasting Country Risk Ratings using a Neural Network
IEEE, 1990, pp. 327—334

Abstract: This research examines the performance of a neural network at predicting Institutional Investor's country risk ratings using eight economic indicators. The network is trained on the period 1983—1985 with the backpropagation of error algorithm. It is then tested on data for 1986 and its results are compared with a logistic regression model. The evidence confirms the potential of the method for economic applications.

Dabrowski L., Pacut A.: A diffusion model of a neuron and its identification
Syst. Anal. Model. Simul., Vol. 7, 1990, No. 2, pp. 139—144

Abstract: In the paper an electrical activity of a neuron is consided. The cell is viewed as an impulse receiver and transmitter. The mechanism of converting input impulse trains into output trains is inherently connecte with a discrete behavior of the input and output signals. An approach proposed here is to analyze the point-behavior of a cell by means of stochastic point processes. An algorithm applied here has a multilayer structure.

Gallant S. I.: A Connectionist Learning Algorithm with Provable Generalization and Scaling Bounds
Neural Networks, Vol. 3, 1990, No. 2, pp. 191—201

Key words: connectionist; neural network; computational learning theory; perceptron.

Abstract: A connectionist learning algorithm, the bounded, randomized, distributed (BRD) algorithm, is presented and formally analyzed within the framework of computational learning theory.

Goodhill G. J., Willshaw D. J.: Application of the elastic net algorithm to the formation of ocular dominance stripes,
NETWORK, Vol. 1, 1990, No. 1, pp. 41—59

Key words: models of nets.

Kahlert C. L., Chua L. O.: A Generalized Canonical Piecewise-Linear Representation
IEEE — Transaction on Circuits and Systems, Vol. 37, 1990, No. 3, pp. 373—382

Lang K. J., Waibel A. H., Hinton E. G.: A Time-Delay Neural Network Architecture for Isolated Word Recognition
Neural Networks, Vol. 3, 1990, No. 1, pp. 23—43

Key words: constrained links; multiresolution learning; multispeaker speech recognition.

Abstract: A translation-invariant back-propagation network is described that performs better than a sophisticated continuous acoustic parameter hidden Markov model on a noisy, 100-speaker confusable vocabulary isolated word recognition task.

Marose R. A.: A Financial Neural-Network Application. (ADAM Hybrid Neural Network Used at Chase Manhattan Bank).
AI-Expert, Vol. 5, 1990, No. 5, pp. 50—53

Key words: artificial intelligence; neural networks; banking; financial software; custom software; pattern recognition.

Abstract: Chase Manhattan Bank uses a statistical based hybrid neural network application based on the ADAM pattern analyse tool developed by Inductive Inference Inc. The neural network is designed to reduce losses on loans made to corporations by performing three-year forecasts indicating likely future risk classifications. Input to ADAM includes each firm's historical financial statement data and industry norms calculated using similar data from companies in specific industries. The Public Loan Company (PCLM) expert system model predicts the likelihood of a public company earning a „good", „criticized", or „charged-off" rating three years in advance based on historical data.

Marrian C. R. K., Mack I. A., Banks C., Peckerar M. C.: Electronic "Neural" net algorithm for maximum entropy solutions to Hl-posed problems — Part II. : multiply connected electronic circuit implementation
IEEE Transactions on Circuits and Systems, Vol. 37, 1990, No. 1, pp. 110—113

Key words: technical electronic realization.

Marshall J. A.: Self-Organizing Neural Networks for Perception of Visual Motion
Neural Networks, Vol. 3, 1990, No. 1, pp. 45—74

Key words: hypercomplex cells; cooperative competitive learning; aperture problem; intrinsic connections.

Abstract: A new approach to the aperture problem is presented, using an adaptive neural network model.

Myers W.: Artificial Neural Networks are Coming (An interview with Caltech's John Hopfield)
IEEE — Expert, Vol. 5, 1990, No. 2, pp. 3—6

Key words: neural networks; IEEE; expert systems.

Abstract: John J. Hopfield, professor of chemistry and biology at California Institute of Technology and member of the National Academy of Sciences, discussed the neural network field: how artificial neuron networks follow the biological structure; how neurobiology can be helpful in developing neural networks; common principles of neural network models; how computation relates to hierarchy; how digital computers relate to neural networks; the application of optical technology to neural networks; existing systems based on neural networks; and future applications of neural networks.

Nadal J. P., Toulouse G.: Information storage in sparsely coded memory nets
NETWORK, Vol. 1, 1990, No. 1, pp. 61—74

Key words: models of nets.

Newquist H. P. I.: Neural Networks that Work
AI-Expert, Vol. 5, 1990, No. 5, pp. 67—69

Key words: artificial intelligence; neural networks; problem solving; expert systems; software design.

Abstract: neural networks are not yet ready for the mass market, but some excellent working systems are now in use. Defense contractor Raytheon Inc. uses NeuralWare's NeuralWorks development tool to create antisubmarine warfare applications, performing feature analysis on data collected from sonar signals. SAIC of San Diego, CA developed the SNOOPE bomb detector, which has been deployed at several airports and can determine chemical configurations from certain materials. Medical applications include a program developed by the University of Pittsburgh to assist in diagnosing eye disorders.

Orfanidis S. J.: Gram-Schmidt Neural Nets
Neural Computation, Vol. 2, 1990, No. 1, pp. 116—126

Abstract: A new type of feedforward multilayer neural net is proposed that exhibits fast convergence properties. It in defined by inserting a fast adaptive Gram-Schmidt preprocessor at each layer, followed by a conventional linear combine-sigmoid part which is adapted by a fast version of the backpropagation rule. The resulting network structure is the multilayer generalization of the gradient adaptive lattice filter and the Gram-Schmidt adaptive array.

Peterson C., Redfield S., Keeler J. D., Hartmen E.: An Optoelectronic Architecture for Multilayer Learning in a Single Photorefractive Crystal
Neural Computation, Vol. 2, 1990, No. 1, pp. 25—34

Key words: optoelectronic; architecture; learning.

Abstract: Simple versatile architecture for implementing supervised neural network models optically with photorefractive technology is proposed: a wide range of supervised learning algorithms can be implemented including mean-field-theory, backpropagation, and Kanerva-style networks. It is based on a single crystal with spatial multiplexing rather than the more commonly used angular multiplexing. It handles hidden units and places no restrictions on connectivity. Associated with spatial multiplexing are certain physical phenomena, rescattering and beam depletion, which tend to degrade the matrix multiplications. Detailed simulations including beam absorption and grating decay show that the supervised learning algorithms (slightly modified) compensate for these degrations.

Rao A., Walker M. R., Clark L. T., Akers L. A., Grondin R. O.: VLSI Implementation of Neural Classifiers
Neural Computaion, Vol. 2, 1990, No. 1, pp. 35—43

Key words: associative memory; implementation.

Abstract: The embedding of neural networks in real-time systems performing classification and clustering tasks requires that models be implemented in hardware. A flexible, pipelined associative memory capable of operating in real-time is proposed as a hardware substrate for the emulation of neural fixed-radius clustering and binary classification schemes.

Rodriguez-Vazquez A., Dominguez-Castro R., Rueda A., Huertas Y. L., Sanchez-Sinencio E.: Nonlinear Switched-Capacitor "Neural" Networks for Optimization Problems
IEEE Transactions on Circuits and Systems, Vol. 37, 1990, No. 3, pp. 384—398

Key words: technical electronic realization.

A VIEW ON NEURAL NETWORKS PARADIGM DEVELOPMENT

(Part 3)

J. Hořejš)*

Here we continue in the tutorial paper concerning the neural network paradigm, which first part was published in the Neural Network Word. No. 1, 1991.

6. Back-propagation [a famous learning algorithm]

Our task now is to apply and illustrate general discussion of adaptation from section 3 for the case of complete multilayered nets. We will introduce the so called *back-propagation algorithm* (BP), one of the most used and most promising adaptation algorithm invented and elaborated by Rumelhart, Hinton, Williams and others from the PDP [Parallel Distributed Processing] group, like McClelland and Sejnowski not many years ago (although Werbos claims he covered the main ideas in his thesis in time of second generation NNs). Let a CMN homogeneous net N with sigmoid nonlinearity S from (7) be given (one input layer, one output layer and several hidden layers).

Suppose we have a training set T, $[\mathbf{x}^j, \mathbf{y}^j] \in T$ and we would like to establish all the many weights [in a m-k-l-n net there are $k.(m+1) + l.(k+1) + n.(l+1)$ weights including thresholds] so that the net $N_\mathbf{w}$ would implement the mapping $\varphi_\mathbf{w}$ such that $\varphi_\mathbf{w}(\mathbf{x}^j) \cong \mathbf{y}^j$, where the sign $\cong$ stands for „approximately equal". The error is caused (inevitably) whenever some components of $\mathbf{y}^j$ should be 0 or 1 like in classification problems (because of asymptotic behavior of S) or because the convergence of adaptation process is not ideal (the error function does not reach zero, convergence is too slow). This again may be due to unprecise arithmetic and/or numerical solution or due to the fact that the whole problem is too difficult for the chosen net (too small or too great number of hidden layers and/or hidden neurons, unlucky initial weights etc) or for the BP as such (not a rare case, sometimes avoided by some modifications of BP, preprocessing etc).

In many cases we also use a slightly different terminology, speaking about (input and output) *patterns* instead simply of vectors, borrowing this term from artificial intelligence theory or even cognitive and brain science.

We shall now the assume there is a *teacher*, who knows exact outputs, so that we can immediately calculate the error as it appears in the output layer. What makes the trouble is to see how any one of the particular connection weight (somewhere in the deeper parts of the net) contributes to bad performance of N, whom to blame or praise (*credit assignment problem*).

We know similar problems from the daily life. It is not difficult for headquarters to recognize that something is getting wrong and it is even not so difficult to accuse the hierarchy just below. The real problem is to find exactly who and to what extent in the whole company is responsible for undesirable results and how anyone should change to minimize the overall dysfunction. A right way out might be that on every level the accused leader tries to improve something in his own work and passes a memorandum to his men in such a way that everybody knows what to do to diminish the total error. Things are complicated in CMNs by the fact that everybody may have more chiefs and every chief is responsible for the whole collective in the level just beneath him.

Yet this metaphor can be adopted in BP, each upper layer successively informing the members (weights) of immediately lower layer how much and in which direction they should change. Fortunately enough, everyone is characterized by the only number — its weight and is expected to change it slowly. Unlike the activity spreading, which goes bottom up, the error warnings proceed from top down — hence the term (error) back-propagation.

The main trick how to diminish the error function has been shown already on *Fig. 9*. Choosing any particular weight w [which represents just one coordinate value of the vector $\mathbf{w}$ in the many-dimensional space of weights], we shall try to change w so that E_w diminishes (ever-present parameter $\mathbf{w}$ in E will be in the next often omitted). We simply calculate the slope of the curve [in this coordinate], which is given by the derivative $dE/d\mathbf{w}$ and because of its *hill climbing* orientation (if positive), we move w slightly in the opposite direction, the more, the bigger is its responsibility, i.e. the bigger is the derivative. Thus we obtain the formula for the *update* of w

$$\Delta w = -\eta \cdot dE/dw \text{ for some } \eta,\ 0 < \eta < 1 \qquad (17')$$
$$w^{\text{new}} = w^{\text{old}} + \Delta w$$

However, the considered w runs along one coordinate only. As $E_\mathbf{w}$ is a composite function, we have to replace dE/dw by the partial derivative $\partial E/\partial w$. Thus (17') changes to

$$\Delta w = -\eta \cdot \partial E/\partial w \text{ for some } \eta,\ 0 < \eta < 1 \qquad (17)$$

The main task is now to compute these values $\partial E/\partial w$ for any w in the CMN. To do that we use a picture of a pertinent fragment of a CMN, see *Fig. 18*. Here two adjacent layers are taken into account and w, x, ξ represent weights, activities of stimuli along the connections and net incomes, respectively. All used values are scalars and subscripts/superscripts do here denote

*) Prof. Dr. Jiří Hořejš, CSc., Department of Computer Science, Charles University, 118 00 Prague 1, Malostranské nám. 25, Czechoslovakia

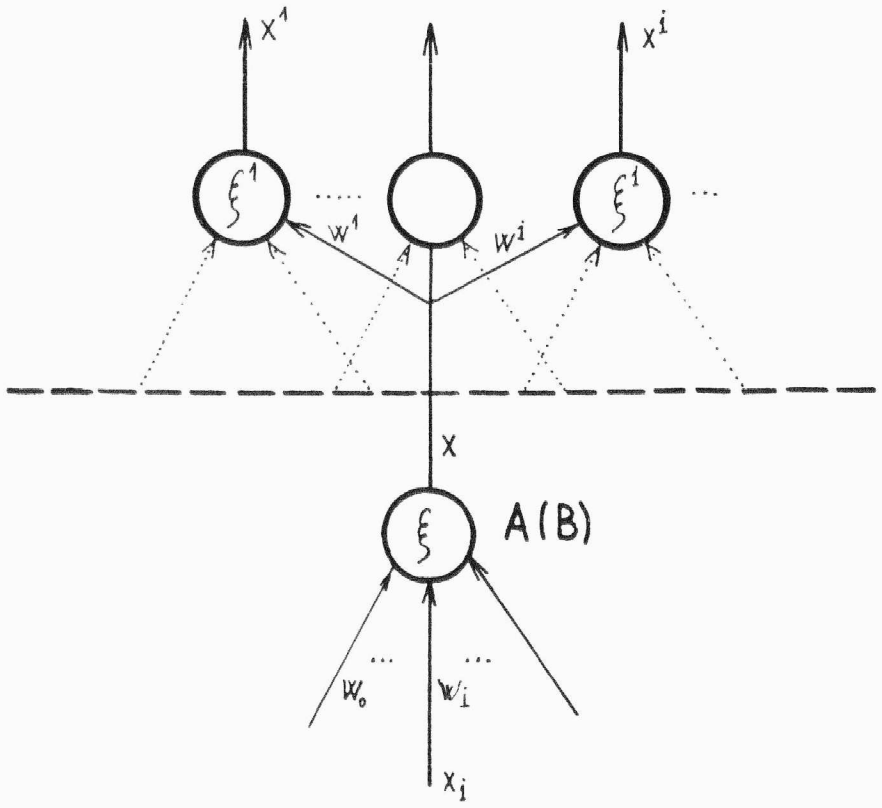


Fig. 18

only that we deal with the lower/upper layer of the two. The dotted arrows indicate that there are other connections in the net, which do not enter directly into the calculations while the horizontal dashed line is used to distinguish two cases: the neuron A can be an output one, in which case the picture above the dashed line is empty; otherwise it is an input or hidden neuron.

The weight we focus now on is w_i and we first try to establish $\partial E/\partial w_i$ when a selected one member of the training set T[!] is examined and the error for this case corresponding the inner sum in (6) should be diminished. We take the following formula for $\partial E/\partial w_i$ and thus for the simple adaptive action:

$$\partial E/\partial w_i = \partial E/\partial x \,.\, dx/d\xi \,.\, \partial \xi/\partial w_i \qquad (18)$$

[For further purposes, let us denote $\delta_i = \partial E/\partial x \,.\, dx/d\xi$ speaking about error for neuron A]

This awfully looking formula only expresses the rule for derivation of composite functions [the chain rule] and because all involved functions have nice formal properties (being linear sums and differentiable S), you can remember the mnemonic rule about dealing with fractions and see in this way why the left-hand side of (18) equals to the right-hand side. Note only that, while $x = S(\xi)$ so that $dx/d\xi = x.(1 - x)$ as was stated in (8). Also $\partial \xi/\partial w_i$ brings no troubles; as $\xi = \Sigma\, w_i x_i$ is a linear function of w_i, $\partial \xi/\partial w_i = x_i$. [See (11ab)]. If there is some problem, it is in computing of $\partial E/\partial x$. When A is an output neuron, x is what we formerly denoted as y_i^j for some i, j and according to the definition (6) or its inner sum when j-th member of the training set is submitted, we easily calculate (perhaps up to a multiplicative constant)

$$\partial E/\partial x = y_i^j - \bar{y}_i^j \qquad (19)$$

(19) which is the difference between expected and actual value of neuron i ($1 \leq i \leq n$) for the pattern j. If, on the other hand, the considered neuron B is somewhere inside the net, we calculate $\partial E/\partial x$ as the sum

$$\partial E/\partial x = \Sigma\, \partial E/\partial x^i \,.\, dx^i/d\xi^i \,.\, \partial \xi^i/\partial x \qquad (20)$$

where the sum is taken over all neurons in the upper layer, summing thus all error driven changes for which upper layer neurons are responsible. It is again easy to see that $dx^i/d\xi^i = x^i \,.\, (1 - x^i)$ and that $\partial \xi^i/\partial x = wu^i$ ($\xi = wu^i x + \ldots$ [sum of contributions along the dotted lines, where x does not occur]). What remains is to establish $\partial E/\partial x^i$ for all i (for all neurons in the upper layer, the number of which has not been denoted).

But even this last step is not too difficult to solve; actually it has already been solved! Provided that we started the whole process from the topmost (output) layer, for which it was solved by (19), we can now assume in this top-down (recursive) computation that actually we at the present state already know the values of $\partial E/\partial x^i$ and can substitute them into (20). This completes the formal derivations behind the BP.

Sometimes the successive changes of $w = w(t)$ [so that $\Delta w = w(t+1) - w(t)$] can lead to faster and smoother convergence if we add the *„momentum"* term and replace (17) by

$$w(t+1) = w(t) - \eta \cdot \partial E/\partial w + \alpha \cdot (w(t) - w(t-1)) \text{ for } \alpha,\ 0 < \alpha < 1, \qquad (21)$$

so that the difference Δw in one step is smoothed by the same difference one step before.

In section 8 we give a general account on possible applications of BP driven CMN adaptation. Because some readers may wish to try small experiments along that line and because it is not so trivial to pass from the theoretical description to the development of a practical numerical algorithm, we present first a summary of the BP algorithm and in sect. 7A the core of corresponding program in Pascal. That section can be omitted without any consequence for further reading; it was however postponed after sect. 7, because there are further needed concepts introduced.

Summary of the basic BP algorithm.

0. Initialize weights **w**0 (including thresholds) to small random numbers and the parameters η, α (say η about 0.3, α about 0.7).

1. For a chosen pair of patterns $[\mathbf{x}, \mathbf{y}] \in T$ compute $\bar{\mathbf{y}} = \varphi_{\mathbf{w}0}(\mathbf{x})$, so that the difference between expected and actual value $\mathbf{y} - \bar{\mathbf{y}}$ should be for next weight vector **w**1 [and then generally and recursively, the vector $\mathbf{w}_t$] possibly diminished. To do so:

2. For a weight w_{ij} leading from some neuron j to a neuron i set

$$\Delta w_{ij} = -\eta \cdot \delta_i \cdot \bar{x}_j,$$

where δ_i is the error for i and $\bar{x}_j$ is either output from j or j-th coordinate of the input vector $\mathbf{x}$, and δ is computed as follows:

$\delta_i = y_i \cdot (1 - y_i) \cdot (y_i - \bar{y}_i)$ if the neuron i is an output one

$\delta_i = \bar{x}_i \cdot (1 - \bar{x}_i) \cdot \sum \delta_k \cdot w_{ki}$ otherwise;

the sum is taken over all neurons in the layer immediately above neuron i. In this way you obtain $\mathbf{w}_{t+1}$ from $\mathbf{w}_t$.

If you like, you can introduce $\Delta = -\eta \cdot \delta_i x_j + \alpha \cdot \Delta'$, where Δ' is the difference between two successive previous values of w_{ij} (according to (21).

3. Choose another pair from T and repeat 2 until you exhaust T according to a chosen training strategy.

4. Repeat from 2 until for all pairs $[\mathbf{x}^j, y^j] \in T$, $\varphi(\mathbf{x}^j)$ is sufficiently close to $\mathbf{y}^j$, i.e. E_w is small enough.

7. Comments and analogies on BP [a continuation]

Note that during the calculation of new weights we need to know what are the values of x, ξ, etc. These were of course computed during the predecessing active mode. In the described case thus both modes of work are interleaved: for an input vector from T, the active mode is performed, followed by the adaptation mode, i.e. by the calculations of $\partial E/\partial w$ for each weight and consequent modification of w.

As already mentioned, it is also possible to use inhomogeneous CMN with nonlinear transfer functions S_λ, where λ depends on particular neurons. Moreover, it can change during the adaptation and the change can be aimed to further accelerate and/or enable the convergence process seeking the best suitable $\mathbf{w}^*$, for which E_{w^*} would reach a low acceptable value. Our experience [Pelikan used it many times] with this modification is generally good. It starts from formal derivation of $\partial E/\partial \lambda$ by only a slightly more difficult calculations than those above. Then you extract from this derivation the adaptation process for λ, interleaving (for the still fixed neuron, for which w and λ are considered) adaptation of w with adaptation of λ. The resulting modification of BP may be called GABP [*Gain Adapting* BP]. Details will appear in an independent contribution in this journal.

Most implementations of BP allow to require that few of the weights remain fixed during the adaptation; for these we simply set $\Delta w = 0$. In this way we can apply BP also to not complete multilayered nets (fixing some weights to 0). Also you can prescribe some constrains on the weight vectors (e.g. requiring that selected weights assume always equal values). Because in such (relatively rare) cases the high connectivity is as a rule preserved, or the constrains are not formally too strong, we shall not treat them separately.

Note that there are some formal similarities between active and adaptive dynamics (you always compute sums of certain contributions passed to adjacent layer). For an original treatment of BP see Hecht-Nielsen monograph mentioned in the introduction.

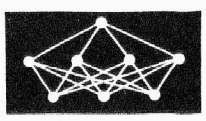
The above derivations concern adaptation of every weight, but for one training pair only. We can however use also *cumulative errors* either for observing the adaptation process or for accumulating errors in several active passes before we start adaptation, not to overload the program with many recomputations of w's, which are rather time-consuming. For example, sometimes we may be interested in responses of only some of the output neurons, the others belonging to a don't care area. Often we are interested in the *layer error*, summing up the (squared) differences between expected (desired) and actual outputs over all neurons in the output top layer, but for one training pair from T only.

However the most useful form of cumulative error is the *global error*, summing up errors for all output neurons and all the pairs from T actually used in training. This may incorporate either one input pair from T just once [following thus directly (6)] or other training strategies as well. A usual training strategy repeats every training pair from T several times (for some number of *iterations*) and only then goes to another pair; only after exhausting the whole set T we start another *cycle* of repeating T. It is also recomended to compute and display the plain (nonsquared) difference between the actual and expected value for the worst case output neuron / pattern pair, for example each iteration, or perhaps cycle if the computation is quick enough.

In some strategies, random decisions are involved, assuming that after a specified *epoch* every training pair will get the same chance for its intervention (the random generator should uniformly cover the set T). The results of adaptation convergence may look different for different strategies then.

The various error measures do not always preserve the nice theoretical property that the error function steadily diminishes. First, numerical approximations may cause e.g. that with too big value of η from (17) or (21) BP skips over some minimum. Second, due to the fact that we at the same time try to adapt the considered weight w as to diminish the overall error for all output neuron/pattern pairs, the adaptation process may happen to „prefer" some particular pairs diminishing their errors, while others, less aggressive, pay for it in such a way that the global error (or other error measures) increases.

Giving up the cases when E_w starts to persistently climb up or to oscillate, we often meet the curve of *Fig. 19*, giving the dependency of global error on time (as the adaptation process proceeds). For many cases, in which we are not satisfied with the shape of global error curve, some remedies are known one of which is the increase of the number of hidden neurons, giving thus the net more "degrees of freedom".



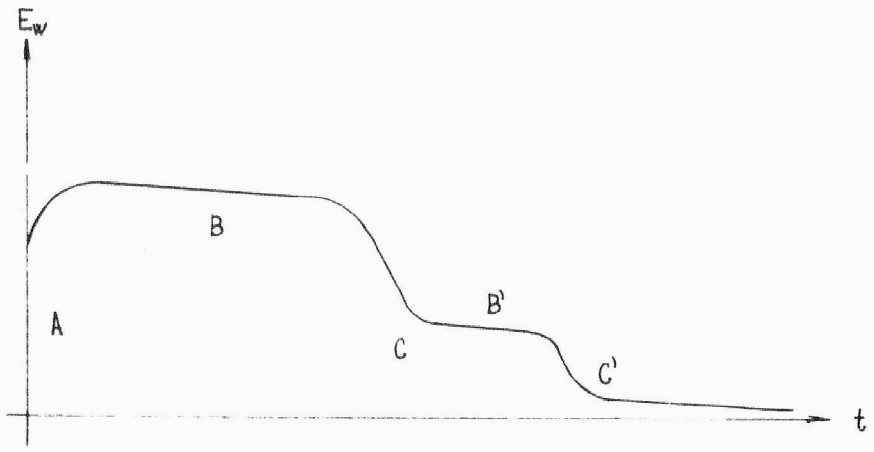


Fig. 19

The frequent behavior of E_w from *Fig. 19*, far from being the only possible, admits again a human-oriented metaphor.

When we meet a task to solve, we first start with a more less arbitrary weight vector **w**0, far from a reasonable one. From the very beginning we are overwhelmed by complexity of the task and the error increases (A). Then we start to look around to understand better our task and try to see some promising ways out. We wander over the landscape which looks flat (B,B') until we get some idea which deserves further attention and elaboration; this enables us to quickly proceed and improve the idea (C,C').

Concerning dependency of the error function on the choice of the training strategy, there is again an analogy: if you have to read a textbook consisting of p chapters, you may prefer to read every chapter r times and only then proceed to the next one. Or you prefer to read the whole book from the beginning to the very end and repeat the complete reading $p \,.\, r$ times. In both cases you read the same number of pages, but (depending on your memorizing and generalization abilities) the result of understanding may be somewhat different. You may also occasionally return to the most difficult parts (pages, chapters) as you feel appropriate. Similar choices of the training strategy project in the adaptation of a NN.

The problems of generalization has already been discussed; for simple functions it reminds the mathematical questions of interpolation (by polynomials, Taylor or Fourier series etc). NNs mappings are specific in this respect in that they handle multivariable mappings, coordinates of input/output vectors may have different meanings and actually (though not formally) be of different types. Moreover, CMNs compose rather wild mixture of linear and nonlinear mappings, and — above all — BP finds these mixtures in the process of adaptation automatically, according to the task given and training strategy chosen. No wonder that NNs often provide better interpolation / generalization solutions than known tools with a structure fixed before. And no wonder they find many applications, provided that hidden neurons choice, initial weight setting, training strategy and the parameters η, α and perhaps λ, $\varkappa$ are well chosen. [To find appropriate values of the parameters reminds tuning a Tv set with several potentiometers.]

7A. A Pascal program for BP.

The program below follows the algorithm, notation and terminology from above sections as closely as possible. All neurons are however denoted by x [with appropriate indices], input and expected output values of patterns from T are generally read out from the file 'PATTERNS', their coordinates forming vectors InPatt and OutPatt, respectively. The file starts by the indication of number of patterns (number of members in T) NoinT. The training pairs are successively copied into pattern — independent vectors Input and Out Expect, respectively. Notice that the sigmoid has been replaced out of the interval [-30,30] by constants, because the differences are then so small, that an underflow could occur.

The procedure InitNetwork chooses initial weights and all parameters referred to (η — eta, α — alpha: the momentum term is included, while adaptation of GABP is not) are as recommended above in the Summary and number of iterations Iters and cycles Cycles arbitrarily as 15 and 40, respectively. In this way the adaptation process is limited by exhausting the specified number of cycles no matter what the global error is at the end. More experienced programmers can easily adapt this criterion, stopping the learning only after this error is less then some constant, ε say.

In form of "comments" (between {and}) it also creates the file for solving a specific problem: in this case the CMN 4-2-4 is taught on identity of 4-dimensional input vectors-patterns ($\boldsymbol{y}^j = \boldsymbol{x}^j$) for a set of T, consisting of 5 pairs (although 11 out of 16 possible pairs could be principally possible to transmit over the net, the training time would be much larger to achieve approximately the same accuracy). Note the structure of the file 'PATTERNS', so that you can taylor it for another tasks. Appropriate information is included in "true" comments {! . . . !}, while your choices depend on manipulating with „program" comments {1 . . . 1} and {2 . . . 2}, which you can remove or retain according to your wish (by a Pascal editor, which is then followed by compilation and run).

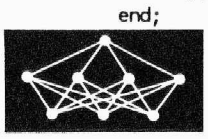
Two possibilities are offered to you; if you let the program as it is, fixed (although initially random) weights are introduced and the behavior of the network is fully deterministic to give you the possibility to repeat the same experiment many times. If you prefer to try your luck (with the random choice of weights every next run), just cross out the first pair of program parenthesis. Omitting the second pair of program parenthesis you should create your own file of PATTERNS. Modifying other information in the procedure InitNetwork, you can solve any problem you wish; section 8 will give you some suggestions.

Summarizing, removing strings {2 and 2}, you have your program ready to solve the identity problem with fixed weights; to pass to another task, you let the program parenthesis as they are and (a) modify the only task dependent procedure InitNetwork (parameters and the number of iterations and cycles and perhaps

the topology of the net), (b) create your own file 'PATTERNS' and let the rest of program untouched as far as you will stay within the ranges of constant definitions and fixed initial weights.

If you first try the built-in example, you will first see on the screen the copy of the training set, then you will be continually informed about progress in the processing. Allow about 4 minutes computation [on an IBM PC AT without numeric coprocessor] before you can check the results (the set T as well as all checks should remain within the screen). [Also, note that after the first run the file PATTERNS has been created and you can avoid creating it again and again restoring the parenthesizes {2 . . . 2}, saving thus another minute]. Observe good memorizing, but poor generalization ability (only 5 out of possible 16 members of T were used!). If you need more accuracy and/or add some other patterns, the number of cycles will have to be larger and the time would considerably increase.

```
program BackPropagation;

const MaxLayer   =  5;                              { max. number of layers }
      MaxNeuron  = 15;                { max. number of neurons in one layer }
      MaxPattern = 50;                             { max. number of patterns }

type  Layers   = 0..MaxLayer;                          { available layers  }
      Neurons  = 1..MaxNeuron;                         { available neurons }
      NeurThrs = 0..MaxNeuron;       { neurons including thresholds source }
      Patterns = 1..MaxPattern;                        { usable patterns   }
      Weights  = array [Layers,NeurThrs,Neurons] of real;
                   { Weights[i,j,k] :                                      }
                   {   if j>0 ... weight from neuron j in layer i to       }
                   {              neuron k in layer i+1                    }
                   {   if j=0 ... threshold of neuron k in layer i+1       }

var   w, wold : Weights;             { values of weights in time t and t-1 }
      x       : array [Layers,NeurThrs] of real;
                    { x[i,j] :                                             }
                    {   if j>0 ... output value of neuron j in layer i     }
                    {   if j=0 ... value -1 used as a threshold source     }
      Delta   : array [Layers,Neurons] of real;
{ Delta[i,j] = see remark after Eq.(18), concerning now neuron j in layer i }
      NoofL : Layers;                    { layers = 0 [bottom]..NoofL [top] }
      NoofN : array [Layers] of Neurons;  { number of neurons in each layer }
      NoinT : Patterns;                    { number of learning patterns    }
      InPatt, OutPatt  : array [Patterns,Neurons] of real;
                           { all input and expected output patterns from T }
      Input, OutExpect : array [Neurons] of real;
            { input and expected output pattern for one chosen pair from T }
      eta, alpha : real;          { parameters of the algorithm - see Eq.(21) }
      Iters : integer;                                 { number of iterations }
      Cycles : integer;                                   { number of cycles }

function S ( Ksi:real ) : real;           { neuron sigmoid transfer function }
const lambda = 1;                                                { sigmoid gain }
      RB     = 30;          { where to extrapolate the sigmoid by a constant }
var   inp : real;
begin inp:=lambda*Ksi;
      if inp>30 then S:=1
                else if inp<-30 then S:=0
                                else S:=1/(1+exp(-inp));
end;

procedure State;                                  { new state of the network }
var   Layer : Layers;
      j     : NeurThrs;
      k     : Neurons;
      Ksi   : real;                                          { neuron potential }
begin for j:=1 to NoofN[0] do
        x[0,j]:=Input[j];                                { set bottom layer inputs }
      for Layer:=1 to NoofL do
        for k:=1 to NoofN[Layer] do
          begin Ksi:=0;
                for j:=0 to NoofN[Layer-1] do
                  Ksi:=Ksi+w[Layer-1,j,k]*x[Layer-1,j];  { neuron potential }
                x[Layer,k]:=S(Ksi)                       { neuron output    }
          end
end;                           { x[NoofL,k] is an actual output of the network }

procedure ChangeWeights ( Layer:Layers );       { new weights for one layer }
var   j     : NeurThrs;
      k     : Neurons;
      saveW : real;
begin for k:=1 to NoofN[Layer+1] do
        for j:=0 to NoofN[Layer] do
          begin saveW:=w[Layer,j,k];
                w[Layer,j,k]:=w[Layer,j,k]-
                              eta*Delta[Layer+1,k]*x[Layer,j] +
                              alpha*(w[Layer,j,k]-Wold[Layer,j,k]);
                Wold[Layer,j,k]:= saveW;
          end;
end;

procedure MakeDelta ( Layer:Layers );          { new Delta's for one layer }
var   j, k    : Neurons;
      CumulEr : real;             { cumulative error over neurons in a layer }
begin for j:=1 to NoofN[Layer] do
        begin if Layer=NoofL                                    { top layer }
                then CumulEr:=x[NoofL,j]-OutExpect[j]
                else begin CumulEr:=0;      { calculate from previous layer }
                           for k:=1 to NoofN[Layer+1] do
                             CumulEr:=CumulEr+Delta[Layer+1,k]*w[Layer,j,k];
                     end;
              Delta[Layer,j]:=x[Layer,j]*(1-x[Layer,j])*CumulEr
        end
end;

procedure NewWeights;                                 { network new weights }
var   Layer : Layers;
begin for Layer:=NoofL-1 downto 0 do
        begin MakeDelta(Layer+1);           { set up Delta's in upper layer }
              ChangeWeights(Layer);       { calculate weights in this layer }
        end
end;

function GlobalError : real;  { global error over all layers of the network }
var   patt : Patterns;
      j    : Neurons;
      Er   : real;
begin Er:=0;
      for patt:=1 to NoinT do
        begin for j:=1 to NoofN[0] do Input[j]:=InPatt[patt,j];
              for j:=1 to NoofN[NoofL] do OutExpect[j]:=OutPatt[patt,j];
              State;
              for j:=1 to NoofN[NoofL] do
                Er:=Er+Sqr(x[NoofL,j]-OutExpect[j]);
        end;
      GlobalError:=Er;
end;

procedure Training;                    { provides learning of the patterns }
var   patt  : Patterns;
      j     : Neurons;
      Error : real;                    { cumulative error for one iteration }
      iter, cycle  : integer;
begin
   writeln;                                  { format for printed information }
   writeln('Iteration   LayerError   Pattern   Cycle   GlobalError');
   for cycle:=1 to Cycles do
     begin write(chr(13),cycle:38,GlobalError:14:5);     { prints of values }
           for patt:=1 to NoinT do
             begin write(chr(13),patt:29);
                   for j:=1 to NoofN[0] do Input[j]:=InPatt[patt,j];
                   for j:=1 to NoofN[NoofL] do OutExpect[j]:=OutPatt[patt,j];
                   for iter:=1 to Iters do
                     begin State;
                           Error:=0;
                           for j:=1 to NoofN[NoofL] do
                             Error:=Error+Sqr(x[NoofL,j]-OutExpect[j]);
                           NewWeights;
                           write(chr(13),iter:5,Error:16:5);
                     end;
             end;
     end;
   writeln(chr(13),GlobalError:52:5);
end;

procedure Testing;  { you can try how well the network is learned,          }
                    { specifying on the request one or more input vectors }
var   i : Neurons;
      c : char;
begin writeln;
      repeat write('Enter network inputs (',NoofN[0],' values) :  ');
             for i:=1 to NoofN[0] do read(Input[i]);
             readln;
             State;
             write('Output of the network is',':':9);
               for i:=1 to NoofN[NoofL] do write(x[NoofL,i]:5:2);
             write('    More testing [Y/N] ? ');
             read(c);
      until (c='N')or(c='n');
      writeln;
end;

procedure InitNetwork;     { !! network parameters initialization routine }
var   i : Layers;          { this is the only task dependent procedure !! }
      j : NeurThrs;
      k : Neurons;
      f : text;
begin NoofL:=2;             {the program will deal with the 4-2-4 network }
      NoofN[0]:=4;  NoofN[1]:=2;  NoofN[2]:=4;
      RandSeed:=3456;
{! remove the following brackets numbered 1 if you want to start always  !}
{! with new random weights; if you wish to repeat your experiments       !}
{! always with the same initialization of weights, let them be there     !}
   {1 Randomize; 1}
      for i:=0 to NoofL-1 do
        for j:=0 to NoofN[i] do
          for k:=1 to NoofN[i+1] do
            w[i,j,k]:=6*(Random-0.5)/10;
      Wold:=w;
      eta:=0.3;   alpha:= 0.7;             { choice of learning parameters }
    . Iters:= 15; Cycles:=40;  { choice of number of iterations and cycles }

{! remove brackets 2 if you do not want to create your own file of patterns}
{! according to similar template. After removing the brackets 2, you will !}
{! teach the net on identity of vertices of 4-dimensional cube as listed; !}
{! note that the file starts with the number of training pairs.           !}
```



```
    { copy patterns into file PATTERNS }
  {2 assign(f,'PATTERNS');
     rewrite(f);
     writeln(f,5);
     writeln(f,'1 1 0 0   1 1 0 0');
     writeln(f,'0 0 1 1   0 0 1 1');
     writeln(f,'1 0 1 0   1 0 1 0');
     writeln(f,'0 1 0 1   0 1 0 1');
     writeln(f,'0 0 0 0   0 0 0 0 ');
     close(f);                          2}
end;

procedure InitPatterns;                    { learning patterns init routine }
var   patt : Patterns;
      j    : Neurons;
      f    : text;
begin assign(f,'PATTERNS'); reset(f);   { using your own file of training set}
      read(f,NoinT); writeln;                          { number of patterns }
      for patt:=1 to NoinT do
        begin for j:=1 to NoofN[0] do
                begin read(f,InPatt[patt,j]);     { read inputs from PATTERNS }
                      write(InPatt[patt,j]:5:2)   { and print them on screen  }
                end;
              write('   ');
              for j:=1 to NoofN[NoofL] do
                begin read(f,OutPatt[patt,j]);  { read outputs from PATTERNS }
                      write(OutPatt[patt,j]:5:2){ and print them on screen   }
                end;
              readln(f); writeln;
        end;
      close(f)
end;

procedure InitImpl;                          { implementation init routine }
var   Layer : Layers;
begin for Layer:=0 to NoofL-1 do
        x[Layer,0]:=-1;          { used as a threshold source for next layer }
end;

begin { main program }
  InitNetwork;  InitImpl;  InitPatterns;
  Training;
  Testing;
end.  { main program }
```

[The program was written by P. Božovský]

8. BP and multilayered nets in action.

It is estimated that 95% of NN applications relies on back-propagation (BP) paradigm. He, who tried to simulate an IQ test, when you have to find (on the base of a few examples) the „natural" continuation of a given sequence of patterns, he surely admits that the generalization abilities of BP are somehow mysterious.

We will now present some ideas, how CMN under BP adaptive training could be used. They will be more schematic than elaborate, although some experience in applications will be noticed [those denoted by an asterisk * have been tried or developed in the „Czech Neurogroup" and will not be included in the bibliography; interested people can ask the author]. To be frank, almost all these applications concern „NN in small", with relatively small number of neurons and not too large amount of data. The usual *complexity barrier* known from other areas applies here as well. Also, some general remarks will extend discussions of preceding sections.

In the following figures, abbreviated forms from *Fig. 12b* will be first used and no concrete dimensions of the layers given. Vectors from the input space are denoted by oblongs. A piece of information will sometimes be specified by a curve [when we have in mind real valued coordinates and the input dimension requires an appropriate number of input coordinates to cover a possibly continuous input function with good approximation], by some 0's and 1's [indicating binary inputs] or simply by an identifier not speaking about

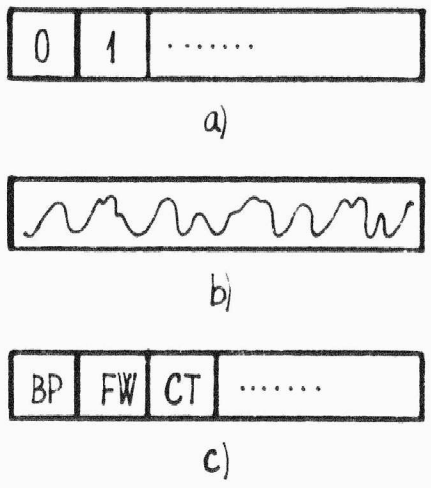


Fig. 20a, b, c

its form. Cf. *Fig. 20abc*. The problem of suitable coding of input data will be mentioned in further sections.

a) Classification / recognition.

Fig. 21 symbolizes the situation where members [$\mathbf{x}^i$, y^i] of T are submitted (according to a training strategy) to the input layer and the teacher classifies the inputs into two categories ($y^i = 0$ or 1).

A simple example of concrete 6-2-1 net has been already presented, namely in *Fig. 6*. It should only be added that the weights shown in *Fig. 6* were really obtained* (up to a multiplicative constant and approximate character of the numbers) by a BP training consisting of several hundreds repetitions of all training pairs of form [symmetric vector, 1], [nonsymmetric vector, 0]. Because the whole set T was exhausted, no test set is significant (provided the net was well adapted, as was indeed the case).

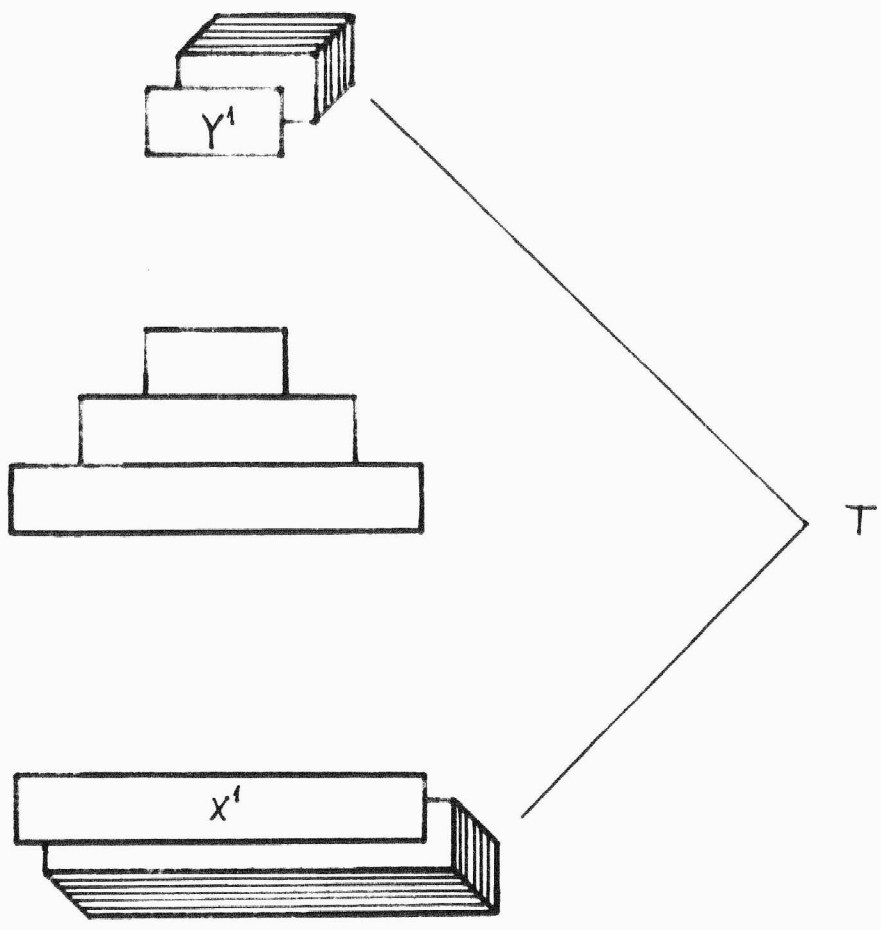


Fig. 21

There is however a good reason to think about this simple example a bit: there are many sets of weights which will perform the classifying task as well — due to many *symmetries* involved in any CMN (renumbering of hidden neurons for example, multiplying the weights — and in case of sigmoid transfer function lowering the gain λ etc), there are always *many solutions* to the given task, once there is at least one. The whole error landscape function reflects this fact (perhaps you may imagine it as a disk in which valleys and other objects are positioned symmetrically round a center and to any satisfactory place you may find many others which are as low as the first one). This ambiguity is generally an intrinsic property of NNs.

Another net we used* was a 40-17-1 net; the inputs were derived from EEG curves so that a window of 40 positions moved along the curve, multiplying thus the number of training pairs without requiring too much empirical data. The question was to classify the EEG signals into two categories: those in which a specific α-rhythm was present and the rest. For 40 members of T (not too much!), the result was impressing. For 50 questions from an independent Q (taken again from reality, so that T and Q were „similar" and no really artificial question was posed — no artifact was presented to be recognized), the net answered correctly. There was the only case of disagreement between an expert doctor and the net, in a not quite typical case; after a more detailed analysis however it turned out, that the net generalized properly and finally got the doctor's approval.

A more general system for signal analysis NESP* (NEuronal Signal processing) has been developed by Höning and Pelikán.

Surprisingly enough, NNs found a broad field of applications in (theoretical) chemistry, e.g. in helping to suggest a theory for the relation between the sequence of amino acids in the protein and the spatial arrangement of its polypeptide chain in the native state. Blazek, Pancoska, and Keiderling* used a 5-8-180 net to generate spectral curves from rentgenostrucural data and an „inverse" net 180-8-5 to obtain rentgenostructural data from spectral curves. The results were reported remarkable, hardly to reach by other methods.

Gorman and Sejnowski used 60-*-2 net (for * up to 24) for classification sonar echo signals to determine whether the signal came from a rock or a cylindrical object. Preprocessing (computing a spectral envelope) was needed; as in many similar cases.

Burr constructed a NN for handwritten character recognition as well as for spoken numeral recognition.

Kufudaki* applied neural nets to study neural processes in the real brain, namely to classicifation of time series of latent periods in conditioning and „learning curves"; a 20—12—2 net was used to solve a problem in which all tools of traditional analysis failed.

Charvat* et al used back-propagation in some stages of processing visual information mapping given areas of the Earth, taken from planes and satelites.

Jirsik, Kasik* et al used several NNs including BP to edge detection under changing light intensities.

Vingralek* used a modified BP in a metod of dithering grey images.

b) Prediction / control

Fig. 22 symbolizes a situation, in which a certain portion of history of some events ($\mathbf{x}^i$) has its continuation ($\mathbf{c}^i$) registered; for members of T this continuation was known and these $\mathbf{c}^i$'s were duplicated as answers to be learned. The training set thus consisted of members $[\mathbf{x}^i\mathbf{c}^i, \mathbf{c}^i]$. The net was thus learned to make predictions.

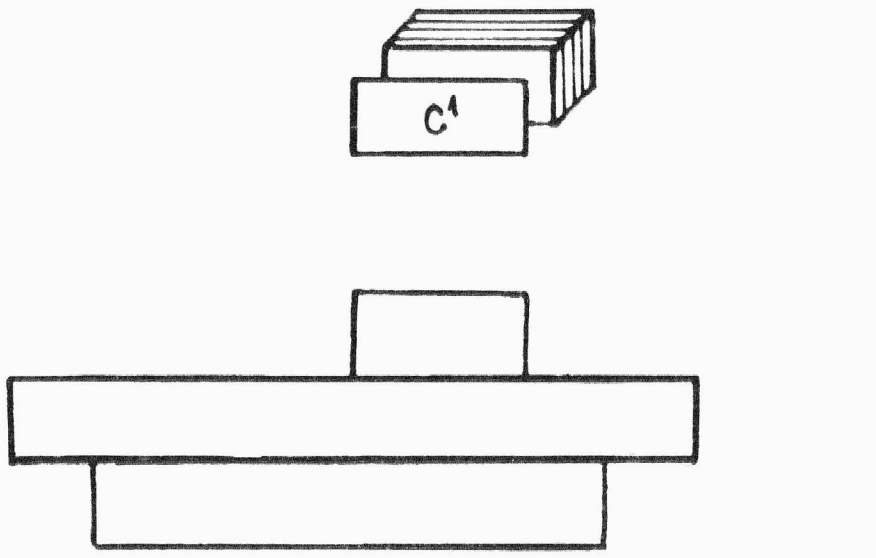


Fig. 22

A concrete example was to forecast a temperature in some given system and various training data differed by the exact place where thermometers were set. Again, it worked with about 80% reliability. Successful experiments with weather forecast were reported even in simpler second generation models.

An often cited example (not yet tried under situation in this country) is to predict behavior of a customer applying for a credit. The training set T is formed on the basis of long term history of the particular bank, which stores in its database characteristics of previous experience in the form: pertinent data about a customer [amount required, income history, frequency of job changes, minimal and maximal profit from the loans etc] — decision to grant the money or to reject the

applicant and/or to predict his/hers next fiscal behavior. In this way the net, which abstracts from such objectively hard to estimate input data as the smile of an applicant, tries to do more competent prediction of his/hers further market success.

Although economic laws are here not yet quite clear, we are* trying to simulate a simple market and the predictive capabilities on a 6-5-3-1 net, which are at least amusing; it is unbelievable how many rules is the net able to notice and extract from 12 simple examples similarly as an IQ test asks. We created 30 training examples of the form [Company (2 possibilities), Customer (4), Product (4), Price (1 real number)],[Expected success (1 real number, probability of selling P made by Co to the Cu at price P)]. Increase of price generally diminishes the success, but differently for different products; e.g. in lower price categories, Customer A prefers Company 1, is somehow richer (can afford more expensive goods) etc. These and other "laws" are deducible from the examples explicitly, but the net (in the role of an advisor) estimates the success of business transaction even in cases which bear no direct relationship to the examples; sometimes you have to note less apparent dependencies to explain the „decision" of the net.

A specific example of prediction is that of *time series.* Weigend, Rumelhart and Huberman applied a 12-8-1 net on a (theoretically infinite) sequence of numbers $\ldots, x_{T+2}, x_{T+1}, x_{T}, \ldots, x_1$, where the twelfth inputs from the past were trained to predict next member of the sequence until they came to the possibility to predict x_T. The empirical material was drawn out of historically long observations of occurrences of sunspots. BP has therefore the ability to discover well-hidden forms of regularities over a great period of time. Again, attempts to predict workers productivity etc. has been applied.

Recall *Fig. 11* as an example of *control.*

c) Data compression.

Fig. 23a shows a net *m-k-m* (generally there can be other hidden layers involved), where $k < m$. It is trained to realize an identity mapping ($\mathbf{y}^i = \mathbf{x}^i$). We already know from sect.5 that not all identity mappings are realizable by such a net (cf. *Fig. 17* and its explanation). This however does not prevent us to try and to be successful in a development of a net, which well implements to transmit a subset J of the input space, especially if there are enough regularities (inner laws) that govern membership to such a subset J. Even in such cases like natural language, radio or Tv signal transmission. If the net happens to find out these regularities and is successfully trained, then the *m*-dimensional information can pass through less-dimensional bottleneck of the narrow hidden layer. *Fig. 23b* then shows how the situation can be utilized: Split the well learnt net into two parts preserving the weights of original net and let the lower part play the role of a transmitter, the upper part the role of a receiver, which can be in space far away; and connect the hidden layer neurons (two in *Fig.23b*) by lines which identify the states of corresponding neurons. The original input information can then be transmitted to a distant place using less „wires" (here *k*) than will be necessary to connect all the m input/output neurons directly.

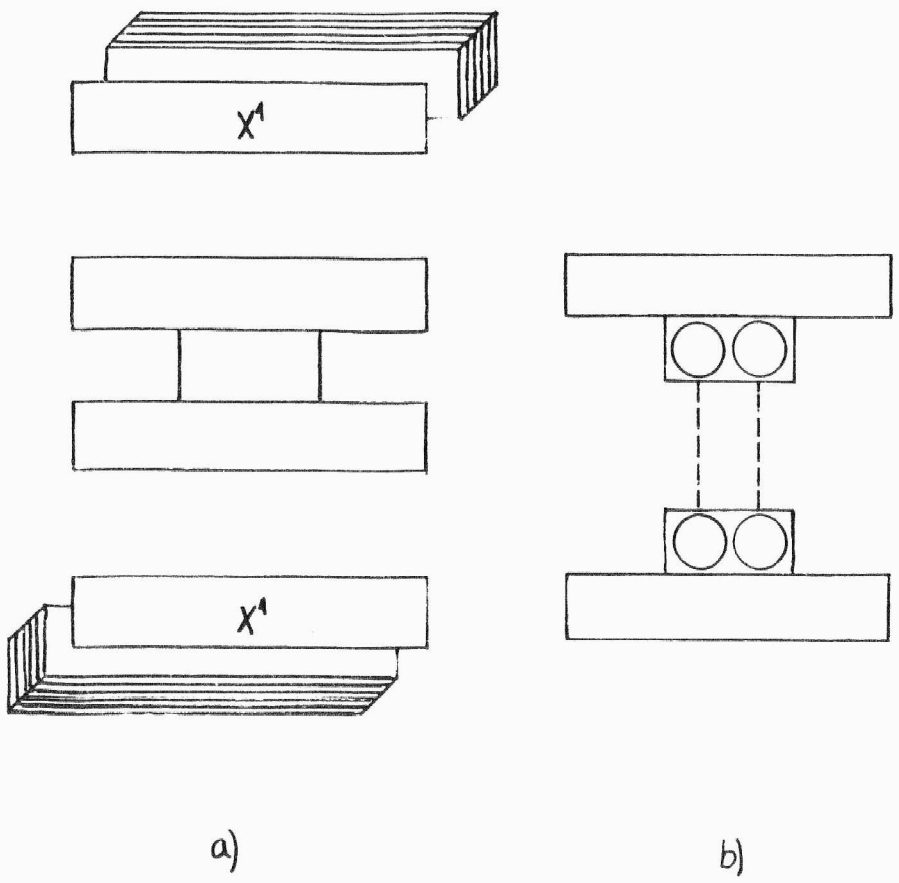


Fig.23ab identita

Cottrell, Munro and Zipser used a 64-16-64 net to image compression in which the squares of 8×8 pixels were approximated by squares of 4×4 pixels, not loosing too much visual information.

(Continuation)

Literature Survey

Saito T.: An Approach Toward Higher Dimensional Hysteresis Chaos
IEEE Transactions on Circuit and Systems, Vol. 37, 1990, No. 3, pp. 399—409

Simic P. D.: Statistical Mechanics as the underlying theory of „elastic" and „neural" optimisations
NETWORK, Vol. 1, 1990, No. 1, pp. 89—103
Key words: application.

Specht D. F.: Probabilistic Neural Networks
Neural Networks, Vol. 3, 1990, No. 1, pp. 109—118
Key words: probability density function; parallel processor; pattern recognition; parzen window; bayes strategy; associative memory.
Abstract: By replacing the sigmoid activation function often used in neural networks with an exponential function, a probabilistic neural network (PNN) that can compute nonlinear decision boundaries which approach the Bayes optimal is formed. Alternate activation functions having similar properties are also discussed.

Instructions to authors

1. Manuscript
Three copies of the manuscript should be submitted to the Editor-in-Chief

2. Copyright
Original papers (not published or not simultaneously submitted to another journal) will be reviewed. Copyright for published papers will be vested in the publisher.

3. Language
Manuscripts must be submitted in English

4. Text
Text (articles, notes, questions or replies) double space on one side of the sheet only, with a margin of at least 5 cm, (2") on the left. Any sheet must contain part or all of one article only. Good office duplication copies are acceptable. Titles of chapters and paragraphs should appear clearly distinguished from the text.
Complete text record on 5 1/4" floppy discs is required.

5. **Equations**
Mathematical equations inserted in the text must be clearly formulated in such a manner that there can be no possible doubt about meaning of the symbols employed.

6. Figures
The figures, if any, must be drawn on separate sheets. They must be clearly numbered and their position in the text marked. They are to be drawn in Indian ink on white paper or tracing peper, bearing in mind that they will be reduced to a width of either 7,5 or 15 (3 or 6") for printing. After scaling down, the normal lines ought to have a minimum thickness of 0,1 mm and maximum of 0,3 mm while lines for which emphasis is wanted can reach a maximum thickness of 0,5 mm. Labelling of the figures must be easy legible after reduction. It will be as far as posible placed across the width of the diagram from left to right. The height of the characters after scaling down must not be less than 1 mm. Photographs for insertion in the text will be well defined and printed on glossy white paper, and will be scaled down for printing to a width of 7,5 to 15 cm (3 to 6"). All markings on photographs are covered by the same recommendations as for figures. It is recommended that authors of cummunications accompany each figure or photograph with a descriptive title giving sufficent information on the content of the picture.

7. Tables
Tables of characteristics or values inserted in the text or appended to the article must be prepared in a clear manner, preferably as Camera Ready text. Should a table need several pages these must be kept together by sticking or other appropriate means in such a way as to emphasize the unity of the table.

8. Summaries
A summary of 10 to 20 typed lines written by the author in the English will precede and introduce each article.

9. Required information
Provide title, authors, affiliation, data of dispatch and a 100 to 250 word abstract on a separate sheet. Provide a separate sheet with exact mailing address for correspondence.

10. Reference
References must be listed alphabetically by the surname of the first author. List author(s) (with surname first), title, journal name, volume, year, pages for journal references, and authors(s), title, city, publisher, and year for the book references. Examples for article and book respectively:

Dawes, R. M. and Corrigan, B.: Linear models in decision making. Psychological Bulletin, Vol. **81**, 1974, 95—106

Brown, R. G.: Statistical Forecasting for Inventory Control, New York, McGraw-Hill, 1959.

All references should be indicated in the manuscript by the author's surname followed by the year of publication (e. g., Brown, 1959).

11. Reprints
Each author will receive 25 free reprints of his article.

PD 3818

This is Seagate Technology.

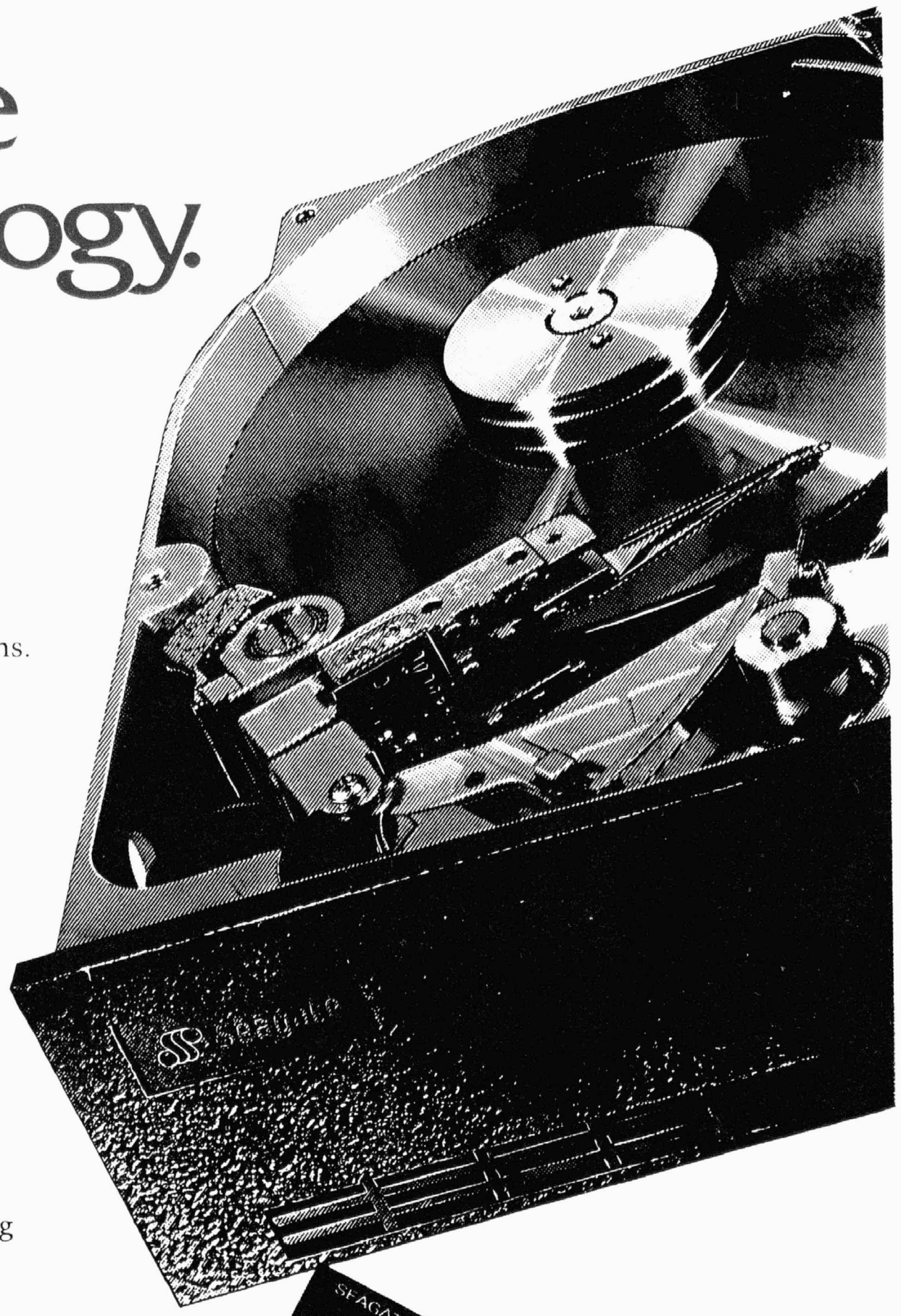
Seagate's line of hard disc drives is packed with high technology. And every one is built to the highest quality and reliability standards in the industry.

And now, Seagate drives are available locally for all your Personal Computer applications.

Only Seagate can offer you full technical support, and a one-year warranty. through our authorised representatives in your country.

Complete technical and interface details are included in the Seagate product brochures, which are free of charge to professional PC buyers and users. Simply use the coupon below to request your copies.

You'll soon see why Seagate has become the world's leading independent manufacturer of disc drives.



To: Seagate Technology Europe,
Seagate House, Fieldhouse Lane,
Globe Park, Marlow SL7 1LW Great Britain.

Please send me technical details of Seagate disc drives

Name _____

Job Title _____

Organisation _____

Address _____

_____

Country _____

Type of business _____

Number of employees _____ Number of PCs _____

☐ I use a PC ☐ I authorise the purchase of PCs

☐ I am a technical support manager

Seagate

Seagate Technology Europe
Seagate House, Fieldhouse Lane, Globe Park, Marlow SL7 1LW Great Britain.
Tel: 0628 890366 Fax: 0628 890660 Telex: 846218 SEAGAT G