Chapter 13

Neural Knowledge Processing in Expert Systems

JIŘÍ ŠÍMA[1] & JIŘÍ ČERVENKA
Department of Theoretical Informatics
Institute of Computer Science
Academy of Sciences of the Czech Republic
sima@uivt.cas.cz

Abstract

In this chapter the knowledge-based neurocomputing will be applied to expert systems. Two main approaches to represent the knowledge base, namely the explicit and implicit representations will first be introduced and compared in rule-based and neural expert systems, respectively. Then, several possible integration strategies that make an effort to eliminate the drawbacks of both approaches in hybrid systems, will be surveyed. To illustrate the full power of knowledge-based neurocomputing, the main ideas of the prototypical, strictly neural expert system MACIE will be sketched. Here, a neural network is enriched by other functionalities to achieve all required features of expert systems. The neural knowledge processing will further be demonstrated on the system EXPSYS which exploits the powerful back-propagation learning to automatically create the knowledge base. In addition, EXPSYS introduces the interval neuron states to cope with incomplete information and it provides a simple explanation of the results. The chapter concludes with an EXPSYS application to a toy economic problem which is compared to a parallel conventional rule-based solution.

13.1 Knowledge Representation in Expert Systems

In this section we will discuss the difference between explicit and implicit representation of knowledge in expert systems. We will mainly focus on two typical representatives

[1]This research was supported by GA ČR Grant No. 201/98/0717 and partially by grants MŠMT ČR OK–304 and INCO-COOP 96–0195.

of these different approaches. Namely, we will compare rule-based systems and neural networks. To make our exposition self-contained we will first recall some basic notions from expert systems (Biondo, 1990; Giarratano and Riley, 1993; Ignizio, 1991; Jackson, 1990) to motivate representation issues.

13.1.1 Expert Systems

13.1.1.1 Definition and Requirements

The representation of large amounts of knowledge which would ensure their integrity, consistency, and effective exploitation is one of the main issues of artificial intelligence. For this purpose expert systems have been proposed to manage knowledge processing. An *expert system* is a computer program that performs a complex decision-making task within a particular narrow *problem domain* that is normally done by a human expert. It is based on the idea of taking on the knowledge from a specialist and expressing it in an appropriate representation to exploit the knowledge in the same way as the human expert does and above all with the same result. The expert system can replace an expert who can be too expensive and sometimes even not available to advise, analyze, consult, diagnose, explain, forecast, justify, monitor, plan, etc.

A typical example of the above introduced programs is a diagnostic medical expert system. Its domain is usually confined to a particular area of medical expertise handled by a physician—a specialist, e. g. an internist. The system can help to diagnose a subclass of diseases and possibly advise the manner of their treatment. In the sequel we will use medical examples to illustrate some key concepts of expert system design. However, it must be stated here that for the sake of clarity of exposition all these examples will be oversimplified from a medical point of view. Furthermore, in Paragraph 13.3.5, we will present an expert system application for a real-world economic problem which will be described in more detail.

Generally, the expert system is required to posses the following functional features:

- A user of the expert system usually consults her/his particular problem in the interactive mode. The system builds an internal model of this case and asks questions to acquire new information which is mostly relevant for solving the task. In our example of the diagnostic medical expert system, the user first presents basic information about a patient's condition to the program, e. g. the apparent symptoms of patient's complaints. Then the system asks additional questions or recommends relevant medical examinations to improve its internal model of the patient's condition which is finally used to infer a diagnosis. The system should ask its questions in the order of their relevance for the case being examined, e. g. it should not require all patients to be X-rayed first.

- The expert system should infer a solution for the problem even from incomplete or imprecise information, e. g. when the user replies "I don't know" to some of the questions or when some numerical values are only determined within intervals, etc. The conclusions of the program are usually evaluated by percentual confidences which measure the credibility of a solution. For the medical expert system, some information on patient's anamnesis, symptoms or results of laboratory analyses may not be available, e. g. because of patient's inability to describe her/his symptoms, lack of time to carry out a particular test, insufficient laboratory equipment, contra-indications that disable required examination etc. In spite of this, the system should always infer some diagnosis and compute its confidence which obviously also depends on the input completeness and accuracy. For example, the medical system infers the diagnosis of hepatitis with a 58% confidence having been told that the patient has yellow skin and the results of urine and blood tests (for bilirubin) recommended by the system are unknown. After providing the program with these results the diagnosis confidence can either increase, e. g. up to 95%, or hepatitis is excluded.

- The expert system should also provide the user with some explanation of its conclusions to justify the inference and to give the user insights into the problem or to avoid fatal errors caused by the computer solution. The system usually exhibits how the conclusions have been reached by a sequence of internal logical steps which lead to its results. For example, the user of the medical system can ask why the program recommends, with high confidence, to take out the appendix by an operation. The system explains this conclusion by the acute appendicitis which has been inferred by a typical stabbing abdomen pain.

13.1.1.2 Modular Architecture

In Paragraph 13.1.1.1, the definition of the expert system as well as the requirements for its functional features have been presented from the outside point of view. Now, we will turn to the description of the internal modular structure of expert systems. Of course, the expert system is a computer program and therefore its architecture can differ as the case may be. However, the following major parts of the expert system are usually distinguishable:

- The **knowledge base** is a typically large static data structure which contains all important information about the problem domain, i. e. knowledge from the area of program expertise, and thus, it is problem-oriented. This base models expert's knowledge which may not only be exact but it can also be intuitive, imprecise, incomplete, and even questionable or contradictory. The way in which to represent such a knowledge base in a computer framework is the main interest of this chapter. For example, the knowledge base of the medical expert system consists of physician–specialist's knowledge. This includes e. g. sets of symptoms and characteristics indicating particular diseases together with the procedures of their differential diagnostics and therapeutical schemes for their treatment. Of course, this knowledge comprises not only known facts about diseases but also subtle techniques of their diagnostics gained over long time by practical experience. Therefore, it is difficult to express in a computer-acceptable representation.

- The **factual base** is a typically small dynamic data structure which contains information about a particular problem which is currently being solved by the system. It includes the input facts provided by the user as well as the solutions of subproblems or partial solutions, i. e. the corresponding inferred conclusions with confidence concerning the considered task. Thus, the factual base serves as an internal model of the case at hand which is built by the system to solve this problem. The manner of representing facts in this base is usually the same or at least consistent with the knowledge base representation. In our medical system example, the factual base contains information about a patient's condition and her/his diagnosis with the confidence being inferred so far.

- The **inference engine** is a controlling procedural part of the expert system. It consists of cooperating programs which establish the general mechanism of reasoning and inference. The inference engine exploits knowledge from the area of expertise which is stored in the knowledge base to build an internal model of the currently solved problem in the factual base. Besides the proper inference, which developes the model internally, this includes collecting relevant input facts in the factual base externally from the user. This means that the inference engine determines the appropriate questions which the user is asked, to obtain relevant information about the case at hand. This information should mostly complete the model and consequently, increase the conclusion confidence. Furthermore, the inference engine is problem-independent, i. e. it implements a general inference algorithm which should still work for other knowledge bases from different problem domains providing that their representation format is preserved. The expert system shell which does not contain a particular knowledge base, is called the *empty expert system*. For example, the inference engine of the diagnostic medical expert system can be exploited for the diagnostics of a nuclear power plant operation after a corresponding knowledge base is created.

- The **explanation mechanism** follows the operation of the inference engine and exploits it to explain a particular conclusion when it is asked.

- The **user interface** of the expert system links the inference engine to the external environment, i. e. it collects and preprocesses information about the problem at hand and outputs the results.

13.1.2 Explicit Knowledge Representation and Rule-Based Systems

The most difficult part of building the expert system is the construction of its knowledge base. This task is usually done by *knowledge engineers* who cooperate with specialists from the problem domain to acquire relevant knowledge. In a conventional approach the representation of this information is based on symbolic expression of human expert's knowledge, skills, and experience. This is called the *explicit knowledge representation* because the expert knowledge is formulated in a comprehensible way. Typically, the knowledge representation is specified by a data format in which the information is expressed to be stored in the knowledge base. This format should handle the natural imprecise and incomplete character of the expert knowledge. The knowledge base representation also determines the way of how information is processed by the inference engine. However, the specialist is usually unable to transform her/his knowledge immediately to a prescribed formalism as well as the explicit representation of the knowledge base may not be entirely adequate for all types of expert knowledge. Therefore, the knowledge engineer spends a lot of effort and time by consulting the expert to debug the knowledge base.

There are many ways of how to represent the expert knowledge explicitly. Traditional expert systems generally employ the so-called *IF–THEN rules* to represent this information and hence, these programs are called *rule-based systems*. For example, the most famous expert system MYCIN (Buchanan and Shortliffe, 1984) which diagnoses microbial diseases of blood is also based on rules. The rules usually have the following form:

```
IF <condition> THEN <conclusion> (<confidence>)
```

The condition is typically a logical expression which contains relevant variables whose values can be inferred from the factual base or are acquired from the user. The conclusion determines the new value of some variable providing that the corresponding condition is satisfied. The probabilistic nature of the rule is expressed by its percentual confidence. For example, the knowledge base of our hypothetical medical system can include the following rule:

```
IF (skin-color=yellow) and (bilirubin-blood-tests=true)
   THEN hepatitis=true (0.9)
```

which means that the diagnosis hepatitis is concluded with 90% confidence if the patient's skin is yellow and her/his blood tests are positive.

The inference engine usually examines several rules to satisfy their conditions starting with those rules whose conclusions are of main interest to the user. If a condition of some rule is fulfilled (i. e., the values of relevant variables in the factual base meet the condition with sufficiently high confidences), then this rule is applied so that a new value of the variable from the conclusion can be inferred and its confidence computed. This information is stored in the factual base. The conclusion confidence depends on the confidences of facts which are tested in the condition and on the rule confidence itself. The confidence manipulations are based upon fuzzy logic. In the above-mentioned example of the IF–THEN rule, suppose that the confidence of the fact that a patient has yellow skin is 0.8 and that her/his bilirubin blood tests are positive with the confidence 0.95. Then the conjunction of the condition is satisfied with the confidence which is determined as the minimum of these confidences, i. e. $\min(0.8, 0.95) = 0.8$. The resulting confidence of the diagnosis of hepatitis is computed as the product of this value and the confidence of the rule, i. e. $0.8 \times 0.9 = 0.72$.

In order to determine whether the conditions are satisfied, the inference engine recursively examines those rules whose conclusions affect the conditions of the current rules being checked. If such rules are not found, the user is asked about relevant facts. Note that the order in which the rules are examined is given by the recursive calls. This inference engine strategy is called *backward chaining* which is, for example, employed in the above-mentioned system MYCIN. An alternative approach is *forward chaining* in which the values of initially known variables are being completed by applying those rules whose conditions have already been satisfied or by asking the user questions if there are no such rules. This approach is employed in the commercially successful expert system R1/XCON (McDermott, 1981) which helps to configure computers. Generally, it is obvious that the rule-based systems can be directly implemented in the programming languages like Prolog, LISP, etc.

Besides the IF–THEN rules, it is possible to represent the expert knowledge in the so-called inference networks which are, for example, used in the system Prospector (Duda and Reboh, 1983) for geological exploration or in the empty expert system FEL–EXPERT (Mařík et al., 1990; Mařík et al., 1992; Mařík et al., 1993). Furthermore, the well-known Internist–I (Miller et al., 1982) which is one of the largest medical expert systems makes use of tables with probability–related information. The probability calculations in these systems were motivated by the Bayesian theory.

13.1.3 Implicit Knowledge Representation and Neural Networks

In Paragraph 13.1.2, we have discussed the explicit knowledge representation. However, the knowledge can often hardly be expressed in some computer formalism. It is a case when the specialist is unable to formulate her/his knowledge and experience using rules because, in fact, human experts do not usually apply formal logic to each situation, but rather they associate the new case with some old pattern to derive a solution. Moreover, some problem domains naturally include numerical data instead of rules, e. g. visual images, signals, etc. This data requires additional processing to be really understood and thus, they represent the implicit knowledge. Within this context, when one tries to represent a knowledge base explicitly (e. g., by rules) a new rather inadequate representation is created. This sterile representation may not preserve or can even violate the original knowledge in such a way that the inference engine fails to infer correct conclusions from the knowledge base.

A more natural approach suggests to model the manner of how the knowledge is represented in an expert's brain directly and to avoid a further knowledge transformation into an inadequate explicit format which could deform the original information. On the other hand, one can argue that the human brain is so complex that no one really understands how information is stored there. However, the simulations of even very simplified mathematical models of neural networks exhibit surprisingly "intelligent" behavior similar to human intelligence, e. g. the ability to learn new knowledge and to generalize previous experience. Therefore, the so-called *neural expert systems* (*connectionist expert systems* or briefly, *expert networks*) (Gallant, 1993; Mitra and Pal, 1995; Šíma, 1995) in which the knowledge base is implemented by a neural network present a promising alternative to rule-based systems. In what follows we will recall basic notions from the neural network theory (Fausett, 1994; Haykin, 1994; Hecht-Nielsen, 1990; Hertz et al., 1991; Lippmann, 1987; Rojas, 1996; Simpson, 1990) which will be used later in technical parts of this chapter. On the other hand, the reader's experience with neural networks would be helpful for understanding all details.

Neural networks are computational models which have been inspired by neurophysiology. They consist of a multitude of simple units—the so-called *neurons* which are densely interconnected. Every inter-neuron connection in the network is associated with a numerical *weight*. First, we will describe the function of a single neuron. The neuron j collects its real *inputs* from the *outputs* y_i of incident neurons $i \in j_{\leftarrow}$ where $j_{\leftarrow}$ denotes the set of all neurons which are connected to the neuron j. Let these connections be labeled with the real weights w_{ji} for $i \in j_{\leftarrow}$. Moreover, denote by w_{j0} the so-called *bias* of the neuron j which can be viewed as the weight of a formal input $y_0 = 1$ whose value is constantly 1. Then, the so-called *excitation level* ξ_j of the neuron j is computed as the weighted sum of its inputs:

$$\xi_j = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} y_i \,. \tag{13.1}$$

The *state* (i. e., the output) y_j of the neuron j is determined from its excitation level ξ_j by applying an *activation function* σ as follows:

$$y_j = \sigma(\xi_j) \tag{13.2}$$

where

$$\sigma(\xi) = \begin{cases} -1 & \text{for } \xi < 0 \\ 1 & \text{for } \xi \geq 0 \end{cases} \tag{13.3}$$

is the *hard limiter*.

Furthermore, we will restrict ourselves to the most widely used neural network architecture—the *feedforward neural network* which is briefly outlined in the following. In this network, neurons can be grouped into a sequence of layers in such a way that neurons in one layer are connected only to neurons in subsequent layers. The first so-called *input layer* which consists of n *input neurons* serves as the input for the network while the last so-called *output layer* composed of m *output* neurons is used for the output. The intermediate layers are called *hidden layers* and they include *hidden neurons*. The computation proceeds from the input layer via hidden layers up to the output layer. At the beginning the states of the input neurons are set to the input of the network. In a general step, suppose that all neuron outputs are determined up to a certain layer, then the states of neurons in the next layer are computed according to (13.1), (13.2). At the end the states of output neurons represent the output of the network.

It is clear that the function $\mathbf{y}(\mathbf{w}) : \Re^n \longrightarrow \Re^m$ computed by a neural network (briefly, the *network function*) is parametrized by the vector $\mathbf{w}$ of all its weights which is called the *configuration* of the network. Neural networks learn this function (i. e., are "programmed") from example data—the so-called *training patterns*. A training pattern is a pair $(\mathbf{x}, \mathbf{d})$ of a sample input $\mathbf{x} \in \Re^n$ with the corresponding desired output $\mathbf{d} \in \Re^m$. The patterns create the so-called *training set*

$$T = \{(\mathbf{x}_k, \mathbf{d}_k)\,;\, \mathbf{x}_k \in \Re^n,\, \mathbf{d}_k \in \Re^m,\, k = 1, \ldots, p\} \tag{13.4}$$

which is used during the learning phase to adapt the configuration automatically in such a way that the new network function is consistent with T, i. e. $\mathbf{y}(\mathbf{w}, \mathbf{x}_k) = \mathbf{d}_k$ for $k = 1, \ldots, p$. In addition, the network function should generalize the implicit rules from the training set and respond reasonably to previously unseen inputs.

There are many learning rules and heuristics used in neural networks. They usually minimize an error between the actual network function and the desired behavior which is specified by a training set. Given a training set (13.4), the error $E(\mathbf{w})$ of the network function $\mathbf{y}$ with respect to T is defined as the function of $\mathbf{w}$ in the following way:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{p} \sum_{j \text{ output}} (y_j(\mathbf{w}, \mathbf{x}_k) - d_{kj})^2 \tag{13.5}$$

where $y_j(\mathbf{w}, \mathbf{x}_k)$ is the actual state of the output neuron j for the input $\mathbf{x}_k$ (and the configuration $\mathbf{w}$) while d_{kj} is the corresponding desired value. The task of the learning algorithm is to minimize the error (13.5) in the configuration space, i. e. to find $\mathbf{w}$ which minimizes $E(\mathbf{w})$. This represents a non-trivial optimization task because the error includes the complex non-linear network function. Gradient methods can be applied here providing that the gradient of the error function $E(\mathbf{w})$ can be computed. For this purpose, the hard limiter (13.3) is approximated with a differentiable activation function, e. g. hyperbolic tangent:

$$\sigma(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}\,. \tag{13.6}$$

The gradient computation can be achieved by the well-known *back-propagation* algorithm (Rumelhart et al., 1986). However, the whole learning procedure is a very complex time-consuming optimization process in which the implicit knowledge contained in the training set is automatically learnt and represented by weight parameters.

As we have mentioned above, to perform the desired task the network function should not only memorize the training set but it should generalize the rules from the training set in order to be able to derive solutions for similar inputs. It is known that the generalization ability of the network depends on the *architecture* (*topology*), i. e. on how many neurons and how they are connected in the network which also determines the dimension of configuration. If the architecture is rich enough, then the network can memorize the training set without problems, however, it may respond incorrectly to previously unseen inputs. This is called *overfitting*. On the other hand, poor architectures are probably too weak to solve complex tasks. It has been experimentally confirmed that, for a particular task, there exists an "optimal" architecture which is strong enough to solve the problem and, at the same time, it generalizes well. This architecture is usually searched in such a way that different network architectures are adapted to the training set. Then, these networks are tested using the so-called *test set* which is a part of the training set that is not used for learning. The architecture whose function is the most consistent with the test set is chosen to perform the task.

In neural expert systems, the user interface encodes the inputs and outputs of the system by the states of the input and output neurons and the knowledge base is implemented by a neural network configuration. This means that instead of elaborating artificial rules with the aid of an expert, the neural network is automatically adapted to example inferences during a learning phase and the network function generalizes implicit rules from the training set. The set of example inferences which is encoded into a training set can be generated by observing expert's solutions to the problem. In our medical example the clinical records about patients can be exploited for this purpose where symptoms and test results in each case present a sample input and the corresponding diagnosis and treatment recommended by a physician is the desired output. For example, we can define the following inputs and outputs of the medical system:

```
INPUTS:  TEMPERATURE,ANOREXIA,CHOLESTEROL,SKIN-COLOR,SCLERAE-COLOR,
         ...,LIVER-TENDERNESS,BLOOD-TEST,URINE-TEST
OUTPUTS: DIAGNOSIS,HOSPITALIZE,...
```

Then, an example of such inference pattern can look like:

```
[(38.2,YES,NORMAL,YELLOW,YELLOW,...
   ...,HIGH,unknown,{BILIRUBIN,UROBILINOGEN}),
     (HEPATITIS,YES,...)]
```

Thus, after the network is learnt, the expert knowledge is distributed in numerical weights throughout the network without the possibility of identifying the purpose of a single weight or neuron within this representation. This is called the *implicit knowledge representation* because the expert knowledge cannot be easily extracted from the knowledge base without additional processing.

Furthermore, the inference engine in neural expert systems collects all inputs for the neural network. Then it evaluates the network function using (13.1), (13.2) for these inputs and obtains the corresponding outputs from which the conclusions are decoded. It is clear that, during the network computation, the neuron states implement the factual base.

13.1.4 Comparison of Rule-Based and Neural Expert Systems

In Paragraphs 13.1.2, 13.1.3, two different approaches to representing the knowledge base in expert systems have been introduced, namely rule-based and neural expert systems. Now, we will compare these systems from various points of view.

13.1.4.1 Task Size

The size of a knowledge (and even factual) base and its complexity is not limited in rule-based systems. It can include plenty of rules which serve as a way to solve very large and complex problems. The system can generate many questions including irrelevant ones which can disturb the user. An example of a medical system is known which always started its consultation with the question "Is the patient alive?". Furthermore, the large rule-based system can create a lot of various auxiliary statements and can reach many different output hypotheses.

On the other hand, it has been proven (Šíma, 1996) even for a very simple fixed feedforward architecture that the back-propagation learning is an NP-hard problem. Therefore, learning larger tasks can be intractable in neural expert systems. Practical experiments approved that the learning process is very time-consuming (even weeks of PC computation) and it is possible to manage tasks only up to hundreds of neurons and training patterns. Therefore, the size of a neural knowledge base is limited and neural expert systems are suitable for partial subproblems with a very restricted problem domain. They can be linked in a hierarchy to handle larger tasks.

13.1.4.2 Knowledge Acquisition and Editing

The main difference of the rule-based and neural approach to expert system design consists in knowledge acquisition. In rule-based systems, a knowledge engineer together with an expert from a problem domain formulates the particular rules, evaluates them with confidence and debugs the resulting knowledge base (e. g., appropriate order of rules and confidence refinement) to achieve a reasonable performance of the inference engine. This is a very time-consuming process (even years) and a satisfying result is not guaranteed. Moreover, further editing can cause side-effects. The explicit knowledge representation in these systems confines their applicability to the problems which are sufficiently understood and where the rules are available and adequate to solve the task.

In neural expert systems, the expert knowledge is implicitly included in the training set which consists of example inferences and an expert is generally not necessary to create the neural knowledge base. The training set should include a reasonable number (due to learning time) of representative patterns which sufficiently cover the problem domain. Their combinations and generalization is left for the learning phase. Therefore, the neural approach is more suitable for problems in which a data set is available and where there is a lack of rules. It is also possible to select automatically the representative patterns from a large database by using cluster analysis. On the other hand, it is better to apply the rule-based system than try to translate a complete set of rules into a training set because one rule can cover an unmanageable amount of data. The main advantage of the neural approach is that the knowledge base, which is implemented by a neural network, is created automatically by a learning algorithm after the training set is prepared. Therefore, building a neural expert system takes only a few weeks or months. Moreover, the editing of a neural knowledge base can easily be achieved by additional learning.

13.1.4.3 Partial Matching

The problem of *partial matching* appears in rule-based systems when no condition of all rules in the knowledge base is entirely satisfied in the factual base but the conditions match the facts only partially. The inference engine is sometimes able to exploit the partially satisfied rules to infer reasonable conclusions by using e. g. fuzzy logic. In neural expert systems, this problem corresponds to the situation when the presented input for the network has not occurred in the training set. Supposing this input is from the problem domain which was covered sufficiently by the training set and the neural network generalizes well, then the conclusion provided by the network is a good solution for the problem at hand. Even more, we can state that neural networks are especially suitable for partial matching.

13.1.4.4 Incomplete Information

Incomplete or even imprecise input information is handled in rule-based systems by using confidence calculations and fuzzy logic. The system builds gradually the internal model of the case being currently solved on the basis of a possibly incomplete input information which is being step-by-step acquired from the user and it provides partial hypotheses and conclusions. On the other hand, in neural expert systems, all network inputs (including unknown and irrelevant inputs) have to be specified to compute the network function, i. e. to perform the inference and to get the relevant solution of the case at hand. This makes operating with incomplete information difficult or even impossible (Drucker, 1990). Sometimes, the input and output neuron state can be interpreted as confidence in the corresponding facts to cope with imprecise information in neural networks. However, this should be taken into account in the training set to let the network generalize the confidence calculations during learning.

13.1.4.5 Explanation Capabilities

Because of the explicit knowledge representation the rule-based systems have perfect explanation capabilities. It is sufficient to display the process how the inference engine has reached the conclusions which is transparent and comprehensible due to the explicitness of its logical steps. On the other hand, the implicit knowledge base representation in neural expert systems makes such an explanation difficult or even impossible. The expert knowledge is distributed in the network configuration and the neural network is used only as a "black box" to compute its outputs for given inputs without justifying these conclusions.

As it can be seen from the preceding comparison of rule-based and neural expert systems their advantages and disadvantages are complementary. Because of the explicit knowledge representation, knowledge acquisition, knowledge base creation, and its debugging in rule-based systems are difficult and time-consuming tasks which are usually done manually. On the other hand, for the same reason these systems work well with incomplete information and provide excellent justification of inference. The opposite is true for a neural expert system whose knowledge base, despite its limited size, is created automatically by learning from example inferences in a relatively short time. However, these systems do not make the inference from incomplete information easily possible and have little or no explanation capability.

13.2 Neural Networks in Expert Systems

In this section we will briefly survey several possible integration strategies of explicit and neural knowledge representations in hybrid systems. Then we will focus on a strictly neural approach to illustrate information processing within the framework of the knowledge-based neurocomputing paradigm. Some applications of neural expert systems will be mentioned.

13.2.1 Hybrid Systems

The advantages and disadvantages of explicit and implicit knowledge base representations which have been demonstrated by rule-based and neural expert systems in Section 13.1 are complementary. Therefore, there is a natural tendency to integrate the advantages of both representation approaches in hybrid systems. The *hybrid systems* attempt to integrate the explicit knowledge with neural networks to cope simultaneously with different types of information. In the following various integration strategies (Caudill, 1991) will be discussed and illustrated by examples of existing systems.

13.2.1.1 Divide and Conquer

One of the simplest ways of combining neural networks and rule-based systems is a *divide-and-conquer* strategy. A large problem is broken into pieces which are solved separately by using the most appropriate methods. A subtask in which rules are known and justification of inference is required, is suitable for a rule-based solution. The other part of the problem where only data is available, can be processed by neural network technology. Sometimes, a neural solution does not even need any explanation because the solution itself gives reasons for the conclusions, e. g. the optimization process is justified by its result. Thus, in the modular expert system architecture, the problem is first partitioned, i. e. the inputs are distributed to respective rule-based and neural network modules. Then these modules work in parallel to solve relevant subtasks. Finally, their solutions are collected to provide the user with resulting conclusions.

For example, in daily scheduling large numbers of delivery trucks it must be determined what packages should be combined on each truck and what the most efficient route is for that truck to follow for all its stops. This complex problem can be naturally divided into two tasks: grouping the packages into trucks which can be solved by rules and route scheduling where an optimizing neural network is employed to find an efficient path for each truck. The final system which has been proposed by researchers Bigus and Goolsbey (Caudill, 1991) works very well indeed.

13.2.1.2 Embedded Neural Network

Another method of implicit and explicit knowledge integration is to make the neural network a part of the rule-based system. For example, the network can evaluate conditions of some rules. This approach is especially suitable for pattern matching, e. g. for visual or signal inputs, etc. where logical expressions cannot be applied. Thus, the neural network is employed for a rule matching process to decide whether a rule is applicable for the case at hand. Similarly, the applications of some rules can be executed by neural networks. The network can develop the internal model of the currently solved problem by inferring new facts and re-computing the confidences in the factual base or simply by performing the particular desired action (as in application such as robot-arm control). Besides the network function computation this can even include the neural learning and adaptation to the particular problem. This way, the neural networks are embedded into a rule-based system to implement some rules or their parts while remaining ones represent classical explicit knowledge. Such a hybrid system is controlled by a procedural inference engine.

Furthermore, even some parts of the conventional inference engine can be implemented by neural networks. For example, the selection of the most effective rule from a particular set of applicable rules is usually done by some heuristics. Several neural network architectures (e. g. Kohonen networks) respond with a single-best-response category and thus, they can be exploited for the best rule selection task. In this case, the neural networks are embedded into inference engine of rule-based systems to perform some of its heuristics. In addition, since in the embedded approach the application of rules still follows a logical sequence, the explanation capability is preserved.

An example of the embedded approach is a hybrid system *COLE* (**CO**nnectionist **L**ogic programming **E**nvironment) (Kasabov and Petkov, 1992a, 1992b) which is an empty experimental rule-based system. It is implemented in programming language Prolog and, in addition, includes neural network simulators as objects for which three predicates are defined: *create_net, train_net, recall_net.* It is clear that a hybrid rule-based system which exploits neural networks to solve partial subtasks can be built using COLE. A similar example is a hybrid production system *COPE* (**CO**nnectionist **P**roduction systems **E**nvironment) (Kasabov, 1993) which employs forward chaining. The application of a production rule in COPE can again involve commands to cope with neural networks. This integration method can also be applied in the above-mentioned expert system FEL–EXPERT (Mařík et al., 1992) whose knowledge base representation is based on inference networks. In a medical application of FEL–EXPERT, neural networks are substituted for parts of the inference network (Družecký, 1992). Yet another interesting example of the embedded approach is a rule-based system for robotic control (Handelman et al., 1989) which initially finds acceptable first-cut solutions and simultaneously supervises the training of neural network by examples provided by rule-based task execution. The resulting integrated system is controlled by the rule-based part and enables a natural division of labor.

13.2.1.3 Neural Implementation of Explicit Knowledge

Another strategy is to transform explicit knowledge (e. g. rules) into a neural network in order to make use of neural network advantages like parallelism, competition, cyclic (recurrent) architecture, gradient minimization, partial matching, additional learning, etc. The neurons in the created network represent microconcepts, propositions, etc. and the connections among them express causal relationships, logical dependencies, etc. Since the network architecture is built from explicit knowledge each neuron and connection have their own purposes in the network and thus, the explicit knowledge representation is preserved although neural network is utilized. The inference in this network is achieved by neural network computational dynamics which can be interpreted in logical terms due to the knowledge explicitness. Hence, justification of conclusions is guaranteed in such systems.

For example, the system *RUBICON* (Samad, 1988) implements rule-based architectures using feedforward neural networks in which individual layers represent parts of rules. Similar approaches have been described by Kasabov and Shishkov (1993), and by Kasabov (1994). Or the so-called *fuzzy cognitive maps* (Kosko, 1987; Styblinski and Meyer, 1988; Taber and Siegel, 1987) are generally cyclic networks where neurons represent phenomena and connections are labeled with signs to express positive or negative causal relationships. The computational network dynamics models the actual development of a studied case. An application of the fuzzy cognitive map to modelling the political situation in South Africa has been described (Kosko, 1987). Another example of neural implementation of explicit knowledge employs again cyclic network architecture (Narazaki and Ralescu, 1992) which is created analytically from rules by means of mathematical programming. Thus, the neurons represent propositions and constraint equations and the violation of constraints is formulated as an energy function. The inference is realized as a minimization process of the energy function to search for a truth value distribution that achieves the optimum consistency with the knowledge. In contrast to the conventional microscopic inference technique based on local and piecewise evaluations of the knowledge, this method is macroscopic in that the whole knowledge is taken into consideration simultaneously.

13.2.1.4 Incorporating Rules into Neural Networks

The opposite approach to the preceding two strategies (see Paragraphs 13.2.1.2, 13.2.1.3) is to incorporate explicit knowledge into a neural network after it has been trained from example data to improve its generalization capability. This strategy is suitable in the case when a training set is available as well as several isolated rules are known. Of course, neural network should generalize these rules during learning from training patterns, however, it can fail to respond correctly in a few cases with respect to these rules. It is because the number of all instances covered by these rules can be unmanageable large and thus, it is impossible to include all of them into the training set to ensure the perfect network function consistency with these rules. In addition, some of these rules can be essential for expert system applicability because breaking them is inadmissible or even dangerous. For example, in a medical system application a wrong diagnosis can endanger a patient's life. Therefore, it is important to have a method for incorporating rules into an already trained neural network while preserving the implicit knowledge which has been learnt from data. However, the knowledge representation in such an improved system is still implicit and hence the explanation of inference is problematic.

For example, the weights of two-layered neural network are adjusted to incorporate rules after the network has been trained (Kasabov, 1991). Namely, the weights associated with the connections leading from those input neurons which represent related facts within the condition of the rule, to those output neurons corresponding to relevant conclusions in this rule, are strengthened. This method improves the network generalization of the corresponding rule. Significantly better generalization of the explicit rules which can be expressed as differentiable functions can even be achieved during the back-propagation learning using training pattern derivatives (Šíma, 1994). Thus, the error function (13.5) includes an additional term which penalizes the discrepancies between the actual and desired derivatives of the network function. In another example (Mazný, 1995) of this approach rules are implemented as a separate so-called *rule neural network* using techniques from Paragraph 13.2.1.3. Then the trained neural network is integrated with this rule network so that the rule network has a higher priority. This guarantees perfect network responses if a condition of some rule is satisfied and, at the same time, the computation of the original trained network is performed if this is not the case.

13.2.1.5 Rule Extraction from Neural Networks

It is also possible to use neural networks for rule extraction to replace human specialists for knowledge acquisition in rule-based systems. In this case, a neural network is trained from data to solve the problem acceptably well and then it is analyzed to extract a set of rules from the trained network with the aid of training set. The rule-extraction process is tedious and the results may not be quite what one expects but it is performable. The resulting rules form the knowledge base of a rule-based system, perhaps after being refined or extended by a human expert. This approach can significantly shorten the rule-based system development time. Another exploitation of rule extraction is the explanation of inference in neural expert systems or even the neural knowledge debugging. The rules which are extracted from the trained neural network are presented to a human specialist who can identify the rules which are incorrect. These rules are then used to generate the appropriate training patterns which, after being additionally learnt, correct the network function and improve its generalization capability.

The relation between inputs and outputs is usually analyzed in a trained neural network to extract rules. For example, the so-called *relation factor* (Saito and Nakano, 1988; Krysiak, 1991) has been introduced in medical diagnostic systems based on feedforward networks. Namely, denote by $C(S, D, P)$ the change in an output value of disease D where a symptom S of a patient P (i. e. a training pattern) is switched from *on* to *off*, $Sum(S, D)$ is a summation of $C(S, D, P)$ for all patients (i. e. over the training set) and $N(S)$ denotes the number of switches of a symptom S. Then the relation factor is proportional to the fraction $Sum(S, D)/N(S)$. Similar statistical techniques in the network function analysis have been used to generate rules which are reasonable in physicians' opinions. Or the so-called *causal index* (Enbutsu et al., 1991) is defined to be the partial derivative of a particular network output by inputs. This index should measure the degree of the causal relationship between the input and output neurons. The causal index is evaluated for average training patterns to generate the relevant rules. Another technique of rule extraction exploits the so-called *structural learning with forgetting* (Ishikawa, 1996). In this method, additional penalty criteria are included in the error function (13.5) to delete connections and neurons with little contribution and to force the hidden neurons to have bipolar states in the continuous model (see (13.6)). Thus, a rough skeletal network architecture is obtained and the distributed representation in hidden neurons is dissipated during learning from example data. This enables to extract explicit rules from the sparse architecture and clearer neural representation. Moreover, the rule extraction can be performed even for trained cyclic neural networks. These networks are computationally equivalent with finite automata whose transition functions are discovered by applying clustering algorithms in the network output space (Omlin and Giles, 1996).

13.2.1.6 Fuzzy-Rule Completion in Neural Networks

The neural implementation of explicit knowledge described in Paragraph 13.2.1.3 can be combined with rule-extraction methods (see Paragraph 13.2.1.5) to complete fuzzy rules and to automate the computation and debugging of their confidences during neural learning. In this combined strategy first the network, with various types of constraints, is initialized from explicit fuzzy rules which are available as a first-cut problem solution. This means that the functions of respective neurons evaluate explicitly these rules (and their parts) or combine them, e. g. the neurons compute prescribed fuzzy logic operations. Hence, these neuron functions may completely differ from (13.1), (13.2). Furthermore, the connections among neurons are now labeled with confidences, certainty factors or possibility measures, etc. which are partially estimated from the given rules or they are randomly chosen. These confidence parameters represent the network configuration and they may be restricted (or even fixed) within specific intervals to preserve their fuzzy-quantity interpretation. Thus, the architecture and the computational dynamics of the created network implement explicitly the set of rules and their fuzzy combinations. The knowledge representation in such networks is perfectly explicit because, in fact, the original fuzzy rules are wired into the network and the role of individual neurons and connections is identifiable. In addition, the architecture of the network may be enriched (or even entirely constructed) with "empty" rules, i. e. with additional neurons and connections whose parameters will be specified during learning.

Then, the network which has been initialized from fuzzy rules is trained from example data to complete and refine the original explicit knowledge. Besides the regular training set, even the original rules may be exploited to generate training patterns in order to strengthen the initial explicit knowledge. During the learning process, only the parameters of rules (e. g. confidences) are adapted within specific bounds while the rules themselves are preserved due to the shape of neuron functions and the fixed network architecture. For this purpose, the learning algorithm (e. g. back-propagation) is tailored to the specific network function. After the network has been trained, the adapted or even new rules can be easily extracted because the knowledge representation is explicit in this network. In this way, the confidences of fuzzy rules can be determined or refined using neural learning heuristics. This combined strategy can be modified if the extraction of rules is not needed. In this case the initial network implementation of rules does not require an explicit knowledge representation and the classical neural network function without constraints (e. g. (13.1), (13.2)) can be employed. Then the explicit representation is lost during the network adaptation and rules can be restored only by applying the techniques from Paragraph 13.2.1.5.

For example, this combined strategy can be applied to tune certainty factors in MYCIN-like expert systems using back-propagation learning algorithm (Lacher et al., 1992). Similarly, the belief measures are accommodated in probabilistic logic and Bayesian networks which may be implemented as neural networks (Chen, 1987). The combined method is also widely employed to refine or derive fuzzy rules and fuzzy controllers (Berenji and Khedkar, 1992; Jang, 1992; Lin and Lee, 1991; Yager, 1994). Furthermore, rules for string-to-string mapping can be extracted from trained neural network with representation restrictions (McMillan et al., 1991). Or the implicit neural network implementation of rules (without the necessity of a rule extraction) to refine expert system performance using neural learning has been used by Yang and Bhargava (1990).

13.2.2 Neural Expert Systems

As we have seen in Paragraph 13.2.1, various integration strategies of explicit and neural knowledge representations are possible. Now, we will focus on a strictly neural approach to exploit the full power of knowledge-based neurocomputing and to illustrate neural knowledge processing. Neural expert systems attempt to weaken the disadvantages of implicit representation by introducing heuristics which analyze neural networks to cope with incomplete information, to explain conclusions and to generate questions for unknown inputs. This means that neural networks are enriched by other functionalities so that they have all required features of expert systems. Furthermore, they can be still linked into hybrid systems to manage larger tasks. We will sketch the main ideas of the prototypical neural expert system MACIE to introduce this approach.

13.2.2.1 MACIE

We will briefly outline the architecture of the historically first real empty neural expert system MACIE (**MA**trix **C**ontrolled **I**nference **E**ngine) proposed by Stephen I. Gallant

(1988). This system is based on the discrete feedforward neural network in which neurons compute the function described by (13.1)–(13.3). Moreover, hidden neurons are, in fact, output neurons and hence, their purposes are given by the application. The inputs of the system represent user's answers to questions and they may have only two values either "yes" or "no" which are encoded by 1 or -1, respectively, by using the states of input neurons (similarly for outputs). In addition, an unknown state is encoded by 0.

The network configuration, i. e. the neural knowledge base is created from training patterns using the so-called *pocket algorithm* (e. g., see the paper by Gallant (1988) for details) which computes relevant integer weights. This algorithm works only for a single layer of neurons and that is why all states of neurons in the feedforward network should be prescribed by training patterns including hidden neurons. The simpler learning task may avoid the efficiency problems and simultaneously the visible hidden neurons serve for a better interpretation of implicit neural knowledge representation. However, sometimes auxiliary hidden neurons with random weights must be added to handle more complicated tasks.

In the case when all inputs are known the inference engine of MACIE computes simply the network function (i. e. all outputs). This can be viewed as forward chaining strategy. However, as usual in expert systems a user presents input facts about the currently solved case gradually and thus, some of the inputs are temporarily (or even remain) unknown. For an incomplete network input a modified inference heuristics is performed. The state y_j of neuron j is computed as follows supposing that the outputs y_i of all neurons $i \in j_{\leftarrow}$ has been determined. Firstly, two auxiliary values $KNOWN_j$ and $MAX\_UNKNOWN_j$ are computed:

$$\begin{aligned} KNOWN_j &= w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} y_i & (13.7) \\ MAX\_UNKNOWN_j &= \sum_{i \in j_{\leftarrow};\, y_i = 0} |w_{ji}| \,. & (13.8) \end{aligned}$$

Further, if $|KNOWN_j| > MAX\_UNKNOWN_j$, then the possible contributions of incident neurons $i \in j_{\leftarrow}$ with currently unknown states (i. e. $y_i = 0$) to the excitation level ξ_j (see (13.1)) cannot influence the state y_j which is given by known outputs ($y_i \in \{-1, 1\}$) of incident neurons:

$$y_j = \begin{cases} -1 & \text{if } KNOWN_j < 0 \\ 1 & \text{if } KNOWN_j > 0\,. \end{cases} \tag{13.9}$$

In the opposite case, if $|KNOWN_j| \leq MAX\_UNKNOWN_j$, then the possible contributions of incident neurons with currently unknown states to the weighted sum may change the state y_j and thus, $y_j = 0$ is set to be unknown. This inference method allows MACIE to reach conclusions although only a fraction of the input values is known.

In MACIE, the confidence of the neuron state y_j is also defined to be a real number $Conf(y_j)$ within the interval $[-1, 1]$ which expresses the inclination of the state y_j to -1 or 1, respectively. For input neurons or neurons with known states the confidence equals the state, i. e. $Conf(y_j) = y_j$. For remaining non-input neurons j with unknown states $y_j = 0$, the confidence $Conf(y_j)$ is determined as follows:

$$Conf(y_j) = \frac{w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} Conf(y_i)}{MAX\_UNKNOWN_j}\,. \tag{13.10}$$

Thus, the unknown outputs can be partially evaluated and preliminary conclusions with confidences can be inferred.

The confidence is also used in the heuristics of generating questions for the most relevant unknown inputs. Here, the backward chaining strategy proceeding from the output layer to the input one is employed. At the beginning, the output neuron j with unknown state $y_j = 0$ and with the maximum absolute confidence $|Conf(y_j)|$ is found, i. e.

$$j = \arg \max_{k \,\text{output},\, y_k=0} |Conf(y_k)| \,. \tag{13.11}$$

In a general step when the neuron j is inspected, the incident preceding-layer neuron $i \in j_{\leftarrow}$ with unknown state $y_i = 0$ and with the maximum absolute influence $|w_{ji}|$ on the underlying neuron j is determined, i. e.

$$i = \arg \max_{k \in j_{\leftarrow},\, y_k=0} |w_{jk}| \,. \tag{13.12}$$

If i is an input neuron, then the user is asked for its unknown value otherwise the general step, now with the inspected neuron j replaced by the above-selected neuron i, is repeated.

Finally, the system MACIE provides a simple justification of inference by generating the IF–THEN rules. During a consultation, the user may ask for an explanation why a particular value of output neuron state is -1 or 1, respectively. For example, assume that the system is asked to explain the state $y_j = 1$ (analogously for $y_j = -1$) of the output neuron j. Then the minimal subset of its incident neurons $i \in j_{\leftarrow}$ whose states ensure the state $y_j = 1$, regardless of the remaining ones, is determined in the following way. Let $I = \{i \in j_{\leftarrow}\,;\, w_{ji}y_i > 0\}$ be a set of incident neurons which contribute positively to the excitation level ξ_j of the underlying neuron j, so that in the result, its state is $y_j = 1$. Further, enumerate $I = \{i_1, i_2, \ldots, i_r\}$ so that $|w_{j,i_1}| \geq |w_{j,i_2}| \geq \cdots |w_{j,i_r}|$ is a non-increasing sequence of absolute values of corresponding weights. Now, the minimal s such that

$$w_{j0} + \sum_{k=1}^{s} |w_{j,i_k}| > \sum_{k=s+1}^{r} |w_{j,i_k}| + \sum_{i \notin j_{\leftarrow} \setminus I} |w_{ji}| \tag{13.13}$$

is determined. Because the meanings of hidden neurons in MACIE are known, the following IF–THEN rule can be extracted to justify the inference of $y_j = 1$:

$$\texttt{IF } (y_{i_1} = a_1) \& (y_{i_2} = a_2) \& \cdots \& (y_{i_s} = a_s) \texttt{ THEN } y_j = 1$$

where $a_1, \ldots, a_s$ represent the current actual values of the neuron states $y_{i_1}, \ldots, y_{i_s}$, respectively.

Obviously, the neural expert system MACIE can compete with rule-based systems in all functional features. However, the explicit interpretation of hidden neurons is still assumed and the full power of neural learning (e. g. back-propagation) is not exploited. This may be the restriction for knowledge representation of complex problems which require actual hidden neurons for successful generalization. Therefore, we will illustrate in Section 13.3 how the MACIE architecture can be tailored to back-propagation networks in the empty neural expert system EXPSYS (Šíma, 1995).

13.2.2.2 Applications

From the preceding exposition and discussions it follows that neural expert systems are conveniently applied to middle complex problems (or subproblems) from areas where neural networks govern analytical solutions and where example data instead of rules is available. Namely, the typical neural network domains are pattern recognition, control, prediction, forecasting, signal processing, diagnostics, fault detection, etc. To illustrate the potential of neural expert systems we will mention several successful applications to real-world problems.

A connectionist expert system *RAMBOT* that learns to play a simple computer game, called *robots*, by observing a human player has been constructed by Mozer (1987). RAMBOT is able to learn behavior that is dependent on a large number of variables (i. e. the robot playing board contains 400 cells) despite the inconsistent demonstrator's play. The system is able to suggest multiple hypotheses with varying degrees of certainty. The results achieved by the system were surprisingly better than those by its teacher.

A neural network trained by back-propagation is applied to the diagnosis of acute myocardial infarction (coronary occlusion) in patients presenting to the emergency department with acute anterior chest pain (Baxt, 1990). The network was trained on a randomly chosen set of about 180 retrospectively studied patients including those who had not sustained acute myocardial infarction. Then the network was tested on previously unseen patterns and it correctly identified 92% of the patients with infarction and 96% of the patients without infarction. This is substantially better than the performance reported for either physicians or any other analytical approach.

The first version of the empty neural expert system EXPSYS (see Section 13.3) has been applied to diagnostics of protection warnings for the cooling system in nuclear power plants (Šíma, 1992a). After training on 112 patterns the generalization of the system was approved on a test set of about 300 patterns where more than 90% of conclusions were accepted by an expert who had been disappointed by previous experience with rule-based systems.

A neural-network classifier for detecting vascular structures in angiograms has been developed by Nekovei and Sun (1995). The classifier consists of a feedforward network window in which the center pixel is classified using gray-scale information within the window. The network was trained by using back-propagation algorithm on 75 selected points from a 256×256 digitized cineangiogram. The three-layer network shows good generalization to the entire cineangiogram and other images including direct video angiogram. In a comparative study, the network demonstrates its superiority in classification performance. Its classification accuracy is 92%, as compared to 68% from a maximum likelihood estimation method, etc.

A medical neural expert system for automated brain signal diagnosis has been presented by Moreno et al. (1995) A training set as well as the test set consisted of data extracted from EEG signal and diagnoses carried out by expert neurologist. The neural approach has been shown to have better performance over traditional statistical classifiers. In addition, the integration of neural networks in a higher-level knowledge-based system for brain signal interpretation has been discussed.

A collection of neural networks called *PROMNET* which are interfaced to a computerized medical record has been built by Bassøe (1995). Clinical narratives were subject to automated natural language processing, and relations were established between 14323 diagnoses and 31381 patient findings which were grouped into clinical entities to train PROMNET using Widrow rule. The dictionary contains about 20000 words and the neural network recognizes more than 2800 disorders. PROMNET makes clinical decision in a few seconds with sensitivity of 96.6% and specificity of 95.7%. Thus, PROMNET is a powerful inference engine that learns from clinical narratives and interacts with medical personnel or patients in natural language. This system is comparable with a current standard, e. g. Internist.

13.3 EXPSYS—an Example of Neural Expert System

In this section we will demonstrate the neural knowledge processing on the empty neural expert system *EXPSYS* (Šíma, 1995) in more detail. The architecture of EXPSYS has been inspired by the system MACIE (see Paragraph 13.2.2.1) whose ideas are adapted for feedforward neural networks trained with back-propagation algorithm. Thus, the neural network used in EXPSYS differs from that of MACIE in two points: learning hidden neurons is feasible although their values are not prescribed by training patterns and further, the activation function (13.6) is differentiable. According to (13.6) arbitrary real numbers between -1 and 1 may represent the hidden neuron states which are further used as inputs for the following-layer neurons. Hence, it would be ambiguous and inconsistent for the incomplete information processing within EXPSYS to determine the discrete states (either -1 or 1) of hidden neurons as in MACIE according to (13.9). Therefore, the interval states of neurons covering all their potential values are introduced in EXPSYS and propagated throughout the network to cope with imprecise and incomplete information. The back-propagation algorithm, on one hand, eliminates weak learning of MACIE and improves the generalization capability of the neural network and, on the other hand, it makes the explicit interpretation of hidden neurons difficult. Moreover, the usage of continuous activation function complicates the inference engine and explanation heuristics. Thus, even in neural expert systems the tension between the implicit and explicit representations appears.

13.3.1 Interface

13.3.1.1 Data Types

After one has decided to use the neural network technology for building an expert system, namely, to apply the empty neural expert system EXPSYS (Šíma and Neruda, 1993), the important issue is to choose the right input and output variables which should best describe the problem. The inputs of the system should cover all important information which is sufficient to solve the problem. On the other hand, the input variables should not be duplicit or irrelevant because the great number of inputs decreases the efficiency of the system. One can even exploit some statistical methods to discover the input irrelevancies and duplicities from data. The output variables represent the problem solutions and system conclusions.

Furthermore, the types of these inputs and outputs should appropriately be defined by a user. The system EXPSYS supports three basic data types: the numerical type (i. e. real numbers or integers), the scalar type with user-defined values and the set. The domain of the numerical type is specified by a real (integer) interval. The domain of the scalar type is defined by a vector of possible values which are chosen by a user. Similarly, for the set type the universe of possible elements is given by a vector of these elements. The difference between the scalar and set types is that the scalar value is exactly one item from the domain while any subset (including empty set) of elements from the universe may appear in the set type.

For example, in a simplified medical problem the inputs and outputs and their types are chosen as follows:

```
INPUTS:   TEMPERATURE: real of [36,42]
          SKIN-COLOR: scalar of (NORMAL,YELLOW)
```

```
          SCLERAE-COLOR: scalar of (NORMAL,YELLOW)
          LIVER-TENDERNESS: scalar of (NORMAL,HIGH)
          CHOLESTEROL: scalar of (LOW,NORMAL,HIGH)
          URINE-TEST: set of (BILIRUBIN, UROBILINOGEN)
OUTPUTS:  DIAGNOSIS: scalar of (HEALTHY,PSEUDO-JAUNDICE,
                                OBSTRUCTIVE-JAUNDICE,HEPATITIS)
```

13.3.1.2 Encoding

The interface of EXPSYS encodes the input and output values by the states of input and output neurons, respectively. The states of neurons with the activation function (13.6) are within the interval $[-1, 1]$ and hence, only the values from this range are used for this purpose. Three possible ways of coding are supported: floating point, unary and binary codes. Almost any combination of a variable type and coding is allowed (excluding unary-coded numerical type) where a binary-coded real value is, in fact, integer. We will describe only the natural combinations of type and coding which are floating-point for reals, unary-coded scalar type and binary-coded set, while the remaining ones are similar and their description can be found in the paper by Šíma (1995).

A numerical value may be directly encoded using the real state of one neuron, i. e. floating point representation. However, the values of variables should somehow be normalized because remarkably different scales of two variables would require different scales of the corresponding weights in the network configuration and consequently, complicate the learning process. Therefore, the domain of the numerical type, say $[a, b]$, is linearly mapped to the neuron state $[-1, 1]$. Hence, a real value $x \in [a, b]$ is encoded by the state

$$y = 2\frac{x-a}{b-a} - 1 \in [-1, 1]\,. \tag{13.14}$$

The scalar type with the domain of k possible values is encoded in unary code using k neurons where each neuron is reserved for one value. Thus, a particular value is encoded in such a way that all states of k neurons are set to -1 except the only one which corresponds to this value and has the state 1. Similarly, each element from the universe of the set type corresponds in the binary code to one neuron whose state is 1 if this element is in the set otherwise it is -1. Hence, the number of neurons needed to encode the set with possibly up to k elements is k. For example, the empty set is encoded by all k states being -1.

The way how the inputs and outputs are encoded is crucial for learning and it may even influence its tractability (Blum and Rivest, 1992). The more significant features of the solved problem are extracted, preprocessed and presented to the network in the simplest form the easier the training task is. Besides normalizing numerical values, the expert knowledge about the problem can be useful for deciding which code is suitable to use for individual inputs and outputs to create appropriate knowledge representation. Compressed representation saves the number of input and output neurons (i. e. the network size) but, on the other hand, it employs hidden neurons for its decoding. Therefore, the sparse unary code is always recommended provided that the number of input and output neurons is manageable. For example, even the numerical variable can be represented with the scalar type including items such as low, medium, high, etc.

13.3.1.3 Incomplete information

The system EXPSYS handles the incomplete information by introducing interval states of neurons (Šíma, 1992a). The interval state is any non-empty (even one-point) subinterval of $[-1, 1]$. Then a crisp value is represented by one-point intervals, e. g. $[1, 1]$ stands for 1. Further, an unknown value is encoded by complete intervals $[-1, 1]$. Even the imprecise value can be expressed by using interval neuron states. For example, this is straightforward for the numerical type. Or for the binary-coded set an unknown membership of particular elements can be encoded by corresponding complete intervals, etc. (see EXPSYS—Students' Software Project (1994) for details). In our medical example the values are encoded as follows:

```
[ ( [-0.2,-0.2],                             38.4
    [-1,-1],[1,1],                           YELLOW
    [-1,-1],[1,1],                           YELLOW
    [-1,-1],[1,1],                           HIGH
    [-1,1],[-1,1],[-1,1],                    unknown
    [1,1],[1,1] ),                           {BILIRUBIN,UROBILINOGEN}
  ( [-1,-1],[-1,-1],[-1,-1],[1,1] ) ]        HEPATITIS
```

13.3.2 Neural Knowledge Base

13.3.2.1 Interval Neuron Function

As we have already mentioned the knowledge base of EXPSYS is a feedforward neural network trained with back-propagation algorithm. Hence, the function (13.1), (13.2), and (13.6) of neurons is differentiable. In addition, this function is generalized to intervals to cope with incomplete information (see Paragraph 13.3.1.3). The interval neuron state is the minimal interval which covers its all possible values with respect to incomplete interval inputs. This property should be preserved throughout the network computation. Therefore, the interval function of neurons should preserve the monotony with respect to the interval inclusion. Namely, if an input value is made more accurate, i. e. the corresponding interval is narrowed down to a subinterval (e. g. down to a one-point interval), then the output intervals may only contract and the new intervals are subsets of the preceding ones. Thus, the output intervals cover all possible results for all possible specifications of incomplete inputs. In what follows the corresponding interval function of neurons is derived.

Let $[a_i, b_i]$ be the interval states of neurons $i \in j_{\leftarrow}$ which are connected to the neuron j. The interval excitation level $[\alpha_j, \beta_j]$ and the interval output $[a_j, b_j]$ of the neuron j are determined so that they meet the following requirement. For any single inputs $y_i \in [a_i, b_i]$ $(i \in j_{\leftarrow})$ from the given interval states, the corresponding unique output y_j of the neuron j computed by (13.1), (13.2), and (13.6) must fall to the output interval, i. e. $y_j \in [a_j, b_j]$. Hence, the lower bound α_j of the excitation level (13.1) is determined as its minimum for $y_i \in [a_i, b_i]$. The corresponding contribution of the i-th neuron $(i \in j_{\leftarrow})$ to this minimum is $w_{ji}a_i$ if the weight $w_{ji} > 0$ is positive and it is $w_{ji}b_i$ for negative weights $w_{ji} < 0$. Similarly, the upper bound β_j is computed:

$$\alpha_j = w_{j0} + \sum_{w_{ji}>0; i \in j_{\leftarrow}} w_{ji}a_i + \sum_{w_{ji}<0; i \in j_{\leftarrow}} w_{ji}b_i \qquad (13.15)$$

$$\beta_j \quad = \quad w_{j0} + \sum_{w_{ji}>0; i \in j_{\leftarrow}} w_{ji} b_i + \sum_{w_{ji}<0; i \in j_{\leftarrow}} w_{ji} a_i \, . \tag{13.16}$$

Further, formula (13.2) for the output y_j can be easily rewritten for the interval output $[a_j, b_j]$ because the activation function (13.6) is increasing:

$$a_j = \sigma(\alpha_j) \qquad b_j = \sigma(\beta_j) \, . \tag{13.17}$$

It is obvious that the interval function (13.15)–(13.17) coincides exactly with the original single-state function (13.1), (13.2), and (13.6) for one-point intervals and, simultaneously, it satisfies the monotony with respect to interval inclusion. However, this interval neuron function is not smooth as required for a gradient-based learning, namely the bounds (13.15), (13.16) of the excitation level are not differentiable with respect to weights at the points with a zero weight. In spite of that it is possible either to compute one-sided derivatives when a discontinuity of derivatives appears and the gradient method can be still tailored to this case (Bělohlávek, 1997). Or the interval function (13.15)–(13.17) can be made differentiable by a continuous approximation of the monotony property (Šíma, 1992b). Although the strict monotony property is partly lost in latter case the system EXPSYS employs this approximate approach as it follows.

To make the interval bounds (13.15), (13.16) of the excitation level differentiable, the auxiliary continuous sigmoid functions

$$s(x) = \frac{1}{1+e^{-x}} \qquad \bar{s}(x) = \frac{1}{1+e^{x}} \tag{13.18}$$

are introduced to approximate the weight signs:

$$s(w_{ji}) \doteq \begin{cases} 1 & \text{for } w_{ji} \gg 0 \\ 0 & \text{for } w_{ji} \ll 0 \end{cases} \qquad \bar{s}(w_{ji}) \doteq \begin{cases} 0 & \text{for } w_{ji} \gg 0 \\ 1 & \text{for } w_{ji} \ll 0 \, . \end{cases} \tag{13.19}$$

Note that $s(x) = \bar{s}(-x)$ and $s(x) + \bar{s}(x) = 1$ for every real x. Thus, the formulas (13.15), (13.16) for the interval excitation level can be rewritten:

$$\alpha_j \quad = \quad w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} \left(s(w_{ji}) a_i + \bar{s}(w_{ji}) b_i \right) \tag{13.20}$$

$$\beta_j \quad = \quad w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} \left(\bar{s}(w_{ji}) a_i + s(w_{ji}) b_i \right) \, . \tag{13.21}$$

This means that the weight w_{ji} is proportionally divided into contributions with a_i and b_i according to the positiveness and negativeness of w_{ji} because of $s(w_{ji}) + \bar{s}(w_{ji}) = 1$.

Furthermore, the so-called *gain parameter* λ_j is supplied to the activation function (13.6) of the neuron j:

$$\sigma_j(\xi) = \frac{1 - e^{-\lambda_j \xi}}{1 + e^{-\lambda_j \xi}} \, . \tag{13.22}$$

The gain parameter λ_j determines the steepness of the activation function σ_j around zero, e. g. for $\lambda_j \to \infty$ this function coincides with its discrete version (13.3). This parameter belongs to the network configuration and thus, it is a subject of adaptation which gives the gradient method more freedom to converge (Hořejš and Kufudaki, 1993). However, λ_j can become negative during the adaptation for which the activation function σ_j is decreasing according to (13.22) a hence, the correctness of (13.17) is broken because this would imply

$a_j \geq b_j$. This problem can be solved by introducing the same gain parameter λ_j into the auxiliary sigmoid functions (13.18) of the neuron j:

$$s_j(x) = \frac{1}{1+e^{-\lambda_j x}} \qquad \bar{s}_j(x) = \frac{1}{1+e^{\lambda_j x}}\,. \tag{13.23}$$

Now, for the negative gain parameter $\lambda_j < 0$ the values of $s_j(w_{ji})$ and $\bar{s}_j(w_{ji})$ are exchanged due to $s(w_{ji}) = \bar{s}(-w_{ji})$ and thus α_j and β_j are switched as well, i. e. $\alpha_j \geq \beta_j$. This exchange is then corrected by the activation function σ_j which is decreasing for $\lambda_j < 0$ and hence, $a_j \leq b_j$. In addition, it is notable that for $\lambda_j \to \infty$ the monotony property is satisfied.

Finally, the above-derived interval function of the neuron j is summarized as follows:

$$a_j = \sigma_j(\alpha_j) \qquad b_j = \sigma_j(\beta_j) \tag{13.24}$$

$$\alpha_j = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji}\left(s_j(w_{ji})a_i + \bar{s}_j(w_{ji})b_i\right) \tag{13.25}$$

$$\beta_j = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji}\left(\bar{s}_j(w_{ji})a_i + s_j(w_{ji})b_i\right) \tag{13.26}$$

$$\sigma_j(\xi) = \frac{1-e^{-\lambda_j \xi}}{1+e^{-\lambda_j \xi}} \tag{13.27}$$

$$s_j(x) = \frac{1}{1+e^{-\lambda_j x}} \qquad \bar{s}_j(x) = \frac{1}{1+e^{\lambda_j x}}\,. \tag{13.28}$$

This function is employed by hidden and output neurons during the computation of the interval network function except by those output neurons which encode the expert outputs using a floating point (see Paragraph 13.3.1.2). For these output neurons the activation function σ_j is not applied, instead the output intervals equal the interval excitation levels, i. e. $a_j = \alpha_j$, $b_j = \beta_j$ and e. g. $\lambda_j = 1$ for the auxiliary sigmoid functions s, $\bar{s}$. It is because the activation function σ_j tends the output y_j to saturate in its limits either -1 or 1 and hence, the medium state values which represent current numerical outputs in a floating point would be discriminated. Thus, the underlying interval network function depends on the network configuration $\mathbf{w}$, $\boldsymbol{\lambda}$ which consists of all weights and gain parameters within the network.

13.3.2.2 Learning Algorithm

Before the learning process starts the user of EXPSYS must specify the architecture of the feedforward neural network. It suffices to determine the number of hidden layers and the number of hidden neurons in them since the number of input and output neurons is given by the number of expert inputs and outputs, by their types and the way how they are encoded (see Paragraphs 13.3.1.1 and 13.3.1.2). In addition, it is implicitly assumed that every neuron in one layer is connected to all neurons in the next layer. Thus, by specifying the hidden layers the network size is established (see also Paragraph 13.1.4.1) which should correspond to the complexity of the problem to achieve the best generalization capability as it has been described in Paragraph 13.1.3. Typically, one or two hidden layers are used in which the number of neurons is of the same order as the number of input and output neurons.

Furthermore, the file of example inferences must be provided to the learning algorithm. The interface of EXPSYS can be used either to create or to load this file. The example inferences are immediately transformed to the training set by encoding the input and output values using the input and output neuron states (see Paragraph 13.3.1.2). These example inferences may even include incomplete information which is encoded using interval training patterns as it has been described in Paragraph 13.3.1.3. The principles of appropriate training patterns selection to cover the problem domain have been introduced in Paragraphs 13.1.3 and 13.1.4.2. The third version of EXPSYS even has a preprocessing procedure for automatic selection of representative patterns from larger files based on Kohonen networks (see EXPSYS—Students' Software Project (1994) for details). One should also not forget to reserve a subset of example inferences for the test set which is used in EXPSYS to evaluate the network outputs with generalization confidences (see Paragraph 13.3.2.3).

Thus, the training set of interval patterns has the following form:

$$T = \{(\mathbf{X}_k; \mathbf{D}_k)\ ;\ k = 1, \ldots, p\} \tag{13.29}$$

where for the k-th training pattern, $\mathbf{X}_k$ is the vector of intervals of input-neuron states and

$$\mathbf{D}_k = ([A_{k1}, B_{k1}], \ldots, [A_{km}, B_{km}]) \tag{13.30}$$

is the corresponding desired vector of intervals $[A_{kj}, B_{kj}]$ of the network outputs in which j denotes the relevant output neuron. The error $E(\mathbf{w}, \boldsymbol{\lambda})$ of the network function (depending on the configuration $\mathbf{w}$, $\boldsymbol{\lambda}$) with respect to the training set (13.29) can be generalized for intervals. Because the interval network function introduced in Paragraph 13.3.2.1 approximately preserves the monotony of the interval inclusion, this error depends only on the bounds of relevant intervals:

$$E(\mathbf{w}, \boldsymbol{\lambda}) = \sum_{k=1}^{p} E_k(\mathbf{w}, \boldsymbol{\lambda}) \tag{13.31}$$

where $E_k(\mathbf{w}, \boldsymbol{\lambda})$ is a partial error with respect to the k-th training pattern:

$$E_k(\mathbf{w}, \boldsymbol{\lambda}) = \frac{1}{2} \sum_{j\ \mathrm{output}} \left((a_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k) - A_{kj})^2 + (b_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k) - B_{kj})^2 \right) \tag{13.32}$$

where $[a_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k), b_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k)]$ is the actual interval state of the output neuron j computed for the interval input $\mathbf{X}_k$ (while the network configuration is $\mathbf{w}$, $\boldsymbol{\lambda}$).

The learning algorithm of EXPSYS minimizes the error function (13.31) in the configuration space using the following gradient method. At the beginning the weights $\mathbf{w}^{(0)}$ are chosen randomly close to zero and the gain parameters $\boldsymbol{\lambda}^{(0)}$ around e. g. 1. Then, at the discrete adaptation time $t = 1, 2, \ldots$ the new configuration $\mathbf{w}^{(t)}$, $\boldsymbol{\lambda}^{(t)}$ is computed as follows:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)} \qquad \lambda_j^{(t)} = \lambda_j^{(t-1)} + \Delta \lambda_j^{(t)} \tag{13.33}$$

where the increments $\Delta w_{ji}^{(t)}$, $\Delta \lambda_j^{(t)}$ of the configuration at the adaptation time t are determined in the following way:

$$\Delta w_{ji}^{(t)} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \left(\mathbf{w}^{(t-1)} \right) + \mu \Delta w_{ji}^{(t-1)} \tag{13.34}$$

$$\Delta \lambda_j^{(t)} = -\varepsilon' \frac{\partial E}{\partial \lambda_j} \left(\boldsymbol{\lambda}^{(t-1)} \right) + \mu' \Delta \lambda_j^{(t-1)} \tag{13.35}$$

where $\frac{\partial E}{\partial w_{ji}}(\mathbf{w}^{(t-1)})$ (similarly $\frac{\partial E}{\partial \lambda_j}(\boldsymbol{\lambda}^{(t-1)})$) is the partial derivative of the error function E by w_{ji} at the point $\mathbf{w}^{(t-1)}$. The so-called *learning rate* $0 < \varepsilon, \varepsilon' < 1$ and the *momentum parameter* $0 < \mu, \mu' < 1$ are the adjustable parameters of the gradient method ($\varepsilon, \varepsilon'$ should be sufficiently small to converge and a reasonable value of μ, μ' is 0.9).

To implement the gradient method (13.33) the partial derivatives $\frac{\partial E}{\partial w_{ji}}$ from (13.34) and $\frac{\partial E}{\partial \lambda_j}$ from (13.35) must be determined. For this purpose, the back-propagation strategy is generalized for the interval network function (Šíma, 1992b). For the notational simplicity we will describe the formulas only for the network without output neurons which encode the floating point. The generalization for the opposite case is straightforward and can be found in the paper by Šíma (1995). First, the rule for the derivative of sum is used for (13.31):

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ji}} \qquad \frac{\partial E}{\partial \lambda_j} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial \lambda_j}. \tag{13.36}$$

To compute $\frac{\partial E_k}{\partial w_{ji}}, \frac{\partial E_k}{\partial \lambda_j}$ the rule for composite function derivative is applied:

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial a_j}\frac{\partial a_j}{\partial \alpha_j}\frac{\partial \alpha_j}{\partial w_{ji}} + \frac{\partial E_k}{\partial b_j}\frac{\partial b_j}{\partial \beta_j}\frac{\partial \beta_j}{\partial w_{ji}} \tag{13.37}$$

$$\frac{\partial E_k}{\partial \lambda_j} = \frac{\partial E_k}{\partial a_j}\frac{\partial a_j}{\partial \lambda_j} + \frac{\partial E_k}{\partial b_j}\frac{\partial b_j}{\partial \lambda_j}. \tag{13.38}$$

Thus, the following derivatives from (13.37), (13.38) can be calculated directly using the explicit form (13.24)–(13.28) of the interval neuron functions:

$$\frac{\partial a_j}{\partial \alpha_j} = \frac{1}{2}\lambda_j\left(1 - a_j^2\right) \qquad \frac{\partial b_j}{\partial \beta_j} = \frac{1}{2}\lambda_j\left(1 - b_j^2\right) \tag{13.39}$$

$$\frac{\partial \alpha_j}{\partial w_{ji}} = a_j s_j(w_{ji})\left(1 + \lambda_j w_{ji}\bar{s}_j(w_{ji})\right) + b_j \bar{s}_j(w_{ji})\left(1 - \lambda_j w_{ji} s_j(w_{ji})\right) \tag{13.40}$$

$$\frac{\partial \beta_j}{\partial w_{ji}} = a_j \bar{s}_j(w_{ji})\left(1 - \lambda_j w_{ji} s_j(w_{ji})\right) + b_j s_j(w_{ji})\left(1 + \lambda_j w_{ji}\bar{s}_j(w_{ji})\right) \tag{13.41}$$

for common weights w_{ji} with $i \neq 0$ and $\frac{\partial \alpha_j}{\partial w_{j0}} = \frac{\partial \beta_j}{\partial w_{j0}} = 1$ for the biases. Further,

$$\begin{aligned} \frac{\partial a_j}{\partial \lambda_j} &= \frac{\partial \sigma_j}{\partial \lambda_j} + \frac{\partial a_j}{\partial \alpha_j}\frac{\partial \alpha_j}{\partial \lambda_j} = \\ &= \frac{1}{2}\left(1 - a_j^2\right)\left(\alpha_j - \lambda_j \sum_{i \in j_\leftarrow} w_{ji}^2 s_j(w_{ji})\bar{s}_j(w_{ji})(b_j - a_j)\right) \end{aligned} \tag{13.42}$$

$$\begin{aligned} \frac{\partial b_j}{\partial \lambda_j} &= \frac{\partial \sigma_j}{\partial \lambda_j} + \frac{\partial b_j}{\partial \beta_j}\frac{\partial \beta_j}{\partial \lambda_j} = \\ &= \frac{1}{2}\left(1 - b_j^2\right)\left(\beta_j + \lambda_j \sum_{i \in j_\leftarrow} w_{ji}^2 s_j(w_{ji})\bar{s}_j(w_{ji})(b_j - a_j)\right). \end{aligned} \tag{13.43}$$

The computation of the remaining derivatives $\frac{\partial E_k}{\partial a_j}, \frac{\partial E_k}{\partial b_j}$ in (13.37), (13.38) starts in the output layer and it is propagated back to the input layer as the name "back-propagation" suggests. So, first assume that j is an output neuron. Then, the relevant derivatives can be directly calculated from (13.32) as follows:

$$\frac{\partial E_k}{\partial a_j} = a_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k) - A_{kj} \qquad \frac{\partial E_k}{\partial b_j} = b_j(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{X}_k) - B_{kj}\,. \tag{13.44}$$

Further, let j be a hidden neuron and denote by $j^{\rightarrow}$ the set of neurons to which the connections from the neuron j lead. Moreover, assume that the values of the partial derivatives $\frac{\partial E_k}{\partial a_r}$, $\frac{\partial E_k}{\partial b_r}$ have been computed for all neurons $r \in j^{\rightarrow}$. Then the rule for a composite function derivative can be applied again:

$$\begin{aligned}
\frac{\partial E_k}{\partial a_j} &= \sum_{r \in j^{\rightarrow}} \left(\frac{\partial E_k}{\partial a_r} \frac{\partial a_r}{\partial \alpha_r} \frac{\partial \alpha_r}{\partial a_j} + \frac{\partial E_k}{\partial b_r} \frac{\partial b_r}{\partial \beta_r} \frac{\partial \beta_r}{\partial a_j} \right) = \\
&= \frac{1}{2} \sum_{r \in j^{\rightarrow}} \lambda_r w_{rj} \left(\frac{\partial E_k}{\partial a_r} \left(1 - a_r^2\right) s_r(w_{rj}) + \frac{\partial E_k}{\partial b_r} \left(1 - b_r^2\right) \bar{s}_r(w_{rj}) \right) \qquad (13.45) \\
\frac{\partial E_k}{\partial b_j} &= \sum_{r \in j^{\rightarrow}} \left(\frac{\partial E_k}{\partial a_r} \frac{\partial a_r}{\partial \alpha_r} \frac{\partial \alpha_r}{\partial b_j} + \frac{\partial E_k}{\partial b_r} \frac{\partial b_r}{\partial \beta_r} \frac{\partial \beta_r}{\partial b_j} \right) = \\
&= \frac{1}{2} \sum_{r \in j^{\rightarrow}} \lambda_r w_{rj} \left(\frac{\partial E_k}{\partial a_r} \left(1 - a_r^2\right) \bar{s}_r(w_{rj}) + \frac{\partial E_k}{\partial b_r} \left(1 - b_r^2\right) s_r(w_{rj}) \right). \qquad (13.46)
\end{aligned}$$

This completes the gradient computation as well as the description of the learning algorithm.

13.3.2.3 Expert Checking

After the feedforward neural network has learnt the training set of interval patterns it is necessary to check whether the created neural knowledge base can be exploited to infer usable conclusions from it. This can be done by a human expert who can evaluate the quality of the system answers in typical situations or by the test set which is used for computing generalization confidences of individual expert outputs. A *generalization confidence* of a particular expert output is a fraction of the number of patterns from the test set, whose desired values for the respective output coincide with its actual values computed by the inference engine (see Paragraph 13.3.3) for the corresponding inputs, over the size of the test set. Typically, the neural knowledge base is not perfect after the first learning and it is further debugged. This can even include the change of the network architecture (see Paragraph 13.1.3) and new learning from the beginning. Usually, it suffices to train additionally the network to improve its generalization capability. In this case, the mistaken network responses serve for creating the appropriate training patterns which are additionally learnt to correct the network function. It is useful to iterate the cycle of learning and testing until the neural network inference is sufficiently accurate. At the end of neural knowledge base checking the generalization confidences of all expert outputs are computed by using the test set.

13.3.3 Inference Engine

After the neural knowledge base is created it is used for the inference from generally incomplete inputs. During a consultation, the user of EXPSYS presents some (typically not all) values of inputs to the system which provides him with the partial conclusions and their confidences. These partial conclusions are gradually made precise, refined and the confidences increase when the user completes the inputs. The inference engine always re-computes the outputs and their confidences after every input value is presented. An example of the inference in the medical expert system follows:

```
  TEMPERATURE = unknown
  SKIN-COLOR = YELLOW
  SCLERAE-COLOR = NORMAL
  LIVER-TENDERNESS = NORMAL
  CHOLESTEROL = unknown
  URINE-TEST = unknown

----> DIAGNOSIS = PSEUDO-JAUNDICE         conf.: 0.86
```

Now, we will describe how the inference engine of EXPSYS works. The expert inputs including unknown ones are encoded using the interval states of input neurons by the system interface (see Paragraphs 13.3.1.2 and 13.3.1.3). Then the interval network function is computed according to (13.24)–(13.28) to find out the corresponding interval excitation levels and the interval states for all output neurons which are used to determine the expert outputs and their confidences. Further, we will again restrict ourselves to unary and binary coded outputs while the description of the floating point case can be found in the paper by Šíma (1995). Hence, the states of output neurons should represent only two values either 1 or -1. However, the actual state y_j of the output neuron j computed by (13.24) is a number within the interval $[-1, 1]$ (even $-1 < y_j < 1$), due to the continuity of the activation function (13.27). Therefore, the actual output should be rounded off. For this purpose, the optional so-called *separation parameter* $0 < \gamma < 1$ is introduced so that the state $y_j \in [\gamma, 1]$ of the output neuron j is considered as 1 and similarly, the output $y_j \in [-1, -\gamma]$ is interpreted as -1 while the state $y_j \in [-\gamma, \gamma]$ means an unknown value. It is clear that the separation parameter controls the precision of the network output.

For the output neuron j the separation parameter δ_j for its excitation level is computed by applying the inverse of the activation function (13.27):

$$\delta_j = \sigma_j^{-1}(\gamma) = \frac{1}{\lambda_j} \ln\left(\frac{1+\gamma}{1-\gamma}\right) . \tag{13.47}$$

In the sequel, assume that $\lambda_j > 0$ and hence $\delta_j > 0$ while the opposite case is similar. Now, comparing this parameter δ_j with the bounds $\alpha_j \le \beta_j$ ($\lambda_j > 0$) of the interval excitation level (13.25), (13.26) for the output neuron j, the positive confidence $0 \le c_j^+ \le 1$ of the *rounded output* $v_j = 1$ as well as the negative confidence $0 \le c_j^- \le 1$ of the rounded output $v_j = -1$ are computed as follows:

$$c_j^+ = \begin{cases} 1 & \text{for } \delta_j \le \alpha_j \\ \frac{\beta_j - \delta_j}{\beta_j - \alpha_j} & \text{for } \alpha_j < \delta_j < \beta_j \\ 0 & \text{otherwise} \end{cases} \tag{13.48}$$

$$c_j^- = \begin{cases} 1 & \text{for } \beta_j \le -\delta_j \\ \frac{-\delta_j - \alpha_j}{\beta_j - \alpha_j} & \text{for } \alpha_j < -\delta_j < \beta_j \\ 0 & \text{otherwise}\,. \end{cases} \tag{13.49}$$

Finally, the rounded state v_j of the output neuron j together with its so-called *inference confidence* c_j is determined from the dominant confidence:

$$v_j = \begin{cases} 1 & \text{for } c_j^+ > c_j^- \\ -1 & \text{for } c_j^+ < c_j^- \\ unknown & \text{otherwise} \end{cases} \qquad (13.50)$$

$$c_j = \max\left(c_j^+, c_j^-\right) . \qquad (13.51)$$

Then, every (known) output of the expert system is decoded from the corresponding known rounded states of output neurons and it is associated with the confidence. This confidence is computed as a product of the inference confidences of relevant coding states multiplied by the corresponding generalization confidence (see Paragraph 13.3.2.3). It can happen that the rounded states of relevant output neurons do not represent any code of an expert output value (see Paragraph 13.3.1.2), e. g. there are more states equal to 1 in unary code or some of the neuron states are unknown (see (13.50)), etc. In these cases the corresponding expert output is considered as unknown or even imprecise (see EXPSYS—Students' Software Project (1994) for details).

13.3.4 Explanation of Conclusions

The neural expert system EXPSYS provides a simple explanation of the conclusions which are inferred by the inference engine. During a consultation the user can ask why a particular output value has been concluded. The system EXPSYS determines a list of selected input values with their relative percentual influence measures which have mostly influenced the inference of the underlying output. Thus, the corresponding total (100%) influence is distributed only to the influence measures of these selected inputs while the influence of remaining inputs is neglected. In our medical example we can obtain:

```
  DIAGNOSIS = OBSTRUCTIVE-JAUNDICE ?
----> URINE-TEST = {BILIRUBIN}    51%
      CHOLESTEROL = HIGH          27%
      TEMPERATURE = unknown       22%
```

The results of the explanation can be exploited, not only for the justification of the inference, but it is also possible to debug (see Paragraph 13.3.2.3) the neural knowledge base by means of them. For example, using the explanation of incorrect system answers the appropriate example inferences can be proposed which, after being additionally learnt, correct the expert system behavior. In addition, the explanation of the conclusions is used in EXPSYS for generating questions for unknown inputs (see Paragraph 13.3.4.2).

13.3.4.1 Explanation Heuristics

The explanation heuristics for the output which is encoded by using floating point is based on the causal index (Šíma, 1995) and will be again omitted here. The explanation heuristics for the unary and binary coded expert output employs a backward chaining strategy and finds out the dominant influences of neurons on the underlying output. For its description an auxiliary data structure, namely a *LIST* of records Z is used. The record Z consists of the three following items:

Z.N ... neuron identification number
Z.S ... influence sign (-1 or 1)
Z.M ... influence measure

The list of records is ordered according to $Z.N$ in such a way that neurons in one layer always precedes neurons in the preceding layers, e. g. the output neurons are at the top of $LIST$. Moreover, the records with the same $Z.N$ follows each other in this list. Furthermore, the following operations are implemented for $LIST$:

insert(Z) ... insert Z into $LIST$ with respect to the ordering
get(Z) ... get the first (top) record Z from $LIST$
LIST$\uparrow$ ... access (buffer) variable to the first (top) record in $LIST$

Denote by Y the set of output neurons encoding the given (known) expert output value which is asked for an explanation. At the beginning of the explanation procedure the records for these output neurons $j \in Y$ are created and inserted into $LIST$. Their rounded states (13.50) represent influence signs and their inference confidences (13.51) determine influence measures. In addition, the variable F_q for an influence of the expert system input q ($q = 1, \ldots, z$) on the underlying output is introduced and initialized by zero. For the detailed description of the explanation heuristics a Pascal-like pseudocode will be used:

{ initialization }
$LIST \leftarrow$ empty list;
forevery $j \in Y$ **do**
  $Z.N \leftarrow j$;
  $Z.S \leftarrow v_j$;
  $Z.M \leftarrow c_j$;
  $insert(Z)$
enddo;

forevery $q = 1, \ldots, z$ **do** $F_q \leftarrow 0$ **enddo**;

In a general step the total influence sign and measure of the neuron, whose records are on the top of the $LIST$, are determined from its multiple occurrences. Namely, in the course of explanation procedure it may even happen that one of these records considers the current neuron to have the positive influence sign which should explain the underlying expert output while the other one prefers the same neuron to have the negative sign for the same reason. This is caused by the dense network architecture in which a hidden neuron may influence two different following-layer neurons in the opposite way. Thus, the sign associated with the dominant influence measures is assigned to this neuron and its resulting influence measure is evaluated with respect to this sign. Furthermore, if the current neuron is from the input layer (i. e., $LIST$ contains only input neurons), then its contribution to F_q is added:

repeat { general step }

  { multiple occurrences of one neuron in $LIST$ }
  $M1 \leftarrow 0$; $M2 \leftarrow 0$;
  repeat

```
    get(Z);
    if Z.S = 1 then M1 ← M1 + Z.M
              else M2 ← M2 + Z.M
    endif
  until LIST ↑ .N ≠ Z.N;
  if M1 > M2 then Z.S ← 1 else Z.S ← −1 endif;
  Z.M ← |M1 − M2|;

  if Z.N is input neuron
    then
      q ← expert input coded by neuron Z.N;
      F_q ← F_q + Z.M
```

If the current neuron j is not from the input layer, then the contributions R_i of incident neurons $i \in j_{\leftarrow}$ to its excitation level (13.25), (13.26), namely either to α_j or to β_j depending on the associated influence sign, are determined. Further, only those neurons $i \in j_{\leftarrow}$ whose contributions R_i have consistent signs are selected. In particular, assume the gain parameter $\lambda_j > 0$ is positive (similarly for $\lambda_j < 0$), i. e. the activation function σ_j is increasing and $\alpha_j \leq \beta_j$. Now, if the influence sign $Z.S = 1$ is positive then all neurons with positive contributions $R_i > 0$ are selected otherwise all neurons with negative contributions $R_i < 0$ are chosen:

```
    else { Z.N is non-input neuron }

      { contributions to excitation level of Z.N }
      j ← Z.N;
      forevery i ∈ j← do
        if Z.S = 1 then R_i ← w_ji(s̄_j(w_ji)a_i + s_j(w_ji)b_i)
                  else R_i ← w_ji(s_j(w_ji)a_i + s̄_j(w_ji)b_i)
        endif
      enddo;

      { selection of contributions with consistent signs }
      I ← {i ∈ j← ; Z.S × λ_j R_i > 0};
```

Furthermore, only the neurons $i \in I$ with dominant contributions R_i are included into $LIST$. For this purpose the neurons in I are sorted with respect to the associated absolute contributions $|R_i|$ into the non-increasing sequence. Then so many neurons from the beginning of this sequence are selected that the sum of their absolute contributions is enough to exceed or equal the sum of the remaining ones. The influence signs of these selected neurons i are determined from the corresponding weights w_{ji} to strengthen the original sign associated with neuron j. Namely, providing that the gain parameter λ_j is positive (similarly for $\lambda_j < 0$) then if the weight w_{ji} is positive then the influence sign assigned to neuron i is positive otherwise it is negative. Similarly, the influence measures of the selected neurons correspond to their absolute contributions $|R_i|$ multiplied by the original influence measure associated with neuron j. This completes the general step:

```
      { selection of dominant contributions }
      sort I = {i_1, ..., i_r} so that |R_{i_1}| ≥ |R_{i_2}| ≥ ···|R_{i_r}|;
      r' ← min {u ; Σ_{k=1}^{u} |R_{i_k}| ≥ Σ_{k=u+1}^{r} |R_{i_k}|};
```

$I' \leftarrow \{i_1, \ldots, i_{r'}\}$;

{ insertion of influential neurons into $LIST$ }
forevery $i \in I'$ **do**
$Z1.N \leftarrow i$;
if $Z.S \times \lambda_j w_{ji} > 0$ **then** $Z1.S \leftarrow 1$ **else** $Z1.S \leftarrow -1$ **endif**;
$Z1.M \leftarrow Z.M \times |R_i|$;
$insert(Z1)$
enddo

endif { end of non-input neuron processing }

until $LIST = empty$ { end of general step }

By using the preceding procedure the influences $F_q \geq 0$ of expert inputs $q = 1, \ldots, z$ on the given output value which is being explained are determined. Again, only the dominant influences are considered. For this purpose, they are sorted so that $F_{q_1} \geq F_{q_2} \geq \cdots \geq F_{q_z}$ and the minimal z' is determined in such a way that

$$\sum_{k=1}^{z'} F_{q_k} \geq \sum_{k=z'+1}^{z} F_{q_k} . \tag{13.52}$$

In order not to overload the user the number of influential inputs is further restricted to $L = \min(z', 10)$. Thus, the desired list of inputs which have mostly influenced the underlying expert output value is $q_1, \ldots, q_L$ with the corresponding percentual influence measures:

$$M_{q_k} = 100 \frac{F_{q_k}}{\sum_{i=1}^{L} F_{q_i}} \qquad k = 1, \ldots, L . \tag{13.53}$$

The limitation of the above-described explanation heuristics is that it is impossible to give reasons for an unknown expert output value because the influence signs of the respective neurons cannot be determined.

13.3.4.2 Question Generating

In EXPSYS the explanation heuristics described in Paragraph 13.3.4.1 is used for generating questions for relevant unknown expert system inputs. The strategy is to ask for such an unknown input that, given its value, the completeness and confidences of expert system outputs would mostly increase after the next inference. This is not only a matter of one question but of a sequence of questions because the user can respond that the answer to the first question is unknown. Thus, all known expert output values are gradually explained and the corresponding percentual influence measures (13.53), for each unknown input separately, are added. Then the sequence of questions for unknown input values is given by a decreasing order of these sums. If no unknown input has influenced the known output values according to the explanation heuristics, then a random sequence of questions for all unknown inputs is generated.

13.3.5 Example of Application

The three versions of the empty neural expert system EXPSYS have been successfully applied to solve several real-world problems: the diagnostics of protection warnings for the cooling system in nuclear power plants (Šíma, 1992a); the diagnostics and progress prediction of hereditary muscular diseases, namely Duchenne's and Becker's dystrophy (Šíma and Neruda, 1994); the diagnostics of ear diseases on the basis of brainstem auditory evoked potentials (Mechlová, 1994), etc. In this paragraph the application of EXPSYS to a toy economic problem is compared with the rule-based system solution to demonstrate the differences in the processing of implicit and explicit knowledge representations.

13.3.5.1 Problem Description

We are facing a problem of developing a banking expert system that would decide, after screening some personal data of applicants, whether she/he is to be granted credit or not. Our example will be based on Japanese credit screening data (1992). This data set contains examples of people that were or were not granted credit from a Japanese banking company.

As the first natural step towards this goal we need to choose which items of personal data are relevant for our decision making. However, as we employ an existing data set, this choice was made for us beforehand. Thus, our expert system will examine the following information: whether the applicant has a job and if so, how long she/he is with the current company, the item that the loan is for, sex, age, marital status, region she/he lives in, bank deposit, monthly loan payment and, finally, number of months to pay off the loan.

The data set has 125 example inferences: 85 positive (people that were granted credit) and 40 negative (people that were rejected). All examples are complete.

13.3.5.2 Applying EXPSYS

Our task of applying EXPSYS to this problem has to start with definition of data types. We used the following definition:

```
INPUTS:  JOBLESS: scalar of (NO,YES)
         ITEM: scalar of (PC,CAR,STEREO,JEWEL,MEDINSTRUMENTS,
                          BIKE,FURNITURE)
         SEX: scalar of (FEMALE,MALE)
         UNMARRIED: scalar of (NO,YES)
         PROBLEM-REGION: scalar of (NO,YES)
         AGE: real of [0,150]
         MONEY-IN-BANK: real of [0,999]
         MONTHLY-PAYMENT: real of [0,100]
         MONTHS-TO-PAYOFF: real of [0,60]
         YEARS-AT-COMPANY: real of [0,99]
OUTPUT:  CREDIT: scalar of (NO,YES)
```

Then one of our example inferences looks like

```
[(NO,CAR,MALE,YES,NO,25,150,10,20,2),YES].
```

Since we chose unary encoding for the attribute `ITEM`, binary encoding for all the other scalars and natural floating-point encoding for reals, we get altogether 16 neurons in the input layer and 1 neuron in the output one.

The crucial moment in the learning process of a neural network is making a decision about a particular network architecture and finding some good representative subset of the set of patterns.

As a first attempt, we put 10 neurons in one (and only) hidden layer. Then we split patterns into two parts: one containing patterns used in the process of learning, the other patterns used for testing network performance. In our case the learning set has 62 patterns (42 of them positive) and the testing set 63 (43 of them positive).

Thus, we have started learning procedure with architecture 16–10–1. After approximately 7 hours of learning on 133 Mhz Pentium PC the learning algorithm got stuck with an error roughly 5 and three unlearned patterns. By examining these patterns we have found that two of them are almost contradictory and that the third pattern is the only positive example of a person without a job in the whole learning set. As a next step, we let out two patterns from the learning set: one of the contradictory patterns and the positive jobless example, and relearned the whole set. After 2 hours the net learned all the examples and ended with an error of 0.2. This network classified correctly roughly 70% patterns of the test set. We obtained very similar results for architectures with two (16–8–4–1) and three (16–6–4–4–1) hidden layers. The network with architecture 16–8–4–1 showed the best generalization on the test set: 71%.

Yet 71% generalization confidence is quite unsatisfactory. As several variations of the network architecture did not influence the quality of generalization very much, the learning set was probably ill-chosen—it did not capture the knowledge well.

As a second attempt, we have chosen the 16–8–4–1 architecture and divided the patterns into learning, auxiliary and test sets with 42, 42 and 41 patterns, respectively. The purpose of the auxiliary set was to serve as a resource of new example inferences from which it would be possible to draw appropriate patterns and enrich the learning set. The net was then learned during an iterative process comprising three steps, namely (re)learning the current learning set, checking the performance of the net on the test set and successive modification of the learning set. The learning set was modified by removing contingent unlearned patterns and adding some patterns from the auxiliary set that were "similar" to those misclassified in the test set.

We have iterated the whole process three times (see Paragraph 13.3.2.3). After the first run, the net generalized on 68% of cases from the test set. Finally, after the third run, we obtained 93% generalization confidence on the test set (of 41 patterns). In the last run we used 72 patterns in the learning set. Learning in each run took approximately 6–8 hours.

Here we can give some examples of EXPSYS inferences. For instance, when we present the system with the following incomplete information:

```
ITEM = BIKE
SEX = FEMALE
PROBLEM-REGION = YES
MONEY-IN-BANK = 9.5
YEARS-AT-COMPANY = 7.5
```

it infers

```
----> CREDIT = NO                conf.: 0.77
```

Now we can ask for an explanation of the inference and we obtain the following:

```
  CREDIT = NO ?
----> YEARS-AT-COMPANY = 7.5        54%
      MONTHLY-PAYMENT = unknown     20%
      ITEM = BIKE                   15%
      MONEY-IN-BANK = 9.5           11%
```

Not having a human expert at hand, we can judge the quality of inference only by confronting it with the data. However, this is rather vague strategy and can represent applying various types of methods for retrieving information from data. In our simple example, we have just looked into the data and noted some first-sight facts. For example, the average length of present occupation of people who were granted credit is 9.2 years, whereas for those not granted it is only 2.4 years. Moreover, 70% of people applying for a loan for a bike were rejected and the average amount of bank deposits for the accepted applications was 81.6 units. Thus we can conclude at least, that the above inference is not in contradiction with the data.

Now we can present an example of a positive inference:

```
  JOBLESS = NO
  ITEM = MEDINSTRUMENTS
  AGE = 58
  MONEY-IN-BANK = 70
  MONTHLY-PAYMENT = 1.5
  YEARS-AT-COMPANY = 36.5
----> CREDIT = YES               conf.: 0.75
```

and its explanation:

```
  CREDIT = YES ?
----> ITEM = MEDINSTRUMENTS         35%
      MONTHLY-PAYMENT = 1.5         27%
      JOBLESS = NO                  21%
      SEX = unknown                 10%
      PROBLEM-REGION = unknown       7%
```

We can also specify a value of an up-to-now unknown attribute so as to obtain a higher confidence rate of the inference. E. g., we can add a value of attribute PROBLEM-REGION and we obtain:

```
  PROBLEM-REGION = NO
----> CREDIT = YES               conf.: 0.93
```

Again, by looking into the data, we can find that 80% of people applying for a loan for a PC were granted credit, 75% of applicants that have a job also got credit, as well as 72% of people that live in a non-problem region. For other attributes, however, the dependencies are not so clear. We can say at least that the inference performed by the expert system does not contradict the data.

At this point, it is instructive to use the above example of positive inference to demonstrate performance of the EXPSYS interval neuron function (see Paragraph 13.3.2.1). For this purpose we present values of network interval states for the positive example

Expert System Input	**Input Layer**	**Hidden Layers**	**Output**
`JOBLESS = NO`	$[-1,-1]$		$[-0.992,1]$
	$[-1,-1]$	First Layer	
	$[-1,-1]$	$[-0.683,-0.398]$	
	$[-1,-1]$	$[0.583,0.919]$	$\alpha=-0.915$
`ITEM = MEDINSTRUMENTS`	$[-1,-1]$	$[-0.993,0.986]$	$\beta=4.548$
	$[1,1]$	$[-0.059,0.921]$	$v=1$
	$[-1,-1]$	$[-1,0.848]$	$c=0.807$
	$[-1,-1]$	$[-1,-0.916]$	conf.= $c\times 0.93$
`SEX = unknown`	$[-1,1]$	$[-0.527,0.216]$	conf.= 0.75
`UNMARRIED = unknown`	$[-1,1]$	$[0.424,0.522]$	
`PROBLEM-REGION = unknown`	$[-1,1]$		
`AGE = 58`	$[-0.227,-0.227]$	Second Layer	
`MONEY-IN-BANK = 70`	$[-0.86,-0.86]$	$[-1,0.666]$	
`MONTHLY-PAYMENT = 1.5`	$[-0.97,-0.97]$	$[0.319,0.506]$	
`MONTHS-TO-PAYOFF = unknown`	$[-1,1]$	$[-0.523,-0.039]$	
`YEARS-AT-COMPANY = 36.5`	$[-0.263,-0.263]$	$[-0.063,1]$	

Table 13.1: Network interval states for the positive inference example.

in Table 13.1 and, for comparison, values of interval states for the same example after specification of one more attribute (namely, `PROBLEM-REGION = NO`) in Table 13.2.

The first column of both tables contains the expert system input (i. e. values of attributes), the second column contains their corresponding encodings as interval states of input neurons (for details on input encodings see Paragraph 13.3.1.2). The third column contains values of interval states of neurons in both hidden layers—eight neurons in the first hidden layer and four neurons in the second hidden layer. The last column contains the interval state and excitation level of the output neuron together with final results of inference.

We describe here briefly the steps of inference procedure for the example in Table 13.1; the case for Table 13.2 is similar (for details on inference engine see Paragraph 13.3.3). Since we chose the value of the separation parameter γ to be 0.4 and the gain parameter of the output neuron is $\lambda = 6.094$, we obtain the value of the separation parameter of the output neuron $\delta = 0.139$ by (13.47). Then, for the interval excitation level of the output neuron $[\alpha,\beta] = [-0.915, 4.548]$, we get the positive confidence $c^+ = 0.807$ and the negative confidence $c^- = 0.142$ from (13.48) and (13.49), respectively. Now we can determine the rounded output of the output neuron $v = 1$ by (13.50) and the inference confidence $c = 0.807$ according to (13.51). Finally, by multiplying the inference confidence c by the above-mentioned generalization confidence 0.93 we obtain the resulting confidence 0.75 (see Paragraph 13.3.2.3).

By comparing values in Table 13.1 and Table 13.2 we can see how specifying a previously unknown input influence network interval states. Particularly, Table 13.2 shows the network states after setting the value `NO` to the attribute `PROBLEM-REGION`, which was left `unknown` in Table 13.1. Thus the states of the neurons in the input layer differ only for the neuron which encodes the value of `PROBLEM-REGION` attribute and which becomes $[-1,-1]$ in Table 13.2. When comparing states of the neurons in hidden layers we can observe that several interval have contracted. Namely, the intervals of the second and the

Expert System Input	Input Layer	Hidden Layers	Output
`JOBLESS = NO`	$[-1,-1]$		$[0.889, 1]$
	$[-1,-1]$	First Layer	
	$[-1,-1]$	$[-0.686, -0.403]$	
	$[-1,-1]$	$[0.706, 0.939]$	$\alpha = 0.465$
`ITEM = MEDINSTRUMENTS`	$[-1,-1]$	$[-0.947, 0.992]$	$\beta = 4.559$
	$[1,1]$	$[0.045, 0.933]$	$v = 1$
	$[-1,-1]$	$[-1, 0.886]$	$c = 1$
	$[-1,-1]$	$[-1, -0.997]$	conf.$= c \times 0.93$
`SEX = unknown`	$[-1,1]$	$[-0.372, 0.344]$	conf.$= 0.93$
`UNMARRIED = unknown`	$[-1,1]$	$[0.357, 0.45]$	
`PROBLEM-REGION = NO`	$[-1,-1]$		
`AGE = 58`	$[-0.227,-0.227]$	Second Layer	
`MONEY-IN-BANK = 70`	$[-0.86,-0.86]$	$[-1, 0.279]$	
`MONTHLY-PAYMENT = 1.5`	$[-0.97,-0.97]$	$[0.347, 0.525]$	
`MONTHS-TO-PAYOFF = unknown`	$[-1,1]$	$[-0.528, -0.051]$	
`YEARS-AT-COMPANY = 36.5`	$[-0.263,-0.263]$	$[0.281, 1]$	

Table 13.2: Interval states for the positive example with one more attribute specified.

fourth neuron in the first hidden layer and the intervals of the first and the fourth neuron in the second hidden layer show substantial contraction. Indeed, we can also observe that some neurons shifted their interval states slightly and thus do not preserve the strict monotony property (see Paragraph 13.3.2.1). The interval state of the output neuron shows the biggest contraction. However, it is the value of the interval excitation level of the output neuron which is relevant for the expert system output. Here, the lower bound of the excitation level α increases from -0.915 in Table 13.1 to 0.465 in Table 13.2, and thus becomes greater then the separation parameter $\delta = 0.139$. This forces the rounded output $v = 1$ with inference confidence $c = 1$ and resulting confidence 0.93.

We conclude this paragraph by listing the questions that are generated by the system together with the measures of their influence on a particular output. Since there is only one output in the system, the order, in which the questions are generated, coincides with the decreasing significance of individual unknown inputs (see Paragraph 13.3.4.2). This is demonstrated for the case when all input attributes are left unknown:

```
  CREDIT = YES ?                  (conf.: 0.48)
----> YEARS-AT-COMPANY = unknown    56%
      MONTHS-TO-PAYOFF = unknown    20%
      ITEM = unknown                13%
      MONEY-IN-BANK = unknown       11%
```

13.3.5.3 Comparison with Rule-Based Solution

The Japanese credit screening data (1992) also contains a domain theory—i. e. information obtained by interviewing banking experts who make decisions about granting credits. This information, transformed to several rules, represents the knowledge base of a rule-based banking expert system which examines the trustworthiness of people applying for a loan. The list of rules follows:

```
#1: IF jobless=yes & sex=male
    THEN credit=no
#2: IF jobless=yes & sex=female & unmarried=yes
    THEN credit=no
#3: IF jobless=yes & item=bike & sex=female & unmarried=yes
    THEN credit=no
#4: IF jobless=yes & sex=female & unmarried=no &
       & (money-in-bank < monthly-payment * months-to-payoff)
    THEN credit=no
#5: IF problem-region=yes & (years-at-company <= 10)
    THEN credit=no
#6: IF (age > 59) & (years-at-company < 3)
    THEN credit=no
#7: IF credit<>no
    THEN credit=yes
```

Two aspects can be observed here. First, no confidences for individual rules are used in this simple example. This means that all rules have absolute validity. Second, the rules form a chain of exclusions—i. e. each rule rejects a class of applicants for a loan.

Since we are not in contact with the banking expert, the only way to verify validity of the above rules is to test their performance on the data set. For this purpose we have incorporated the rules into an empty expert system toolkit—particularly we employed Prolog-based system MIKE (Eisenstadt and Brayshaw, 1990; MIKE, 1990). Tests have shown that the rules err on 25 examples out of 125. However, it is important to distinguish between positive and negative misclassifications. There are 22 examples wrongly classified by the system as `yes`, whereas only 3 wrongly classified as `no`. With respect to the exclusive form of the rules this means that the rules are incomplete rather than incorrect. Thus, through the consultation of a banking expert we could devise additional exclusive rules that would set aside the remaining wrongly positive cases.

When comparing neural and rule-based approaches to building an expert system for granting credits, we must confine ourselves only to comparison of performance with respect to the data set. Having done so, we can conclude that the neural solution is not worse than the rule-based one. Particularly for the rules contained in Japanese credit screening data (1992), we can say that the neural solution shows much better performance.

Nevertheless, the main purpose of this application example is to demonstrate some key differences between neural and rule-based approaches in the process of developing an expert system. From this point of view it appears that the main difference consists in creating and debugging the knowledge base. While the rule-based solution has been proposed by banking experts who made an effort to express their credit granting strategy analytically, the neural approach issues from particular examples of their credit granting decisions in practice. On the basis of these examples the representative training patterns have been selected as well as the adequate neural network architecture has been determined. It has been confirmed that these two optional parameters are crucial for neural network learning and generalization and their debugging represents the important part of the neural knowledge base development. Thus, instead of improving the analytical solution the neural knowledge engineer trains different network architectures using various training sets to achieve the best performance of the neural inference engine on the test set. It follows that the knowledge-based neurocomputing exploits a different methodology when applying statistical techniques rather than the formal logic which is used in conventional rule-based systems.

In addition, the application has also approved that the expert system functionalities of EXPSYS which analyze the implicit neural knowledge can compete with the rule-based solution. Particularly, the neural expert system decides about credit granting when only incomplete personal data of applicants is available. Furthermore, the system asks additional questions to complete the information about an applicant which is mostly relevant for the resulting decision. Finally, the system simply explains why the loan is provided or rejected. Thus, we can conclude that the neural expert system EXPSYS represents a respectable alternative to rule-based systems.

Bibliography

[1] Bassøe, C.-F. 1995. Automated Diagnoses From Clinical Narratives: A Medical System Based on Computerized Medical Records, Natural Language Processing, and Neural Network Technology. *Neural Networks*, 8(2):313–319.

[2] Baxt, W. G. 1990. Use of an Artificial Neural Network for Data Analysis in Clinical Decision-Making: The Diagnosis of Acute Coronary Occlusion. *Neural Computation*, 2(4):480–489.

[3] Bělohlávek, R. 1997. Backpropagation for Interval Patterns. *Neural Network World*, 7(3):335–346.

[4] Berenji, H. R., and Khedkar, P. 1992. Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–740.

[5] Biondo, S. J. 1990. *Fundamentals of Expert Systems Technology: Principles and Concepts.* Norwood, NJ: Ablex Publishing Co.

[6] Blum, A. L., and Rivest, R. L. 1992. Training a 3-Node Neural Network is NP-complete. *Neural Networks*, 5(1):117–127.

[7] Buchanan, B., and Shortliffe, E. 1984. *Rule-Based Expert Systems.* Reading, MA: Addison–Wesley.

[8] Caudill, M. 1991. Expert Networks. *Byte*, 16(10):108–116.

[9] Chen, S. 1987. Automated Reasoning on Neural Networks: A Probabilistic Approach. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 373–378.

[10] Drucker, H. 1990. Implementation of Minimum Error Expert System. In *Proceedings of the International Joint Conference on Neural Networks IJCNN'90*, San Diego, Vol. I, 291–296.

[11] Družecký, P. 1992. *Expert System for Medical Diagnostics.* Master Thesis, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University, Prague.

[12] Duda, R. O., and Reboh, R. 1983. AI and Decision Making: The PROSPECTOR Experience. In *Artificial Intelligence Applications for Business*, ed. W. Reitman. Norwood, NJ: Ablex Publishing Co.

[13] Eisenstadt, M., and Brayshaw M. 1990. A Knowledge Engineering Toolkit. *Byte*, 15(10):268–282, 15(12):364–370.

[14] Enbutsu, I., Baba, K., and Hara, N. 1991. Fuzzy Rule Extraction from a Multilayered Neural Network. In *Proceedings of the International Joint Conference on Neural Networks IJCNN'91*, Seattle, Vol. II, 461–465.

[15] *EXPSYS—Students' Software Project.* 1994. Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague.

[16] Fausett, L. V. 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.* Englewood Cliffs, NJ: Prentice–Hall International.

[17] Gallant, S. I. 1988. Connectionist Expert Systems. *Communications of the ACM*, 31(2):152–169.

[18] Gallant, S. I. 1993. *Neural Network Learning and Expert Systems.* Cambridge, MA: The MIT Press.

[19] Giarratano, J., and Riley, G. 1993. *Expert Systems: Principles and Practice.* Boston, MA: PWS Publishing.

[20] Handelman, D. A., Lane S. H., and Gelfand, J. J. 1989. Integration of Knowledge-Based System and Neural Network Techniques for Autonomous Learning Machines. In *Proceedings of the International Joint Conference on Neural Networks IJCNN'89*, Washington, Vol. I, 683–688.

[21] Haykin, S. 1994. *Neural Networks.* New York: Macmillan College Publishing Company.

[22] Hecht-Nielsen, R. 1990. *Neurocomputing.* California: Addison–Wesley.

[23] Hertz, J., Krogh, A., and Palmer R. G. 1991. *Introduction to the Theory of Neural Computation.* Lecture Notes, Santa Fe Institute Studies in the Sciences of Complexity, Vol. I, California: Addison–Wesley.

[24] Hořejš, J., and Kufudaki, O. 1993. Neural Networks with Local Distributed Parameters. *Neurocomputing*, 5(4–5):211–219.

[25] Ignizio, J. P. 1991. *Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems.* New York: McGraw–Hill.

[26] Ishikawa, M. 1996. Structural Learning with Forgetting. *Neural Networks*, 9(3):509–521.

[27] Jackson, P. 1990. *Introduction to Expert Systems.* California: Addison–Wesley.

[28] Jang, J. S. 1992. Self-Learning Fuzzy Controllers Based on Temporal Back Propagation. *IEEE Transactions on Neural Networks*, 3(5):714–723.

[29] Japanese Credit Screening Data. 1992. Machine Learning Repository, University of California, Irvine, *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/credit-screening/*.

[30] Kasabov, N. K. 1991. Expert Systems Based on Incorporating Rules into Neural Networks. Technical University in Sofia, Bulgaria.

[31] Kasabov, N. K., and Petkov, S. H. 1992a. Approximate Reasoning with Hybrid Connectionist Logic Programming Systems. In *Proceedings of the International Conference on Artificial Neural Networks ICANN'92*, eds. I. Aleksander and J. Taylor, pp. 749–752. Elsevier Science Publisher B. V.

[32] Kasabov, N. K., and Petkov, S. H. 1992b. Neural Networks and Logic Programming—a Hybrid Model and its Applicability to Building Expert Systems. In *Proceedings of the* 10^{th} *European Conference on Artificial Intelligence ECAI'92*, ed. B. Neumann, pp. 287–288. John Wiley & Sons.

[33] Kasabov, N. K. 1993. Hybrid Connectionist Production Systems: An Approach to Realising Fuzzy Expert Systems. *Journal of Systems Engineering*, 1:15–21.

[34] Kasabov, N. K., and Shishkov, S. I. 1993. A Connectionist Production System with Partial Match and its Use for Approximate Reasoning. *Connection Science*, 5(3&4):275–305.

[35] Kasabov, N. K. 1994. Connectionist Fuzzy Production Systems. In *Proceedings of the Fuzzy Logic in Artificial Intelligence: IJCAI'93 Workshop*, Chambery, France, Vol. 847 of LNAI, 114–127. Berlin: Springer Verlag.

[36] Kosko, B. 1987. Adaptive Inference in Fuzzy Knowledge Networks. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 261–268.

[37] Krysiak, A. 1991. Application of Parallel Connectionist Model in Medical Expert System. Institute of Computer Science, Polish Academy of Sciences.

[38] Lacher, R. C., Hruska, S. I., and Kuncicky, D. C. 1992. Back-Propagation Learning in Expert Networks. *IEEE Transactions on Neural Networks*, 3(1):62–72.

[39] Lin, C. T., and Lee, C. S. G. 1991. Neural-Network-Based Fuzzy Logic Control and Decision System. *IEEE Transactions on Computers*, 40(12):1320–1336.

[40] Lippmann, R. P. 1987. An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, 4(2):4–22.

[41] Mařík, V., Zdráhal, Z., Kouba, Z., and Lhotská, L. 1990. The FEL–EXPERT Project: Applications in Education. In *Proceedings of the CEPES-UNESCO International Symposium: Artificial Intelligence in Higher Education.* Berlin: Springer Verlag.

[42] Mařík, V., Vlček, T., Kouba, Z., Lažanský, J., and Lhotská, L. 1992. Expert System FEL–EXPERT version 3.5: Description and User's Manual. Technical Report TR–PRG–IEDS–06/92, FAW Linz–Hagenberg–Prague–Vienna.

[43] Mařík, V., Vlček, T., Šedivá, I., Maříková, T., Hyanek, J., Campr, V., Tjoa, A. M., and Gierlinger, Ch. 1993. FEL–EXPERT Applications—Some Pilot Case Studies. Technical Report TR–PRG–IEDS–13/93, FAW Linz–Hagenberg–Prague–Vienna.

[44] Mazný, M. 1995. *Integrating Rule-Based and Neural Approaches to Expert System Design.* Master Thesis, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague.

[45] McDermott, J. 1981. R1: The Formative Years. *AI Magazine*, 2(2):21–29.

[46] McMillan, C., Mozer, M. C., and Smolensky, M. 1991. Learning Explicit Rules in a Neural Network. In *Proceedings of the International Joint Conference on Neural Networks IJCNN'91*, Seattle, Vol. II, 83–88.

[47] Mechlová, S. 1994. *Applications of Neural Networks.* Master Thesis, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University, Prague.

[48] MIKE: Micro Interpreter for Knowledge Engineering. 1990. *ftp://hcrl.open.ac.uk/pub/software/src/MIKEv2.03/*.

[49] Miller, R. A., Pople, H. E., and Myers, J. D. 1982. INTERNIST–I, an Experimental Computer-Based Diagnostic Consultant for General Internal Medicine. *New England Journal of Medicine*, 307:468-476.

[50] Mitra, S., and Pal, S. K. 1995. Fuzzy Multi-Layer Perceptron, Inferencing and Rule Generation. *IEEE Transactions on Neural Networks*, 6(1):51–63.

[51] Moreno, L., Piñeiro, J. D., Sanchez, J. L., Mañas, S., Merino, J. J., Acosta, L., and Hamilton, A. 1995. Using Neural Networks to Improve Classification: Application to Brain Maturation. *Neural Networks*, 8(5):815–820.

[52] Mozer, M. C. 1987. RAMBOT: A Connectionist Expert System That Learns by Example. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 693–701.

[53] Narazaki, H., and Ralescu, A. L. 1992. A Connectionist Approach for Rule-Based Inference Using Improved Relaxation Method. *IEEE Transactions on Neural Networks*, 3(5):741–751.

[54] Nekovei, R., and Sun, Y. 1995. Back-Propagation Network and its Configuration for Blood Vessel Detection in Angiograms. *IEEE Transactions on Neural Networks*, 6(1):64–72.

[55] Omlin, C. W., and Giles, C. L. 1996. Extraction of Rules from Discrete-Time Recurrent Neural Networks. *Neural Networks*, 9(1):41–52.

[56] Rojas, R. R. 1996. *Neural Networks: A Systematic Introduction.* Berlin: Springer Verlag.

[57] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart and J. L. McClelland, Vol. I, pp. 318–362. Cambridge, MA: The MIT Press.

[58] Samad, T. 1988. Towards Connectionist Rule-Based Systems. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 525–532.

[59] Saito, K., and Nakano, R. 1988. Medical Diagnostic Expert System Based on PDP Model. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. I, 255–262.

[60] Šíma, J. 1992a. The Multi-Layered Neural Network as an Adaptive Expert System with the Ability to Work with Incomplete Information and to Provide Justification of Inference. *Neural Network World*, 2(1):47–58.

[61] Šíma, J. 1992b. Generalized Back Propagation for Interval Training Patterns. *Neural Network World*, 2(2):167–173.

[62] Šíma, J., and Neruda, R. 1993. Designing Neural Expert Systems with EXPSYS. Technical Report V–563, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague.

[63] Šíma, J. 1994. Generalized Back Propagation for Training Pattern Derivatives. *Neural Network World*, 4(1):91–98.

[64] Šíma, J., and Neruda, R. 1994. The Empty Neural Expert System and its Application in Medicine. In *Proceedings of the* 12^{th} *European Meeting on Cybernetics and Systems Research*, ed. R. Trappl, Vienna, pp. 1825–1832.

[65] Šíma, J. 1995. Neural Expert Systems. *Neural Networks*, 8(2):261–271.

[66] Šíma, J. 1996. Back-Propagation Is Not Efficient. *Neural Networks*, 9(6):1017–1023.

[67] Simpson, P. K. 1990. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations.* New York: Pergamon Press.

[68] Styblinski, M. A., and Meyer, B. D. 1988. Fuzzy Cognitive Maps, Signal Flow Graphs, and Qualitative Circuit Analysis. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 549–556.

[69] Taber, W. R., and Siegel, M. A. 1987. Estimation of Expert Weights Using Fuzzy Cognitive Maps. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II, 319–326.

[70] Yager, R. R. 1994. Modelling and Formulating Fuzzy Knowledge Bases Using Neural Networks. *Neural Networks*, 7(8):1273–1283.

[71] Yang, Q., and Bhargava, V. K. 1990. Building Expert Systems by a Modified Perceptron Network with Rule-Transfer Algorithms. In *Proceedings of the International Joint Conference on Neural Networks IJCNN'90*, San Diego, Vol. II, 77–82.