Designing Neural Expert Systems with *EXPSYS*

J. Šíma, R. Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Praha 8, CZ
e-mail: roman@uivt.cas.cz

July 28, 1993

Abstract

The user-friendly software product *EXPSYS* for PC compatible machines is presented as a tool for neural expert system design. The program allows to define types of expert inputs and outputs based on certain Pascal-like data types together with their inner coding. The core of the system is a generalized back propagation learning algorithm capable to create the neural knowledge base automatically from the set of possibly incomplete example inferences. The neural inference engine works with incomplete and irrelevant information and evaluates the resulting outputs by confidences. The *EXPSYS* also provides the simple explanation of the inference. This program, because of its generality, can be applied in arbitrary areas where the middle-complex expert system is needed to be created in a relatively short time.

Contents

1 Neural Expert Systems **2**

2 Program EXPSYS for the Neural Expert System Design **4**

3 Using EXPSYS **8**

3.1 System requirements and standards 8
3.2 Defining Data Types 9
3.3 Defining the Topology 10
3.4 Defining the Training Set 11
3.5 Learning the Network 12
3.6 Expert Checking 15
3.7 Inferences 16
3.8 Configuring the Program 17

A Some Environment Notions **18**

B Hotkeys **19**

Chapter 1

Neural Expert Systems

The great hopes originally invested in expert systems have not by far satisfied expectations. Research has brought only limited applicable results. Seeking a tool, which should make the representation of much knowledge possible as well as consistent and effectively usable, is one of the basic problems of artificial intelligence. The exact tools of the representation in the classical rule-based expert systems are not adequate for storing and handling inaccurate, fuzzy and intuitive expert knowledge. Therefore building such a consistent relevant knowledge base is a difficult matter and the average development time of a middle-complex rule-based expert system is approximately 12 to 18 months.

The alternative approach to the expert system design is connected with the rapid progress of the neural network theory. The theory of modeling nervous systems of the live organisms could be the inspiration for the way out of these knowledge representatation problems. The simulations of the simplified neural network models show symptoms which are similar to human intelligence, for instance the ability to learn and to generalize from examples as well as to store and to recall this extracted information. As we have already mentioned, the knowledge of an expert does not have the form of rules, it is rather a fuzzy abstraction of his experience. From this point of view, the application of the neural networks to represent the knowledge base seems the natural alternative. In this connections, we speak about so-called *neural expert systems* [3], [4]. In these systems the knowledge representation is based on the numerical synaptic weights corresponding to the connections among neurons. The main advantage of these systems is that the synaptic weights are computed automatically by a learning algorithm from the set of example inferences. That is why building these systems takes just a few weeks and the main work is done by the computer.

We have investigated the possibility of the application of the neural networks in expert systems. For this purpose we employ the multi-layered neural network trained with the back propagation learning algorithm [5], which is the most widely applied neural network architecture. For this architecture we have proposed the heuristics for the work with incomplete and irrelevant information and the simple conclusion's explanation.

Our theoretical results [6], [7], [8], [9], [10], [11] are currently used in the user-friendly software product called *EXPSYS* for the neural expert system design. This program, because of its generality, can be applied in arbitrary areas where the middle-complex expert system is needed to be created in a relatively short time. This ability of the program has been approved on a non-trivial real-world problem - the diagnostics and progress prediction of hereditary muscular disease. In this case, after the preparation of the clinical data from the medical documentation about patients, the neural expert system has been created automatically on the PC during 40 minutes. Then it determines correctly the output diagnosis from generally incomplete inputs. The explanation heuristics has unambiguously pointed out two mostly relevant inputs, which according to the expert's opinion are sufficient for the resulting output diagnosis determination. This illustrates the great generalization ability of the system, comparable to the human intelligence, and confirms the correctness of the heuristics proposal.

Chapter 2

Program EXPSYS for the Neural Expert System Design

In the particular design of the diagnostic expert system it is needed to choose the expert inputs and outputs which should best render a problem solved by this system. Program *EXPSYS* makes various types of expert inputs and outputs possible as well as various ways of their coding with respect to the needs of particular application. The possible types are *real (integer)* number, *scalar* type with user-defined values, and *set*. The domain of the real (integer) number is given by real (integer) interval. The domain of the scalar type is defined by the vector of possible values, and the domain of the set type is determined by the universe of possible elements given also by the vector of these elements. An example of a simple medical diagnostic expert system could have the following parameters:

```
INPUTS:   SORE-THROAT: scalar of (NO,YES)
          TEMPERATURE: real of <36,42>
OUTPUTS:  ILLNESS: scalar of (HEALTHY,COLD,ANGINA,INFLUENZA)
          INABILITY-TO-WORK: scalar of (NO,YES)
          MEDICAMENTS: set of (DROPS,ASPIRIN,PENICILLIN)
```

The values of inputs and outputs of the neural expert system are encoded by the states of the input and output neurons of the multi-layered neural network using *floating point*, *binary* and *unary* codes. Almost any combination of type and coding is allowed, with the exception of unary coded real type, which does not make good sense. Binary coded real type is in fact integer type. Just one neuron is used in the case of the floating point coding. In the unary and binary codes there are generally more coding neurons at disposal, according to the type definition. The natural combinations of the type and coding follow: the floating point coding of the real type, the binary coding of the set type and the unary coding of the scalar type.

To create the neural knowledge base, an expert formulates his knowledge in the form of typical example inferences instead of general rules. An example inference is a pair composed of the vector of the typical values of the expert inputs and the corresponding vector of the values of the expert outputs observed from the expert's reactions on these inputs. The set of example inferences should cover the basic required regularities of the studied area. Their combining and generalization is left to the learning process. For example in medicine the medical documentation about patients can be taken as a basis for the creation of the set of the typical example inferences. In this case the expert inputs correspond to the outcomes of various examinations and the expert outputs are diagnoses or medications recommended by the doctor. Continuing in our example, the set of example inferences can take the following form:

```
{ [ (NO,39.6), (INFLUENZA,YES,{DROPS,ASPIRIN}) ],
  [ (YES,37.2), (ANGINA,YES,{PENICILLIN}) ],...,
  [ (NO,unknown), (HEALTHY,NO,{ASPIRIN}) ] }
```

The values of the expert inputs and outputs can be generally unknown or irrelevant in the particular example inference as it has occured in the last element of the above-mentioned set. For example in the medical application the X-ray is not needed for the diagnosis of angina.

To start the learning process, we must first specify the topology of the multi-layered neural network which means the number of hidden layers and the number of neurons in them. These are estimated with respect to the practical problem. For example we can choose two hidden layers, the first one with a few more neurons than input ones and the second one with number of neurons equal to the average of the numbers of the output and first hidden neurons. Further we can edit this topology if the learning process is not successful.

The learning process itself is a long-term computationally exacting procedure with uncertain result which determines the synaptic weights of the multi-layered neural network from the set of example inferences. This process should be controlled by the user that can change various optional parameters during the computation. How to choose the appropraite values at the right time to minimize the learning error is an issue of experience.

After the multi-layered neural network has learned the given set of example inferences, it is needful to check wheather the created neural knowledge base is able to infer usable conclusions from it. This can be done by an expert, e.g. by testing reactions to the randomly generated sensible expert inputs, and through eliminating the incorrect network behaviour by learning correctly formulated example inferences in which the system fails. It is useful to iterate this process until the expert is satisfied by the network's ability to infer correct outputs. In the end of the expert checking of the neural knowledge base we determine the generalization confidences of the particular expert outputs by statistically measuring the percentage of right reactions of these outputs.

The user's view on the work of the inference engine can take the following form in our example. The user puts in some values of the inputs and the neural expert system provides the values of outputs with the corresponding confidences:

```
SORE-THROAT = YES, TEMPERATURE = unknown
 ---> ILLNESS = ANGINA              conf. = 0.77
      INABILITY-TO-WORK = YES       conf. = 0.81
      MEDICAMENTS = { PENICILLIN }  conf. = 0.62
```

The explanation heuristics of our neural expert system creates, for the given expert output, the list of the most relevant expert input values with the percentage influence measures. It is expected that these input values have the greatest influence on the inferred expert output. In our example the explanation can take the following form:

```
MEDICAMENT = { PENICILLIN } ? ---> SORE-THROAT = YES  100%
INABILITY-TO-WORK = YES ? ---> SORE-THROAT = YES  65%
                               TEMPERATURE = unknown  35%
```

The results of the explanation can be used not only for expert analysis, but it is also possible to debug the neural knowledge base by them. For example if we want to force some rule into the neural network and the system responds incorrectly in some cases with respect to this rule, we can create the appropriate example inferences using the explanation of failed answers. These examples, after being learned, regulate the system function correctly.

The user usually consults the particular problem in the interactive mode, and the neural expert system asks him questions for the unknown values of the expert inputs. The strategy of the selection of the appropriate question consists of finding such unknown expert input, that giving its value, the confidence and completeness of expert outputs would mostly increase after the next inference.

Chapter 3

Using EXPSYS

3.1 System requirements and standards

The program *EXPSYS* is written in Borland Pascal v. 7.0. The object-oriented event-driven library Turbo Vision v. 2.0 was used to develop the state-of-art user interface [2]. Therefore the program provides standard system of pull-down menus, status lines, hotkeys, windows and dialog boxes usual in any Turbo Vision application. Naturally, mouse support is also built in.

EXPSYS can be used on any IBM PC-AT compatible computer with processor i80286 or higher and operating system DOS 3.3 or later. It is a protected mode application, which uses standard Borland DOS Protected Mode Interface based on DPMI 0.9 specification enhanced with INT 21h support [1]. This allows the program to use the extended memory, if it is available (besides the conventional 640KB). The more memory is present, the more can be used to store the data. The DPMI server requires at least the processor i80286.

Even if the program is able to run on any AT computer, the i80386-based machine with numerical co-processor (or better) is strongly recommended, especially for the learning phase. The back-propagation learning algorithm usually requires from several minutes to several hours (even days) depending on the task and platform.

Typical work with *EXPSYS* consists of the followig basic steps:

- defining data types (i.e. the types of expert inputs and outputs)

- defining the topology
- defining the training set
- learning the network
- expert checking
- inferences

Some actions must obey the above sequence, some do not. Each item is in detail described in the following sections.

The reader should be familiar with certain notions from graphic user interface terminology, as menu, window, icon, dialog box, button, list box, etc. To use the *EXPSYS* some basic practical skills with the Borland-type interface is needed. For those who are not familiar with such environment the very brief cook-book is provided in the Appendix.

3.2 Defining Data Types

As was already said, one example inference consists of a pair *(input;output)*, where input and output can have (and usually do) more items. Each item represents a value of a user-defined basic type that corresponds to three type categories provided by the system - real, scalar and set.

In the beginning of the work with *EXPSYS* the user should define the structure of inputs and outputs of the particular expert system. Menu item `Data-types` collects all operations necessary for the work with these prototypes of example inferences. Let us focus on the creating of a new data type.

When the user chose the work with data-types, the dialog box containing two lists of types (one for input and one for output) appears on the screen. Initially both lists are empty. The user can add, modify and delete their items by the buttons placed beside. Each list has its corresponding buttons for editting and deleting focused item of the list and for inserting a new item either before or after the focused item.

In the list, each item is identified by its name and the number showing how many neurons is needed to encode the item. This number depends on the combination of the type and its coding. Floating coding always uses only one neuron. Binary coding needs $\lceil \log_2 N \rceil$ neurons to encode N possible values. It means it can encode an integer value from the domain interval having N numbers with $\lceil \log_2 N \rceil$ neurons. The same situation is in the case of scalar values of the N-element domain. When coding the values of the set of the cardinality N, also N neurons are necessary. Unary coding requires N neurons to encode N values (it corresponds to the case where just one neuron of the N-tuple fires). It means that unary coding of scalar values is rather straightforward. Unary coded set types of N-element domain requires 2^N neurons (one neuron for each subset of the domain set). Unary coded real types are not allowed.

When inserting or editting an item, the dialog box with the description of this type appears. The description consists of the item name, its type and coding, which is choosen by two sets of radio buttons. Each type category needs further specification. In the case of the real type user should enter the interval of correct values. Both set and scalar types requires the definition of their domains. This is done by a simple list of values separated by commas.

User can mark one of domain values as the implicit one by the character @ placed right in front of the value. The *implicit value* is automatically selected as default in the process of adding new elements to the training set (cf. Defining the Training Set). If none of the domain values is marked by @ the `Unknown` value will be implicit. This mechanism can be used only with the scalar type domain values, which in particular means one cannot mark the implicit set.

For example, if the domain for a scalar type is

```
{red, blue, green, cyan}
```

and the value `blue` should be implicit, the user enters:

```
red, @blue, green, cyan
```

Complete definition of the data types is stored on the disk with the name specified by user and the extension `.DAT`.

3.3 Defining the Topology

After defining the data types, the number of neurons in the input and output layer is computed as the sum of nerons necessary for coding all the inputs and outputs. It means that this part of topology is determined automatically and user cannot effect it.

But the user can (and should) effect number of hidden layers and number of neurons in them. This is done by the commands gathered in the menu item `Topology`. Topology should relate already defined data type.

The definition of topology is done by a simple dialog box consisting of a list whose each item defines the number of neurons in the corresponding hidden layer. The list is organized in the reverse order than the usual scheme of the network is presented. In our case the topmost list member relates to the hidden layer connected to the input layer.

After invoking the button `Change layer` the user can modify number of neurons in the hidden layer corresponding to the focused item of the list. This is done in the simple dialog box with one input line and confirmation and cancellation button.

Every invalid combination, such as any zero number followed by nonzero(s), is indicated and not allowed by the system. After succesfull definition the topology is saved on the disk to the file with extension `.TOP`.

3.4 Defining the Training Set

The next natural step after the definition of expert data types of the network is to build a collection of example inferences (i.e. the training set for the network). Menu item `Training-patterns` gathers all available training set operations.

Each training set is stored on the disk as a file with an extension `.PAT`. The training set can be considered as a sequence of cards, where each card contains one pair *(input;output)*. During the work with the training set, each card is presented as a dialog box which is assembled according to the definition of data type (cf above). Each line of the dialog represent one basic input item. It consists of its name and the input line for inserting the concrete values.

Inserting a number value is done simply by typewriting it, but to enter values of set and scalar type, the user should use the minibox which is placed beside each input line that contains values of set and scalar types. When the minibox is focused, user can browse its values with Up and Down arrows. Desired value can be placed to the corresponding input line by Ins key. All values can be deleted by Del. When creating an input of the set type new values are added to the current content of the input line, while in the case of scalar type the new value overwrites the preceeding one.

The work with entire cards representing one training pattern (example inference) is done with the help of buttons at the bottom of the dialog box. These allow to browse the training set by accessing the previous and next pattern, to insert new pattern after or before the current one and to delete the current pattern. For the movement throughout the training set the user can also use PgUp and PgDown keys.

3.5 Learning the Network

The network is trained by the generalized back-propagation algorithm [6] in order to find such neural network parameters (weights, gains) that minimize the error of network behavior (training error) with respect to the training set (i.e. encoded set of example inferences). Menu item `Learning|Back propagation` starts the learning process. First, the user should set number of the algorithm parameters that are described bellow. The options of these parameters should ensure that the training error converges to zero during the learning process. The parameters are all gathered in one dialog box and their initial values are set to coincide with the most frequent cases.

Weights and gains initialization: The network parameters are either initialized from the data file or randomly initialized - gains equal to the predefined value (see *Gains value*) and weights in absolute value less than predefined parameter which is usually chosen to be small positive real number (it is set in input line labeled `In int.`). To increase this parameter slightly can sometimes help with the convergence of the learning process.

Uniform gains: The gain parameters of the neural network may not be necassarily changed during the learning process and can retain constant, equal to the predefined value (*Gains value*). But their adaptation gives greater degree of freedom to the learning process and improves its convergence.

Learning only unlearned patterns: This makes learning only those training patterns possible that are still not learned by the neural network according to the preset criterion. The learning behavior of the neural network is analogous to the human learning in the way that the neural network just slowly forgets already learned knowledge so it is possible to focus on the remaining unlearned data. According to our experience this option is not too much strong tool to improve the convergence of the learning process but it can sometimes help to scatter network weights when they do not allow the adaptation to the last several unlearned patterns. However, before this it is always useful to check the consistence of the unlearned patterns with the rest of the training set due to the neural network can sometimes defend itself from contradiction possibly hidden in the training set.

Accumulated learning: This makes accumulation of the weight and gain decrements for the adaptation after the whole training cycle possible instead of the adaptation after each training pattern presented to the neural network. Our experience shows that the accumulated learning is more efficient approach.

Complete pattern learning: Employing again the analogy with human lerning this determines the following learning strategy. The neural network memorizes each pattern until it is learned according to the preset criterion. Of course, during learning next training patterns, the neural network, confused by the intensity of learning the particular pattern, probably forgets already learned knowledge. But repeating this process sufficiantly long it starts to converge.

Gains value: The predefined value of the gains for the initialization and for the uniform gains can be chosen here.

Epsilon: This is the most relevant parameter to control the convergence of the learning process. According to the experience it is useful to start the learning process with epsilon to be sufficiantly small positive real number to ensure the convergence and when the learning process successfully continues it is possible to increase it carefully in order to speed up its convergence. The excessive increase of epsilon can cause the divergence of the learning process.

Alpha: This paramater is almost of the same importance and use as the epsilon parameter. According to our experience this parameter can be normally approximately equal to one fifth of the epsilon and it is sometimes useful to increase it when the convergence slows down. But other strategies of its option are possible.

newpage

Gains espilon,

Gains alpha: The preceding *Epsilon* and *Alpha* parameters are used for the weights adaptation and these are exploited for the gains adaptation if it is needed. Our experience has shown that it is sufficiant to choose them the same.

Learning intensity: Because the variant *Complete pattern learning* can be too strong and cause the divergence of the learning process this option establishes the number of repeated learning of each training pattern. This approach does not have much good influence on learning accordning to our experience.

Error computation frequence: The training error is a relevant information to control the convergence of the learning process, but computing it very often slows down this process. That is why the user can determine the frequence of error computation which is the number of training cycles after that the training error is computed.

Weights saving frequence: To preserve the results of the long computation of the learning process from damage it is certainly necessary to save them on the disc several times during the computation. The user can determine the frequence of weights and gains back up which is the number of error computations after that the weights and gains are saved.

Time limit: Because the learning process can take several hours or even days it is useful to put in the time limit of the learning process after which the computation is interrupted.

After setting the parameters, the learning algorithm is started. The user is informed about the current state of learning by the window containing these values:

Training cycle no.: Each training cycle is a sequence of presenting the complete training set to the network.

Pattern no.: Specifies the currently learned pattern within the training cycle.

Error: The last computed error value. The typical progress of the error value is the alternation from the fast decrease passing to the long lasting period of the almost constant error value even to the temporal increase after that the fast decrease follows again.

Error decrease: The difference between the the preceeding and the last error value.

No.of unlearned patterns: Specifies how many training patterns are still not learned according to the preset criterion.

Unlearned patterns: Specifies the concrete numbers of unlearned patterns.

Time remaining: Counts down the time according to the *Time limit*.

During the learning user can pause the process by pressing the `Stop` button. In this case the system finishes all the computations necessary for correct termination. This may take a while depending on the training set and parameters. After pausing the dialog box with back-propagation parameters appears again. The parameters can be modified and the learning process resumed. When the learning process is halted by `Cancel` button, it stops immediately, but cannot be restarted.

According to the *Weights saving frequence* parameter, the complete weight set of the network is stored on disk with name corresponding to the training set and extension `.WGT`. The actual saving mechanism is more complicated and uses the temporary weights file with `.TMP` extension. The weights are first saved to this file and can be copied to the `.WGT` file only if the error corresponding to the `.TMP` weigts is lower.

3.6 Expert Checking

The system is able to check the behavior of the expert system by generating randomly expert inputs and testing the system responses. An expert can determine wrong cases and include them to the training set to improve expert system reactions. Expert checking is also needed for the computing of generalization confidences that are used in inference machine.

Expert checking is accesible via menu `Learning|Expert checking`.

3.7 Inferences

The inference machine of the expert system is available via the menu item `Inference`. It serves for three tasks:

- It can compute the output values according to (some of) input values. Together with the output values it also computes the corresponding confidences.
- It provides the inference explanation by marking the most relevant input values that have the greatest influence on the given output. The percentage influence measure is also computed and displayed.
- It can generate a question for the most relevant and so far unknown input that would help mostly to improve the system answer.

The inference machine is controlled by the dialog box containing the list of input values, the list of output values and buttons for editting input value, explaining and question generating. The list of input values initially consists of only unknown inputs. Output values are always computed according to the current inputs. Each output has a corresponding confidence value determining its reliability.

To edit the focused input, one should invoke the `Edit input` button. Then a small dialog box with a line similar to the one in training patterns dialog (see 3.4) appears and allows to insert or edit the value. After inserting the outputs and their confidences are automatically recalculated.

The explanation heuristics which is called after pressing the `Explain` button computes the influence measures for input values and displayes all values corresponding to specified outputs in the input list.

Question generating (button `Question`) routine simply sets focus to the input line which is unknown and should be filled in the most. In fact, it sorts some of unknown inputs into the sequence according to their relevance.

If the user is not able to fill in the most desired input, he/she can press the `Question` button again and the next relevant input will be focused, etc. After all inputs were tried, the sequence is presented again.

3.8 Configuring the Program

Some of the actions described above depend on the previous ones. It means one cannot e.g. define the training set without already defined data types. In practice it regards at least to load the appropriate data types file. To ease the work with the system, user can create so *called configuration file* that contains names of the data types file, topology file, training patterns file and weights and gains file that corresponds each other. After loading such configuration file (with the extension `.CFG`) no loading of other files is necessary.

The work with configuration files is provided in menu item `Files`, where user can create new file, load already defined configuration or view the currently loaded one. The format of the configuration file is simple as it consists only of four file names on separate lines, so it can be easily created in outside the system, too.

When running the *EXPSYS* , the user can add a configuration file name (with or without the extension `.CFG`) as a parameter. In this case the configuration is loaded automatically and the program does not start with the `About` window.

Other additional functions that might be of use are deleting a file, and jump to the Dos shell, all situated in menu item `Files`, and a set of commands for the work with windows without a mouse and a change of video mode. They are placed in menu item ≡ that can be easily reached also by `Alt-Space` combination.

Appendix A

Some Environment Notions

View is a rectangular area of the screen that acts as whole.

Window is a framed view which contains some text and can be moved and/or resized.

Dialog box is a not resizable window that usually serves to enter data. It contains various views such as input lines, radio buttons, check boxes or buttons.

Input line is a line for input of various alphanumerical data. It is a simple line editor and usually does some data validation.

Radio buttons represent a set of values among which only one at a time can be selected.

Check boxes represent a set of values that can be selected independently upon each other.

Button is a view that usually starts some action when it is invoked.

List box is a list of values that allows scrolling and focusing its items. It usually has a scrollbar position indicator.

Mini box is a list box that is only one line high and has no indicator. It is connected with an input line as it serves to insert a text to it.

Appendix B

Hotkeys

1. **General**

   F3 – Load configuration file

   F4 – New configuration file

   F5 – View/edit data types

   F6 – View/edit topology

   F7 – View/edit Training patterns

   F8 – Start back-propagation learning

   Alt-F8 – Expert checking

   F9 – Inference machine

   F10 – Menu

   Alt-Space – Menu item ≡

   Alt-X – Exit the *EXPSYS*

   Ctrl-F5 – Move window

   Ctrl-F6 – Next window

   Shift-F6 – Previous window

   Alt-F3 – Close window

2. **In dialog**

   Tab – Move focus to the next view in dialog

Shift-Tab – Move focus to the previous view in dialog

Ctrl-$\rightarrow$*, Ctrl-*$\leftarrow$ – in the input line designed for comma-separated values: move to the nearest right (left) comma

Ctrl-U – in the input line: insert the `Unknown` value

PgUp, PgDn – in the training patterns dialog: move to the previous (next) pattern

$\uparrow, \downarrow$ – in the minibox: move to the previous (next) item

Ins – in the minibox: move current value to the input line

Del – in the minibox: delete input line content

Bibliography

[1] Borland Pascal with Objects 7.0 - Language Guide, Borland International, Inc., Scotts Valley, 1992

[2] Turbo Vision 2.0 - Programming Guide, Borland International, Inc., Scotts Valley, 1992

[3] Caudill M.: Expert Networks,Byte, 1991, 16, No.10, 108 - 116.

[4] Gallant S.I.: Connectionist Expert Systems, Com. ACM, 1988, 31, No.2, 152 - 169.

[5] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning Representations by Back-Propagating Errors, Nature, 1986, 323, 533 - 536.

[6] Šíma J.: Generalized Back Propagation for Interval Training Patterns, Neural Network World, 1992, 2, 167 - 174.

[7] Šíma J.: Generalized Back Propagation for Training Pattern Derivatives, submitted for publication in Neural Network World, 1992.

[8] Šíma J.: Neural Expert Systems, submitted for publication in Neural Networks, 1993.

[9] Šíma J.: 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, 1992, 2, 47-58.

[10] Šíma J., Neruda R.: Neural Networks as Expert Systems, Neural Network World, 1992, 2, 775 - 784.

[11] Šíma J., Neruda R.: Neural Expert Systems, Proceedings of IJCNN Beijing, 1992.