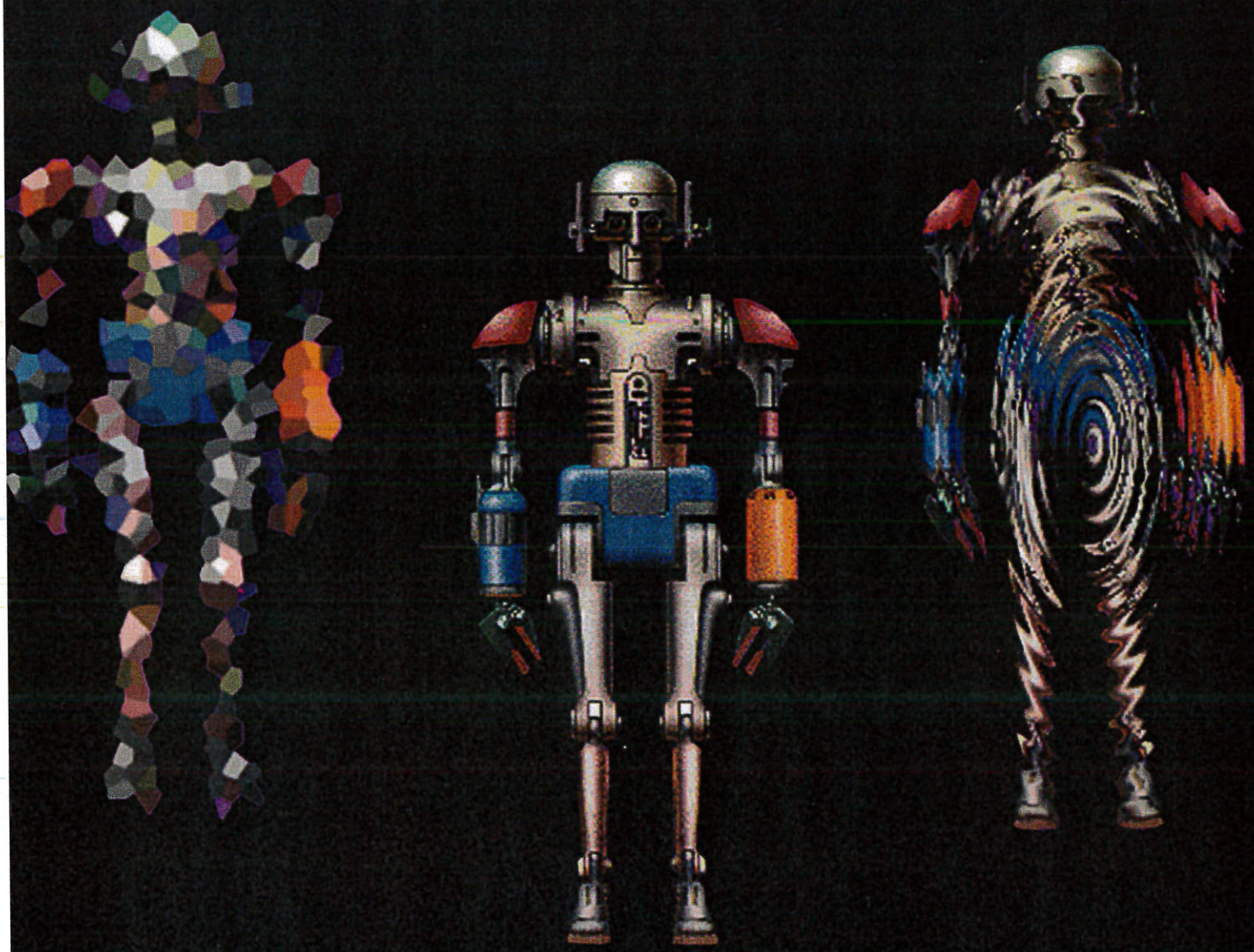
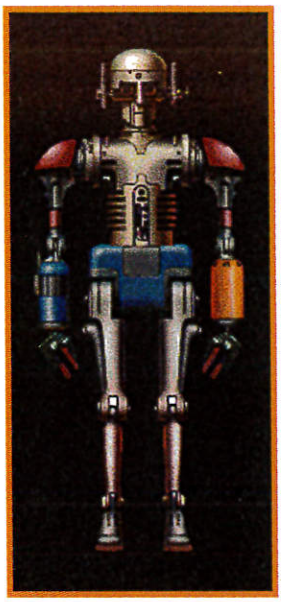
FUZZY SETS, NEURAL NETWORKS, AND SOFT COMPUTING



EDITED BY
RONALD R. YAGER ▪ LOFTI A. ZADEH

FUZZY SETS, NEURAL NETWORKS, AND SOFT COMPUTING

This book brings together the outstanding experts involved in a new area of research that is based on the confluence of genetic algorithms, fuzzy systems, and neural networks. Chapters by these researchers will give the reader a look into the future of a rapidly developing technology.



Included are these important topics:

- Fuzzy logic control
- Neural fuzzy systems
- Genetic fuzzy systems
- Process control
- Adaptive systems

These technologies will have an important influence on products and applications that are being developed today, and may become the basis of a new generation of intelligent computers that mimic the human ability to reason. No other book available combines these technologies to provide the most up-to-the-minute information on this subject.

The international experts contributing to this volume include:

D. Pileve, George Vachtsevanos, Benjamin Kuipers, Karl Astrom, Julie Dickerson, Bart Kosko, Gail Carpenter, Stephen Grossberg, James Buckley, Yoichi Hayaski, Patrick Eklund, John Joon Park, and others.

Lofti Zadeh is the originator of fuzzy set theory. He is currently Professor of Electrical Engineering and Computer Science at the University of California at Berkeley. A holder of many fellowships and awards, Dr. Zadeh is internationally known for his work in fuzzy systems. Dr. Ronald Yager is Professor of Information Systems at Iona College in New Rochelle, New York. He is an active researcher, a well-known conference organizer, and an international scholar. Dr. Yager holds the Meritorious Award for Scientific Research from the International Congress on Applied Systems.

Cover art by Ronald R. Yager
Cover design by Angelo Papadopoulos

ISBN 0-442-01621-2

9 780442 016210 90000

VAN NOSTRAND REINHOLD
115 Fifth Avenue, New York, NY 10003

Contents

12

Network Size versus Preprocessing

Patrick Eklund

Åbo Akademi University
Department of Computer Science
SF-20520 Åbo, Finland

12.1 INTRODUCTION

In his plenary talk at the 3rd IFSA meeting [13], Lotfi Zadeh predicted a strong development toward neuralism and suggested research on fuzzy logic to be placed somewhere between crisp logic and neural nets. Zadeh's talk can indeed be seen as a starting point for the development of *neural fuzzy systems* (NFSs), with the inevitable need for developing and invoking more sophisticated learning and tuning capabilities, even on silicon (second generation fuzzy controllers).

Neural fuzzyness today is a marriage between neural computing and fuzzy logic, with engineering activities ranging from diagnostics to control. A unifying NFSs paradigm is, however, still to be developed, where neuralism, logic, and control are understood in a common framework. An appropriate foundational understanding must accompany this search for engineering platforms. Today's purely mathematical discipline of fuzzy set foundations leans much on algebraic foundations [1], and are, from an application point of view, still rather sterile formal descriptions. Approaches to first order fuzzy logic [9] are promising for the development of fuzzy logic programming, but as a support for the construction of serious applications, work remains to be done.

Classic neuralists, fuzzy logicians, and control engineers show an obvious tendency to join forces, as we already witness in several conference organizations. From this we cannot but predict a bright future for NFSs, and a development toward establishing NFS as a well-founded discipline.

In this chapter we recommend going deep into case studies in order to achieve virtuosity in handling and combining available NFSs techniques and technologies. We emphasize *preprocessing* (or *fuzzification*) of data. By this we mean extraction of cut-off values and "softening," or fuzzifying, the cut-off barrier. In general, we are specifying system parameters, as informative ingredients in the decision support environment. When parameters, and thereby knowledge, have been extracted, our raw data are transformed into a corresponding logical form. Transformed data can (and should) be further used in more traditional neural network environments in order to optimize diagnostic performance. Note that for resulting networks and computations, code can often be generated.

We show how preprocessed data from single layer networks lead to faster convergence and better diagnostic performance in backpropagation networks. It is remarkable that single layer networks together with preprocessing can compete with multilayer networks that rely on mere linear transformations.

The chapter is organized as follows. Section 12.2 compares neural computational and fuzzy inferential viewpoints. In Section 12.3 we pinpoint the conventional use of neural networks. Section 12.4 initiates our search for a unifying NFSs paradigm with an identification of the logical nature of neural nets. Section 12.5 introduces basic learning rules for NFSs parameters, with supporting case studies in Section 12.6. In Section 12.7 we discuss problems related to tuning the inference mechanism.

Throughout the chapter we prefer to use the medical diagnostic paradigm of pattern recognition problems. We thus speak of "patients" for the input pattern, "diagnosis" for the output classification, and "diagnostic engine" for the network.

12.2 ENLIGHTENING THE BLACK BOX

A typical application of neural networks consists of "throwing in numbers" and experimenting with different network structures. Partial success is acknowledged as a discovery and unlocking of secrets of nature. The fact that resulting networks are huge and impossible to interpret or explain remains as a sign of complexity, and that "this shows we must have solved something really difficult."

We claim that network size can be drastically reduced by paying attention to preprocessing of raw numerical data. We can see that even specifying cut-off values instead of using raw data will not just reduce network size, but also give a better diagnostic performance than that of a multilayered network (Fig. 12.1).

Strategically, our suggestion is to preprocess before enlarging networks. In doing so, network structures are kept small and explainable. The possibility of explaining networks and its parameters is of utmost importance to practitioners both in medical and technical diagnostics.

Explanations and interpretations of the network are logical in nature. In many applications, such as diagnostic tasks, network summations are to be interpreted as incremence of evidence. We are asked to deliver inputs simultaneously, with data not available classified as "missing data." All inputs (also in hidden units) are used to calculate the weighted sum, which is activated and thereafter is fit to act as an input in further feedforward. Note that partial evidence cannot be

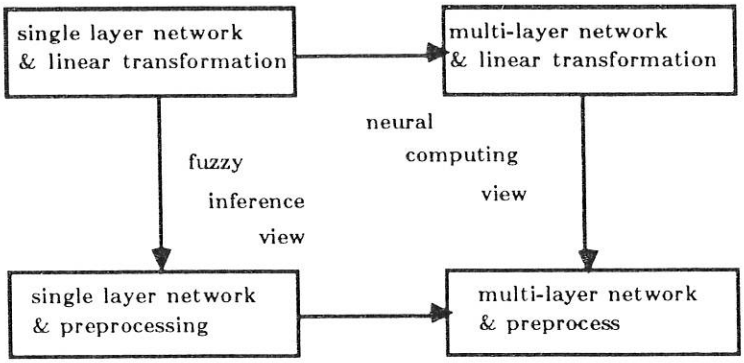


FIGURE 12.1. Fuzzy inference vs neural computing

estimated and saved for a separate incremental procedure. Formally, of course, this is possible. In this case we would activate partial sums and have to choose some function to aggregate all separate activations. From a (fuzzy) logical point of view, some properties should then be satisfied. One such is associativity of the incremence procedure, which is difficult to achieve with activations. Another aspect concerns the linearity of the weighted sum. Independent evidence will add to the result always with a static quantity regardless of the current measure of belief. When dealing with values in the unit interval we expect the speed of incremence to slow down the closer we come to maximal belief.

12.3 CONVENTIONAL USE OF NEURAL NETS

Neural networks provide an environment for classifying patterns without having to design and organize sophisticated knowledge-based architectures for the classifying machinery. Numbers are used as such in experiments with different network types/sizes, and diagnostic performance is evaluated with respect to choices of system parameters.

The schema for "walking through" an application is as follows [7]:

1, Collect data and transform into network input.
2. Separate data into training and test sets.
3. Select, train, and test the network.

Iterate (1) to (3) as appropriate.

4. Deploy the network in your application.

For neuralists, the first step is the most "magic." (A fuzzy practitioner considers this step to be "logic.") In this step, transformations like "1-of-N Codes" is applied to handle symbolic data, and "Histogram Equalization" overcomes difficulties with out-of-normal-range values. For the second step there are few rules concerning splitting of the data file into pieces for, respectively, training and testing. The third step is experimental, and is a generate-and-test approach to finding (what

FIGURE 12.2. Traditional use of neural networks.

in the end is believed to be) near-optimal networks. The fourth phase is often nicely supported by code generating modules in software packages.

According to the neural computing view, raw data are to be collected from a physical description of the disease. Data are inserted and used as such by the network. Preprocessing of data is a postactivity after the final network has been extracted. By preprocessing, the diagnostic performance might be further improvable (Fig. 12.2).

The fuzzy inference view is the opposite: Preprocess first, and organize your system based on knowledge extracted from the preprocessing phase. The preprocessing phase includes a reorganization (often symptom combinations) of the physical disease description. Data are pushed through (nonlinear) transformation functions, and resulting transformed data are used instead of raw data (Fig. 12.3). In subsequent sections we will see examples of the power of using transformed data.

12.4 THE LOGICAL NATURE OF NEURAL NETS

In diagnostics, a pattern (measurements or symptoms from patient) acts as *antecedents* from which we *infer* a classification (diagnosis) of the pattern (patient). This logical view calls for analyzing the neural computing mechanisms as a parameterized inference mechanism including uncertainty factors. Specifying and tuning parameters, such as transformation function cut-offs and slopes, constituted a numerical platform for knowledge extraction and description.

Referring to the discussion in the previous section, we can compare the neural computing and fuzzy inference views as follows:

Neural computing view	Fuzzy inference view
Parameters to *explore* In learning rules and learning rates, in transfer and error functions	Parameters to *learn* In preprocessing (or fuzzification) functions, in the inference mechanism, in the defuzzification function
Network type	Inference mechanism type
How many hidden layers?	
Preprocessing objective Convert a data record to numbers in a way that is easy for the network to do its job, usually only in the form of scaling and normalization	Preprocessing objective Transform data into a logical form, thereby extracting transformation function parameters as a basis for knowledge generation; consider the neural preprocessing as a *pre*preprocessing, usually performed by the expert, outside the logical apparatus

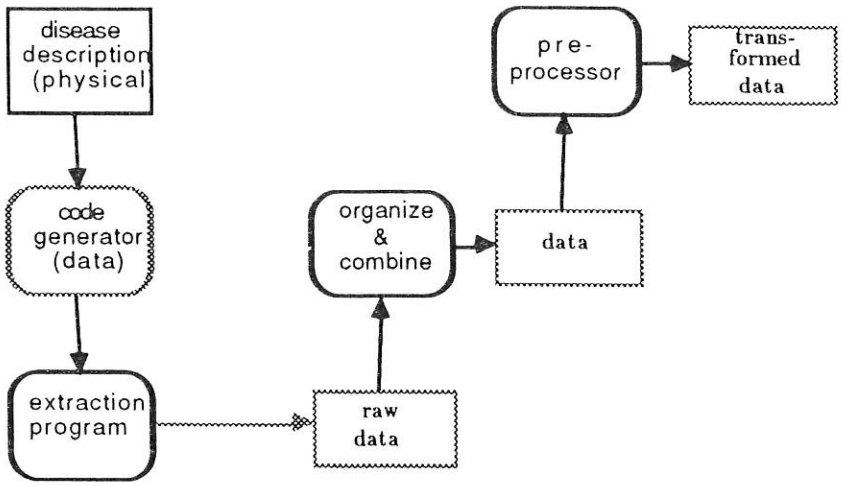


FIGURE 12.3. Preprocessing schema.

Our case studies on both *Nephropathia epidemica* and the *Arenaria interpres* will clarify the distinction.

The fuzzy inference view can architecturally be specified according to Figure 12.4.

12.4.1 Input

In any classification task we have a *measurement space* from which we receive *physical input* data. If and when organized (or prepreprocessed) into appropriate raw numerical form, data are *linearly transformed* into values in, say, the unit interval. We thus have *normalized physical input* as an initial data representation appropriate for our logical explorer.

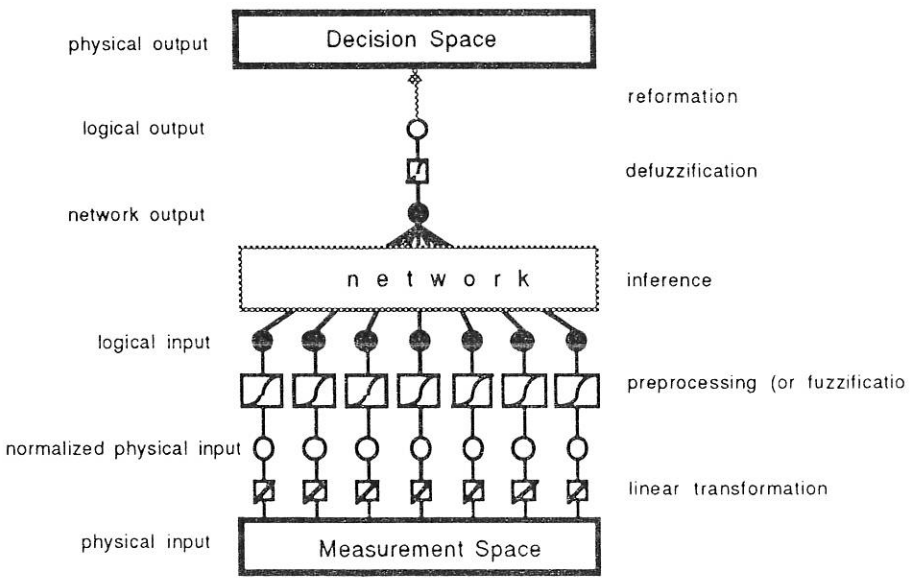


FIGURE 12.4. Logical activities in feedforward.

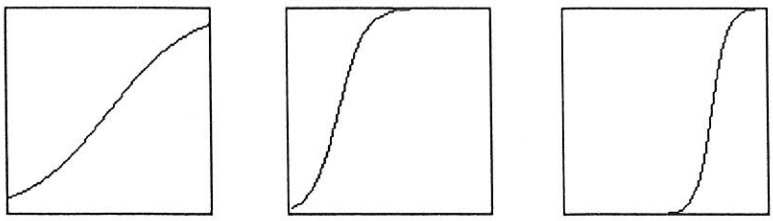


FIGURE 12.5. Typical stigmoid preprocessing functions.

The *preprocessing* (or *fuzzification*) functions transform unit interval physical input into unit interval *logical input*. The problem of using physical data is obvious. In medical applications, the significance of, i.e., the general symptom "fever" is related to the diagnosis under consideration. For the flu, just above 37°C already is very significant, whereas a patient with pneumonia usually has a higher fever. For specific symptoms, this fact is even more unavoidable. Further, a slope at a cut-off point specifies the speed by which a measurement around the cut-off changes significance. Using sigmoid transformation functions, Figure 12.5 describes almost linear transformation (Fig. 12.5a) lower significant values and (Fig. 12.5b) higher values significant with a steep slope at the cut-off (Fig. 12.5c).

Note that functions shapes as such are very informative for the expert. Thus, much of the knowledge extracted already resides in function parameters.

12.4.2 Inference

The logical input now is fed into the *network*. Here we are tempted to immediately shift to the neural computing style, and proceed to experiment with different network types. However, our recommendation is to start simply with a single layer. Again, simplicity results in fewer parameters, and thereby to a better defined knowledge extraction process. Moreover, several examples show that a perceptron together with preprocessing can perform as good as backpropagation using only normalized physical input. Only when simpler networks have been investigated, weights adapted and interpreted, and knowledge extracted, we should complete out task by using more conventional neural techniques (recall Fig. 12.1).

So far we have only a dangling logical nature in preprocessing functions. These functions, however, obtain a well-defined logical interpretation in describing the network as an inference schema. To begin with, note the almost religious nature of the weighted sum

$$net = \sum_i w_i s_i$$

which, with data in the unit interval, typically is activated by a sigmoid

$$f(net) = \frac{1}{1 + e^{-\phi(net - \theta)}}$$

where θ is the threshold value and ϕ represents the slope at the threshold in the activation function. Here we have almost an *a*logical schema where intermediate values shift back and forth between the unit and real lines, respectively. Input values start off at the unit interval. The weighted sum is in the real line, and therefore the sum must be activated back to the unit interval.

In Eklund et al [5] we have a framework where input aggregation and activations are lumped together into a logical notion of incremental evidence. The activated weighted sum is rewritten in a more logical form according to

$$\Phi_{i \in I \varphi_i}(w_i, s_i)$$

where $\varphi_i(w_i, s_i)$ denotes the semantics of implication, and Φ calculates an incremence of evidence. Typically, Φ has a generator function, and is an assembly of co-t-norms. This opens up a wide spectrum of parameterizations of the weighted sum, together with interpretations of activation functions in terms of co-t-norms and/or OWA-operators [12]. On the other hand, well-established activation functions, like the *tanh* function used in the $[-1, 1]$ interval case, correspond to co-t-norms hardly ever appearing in the fuzzy literature.

Note that both $\Phi_{i \in I}$ and φ_i can be transformed to range in the unit interval. In fact, rewriting and parameterizing the activated sum in this fashion provide the basic elements for presenting neural feedforward in a logic framework. Learning algorithms also carry over. See Eklund et al. [5] for details. Syntactic schemas to conjoin fuzzy logic and neural network computational paradigms have been described [4].

12.4.3 Output

The *network output* is now *defuzzified* by some activation-type function. The resulting *logical output* is passed through a *reformation* process, and we obtain the final *physical output* in the *decision space.*

The decision space can be binary, in the sense of classifying patients with respect to a diagnosis according to *yes* or *no*. The space can be "quasiternary" in the sense of a diagnosis "maybe," where a patient remains under observation, and the *yes/no* decision is postponed. Nonbinary, finite decision spaces exist, i.e., for the *thyroedea*, where we have a typical 5-value space in diagnostic classifications according to {hypo, subhypo, normal, subhyper, hyper}. In diagnostics, continuum decision spaces are rare.

12.5 TUNING SYSTEM PARAMETERS

The delta rule can be derived in a general feedforwarding framework, as has been emphasized by many authors.

In general, for inputs $s_1, \ldots, s_n$, and outputs $o_1, \ldots, o_m$, a parameterized input–output function can be written as

$$(o_1, \ldots, o_m) = \mathcal{M}[\alpha_1, \ldots, \alpha_k](r_1, \ldots, r_n)$$

where $\mathcal{M}[\alpha_1, \ldots, \alpha_k]$ represents the parameterized system function.

Given a pattern $p = (r_1, \ldots, r_n)$, a parameter α_j, $j = 1, \ldots, k$, is updated according to

$$\alpha_j := \alpha_j \oplus \Delta_p \alpha_j$$

and

$$\Delta_p \alpha_j = -\frac{\partial E_p}{\partial \alpha_j}$$

where E_p is a function measuring the error of the output with respect to some target (expected) values $t_1, \ldots, t_m$. Typically, E_p is the mean square sum

$$E_p[\alpha_1, \ldots, \alpha_k] = \sum_{i=1}^{m} (t_i - o_i)^2$$

Of course, several error functions are available, and update expressions are derived accordingly.

The parameters $\alpha_1, \ldots, \alpha_k$ reside, respectively, in the fuzzification, inference, and defuzzification modules of the system (or mechanism) $\mathcal{M}$. We shall denote physical normalized input by r_i (raw data) and logical preprocessed inputs by s_i (symptoms).

Let now f be a differentiable activation function. For

$$o_p = \mathcal{M}[w_1, \ldots, w_k](s_1, \ldots, s_n) = f\left(\sum_{i=1}^{n} w_i s_i\right)$$

we obtain

$$\Delta_p w_i = \eta f'(o_p)(t_p - o_p)s_i$$

Considering the weighted sum as a (pseudo)logical disjunction, this rule should be seen as adjusting parameters in the inference part of the network. Since weights appear all over the real line, the operation $\oplus$ can be taken to be the ordinary sum.

Parameters in fuzzification (preprocessing) relate to specification of membership functions (or transformations of inputs). As an example, suppose we are tuning sigmoids

$$g[\alpha, \beta](r) = \frac{1}{1 + e^{-\beta(r-\alpha)}}, \qquad \beta > 0$$

Then

$$o_p = \mathcal{M}[w_1, \ldots, w_k, \alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k](s_1, \ldots, , s_n) = f\left(\sum_{i=1}^{n} w_i g[\alpha_i, \beta_i](r_i)\right)$$

and the corresponding delta rules become

$$\Delta_p \alpha_i = -\frac{\partial E_p}{\partial \alpha_i} = -\eta(t_p - o_p)o_p(1 - o_p)w_i g[\alpha_i, \beta_i](r_i)(1 - g[\alpha_i, \beta_i](r_i))\beta_i$$

and

$$\Delta_p \beta_i = -\frac{\partial E_p}{\partial \beta_i} = \eta(t_p - o_p)o_p(1 - o_p)w_i g[\alpha_i, \beta_i](r_i)(1 - g[\alpha_i, \beta_i](r_i))(r_i - \alpha_i)$$

respectively.

Note that we must update αs and βs according to

$$\alpha_i(t + 1) = \alpha_i(t) \oplus_1 \Delta_p \alpha_i(t)$$

and

$$\beta_i(t + 1) = \beta_i(t) \oplus_2 \Delta_p \beta_i(t)$$

where $\oplus_1$ and $\oplus_2$ are corresponding (homeomorphic) operations in the unit interval and positive real line, respectively.

For the defuzzification module, with the sigmoid activation function, $\phi = \beta_{\text{act}}$, $\theta = \alpha_{\text{act}}$, corresponding update rules are

$$\Delta_p \alpha_{\text{act}} = -\frac{\partial E_p}{\partial \alpha_{\text{act}}} = -\eta(t_p - o_p)g[\alpha_{\text{act}}, \beta_{\text{act}}](net)(1 - g[\alpha_{\text{act}}, \beta_{\text{act}}](net))\beta_{\text{act}}$$

and

$$\Delta_p \beta_{\text{act}} = -\frac{\partial E_p}{\partial \beta_{\text{act}}} = \eta(t_p - o_p)g[\alpha_{\text{act}}, \beta_{\text{act}}](net)(1 - g[\alpha_{\text{act}}, \beta_{\text{act}}](net))(net - \alpha_{\text{act}})$$

respectively.

Defuzzification in diagnostics has pragmatic aspects such as choosing the output either to indicate decision support, or to conclude an expert judgment. In control, defuzzification is always a signal, and the defuzzification schema tends to be bound to the application.

12.6 CASE STUDIES AND DEVELOPMENTS

Several simulations show that parameter tuning is both fast and reliable.

For real applications in medical diagnostics, we will describe results for diagnosing *hemorrhagic fever with renal syndrome*. A similar analysis (described elsewhere) has been carried out for urinary tract infections, thyroedea, and infarctus cordis.

The generality of the medical diagnostic paradigm is further exemplified with an ornithological case study in "diagnosing" whether the *turnstone* breeds on a certain island in the Åland archipelago (situated between the Finnish and Swedish mainlands).

We also briefly mention technical diagnostics, and, in particular, refer to *diesel engine monitoring and fault avoidance.*

Finally, we describe a product development program aimed at design and implementation of a clinician's workstation for doing NFSs analysis, together with related application building.

12.6.1 Nephropathia Epidemica

The clinical course of the NE has been described [8].

The differential diagnosis between NE and other infectious diseases and renal diseases is often difficult. The syndrome has 27 signs and symptoms, which are sufficient for drawing diagnostic conclusions, but none of them is diagnostic alone. For more details, see Eklund et al. [2, 3].

For the NE we carried out the following tuning experiment. First, we started off measuring diagnostic performance with cut-offs fixed at $\alpha = 0.5$ as compared to cut-offs suggested by the best expert. Here we fixed slope values at $\beta = 10$. The expert version wins. Second, we tuned cut-offs from the $\alpha = 0.5$ points, and compared the success of tuning slopes with, respectively, cut-offs tuned from $\alpha = 0.5$ and cut-offs fixed by the expert. Now it turns out the completely tuned version shows highest diagnostic rates (see Table 12.1).

Note that we do *not* draw the conclusion that the expert can be deleted from the knowledge acquisition process. The expert still provides information about transformation function types (i.e., sigmoids or bell shapes, sigmoids being increasing or decreasing), and is in the end the evaluator of the usefulness of constructed decision support systems (*not* expert systems).

Figure 12.6 shows a sample of tuned transformation functions for the NE.

From the above we see how parameter tuning is successful even without starting off at some expert information about cut-offs and/or slopes. Moreover, when using preprocessed data, instead of physical normalized input, we can increase convergence speed for the learning phase also in traditional neural computing (see Fig. 12.7). For the NE, we used a backpropagation network with 5 nodes in the hidden layer, together with 28,000 patterns (100 iterations of 280 patients) for training.

TABLE 12.1 Tuning with and without Expert

	Correctness Rate (%)	Mean Absolute Error Rate
Cut-offs fixed at 0.5	85.36	0.27
Cut-offs given by expert	87.86	0.25
Slopes tuned, expert cut-offs	88.57	0.22
Slopes tuned, cut-offs tuned	89.29	0.21

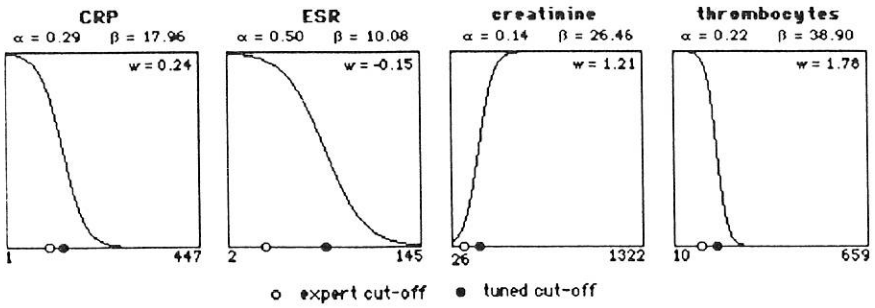


FIGURE 12.6. Sample transformation functions (NE).

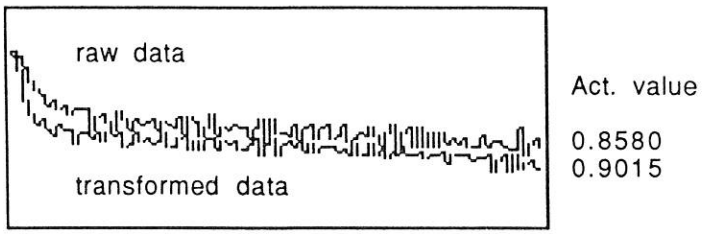


FIGURE 12.7. Convergence of learning (NE).

12.6.3 Arenaria Interpres

The turnstone (*Arenaria interpres*) breeds on islands in the Åland archipelago, and is quite particular concerning island characteristics. Typically, the favorite island is small and rocky, has a higher maritimity value (PCA gradient), and is already a breeding place for pairs of small gulls.

The landtype symptom is a good example how preprocessing relates to the neural "1-of-N Codes" approach. We used a 5-scale according to the following: 1 = bare rock, 2 = rocky islet, 3 = bare rock surrounded by reed, 4 = partially forested, and 5 = completely forested. The turnstone accepts types 1 and 2, and rejects types 3, 4, and 5. According to the expert ornithologist, the rejection of 4 and 5 is clear, whereas type 3 might be acceptable. In a corresponding transformation function, using $\{0, 0.25, 0.5, 0.75, 1\}$ as normalized physical input, the slope should be steep, and the cut-off between type 2 and 3, but somewhat closer to 3. In our tuning, the slope became $\beta = 20.07$ and the cut-off value converged to $\alpha = 0.40$, perfectly corresponding to reality.

Figure 12.8 shows transformation functions for all eight symptoms.

The correctness rate for the single-layer preprocessing network was just above 80%. Interestingly enough, a backpropagation network (5 nodes in the hidden layer) using logical inputs from the preprocessing network showed a correctness rate just under 80%. The same network using physical normalized input performed even worse.

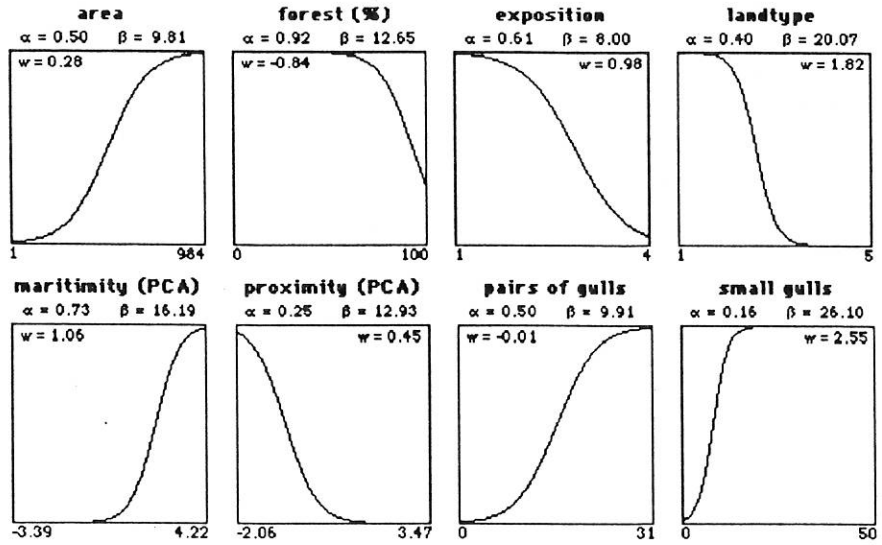


FIGURE 12.8. Transformation functions (turnstone).

12.6.3 Technical Diagnostics

Heuristic cut-off values are widely used also in technical systems. Taking measurings like temperatures, and transforming numerical values respectively to *high*, *normal*, or *low* values depending on cut-off points, has been taken as a self-evident initial step in a straightforward attribute-value approach. However, this extraction means a severe loss of information in the diagnostic process. Referring to our discussion above, we can easily see how similar input shape techniques can be used also in these contexts. Contrary to medical applications, the number of symptoms is usually smaller for particular conclusions, although the number of measurement points might be very large.

We have applied out techniques for diagnostics and monitoring of medium speed diesel engines, using in power generation in ships and power plants. The diagnostic system FAKS is operational on a train ferry sailing between Hangö (Finland) and Travemünde (Germany).

12.6.4 The DiagiaD Architecture

The *DiagiaD* workplan is part of the research program GeDeMeDeS, which aims at developing generic NFSs software. Apart from diagnostics, also control applications and packages are being developed.

The *DiagaiD* architecture is described in Figure 12.9.

The workplan has three distinct submodules. Initially, the diagnostic problem must be specified, and corresponding data extracted from hospital databases. The key engineering task here is to create the code generator for MUMPS programs, thereby giving straightforward access to databases. When raw data are available, the preprocessing machinery takes over, the principles of which are described in

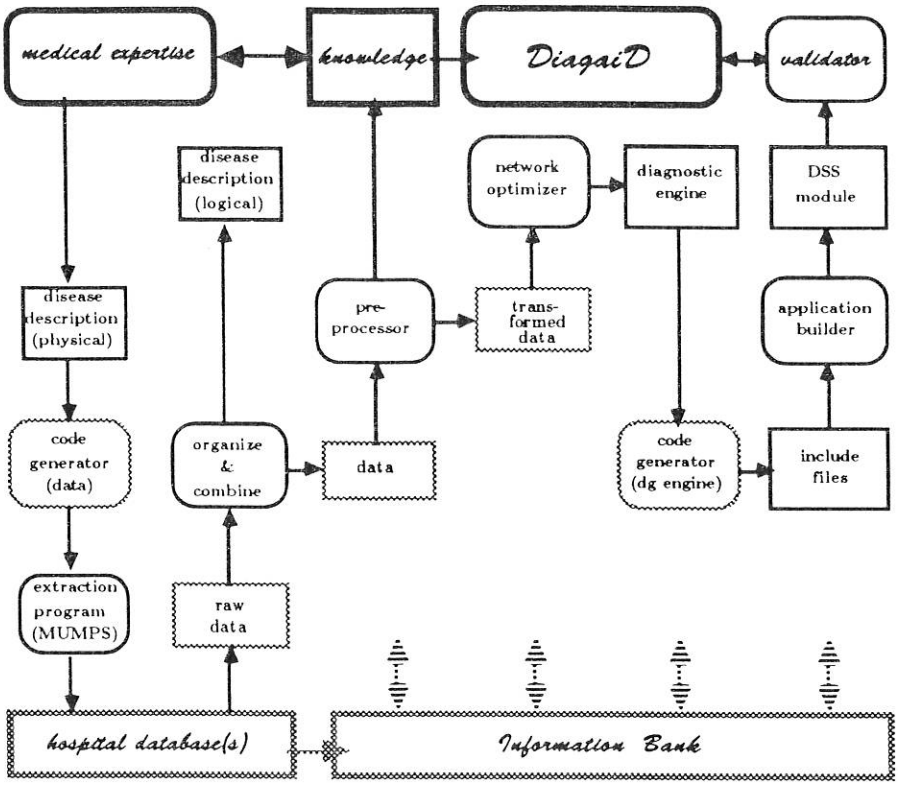


FIGURE 12.9. *DiagainD* architecture.

preceding sections. Finally, the application builder typically uses generated C coded include files in its application builder environment (ranging from prograph™ for the Macintosh™, to independent XVT™ platforms with portabilities for free), and aims at minimizing engineering time and effort in creating final DSS modules.

12.7 TUNING THE INFERENCE MECHANISM

In the preprocessing module we have seen how the choice of function type still remains to be specified by the expert. For the inference mechanism, we also have to select appropriate types that correspond to actual inference procedures.

In NE, symptoms like "fever" and "myopia" exemplify the situation. Fever occurs in almost every NE patient. It is not a very significant symptom, not even if the fever value is high. However, if a patient does not have fever, we might initially suspect that the patient does not suffer from any infectious disease. In the case of myopia, the symptom is very characteristic for NE. However, very few NE patients have myopia. Thus, myopia not being present gives very little information in diagnosing the NE.

Only some very general observations can be done. A positive value on a weight together with an almost *yes* indication on the corresponding symptom almost always results in a diagnostic indication in favor of the disease (see Fig. 12.10).

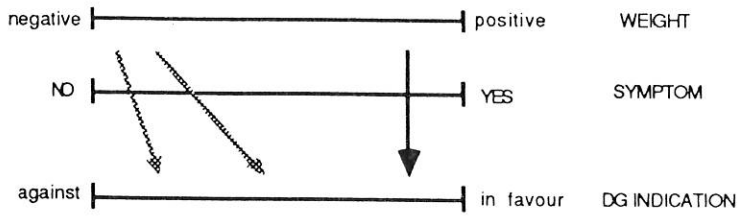


FIGURE 12.10. Inference type.

For a symptom not being indicated, the diagnostic indication can either directly react on the magnitude of the weight, or else stay neutrally at "cannot say" regardless of the weight value.

Thus, the choice of the inference function is much more elaborate than as for the preprocessing functions. Table 12.2 presents a summarizing (incomplete) framework for the type selection.

Note also how the weighted sum inferencewise behaves differently depending on the domain of normalized values being [0, 1] or [−1, 1]. For the [0, 1] interval, we get $\varphi_i(+, -) = 0$, whereas for the [−1, 1] interval we would have $\varphi_i(+, -) = -$.

A complete examination of different inference function types, together with their diagnostic performance in real applications, is outside the scope of this chapter, but will be described in forthcoming papers.

12.8 CONCLUSIONS

The topic of NFSs is diagnostics and control through neural interpretations of fuzzy sets. The focus is, and should be, on applications. We emphasize integration of learning and acquisition techniques as key engineering tasks. Openness and curiosity for supporting foundational aspects are, however, also required from the engineering side.

In this chapter we specifically propose an architecture for a generic tool that supports both data analysis and development of diagnostic modules. The architecture provides a computerized environment for (semi)automatically creating diagnostic software packages, starting from data collecting phases through data

TABLE 12.2 Inference Function Characteristics

w_i	x_i	$\varphi_i(w_i, s_i)$
+	+	+
+	−	0 or −
−	+	−
−	−	0 or +

analysis and knowledge acquisition phases all the way to the final engineering of the module under consideration.

ACKNOWLEDGMENTS

Work is carried out within the GeDeMeDeS-project, supported by the Technology Development Centre (TEKES), Helsinki.

References

1. C. C. Change. Algebraic analysis of many-valued logic. *Trans. Am. Math. Soc.* 88, 467–490, 1958.
2. P. Eklund and J. Forsström. Diagnosis of nephropathia epidemica by adaptation through Łukasiewicz inference. In *COMPUTATIONAL INTELLIGENCE, III—The International Conference on Computational Intelligence 90*, Milan, Italy. N. Cercone, F. Gardin, and G. Valle, eds. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 237–246, 1991.
3. P. Eklund and J. Forsström. A generic neuro-fuzzy tool for developing medical decision support. Manuscript, 1992.
4. P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Trans. Neural Networks*, special issue of the *FUZZ-IEEE '92*, San Diego, March 1992, 3(5), 815–818, 1992.
5. P. Eklund, T. Rüssanen, and H. Virtanen. On the fuzzy logic nature of neural nets. *Proc. Neuro-Nimes '91*, Nimes, November 4–8, 293–300, 1991.
6. J.-S. R. Jang. Self-learning fuzzy controllers based on temporal back propagation. *IEEE Trans. Neural Networks*, special issue of the *FUZZ-IEEE '92*, San Diego, March 1992, 3(5) 714–723, 1992.
7. C. C. Klimasauskas. Neural networks: Application walkthrough. Tutorial, Neuro-Nimes '91, Nimes, November 4–8, 1991. (Contains: Applying neural networks, Parts I-V, reprinted from *PC/AI Magazine*.)
8. J. Lähdevirta. *Nephropathia epidemica* in Finland: A clinical, histological and epidemiological study. *Ann. Clin. Res.* 3 (Suppl 8), 1–154, 1971.
9. V. Novak. On the syntacto-semantical completeness of first order logic, I, II. *Kybernetica* 26, 47–66, 134–154, 1990.
10. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, Vol 1. J. L. McClelland and D. E. Rumelhart, eds. Cambridge, MA: MIT Press, 1986, 318–364.
11. L. Wang and J. M. Mendel. Back-propagation fuzzy systems as nonlinear dynamic system identifiers. *Proc. FUZZ-IEEE*, March, 1992.
12. R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybernet.* 18, 183–190, 1988.
13. L. Zadeh. The coming age of fuzzy logic. Plenary talk at *3rd IFSA*, Seattle, August 6–11, 1989.