PROCESS IDENTIFICATION THROUGH MODULAR NEURAL NETWORKS AND RULE EXTRACTION

BEREND JAN VAN DER ZWAAG, KEES SLUMP
Dept. of Electrical Engineering, University of Twente
P.O.Box 217, 7500 AE Enschede (The Netherlands)
T: *+31.53.489.2842,* ***F:*** *1060,* ***E:*** *{b.j.vanderzwaag, c.h.slump}@el.utwente.nl*

LAMBERT SPAANENBURG
Dept. of Mathematics & Computing Science, Groningen University
P.O.Box 800, 9700 AV Groningen (The Netherlands)
T: *+31.50.363.3925,* ***F:*** *3800,* ***E:*** *l.spaanenburg@cs.rug.nl*

Monolithic neural networks may be trained from measured data to establish knowledge about the process. Unfortunately, this knowledge is not guaranteed to be found and – if at all – hard to extract. Modular neural networks are better suited for this purpose. Domain-ordered by topology, rule extraction is performed module by module. This has all the benefits of a divide-and-conquer method and opens the way to structured design. This paper discusses a next step in this direction by illustrating the potential of base functions to design the neural model.

Keywords: Process Model, Neural Networks, Modularity, Rule Extraction, Functional Base.

1 Introduction

Process Identification is a major part of Control Theory, where the (partly) unknown process must be monitored and modeled before it can be controlled. This has been pursued through numerous mathematical schemes[1] but also through fuzzy logic[2] and neural networks.[3] The observations on the process (temperature, sound, vibration, flow, etc.) are assumed to become available as signals, though lately also (spectral) images have come into use.

From centuries of scientific research, some basic knowledge on the process behavior is available as natural laws. Years of experience in operating the process also produces a degree of understanding: the operator knowledge. Where this has not been made operational by scientific research, fuzzy modeling provides a means to capture such knowledge for analysis and simulation.

However, processes tend to change over time. Hence a lengthy analysis based on the single set of historic measurements will fail to properly model the current situation. The model must be formally proven to be robust or periodically refreshed by observing the measurement data. It has been shown in various publications, that neural networks can be used to capture reality. However, neural networks may be hard to learn and combinatorial difficult to understand. Hence neural networks are often only used to provide operator assistance on a strategic level.

This paper sets out to explore the possibilities to provide in-line assistance to the process modeler. It is based on advances in two directions. Neural networks are used to capture the process knowledge in a fusion of existing knowledge and measured data. Such networks are modular to guarantee learnability and to allow for knowledge re-use. Using a large number of small nets in stead of a single one suppresses the combinatorial explosion of the search space from which the rule extraction on monolithic neural networks suffers. This can be further supported by the introduction of base functions: domain-specific pieces of common truth that help to construct the model by reading from a heterogeneous network with frozen parts.

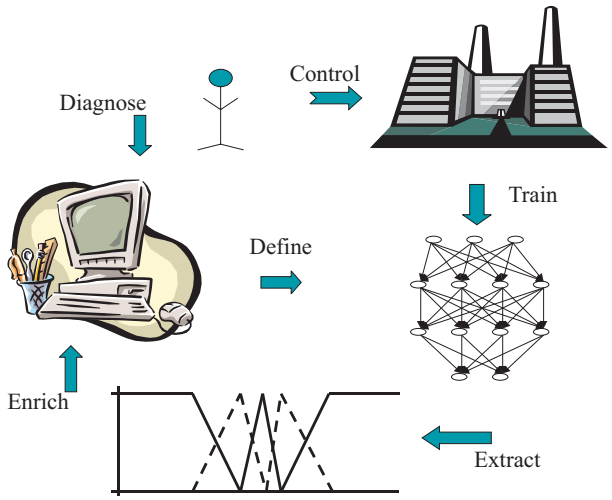

Figure 1. The knowledge–in–the–loop architecture is based on the iterative improvement of the process model through the extraction of the knowledge within the (post-)training neural network.

2 Modular Neural Networks

Compositions of neural networks have been studied for some time now to increase the capacity and accuracy of neural networks.[4] Though claimed to provide a better performance, the fundamental difficulty in network training[5] remains a major concern and several studies have been made for solutions to this problem.

Modular networks typically consist of a network of interconnected neural networks (also called modules or rule blocks) that each solve a sub-problem.[6] For this approach, extensions to the standard back-propagation learning algorithm are necessary; for instance, techniques to schedule the learning of the modules to prevent unlearning, as proposed by Spaanenburg.[7]

Hierarchical networks go one step further in the compositional sense.[8] Each node in a neural network may again be a neural network, or a specialized function. So a node's evaluation function is implemented by a neural network, while preserving the weights on its inputs from the upper layer. It is sometimes difficult to distinguish these two notions, as a modular network may have a hierarchical construction while a hierarchical construction may locally use a modular function.

The major benefit of modular networks follows from the inherent *separation of concerns*. The disreputed failure to learn in monolithic neural nets is caused by the presence of conflicting features, where the feed-forward arrangement tends to compromise instead of to select. During initial learning this leads to plateaus or local extrema in the error space; later on it may cause unlearning or catastrophic forgetting.[5] Separating such conflicts and solving them in separate modules before assembling the overall network solves such problems. This is similar to the effect of feature preprocessing,[9] which reputedly involves more than 80% of the project effort.

Modular networks can be based on a natural partition of the problem space reflecting the domain knowledge. Where such knowledge is not fully available, but a linguistic description is feasible, fuzzy modeling can be applied. Using a multi-block fuzzy representation, the fuzzy model will have a structure that mimics the domain structure and which can block-by-block be transformed into a modular neural net.[6]

When the process deviates too much from the model, special gates and guards may be required to check whether a module remains within the assigned validation area. In a *gating network*, each module receives only the data within the assigned validation area, whereas in a *guarded network* it receives all data but it is monitored whether the module behavior does not deviate too much from the expectations.[10]

Spaanenburg *et al.*[11] monitor the errors on the signals between the modules. Each module is trained in isolation and connected when the inter-module error becomes small. Vice versa, when the network is post-trained and the error appears to grow, the connection is broken again. This method requires that the error size can be qualified as being "too large". This is not always easy to do.

It has already been observed[5] that a conflict-free neural network has a stable and reproducible learning behavior. Similar observations while attempting to map neural networks on small micro-controllers have led to the understanding that irreproducible behavior is caused by a re-orientation of the neural network in its selection of hidden features.[12] When a neural network has to change the selection of hidden features in order to optimize the output function, this takes considerably more learning time than when it merely has to adapt the output weights.

Such has two consequences. The first is that learning time can be used as a measure to activate modules during training.[13] In a time-ordered schedule the modules start to learn while being part of the overall network; the timing of activation is decided on the momentary learning times per module (Figure 2). The second consequence is that abnormal behavior can be detected by monitoring the learning time.[14]

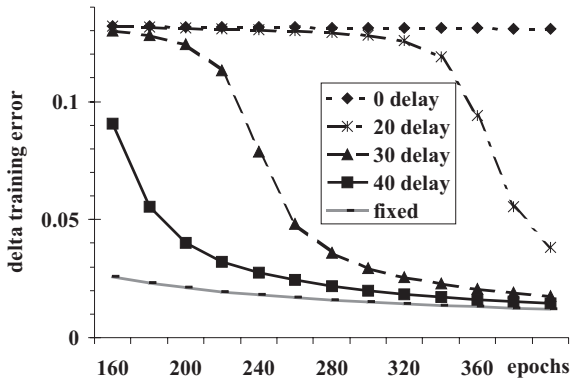

Figure 2. Impact of a small delay in the start of module activation, according to Venema *et al.*[13]

This is shown by vanVeelen *et al.*[15] for novelties in process identification on the network level, but can also be applied on the module level. This measure provides therefore an easy means to guarantee model integrity during the (post)learning of autonomous functions such as agents. But such networks are still hard to prove correct by the lack of documentation on the stored knowledge.

3 Knowledge Extraction

Compared to the amount of literature on theory and applications of neural networks, information on the analysis of neural networks is scarce and mostly limited to either *sensitivity analysis* or *rule extraction*. Sensitivity analysis[16] is a nonparametric statistical analysis technique, occasionally applied to neural networks, but more often appearing in other mathematical modeling and decision-making systems. Rule extraction[17] from neural networks originates from the field of symbol processing methods, which is rule-based rather than case-based. Neural network behavior described in sets of rules can provide insight into how the network comes to an answer.

Literature also gives some alternative approaches. Feng[18] uses multi-layer Perceptrons for parameter estimation and takes a first step to analyze the hidden units. Worth and Spencer[19] describe a neural network for tactile sensing. After learning, the weight vectors of all hidden units have the same structure, but the authors did not find any correlation between the relative sizes of the weight vectors and the outputs of the hidden units. VanderSteen[20] indicates that the correlation between weight changes provides more information. Mitchell[21] presents a method for interpreting the role of the hidden neurons geometrically. For the case in which function approximation is of most interest over a bounded region of the input space, the author shows that this interpretation may be used to check for redundancy among hidden units.

In general, neural networks with unsupervised training merely reorganize the input space by a winner-takes-all strategy, so analyzing them after training becomes fairly simple: an investigation into the reorganized input space reveals how the network has restructured the input space. Analyzing neural networks trained under supervision is far more complicated, as they operate by non-linear vector manipulation. They compromise rather than select and build a set of internal features that are not easily related to the input features.

Sensitivity analysis is a nonparametric statistical analysis technique[16] with the general idea to investigate the effect that perturbations in the input space have on the outputs, thus determining the sensitivity of the outputs to the inputs.[22] This is usually done for every input and every output, resulting in a matrix of sensitivities. These can then be used to determine whether any insignificant inputs can be ignored. In the neural network case, if a sensitivity analysis is performed on the hidden neurons, it could be used for pruning if some hidden neurons appear to have no influence on the network outputs.[23] Fu and Chen[24] use sensitivity analysis to investigate the generalization capability and error-correcting property of a neural network. Another example of the use of sensitivity analysis of neural nets is given by Choi and Choi.[25]

A weakness of sensitivity analysis is the fact that many applications of neural networks operate in high-dimensional input spaces, which would result in large sensitivity matrices that are hard to interpret. Another drawback is caused by the often strong non-linearity of neural networks. This can cause outputs to have many different sensitivities over the full range of particular inputs which in turn complicates the determination of the minimal value representation width or crisping.[26]

Craven and Shavlik[27] distinguish two approaches to *reasoning* on multi-layer neural nets: the *de-compositional* approach and the *pedagogical* approach (validity-interval analysis[28]). The de-compositional approach is to extract rules for each hidden and output unit separately, thus providing a certain transparency, whereas the pedagogical approach enables the extraction of rules that directly map inputs to outputs for a network as a whole, thus basically not opening the "black box". Pedagogical techniques are typically used in conjunction with a symbolic learning algorithm. The basic motif is to use the trained neural network as an example generator for the learning algorithm.[17]

Such approaches are applicable both in the Boolean and in the fuzzy domain. Due to its nature, Boolean rule extraction is mainly used in problems with discrete-valued features. Fuzzy rule extraction can be applied to both problems with discrete-valued features and those with real-valued features. However, there are problem domains where solutions cannot easily or comprehensively be described in sets of rules or decision trees, due to, e.g., high dimensionality of the input space or very large sets of independent features. Crisping is required to discretisize the value space, but such may shudder the carefully constructed balance, that comprises the internal knowledge storage.[26]

4 Base Functions

Mainly sensitivity analysis and rule extraction methods have been used to analyze neural networks, but the previous section makes clear that these can only be applied in limited subsets of problem domains. This can be remedied by identifying base functions with which users in a given domain are already familiar, and to describe trained networks, or parts thereof, in terms of those base functions. This will provide a comprehensible description of the neural network's function and, depending on the chosen base functions, it may also provide an insight into its inner "reasoning".

This concept is not so surprising, as many applications are based on a small number of arithmetic primitives. For instance, Ter Brugge *et al.*[9] describe the development of a stitch-weld tester. After a lengthy evaluation of various input features for the neural classifier, the acceptable solution is still reflecting the physical principle of operation: a dampened oscillation caused by reflections over different path lengths. In other words: it is based on a sine-wave and an exponential decay function.

Domain-specific analysis of neural networks through base functions will not only provide insight into the in- and external behavior of neural networks and show possible limitations of neural networks in particular applications, but it will also lower the acceptability threshold for future users unfamiliar with neural networks. Further, domain-specific neural network analysis methods that utilize domain-specific base functions can also be used to optimize neural network systems. An analysis in terms of base functions may even make clear how to (re)construct a superior system using those base functions, thus using the neural network merely as a construction advisor.

For many problems in certain domains, such as linguistics and decision theory, the common domain-dependent base functions could be chosen to be *if–then* rules or decision trees, in which case the analysis reduces to rule extraction. For the image processing domain, an example of neural network analysis using base functions has been discussed by van der Zwaag.[29] Table 1 lists a few problem domains where neural networks have been successfully applied. Possible base functions are presented for each of these domains.

Table 1. Some application domains with potential domain-specific base functions.

application domain	potential base functions
signal processing (1-D)	basic operational filters
digital image processing (2-D)	differential operators
general classification problems	feature map regions (cp. Kohonen SOFM)
decision theory	if-then rules (i.e., rule extraction as a special case of the proposed method)
control theory	basic control operators

5 Domain Development

Many production processes in the heavy industry are based on a judicious mixture of physical and chemical receipts. It is the shady world between molecular physics, metallurgy and mechanical engineering. Such processes are not always easily understood, let alone modeled. It is not that the fundamental knowledge is missing but rather that the dimensions of reality are not easily accounted for. More often than not the production line must be carefully adapted and tuned by an interdisciplinary team.

The 3-tier modeling technique has become very popular in modern computing science as it allows to match the needs of the application to the promise of the technological fundament. Of course, nature is too complicated to be summarized in merely three layers. Comparatively simple things as the ISO/OSI digital network model take already a 2-level hierarchy of 3-tier composites. We conjecture that the modular, physically plausible 3-tier can be iteratively detailed by a hierarchy of self-similar 3-tiers to handle a higher problem complexity.

A suitable domain model for production processes can be defined along the lines of the 3-tier concept. The basic rules emanate from micro-corpuscular considerations, where the attractive and distractive forces between the atomic elements are handled. The macro-corpuscular view is an abstraction thereof but this relation is of a stochastic nature. Where the present is not a mirror of the past, the predictive power of stochastics is sometimes overrated. In turn, the macro-corpuscular view can be abstracted to application-oriented phenomena. Such a domain model is shown in Figure 3.

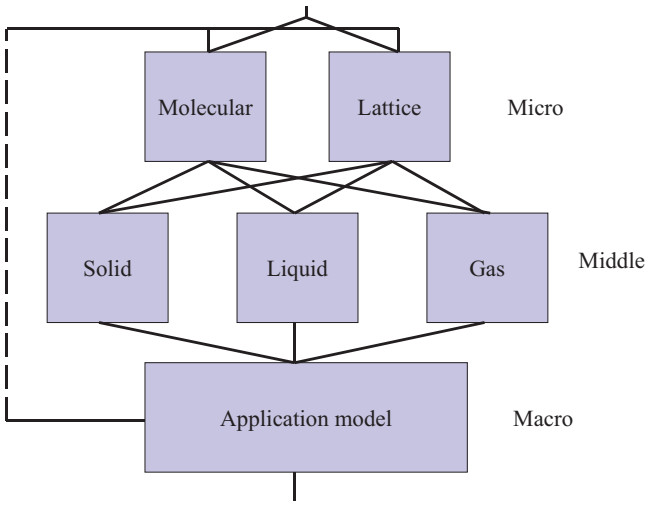

Figure 3. The process domain model.

The proper choice of base functions allows to introduce existing knowledge rather than to start from blind learning. The micro layer uses existing molecular formulae as base functions in a neural setting, because the precise formulation for the various phases lacks robustness. The application model is a regular neural network that reflects the itinerary through the phases and uses a simple sigmoid. The middle layer glues these two together and is based on conventional physical principles to relate pressure, time and volume. It uses the 2^{nd}-order differential transfer function as proposed by Meijer.[30]

Striking in the domain model is the role of the phases. The various ways in which matter can be present is usually taken as the first module layer. This is similar to the situation with weather prediction where the seasons are used to derive the prediction.[10] Such an architecture makes it hard to distinguish between a cool day in summer and a warm day in winter time; simultaneously transitional regions as spring and autumn become hard to handle. It has been shown that a first distinction between warm and cold days before defining seasons gives a better model. Along the same reasoning it only complicates things to start directly from the different phases to handle the various itineraries of the overall process and it is required to provide a proper base in molecular physics through the phase diagram.

The above process domain model has been suggested for the case of a metal scrap furnace, where the exhaust fumes are measured in controlling the process to optimize the industrial product and minimizing the environmental damage. Lack of facilities to measure directly on the process makes it necessary to bring all available information into one model and to enhance this model from experience (Figure 1).

A typical measurement is shown in Figure 4. The relation with the itinerary through the phase diagram is not immediately comprehensible, but the principle P-V-T behavior of metal scrap is known. This makes that the basic requirements for training the network are fulfilled and similarly shaped curves can be routinely handled.

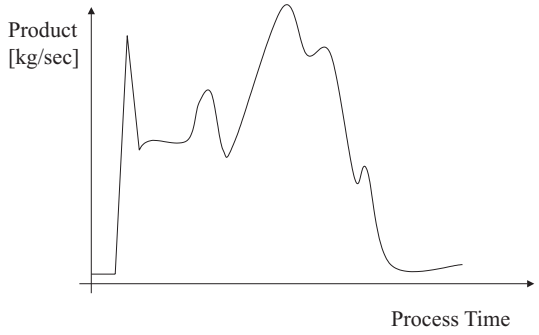

Figure 4. Typical indirect exhaust measurement.

However, as the aggregation changes at seemingly arbitrary moments, the feedback in Figure 3 between the application and the molecular physics layer is mandatory to characterize the future exhaust level on basis of the itinerary drift.

The methodology will be most attractive for the characterization of production processes, where the nature and location of the process make a direct measurement of process variables difficult, if not impossible. In such cases, sensors like the camera or the audiometer can be used to measure non-electrical process parameters and carry them into the input domain where a neural net can operate.

References

1. Trentelman, H.L. and Willems, J.C. (1993), *Essays on Control: Perspectives in the Theory and Its Applications*, Birkhaeuser (Boston).
2. Lam, H.K., Leung, F.H.F., and Tam, P.K.S. (2002), "A switching controller for uncertain non-linear systems," *IEEE Contr. Syst. Mag.*, Vol. 22, No. 1, pp. 7-14.
3. Narendra, K.S. and Parthasarathy, K. (1990), "Identification and control of dynamical systems using neural networks," *IEEE Tr. NN*, Vol. 1, No. 1, pp. 4-27.
4. Auda, G. and Kamel, M. (1999), "Modular neural networks: a survey," *Int. Journal of Neural Systems*, Vol. 9, No. 2, pp. 129-151.
5. Barakova, E. (1999), *Learning Reliability: a Study on Indecisiveness in Sample Selection*, Ph.D. thesis, Rijksuniversiteit Groningen, Groningen, Netherlands.
6. Spaanenburg, L., Jansen, W.J. and Nijhuis, J.A.G. (1997), "Over multiple rule-blocks to modular nets," *Proceedings Euromicro'97*, pp. 698-705.
7. Spaanenburg, L. (2000), "Knowledge fusion in modular neural networks," *Proceedings of the NC2000*, pp. 356-362.
8. Keegstra, H. *et al.* (1996), "Exploiting network redundancy for lowest-cost neural network realizations," *Digest ICNN'96* (Washington D.C.), pp. 951-955.
9. terBrugge, M.H. *et al.* (1995), "On the representation of data for optimal learning," *Proceedings ICNN'95* (Perth, Western Australia), Vol. VI, pp. 3180-3184.
10. Venema, R.S. (1999), *Aspects of an Integrated Neural Prediction System*, Ph.D. thesis, Rijksuniversiteit Groningen, Groningen, Netherlands.
11. Spaanenburg, L., Jansen, W.J., and Nijhuis, J.A.G. (1997), "Injecting functions in neural nets by controlled dedication," *Proceedings ECCTD'97* (Budapest, Hungary), pp. 1108-1113.
12. Spaanenburg, L., Ter Haseborg, H.M.G., and Peng, W. (2001), "Trimming neural networks for embedded intelligence," *Int. Journal of Knowledge-based Intelligent Engineering Systems*, Vol. 5, No. 3, pp. 171-178.
13. Venema, R.S. and Spaanenburg, L. (2001), "Learning feed-forward multi-nets," *Proceedings ICANNGA'01* (Prague), pp. 102-105.
14. Spaanenburg, L. (2001), "Unlearning in feed-forward multi-nets," *Proceedings ICANNGA'01* (Prague), pp. 106-109.

15. vanVeelen, M., Nijhuis, J.A.G., and Spaanenburg, L. (2000), "Emergence of learning methodology for abnormality detection," *Proceedings BNAIC'00* (Kaatsheuvel), pp. 283-292.
16. Rios Insua, D. (1990), *Sensitivity Analysis in Multi-Objective Decision Making*, Lecture Notes in Econ. and Math. Systems No. 347, Springer Verlag, Berlin.
17. Andrews, R., Diederich, J., and Tickle, A.B. (1995), "A survey and critique of techniques for extracting rules from trained artificial neural networks," Neurocomp. Research Centre report, Queensland University of Technology, Australia.
18. Feng, T.-J. (1991), "Backpropagation networks for parameter estimation," Tech. Report, University of Twente, Enschede, Netherlands.
19. Worth, A.J. and Spencer, R.R. (1989), "A neural network for tactile sensing: the Hertzian contact problem," *International Joint Conference on Neural Networks*, part I, pp. 267-274.
20. vanderSteen, E.W. et al. (2001), "Sampling casts plasticity in adaptive batch control," *Proceedings ProRISC'01*, pp. 646-651.
21. Mitchell, J. (1992), "A geometric interpretation of hidden layer units in feedforward neural networks," *Network*, Vol. 3, pp. 19-25.
22. Klimasauskas, C.C. (1991), "Neural nets tell why," *Dr. Dobbs's Journal*, April, pp. 16-24.
23. Engelbrecht, A.P. and Cloete, I. (1996), "A sensitivity analysis algorithm for pruning feedforward neural networks," *IEEE Int. Conf. on Neural Networks*, Vol. 2, pp. 1274-1277.
24. Fu, L. and Chen, T. (1993), "Sensitivity analysis for input vector in multilayer feedforward neural networks," *Proceedings IEEE Int. Conf. on Neural Networks*, Vol. I, pp. 215-218.
25. Choi, J.Y. and Choi, C.-H. (1992), "Sensitivity analysis of multilayer perceptron with differential activation functions," *IEEE Transactions on Neural Networks*, Vol. 3, No. 1, pp. 101-107.
26. Blattner, J., Neuer, S., Spaanenburg, L. and Nijhuis, J.A.G., "Optimizing fuzzy rules by neural learning," *Digest Int. Conf. on Mathematical and Intelligent Models in System Simulation MISS'93* (Brussels), pp. 238-246.
27. Craven, M.W. and Shavlik, J.W. (1994), "Using sampling and queries to extract rules from trained neural networks," *Proceedings of the Eleventh International Conference on Machine Learning ML-94* (Rutgers University, NJ), pp. 37-45.
28. Thrun, S.B. (1993), "Extracting provably correct rules from artificial neural networks," Tech. Report IAI-TR-93-5, Institut für Informatik III, Universität Bonn.
29. Van der Zwaag, B.J. (1992), "Analysis of neural network edge detectors," report no. 92M185, University of Twente, Enschede, Netherlands.
30. Meijer, P.B.L. (1996), *Neural Network Applications in Device and Sub-circuit Modeling for Circuit Simulation*, Ph.D. thesis, Eindhoven University, Eindhoven, Netherlands.