A Metamodeling Approach Based on Neural Networks.

Article · January 1996

Source: DBLP

A METAMODELING APPROACH BASED ON NEURAL NETWORKS

Henri Pierreval

LIMOS UMR 6158

French Institute of Mechanical Engineering (IFMA)

Campus des Cézeaux, B. P. 265

F-63175 Aubière, Cedex, France.

ABSTRACT

Simulation metamodels are models of simulation models. Several techniques exist to build metamodels, such as multiple regression. We are interested here in discussing an application of neural networks to the development of simulation metamodels. The network learns from a training set, designed from simulation experiments. The resulting network is capable of predicting the output values of the simulation model, corresponding to given input values, without having to simulate the behavior of the system over time. In comparison to the direct use of simulation, the computing times, as well as the memory needed, are drastically reduced. These advantages allow neural network metamodels to be used to support manufacturing decisions, such as manufacturing scheduling and control. This approach is illustrated through an example of a manufacturing job shop.

KEY WORDS

Simulation, neural networks, metamodels, back-propagation, manufacturing.

1. INTRODUCTION

Simulation is now a widely recognized technique for decision support, frequently used to answer "what-if" questions. Parameters describing important characteristics of the system under study are presented as input values to the model, in order to evaluate the outputs, with regards to desired performances. If the outputs do not correspond to these desired performances, parameters are changed and new simulation runs are performed. In this trial and error process, the use of simulation may present some inconveniences. In particular, it often takes a long time to obtain the results. This may not be acceptable when simulation is used as a decision support to manage or control an existing system that is waiting for a decision. It may be also unacceptable if the simulation model is embedded as a component of a problem solving software, such as an expert system or a production management system. Moreover, if developed with such simulation languages as SLAM II, SIMAN or GPSS IV, the simulation component of the software would require a large amount of memory.

Because of such problems, simulation models can be replaced by other kinds of models. The first alternative is to use a mathematical model. Unfortunately, due to their often limited assumptions, mathematical models are generally limited to a class of systems, so that for more complex cases these tools may not be relevant. The other alternative is to use a simulation metamodel, i. e. a model of the simulation model. Several techniques exist to build simulation metamodels. This paper is intended to discuss and to illustrate a metamodeling approach based on back-propagation neural networks (NNs). The NN metamodel may be used to replace the simulation model for decision making support, or may be embedded in a more specialized software, such as a software for the control and monitoring a given system, or a production management software.

The paper is organized as follows: first, the main notions of simulation metamodels are recalled. Then, an approach based on back-propagation neural networks to build metamodels is presented and illustrated through an

example of a manufacturing job-shop. The advantages and limitations of this approach are finally discussed.

2. SIMULATION METAMODELS

Due to the restrictive assumptions of mathematical methods, simulation is a very popular analysis and design tool. It is also widely used as a decision support tool in monitoring and controlling complex systems, such as flexible manufacturing systems. Unfortunately, simulation is often time and memory consuming. Moreover, the relationships between the inputs and the outputs may be difficult to analyze. Due to these problems, it may be more reasonable to use another model, which approximates the simulation model, and which is simpler than the simulation model itself. Over the last decades, this statement has led to a growing research effort in simulation metamodels. A *simulation metamodel* (or repro-model, or auxiliary model) is a model of the simulation model, which can be more efficiently and effectively employed in an operational or system evaluation context than the original model [1], [2], [3]. This approach is summarized in Figure 1.

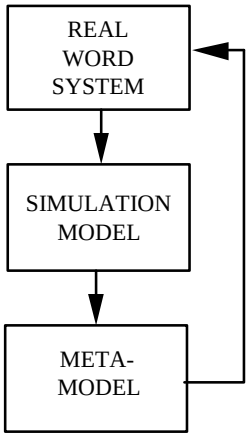

Fig. 1. Modeling and metamodeling.

Simulation metamodels are now more and more used, in particular in manufacturing, which is one of their privileged application areas. Several methods have been proposed to formalize and construct metamodels (see for example: [1], [2], [3], [4], [5], [6]. In general, metamodels are used either to perform sensitivity analysis, to optimize a system [7], to identify important factors [8], or to gain understanding about the relationships between the inputs and the outputs of a system [9].

We are interested in this paper in using metamodels in order to reduce the computing costs induced by simulation models, for such applications as real time decision support systems (e. g., scheduling and control). Such a purpose is a primary issue in the operation of manufacturing systems [10]. In this respect, a metamodeling approach based on back-propagation neural networks is suggested and presented in the next section.

3. A NEURAL NETWORK APPROACH

3.1 General aspects of back-propagation neural networks

Due to their powerful learning capabilities, NNs are more and more used in various application domains. In the following, we will concentrate on the back-propagation paradigm that is frequently used in practice. However, other NN approaches are also suited to our purpose.

A network is composed of nonlinear computational elements called cells (or neurodes), organized in layers. There are three kinds of layers: the input layer, the hidden layer(s) and the output layer. The NN's architecture may be specified using notation of the form L_1-L_2- .... -L_f, where L_1 denotes the number of cells in the input layer, and L_2, ..., L_{f-1} denote the number of cells in each of the hidden layers. Each cell of a layer is connected to each cell of any previous and following layers by links, which represent the transmission of information between cells. Figure 2 gives an example of a NN topology.

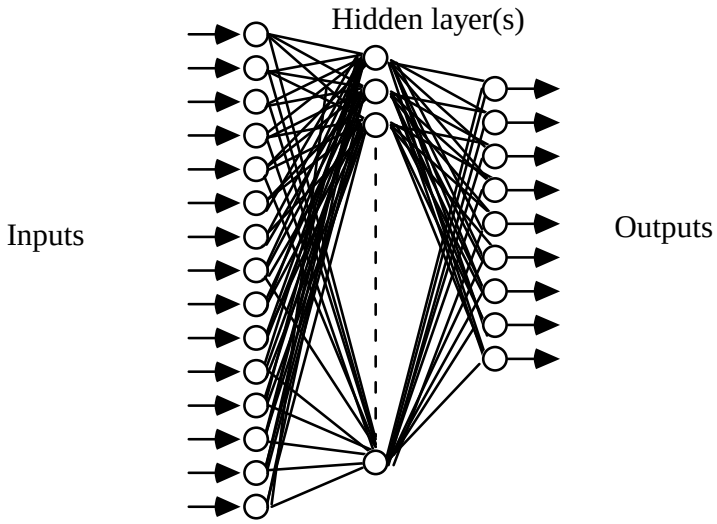

Fig. 2. Neural Network Topology

A set of weights, associated with the links, characterizes the state of the network. Let us denote by w_{ji} the weight of the link between cell j and cell i. As depicted in Figure 3, to compute locally the output S'_i of each cell i, the weighted sum of the outputs S_1, S_2,...., S_n of the n cells of the previous layer, i. e. $\sum_j w_{ji} \times S_j$, is calculated first. The output S'_i is then determined by a nonlinear function, such as the sigmoid function $f(\alpha) = \frac{1}{1+e^{-(\alpha+\theta)}}$, frequently used in practice. The parameter θ is called a threshold. The NN's outputs are the outputs of the cells of the output layer.

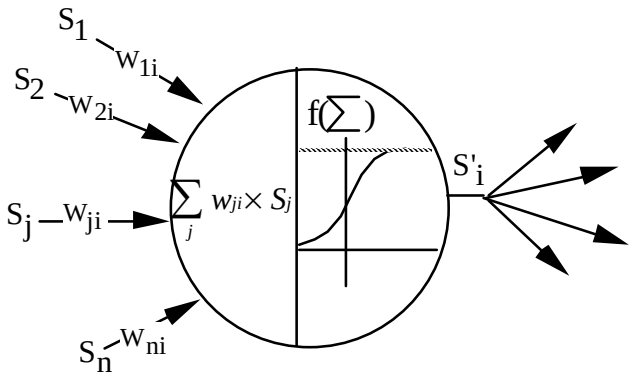

Fig. 3. The Cell Level

A NN learns by updating its weights according to a learning rule used to train it. Back-propagation networks are based on the generalized delta rule. This algorithm provides a method of updating the weights so as to minimize the errors, i. e. the differences between the desired output and the output values computed by the NN. Weights are updated during the learning stage, according to examples which are presented to the network. These examples are pairs constituted of an input vector of p values and an output vector of n values, as follows:

$$((\text{input}_1, \text{input}_2, \ldots, \text{input}_p), (\text{output}_1, \text{output}_2, \ldots, \text{output}_n)) \quad (1).$$

For each example, the NN computes from the p inputs (which are presented as input values to the input layer) the n predicted outputs (i. e., the values taken by the cells of the output layer). The differences between the predicted outputs (computed by the NN) and the desired outputs (given by the example) is computed and "back propagated" (the weights are modified according to error gradient computations). Once numerous examples have been given to the NN one or more times, it is expected to be able to find the unknown output for new cases. This description is somewhat simplified (for a more complete description, see for example: [11], [12], [13].

The topology of the network (number of hidden layers and number of cells in each layer), the activation function used to compute the values (e. g. sigmoid and threshold) and other parameters (e. g. the gain), are factors that greatly influence the learning capabilities of the NN [14].

3.2 Metamodeling

If a NN is trained from examples, of the form of (1), which are derived from direct experimentation on the system under study, the NN becomes a model of this system [13], [15]. If the NN learns from examples generated by a simulation model, the NN becomes a simulation metamodel.

The application of learning methodologies (e. g., rule induction and conceptual clustering) to simulation results in order to extract useful information has already been suggested in such papers as: [4], [6], [16], [17], [18], [19], [20]. More specifically, NNs have been proposed to address scheduling problems, or organize and size manufacturing systems [20], [22], [23], [24], [25]. Hurrion [26], has suggested to connect a NN to a simulation model in order to enhance the decision making process.

Our purpose is to focus on the specific use of NNs as simulation metamodels. Indeed, the powerful learning capabilities of NNs render their use very attractive to predict the outputs to a simulation model given its inputs [27], and to predict future events (e. g., bottlenecks in a manufacturing systems). The ability of certain NNs with a sufficient number of hidden cells to approximate non linear functions (i. e. the simulation outputs in our case,) allows them to compete very well with statistical techniques in term of accuracy [13], [28], [29], [30].

To be used as a simulation model, a NN must be designed and trained in a proper manner. Its architecture has to be designed accordingly, so as to establish a mapping between the input layer and the vector of the p input variables that have to be taken into account, and the output layer and the vector of the n output variables that have to be taken into account. Therefore, in order to be able to learn from example pairs as (1), the NN's architecture must be of the following form: p-L_2- ... -L_{f-1}-n, where L_2,....,L_{f-1} denote the number of cells in the hidden layers. In order to be capable of predicting output values from input values, in the same way as the simulation model, the NN must be trained appropriately, as described in the following section.

3.3 Training of the network

To train the network, it is necessary to build a training set constituted of example pairs as in (1). The simulation model allows us to obtain as many examples as required, whithin a reasonable computing time limit. If an input vector, say x is given as an input to the simulation model, it provides an output vector, say y as a result, or several y^i if several replications are carried out. One (input, output) pair as in (1) can be derived from (x, y) (several pairs in the case of several replications). Generally (x, y) has to be modified to constitute an example, especially in the following cases.

(1) In order to reduce the number of input or output variables. For example, all the processing times of the t jobs on a given machine $\theta_1, \theta_2, ..., \theta_t$, may be replaced by the mean processing time of these jobs $\overline{\theta}$ on this machine. Thus, $\overline{\theta}$, which represents $\theta_1, \theta_2, ..., \theta_t$, will require one input cell only.

(2) The input and output values to the simulation model may need to be converted in another form to be meaningful to the NN. Depending on the type of data (e. g. continuous and binary) and on the NN algorithm used (e. g., based on a sigmoid function, which only yields values between 0 and 1), an appropriate coding may be necessary. Frequently certain values need to be "normalized", in [0, 1] or [-1, 1] by a transformation function. This aspect will be further illustrated in our example. Useful insight on the proper manner to code the data for training a NN can be found in [31].

The training set, i. e., the set of examples, can be built using two possible techniques, namely sampling and experimental design. The sampling technique consists of randomly generating N input vectors x of the simulation model and simulating these N instances, in order to compute the N corresponding y output vectors (NxR output vectors are obtained if R replications are performed). After being properly prepared, as indicated above, these experiments provide a training set of N examples (or NxR examples if R replications are performed).

The second alternative to build the training set is to use experimental design techniques. Rather than a random selection, the x values are chosen according to a factorial design, which aims at reducing the number of simulation runs to perform (for more details see for example [32]).

Other examples, not included in the training set, are necessary to evaluate and to validate the network performance. These examples will constitute the generalization base (or test set). Once trained using the training set, these new cases, which are randomly generated, will be presented to the NN to evaluate its generalization capabilities, i. e., its accuracy.

In order to illustrate the development of a NN metamodel, the next section presents a job shop example.

4. A JOB SHOP EXAMPLE

4.1 The system

The job shop is composed of four machines : a lathe, a drill, a mill, and a wash station. The parts are transported using three automated guided vehicles (AGVs). Three types of parts are processed in the job shop. The part routings and processing times are given in Table 1. The transportation times between the machines and the entrance and exit stations are given in table 2.

	LATHE	MILL	DRILL	WASH
job 1	op1: 7	op3: 5	op2: 4	op4: 9
job 2	op1: 6	op2: 6	op3: 6	-
job 3	-	op3: 8	op1: 9	op2: 12

Table 1. Part routings

to	Drill	Mill	Wash	Exit	Entrance

from					
Lathe	0.6	1	1.5	1.7	1
Drill	-	0.5	1	0.9	1
Mill	-	-	0.5	0.7	1.5
Wash	-	-	-	0.9	2
Exit	-	-	-	-	1.9

Table 2. Transportation times

The jobs arrive independently in the system at constant rates: λ_1 [14, 85], λ_2 [14, 85], λ_3 [14, 85]. Parts await for the availability of a machine in queues, according to a waiting discipline Φ, which can be either FIFO (first in first out) or SPT (shortest processing time served first). The production is performed in two shifts.

4.2 Data collection and preparation

A simulation model of this system has been developed, and is used to simulate the job shop during a given period Δt, of one week (5 days), plus one day of transient phase, not taken into account in the statistics. We are interested in the occupation rates of the machines, μ_1, μ_2, μ_3, μ_4, in order to detect bottlenecks, as well as under-utilized machines. We can consider that this system has four inputs: λ_1, λ_2, λ_3, Φ, and four outputs: μ_1, μ_2, μ_3, μ_4. Therefore, the architecture of the corresponding NN needs an input layer and an output layer, both composed of four cells.

The simulation model was utilized to perform 1100 runs. The inputs to this model, λ_1, λ_2, λ_3 were uniformly randomly selected in [14, 85], and Φ in {0,1} (FIFO=0 and SPT=1). At the end of each simulation run, the values λ_1, λ_2, λ_3 were transformed to λ_1', λ_2', λ_3', where each λ_i' belongs to the interval [0,1]. Then, λ_1', λ_2', λ_3',Φ, μ_1, μ_2, μ_3, and μ_4, were recorded in a file. This file was split in two files. The first one, containing 700 examples, was used as training

set, while the second one containing 400 examples, was used as generalization base, to test the accuracy of the network.

4.3 Training the neural network

Our computations were carried out with a retro-propagation NN software developed by Paugam-Moisy [33]. The training set was utilized to train several NNs, characterized by different architectures (i. e. more or less hidden layers and more or less cells in these hidden layers), and by different learning parameters (learning rate, sigmoid parameter, and saturation point), empirically selected. No bias was introduced. The performance of these NNs was evaluated according to the number of examples they could "recognize". An example is said to be *recognized* if the maximum error of the 4 output cells i=1, ..,4 is less than a given precision (or tolerance) α, that is $\underset{i}{Max}\ |s_i' - \mu_i| < \alpha$.

The best results were achieved by a NN composed of three hidden layers, with the following architecture: 4-6-7-6-4. This network recognizes 100% of the examples of the training set and 100% of the generalization base, with a precision α = 0.1, and 97% of the examples of the training set and 97% of the generalization base, with a precision α = 0.05. These results are quite successful with regards to the "unstable" nature of job-shops. This precision was achieved after 682 learning cycles, i. e., 682 presentations of the training set to the NN for adjustment of its weights. This number of cycles can be reduced if one accepts a lower precision. The various trials we have conducted to design and train the metamodel indicate that, for this case, the number of examples in the training set could have been reduced (the results are almost the same with 400 examples in the training set). It should also be possible to improve this result by using more sophisticated learning algorithms (no bias was used, nor techniques to dynamically adjust the learning parameters).

This example has shown that it is possible to develop a NN metamodel with a small number of hidden cells (19 hidden cells), which models the simulation model, with an acceptable precision to be useful in practice. The achieved

precision is sufficient to allow the NN the capability to predict bottlenecks and under-utilization of machines without running simulation experiments. Such information could greatly contribute to the elaboration of dynamic scheduling strategies of the job shop, which could be incorporated in a knowledge based system [6], [25]. The next section discusses the advantages and inconveniences induced by this metamodeling technique.

5. DISCUSSION

5.1 Advantages

The utilization of NN metamodels can be justified by several advantages, which are presented below.

(1) NN are known to be capable to handle different kinds of non linearity, according to the activation function that they use. This advantage is important since the relationships between simulation variables can often be complex. Moreover, both qualitative and quantitative variables can be used to describe inputs or outputs and no particular assumptions, such as normality, are made about the data. Let also note that a NN metamodel can have multiple outputs, which is not the case of models built using regression or discriminant analysis. Due to these advantages, the NN approach may appear as an interesting alternative to traditional statistical techniques. In its comparison to other statistical models, Wildberger [13] writes: "neural networks can be designed that are at least as accurate as projection poursuit, least absolute deviation, cross-validated local smoothing, etc., when the data relationship actually requires these elaborations. If the interactions among independent variables are significant (i. e., if the secondary structure of the mapping is important compared to its primary structure), then multi-layer, non-linear NNs have a decided advantage over conventional statistical techniques". Meanwhile, he adds that "if the input/output data relationship is only linear, then the use

of any more complex model (than a linear regression), including a back-propagation NN, will only increase the errors".

(2) A NN metamodel generally needs less time than the corresponding simulation model to provide the results. In a simulation model, the time necessary to obtain the results typically depends upon such criteria as the run length, the number of events which occur during the simulation (in discrete event simulation), and the number of replications. In a back-propagation NN, the computing time which is needed is constant; it only depends on its architecture (number of layers and number of cells in each layer). The NN does not depict the dynamic behavior of the system, but only transforms the inputs into outputs using simple arithmetic operations, so that the resulting computing time is generally much lower. In our previous job shop example, it takes on an average 1.73 minute to simulate five days of production, on an 386 IBM PC (the time necessary for SIMAN to read the model is not taken into account). The 4-6-7-6-4 NN takes about 0.8 second (the time necessary to read the weights and parameters is not taken into account). Only one replication has been carried out in our example since the model was deterministic. If random variables have had to be used, several replications would have been necessary, increasing accordingly the computing time. This time saving advantage is particularly attractive in the design of such software as real time decision support systems, control systems, and monitoring systems [27]. Frequently, in this type of applications, several decision strategies have to be evaluated within a very short time interval; as a consequence, the results must be obtained very quickly. For example, the multi-pass scheduling approach [34], [35] necessitates, each Δt (a time interval that can be constant or variable), N simulations of the flexible manufacturing system to compare the N scheduling rules that may be applied. The use of a metamodel instead of the simulation model would save computing time and hence, allow more scheduling rules to be tested (e. g. combinations of different scheduling rules on each machine), or Δt to be reduced. Let us note that for particular areas of application, hardware implementation of the NN could also be contempleted.

(3) The NN program necessary to run the metamodel is simple and does not require much memory. The “run-time” component of SIMAN IV [36] used to simulate our example requires 433 083 bytes on a compatible IBM PC (386), while the "run time" component of the NN program uses only 9022 bytes on the same machine. To give another comparison, GPSSH on the same machine needs 642066 bytes. This advantage is interesting in the design of decision support systems which incorporate a simulation componant [17], [25], [27], [35]; for example, a SLAM II simulation model was incorporated in a real time scheduling algorithm in [34]. Replacing the commercial available simulation software by a NN metamodel would allow memory to be saved.

(4) When the simulation model is used as a real time decision support to an existing system, such as to aid control decision making, the accuracy of the simulation model is important. Unfortunately, some characteristics of the system can change over time. For example, an improvement of the quality in a manufacturing system may lead to the reduction of rejected parts. In such a case, the simulation model should be modified in order to remain accurate. Thanks to its learning possibilities, a NN is capable of updating its weights, according to new examples coming directly from the real system, so that it can integrate certain changes in this system [27]. If more inputs or outputs had to be taken into account, the NN architecture would also have to be modified.

5.2 Shortcomings

Although NNs are very attractive tools for metamodeling purposes, their use presents several inconveniences, which are as follows.

(1) The development of a NN metamodel may be time consuming. Depending on the cases (number of inputs, complexity of the system behavior) and on the precision needed, numerous simulations may be necessary to build the training set. Moreover, the selection of the NN parameters and the learning phase are also likely to be time consuming.

(2) Precision is lost comparing to the simulation model results (meanwhile, in the job-shop example, the achieved precision is greatly sufficient to predict bottlenecks). The precision that can be obtained by a NN depends on such criteria as the "regularity" of the behavior of the system under study, and the variance of the outputs.

(3) This approach requires to be able to define the factors that have an effect on the system response, in terms of a reasonable number of qualitative or quantitative variables. Such a problem can arise for example, when one want to describe the production schedule of a manufacturing system with a limited number of input cells. Too many inputs would necessitate to perform too many simulation runs to obtain enough examples to train the network.

(4) Simulation metamodels are frequently used in order to gain insight into the relationships between inputs and outputs of the system under study. In a NN metamodel, the knowledge about how the inputs are transformed into outputs is represented through the weights. Hence, the analysis of these weights may provide useful insight about the system behavior [37]. However, the information contained in the NN's weights is not as explicit as the linear equations of regression metamodels [9], [38], or as the production rules of rule-based metamodels [6].

The three first inconveniences are not completly specific to NN metamodels. Whatever the metamodeling technique used, the development of the metamodel requires time, precision is lost and some factors may be difficult to represent as metamodel inputs. The difficulty to exploit the information about the system behavior contained in the weights represents a more specific limitation of the use of back-propagation NN metamodels.

In this discussion, emphasis was put on retro-propagation NNs, since our purpose was to illustrate the use of NNs as metamodels, especially to reduce the computing costs induced by the use of simulation models. Meanwhile,

other NN paradigms, such as the generalized regression neural network [39], could also be used for the same purpose.

CONCLUSION

In this paper, we have discussed a metamodeling approach based on neural networks. In this technique, the neural network learns from a training set composed of simulation results. The resulting network is capable of predicting the output values of the simulation model, corresponding to given input values, without having to depict the dynamic behavior of the system over time. For such purposes as gaining understanding about the system or reducing the number of inputs, regression or rule-based metamodels seem to be currently more suitable. One of the main advantages of neural networks metamodels is their capability of dealing with nonlinearities and complex interactions between variables, if a sufficient number of hidden cells is used. Moreover, neural networks can handle both quantitative and qualitative variables as inputs or outputs. Therefore, they represent a interesting alternative when traditional statistical methods failed or are not suited.

Compared to simulation models, they reduce the computing costs and are capable of learning from new data. Therefore, neural networks metamodels can be used in lieu of simulation in decision making processes where the results must be obtained very quickly. In such cases, it may be justified to spend time “off line”, before the model is involved in the decision making process (“on line”). Moreover, a neural network can be more easily embedded in another decision support software (e. g., a real time scheduling system, monitoring and control software), since it generally requires a smaller amount of memory than the usual simulation packages.

REFERENCES

[1] R. W. Blanning, "The construction and implementation of metamodels", *Simulation*, Vol. 24, pp. 177-184, 1975.

[2] J. P. C. Kleijnen, "Regression metamodels for generalyzing simulation results", *IEEE Transactions on Systems Man and Cybernetics*, Vol. SMC 9, No. 2, pp. 93-96, 1979.

[3] W. S. Meisel and D. C. Collins, "Repro-modeling: an approach to efficient model utilization and interpretation", *IEEE Transactions on man systems and cybernetics*, Vol. SMC-3, No. 4, pp. 349-358, 1973.

[4] C. Héritier, *Une aide à la conception des système de production basée sur la simulation et l'analyse de données*, Ph.D. Thesis, Ecole Nationale des Mines, St Etienne, France, 1991.

[5] L. Lin and J. K. Cochran, "Metamodels of production line transient behavior for sudden machine breakdowns", *International Journal of Production Research*, Vol. 28, No. 10, pp. 1791-1806, 1990.

[6] H. Pierreval, “Rule-based simulation metamodels”, *European Journal of Operational Research*, Vol. 61, pp. 6-17, 1992.

[7] C. N. Madu, C. H. Kuei, "Simulation metmodels of system availability and optimum spare and repair units", *IIE transactions*, Vol. 24, No. 5, pp. 99-104, 1992.

[8] B. Bettonvil and J. P. C. Kleijnen, "Identifying the important factors in simulation models with many factors", *Research memorandum*, Department of economics, Tilburg University, The Netherlands, 1991.

[9] J. P. C. Kleijnen and C. R. Standridge, "Experimental design and regression analysis in simulation: a FMS case study", *European Journal of Operational Research*, Vol. 33, pp. 257-261, 1988.

[10] C. M. Harmonosky, S. F. Robohn, "Real time scheduling in computer integrated manufacturing: a review of recent research", *Int. J. Computer Integrated Manufacturing*, Vol. 4, No. 6, pp. 331-340, 1991.

[11] R. P. Lippmann, "An introduction to computing with neural nets", *IEEE ASSP Magazine*, pp. 4-22, April, 1987.

[12] P. D. Wasserman, *Neural Computing: theory and practice*, Van Nostrand Reinhold, New York, 1989.

[13] M. A. Wildberger, “Neural network as a modeling tool”, in *AI and simulation: theory and applications*, W. Webster and R. Uttamsingh, Eds., S. C. S. I., San Diego, CA, pp. 65-74, 1990.

[14] H. Paugam-Moisy, "On the convergence of a block gradient algorithm for back propagation learning", *proc. of the international joint conference on neural networks*, Baltimore, Maryland, Vol. III, p. 919-924, June, 1992.

[15] P. A. Fishwick, "Neural network models in simulation: a comparison with traditional modeling approaches", *Proc. of the Winter Simulation Conference*, Washington DC, p. 702-710, 1989.

[16] C. Chu and M. C. Portmann, "Application of artificial memory in scheduling problems", *Journal of Intelligent Manufacturing*, Vol. 4, pp. 151-161, 1993.

[17] R. M. O'Keefe, "Simulation and Expert Systems: a taxonomy and some examples", *Simulation,* Vol. 46, pp. 10-16, 1986.

[18] S. C. Park, N. Raman and M. J. Shaw, "Heuristic learning for pattern directed scheduling in a flexible manufacturing system", *proc. of the Third ORSA/TIMS Conference on FMS: operations Research models and applications*, Elsever Science Publishers, The Netherland, p. 369-376, 1989.

[19] H. Pierreval, "Data analysis oriented techniques for learning about manufacturing control with simulation", *Proc. of the 2nd European Simulation Multiconference: factory of the future*, Nice, France, pp. 61-66, June 1988.

[20] H. Pierreval and H. Ralambondrainy, "A simulation and learning technique for generating knowledge about manufacturing systems behavior", *Journal of the Operational Research Society*, Vol. 41, No. 6, pp. 461-474, 1990.

[21] E. Chappaz, "A neural network for optimization of manufacturing lines", *Proc. 10th MICAD conf. on CAD/CAM computer graphics and computer aided technologies*, Paris, France, p. 624-634, February, 1991.

[22] G. Chryssolouris, M. Lee, M. Domroese, "The use of neural networks in determining operational policies for manufacturing systems", *Journal of manufacturing systems*, Vol. 10, No. 2, pp. 166-175, 1991.

[23] P. Guillard, P. Baptiste and J. Favrel, "Modelling and simulation for self-organization in modern production workshops", *Proc. of the CAPE'91 Int. conf. on Computer Application in Production Engineering*, Bordeaux, France, p. 705-712, September, 1991.

[24] H. Pierreval, "Neural Network to Select Dynamic Scheduling Heuristics", To appear in the *Journal of Decision systems / Revue des systèmes de décision*, Vol. 2, No. 2, pp. 173-190, 1993.

[25] L. C. Rabelo and S. Alptekin, "Synergy of neural networks and expert systems for FMS scheduling", *Proc. third ORSA/TIMS Conf. on Flexi. Manuf. Sys.,* Elsever Science Pub., p. 361-366, 1989.

[26] R. D. Hurrion, "Using a neural network to enhance the decision making quality of a visual interactive simulation model", *J. Opl. Res. Soc.*, Vol. 43, No. 4, pp. 333-341, 1992.

[27] M. A. Wildberger and K. A. Hickok, “Power plant modeling and simulation using artificial intelligence and neural networks”, in *Progress in Simulation*, Ablex Pub., Vol. 1, pp. 100-125, 1992.

[28] P. Gallinari, S.Thiria, and F. Badran, "On the relation between discriminant analysis and multilayer perceptron, *Neural networks*, Vol. 4, No. 3, pp. 349-360, 1991.

[29] C. Muller and M. Radoui, "Réseaux neuromimétiques et analyse de données", *proc. XXIIième journées de statistique*, Tours, France, 1990 and *research report* of the Direction des Etudes et Recherches of Electricité De France, Clamart, France.

[30] H. White, “Neural networks; learning and statistics”, *AI expert*, pp. 48-52, December, 1989.

[31] J. Lawrence, “Data preparation for neural networks”, *AI expert*, pp. 34-41, November, 1991.

[32] J. P. C. Kleijnen, *Statistical tools for simulation practitioners*, Marcel Dekker Inc., New York, U. S. A, 1987.

[33] H. Paugam-Moisy, *Optimisation des réseaux de neurones artificiels, analyse et mise en oeuvre sur un ordinateur massivement parallèle*, Ph. D. Thesis, Ecole Normale Supérieure de Lyon et Université Lyon1, January, 1992.

[34] N. Ishii and J. J. Talavage, “A transient-based real-time scheduling algorithm in FMS”, *International Journal of Production Research*, Vol. 29, No. 12, pp. 2501-2520, 1991.

[35] S. Y. Wu and R. A. Wisk, “An application of discrete event simulation to on-line control and scheduling in flexible manufacturing”, *International journal of production research*, Vol. 27, pp. 1603-1623, 1989.

[36] C. D. Pegden, R. E. Shannon and R. P. Sadowski, *Introduction to simulation using SIMAN,* McGraw-Hill Inc., New Jersey and System Modeling Corp., USA, 1990.

[37] M. Caudill, “Using neural networks: Representing knowledge”, *AI Expert*, part 1 and 2, December, 1989.

[38] C. N. Madu, "Simulation in manufacturing: a regression metamodel approach", *Computer ind. Engng*, Vol. 18, No. 3, pp. 381-389, 1990.

[39] M. Caudill, "GRNN and bear it", *AI Expert*, pp. 28-33, May, 1993.

APPENDIX: SOME FEATURES OF THE NEURAL NETWORK USED IN THE EXAMPLE [33]

The weights are randomly initialised in [-0.5, 0.5]. They are updated during the learning process according to the following function, which is a sigmoid with two parameters: λ and γ (saturation).

$$\begin{cases} f(x)=\dfrac{e^{\lambda x}-1}{e^{\lambda x}+1} \quad \text{si} \ \dfrac{e^{\lambda x}-1}{e^{\lambda x}+1}\in[-\gamma,\gamma] \\ f(x)=1 \ \ \text{si} \ \dfrac{e^{\lambda x}-1}{e^{\lambda x}+1}>\gamma \\ f(x)=-1 \ \text{si} \ \dfrac{e^{\lambda x}-1}{e^{\lambda x}+1}<-\gamma \end{cases}$$

The learning rate η is adaptative, that is:

$\eta = \frac{k\alpha}{\text{number of cells of the previous layer}}$, where k_α is a constant.