

Monitoring and Real-Time Simulation of Power Plants: a Neural-based Environment

C. Alippi*, A. Cori⁺, V. Piuri*, F. Pretolani⁺

*Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
Phone +39-2-2399-3623
Fax +39-2-2399-3411
Email {alippi, piuri}@elet.polimi.it

⁺Centro Ricerche di Automatica
ENEL
Cologno Monzese, Italy
Phone +39-2-72241
Fax +39-2-72245525
Email {cori,pretolani}@cra.enel.it

Abstract - An efficient simulation environment on low-cost computing systems is mandatory to implement advanced control systems, to train the personnel on complex plants, to provide a real-time decision support, and to create a computer-based intelligent automatic system. This allows to improve the competitiveness and the quality of the plant itself and of the operating personnel. In this paper, the use of an advanced environment is presented for the case of power plants, which integrates both conventional equation-based and innovative neural-based models in a single dynamic simulator.

I. INTRODUCTION

The problem of monitoring an industrial process requires development of suitable simulators to infer and foresee measurements of critical entities, to design optimised controllers, to validate the chosen strategies before any effective action is applied to the plant and to train the operating personnel [1, 2]. These features are particularly relevant in complex plants, e.g., in thermoelectric power plants, where relationships between components and dynamics are difficult to be taken into account and managed by exploiting limited operator sensitivity and experience only.

Conventional tools of this kind simulate the evolution of the process variables by solving computationally expensive systems of thousands of generally non-linear coupled equations (both algebraic and differential) which represent the physical description of the plant. Besides, to solve the differential equations, conventional simulators require the complete knowledge of the initial states of many process variables which should be either predicted or measured in the plant; the former solution requires ad hoc state reconstructors, the latter may be unfeasible.

The computational complexity is not a drawback if the goal is to define and tune the final controllers since the task is accomplished once and off-line. On the contrary, its minimization becomes mandatory in implementing advanced features for real-time personnel training, real-time decision supports during operation of the plant, or even automatic decisions autonomously taken by the computer-based control both on previous experiences and simulations.

In particular, monitoring and forecasting the evolution of critical variables of the plant (e.g. temperatures and pressures) may pose strict real-time requirements. Whenever real-time decision support and on-line predictive control are taken into account, it is desirable to forecast the future

behavior of a set of variables in a neglectable time with respect to their time constants; this allows the operator or the automatic control system to take actions for safe operations.

In all these cases, approximation of the system model can be performed by simplifying the equations describing the system. Even if it may be mathematically feasible, such an approach leads to loose accuracy in a relevant way and, as a consequence, cannot be usually accepted when this characteristic is mandatory for the simulation application.

To provide the necessary computational capacity, high performance workstations are often used to host the simulator both on the plant's control console and in the personnel training sites. However, the cost of these computing systems becomes often very high or even unaffordable to guarantee the required speed-up. As a consequence, the competitiveness of control systems with advanced features and the possibility of personnel training are quite limited. To increase the economic competitiveness and to spread the use of such control systems, it is necessary to reduce the computation complexity of the simulators in order to scale down the computing power and the cost of the workstations.

Neural-based identification of subparts of the system is a natural candidate to solve all the above issues, with an acceptable loss in accuracy [3]. When properly configured, the neural representation is able to capture the whole behavior of the plant's subpart in a compact description. In particular, the subpart model gives the input-output dynamic behavior by associating the output to the actual inputs and to the previous system history as a whole; conversely, the conventional description of the subpart is based on physical relationships and, as a consequence, has to deal with several interrelated differential non-linear equations defining the interactions among the atomic components generated by partitioning in the finite-elements' model. We experimented that the computational complexity associated with neural identifiers is generally considerably lower than the one obtainable by considering the conventional approaches. As a consequence, real-time requirements may be satisfied at a lower cost.

The system model may therefore contain both neural and classical blocks, according to the most efficient representation for each component. Configuration of the system model is performed in a different way for these two kind of representations. Design and tuning of a neural model for identification and control in a heterogeneous simulator is not a simple task since it involves topology and interconnection specifications, data sampling and synchronisation, scheduling of different tasks, input generation and output analysis.

This paper is directed to discuss the problem of integrating these heterogeneous components in a single simulation environment in order to provide the methodological and software frameworks to build dynamic systems' simulators for plant management and control and for personnel training. Our integrated simulation environment allows design, training, testing, validation both of the neural component and the overall model of the dynamic non-linear system. This environment has been specifically designed by exploiting the experiences both of conventional simulators for identification and control (in particular, the ones for power plants) and general-purpose neural simulators. In the paper, we describe the overall architecture of the simulation environment and the detailed functionalities.

II. THE SIMULATION ENVIRONMENT

Real-time equation-based dynamic simulators are widely used in designing and implementing automatic control systems for power plants; in particular, they are relevant to design the control functions, to verify the operating procedure, to study the transient phenomena, to certify and check the regulating subsystems, and to train the personnel. As already said, behavior forecast in response to events and operations performed on the plant are an attractive feature to provide the necessary support to on-line operating decisions whenever the complexity of the system becomes large or when the personnel have not cumulated enough experience to be confident in difficult or rare operations.

An example of this class of simulators is LEGO [4], a tool developed by ENEL-CRA. In this environment, it is possible to model complex industrial processes with their regulations by means of their mathematical representation consisting, generally, of non-linear differential equations at the partial derivative (in specific cases, they can be reduced to linear and/or algebraic and/or total derivative equations).

With respect to traditional simulators consisting of ad hoc tools modelling the plant, LEGO is a highly-flexible system due to its modular structure. The simulation system is composed of a simulation engine (the *scheduler*) and by the process description (the *blocks* and the *tasks*), as shown in Fig. 1. User's interaction is managed by the *user interface*, which provides commands to configure the system, to set-up, control and run the simulation, to observe dynamically and graphically the system behavior and the simulation statistics.

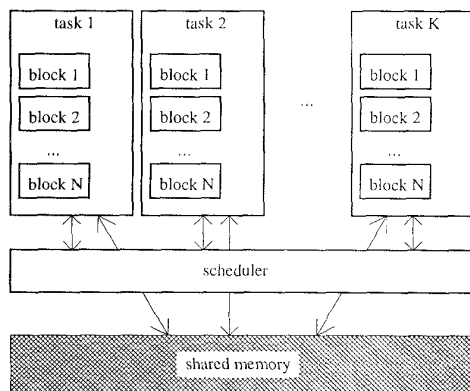


Figure 1 - LEGO: the structure of the simulation environment.

The modular and hierarchical definition of the plant's behavior is given by the set of *tasks* and by the *blocks* contained in the tasks themselves. Each task is associated with an individual subsystem of the whole plant (e.g., the furnace/boiler, the condenser, the turbine/alternator), while each block within a task represents the behavior of a small low-level component of the plant (e.g., the valves, the pumps, the superheaters). A block consists of a software procedure computing the dynamic non-linear equations modelling the corresponding component. The task is the software procedure that merges the descriptions of the subsystem's blocks and defines the connections between the input and output variables of such blocks. Libraries of blocks and tasks have been developed to support the description of a wide spectrum of power plants; elements in these libraries are parametric for easy adaptation to real cases.

From the software point of view, each task in LEGO is a concurrent process running on a multitasking operating system. To achieve correct and consistent results, it is necessary to coordinate and synchronize the computation of all the tasks modelling a plant. The *scheduler* is the simulation engine which is responsible of guaranteeing both communication and synchronization among tasks. Practically, it manages the transfer of values from the output variables of a task to the input variables of the connected tasks. This is performed by synchronizing the mutually exclusive access to the shared data memory, which contains the variables associated to the input/output signals and to the internal status of each plant's component.

During the system configuration, the equations contained in each block and task are collected in a single system of algebraic and differential equations. The simulation engine operates on this global equations' system by solving a Cauchy's problem at each simulation step. The duration of each simulation step is related to the speed of the transient events occurred in the previous steps.

Within each simulation step, the simulation engine identifies the Cauchy's solution (i.e., the inputs, the outputs, and the final status of each component) by applying traditional iterative techniques (e.g., the Newton-Rapson method) on the discretized equations. The number of iterations required to identify the solution is related to the desired precision and it may increase in the presence of fast or wide-amplitude transient events.

The main problem is the number of the resulting equations for the simulation engine. Typically, each subsystem of a power plant is described by several hundreds of equations and, as a consequence, the whole plants is modelled by thousands of equations. Therefore, even if the resulting equations' system is represented by a block matrix, the computational complexity of the simulation becomes very high so that high-performance computing systems are required to deal with accurate models. Obviously, if real time is a relevant requirement, the performance must become high in order to generate results for comprehensive training or for on-line decision support. On the other hand, massively-computing systems are expensive, thus reducing the feasibility of use in training classes or in the plant's control console. Presently, reasonable-priced workstations for the mentioned applications are capable of speeding up the simulated behavior at most twice the actual plant's behavior, while it is required tens or hundreds times.

III. THE NEURAL COMPONENTS

Neural networks are nowadays a modelling technique that has been shown effective to describe the behavior of dynamic non-linear systems. In several experiments, we observed that they are useful also in classes of applications related to power plants and transmission. To enhance the simulation capabilities of the LEGO environment, we implemented suited tools to support neural-based modelling for non-linear static and dynamic systems.

The regressive neural networks have been proved in the literature to be universal approximators of linear static functions and, under some constraints, have shown very good approximation capability also for non-linear static and dynamic functions.

A neural network for modelling static systems is a *regression-type network*. This computational paradigm is composed by neurons connected in a multilayered feed-forward topology. Each neuron computes the weighted summation of its inputs and generates its output by applying a non-linear transfer function (usually, sigmoid or step function for the hidden neurons, linear for the output ones) to the weighted summation. It was shown in the literature that a single hidden layer is enough to approximate any non-linear function.

Some other classes of neural networks have been explored to represent the behavior of dynamic systems. We performed extensive experiments to verify the effectiveness of such models in real complex applications in the area of power plants. We considered at first models containing feed-back loops with memory elements *locally* within each neuron (e.g., the network presented in [5]). These networks are quite difficult to be configured since they have too many degrees of freedom associated with the number of network's parameters and neurons. This leads to a complex training procedure and convergence may become a non trivial issue. Such drawbacks should be not a problem if training is performed only once and on-line parameters' adaptation is not taken into account.

Interesting results have been achieved by adopting neural networks based on *global feed-back loops* (e.g., see [6]), instead of the local ones. In this classes of networks, the general structure of the network is identical to the classic multi-layered feed-forward topology; memory elements are associated to the primary inputs and to the network's outputs in order to provide information about the behavior history that is fed back to the network's inputs. Since typical application processes and systems that we are modelling have bounded inputs and outputs, in this paper, we restrict our attention to this case. Neural experiments on Single-Input Single-Output systems had proved to give satisfactory results; our research, due to the specific characteristics of the modelled systems, focused on extending the approach to deal with Multiple-Input Single-Output networks. At the moment, experiments are in progress also for multiple inputs and outputs systems to overcome difficulties in directing the network towards multiple goals (the expected output behaviors) contemporaneously.

When prediction is concerned [3], we use the network topology shown in Fig. 2a for the SISO case (generalization to MISO is straightforward): network's inputs are the primary inputs of the real dynamic system and as many previous system's outputs as required by the kind of dynamics. The

network's output is the system's forecast output (forecasting is performed by k time unit in advance according with the network configuration). The best results were obtained by using the sigmoid or the hyperbolic tangent as transfer function of hidden neurons, while the output neuron is still linear; one hidden layer is sufficient.

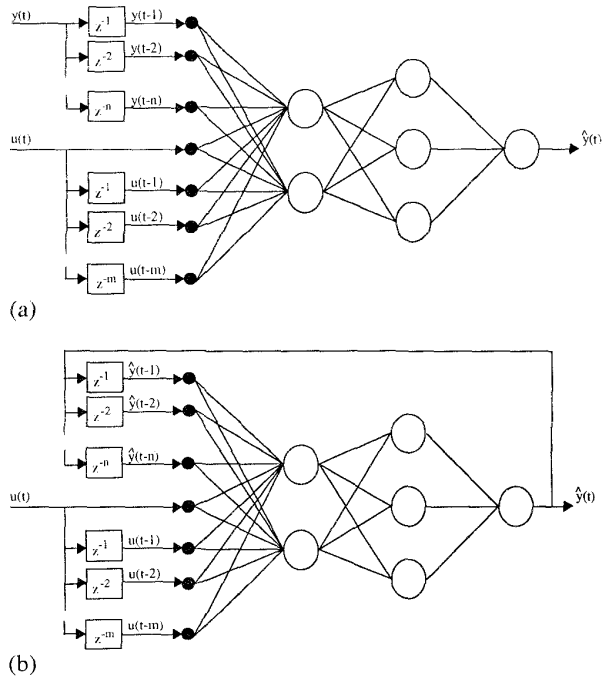


Figure 2 - Neural networks for dynamic system modelling: the prediction network (a), the identification network (b).

The network topology for a neural identifier can be easily derived from the prediction topology [3] when the training procedure leads to an absolute error smaller than a predefined maximum value. In this case, the expected network's output is quite identical to the actual output of the modelled system at every time. As a consequence, the actual system output could be replaced by the expected output at any time. During training the network's topology is the one shown in Fig. 2a; after training (i.e., when the interconnection weights have been frozen), the network feed-back topology is shown in Fig. 2b. No actual system output is required to maintain the network behavior near to the real system behavior, since the network has captured completely the system behavior and it is able to replicate it exactly and autonomously.

This approach to generate identifiers is effective for SISO, while possible instabilities may be introduced in MISO systems due to feeding the network's estimated output back to the input instead of the actual system's output.

This problem may be usually solved by training the network topology of Fig. 2b directly, instead of configuring first a predictor and then rearrange the feed-back loops to achieve the identifier [3]. Stability, whenever possible, is learnt during training. Acceptable results are often achieved, but higher difficulties may occur during training since there is no immediate guidance for the network's output and prevention of output drifts due to the cumulated output errors.

When a family of models (i.e., the network topology) has been selected for the specific application, possibly according to the experience of the model designer or to experiments, one specific model must be created, i.e., the number of layers, the numbers of neurons in each layer, and the interconnection weights must be estimated with respect to a given figure of merit. In general, no direct relationship is available to guide the choice both of the number of layers and of the number of neurons within each layer. Usually, the designer (from the personal experience or from the literature) identifies a choice large enough to deal with the specific application case and tries to verify the correctness of this assumption by experiments. In the literature, some techniques are available for static non-linear function identification to minimize the network structure either during and at the end of the configuration procedure. Due to the complexity of these operations and to strict relationship with the specific application, no support has been considered within the LEGO extension dealing with neural models.

The evaluation of the interconnection weights is obtained by means of the *learning* iterative procedure. Inputs (belonging to a suited *training* data set) are presented to the network under configuration, outputs are computed and compared to the expected values: the weights are adjusted in order to minimize a suitable error function. In prediction and identification, the optimized figure is generally a mean square error function evaluated on the whole outputs' trajectory or on an outputs' subset. Adjustment is performed by an external supervisor (for this reason, such an approach is called supervised learning). At the end of learning, the configuration is frozen and preserved during the whole *recall*, i.e., the normal operation.

The descendent-gradient algorithm as well as the conjugate-gradient algorithm and the quasi-Newton one have been included in the extension of the LEGO environment to achieve this goal according to specific characteristics of the considered application case. In prediction, the output evaluation depends only on the system inputs and outputs: as a consequence, the optimization procedure coincides with the traditional one adopted for algebraic networks. Conversely, in identification, evaluation of the system's output is based on the output's evaluations at the previous time steps, i.e., minimization and weight adjustments must be performed in a recursive environment.

Weight modification can be performed either on-line at each presentation of the input-output pair, or off-line at the end of each batch of input-output pairs. The second technique allows to decrease the training time without reducing the accuracy of the solution.

The mean square error, evaluated by using a *test* data set at the end of a batch of iterations during learning, gives information both about the quality of learning not only with respect to the training data set and about the feasibility of terminating the learning itself. Besides, this figure, evaluated by using a *validation* data set at the end of learning, is a measure of the intrinsic quality of the whole learning, independently from the actual training data set, with respect also to the generalization ability.

Generating training, test and validation data sets is the most critical task to achieve a good configuration of the neural network and an accurate system neural model. Since the dynamics have to be captured and not only the static (even

non-linear) correspondence between inputs and outputs, the data sets must not actually consist of separate pairs of input-output values. Data sets must be composed of groups of consecutive input samples extracted through a window sliding on the continuous input sequence and presented in parallel to the network in order to provide enough information on the time dependencies. The best approach to excite all dynamics and to create data sets is using white-noise signal for each system input and in observing the corresponding outputs.

Unfortunately, this is not always feasible in real applications due to possible actual limitations on the input values (e.g., for safe operation), to correlations between inputs (e.g., for physical reasons), and to the presence of automatic regulation loops (e.g., for efficiency, safety, and cost reasons). The same constraints may be present also in the traditional simulators of such systems, so that the effects of the ideal sollicitations can neither be simulated.

The feasible approach is therefore often based on the generation of transient events in the traditional LEGO environment without excluding the regulation loops. At first small variations are applied around characteristics operating points to create a set of experiments for local identification. By applying such data, the basic structure of the network (i.e., the topology, the number of layers and the number of neurons) can be initially identified and checked. Then, large variations are applied to the inputs to perform identification in the large in order to refine the model by incorporating the capability of generalizing the system behavior between the operating points considered during identification in the local. Partial uncorrelation of the inputs in regulated systems (i.e., meaningful data sets) can be obtained by using pseudo-random binary signals as added exciting inputs; they consists of small-amplitude steps and ramps which are added to the output of the regulating subsystem.

Normalizations both of inputs and outputs is mandatory to provide equal opportunity to excite the system dynamics to every input and to avoid polarization of weight adjustment with respect to dominant-amplitude outputs.

IV. INTEGRATING CONVENTIONAL AND NEURAL COMPONENTS

The exploitation both of the computational complexity reduction and of the generalization accuracy increase, due to the use of neural models instead of the traditional equation-based description of system components, is strictly related to the effective integration of these innovative modelling techniques in the conventional simulation environment. To achieve this goal, suited modifications and integrations have been performed in LEGO by using its modular structure and, mainly, the implementation based on concurrent processes.

From an abstract point of view, every component and subsystem in the plant could be represented by using a neural model. However, this could not be the best solution as computational and configuration complexities are concerned. In particular, whenever a component or a subsystem is described by very few algebraic equations, it is often more efficient to hold such a description since it is more immediate. Conversely, when differential or complex non-linear equations are concerned, better results may be achieved with neural techniques. Therefore, first of all, it is necessary to identify a suited *partitioning* of the whole plant in order to

specify which subsystems need to be described neurally and which ones are better represented traditionally by equations. Components are usually described by a limited number of equations and, therefore, are not suited to be modelled by neural networks: besides, whenever a subsystem contains one or more components suited for neural representation, it is easier to describe neurally the whole subsystem by collapsing also the simple components in the neural model. Partitioning can be performed by exploiting the designer's experience or the literature, according with the experimented neural modelling capabilities, as discussed in the previous section.

A neural network representing a subsystem is simulated in LEGO by a *neural task* either during recall and learning, i.e., by a special concurrent process running in parallel with the other traditional LEGO tasks. The internal structure of a neural task is composed by blocks as shown in Fig. 3. The neural task extracts the subsystem's inputs from the shared memory of the LEGO simulator and stores the computed outputs, exactly in the same way of the traditional tasks. The *input normalization block* is directed to normalize the nominal input values, as required by the neural operations, while the *output normalization block* restores the normalized network's output to the nominal values expected for the subsystem; the parameters characterizing the normalization processes must be specified at the simulator initialization.

The *noise block* allows the simulator user to introduce additive noise on the subsystem's inputs and reference points; at the moment, only the pseudo-random noise with maximum amplitude specified by the user is supported. This is useful to increase, during learning, the network insensitivity to possible external noise occurring during recall.

The *delay block* implements the delays contained in the neural model to capture the dynamic behavior. It stores the subsystem's inputs and the network's outputs specified by the user to realize the desired neural model with global feed-back loops. If no delay is defined, the dynamic model degenerates into a multilayered feed-forward network for static modelling.

The *neural block* is the core of the neural model: it computes the neural paradigm with the given interconnection topology and weights.

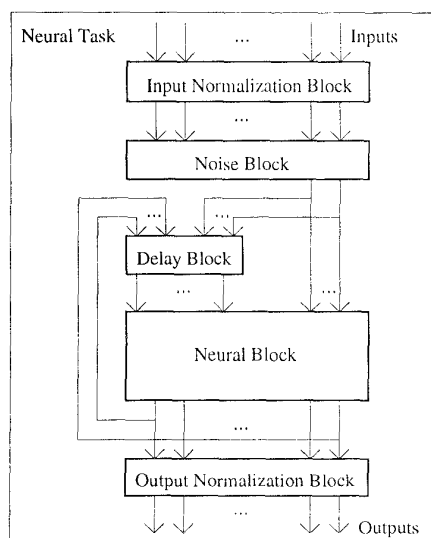


Figure 3 - The internal structure of the neural task.

During recall, the neural task reads the subsystem's inputs from the LEGO's shared memory, when its operation is allowed by the LEGO's scheduler. The cascade of operations are performed without requiring any iterative behavior: the input normalization is followed by the feed-forward computation of all neurons and by storing inputs and outputs in the delay block, while the output normalization completes the sequence. This internal simplification with respect to the traditional equation-based model is relevant to reduce the computational complexity since internal iterations required to solve the possible Cauchy's problems are removed. The neural model achieves the solution in a single iteration, even if it may be more complex than the individual iteration of the traditional method: in general, the result is generated more quickly in the neural approach. The denormalized results are stored in the shared memory for subsequent operations.

During learning, the neural task performs the same operations described above for recall, but noise may be introduced on the subsystem's inputs before running the computation of the neural block. Besides, the network parameters need to be adjusted in order to tailor the neural model on the specific application subsystem, according to the selected learning procedure.

On-line generation of the training data sets as well as the data sets for test and validation can be easily performed by running in parallel the LEGO task representing the traditional equation-based model of the subsystem. For off-line generation, the equation-based model is run in advance and the input-output pairs are stored in a file for the subsequent simulation with the neural model. Data are transferred to the neural task through the LEGO shared memory.

The user is allowed to select the parameters characterizing the learning procedure, namely, the number of samplings contained in each data window on the input and output streams (i.e., the history related to the signal dynamics), the number of windows required to evaluate the individual weight updating, the number of repetition with the same data window, the number of samplings by which the window must be moved after parameter updating.

Due to the structure of the LEGO environment based on concurrent processes, more than one neural network can be treated within the same system. Each network is associated to a separate neural task and to the corresponding neural termination task. Simulation both for learning and recall is performed independently one network from the others. The LEGO scheduler guarantees the correct synchronization and data transfer among tasks.

The neural models can be connected in any way as the traditional equation-based models. In particular, they can be cascaded with data flowing from the first to the second one completely or only partially. A cascade of subsystems containing neural models can also be configured to create cycles by feeding one output of a subsystem back into the input of another subsystem proceeding the first one in the cascade.

V. CONCLUSIONS

In this paper, we presented an integrated simulation environment for dynamic systems, which achieves high performance suited to real-time applications. In particular, our attention was focused on the introduction of neural

models for non-linear dynamic components which replace conventional descriptions based on equations derived from the physical behavior.

The basic goal was to reduce the computational complexity of the models in order to allow for fast simulations. In turn, this allows to speed up the simulation with respect to the time constants typical of the considered dynamic system and to provide a framework for real-time foreseeing, decision support and comprehensive training of complex systems.

When strict real time is not highly mandatory or the system complexity is not too high, the reduced computational requirements of the system model may be exploited to reduce the computational power of the workstation hosting the simulator (e.g., by adopting high-performance IBM-compatible personal computers) and, as a consequence, the cost of the simulation system. For some lucky applications, both of these improvements are feasible: i.e., computational complexity is so decreased that real-time simulation is possible on low-cost computers.

Availability of efficient and flexible simulators constitutes the starting point to study, design and implement advanced control systems capable of adaptation both to system aging and application environment modifications. Besides, benefits can be achieved also by research and experiments on control systems with autonomous decision capacity based both on the stored knowledge derived from the real system activity and on on-line simulation of selected control operations to identify the best strategy among them.

Use of efficient simulation environments for complex systems related to real cases is particularly interesting not only for industrial applications, but also for academic purposes. For example, training on real complex systems is useful to guarantee a better understanding of the basic concepts and of systems' dynamics both in high school courses, in the university and in specialization courses. On-line interactive simulation provides an effective environment to verify and improve the knowledge by direct experimentation in a reasonable time.

Even if attractive results have been achieved in our experiments, some minor problems still remain in using neural models for dynamic parts of complex systems. In very few experiments, we discovered that some residue (possibly high) errors remain in the non-linear system approximation for the case of subsystems having multiple inputs and a single output, in particular when the control loop is open. By comparing the training procedure that we were able to create for the normal cases with the training related to the anomalous ones, it is possible to observe that not all the possible dynamics seem to have been excited in this second class of cases. This is due to the characteristics of the conventional (but very sophisticated) simulator we used to generate the training data sequences for the neural configuration: some input sequences cannot in fact be presented to the conventional simulator since they generate problems in solving the model equations.

As a consequence, an uncomplete knowledge is stored in the neural network leading to relevant approximation errors during neural recall in the configured simulation environment. In this cases, the only solution seems to be creation of a comprehensive training sequence by using more complex simulators. Note that the real plant cannot often be excited to extract the sequence directly from the real system due to the possible criticality of the input excitations for the security and safety of the plant itself.

The second problem is related to the preservation of physical quantities such as mass and energy, in particular in steady conditions. In some cases, both the dynamics and the steady behavior of the system are captured by the neural model, correctly within the maximum approximation error predefined for the network training. Since the neural output is accepted as correct even if it differs from the exact value generated by the real system for less than the maximum approximation error, the neural outputs representing some of the physical quantities mentioned above may not satisfy the preservation equations *exactly*. This is actually not a problem for the subsequent simulation steps since the outputs will continue to remain within the acceptable value band centered on the exact value and large as twice the maximum approximation error. However, the simulator user sees these outputs (e.g., since they are the water incoming or outcoming the boiler in the power plant), he may be shocked and he may lose confidence in capabilities and correctness of the simulator itself. Tricks could be experimentally found to guarantee preservation at steady conditions, but they usually lead to unacceptable errors in dynamics. Research is in progress to create some correlations between these physical quantities within the neural model itself without modifying the dynamic behavior of the whole system.

REFERENCES

- [1] V. Piuri, "Design of a fault-tolerant distributed control systems", *IEEE Transaction on Instrumentation and Measurement*, Vol. 43, April 1994.
- [2] A. Gandelli, V. Piuri, "Design and implementation of a massively parallel architecture for high-level electronic measurements", *Microprocessors and Microsystems*, Vol. 19, August 1995.
- [3] C. Alippi, V. Piuri, "Experimental Neural Networks for Prediction and Identification", *IEEE Transaction on Instrumentation and Measurement*, Vol. 45, April 1996.
- [4] L. Marcocci, S. Spelta, "Computer aided modelling of complex processes: a program package", *Proc. IMACS'83*, Nantes, France, 1983.
- [5] P.S. Sastry, G. Sanatharam, "Memory neuron networks for identification and control of dynamical systems", *IEEE Trans. on Neural Networks*, Vol. 5, March 1994.
- [6] K.S. Narendra, K. Parthasarathy, "Identification and control of dynamical systems using neural networks", *IEEE Transactions on Neural Networks*, Vol. 1, 1990.