Cadini Francesco

Zio Enrico

Pedroni Nicola
Department of Nuclear Engineering, Polytechnic of Milan, Milan, Italy

Recurrent neural networks for dynamic reliability analysis

Keywords

dynamic reliability analysis, infinite impulse response-locally recurrent neural network, long-term non-linear dynamics, system state memory, simplified nuclear reactor

Abstract

A dynamic approach to the reliability analysis of realistic systems is likely to increase the computational burden, due to the need of integrating the dynamics with the system stochastic evolution. Hence, fast-running models of process evolution are sought. In this respect, empirical modelling is becoming a popular approach to system dynamics simulation since it allows identifying the underlying dynamic model by fitting system operational data through a procedure often referred to as 'learning'. In this paper, a Locally Recurrent Neural Network (LRNN) trained according to a Recursive Back-Propagation (RBP) algorithm is investigated as an efficient tool for fast dynamic simulation. An application is performed with respect to the simulation of the non-linear dynamics of a nuclear reactor, as described by a simplified model of literature.

1. Introduction

Dynamic reliability aims at broadening the classical event tree/ fault tree methodology so as to account for the mutual interactions between the hardware components of a plant and the physical evolution of its process variables. The dynamical aspects concern the ordering and timing of events in the accident propagation, the dependence of transition rates and failure criteria on the process variable values, the human operator and control actions. Obviously, a dynamic approach to reliability analysis would not bear any significant added value to the analysis of systems undergoing slow accidental transients for which the control variables do not vary in such a way to affect the component transition rates and/or to demand the intervention of the control.

Dynamic reliability methods are based on a powerful mathematical framework capable of integrating the interactions between the components and the environment in which they function. These methods perform a more realistic modelling of the system and hence improve the quality and accuracy of risk assessment studies. A formal approach to incorporating the dynamic behaviour of systems in risk analysis was formulated under the name Probabilistic Dynamics [10]. Several methods for tackling the solution to the dynamic reliability problem have been formulated over the past ten years [1], [9], [13], [15], [16], [20]. Among these, Monte Carlo methods have demonstrated to be particularly efficient in taking up the numerical burden of such analysis, while allowing for flexibility in the assumptions and for a thorough uncertainty and sensitivity analysis [14], [16].

For realistic systems, a dynamic approach to reliability analysis is likely to require a significant increase in the computational efforts, due to the need of integrating the dynamic evolution, with its characteristic times, with the system stochastic evolution characterized by very different time constants. The fast increase in computing power has rendered, and will continue to render, more and more feasible the incorporation of dynamics in the safety and reliability models of complex engineering systems. In particular, as mentioned above, the Monte Carlo simulation framework offers a natural environment for estimating the reliability of systems

with dynamic features. However, the high reliability of systems and components favours the adoption of forced transition schemes and leads, correspondingly, to an increment of the integration of physical models in each trial. Thus, the time-description of the dynamic processes may render the Monte Carlo simulation quite burdensome and it becomes mandatory to resort to fast-running models of process evolution. In these cases, one may resort to either simplified, reduced analytical models, such as those based on lumped effective parameters [2], [7], [8], or empirical models. In both cases, the model parameters have to be estimated so as to best fit to the available plant data.

In the field of empirical modelling, considerable interest is devoted to Artificial Neural Networks (ANNs) because of their capability of modelling non-linear dynamics and of automatically calibrating their parameters from representative input/output data [16]. Whereas feedforward neural networks can model static input/output mappings but do not have the capability of reproducing the behaviour of dynamic systems, dynamic Recurrent Neural Networks (RNNs) are recently attracting significant attention, because of their potentials in temporal processing. Indeed, recurrent neural networks have been proven to constitute universal approximates of non-linear dynamic systems [19].

Two main methods exist for providing a neural network with dynamic behaviour: the insertion of a buffer somewhere in the network to provide an explicit memory of the past inputs, or the implementation of feedbacks.

As for the first method, it builds on the structure of feedforward networks where all input signals flow in one direction, from input to output. Then, because a feedforward network does not have a dynamic memory, *tapped-delay-lines* (temporal buffers) of the inputs are used. The buffer can be applied at the network inputs only, keeping the network internally static as in the buffered multilayer perceptron (MLP) [11], or at the input of each neuron as in the MLP with Finite Impulse Response (FIR) filter synapses (FIR-MLP) [4]. The main disadvantage of the buffer approach is the limited past-history horizon, which needs to be used in order to keep the size of the network computationally manageable, thereby preventing modelling of arbitrary long time dependencies between inputs and outputs [12]. It is also difficult to set the length of the buffer given a certain application.

Regarding the second method, the most general example of implementation of feedbacks in a neural network is the fully recurrent neural network constituted by a single layer of neurons fully interconnected with each other or by several such layers [18]. Because of the required large structural complexity of this network, in recent years growing efforts have been propounded in developing methods for implementing temporal dynamic feedback connections into the widely used multi-layered feedforward neural networks. Recurrent connections can be added by using two main types of recurrence or feedback: *external* or *internal*. *External recurrence* is obtained for example by feeding back the outputs to the input of the network as in NARX networks [5], [17]; *internal recurrence* is obtained by feeding back the outputs of neurons of a given layer in inputs to neurons of the same layer, giving rise to the so called *Locally Recurrent Neural Networks* (*LRNNs*) [6].

The major advantages of LRNNs with respect to the buffered, tapped-delayed feedforward networks and to the fully recurrent networks are [6]: 1) the hierarchic multilayer topology which they are based on is well known and efficient; 2) the use of dynamic neurons allows to limit the number of neurons required for modelling a given dynamic system, contrary to the tapped-delayed networks; 3) the training procedures for properly adjusting the network weights are significantly simpler and faster than those for the fully recurrent networks.

In this paper, an Infinite Impulse Response-Locally Recurrent Neural Network (IIR-LRNN) is adopted together with the Recursive Back-Propagation (RBP) algorithm for its batch training [6]. In the IIR-LRNN the synapses are implemented as Infinite Impulse Response digital filters, which provide the network with system state memory.

The proposed neural approach is applied to a highly non-linear dynamic system of literature, the continuous time Chernick model of a simplified nuclear reactor [8]: the IIR-LRNN is devised to estimate the neutron flux temporal evolution only knowing the reactivity forcing function. The IIR-LRNN ability of dealing with both the short-term dynamics governed by the instantaneous variations of the reactivity and the long-term dynamics governed by *Xe* oscillations is verified by extensive simulations on training, validation and test transients.

The paper is organized as follows: in Section 2, the IIR-LRNN architecture is presented in detail together with the RBP training algorithm; in Section 3, the adopted neural approach is applied to simulate the reactor neutron flux dynamics. Finally, some conclusions are proposed in the last Section.

2. Locally Recurrent Neural Networks

2.1. The IIR-LRNN architecture and forward calculation

A LRNN is a time-discrete network consisting of a global feed-forward structure of nodes interconnected by synapses which link the nodes of the k-th layer to those of the successive $(k + 1)$-th layer, $k = 0, 1, \ldots, M$, layer 0 being the input and M the output. Differently from the classical static feed-forward networks, in an LRNN each synapse carries taps and feedback connections. In particular, each synapse of an IIR-LRNN contains an IIR linear filter whose characteristic transfer function can be expressed as ratio of two polynomials with poles and zeros representing the AR and MA part of the model, respectively.

For simplicity of illustration, and with no loss of generality, we start by considering a network constituted by only one hidden layer, i.e. $M = 2$, like the one in *Figure 1*. At the generic time t, the input to the LRNN consists of a pattern $x(t) \in \Re^{N^0}$, whose components feed the nodes of the input layer 0 which simply transmit in output the input received, i.e. $x^0_m(t) = x_m(t)$, $m = 1, 2, \ldots, N^0$. A bias node is also typically inserted, with the index $m = 0$, such that $x^0_0(t) = 1$ for all values of t. The output variable of the m-th input node at time t is tapped a number of delays $L^1_{nm} - 1$ (except for the bias node output which is not tapped, i.e. $L^1_{n0} - 1 = 0$) so that from each input node $m \neq 0$ actually L^1_{nm} values, $x^0_m(t)$, $x^0_m(t - 1)$, $x^0_m(t - 2)$, …, $x^0_m(t - L^1_{nm} + 1)$ are processed forward through the synapses connecting input node m to the generic hidden node $n = 1, 2, \ldots N^1$. The L^1_{nm} values sent from the input node m to the hidden node n are first multiplied by the respective synaptic weights $w^1_{nm(p)}$, $p = 0, 1, \ldots, L^1_{nm} - 1$ being the index of the tap delay (the synaptic weight $w^1_{n0(p)}$ connecting the bias input node $m = 0$ is the bias value itself) and then processed by a summation operator to give the MA part of the model with transfer function

$$w^1_{nm(0)} + w^1_{nm(1)}B + w^1_{nm(2)}B^2 + \ldots + w^1_{nm(L^1_{nm}-1)}B^{L^1_{nm}-1}, \quad (1)$$

B being the usual delay operator of unitary step. The finite set of weights $w^1_{nm(p)}$ which appear in the MA model form the so called impulse response function and represent the components of the MA part of the synaptic filter connecting input node m to hidden node n. The weighed sum thereby obtained, y^1_{nm}, is fed back, for a given number of delays I^1_{nm} ($I^1_{n0} = 0$ for the bias node) and weighed by the coefficient $v^1_{nm(p)}$ (the AR part of the synaptic filter connecting input node m to hidden node n, with the set of weights $v^1_{nm(p)}$ being the so-called AR filter's impulse response function), to the summation operator itself to give the output quantity of the synapse ARMA model:

$$y^1_{nm}(t) = \sum_{p=0}^{L^1_{nm}-1} w^1_{nm(p)} x^0_n(t-p) + \sum_{p=1}^{I^1_{nm}} v^1_{nm(p)} y^1_{nm}(t-p). \quad (2)$$

This value represents the output at time t of the IIR-filter relative to the nm-synapse, which connects the m-th input neuron to the n-th hidden neuron. The first sum in (2) is the MA part of the synaptic filter and the second is the AR part. As mentioned above, the index $m = 0$ usually represents the bias input node, such that $x^0_0(t)$ is equal to one for all values of t, $L^1_{n0} - 1 = I^1_{n0} = 0$ and thus, $y^1_{n0}(t) = w^1_{n0(0)}$.

The quantities $y^1_{nm}(t)$, $m = 0, 1, \ldots, N^0$, are summed to obtain the net input $s^1_n(t)$ to the non-linear activation function $f^1(\cdot)$, typically a sigmoid, Fermi function, of the n-th hidden node, $n = 1, 2, \ldots N^1$:

$$s^1_n(t) = \sum_{m=0}^{N^0} y^1_{nm}(t). \quad (3)$$

The output of the activation function gives the state of the n-th hidden neuron, $x^1_n(t)$:

$$x^1_n(t) = f^1\left[s^1_n(t)\right]. \quad (4)$$

The output values of the nodes of the hidden layer 1, $x^1_n(t)$, $n = 1, 2, \ldots, N^1$, are then processed forward along the AR and MA synaptic connections linking the hidden and output nodes, in a manner which is absolutely analogous to the processing between the input and hidden layers. A bias node with index $n = 0$ is also typically inserted in the hidden layer, such that $x^1_0(t) = 1$ for all values of t.

The output variable of the n-th hidden node at time t is tapped a number of delays $L^M_{rn} - 1$ (= 0 for the bias node $n = 0$) so that from each hidden node n actually L^M_{rn} values, $x^1_n(t)$, $x^1_n(t - 1)$, $x^1_n(t - 2)$, …, $x^1_n(t - L^M_{rn} + 1)$, are processed forward through the MA-synapses connecting the hidden node n to the output node $r = 1, 2, \ldots, N^M$. The L^M_{rn} values sent from the hidden node n to the output node r are first multiplied by the respective synaptic weights $w^M_{rn(p)}$, $p = 0, 1, \ldots, L^M_{rn} - 1$ being the index of the tap delay (the synaptic weight w^M_{r0} connecting the bias hidden node $n = 0$ is the bias value itself) and then processed by a summation operator to give the MA part of the model with transfer function

$$w^M_{rn(0)} + w^M_{rn(1)}B + w^M_{rn(2)}B^2 + \ldots + w^M_{rn(L^M_{rn}-1)}B^{L^M_{rn}-1}. \quad (5)$$

The sum of these values, y^M_{rn}, is fed back, for a given number of delays I^M_{rn} ($I^M_{r0} = 0$ for the bias node) and weighed by the coefficient $v^M_{rn(p)}$ (the AR part of the synaptic filter connecting hidden node n to output node r, with the set of weights

$v^M_{rn(p)}$ being the corresponding impulse response function), to the summation operator itself to give the output quantity of the synapse ARMA model:

$$y^M_m(t) = \sum_{p=0}^{L^M_{rn}-1} w^M_{m(p)} x^1_n(t-p) + \sum_{p=1}^{I^M_{rn}} v^M_{m(p)} y^M_m(t-p). \quad (6)$$

As mentioned before, the index $n = 0$ represents the bias hidden node, such that $x^1_0(t)$ is equal to one for all values of t, $L^M_{r0} - 1 = I^M_{r0} = 0$ and thus, $y^M_{r0}(t) = w^M_{r0(0)}$.
The quantities $y^M_{rn}(t)$, $n = 0, 1, \ldots, N^1$, are summed to obtain the net input $s^M_r(t)$ to the non-linear activation function $f^M(\cdot)$, also typically a sigmoid, Fermi function, of the r-th output node $r = 1, 2, \ldots, N^M$:

$$s^M_r(t) = \sum_{n=0}^{N^1} y^M_m(t). \quad (7)$$

The output of the activation function gives the state of the r-th output neuron, $x^M_r(t)$:

$$x^M_r(t) = f^M\left[s^M_r(t)\right]. \quad (8)$$

The extension of the above calculations to the case of multiple hidden layers ($M > 2$) is straightforward. The time evolution of the generic neuron j belonging to the generic layer $k = 1, 2, \ldots, M$ is described by the following equations:

$$x^k_j(t) = f^k\left[s^k_j(t)\right] (= 1 \text{ for the bias node, } j = 0), \quad (9)$$

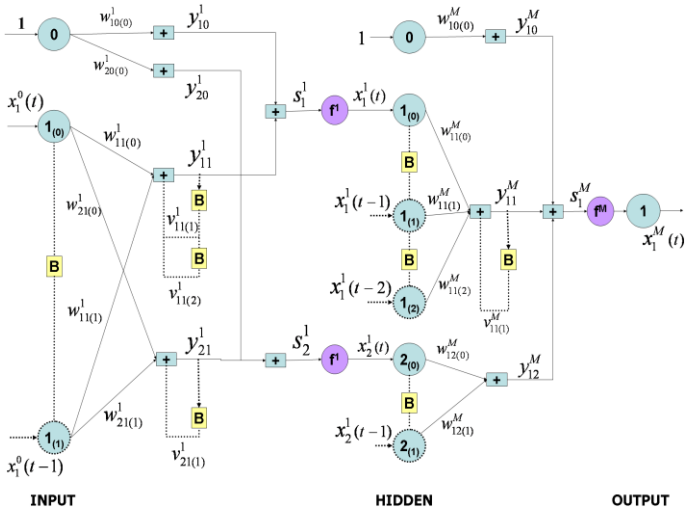
$$s^k_j(t) = \sum_{l=0}^{N^{k-1}} y^k_{jl}(t), \quad (10)$$

$$y^k_{jl}(t) = \sum_{p=0}^{L^k_{jl}-1} w^k_{jl(p)} x^{k-1}_l(t-p) + \sum_{p=1}^{I^k_{jl}} v^k_{jl(p)} y^k_{jl}(t-p). \quad (11)$$

Note that if all the synapses contain only the MA part (i.e., $I^k_{jl} = 0$ for all j, k, l), the architecture reduces to a FIR-MLP and if all the synaptic filters contain no memory (i.e., $L^k_{jl} - 1 = 0$ and $I^k_{jl} = 0$ for all j, k, l), the classical multilayered feed-forward static neural network is obtained.

2.2. The Recursive Back-Propagation (RBP) algorithm for batch training

The Recursive Back-Propagation (RBP) training algorithm [6] is a gradient - based minimization algorithm which makes use of a particular chain rule expansion rule expansion for the computation of the necessary derivatives. A thorough description of the RBP training algorithm is given in the Appendix at the end of the paper.



INPUT ($k = 0$)	HIDDEN ($k = 1$)	OUTPUT ($k = 2 = M$)
$N^0 = 1$	$N^1 = 2$	$N^M = 1$
	$L^1_{11} - 1 = 1$ $L^1_{21} - 1 = 1$	$L^M_{11} - 1 = 2$ $L^M_{12} - 1 = 1$
	$I^1_{11} = 2$ $I^1_{21} = 1$	$I^M_{11} = 1$ $I^M_{12} = 0$

Figure 1. Scheme of an IIR-LRNN with one hidden layer

3. Simulating reactor neutron flux dynamics by LRNN

In general, the training of an ANN to simulate the behaviour of a dynamic system can be quite a difficult task, mainly due to the fact that the values of the system output vector $\boldsymbol{y}(t)$ at time t depend on both the forcing functions vector $\boldsymbol{x}(\cdot)$ and the output $\boldsymbol{y}(\cdot)$ itself, at previous steps:

$$\boldsymbol{y}(t) = F(\boldsymbol{x}(t), \boldsymbol{x}(t-1), \ldots, \boldsymbol{y}(t-1), \ldots, \boldsymbol{\Theta}), \quad (12)$$

where $\boldsymbol{\Theta}$ is a set of adjustable parameters and $F(\cdot)$ the non-linear mapping function describing the system dynamics.
In this Section, a locally recurrent neural network is trained to simulate the dynamic evolution of the neutron flux in a nuclear reactor.

3.1. Problem formulation

The reference dynamics is described by a simple model based on a one group, point kinetics equation with non-linear power reactivity feedback, combined with Xenon and Iodine balance equations [8]:

$$\Lambda \frac{d\Phi}{dt} = \left[(\rho_0 + \Delta\rho) - \frac{\sigma_{Xe}}{c\Sigma_f} Xe - \gamma\Phi \right] \Phi$$

$$\frac{dXe}{dt} = \gamma_{Xe}\Sigma_f \Phi + \lambda_I I - \lambda_{Xe} Xe - \sigma_{Xe} Xe\Phi \qquad (13)$$

$$\frac{dI}{dt} = \gamma_I \Sigma_f \Phi - \lambda_I I$$

where Φ, Xe and I are the values of flux, Xenon and Iodine concentrations, respectively.
The reactor evolution is assumed to start from an equilibrium state at a nominal flux level $\Phi_0 = 4.66 \cdot 10^{12}$ n/cm^2s. The initial reactivity needed to keep the steady state is $\rho_0 = 0.071$ and the Xenon and Iodine concentrations are $Xe_0 = 5.73 \cdot 10^{15}$ $nuclei/cm^3$ and $I_0 = 5.81 \cdot 10^{15}$ $nuclei/cm^3$, respectively. In the following, the values of flux, Xenon and Iodine concentrations are normalized with respect to these steady state values.
The objective is to design and train a LRNN to reproduce the neutron flux dynamics described by the system of differential equations (13), i.e. to estimate the evolution of the normalized neutron flux $\Phi(t)$, knowing the forcing function $\rho(t)$.
Notice that the estimation is based only on the current values of reactivity. These are fed in input to the locally recurrent model at each time step t: thanks to the MA and AR parts of the synaptic filters, an estimate of the neutron flux $\Phi(t)$ at time t is produced which recurrently accounts for past values of both the network's inputs and the estimated outputs, viz.

$$\hat{\Phi}(t) = F(\rho(t), \rho(t-1), ..., \hat{\Phi}(t-1), ..., \boldsymbol{\Theta}) \qquad (14)$$

where $\boldsymbol{\Theta}$ is the set of adjustable parameters of the network model, i.e. the synaptic weights.
On the contrary, the other non-measurable system state variables, $Xe(t)$ and $I(t)$, are not fed in input to the LRNN: the associated information remains distributed in the hidden layers and connections. This renders the LRNN modelling task quite difficult.

3.2. Design and training of the LRNN

The LRNN used in this work is characterized by three layers: the input, with two nodes (bias included); the hidden, with six nodes (bias included); the output with one node. A sigmoid activation function has been adopted for the hidden and output nodes.
The training set has been constructed with $N_t = 250$ transients, each one lasting $T = 2000$ minutes and sampled with a time step Δt of 40 minutes, thus generating $n_p = 50$ patterns. Notice that a temporal length of 2000 minutes allows the development of the long-term dynamics, which are affected by the long-term Xe oscillations. All data have been normalized in the range [0.2, 0.8].
Each transient has been created varying the reactivity from its steady state value according to the following step function:

$$\rho(t) = \begin{cases} \rho_0 & t \le T_s \\ \rho_0 + \Delta\rho & t > T_s \end{cases} \qquad (15)$$

where T_s is a random steady-state time interval and $\Delta\rho$ is random reactivity variation amplitude. In order to build the 250 different transients for the training, these two parameters have been randomly chosen within the ranges [0, 2000] minutes and $[-5 \cdot 10^{-4}, +5 \cdot 10^{-4}]$, respectively.
The training procedure has been carried out on the available data for $n_{epoch} = 200$ learning epochs (iterations). During each epoch, every transient is repeatedly presented to the LRNN for $n_{rep} = 10$ consecutive times. The weight updates are performed in batch at the end of each training sequence of length T. No momentum term nor an adaptive learning rate [6] turned out necessary for increasing the efficiency of the training, in this case.
Ten training runs have been carried out to set the number of delays (orders of the MA and AR parts of the synaptic filters) so as to obtain a satisfactory performance of the LRNN, measured in terms of a small root mean square error (RMSE) on the training set.
As a result of these training runs, the MA and AR orders of the IIR synaptic filters have been set to 12 and 10, respectively, for both the hidden and the output neurons.

3.3. Results

The trained LRNN is first verified with respect to its capability of reproducing the transients employed for the training itself. This capability is a minimum requirement, which however does not guarantee the proper general functioning of the LRNN when new transients, different from those of training, are fed into the network. The evolution of the flux, normalized with respect to the steady state value Φ_0, corresponding to one sample training transients is shown in *Figure 2*: as expected, the LRNN estimate of the output (crosses) is in satisfactory agreement with the actual transient (circles).
Notice the ability of the LRNN of dealing with both the short-term dynamics governed by the instantaneous variations of the forcing function (i.e., the reactivity step) and the long-term dynamics governed by Xe oscillations.

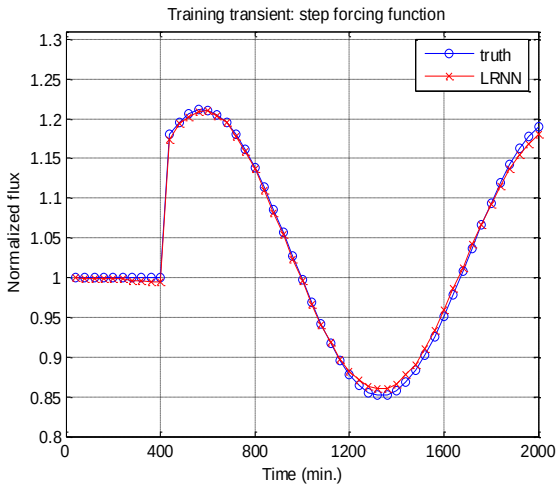


Figure 2. Comparison of the model-simulated normalized flux (circles) with the LRNN-estimated one (crosses), for two sample transients of the training set

3.3.1. Validation phase: training like dynamics

The procedure for validating the generalization capability of the LRNN to transients different from those of training is based on N_t = 80 transients of T = 2000 minutes each, initiated again by step variations in the forcing function $\rho(t)$ as in eq. (15), with timing and amplitude randomly sampled in the same ranges as in the training phase.
The results reported in *Figure 3* confirm the success of the training since the LRNN estimation errors are still small for these new transients. Furthermore, the computing time is about 5000 times lower than that required by the numerical solution of the model. This makes the LRNN model very attractive for real time applications, e.g. for control or diagnostic purposes, and for applications for which repeated evaluations are required, e.g. for uncertainty and sensitivity analyses.

3.3.2. Test phase

The generalization capabilities of the trained and validated LRNN have been then tested on a new set of transients generated by forcing functions variations quite different from those used in both the training and the validation phases. The test set consists of three transients batches created by three functional shapes of the forcing function $\rho(t)$ never seen by the LRNN:

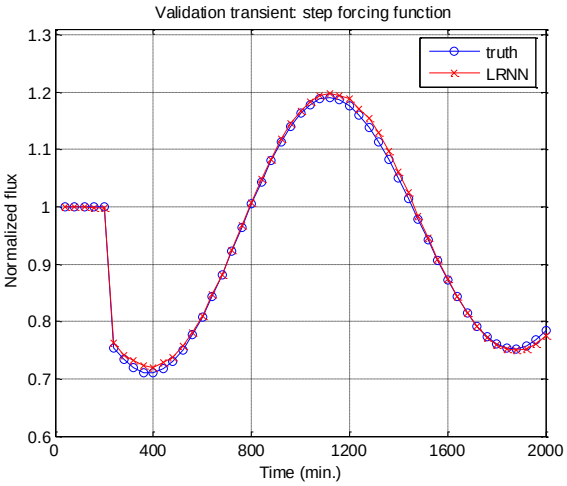


Figure 3. Comparison of the model-simulated normalized flux (circles) with the LRNN-estimated one (crosses), for one sample transient of the validation set

- A ramp function:

$$\rho(t) = \begin{cases} \rho_0, t \le T_s \\ \rho_0 + (\Delta\rho / T_v)t - (\Delta\rho / T_v)T_s, T_s < t \le T_s + T_v \\ \rho_0 + \Delta\rho, t > T_s + T_v \end{cases} \quad (16)$$

where the steady-state time interval T_s ($0 \le T_s \le$ 2000 min), the ramp variation time interval T_v ($0 \le T_v \le$ 2000 min) and the reactivity variation amplitude $\Delta\rho$ ($-5\cdot10^{-4} \le \Delta\rho \le +5\cdot10^{-4}$) are randomly extracted in their ranges of variation in order to generate the different transients;

- A sine function:

$$\rho(t) = \Delta\rho \cdot \sin(2\pi f t), \quad (17)$$

where f is the oscillation frequency ($1 \le f \le 2$ min^{-1}) and $\Delta\rho$ ($-5\cdot10^{-4} \le \Delta\rho \le +5\cdot10^{-4}$) is the reactivity variation amplitude;

- Random reactivity variation amplitude with a uniform probability density function between $-5\cdot10^{-4}$ and $+5\cdot10^{-4}$.

A total of N_t = 80 temporal sequences has been simulated for each batch, producing a total of 240 test transients. The temporal length and the sampling time steps of each transient are the same as those of the training and validation sets (2000 and 40 minutes, respectively).
Figures 4, 5 and *Figure 6* show a satisfactory agreement of the LRNN estimation with the model simulation, even for cases quite different from the dynamic evolution considered during training.

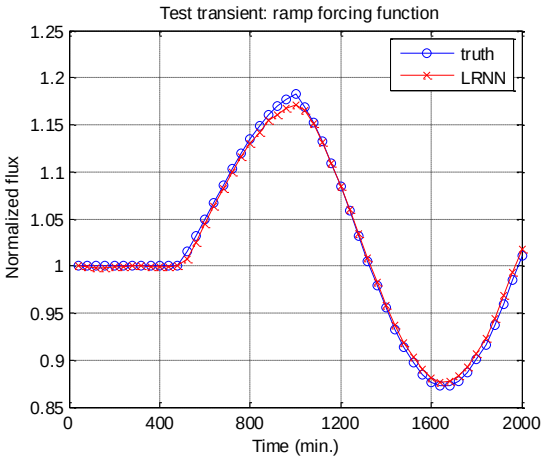


Figure 4. Comparison of the model-simulated normalized flux (circles) with the LRNN-estimated one (crosses), for one sample ramp transient of the test set

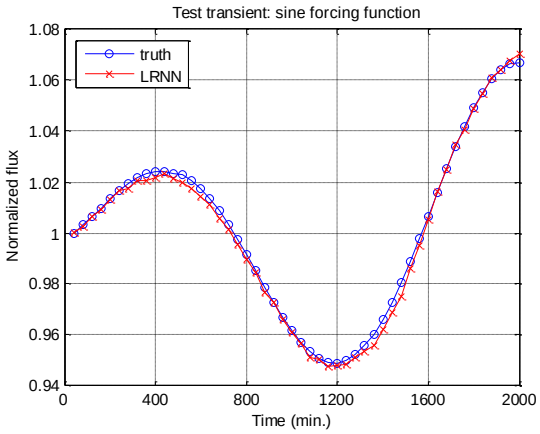


Figure 5. Comparison of the model-simulated normalized flux (circles) with the LRNN-estimated one (crosses), for one sample sinusoidal transient of the test set

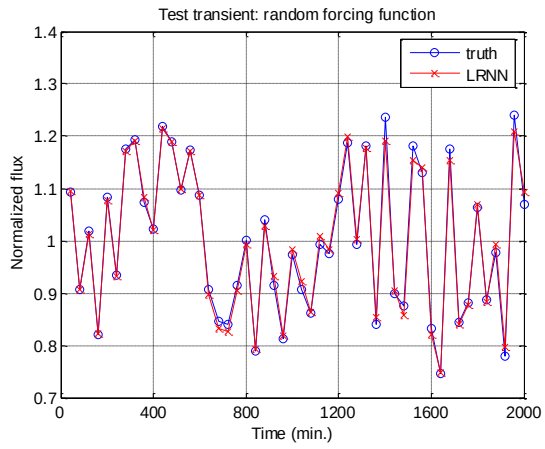


Figure 6. Comparison of the model-simulated normalized flux (circles) with the LRNN-estimated one (crosses), for one sample random transient of the test set

These results are synthesized in *Table 1*, in terms of the following performance indices: root mean square error (RMSE) and mean absolute error (MAE).

Table 1. Values of the performance indices (RMSE and MAE) calculated over the training, validation and test sets for the LRNN applied to the reactor neutron flux estimation

			ERRORS	
Set	***Forcing function***	***n. of sequences***	***RMSE***	***MAE***
Training	Step	250	0.0037	0.0028
Validation	Step	80	0.0098	0.0060
Test	Ramp	80	0.0049	0.0039
	Sine	80	0.0058	0.0051
	Random	80	0.0063	0.0054

4. Conclusion

Dynamic reliability analyses entail the rapid simulation of the system dynamics under the different scenarios and configurations, which occur during the system stochastic life evolution. However, the complexity and nonlinearities of the involved processes are such that analytical modelling becomes burdensome, if at all feasible.

In this paper, the framework of Locally Recurrent Neural Networks (LRNNs) for non-linear dynamic simulation has been presented in detail. The powerful dynamic modelling capabilities of this type of neural networks has been demonstrated on a case study concerning the evolution of the neutron flux in a nuclear reactor as described by a simple model of literature, based on a one group, point kinetics equation with non-linear power reactivity feedback, coupled with the Xenon and Iodine balance equations.

An Infinite Impulse Response-Locally Recurrent Neural Network (IIR-LRNN) has been successfully designed and trained, with a Recursive Back-Propagation (RBP) algorithm, to the difficult task of estimating the evolution of the neutron flux, only knowing the reactivity evolution, since the other non measurable system state variables, i.e. Xenon and Iodine concentrations, remain hidden.

The findings of the research seem encouraging and confirmatory of the feasibility of using recurrent neural network models for the rapid and reliable system simulations needed in dynamic reliability analysis.

References

[1] Aldemir, T., Siu, N., Mosleh, A., Cacciabue, P.C. & Goktepe, B.G. (1994). Eds.: *Reliability and Safety Assessment of Dynamic Process System*

NATO-ASI Series F, Vol. 120 Springer-Verlag, Berlin.

[2] Aldemir, T., Torri, G., Marseguerra, M., Zio, E. & Borkowski, J. A. (2003). Using point reactor models and genetic algorithms for on-line global xenon estimation in nuclear reactors. *Nuclear Technology*, 143, No. 3, 247-255.

[3] Back, A. D. & Tsoi, A. C. (1993). A simplified gradient algorithm for IIR synapse multi-layer perceptron. *Neural Comput.* 5: 456-462.

[4] Back, A. D. et al. (1994). A Unifying View of Some Training Algorithms for Multilayer Perceptrons with FIR Filter Synapses. *Proc. IEEE Workshop Neural Netw. Signal Process.*: 146.

[5] Boroushaki, M. et al. (2003). Identification and control of a nuclear reactor core (VVER) using recurrent neural networks and fuzzy system. *IEEE Trans. Nucl. Sci.* 50(1): 159-174.

[6] Campolucci, P. et al. (1999). On-Line Learning Algorithms of Locally Recurrent Neural Networks. *IEEE Trans. Neural Networks* 10: 253-271.

[7] Carlos, S., Ginestar, D., Martorell, S. & Serradell, V. (2003). Parameter estimation in thermalhydraulic models using the multidirectional search method. *Annals of Nuclear Energy* 30, 133-158.

[8] Chernick, J. (1960). The dynamics of a xenon-controlled reactor. *Nuclear Science and Engineering* 8: 233-243.

[9] Cojazzi, G., Izquierdo, J.M., Melendez, E. & Sanchez-Perea, M. (1992). The Reliability and Safety Assessment of Protection Systems by the Use of Dynamic Event Trees (DET). The DYLAM-TRETA package. *Proc. XVIII annaul meeting Spanish Nuclear Society*.

[10] Devooght, J. & Smidts, C. (1992). Probabilistic Reactor Dynamics I. The Theory of Continuous Event Trees, *Nucl. Sci. and Eng.* 111, 3, pp. 229-240.

[11] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. New York: IEEE Press.

[12] Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8): 1735-1780.

[13] Izquierdo, J.M., Hortal, J., Sanchez-Perea, M. & Melendez, E (1994). Automatic Generation of dynamic Event Trees: A Tool for Integrated Safety Assessment (ISA), *Reliability and Safety Assessment of Dynamic Process System NATO-ASI Series F*, Vol. 120 Springer-Verlag, Berlin.

[14] Labeau, P. E. & Zio, E. (1998). The Cell-to-Boundary Method in the Frame of Memorization-Based Monte Carlo Algorithms. A New Computational Improvement in Dynamic Reliability, *Mathematics and Computers in Simulation*, Vol. 47, No. 2-5, 329-347.

[15] Labeau, P.E. (1996). Probabilistic Dynamics: Estimation of Generalized Unreliability Trhough Efficient Monte Carlo Simulation, *Annals of Nuclear Energy*, Vol. 23, No. 17, 1355-1369.

[16] Marseguerra, M. & Zio, E. (1996). Monte Carlo approach to PSA for dynamic process systems, *Reliab. Eng. & System Safety*, vol. 52, 227-241.

[17] Narendra, K. S. & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks* 1: 4-27.

[18] Pearlmutter, B. (1995). Gradient Calculations for Dynamic Recurrent Neural networks: a Survey. *IEEE Trans. Neural Networks* 6: 1212.

[19] Siegelmann, H. & Sontag, E. (1995). On the Computational Power of Neural Nets. *J. Computers and Syst. Sci.* 50 (1): 132.

[20] Siu, N. (1994). Risk Assessment for Dynamic Systems: An Overview, Reliab. Eng. & System Safety, vol. 43, 43-74.

Appendix: the Recursive Back-Propagation (RBP) Algorithm for batch training

Consider one training temporal sequence of length T and denote by $d_r(t)$, $r = 1, 2, \ldots, N^M$, the desired output value of the training sequence at time t.
The instantaneous squared error at time t, $e^2(t)$, is defined as the sum over all N^M output nodes of the squared deviations of the network outputs $x^M_r(t)$ from the corresponding desired value in the training temporal sequence, $d_r(t)$:

$$e^2(t) = \sum_{r=1}^{N^M} [e_r(t)]^2 , \qquad (1')$$

where

$$e_r(t) = d_r(t) - x_r^M(t) . \qquad (2')$$

The training algorithm aims at minimizing the global squared error E^2 over the whole training sequence of length T,

$$E^2 = \sum_{t=1}^{T} e^2(t) , \qquad (3')$$

This is achieved by modifying iteratively the network weights $w^k_{jl(p)}$, $v^k_{jl(p)}$ along the gradient descent, viz.

$$\Delta w^k_{jl(p)} = -\frac{\mu}{2}\frac{\partial E^2}{\partial w^k_{jl(p)}}, \qquad \Delta v^k_{jl(p)} = -\frac{\mu}{2}\frac{\partial E^2}{\partial v^k_{jl(p)}}, \tag{4'}$$

where μ is the learning rate.
Introducing the usual *backpropagating error* and *delta* quantities with respect to the output, $x^k_j(t)$, and input, $s^k_j(t)$, of the generic node j of layer k:

$$e^k_j(t) = -\frac{1}{2}\frac{\partial E^2}{\partial x^k_j(t)}, \tag{5'}$$

$$\delta^k_j(t) = -\frac{1}{2}\frac{\partial E^2}{\partial s^k_j(t)} = -\frac{1}{2}\frac{\partial E^2}{\partial x^k_j(t)}\frac{\partial x^k_j(t)}{\partial s^k_j(t)} = e^k_j(t)\, f_k{}'\left[s^k_j(t)\right], \tag{6'}$$

the chain rule for the modification (4') of the MA and AR synaptic weights $w^k_{jl(p)}$, $v^k_{jl(p)}$ can be written as

$$\Delta w^k_{jl(p)} = -\frac{\mu}{2}\sum_{t=1}^{T}\frac{\partial E^2}{\partial s^k_j(t)}\frac{\partial s^k_j(t)}{\partial w^k_{jl(p)}} = \sum_{t=1}^{T}\mu\delta^k_j(t)\frac{\partial s^k_j(t)}{\partial w^k_{jl(p)}}, \tag{7'}$$

$$\Delta v^k_{jl(p)} = -\frac{\mu}{2}\sum_{t=1}^{T}\frac{\partial E^2}{\partial s^k_j(t)}\frac{\partial s^k_j(t)}{\partial v^k_{jl(p)}} = \sum_{t=1}^{T}\mu\delta^k_j(t)\frac{\partial s^k_j(t)}{\partial v^k_{jl(p)}}.$$

Note that the weights updates (7') are performed in batch at the end of the training sequence of length T.
From (10),

$$\frac{\partial s^k_j(t)}{\partial w^k_{jl(p)}} = \frac{\partial y^k_{jl}(t)}{\partial w^k_{jl(p)}} \quad ; \quad \frac{\partial s^k_j(t)}{\partial v^k_{jl(p)}} = \frac{\partial y^k_{jl}(t)}{\partial v^k_{jl(p)}}, \tag{8'}$$

so that from the differentiation of (11) one obtains

$$\frac{\partial s^k_j(t)}{\partial w^k_{jl(p)}} = x^{k-1}_l(t-p) + \sum_{\tau=1}^{I^k_{jl}} v^k_{jl(\tau)}\frac{\partial s^k_j(t-\tau)}{\partial w^k_{jl(p)}}, \tag{9'}$$

$$\frac{\partial s^k_j(t)}{\partial v^k_{jl(p)}} = y^k_{jl}(t-p) + \sum_{\tau=1}^{I^k_{jl}} v^k_{jl(\tau)}\frac{\partial s^k_j(t-\tau)}{\partial v^k_{jl(p)}}. \tag{10'}$$

To compute $\delta^k_j(t)$ from (6'), we must be able to compute $e^k_j(t)$. Applying the chain rule to (5'), one has

$$e^k_j(t) = \sum_{q=1}^{N^{k+1}}\sum_{\tau=1}^{T} -\frac{1}{2}\frac{\partial E^2}{\partial s^{k+1}_q(\tau)}\frac{\partial s^{k+1}_q(\tau)}{\partial x^k_j(t)}, \; k < M\,. \tag{11'}$$

Under the hypothesis of synaptic filter temporal causality (according to which the state of a node at time t influences the network evolution only at successive times and not at previous ones), the summation along the time trajectory can start from $\tau = t$. Exploiting the definitions (6') and (8'), changing the variables as $\tau - p \rightarrow t$ and considering that for the output layer, i.e. $k = M$, the derivative $\partial E^2/\partial x^M_j(t)$ can be computed directly from (2'), the back-propagation of the error through the layers can be derived

$$e^k_j(t) = \begin{cases} e_j(t)\,(eq.2'), k = M \\ \sum_{p=0}^{T-t}\sum_{q=1}^{N^{k+1}} \delta^{k+1}_q(t+p)\frac{\partial y^{k+1}_{qi}(t+p)}{\partial x^k_j(t)}, k < M, \end{cases} \tag{12'}$$

where from (11)

$$\frac{\partial y^{k+1}_{qj}(t+p)}{\partial x^k_j(t)} = \sum_{\tau=1}^{\min(I^{k+1}_{qj},\,p)} v^{k+1}_{qj(\tau)}\frac{\partial y^{k+1}_{qj}(t+p-\tau)}{\partial x^k_j(t)} + \begin{cases} w^{k+1}_{qj(p)}, 0 \le p \le L^{k+1}_{qj} - 1 \\ 0, otherwise. \end{cases} \tag{13'}$$