ICOSSAR 2005, G. Augusti, G.I. Schuëller, M. Ciampoli (eds)
© 2005 Millpress, Rotterdam, ISBN 90 5966 040 4

Neural network based Monte Carlo simulation of random processes

M. Beer
Department of Mechanical Engineering and Materials Science, Rice University, Houston, TX, USA
Institut für Statik und Dynamik der Tragwerke, Technische Universität Dresden, Germany

P. D. Spanos
Ryon Chair in Engineering, Rice University, Houston, TX, USA

Keywords: Monte Carlo simulation, neural networks, random processes

ABSTRACT: In this paper a procedure for Monte Carlo simulation of univariate stationary stochastic processes with the aid of neural networks is presented. As an alternative to traditional model-based simulation procedures, this one circumvents the difficulty of specifying a priori statistical properties of the process. This is particularly advantageous when only limited data are available. Neural networks operate model-free and learn directly from the data observed. They capture the pattern of short time series during the training procedure. The trained network can then generate a set of random process realizations which reflect the properties of the training data. In the present study a time lagged feed-forward network with logistic sigmoid activation functions is applied. The training of the network is realized by back-propagation with a generalized delta rule. An example demonstrates the usefulness of the proposed procedure.

1 MOTIVATION

Numerical simulations of stochastic processes have become an important task in many engineering fields. Monte Carlo approaches are particularly suitable tools for those simulation purposes. Their usefulness in diverse engineering applications has been well established over a period of decades; see Schuëller and Spanos (2001). If sufficient information about the underlying physics of a stochastic process is available, a numerical model for the process may be formulated. However, difficulties may be experienced in the case of limited information. If the data bank comprises only a single short data record, and no physical background knowledge about the process is available, the specification of power spectra and probability distributions to a sufficient degree of reliability may be problematic. Even if several process records exist, estimates of the process properties may be uncertain and perhaps cannot be obtained with an appropriate degree of confidence.

Neural network based procedures may provide a suitable tool for analyzing and representing stochastic processes in these problematic cases. Neural networks operate model-free, learn directly from the data observed, and generate predictions based on perceptions only. Assumptions or knowledge beyond the data set are not required. Moreover, it can be shown that the universal function approximation theorem is valid for neural networks which meet some minimum requirements; see e.g. Haykin (1999). That is, neural networks are capable of uniformly approximating any kind of nonlinear functions over a compact domain of definition to any degree of accuracy. Specifically, for any given real continuous function $g(\underline{x})$ on a compact set $U \subset \mathbb{R}^n$ and arbitrary $\varepsilon > 0$, there exists a neural network with the output $f(\underline{x})$ such that

$$\sup_{x \in U} \left| f(\underline{x}) - g(\underline{x}) \right| < \varepsilon \,. \qquad (1)$$

Each realization of a stochastic process which can be understood as such a continuous function g can then be approximated by a neural network. Conceptually, this relates to traditional methods such as ARMA for generating process realizations. However, the neural network procedure does not require a model specification. An appropriate randomization of the network based synthesis of process realizations then allows for a Monte Carlo simulation.

Starting from fundamental properties of neural networks discussed in references such as Bishop (1995) and Haykin (1999), an attempt is made herein to use their unique features for addressing problems in process simulation. In this context, it is assumed that only one short process realization has been observed, and no further information is available.

2 NETWORK COMPOSITION

2.1 General network configuration

In the present study, it is attempted to find a neural network structure that is simple and clearly arranged while it yields reliable results. In selecting an appropriate network configuration, a broad variety of neural network layouts are available. Basically, they consist of simple information processing units called neurons and information transferring links between the neurons – the synaptic connections. The neurons are arranged in a layered structure. Input signals are processed along a variety of paths through several neurons to compute output signals. The specific layout of the synaptic connections and the information processing rules within the neurons have to be defined in dependence on the particular problem.

If the generation of process realizations is understood as a problem of function approximation, we can make use of some experience from that field. Following these experiences a multi-layer perceptron network is selected as a basis. This network kind can readily satisfy the minimum requirements for the validity of the universal function approximation theorem. Only three layers, appropriate nonlinear activation functions φ(.) for processing signals within the neurons, a simple feed-forward architecture, and a sufficiently high number of hidden neurons are required (Fig. 1). The number of output neurons is determined by the dimensionality of the problem. To consider univariate processes, only one output neuron is sufficient.

Each neuron may receive several input signals x_j only from the previous layer. These are multiplied by the assigned synaptic weights w_{kj} and together with a bias value b_k are introduced into a summing junction. The weights and the bias allow the neuron to be adjusted to particular conditions. The summing junction generates the input argument for the subsequently called activation function φ(.). This yields the output signal y_k generated by neuron k. Specifically,

$$y_k = \varphi\left(\sum_{j=1}^{m} w_{kj} \cdot x_j + b_k \right) . \tag{2}$$

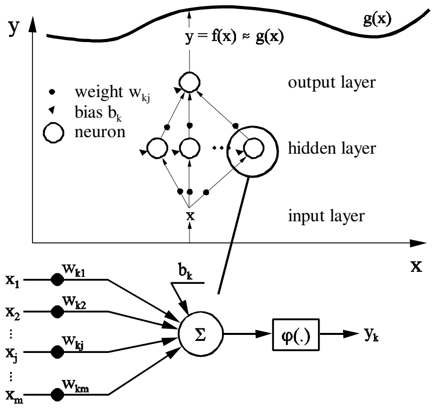

Figure 1. Multi-layer perceptron network for function approximation and general construct of a neuron

The activation function φ(.) is required to be nonlinear and monotonically increasing from zero to unity. Herein, the logistic sigmoid function

$$\varphi(x) = \left(1 + \exp(-x)\right)^{-1} \tag{3}$$

is selected. This function possesses the advantage that its derivatives φ'(x), which must be computed millions of times during the network training, can be obtained easily when the functional values φ(x) are already known. Specifically,

$$\varphi'(x) = \varphi(x) \cdot \left(1 - \varphi(x)\right) \tag{4}$$

This reduces the computational cost of the training procedure drastically; see Eqs. (14) and (15).

2.2 Process specific network features

2.2.1 Network layout elements

A neural network of the selected basic kind can realize an arbitrary nonlinear mapping. However, for being able to simulate a random process, it must additionally operate as a dynamic mapper. Since a feed-forward network works statically, it must be provided with memory for that purpose. Specifically, both short-term memory, and long-term memory features are required.

Short-term memory is established by implementing time into the network structure in an implicit manner. In this paper the network development is focused on dealing with stationary processes as a first step. For this purpose it is sufficient to attach a focused neuronal filter to the front end of the multi-layer perceptron; see Fig. 2. That is, account is taken of input signals not only from the current but also from previous time steps. A series of past input values are fed to the network at once. They are stored in a tapped delay line memory. This network kind is called focused time lagged feed-forward network and represents a nonlinear filter. It is capable of recognizing temporal patterns.

Long-term memory is built by adjusting the synaptic weights and biases of the network. In each time step and training cycle, one more piece of information is embedded into these parameter values. Once the training of the network is completed, the whole information contained in the training data set is stored as the final adjustment of the weights and biases. To extract local and global process features at least two hidden layers are required. This enables the neural network to analyze patterns that evolve over time.

The signal flow through the neural network in Fig. 2 may be summarized as

$$\begin{aligned} y &= \varphi\left(\sum_{j^{(3)}=1}^{J^{(3)}} \left[w_{j^{(4)} j^{(3)}} \cdot \varphi\left(v_{j^{(3)}}\right)\right] + b_{j^{(4)}} \right) \\ v_{j^{(3)}} &= \sum_{j^{(2)}=1}^{J^{(2)}} \left[w_{j^{(3)} j^{(2)}} \cdot \varphi\left(\sum_{j^{(1)}=1}^{J^{(1)}=m+1} \left[w_{j^{(2)} j^{(1)}} \cdot x_j \right] + b_{j^{(2)}} \right) \right] + b_{j^{(3)}} , \end{aligned} \qquad (5)$$

with $J^{(L)}$ denoting the number of neurons $j^{(L)}$ in layer L, $w_{j^{(L)} j^{(L-1)}}$ representing the weight for the signal from neuron $j^{(L-1)}$ in layer $L-1$ to neuron $j^{(L)}$ in layer L, and $b_{j^{(L)}}$ being the bias of neuron $j^{(L)}$ in layer L.

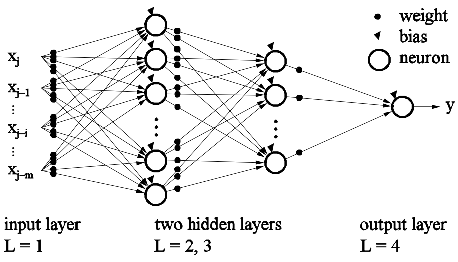

Figure 2. Focused time lagged feed-forward network

2.2.2 *Data conditioning*

A further prerequisite for the neural network is that it must be properly sensitive to fluctuations in the input signals. That is, a proper change in the value of an input signal to a neuron should generally lead to a noticeable difference in the neuron output. This is ensured if the neuron input signals preferably meet the effective part of the activation function φ(.). For the logistic sigmoid function from Eq. (3) the effective part may be defined as $x \in [x_{\min}, x_{\max}]$ with

$$\begin{aligned} x_{\min} &= -\ln\left(\frac{1}{\varepsilon} - 1 \right) , \\ x_{\max} &= -x_{\min} , \end{aligned} \qquad (6)$$

and ε being a prescribed minimum distance of the activation function φ(x) from its limits zero and unity, see Fig. 3. For example, $\varepsilon = 0.01$ yields the effective part $x \in [-\ln(99), +\ln(99)] \approx [-4.595, +4.595]$. Input signals from this interval lead to neuron outputs $y \in [0.01, 0.99]$. Fluctuations of input signals with values far outside the effective part virtually do not affect the further signal processing. In this manner, an impact from statistical outliers is eliminated.

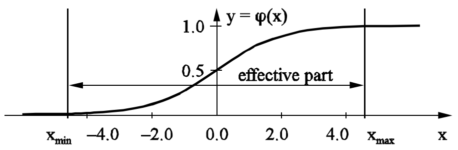

Figure 3. Effective part of the activation function

To be processed properly, the raw data must thus be preconditioned by applying some normalizing data transformation. Common methods make use of the extreme values of the training data set to define a possible range of signals and subsequently transform that range linearly to the effective part of the activation function. Though this is reasonable for analyzing data with a bounded value range, e.g. for controlling purposes, it may cause problems when processing statistical data. If the data bank consists of a small random sample as envisaged herein, there exists non-negligible probability that further sample elements may lie beyond the extreme values of the sample. According to the statistical estimation theory, these probabilities must be taken into account. They are of importance, in particular, in safety assessment. When applying those common transformations, however, these input signals which lie moderately outside the effective part are ignored. Thus, a common normalization with defined extrema is too rigorous in these cases.

A less restrictive and problem specific data transformation rule can be formulated by incorporating some statistics. Due to the small sample size only the first two statistical moments of the sample are employed. From the original (raw) data values z_j, the mean value $\bar{z}$ and the standard deviation σ_Z are computed. Subsequently, the z_j are transformed into the input signals x_j using the equation

$$x_j = \left(z_j - \bar{z}\right) \cdot \sigma_Z^{-1} \ . \tag{7}$$

In terms of probability, this circumvents ignoring input information from a reasonable neighborhood of the input data set. This can be expressed numerically by means of the probability P_{eff} with which the effective part of the activation function in the neurons of a network is met. The following example gives representative values of these probabilities. Again, the effective part of the activation function is defined by $\varepsilon = 0.01$. The weights and biases are assumed to be uniformly distributed in $[-1, +1]$. The probability P_{eff} for a neuron of the first hidden layer is estimated roughly by evaluating the summing junction in Eq. (2) with standard normal random variables for the x_j and the associated random weights and biases. If only one input signal x_1 is introduced, a probability of $P_{eff} = 0.99999777$ is obtained. With a tapped delay line memory that consists of six input nodes $P_{eff} = 0.995743$ is computed. For twelve input nodes the probability decreases to $P_{eff} = 0.9708$. And a memory of 24 input nodes still leads to $P_{eff} = 0.8891$. These results indicate that the network can process the input data records properly.

According to the definition of the activation function, the neuron output and hence the neural network output can take values $y \in (0, 1)$. The network output must thus be transformed into the scale of the particular problem. For generating process realizations a back-transformation of the network output y into the scale of the original process values z_j from the observed data record is required. That is, the envisaged back-transformation involves the inverse problem of transformation of the original process values z_j into the network input signals x_j. To ensure a proper scale matching between input and output signals, and to avoid a bias caused by the transformation rules the back-transformation is formulated as follows. In a first step, the network output y is brought into the standardized network input structure by introducing y in the inverse function of the standard normal distribution function $\Phi^{SN^{-1}}(y)$. The result is then introduced in a rewritten form of Eq. (7) to adapt the generated values to the properties of the original data record. That is, the generated process values are determined using the equation

$$p = \Phi^{SN^{-1}}(y) \cdot \sigma_Z + \bar{z} \ . \tag{8}$$

This back-transformation does not restrict the generated process values p to lie within artificially prescribed bounds.

3 GENERATING PROCESS REALIZATIONS

3.1 Network operation mode

The composed neural network must be supplemented with a suitable operation mode for being able to perform a process simulation. The starting point is the following basic situation, see Fig. 4, involving univariate processes. The process realizations consist of ordered sequences of real numbers z_n. The order is given by the time axis. The network input signals x_j, ..., x_{j-m}, see Fig. 2, are thus the transformed process values at the previous $r = m + 1$ successive points in time t_{n-r}, ..., t_{n-1}. Specifically,

$$\begin{aligned} z_{n-1} &= z(t_{n-1}) \rightarrow x_j \\ &\vdots \\ z_{n-i-1} &= z(t_{n-i-1}) \rightarrow x_{j-i} \\ &\vdots \\ z_{n-r} &= z(t_{n-r}) \rightarrow x_{j-m} \end{aligned} \ . \tag{9}$$

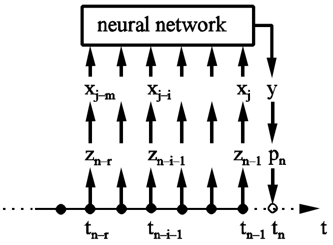

Figure 4. Process value prediction

The tapped delay line memory of that network is of order r. The single network output y then yields the predicted value p_n of the process at the present time t_n.

An application of this prediction scheme leads to the operation mode of a one-step prediction. This is generally initiated by the last observed sequence of process values as input vector. Exclusively observed values are fed to the neural network to predict only one subsequent process value, see Haykin (1999). That is, the neural network generates the process value for the time step t_n. And the prediction for t_{n+1} is made not until the observation at t_n is available. The time horizon of that network prediction is thus only one step. The accuracy of one-step predictions is quite high. Applications may be found in the forecasting of financial market developments, see Giles et al. (2001) and Li and Kozma (2003). Also, initial attempts to exploit the capability of this procedure for process prediction in engineering have been made. For instance, More and Deo (2003) have presented a neural network model for the prediction of maximum wind speeds for one day, one week, and one month.

The restriction to a time horizon of one step, however, is critical and does not exploit the capability of a neural network by far. For extending the time horizon, the following two approaches may be pursued. First, the envisaged time horizon may be incorporated directly into the neural network by furnishing the output layer with several neurons. The number of output neurons then determines the time horizon. This is advantageous for predicting short data sequences, for example, the development of the concrete strength over the first 28 days, see Lee (2003). The second approach involves a recursive application of the network in Fig. 4. Specifically, the network prediction for the process value at t_n is used as an input signal for predicting the process value at t_{n+1}. That is, the network predictions are fed back to the input layer instead of using observed data. A step-by-step marching in this manner then yields a long sequence of predicted process values as a first advancement. The time horizon is not limited a priori. This meets the requirements for a Monte Carlo simulation, and is thus chosen as a basis herein. The associated operation mode is a progressive prediction.

3.2 Single process realizations

In a first step, the progressive prediction mode is used to generate a single process realization. This requires a "perfectly trained" network with a zero prediction error. It is assumed that the network predictions p_n coincide with the observations z_n for all time steps t_n. This justifies feeding the network predictions back to the input layer of a feed-forward network. The feed-back of the network output is not activated until the network training is finished. That is, a recurrent network application is not pursued – even though it resembles the latter. The precondition of providing a "perfectly trained" network is satisfied by training the feed-forward network in Fig. 4 until the network prediction error vanishes, see Sect. 4.

With regards to a neural network based function approximation, this procedure represents an exceptional case. Ordinarily, it is intended to approximate a discrete data set by a smooth function that reflects the essential properties of the data but does not reproduce subordinated, unimportant, disturbing, or even spurious data fluctuations. The minimum requirements according to the universal function approximation theorem then depend on the desired degree of smoothness of the approximation function. According to the statistical estimation theory, each particular data point is important and contributes to the result of an estimation. Thus, the envisaged simulation must not be performed with smoothed process realizations, which would lead to erroneous results, for example, in safety assessment. Furthermore, a separation of noise as in traditional models cannot be realized since the neural network procedure is model-free. That is, the process realizations must be generated including noise. Hence, the minimum requirements according to the universal function approximation theorem are determined by the requirement that the network must be able to reproduce the observed data record from the process with no deviations. This is checked when training the network. The number of neurons in the hidden layers must be increased until the network prediction error can be brought to zero by adjusting the weights and biases. The problem of overfitting, which is significant in function approximation, is thus not relevant in network based process simulation.

A neural network with zero prediction error can reproduce the complete observed data record. For each particular sequence of process values at the time steps t_{n-r} to t_{n-1}, the network generates the process value at t_n. If the time steps t_{n-r} to t_{n-1} represent the last sequence of the observed data record, the network predicts the first unknown process value at t_n. A progressive prediction started at the end of the observed data record then yields a prognosis for the future behavior of a single process realization, see Beer and Spanos (2004).

3.3 Simulation initialization

A Monte Carlo simulation of random processes requires the generation of a sufficiently high number of process realizations running over a proper period of time. For that purpose, the algorithm for generating process realizations must be capable of numerically reproducing the process characteristics. According to the statistical estimation theory, all information which may be gathered about the process is contained in the data. Once the neural network is trained, this information is stored in the adjustment of the weights and biases. That is, the numerical algorithm for generating process realizations is defined. A variety of process realizations can only be produced by starting the generation with different initial conditions.

For initializing the neural network based process generation the starting input vector for the tapped delay line memory must be defined. These are generated randomly by being drawn from a smoothed empirical first-order distribution of the observed process record.

4 NETWORK TRAINING

The network training is accomplished based on the standard method of error back-propagation, see e.g. Haykin (1999). An observed data record from the process is used as training data. The aim is to adjust the free values of the neural network (weights and biases) so that the network is capable of reproducing the training data with a sufficient precision. That is, for each sequence of process values $z(t_{n-r})$, ..., $z(t_{n-1})$ the network is intended to generate a prognosis p_n for the subsequent process value $z(t_n)$ with a minimum prediction error

$$e = z(t_n) - p_n \ , \tag{10}$$

and error energy

$$E = \frac{1}{2} \cdot e^2 \ . \tag{11}$$

Specifically,

$$\sum_{n=r+1}^{N} \left(z(t_n) - p_n\right)^2 \ \Rightarrow \ \text{MIN} \ , \tag{12}$$

with N being the length of the observed process record (training data) and r denoting the order of the tapped delay line memory. Mathematically, Eq. (10) represents the objective function of an optimization problem in a multi-dimensional space in which the weights and biases are the design parameters. The search for the optimum adjustment of the weights and biases is realized with the aid of a gradient descend method operating with a generalized delta rule. For each predicted process value the prediction error of the neural network is retraced through the complete network (back-propagation) to compute changes of the weights and biases. This is done iteratively, as described subsequently, until the prediction error approaches the global minimum.

The training starts with randomly initialized weights and biases uniformly distributed in the interval [–1, +1]. This ensures that the training data meet the effective part of the activation functions of the neurons to a high percentage, see Sect. 2.2.2. Initializations using narrower or wider intervals or random specifications of the weights and biases according to other distributions did not lead to improved results.

One sequence of r+1 successive process values $z(t_{n-r})$, ..., $z(t_{n-1})$, $z(t_n)$ is randomly selected from the training data with the aid of a discrete uniform distribution over the $N-r$ possible choices. Then, the error signal e(q) in the current iteration step q is determined with Eq. (10) and is used to compute the local gradients $\partial E/\partial w_{kj}$ in the weight space, proportional to which the weights and biases are to be changed. Let $v_{j^{(L)}}(q)$ be the argument of the activation function $\varphi_{j^{(L)}}$ in neuron $j^{(L)}$ of layer L and $y_{j^{(L-1)}}(q)$ denote the neuron output of neuron $j^{(L-1)}$ in the previous layer L – 1. The new weights for the next iteration step are then

$$\begin{aligned} w_{j^{(L)}j^{(L-1)}}(q+1) = \ & w_{j^{(L)}j^{(L-1)}}(q) \\ & + \alpha \cdot [w_{j^{(L)}j^{(L-1)}}(q-1)] \\ & + \eta \cdot \delta_{j^{(L)}}(q) \cdot y_{j^{(L-1)}}(q) \ , \end{aligned} \tag{13}$$

with

$$\delta_{j^{(L)}}(q) = e(q) \cdot \varphi'_{j^{(L)}}\left(v_{j^{(L)}}(q)\right) \tag{14}$$

for the neuron $j^{(L)} = 1$ in the output layer, and

$$\delta_{j^{(L)}}(q) = \varphi'_{j^{(L)}}\left(v_{j^{(L)}}(q)\right) \cdot \sum_{j^{(L+1)}=1}^{J^{(L+1)}} \delta_{j^{(L+1)}}(q) \cdot w_{j^{(L+1)}j^{(L)}}(q) \tag{15}$$

for the neurons $j^{(L)}$ in layer L. Hereby, the biases $b_{j^{(L)}}$ are treated as weights $w_{j^{(L)}j^{(L-1)}}$ for constant input signals $y_j^{(L-1)} = +1$ from the previous layer L – 1. The parameters α and η are introduced to control the numerical behavior of the iteration. Whereas the learning rate $\eta > 0$ determines the degree with which the actual error gradients effect the weight change, the momentum factor $\alpha \in [0, 1)$ acts as a delay parameter in the weight adjustment.

When the weight adjustment in iteration step q is completed, the next sequence of r+1 successive process values $z(t_{n-r})$, ..., $z(t_{n-1})$, $z(t_n)$ is randomly selected to proceed with the weight adjustment in iteration step q + 1. This procedure of iteratively adjusting the weights and biases is referred to as sequential training mode, which possesses the advantage of being stochastic in nature. This induces a good performance in the search for the global minimum of the objective function.

To assess the network training results, various termination criteria may be defined. For example, the training may be terminated when the relative weight change or the prediction error becomes sufficiently small. Herein, the prediction error is selected as termination criterion as this is intended to be zero for process simulation. For an optimal network configuration, the prediction error approaches zero asymptotically with progressive training. The determination of a particular appropriate structure of the network, however, is problem dependent. The number of layers, neurons in each layer, and the memory order r have to be specified iteratively, too. Depending on the training result and the quality of the network prediction of the trained neural network, the suitability of the particular network structure may be assessed.

5 NUMERICAL EXAMPLE

The capability and the features of the neural network based process simulation are shown in a numerical example. A numerically generated time series of a random process with defined properties is taken as the basis. This enables an evaluation of the network simulation result not only with respect to the statistical properties of the input data record, but also with respect to the actual process characteristics.

An ARMA model of order 25 is applied to generate a long record of a stationary ergodic Gaussian process with the Kanai-Tajimi-like power spectrum

$$S(\omega) = \frac{1 + \alpha \cdot \omega^2}{\left(\omega_n^2 - \omega^2\right)^2 + \left(2 \cdot \zeta \cdot \omega \cdot \omega_n\right)^2} \,. \tag{16}$$

The parameters are selected as $\alpha = 5$, $\zeta = 0.25$, and $\omega_n = 10$ rad/s^{-1}. The curve of this spectrum is plotted as a normalized target spectrum in Figs. 7 and 8. The step width for the process values is $\Delta t = 0.0628$ s.

For the neural network based simulation a subsequence of only 50 successive values from the generated time series is taken as the basis. An appropriate network architecture is found with a tapped delay line memory of order 12 and two hidden layers with 13 and 7 neurons, respectively. This network possesses a total of 275 free values (weights and biases), which are adjusted iteratively via error back-propagation during the network training. The development of the prediction error with proceeding network training is shown in Fig. 5. After 80,000 iteration steps the error remains under 10^{-6}.

The trained network is applied to generate long process realizations initialized by randomly generated "seed" vectors. The quality of these realizations is evaluated by statistically comparing with the training record as well as with the actual, underlying, process.

Mean value and standard deviation show a good agreement; see Table 1. Also, the empirical first-order distribution functions run close to each other with a smooth curve from the network prediction, see Fig. 6. The deviation from the underlying distribution at the left end of the curves results from a process value concentration in the training record, which is reflected in the network prediction. The figure shows the complete value range of the network prediction. Homogeneity tests and goodness-of-fit tests yield small rejection probabilities for the H_0 hypothesis. With regard to the training data the distribution of the network prediction is rejected with probability $P_{KS} = 0.024$ according to the Kolmogorov-Smirnov test and with $P_{\chi^2} = 0.028$ according to the chi-squared test. The underlying Gaussian distribution does not lead to significantly better results with the rejection probabilities $P_{KS} = 0.000$ and $P_{\chi^2} = 0.217$.

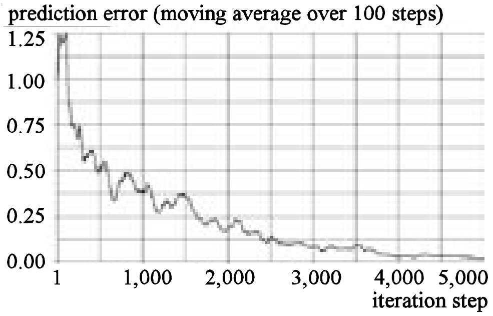

Figure 5. Prediction error decrease during network training

Table 1. Comparison of mean values and standard deviations

	mean value μ	standard deviation σ
network prediction	– 0.007	0.176
training record	– 0.001	0.169
underlying process	0.000	0.174

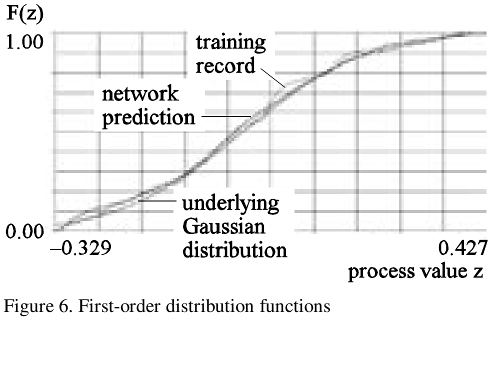

Figure 6. First-order distribution functions

Further, spectral density estimations are generated from the training record and from the network prediction via averaged periodograms over several subsequences of 10 successive time steps. Again, the curves for the network prediction and for the training record show a good agreement, see Fig. 7. A comparison with the target spectrum shows reasonable deviations owing to the short training record.

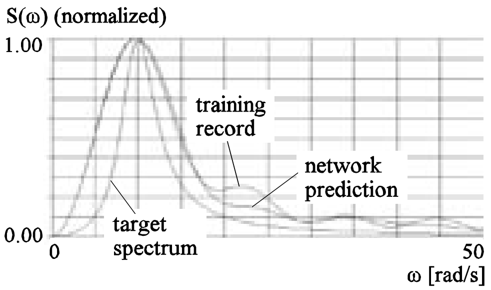

Figure 7. Spectral density estimations – one network based simulation

In an expanded study, ten different, non-overlapping training records of 50 values each are taken from the original process record from the initial ARMA application. For each of these training records a neural network based simulation is carried out. As above, spectral density estimations are generated for each training record and each prognosis record via averaged periodograms over subsequences of 25 process values. Then, the obtained ten spectral density estimations for the training data and for the prognosis data are again averaged, see Fig. 8. The resulting, averaged spectral density curves document an adequate quality of the network with regard to the training data. Moreover, the deviations from the target spectrum are now much smaller. This indicates that the neural network has recognized the essential characteristics of the underlying process and that the network prognosis converges towards the exact solution in mean sense.

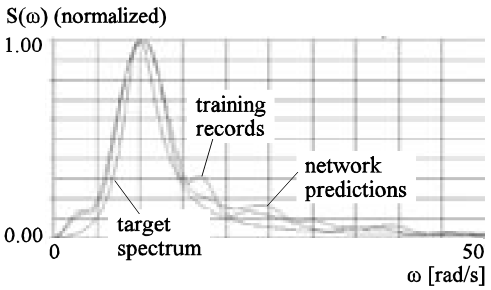

Figure 8. Spectral density estimations – averaged over ten network based simulations

6 CONCLUSIONS

Neural networks may represent a viable approach for simulating stochastic processes in various engineering fields. They are particularly powerful in cases in which various properties of the process cannot be identified or specified precisely. Contrary to traditional methods, a model specification is not required. Moreover, neural networks possess a higher intrinsic complexity than traditional simulation models. Their numerical efficiency is thus lower for simple problems but can be expected as being higher for complicated ones.

Further developments of the presented procedure may focus on extensions for applicability to non-stationary and multivariate problems.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the Alexander von Humboldt-Foundation (AvH) and of the Office of Naval Research (ONR), USA.

REFERENCES

Beer, M. & Spanos, P.D. 2004. A Neural Network Approach for Representing Realizations of Random Processes. *Proceedings 9th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability.* Albuquerque, NM, July 2004, CD-ROM, Doc. 04_104.

Bishop, Ch. M. 1995. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.

Giles, C.L., Lawrence, S. & Tsoi, A.C. 2001. Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Inference. *Machine Learning* Vol. 44: 161–183.

Haykin, S. 1999. *Neural Networks: a comprehensive foundation*. Upper Saddle River, NJ: Prentice Hall.

Lee, S.-C. 2003. Prediction of concrete strength using artificial neural networks. *Engineering Structures* Vol. 25: 849–857.

Li, H. & Kozma, R. 2003. A Dynamical Neural Network Method for Time Series Prediction Using the KIII Model. *Proceedings of the International Joint Conference on Neural Networks IJCNN*. Portland, OR, 2003. IEEE press: 347–352.

More, A. & Deo, M.C. 2003. Forecasting wind with neural networks. *Marine Structures* Vol. 16: 35–49.

Schuëller, G.I. & Spanos, P.D. (eds) 2001. *Monte Carlo simulation: proceedings of the International Conference on Monte Carlo Simulation*. Monaco, June 18–21, 2000. Lisse, Exton, PA: A.A. Balkema.