Structural Engineering and Mechanics, Vol. 32, No. 1 (2009) 71-94
DOI: http://dx.doi.org/10.12989/sem.2009.32.1.071

A neural network approach for simulating stationary stochastic processes

Michael Beer†

Department of Civil Engineering, National University of Singapore, Singapore

Pol D. Spanos

Ryon Endowed Chair in Engineering, Rice University, Houston TX, USA

(Received June 11, 2008 Accepted October, 8, 2008)

Abstract. In this paper a procedure for Monte Carlo simulation of univariate stationary stochastic processes with the aid of neural networks is presented. Neural networks operate model-free and, thus, circumvent the need of specifying a priori statistical properties of the process, as needed traditionally. This is particularly advantageous when only limited data are available. A neural network can capture the "pattern" of a short observed time series. Afterwards, it can directly generate stochastic process realizations which capture the properties of the underlying data. In the present study a simple feed-forward network with focused time-memory is utilized. The proposed procedure is demonstrated by examples of Monte Carlo simulation, by synthesis of future values of an initially short single process record.

Keywords: Monte Carlo simulation; neural networks; stochastic processes.

1. Introduction and Motivation

Numerical simulations of stochastic processes have become quite important in many engineering problems. Monte Carlo approaches are particularly suitable tools for these simulation purposes. Indeed, their usefulness in diverse applications has been well established over a period of several decades; see Schuëller *et al.* (2001). The major focus of current research in this field is the improvement of the associated numerical efficiency, which is of great importance in stochastic mechanics themes (Schuëller 2001, Ghanem and Spanos 2003, Du *et al.* 2005, Spanos *et al.* (2007). This is particularly true when analyzing large dimension systems (Schenk *et al.* 2005, Schenk and Schuëller 2005).

Mathematically, a stochastic process is an indexed collection of random variables, which is for engineering mechanics problems commonly considered in a real-valued environment. Given a probability space $[\Omega, \mathfrak{S}, P]$ with random events $\theta \in \Omega$, $Z(t, \theta)$ with $t \in D$ is understood as a stochastic process defined over a real-valued domain D; see, for example, (Vanmarcke 1983).

† Corresponding author, E-mail: cvebm@nus.edu.sg

Generally, both t and Z can be multi-dimensional corresponding to multi-dimensional and multivariate processes, respectively. The process parameters t are usually temporal or spatial coordinates, and the random process values Z may represent various physical or mechanical quantities such as material strength or load intensity. Herein, attention is limited to one-dimensional univariate processes; t, $Z \in \mathbb{R}$. Each process realization then represents an ordered set of real numbers $\{z_1, z_2, z_3, \ldots\}$.

For the treatment of stochastic processes within the simulation procedures models in the form of probability distributions and spectral density functions are specified. A variety of process models are available for diverse physical themes (Vanmarcke 1983). Also, various methods for process simulation have been developed (Spanos and Zeldin 1998), from which the most appropriate one can be selected in a particular case. Among the available numerical models for representing stochastic processes, spectral approaches appear as the most popular ones. In particular, the Karhunen-Loéve expansion and the Polynomial Chaos expansion have attracted considerable attention (Ghanem and Spanos 2003, Schenk and Schuëller 2005, Iwan and Jensen 1993, Phoon *et al.* 2002, Field and Grigoriu 2004, Sakamoto and Ghanem 2002).

An essential condition for obtaining realistic results from a simulation is the availability of a statistically validated process description. The specification of the probabilistic model, thus, plays a significant role; see (Schuëller 2003). For this reason, further research effort is devoted to specifying proper models (Sakamoto and Ghanem 2002, Schuëller 2003, Yuen *et al.* 2002, Grigoriu *et al.* 2003, Cai and Wu 2004) to capture the underlying process characteristics. This includes the analysis of the covariance function, of the spectral density function, of the distribution of the noise, of non-stationarities, and of bounds for the process value. If sufficient information about the underlying physics of a stochastic process is available, a numerical model for the process may be selected from the variety of choices and used in the particular situation. However, difficulties may be experienced in the case of limited information. If the data bank comprises only a single short process realization, 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 in these problematic cases to analyze observed data records $\{z_1, z_2, z_3, \ldots\}$ as realizations of a stochastic process and to simulate, subsequently, the stochastic process. 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 (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 $\underline{\mathbf{X}} \subset \mathbb{R}^n$ and arbitrary $\varepsilon > 0$, there exists a neural network with the output $f(\underline{x})$ such that

$$\sup_{\underline{x} \in \underline{\mathbf{X}}} \left| f(\underline{x}) - g(\underline{x}) \right| < \varepsilon \tag{1}$$

Each realization of a stochastic process which can be understood as such a continuous function $g(\underline{x})$ can then be approximated by a neural network. Conceptually, this is related to traditional methods (Spanos and Zeldin 1998) 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 Haykin (1999) and Bishop (1995), an attempt is made herein to use their appealing features for addressing problems in stochastic process simulation. In this context, it is assumed that only one short process realization $\{z_1, z_2, z_3, \ldots\}$ has been observed, and no further information is available.

This endeavor is made in compliance with the basic consensus that a neural network may represent a powerful tool whenever a model specification is problematic. From this perspective, extensive research efforts have been undertaken in the development and application of neural networks in various directions and in diverse fields.

Particular activities in engineering can be observed in environmental fields. This is driven by a combination of problems in model specification and a recently increasing demand for forecast in view of natural hazards. Quite popular are applications is hydrology. An overview regarding the basis of the recent developments in this field and beyond is provided in Govindaraju (2000) in terms of technical network concepts, and in Govindaraju (2000) regarding applications. Generally, the neural networks are applied to analyze observed time series to generate conclusions and predictions. The progress in the past decade shows an increase of stochastic components incorporated in the network solutions. In Furundzic (1998) a network-based variable selection method is proposed as an alternative to a traditional regression analysis. Rainfall data were analyzed under consideration of noise to identify significant input variables for a forecast. It was found that the network possesses advantages over traditional regression techniques in the cases considered. A superiority of neural networks with respect to traditional models was also shown in Toth *et al.* (2000) for the short-term rainfall forecast. This investigation included a comparison with linear ARMA models such as the ones used in stochastic mechanics. Most recently, stochastic neural networks have attracted increasing attention. An application for the prediction of droughts was reported in Ochoa-Rivera (2008). A stochastic multilayer perceptron network was used to build a non-linear multivariate prediction model. This network has shown a better performance than a respective second order AR model.

The trend towards neural networks with stochastic components is also reflected in further application fields. Financial market developments in form of time series are investigated with different network concepts. These include, for example, recurrent networks Giles *et al.* (2001), dynamical network methods Li and Kozma (2003), and support vector machines Cao and Tay (2003). The forecasting horizon of these applications is rather short. The challenging issue here is nonstationarity of the time series. As an alternative to stochastic concepts, chaotic reproduction and forecasting schemes were considered (Shi and Han 2007). An evaluation of geological data of several types was used in Cimino *et al.* (1999) to compare the proposed neural network approach with an ARIMA model for stationary time series prediction. It was reported that the neural network approach can be used in a more general context than the ARIMA model and can distinguish between deterministic, chaotic and random patterns. A neural network concept designated to long-term predictions of traffic flow is proposed in Jiang and Adeli (2005). A recurrent wavelet neural network with dynamic and time-delay features was developed for this purpose.

Besides neural network concepts for specific applications, a variety of more general developments exist. For example, the treatment of seasonal effects in time series is addressed in Zhang and Kline (2007). It was found that simpler models usually exhibit a better performance than complex models. In Qi and Zhang (2008) a network-based modeling of the trend in time series was investigated. A

neural network trained with a stochastic process was proposed in Sanchez and Sarabia (2002) for statistical hypothesis testing. Properties such as convergence and stability of the network-based test were verified successfully. Universal learning networks were examined in Hirasawa *et al.* (2006) with respect to the propagation and control of stochastic signals. The goal was to control and to identify non-linear dynamic systems interfered by noise. The method was developed in view of investigating large-scale complex systems. The neural network-based representation and generation of stochastic processes was examined in detail in Turchetti *et al.* (1998) and Belli *et al.* (1999). Stochastic neural networks were selected as the basis and formulated as a generalization of deterministic neural networks. Non-stationary processes were approximated in the mean square sense. In the core of the method a canonical representation is applied to express the stochastic process in terms of deterministic functions combined with stochastic noise. This corresponds to a spectral representation as discussed in Ghanem and Spanos (2003). The deterministic functions were approximated with a neural network, and the noise was generated in stochastic neurons. A further development of this concept was recently reported in Turchetti *et al.* (2008). The extension mainly concerned the approximation of nonlinear input-output random transformations. The Karhunen-Loéve expansion was applied, and the process generation was realized based on an approximation of the eigenfunctions of the covariance kernel.

In the present work the idea is pursued to generate stationary stochastic processes without any model specification. The neural network is even supposed to identify and to reproduce the noise of the process.

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 proper 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 of information transferring links between the neurons – the synaptic connections; see Fig. 2. 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 must be determined in context with the application considered.

If the generation of process realizations is treated as a problem of function approximation, use can be made of some experience from that field. Following this experience 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 $\varphi(.)$ 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; see Fig. 1. 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

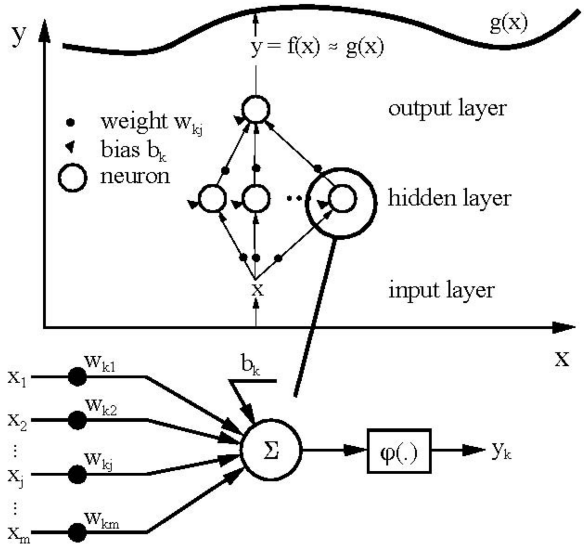

Fig. 1 Multi-layer perceptron network for function approximation and general construct of a neuron

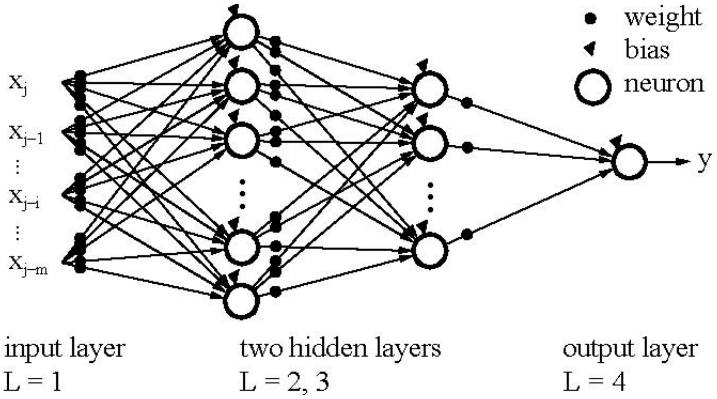

Fig. 2 Focused time lagged feed-forward network

conditions. The summing junction generates the activation potential v_k as input argument for the subsequently called activation function $\varphi(.)$. This yields the output signal y_k generated by neuron k. Specifically

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

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

$$\varphi(v) = (1 + \exp(-v))^{-1} \tag{3}$$

is selected in view of numerical efficiency. This function possesses the advantage that its derivatives $\varphi'(v)$, which must be computed millions of times during the network training, can be obtained easily

when the functional values $\varphi(v)$ are already known. Specifically

$$\varphi'(v) = \varphi(v) \cdot (1 - \varphi(v)) \tag{4}$$

This reduces the computational cost of the training procedure drastically; see Eqs. (36) and (37). A selection of other, commonly used, activation functions with the required properties is basically of minor influence as these have a similar shape. Differences can be compensated mostly by a scaling of the activation functions with respect to the activation potential v, namely by introducing a factor β to build $\varphi(\beta \cdot v)$. Remaining differences can then be accounted for by the weight adjustment and a modified network layout. For the hyperbolic tangent activation function

$$\varphi_{\tanh}(v) = \tanh(v) \tag{5}$$

which is frequently proposed as an alternative to the logistic sigmoid function, it can even be shown that it is equivalent to the logistic sigmoid function. Normalizing the function Eq. (5) to $\varphi(v) \in (0, 1)$ and introducing a scaling factor β for the activation potential lead to

$$\begin{aligned} \varphi_{\tanh}(v) &= 0.5 + 0.5 \cdot \tanh(\beta \cdot v) = 0.5 + 0.5\left(1 - \frac{2}{\exp(2\beta \cdot v) + 1}\right) \\ &= \frac{\exp(2\beta \cdot v)}{\exp(2\beta \cdot v) + 1} = \frac{1}{1 + \exp(-2\beta \cdot v)} \end{aligned} \tag{6}$$

which is equal to the expression Eq. (4) if $\beta = 0.5$.

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 incorporating 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 x_j, ..., x_{j-m} 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. That is, dependencies between consecutive process values Z, which are characteristic for the process, can be identified. This includes components of both noise and covariance features.

Long-term memory is built by adjusting the synaptic weights w_{kj} and biases b_k of the network. This is realized in the network training, which may be understood as “observation” of the process pattern for one selected time sequence. In each step of the training, one more piece of information is embedded into the weights and biases. Once the training of the network is completed, the entire

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. Construed in the context of stochastic processes, the local process features concern the noise of the process and quite high frequencies. Global process features are the medium and lower frequencies. The first hidden layer primarily gathers information about local data fluctuations, whereas the second one is designated as assimilating global process characteristics. The global characteristics become apparent to the network by assembling the local information via the synaptic connections between the hidden layers. This enables the neural network to analyze patterns that evolve over time.

The signal flow through a network of the selected kind can be summarized as a nested sum of weighted and combined signals. Let $J^{(L)}$ be the number of neurons $j^{(L)}$ in layer (L), $w_{j^{(L)}j^{(L-1)}}$ represent the weight for the signal from neuron $j^{(L-1)}$ in layer $(L-1)$ to neuron $j^{(L)}$ in layer (L), and $b_{j^{(L)}}$ denote the bias of neuron $j^{(L)}$ in layer (L). Then, the signal flow through the neural network in Fig. 2 is given by

$$\begin{aligned} y &= \varphi\left(\sum_{j^{(3)}=1}^{J^{(3)}} [w_{j^{(4)}j^{(3)}} \cdot \varphi(v_{j^{(3)}})] + b_{j^{(4)}} \right) \\ v_{j^{(3)}} &= \sum_{j^{(2)}=1}^{J^{(2)}} \left[w_{j^{(3)}j^{(2)}} \cdot \varphi\left(\sum_{j^{(1)}=1}^{J^{(1)}} [w_{j^{(2)}j^{(1)}} \cdot x_j] + b_{j^{(2)}} \right) \right] + b_{j^{(3)}} \end{aligned} \tag{7}$$

In Eq. (7), the numbers $j^{(L)}$ for the neurons are independent running indices for the separate layers (L).

2.2.2 Signal conditioning

The neural network must be adequately sensitive to fluctuations in the input signals. That is, a proper change in the values of the input signals to a neuron should generally lead to a noticeable difference in the neuron output. Let $\Delta y_{j^{(L-1)}}$ be the differences in the output from the neurons $j^{(L-1)}$ in layer $(L-1)$. Then,

$$\Delta v_{j^{(L)}} = \sum_{j^{(L-1)}=1}^{J^{(L-1)}} w_{j^{(L)}j^{(L-1)}} \cdot \Delta y_{j^{(L-1)}} + b_{j^{(L)}} \tag{8}$$

is the resulting change in the activation potential $v_{j^{(L)}}$ of neuron $j^{(L)}$ in the subsequent layer (L). This leads to the difference

$$\Delta y_{j^{(L)}} = \varphi(v_{j^{(L)}} + \Delta v_{j^{(L)}}) - \varphi(\Delta v_{j^{(L)}}) \tag{9}$$

in the output $y_{j^{(L)}}$ of neuron $j^{(L)}$, for which the property

$$\Delta v_{j^{(L)}} - \Delta y_{j^{(L)}} \approx \Delta v_{j^{(L)}} \tag{10}$$

should be satisfied regularly. For the neurons of the first hidden layer $(L = 2)$, Eq. (8) becomes

$$\Delta v_{j^{(2)}} = \sum_{j^{(1)}=1}^{J^{(1)}} w_{j^{(2)}j^{(1)}} \cdot \Delta x_j + b_{j^{(2)}} \tag{11}$$

with Δx_j being changes in the input signals x_j received by the input layer ($L = 1$), see Fig. 2. The property in Eq. (10) is ensured if the activation potentials v of the neurons primarily meet the effective part V_{eff} of the activation function $\varphi(v)$. For the logistic sigmoid function from Eq. (3) this effective part may be defined, for example, as the interval

$$V_{\text{eff}} = [v_{\min}, v_{\max}] \tag{12}$$

with

$$\begin{aligned} v_{\min} &= -\ln\left(\frac{1}{\varepsilon} - 1\right) \\ v_{\max} &= -v_{\min} \end{aligned} \tag{13}$$

The parameter ε is a prescribed minimum distance of the activation function $\varphi(v)$ from its limits zero and unity

$$\varepsilon \leq \varphi(v) \leq 1 - \varepsilon \tag{14}$$

see Fig. 3. If ε is prescribed with $\varepsilon = 0.01$, the effective part V_{eff} is obtained as

$$V_{\text{eff}} = [-\ln(99), \ln(99)] = [-4.595, +4.595] \tag{15}$$

Activation potentials v from this interval lead to neuron outputs

$$y(v \in V_{\text{eff}}) \in [0.01, 0.99] \tag{16}$$

Fluctuations of input signals that cause values of activation potentials far outside the effective part do virtually not affect the further signal processing. In this manner, an impact from extreme statistical outliers is eliminated automatically. Such outliers may be caused, for instance, by severe malfunctions of a measuring device or by specimens with severe defects. The associated extreme values do not represent physical properties and represent conditions which do not justify further consideration. Consequently, these outliers must be eliminated prior to a statistical evaluation. This practice is also adopted in other application fields; see, for example, Jiang and Adeli (2005).

According to this, two quantities of influence can be controlled to achieve a proper sensitivity of the network: first, the weights and biases, and second, the input signals x_j.

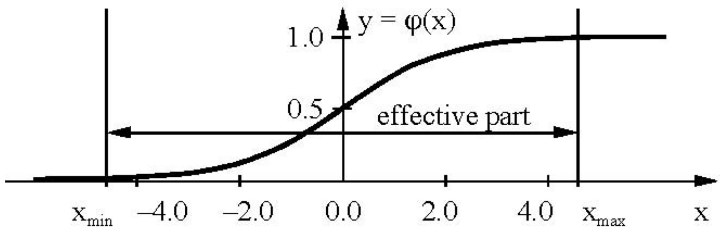

Fig. 3 Effective part of the activation function

I) Weights and biases. The final values of the weights and biases of a network are primarily determined by the particular underlying problem. The network is adjusted to this problem during the network training; see Section 4. The weights and biases represent the network perception of the observed data in form of a process realization $\{z_1, z_2, z_3, \ldots\}$. For a feasible and effective training, however, the weights w and biases b must be properly initialized. With respect to Eqs. (3) and (4) activation potentials v that are close to zero are most effective; see Fig. 3. Further, the first derivative of the selected activation function, which appears as a factor in the weight adjustment in the network training, is symmetric to $v = 0$; see Eq. (4). It is thus reasonable to limit the initial values w and b to some interval A symmetric to zero. Specifically,

$$A = [-a, +a], \quad a = 1 \tag{17}$$

has been found adequate. Moreover, each synaptic connection should contribute its own individual portion $w \cdot y$ with $y \in (0, 1)$ (see Eqs. (2) and (3)) to the combined signal v. These contributing signals $w \cdot y$ are supposed to be non-zero to prevent the synaptic connections from inactivity, and they should be different from one another to ensure individuality. This motivates the initialization of the weights and biases with the aid of a continuous uniform distribution over the interval A

$$(X \sim U(A)) \rightarrow \{w_{j^{(L)}j^{(L-1)}}, b_{j^{(L)}}\} \tag{18}$$

Initializations using narrower or wider intervals or random specifications of the weights and biases according to other distributions did not lead to improved results.

II) Input signals. The input signals x_j must lie within a reasonable value range to be processed properly. The raw data are thus preconditioned with the aid of a 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 V_{eff} (see Eq. (15)) of the activation function. Though this is reasonable for analyzing data with a bounded value range, for example, for controlling purposes in automation, it may cause problems when processing statistical data. If the data bank consists of a small random sample, as considered herein, there exists a non-negligible probability that further sample elements may lie moderately 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, these input signals, however, lead to activation potentials v that lie outside the effective part V_{eff} and, consequently, may be virtually effectless. Thus, a common normalization 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

$$x_j = \frac{z_j - \bar{z}}{\sigma_Z} \forall j \tag{19}$$

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 V_{eff} of the activation function in the neurons of a network is met

Table 1 Probability P_{eff} for meeting the effective part of the activation function

Number of simultaneously considered input signals x_j	Probability $P_{\text{eff}}(j^{(L=2)})$ for neuron $j^{(L=2)}$
1	0.99999777
6	0.995743
12	0.9708
24	0.8891

$$P_{\text{eff}} = P(v \in V_{\text{eff}}) \tag{20}$$

The following example gives representative values of these probabilities. Again, the effective part of the activation function is defined by Eq. (15). According to Eqs. (17) and (18), the weights and biases are assumed to be uniformly distributed in [–1, +1]. The probability P_{eff} $(j^{(L=2)})$ for a neuron $j^{(L=2)}$ of the first hidden layer ($L = 2$) is estimated roughly by evaluating the summing junction in Eq. (2) with standard normal random variables for numerically producing the x_j. Neurons $j^{(L>2)}$ in the subsequent layers ($L > 2$) are only very slightly affected by the conditioning of the input signals and are thus not considered here. Table 1 shows the result of this investigation in dependence on the order of the tapped delay line memory, which specifies the number of input values simultaneously fed to the network. These results indicate that the network can process the input data records properly.

According to the definition of the activation function $\varphi(.)$, see Section 2.1, the neuron output and hence the neural network output are restricted to the value range $y \in (0,1)$. The network output must thus be back-transformed into the original scale of the particular problem. For the generation of process realizations, a back-transformation of the network output y into the scale of the original process values z_j (observed process realization) is required. That is, the back-transformation is inverse with respect to 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. The inverse function of the standard normal distribution function is applied to the output y, $\Phi^{SN^{-1}}(y)$. The result is then introduced in a rewritten form of Eq. (19) to scale the generated values based on the original data record. That is, the generated process values p are determined using the equation

$$p = \Phi^{SN^{-1}}(y) \cdot \sigma_Z + \bar{z} \tag{21}$$

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

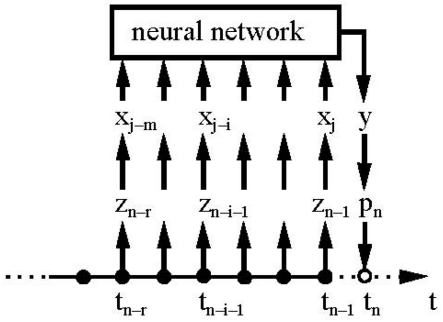

Fig. 4 Process value prediction

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}. That is, the tapped delay line memory of the network is of order $r = J^{(1)}$. Specifically,

$$\begin{array}{lll} z_{n-1} & = z(t_{n-1}) & \rightarrow x_j \\ \vdots & \vdots & \vdots \\ z_{n-i-1} & = z(t_{n-i-1}) & \rightarrow x_{j-i} \\ \vdots & \vdots & \vdots \\ z_{n-r} & = z(t_{n-r}) & \rightarrow x_{j-m} \end{array} \tag{22}$$

The single network output y then yields the predicted value p_n of the process at the present time t_n

$$(x_{j-m}, x_{j-i}, \ldots, x_j) \rightarrow y \rightarrow p_n \tag{23}$$

An elementary 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, $(z_{n-r}, \ldots, z_{n-1}) = (z_{N-r+1}, \ldots, z_N) \rightarrow (x_{j-m}, \ldots, x_j)$, in which N is the size of the observed process realization. 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 $p_n = p_{N+1}$ for the time step $t_n = t_{N+1}$. And the subsequent prediction p_{n+1} for t_{n+1} is made not until the observation z_n at t_n is available. The time horizon of this network prediction is only one step. The accuracy of one-step predictions is quite high. Applications may be found, for example, in the forecasting of financial market developments, see Giles *et al.* (2001) and Li and Kozma (2003). Also, attempts to exploit the capability of this procedure for short-term process prediction in engineering have been made. For instance, in More and Deo (2003) a neural network model is presented 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 capabilities of a neural network by far. To extend 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. It has been

applied, for example, for the prediction of the development of the concrete strength over the first 28 days; see Lee (2003). The second approach involves a recursive application of the network from Fig. 4. The network prediction for the process value p_n at t_n is used as an input signal z_n for predicting the process value p_{n+1} at t_{n+1}

$$z_n = p_n(z_{n-r}, \ldots, z_{n-1}), \quad n = r+1, r+2, r+3, \ldots \tag{24}$$

That is, the network predictions are fed back to the input layer instead of using observed process values. A step-by-step progression 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. It is assumed - under the awareness of overfitting - that the network predictions p_n coincide with the observations z_n for all time steps t_n

$$z_n - p_n = 0 \forall n \tag{25}$$

This justifies feeding the network predictions back to the input layer of a feed-forward network according to Eq. (24). The feed-back of the network output is not activated until the network training is finished. The training of the network in Fig. 4 is performed until the network prediction error is eliminated, see Sect. 4. An application of a recurrent network (Haykin 1999), which provides a feed-back of the network output in a network-internal fashion, is not pursued. This would decrease the transparency of the network and complicate the training procedure.

With regards to a neural network-based function approximation, the selected 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, however, each particular data point is important and contributes to the result of an estimation. Thus, the considered simulation must not be performed with smoothed process realizations, which would lead to erroneous results, for example, in reliability assessment. Also, the option of using smooth network outputs in combination with a separate noise generator is not pursued herein. This approach would follow the concept of traditional models such as ARMA and does not open new perspectives. A prescription of a degree of smoothness by some predefinition of network parameters, such as the number of neurons, would, further, introduce some subjectivity and would require a model for reproducing the noise of the process. It would, further, require the specification of a model for noise generation. This is, however, supposed to be circumvented with the present approach. Herein, it is assumed that no background information is available about the underlying process, which is needed for those external specifications. Consequently, a pure neural network procedure, which operates model-free, is considered. It is intended to generate the process realizations including noise, directly.

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 process realization with no deviations. This is checked when training the network. The number of neurons in the hidden layers is 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 the proposed network-based process simulation.

A neural network with a prediction error of zero can reproduce the complete observed process realization except the first r values. The first r values are needed as initial network input. For each particular sequence of process values $(z_{n-r}, \ldots, z_{n-1})$ at the time steps t_{n-r} to t_{n-1}, the network generates the process value $z_n = p_n$ at t_n. The observed process realization is so reproduced as $z_{r+1} = p_{r+1}$, ..., $z_N = p_N$, with N being its length. For the last sequence of the observed process realization $(z_{N-r+1}, \ldots, z_N)$ at the time steps t_{N-r+1} to t_N as input, the network predicts the first unknown process value $z_{N+1} = p_{N+1}$ at t_{N+1}. A progressive prediction

$$z_n = p_n(z_{n-r}, \ldots, z_{n-1}), \quad n = N+1, N+2, N+3, \ldots \tag{26}$$

started at the end of the observed process realization then yields a prognosis for the future behavior of a single process realization, see Section 5.2.

3.3 Monte Carlo simulation

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 this 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 $(z_1, \ldots, z_r)$ for the tapped delay line memory must be defined. For this purpose, the empirical first-order distribution function $F^{(e)}(z)$ is built from the observed process realization. Specifically

$$F^{(e)}(z) = \frac{1}{N} \cdot \sum_{n=1}^{N} I(z_n) \tag{27}$$

with the indicator function

$$I(z_n) = \begin{cases} 1 & \text{if} \quad z_n < z \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

The function $F^{(e)}(z)$ is then interpolated linearly between the points $(z_n, F^{(e)}(z_n))$ to obtain $F^{(\bar{e})}(z)$ without jumps. The starting input vector for each simulation of a process realization is then drawn from $F^{(\bar{e})}(z)$

$$(Z \sim F^{(\bar{e})}(z)) \rightarrow z_1, \ldots, z_r \tag{29}$$

4. Network training

The network training is accomplished based on the standard method of error back-propagation, see, e.g., Haykin (1999). An observed process realization 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 $(z_{n-r}, \ldots, z_{n-1})$ the network is intended to generate a prognosis p_n for the subsequent process value z_n with a minimum prediction error

$$e_{p_n} = z_n - p_n \Rightarrow \text{Min } \forall n | r < n \leq N \tag{30}$$

To be independent of the absolute scale of process values, this error minimization is realized in a standardized form. With the data transformations in Eqs. (19) and (21)

$$e^S_{p_n} = x(z_n) - \Phi^{SN^{-1}}(y_n) \Rightarrow \text{Min } \forall n | r < n \leq N \tag{31}$$

is obtained. According to the back-propagation procedure the error energy

$$E_n = \frac{1}{2} \cdot (e^S_{p_n})^2 \tag{32}$$

is used to formulate the error minimization as

$$\sum_{n=r+1}^{N} (x(z_n) - \Phi^{SN^{-1}}(y_n))^2 \Rightarrow \text{Min} \tag{33}$$

with N being the length of the observed process realization (training data) and r denoting the order of the tapped delay line memory. Mathematically, Eq. (33) 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, see Haykin (1999). 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 until the prediction error approaches the global minimum. The training starts with randomly initialized weights and biases uniformly distributed in the interval [–1, +1] according to Eqs. (17) and (18). This ensures that the training data meet the effective part of the activation functions of the neurons to a high percentage, see Section 2.2.2.

A sequence of $r+1$ successive process values $(z_{n-r}, \ldots, z_{n-1}, z_n)$ are randomly selected from the training data with the aid of a discrete uniform distribution over the $N-r$ possible choices

$$(Z \sim U(1, 2, \ldots, N-r)) \rightarrow z_{n-r} \rightarrow (z_{n-r}, \ldots, z_{n-1}, z_n) \tag{34}$$

Then, the local gradients $\partial E_n(q) / \partial w_{j^{(L)} j^{(L-1)}}(q)$ in the weight space are determined for the current iteration step q. The weights and biases are changed proportional to these local gradients. The computation of the gradients is usually based on the error signal $e_0(q)$ from the direct network output $y_n \in (0, 1)$. In the present approach, however, the standardized error signal $e^S_{p_n}(q)$ as defined

in Eq. (31) and E_n from Eq. (32) are used instead of $e_0(q)$, without changing the computation rules. That is, in contrast to the standard version of error back-propagation, a weighted error signal is employed. Due to the computation of $e^S_{p_n}(q)$ with respect to predictions $\Phi^{SN^{-1}}(y_n)$ this makes the training procedure much more sensitive to errors in process values farther away from the process mean. The (weighted) error signal $e^S_{p_n}(q)$ is then traced back through the network to compute the new weights and biases for the next iteration step. Let $v_{j^{(L)}}(q)$ be the argument of the activation function $\varphi_{j^{(L)}}(\cdot)$ in neuron $j^{(L)}$ of layer (L) and $y_{j^{(L-1)}}(q)$ denote the output of neuron $j^{(L-1)}$ in the previous layer $(L-1)$. The new weights for the next iteration step $q+1$ are then

$$w_{j^{(L)}j^{(L-1)}}(q+1) = w_{j^{(L)}j^{(L-1)}}(q+1) + \alpha \cdot w_{j^{(L)}j^{(L-1)}}(q-1) + \eta \cdot \delta_{j^{(L)}}(q) \cdot y_{j^{(L-1)}}(q) \tag{35}$$

with

$$\delta_{j^{(L)}}(q) = e^S_{p_n}(q) \cdot \varphi'_{j^{(L)}}(v_{j^{(L)}}(q)) \tag{36}$$

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

$$\delta_{j^{(L)}}(q) = \varphi'_{j^{(L)}}(v_{j^{(L)}}(q)) \cdot \sum_{j^{(L+1)}=1}^{J^{(L+1)}} \delta_{j^{(L+1)}}(q) \cdot w_{j^{(L+1)}j^{(L)}}(q) \tag{37}$$

for the neurons $j^{(L)}$ in layer (L). In this procedure, the biases $b_{j^{(L)}}$ are treated as weights $w_{j^{(L)}j^{(L-1)}}$ for constant signals $y_{j^{(L-1)}} = +1$ from the previous layer $(L-1)$. Specifically

$$b_{j^{(L)}}(q+1) = b_{j^{(L)}}(q) + \alpha \cdot b_{j^{(L)}}(q-1) + \eta \cdot \delta_{j^{(L)}}(q) \tag{38}$$

with $\delta_{j^{(L)}}(q)$ from Eqs. (36) and (37), respectively. The derivative of the activation function in Eqs. (36) and (37) is computed according to Eq. (4). 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 $(z_{n-r}, \ldots, z_{n-1}, z_n)$ is randomly selected according to Eq. (34) 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 Eq. (33).

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,

$$\lim_{q \to \infty} (e^S_{p_n}) = 0 \quad \forall n | r < n \leq N \tag{39}$$

This justifies the assumption in Eq. (25) as a basis for applying the trained network with a

progressive prediction mode.

The determination of a particular appropriate structure of the network is problem dependent. The number of layers, neurons in each layer, and the order r of the memory 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. This network synthesis is generally an interactive procedure. Strategies and algorithms for an automated synthesis of a neural network such as discussed in Haykin (1999) may be used to support the search for a solution. The self-organization of a neural network may, however, limit the variety of network solutions and may preclude the identification of the optimum configuration. Respective strategies are developed for certain problems; see, for example, Su and Chang (2001). These research efforts concern not only complex networks but feed-forward networks as well, with a single hidden layer Teoh *et al.* (2006). A general solution for all cases does not exist.

5. Examples

5.1 Numerically generated time series

The capabilities and the features of the neural network-based process simulation are first 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 realization $\{z_1, z_2, z_3, \ldots\}$ of a stationary ergodic Gaussian process $Z(t, \theta)$ with the filtered noise power spectrum

$$S(\omega) = \frac{1 + a \cdot \omega^2}{(\omega_n^2 - \omega^2) + (2 \cdot \zeta \cdot \omega \cdot \omega_n)^2} \tag{40}$$

The parameters are selected as

$$a = 5, \quad \zeta = 0.25, \quad \omega_n = 10 \text{ rad/s} \tag{41}$$

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

$$\Delta t = 0.0628 \text{ s} \tag{42}$$

For the neural network-based simulation a subsequence of only $N = 50$ successive values z_n from the generated time series is taken as the basis. An appropriate network architecture is found with a tapped delay line memory of order $r = J^{(1)} = 12$ and two hidden layers with $J^{(2)} = 13$ and $J^{(3)} = 7$ neurons, respectively. The network possesses one output neuron, $J^{(4)} = 1$. This network thus contains a total of

$$\sum_{L=1}^{3} (J^{(L)} + 1) \cdot J^{(L+1)} = 275 \tag{43}$$

free values (weights and biases), which are adjusted iteratively via error back-propagation during the network training. Learning rate and momentum factor are chosen as

$$\eta = 1.0, \quad \alpha = 0.7 \tag{44}$$

The development of the prediction error $e_{p_n}^S$ with proceeding network training is shown in Fig. 5. After $q = 80{,}000$ iteration steps the error remains under 10^{-6}. The trained network is then applied to generate long process realizations initialized by randomly generated "seed" vectors according to Eq. (29).

The quality of the generated realizations is evaluated by a statistical comparison with the training record as well as with the actual, underlying, process. Mean value and standard deviation show a good agreement, see Table 2. 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

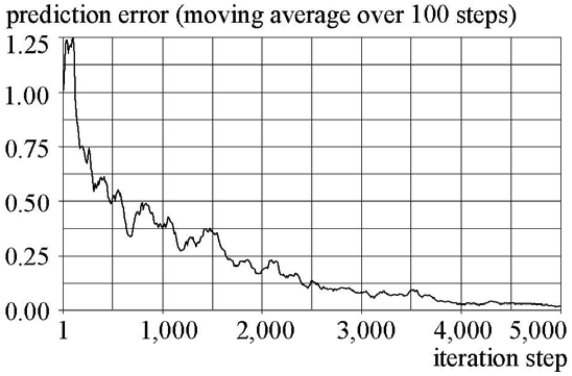

Fig. 5 Prediction error decrease during network training

Table 2 Comparison of mean values and standard deviations

	Mean value μ	Standard deviation σ
Training record	−0.001	0.169
Network prediction	−0.007	0.176
Underlying process	0.000	0.174

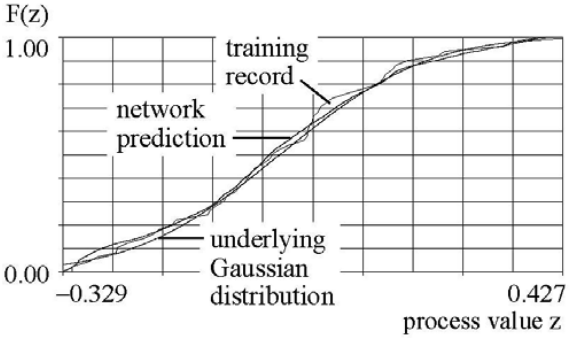

Fig. 6 First-order distribution functions

Table 3 Rejection probabilities of the H_0 hypothesis

	Neural network prediction	Underlying Gaussian distribution
Kolmogorov-Smirnov test	0.024	0.000
Chi-squared test (six classes)	0.028	0.217

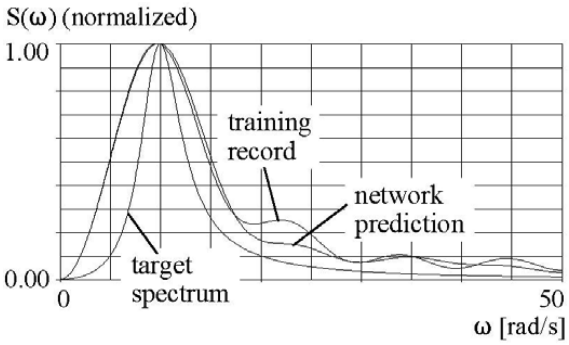

Fig. 7 Spectral density estimations – one network based simulation

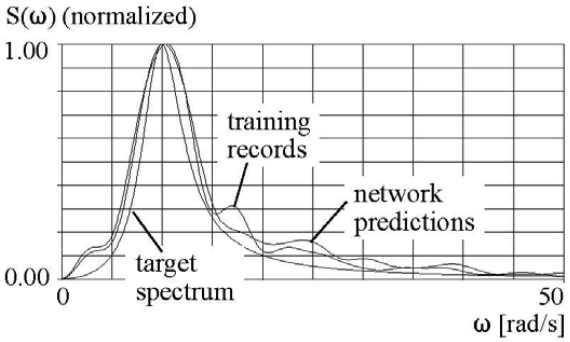

Fig. 8 Spectral density estimations – averaged over ten network based simulations

underlying distribution at the left end of the curves results from a concentration of process values in this range 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, according to which the training data and the generated data originate from the same population, see Table 3. Even the underlying Gaussian distribution does not lead to significantly better results.

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.

In an enhanced study, ten different, non-overlapping training records of $N = 50$ values each are taken from the original process realization 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 for 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 prediction 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 possesses features of convergence towards the exact solution in mean sense.

5.2 Measurement series

In this example the network-based process prediction is applied to a real series of measurements of material strength. Data are provided as an ordered sample of 250 measured values of the compressive strength of a building material. It is stated that this sample represents an observed realization $\{z_1, z_2, z_3, \ldots\}$ of a stochastic process $Z(t, \theta)$. A physical or mechanical background of the process, however, is not disclosed. Even the process parameter t remains unknown. The data are analyzed blindly.

In the first part of the example, the entire observed process realization is used for training the network, $N = 250$. An appropriate network architecture has been found with $r = J^{(1)} = 12$, $J^{(2)} = 25$, $J^{(3)} = 21$, and $J^{(4)} = 1$. Again, the training procedure is performed until the prediction error approaches zero. Then, the network-based Monte Carlo simulation is carried out with randomly specified "seed" vectors as described by Eq. (29).

For evaluating the results, the process is assumed to be ergodic. Table 4 shows the comparison between training data and network prognosis by means of their mean values and standard deviations. There is reasonable matching in the mean values and a moderate difference in the standard deviations. To assess the differences, two-sided interval estimations are applied based on the assumption of a Gaussian distribution, which was suggested as common for modeling the investigated material strength. For a confidence level of 98% both parameters of the network prognosis lie within the intervals, see Table 4.

The empirical first-order distribution functions of the training data and of the network prediction, and an estimated Gaussian distribution are compared in Fig. 9. Only minor deviations appear. The curve from the network prediction exhibits a noticeable smoothness. Homogeneity tests and goodness-of-fit tests lead to acceptable rejection probabilities for the H_0 hypothesis, see Table 5. The estimated Gaussian distribution, which represents the common treatment of such measurement series, does not reach the quality of the network prognosis. In the present case, it would even be

Table 4 Mean values and standard deviations

	Mean value μ	Standard deviation σ
Training record	53.17	7.23
Network prediction	53.15	6.57
Confidence intervals on the level of 98%	[52.10, 54.24]	[6.55, 8.07]

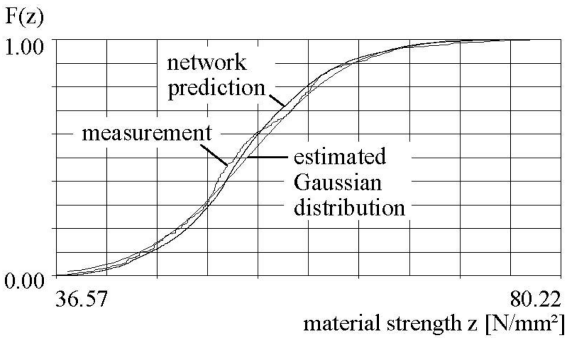

Fig. 9 Comparison of first-order distribution functions

Table 5 Rejection probabilities

	Neural network prediction	Estimated Gaussian distribution
Kolmogorov-Smirnov test	0.81	0.85
Chi-squared test (14 classes)	0.57	0.98

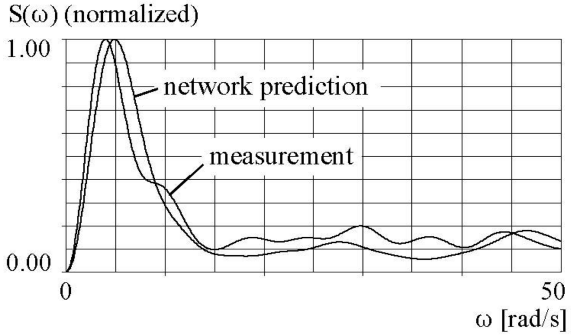

Fig. 10 Comparison of spectral density estimations

rejected. In comparison with the first example in Section 5.1, the absolute values of the rejection probabilities are significantly higher. This, however, is caused by the larger size of the training record with $N = 250$ in comparison to $N = 50$ in Section 5.1 and does not indicate a worse result in general.

To generate spectral density estimations from the training record and from the network prediction, a fictitious time scale is introduced. The time period $\Delta t = 2\pi$ seconds is assigned to a process sequence of 125 measured value. The spectral densities are then determined by averaging periodograms over non-overlapping subsequences of 25 successive time steps. The resulting curves show again a reasonable matching, see Fig. 10. The network prediction yields a slightly smoother curve.

The second part of the example is devoted to the prediction of the future behavior of a single process realization. For demonstration, only the first half of the measured values is used for training the network, $N = 125$. Then, a progressive prediction of the measurement process is generated and compared with the actually measured values. In contrast to the previous cases, a reasonable network structure is now found with three hidden layers. Specifically, $r = J^{(1)} = 21$, $J^{(2)} = 30$, $J^{(3)} = 20$, $J^{(4)} = 10$, and $J^{(5)} = 1$.

After the training of the network, the prediction is started with last sequence of training data according to Eq. (26). That is, the network generates data

$$z_n = p_n(z_{n-r}, \ldots, z_{n-1}), \quad n = 126, 127, 128, \ldots \tag{45}$$

This prediction is compared with the actual further behavior of the measurement series; see Fig. 11. The curves show a reasonable matching. The essential process behavior has been recognized by the network. This becomes even more obvious in Fig. 12 showing the two-sided moving average over seven process values. The average error in this prediction is only 5.15%.

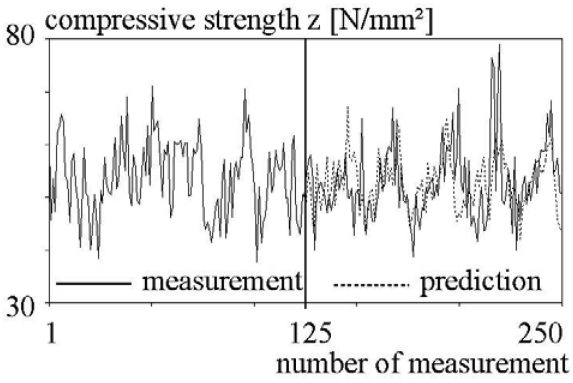

Fig. 11 Prediction of a single process realization

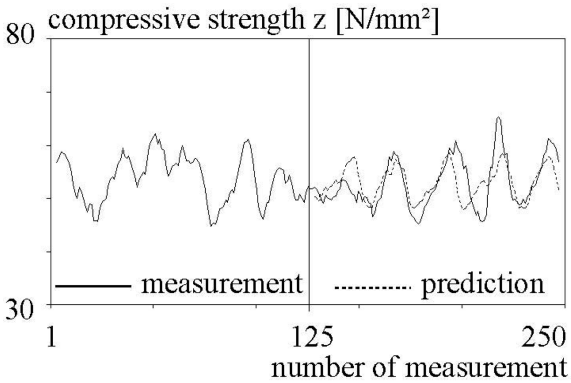

Fig. 12 Moving average of the single realization prediction

6. Conclusions

Neural networks may represent a viable approach for simulating stochastic processes in various engineering fields. They can be particularly helpful in cases in which various properties of the process cannot be identified or specified precisely. Compared to traditional methods, a model specification is not required. Moreover, neural networks possess a greater intrinsic complexity compared to the traditional simulation models. It may, thus, be expected that they are suitable in capturing process behavior for complex problems.

On the downside, a proof for the appropriateness of a network-based stochastic simulation does not yet exist to the authors' best knowledge and may be difficult to achieve in general terms. Further, a network solution is not unique with respect to both the network configuration and the weight adjustment. The determination of a suitable network configuration includes a trial-and-error component. And the error surface in the network training may possess several local minima of comparable quality. Nevertheless, the examples in this paper, and general other applications of neural networks in engineering have demonstrated their capabilities and usefulness.

Further developments associated with the presented procedure may focus on extensions aiming at applicability to non-stationary and multivariate problems. The development of an automated generation of a suitable network configuration for process simulation may also be achievable.

Ackowledgements

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

References

Bishop, Ch.M. (1995), *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.
Govindaraju, R.S. (2000), Artificial Neural Networks in Hydrology I: Preliminary Concepts. ASCE Task

Committee on Application of Artificial Neural Networks in Hydrology. *J. Hydrologic Eng.*, ASCE, **5**(2), 115-123.
Toth, E., Brath, A. and Montanari, A. (2000), "Comparison of short-term rainfall prediction models for real-time flood forecasting", *J. Hydrology*, **239**(1-4), 132-147.
Belli, M.R., Conti, M., Crippa, P. and Turchetti, C. (1999), "Artificial neural networks as approximators of stochastic processes", *Neural Networ.*, **12**(4-5), 647-658.
Cai, G.Q. and Wu, C. (2004), "Modeling of bounded stochastic processes", *Probabilist. Eng. Mech.*, **19**(3), 197-203.
Cao, L.J. and Tay, F.E.H. (2003), "Support vector machine with adaptive parameters in financial time series forecasting", *IEEE T. Neural Networ.*, **14**(6), 1506-1518.
Cimino, G., Del Duce, G., Kadonaga, L.K., Rotundo, G., Sisani, A., Stabile, G., Tirozzi, B. and Whiticar, M. (1999), "Time series analysis of geological data", *Chemical Geology*, **161**(1-3), 253-270.
Du, S., Ellingwood, B.R. and Cox, J.V. (2005), "Solution methods and initialization techniques in SFE analysis of structural stability", *Probabilist. Eng. Mech.*, **20**(2), 179-187.
Field, Jr. R.V. and Grigoriu, M. (2004), "On the accuracy of the polynomial chaos approximation", *Probabilist. Eng. Mech.*, **19**(1-2), 65-80.
Furundzic, D. (1998), "Application example of neural networks for time series analysis: Rainfall-runoff modeling", *Signal Process.*, **64**(3), 383-396.
Ghanem, R. and Spanos, P.D. (2003), Stochastic Finite Elements: A Spectral Approach. New York Berlin Heidelberg: Springer, 1991. Mineola, New York: Dover Publications, Revised Edition.
Giles, C.L., Lawrence, S. and Tsoi, A.C. (2001), "Noisy time series prediction using a recurrent neural network and grammatical inference", *Mach. Learn.*, **44**, 161-183.
Govindaraju, R.S. (2000), Artificial Neural Networks in Hydrology I: Hydrologic Applications. ASCE Task Committee on Application of Artificial Neural Networks in Hydrology. *J. Hydrologic Eng.*, ASCE, **5**(2), 124-137.
Grigoriu, M., Ditlevsen, O. and Arwade, S.R. (2003), "A Monte Carlo simulation model for stationary non-Gaussian processes", *Probabilist. Eng. Mech.*, **18**(1), 87-95.
Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice Hall.
Hirasawa, K., Mabu, S. and Hu, J. (2006), "Propagation and control of stochastic signals through universal learning networks", *Neural Networks*, **19**(4), 487-499.
Iwan, W.D. and Jensen, H. (1993), "On the dynamic-response of continuous systems including model uncertainty", *J. Appl. Mech., Trans.,* ASME, **60**(2), 484-490.
Jiang, X. and Adeli, H. (2005), "Dynamic wavelet neural network model for traffic flow forecasting", *J. Transp. Eng.*, **131**(10), 771-779.
Lee, S.-C. (2003), "Prediction of concrete strength using artificial neural networks", *Eng. Struct.*, **25**, 849-857.
Li, H. and Kozma, R. (2003), A Dynamical Neural Network Method for Time Series Prediction Using the KIII Model. In: Proceedings of the International Joint Conference on Neural Networks, IJCNN 2003. Portland, OR; 2003: IEEE press, 347-352.
More, A. and Deo, M.C. (2003), "Forecasting wind with neural networks", *Marine Structures*, **16**, 35-49.
Ochoa-Rivera, J.C. (2008), "Prospecting droughts with stochastic artificial neural networks", *J. Hydrology*, **352**(1-2), 174-180.
Phoon, K.K., Huang, S.P. and Quek, S.T. (2002), "Implementation of Karhunen-Loeve Expansion for simulation using a wavelet-Galerkin scheme", *Probabilist. Eng. Mech.*, **17**(3), 293-303.
Qi, M. and Zhang, G.P. (2008), "Trend time-series modeling and forecasting with neural networks", *IEEE T. Neural Networ.*, **19**(5), 808-816.
Sakamoto, S. and Ghanem, R. (2002), "Simulation of multi-dimensional non-gaussian non-stationary random fields", *Probabilist. Eng. Mech.*, **17**(2), 167-176.
Sanchez, M.S. and Sarabia, L.A. (2002), "A stochastic trained neural network for nonparametric hypothesis testing", *Chemometr. Intell. Lab. Syst.*, **63**(2), 169-187.
Schenk, C.A. and Schuëller, G.I. (2005), Uncertainty Assessment of Large Finite Element Systems. Berlin Heidelberg: Springer.
Schenk, C.A., Pradlwarter, H.J. and Schuëller, G.I. (2005), "Non-stationary response of large, non-linear finite

element systems under stochastic loading", *Comput. Struct.*, **83**(14), 1086-1102.
Schuëller, G.I. (2001), "Computational stochastic mechanics – recent advances", *Comput. Struct.*, **79**(22-25), 2225-2234.
Schuëller, G.I. (2003), On Computational Procedures for Processing Uncertainties in Structural Mechanics. In: Waszczyszyn, Z., Pamin, J., editors. 2nd European Conference on Computational Mechanics, ECCM 2001, Cracow; 2003: CD-ROM, Doc 608, 1-24.
Schuëller, G.I., Spanos, P.D. editors. (2001), Monte Carlo simulation: Proceedings of the International Conference on Monte Carlo Simulation 2000. Lisse, Exton, PA: A.A. Balkema.
Shi, Z. and Han, M. (2007), "Support vector echo-state machine for chaotic time-series prediction", *IEEE T. Neural Networ.*, **18**(2), 359-372.
Spanos, P.D. and Zeldin, B.A. (1998), "Monte Carlo treatment of random fields: A broad perspective", *Appl. Mech. Rev.*, **51**, 219-237.
Spanos, P.D., Beer, M. and Red-Horse, J. (2007), "Karhunen-Loéve expansion of stochastic processes with a modified exponential covariance kernel", *J. Eng. Mech.*, ASCE, **133**(7), 773-779.
Su, M.C. and Chang, H.T. (2001), "A new model of self-organizing neural networks and its application in data projection", *IEEE T. Neural Networ.*, **12**(1), 153-158.
Teoh, E.J., Tan, K.C. and Xiang, C. (2006), "Estimating the number of hidden neurons in a feedforward network using the singular value decomposition", *IEEE T. Neural Networ.*, **17**(6), 1623-1629.
Turchetti, C., Conti, M., Crippa, P. and Orcioni, S. (1998), "On the approximation of stochastic processes by approximate identity neural networks", *IEEE T. Neural Networ.*, **9**(6), 1069-1085.
Turchetti, C., Crippa, P., Pirani, M. and Biagetti, G. (2008), "representation of nonlinear random transformations by non-gaussian stochastic neural networks", *IEEE T. Neural Networ.*, **19**(6), 1033-1060.
Vanmarcke, E.H. (1983), *Random Fields*. Cambridge, MA: MIT Press.
Yuen, K.-V., Katafygiotis, L.S. and Beck, J.L. (2002), "Spectral density estimation of stochastic vector processes", *Probabilist. Eng. Mech.*, **17**(3), 265-272.
Zhang, G.P. and Kline, D.M. (2007), "Quarterly time-series forecasting with neural networks", *IEEE T. Neural Networ.*, **18**(6), 1800-1814.

Notation

The following symbols are used in this paper:

a	: real number, parameter
A	: interval, set
b	: bias
∂	: partial derivative
D	: real valued domain of definition of a stochastic process
e_p	: prediction error
E	: error energy
$f(\underline{x})$, $g(\underline{x})$	: functions of $\underline{x}$
$F^{(e)}(\cdot)$	: empirical distribution function
$I(\cdot)$	: indicator function
$j^{(L)}$	: number of neuron in layer (L)
L	: number of a layer
N	: size of observed process realization
p	: network prediction
P	: probability measure
q	: iteration step counter
r	: order of tapped delay line memory
$S(\cdot)$	: power spectrum
$\mathfrak{S}$	: σ-algebra
$t \in D$	: process parameter (e.g., spatial or temporal coordinate)

$U(\cdot)$	: uniform distribution over some domain
v	: activation potential
V_{eff}	: effective part of the activation function
w	: weight
x	: network input signal
X, Z	: random variable
y	: neuron and network output signal
$Z(t, \theta)$	: stochastic process
$\{z_1, z_2, z_3, \ldots\}$	: process realization
$\bar{z}$	: mean of Z
α	: momentum factor
δ	: local error gradient
ε	: small real number
η	: learning rate
$\varphi(.)$	: activation function
$\Phi^{SN\,-1}(\cdot)$	: inverse function of the standard normal distribution function
$\theta \in \Omega$	: elementary event
σ_Z	: standard deviation of Z
ω	: angular frequency
Ω	: space of elementary events
ζ	: real number, parameter
$[\Omega, \mathcal{S}, P]$	: probability space
$\square'$	: derivative