



Using neural network function approximation for optimal design of continuous-state parallel–series systems

Peter X. Liu^a, Ming J. Zuo^{b,*}, Max Q.-H. Meng^a

^a*Department of Electrical and Computer Engineering University of Alberta, Edmonton, Alta., Canada T6G 2G7*

^b*Department of Mechanical Engineering, University of Alberta, 4-9 Mechanical Engineering Building, Edmonton, Alta., Canada T6G 2G8*

Received 1 November 2000; received in revised form 1 July 2001

Abstract

This paper presents a novel continuous-state system model for optimal design of parallel–series systems when both cost and reliability are considered. The advantage of a continuous-state system model is that it represents realities more accurately than discrete-state system models. However, using conventional optimization algorithms to solve the optimal design problem for continuous-state systems becomes very complex. Under general cases, it is impossible to obtain an explicit expression of the objective function to be optimized. In this paper, we propose a neural network (NN) approach to approximate the objective function. Once the approximate optimization model is obtained with the NN approach, the subsequent optimization methods and procedures are the same and straightforward. A 2-stage example is given to compare the analytical approach with the proposed NN approach. A complicated 4-stage example is given to illustrate that it is easy to use the NN approach while it is too difficult to solve the problem analytically.

Scope and purpose

The classical reliability theory assumes that the system and each component may only be in one of two possible states: working or failed. Thus, it is also referred to as binary reliability theory. A well-known reliability design problem under the binary reliability theory involves the determination of the number of redundancies in a parallel–series system which consists of N subsystems connected in series whereas each subsystem consists of a few components connected in parallel. In this paper, we consider the optimal design problem of a multi-state parallel–series system wherein both the system and its components may assume more than two levels of performance. Specifically, we assume that the state of each component and the system may be represented by a continuous random variable that may take values in the closed interval $[0, 1]$. An optimization model is formulated for the determination of the number of redundancies in order to maximize

* Corresponding author. Tel.: +1-780-492-4466; fax: +1-780-492-2200.

E-mail address: ming.zuo@ualberta.ca (M.J. Zuo).

the system's expected utility function. Because of the complexity of the optimization problem, we propose a neural network (NN) approach to approximate the objective function. The resulting optimization model is much easier to solve. Examples are given to illustrate the proposed approach. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Neural networks; Continuous-state system; Parallel-series system; Optimal system design; Multi-state system

1. Introduction

A system consists of n components, each of which may perform a different function. One of the most important measures of the performance of a system is its reliability. The reliability of a system is defined to be the probability that the system will perform its functions satisfactorily for a certain period of time under specified conditions.

The traditional reliability theory assumes that a system and its components may only experience one of two possible states: working or failed. As a result, we call it binary reliability theory. Under the binary assumption, reliability as defined above is an excellent measure of the performance of systems. There exist several methods for designing systems with high reliability. These methods include using large safety factors, reducing the complexity of the system, increasing the reliability of constituent components, and using structural redundancy. Kuo et al. [1] provides an extensive coverage on optimal system design.

A parallel-series system consists of N subsystems connected in series such that the system works if and only if all the subsystems work wherein subsystem i ($1 \leq i \leq N$) consists of M_i components connected in parallel such that the subsystem fails if and only if all the components in this subsystem fail. Fig. 1 shows such a parallel-series configuration. The reliability of such a parallel-series system is expressed as:

$$R_s = \prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} (1 - p_{ij}) \right), \quad (1)$$

where p_{ij} is the reliability of component j in subsystem i . For such a system, a typical optimization problem involves finding the optimal number of parallel components in each subsystem either to maximize system reliability or minimize total system cost. The constraints for such problems are either resource limitations or reliability requirements. Resource limitations usually represent constraints of cost, weight, volume or some combinations of these factors. The reliability constraint imposes a minimum requirement of system reliability. In either case, the system design problem is a nonlinear integer-programming problem. Many algorithms have been proposed but none has proven to be superior over the others. Kuo et al. [1] surveyed and classified optimization techniques related to nonlinear programming problems. They compared the pros and cons of the following optimization techniques: integer programming, transforming nonlinear to linear functions, dynamic programming, the sequential unconstrained minimization technique (SUMT), the generalized reduced gradient method (GRG), the modified sequential simplex pattern search, and the generalized Lagrangian

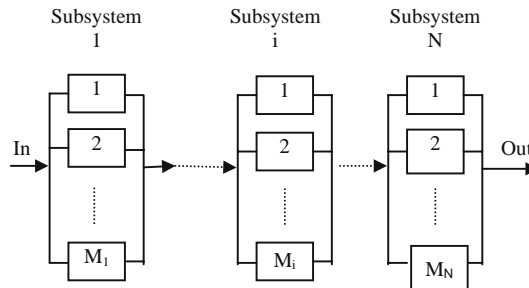


Fig. 1. The structure of a parallel-series system.

function method. Other examples of integer programming solutions to the redundancy allocation problem are presented by Misra and Sharma [2], and Gen et al. [3,4]. In recent years, genetic algorithms have been used for solving reliability based design problems, for example, see [5,6].

The binary assumption has served as a unifying foundation for the mathematical theory of reliability. However, in many real-life situations, a multi-state system model is needed to allow both the system and its components to assume more than two levels of performance. In a *discrete multi-state system*, it is assumed that the system and its components may each experience $M + 1$ possible states ($M \geq 1$): $0, 1, \dots, M$, where 0 represents the completely failed state, M represents the completely working state, and others are intermediate states. For studies of discrete multi-state systems, users are referred to Barlow and Wu [7], Griffith [8], Xue and Yang [9], and Huang et al. [10]. Zuo et al. [11] considered the redundancy allocation problem for discrete multi-state parallel-series systems and provided a heuristic algorithm for solving the optimal design problem.

In a *continuous multi-state system*, it is assumed that the states of the system and its components may each be represented by a continuous random variable defined in the closed interval $[0, 1]$, where 0 represents the complete failure state and 1 the completely working state. For studies of continuous multi-state systems, readers are referred to Ross [12], Block and Savits [13], and Cappelle and Kerre [14].

In a multi-state system, be it continuous or discrete, the definition of reliability as given under the binary assumption is no longer valid. Different measures of system performance need to be defined. In this paper, we consider the redundancy allocation problem for continuous-state parallel-series systems. The system utility function is used to measure the performance of the system. The NN approach is used to solve the nonlinear integer optimization problem. Examples are given to illustrate the advantages of the NN approach.

2. The continuous-state system design model

Consider a parallel-series system as shown in Fig. 1. There are N subsystems connected in series. Subsystem i (for $1 \leq i \leq N$) has M_i components connected in parallel. The state of a component can be represented by a continuous random variable defined in the range from 0 to 1, inclusive. Before introducing the optimal design model of the continuous-state parallel-series system, we provide the

following list of notation:

Ω	range of possible state, $\Omega = [0, 1]$, where 1 indicates perfectly functioning state and 0 the completely failed state
s	state index, a deterministic value, $s \in \Omega$
X_{ij}	state of component j in subsystem i , a random variable, $0 \leq x_{ij} \leq 1$
\mathbf{x}	the vector representing the states of all components in the system, $\mathbf{x} = (x_{ij})$, $i = 1, 2, \dots, N$; $j = 1, 2, \dots, M_i$
N	total number of subsystems of the system
M_i	total number of components in the i th subsystem
$\phi(\mathbf{x})$	system structure function, or system state as a function of component states, a random variable, $0 \leq \phi(\mathbf{x}) \leq 1$
$f_{ij}(s)$	probability density function of x_{ij}
$f_i(s, M_i)$	probability density function of the state of the i th subsystem
$f(s, M_1, \dots, M_N)$	probability density function of $\phi(\mathbf{x})$, the state of the system
$\mu(s)$	utility function of the system when it is in state s
C_{ij}	cost of the j th component in the i th subsystem
C_T	total budget available for the system to be designed

In addition, we also use the following assumptions:

1. The states of the components in the same subsystem are independently and identically distributed (iid).
2. The utility function of the system, $\mu(s)$, is known.
3. The state probability density function $f_{ij}(s)$ of each component is known.

According to Barlow and Wu [7], the state of a parallel system is equal to the state of the best component in the system while the state of a series system is equal to the state of the worst component in the system. As a result, the state of the parallel–series system shown in Fig. 1 can be expressed as

$$\phi(x) = \min_{1 \leq i \leq N} \max_{1 \leq j \leq M_i} x_{ij}, \tag{2}$$

where x_{ij} has probability density function $f_{ij}(s)$ for $0 \leq s \leq 1$. Using Eq. (2), we can evaluate the following probability:

$$\Pr(\phi(x) \geq s) = \prod_{i=1}^N \left(\int_s^1 f_i(t, M_i) dt \right) = \prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) \text{ for } 0 \leq s \leq 1. \tag{3}$$

It is obvious that $\Pr(\phi(x) \geq s)$ as shown in Eq. (3) is a function of s, M_1, M_2, \dots , and M_N .

Since $\Pr(\phi(x) \geq s) \equiv \int_s^1 f(t, M_1, M_2, \dots, M_N) dt$, we obtain the following expression of the probability density function of $\phi(\mathbf{x})$:

$$f(s, M_1, M_2, \dots, M_N) = -\frac{d}{ds} \Pr(\phi(\mathbf{x}) \geq s) = -\frac{d}{ds} \left(\prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) \right). \tag{4}$$

The expected utility of the system is:

$$\begin{aligned}
 U(M_1, M_2, \dots, M_N) &= \int_0^1 \mu(s) f(s, M_1, M_2, \dots, M_N) ds \\
 &= - \int_0^1 \mu(s) \left(\frac{d}{ds} \left(\prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) \right) \right) ds. \tag{6}
 \end{aligned}$$

The expected system utility, $U(M_1, M_2, \dots, M_N)$, depends on the following factors: (1) the number of sub-systems, N ; (2) the number of components in each subsystem, M_1, M_2, \dots, M_N ; (3) state probability density function of each component, $f_{ij}(s)$; and (4) utility function of the system when it is in the state s , $\mu(s)$. The number of subsystems, N , is usually determined by the system function required. In this paper, we also assume that both $f_{ij}(s)$ and $\mu(s)$ are known. Thus the optimal design problem is concerned with finding the optimal values of M_1, M_2, \dots , and M_N to maximize $U(M_1, M_2, \dots, M_N)$ subject to the cost constraints, as shown below:

$$\begin{aligned}
 \text{Maximize : } U(M_1, M_2, \dots, M_N) &= - \int_0^1 \mu(s) \left(\frac{d}{ds} \left(\prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) \right) \right) ds \\
 \text{Subject to : } \sum_{i=1}^N \sum_{j=1}^{M_i} C_{ij} &\leq C_T.
 \end{aligned}$$

This optimization model includes both integration and differentiation in the objective function. The objective function is very complicated and it is very difficult to use a classical optimization algorithm. This situation arises when (1) the number of subsystems, N , is large; (2) f_{ij} 's in the same subsystem are not identical; and/or (3) f_{ij} is not a simple distribution. In addition, in some cases, the component state distribution function may have to be expressed in an empirical form and, as a result, no analytical expression of $U(M_1, M_2, \dots, M_N)$ is available. In the following section, we propose to use the NN approximation to solve this optimization problem.

3. The NN approximation

We know that $\Pr(\varphi(x) \geq s)$ is a nonlinear function of variables, s, M_1, M_2, \dots, M_N . For convenience, we define that,

$$g(s, M_1, M_2, \dots, M_N) \equiv \Pr(\varphi(x) \geq s) = \prod_{i=1}^N \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right). \tag{7}$$

Thus, the objective function becomes,

$$U(M_1, M_2, \dots, M_N) = - \int_0^1 \mu(s) \frac{d}{ds} (g(s, M_1, M_2, \dots, M_N)) ds. \tag{8}$$

If $g(s, M_1, M_2, \dots, M_N)$ is only a linear combination of standard sigmoidal functions, $U(M_1, M_2, \dots, M_N)$ has a simple expression as to be demonstrated through examples later. This observation stimulates us to use neural networks to approximate $g(s, M_1, M_2, \dots, M_N)$. Because of the monotonic

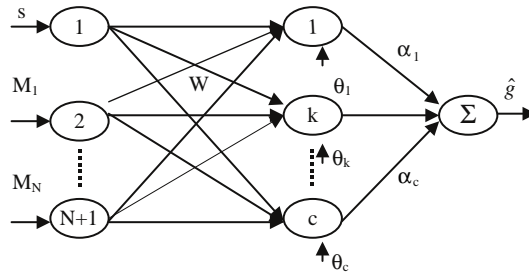


Fig. 2. The structure of a feedforward neural network with one hidden layer.

and nonlinear characteristics of neurons, sigmoidal neurons for example, NNs are universal approximators. Cybenko [15] and Aourid and Do [16] show that a finite linear combination of sigmoidal functions can approximate any continuous function of n real variables with support in the unit hypercube to any degree of accuracy. As a result, it is possible to use a feedforward NN with a single hidden layer as shown in Fig. 2, to approximate $g(s, M_1, M_2, \dots, M_N)$.

Generally, analytical expressions of multilayer NNs (the activation functions of two or more layers are nonlinear) are generally impossible to obtain. However, for the NN model in Fig. 2, only the single hidden layer is of nonlinear functions and the activation function of the output layer is a pure linear sum operator. Thus, the output of the NN is actually a weighted sum of a finite number of sigmoidal functions. Once the training is completed, we know the number of hidden neurons and all the weights and bias factors associated with each neuron. An analytical expression of the NN output, i.e., the approximate system distribution function in this paper, can be written as

$$\hat{g}(s, M_1, \dots, M_N) = \sum_{k=1}^c \alpha_k \phi(W_k^T \mathbf{y}' + \theta_k) \tag{9}$$

where $\mathbf{y}' = [s, M_1, \dots, M_N]^T$ is the input vector; $\phi(u) = 1/(1 + e^{-u})$ is the activation function of the hidden units; $W_k = [w_{k1}, w_{k2}, \dots, w_{k(N+1)}]^T$ is the weight vector associated with the hidden neuron k ; c is the number of hidden units, α_k is the weight between the hidden unit k and the output neuron, w_{ki} is the weight between the i th input neuron and the k th hidden neuron, and θ_k is the bias factor of the k th hidden neuron.

From Eq. (9), we can also see that no matter how complicated $g(s, M_1, M_2, \dots, M_N)$ might be, the approximate analytical expression of system distribution function, $\hat{g}(s, M_1, M_2, \dots, M_N)$, is always a linear combination of a finite number of sigmoidal functions. As stated earlier, the approximate objective function $\hat{U}(M_1, M_2, \dots, M_N)$ constructed from $\hat{g}(s, M_1, M_2, \dots, M_N)$ is usually solvable. Thus, the subsequent optimization procedure is exactly the same for various component state distribution functions. For simple and analytically solvable problems, NN approximation is unnecessary. However, for large, complicated and analytically unsolvable problems, the NN approximation can be as accurate as desired.

The basic idea to use NNs in this paper is to replace $g(s, M_1, M_2, \dots, M_N)$, which is a known nonlinear function involving integrals, with $\hat{g}(s, M_1, M_2, \dots, M_N)$, which is a linear combination of standard sigmoid functions by using the continuous and nonlinear mapping of the inputs to the outputs inherent in NNs.

Table 1
Speed comparison of backpropagation algorithms

Training optimization methods	Training CPU times (s)
Gradient descent (GDBP)	2467
Quasi-Newton (QNPB)	583
Levenberg–Marquardt (LMBP)	356

The NN: 3 input nodes, 1 single 12-neuron hidden layer, 1 output neuron, and the performance gradient is $1e-4$.

The NN in this paper is designed and implemented by using MATLAB[®] M-scripts. Levenberg–Marquardt backpropagation (LMBP) training algorithm [17,18] is adopted. Like the Quasi-Newton (or secant) methods, the LMBP algorithm can approach second-order training speed without having to compute the Hessian matrix [19]. When the performance function has the form of a sum of squares (as is typical in training feedforward NNs), the Hessian matrix H can be approximated as $\widehat{H} = J^T J$ and the gradient can be computed as $g = J^T e$, where J is the Jacobian matrix that contains the first derivatives of the NN errors with respect to the weights and biases and e is the vector of NN errors. The Jacobian matrix can be computed through a standard backpropagation technique that is much less complex than computing the Hessian matrix. The update scheme of the LMBP algorithm is thus as:

$$X_{k+1} = X_k - [J^T J + \lambda I]^{-1} J^T e,$$

where X_k is the vector of weights and biases at the k th iteration. When the scalar λ is zero, this is exactly the Newton's optimization, using the approximate Hessian matrix. While λ is large, this becomes the gradient descent method with a small step size. The Newton's method is faster and more accurate near the error minimum. The aim is to switch to the Newton's method as quickly as possible. Thus, λ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm. Another reason to use the LMBP algorithm is that it is much faster than other backpropagation algorithms as shown in Table 1 for Example 1.

Training and testing (validation) data sets are generated as follows: first, a suitable number of input vectors, $(s, M_1, M_2, \dots, M_N)$, are chosen or generated randomly from the expected value ranges of s, M_1, M_2, \dots , and M_N ; next, the input vectors are normalized and for each input vector, the desired output, $g(s, M_1, M_2, \dots, M_N)$, is calculated with Eq. (7). The training and testing data sets consist of different pairs of the input vector and its corresponding output.

An important but hard problem in NN design and implementation is to determine the number of hidden units. The difficulties root from the so-called overfitting and underfitting. The optimal number of hidden nodes depends in a complex way on the complexity of the function to be approximated, the type of activation functions in hidden layers, the numbers of input and output nodes; the training algorithm, the size of training data set, the amount of noise in training data and so forth. This topic remains an art and no once-for-all rules are available to follow. Sarle [20], Weigend [21] and Tetko et al. [22] report that there seems to be no upper limit on the number of hidden units, other than that imposed by computer time and memory requirements. In the LMBP algorithm used in the paper,

validation vectors (the elements of which are the global parameters such as the maximum number of epochs, the maximum amount of time, performance goal, and the minimum performance gradient, etc.) are used to stop training early if the NN performance on the validation vectors fails to improve or remains the same. Thus a relatively large number of hidden units can be used without bringing in overfitting. Hence, both underfitting and overfitting on NN training can be avoided effectively to a certain degree. In addition, test vectors are used as a further check that the NN is generalizing well.

4. Examples

4.1. Example 1

In order to visualize the comparison between the analytical and the NN solutions (as any entity beyond 3 dimensions is unable to be visualized effectively), we first consider the design problem of a 2-stage, analytically solvable parallel–series system in which $N = 2$; $C_{1i} = 15$; $C_{2i} = 20$; $C_T = 150$; $\mu(s) = 10s$; $f_{1i} = 1$ and $f_{2i} = 2s$.

Case 1: Analytical solution.

The objective function $U(M_1, M_2)$ is derived as

$$U(M_1, M_2) = 10 \left(\frac{2M_1}{2M_1 + 1} + \frac{M_2}{M_2 + 1} - \frac{2M_1 + M_2}{2M_1 + M_2 + 1} \right) \tag{10}$$

and the solution that maximizes this objective function subject to the resource constraint is:

$$M_1 = 3,$$

$$M_2 = 6,$$

$$U(3, 6) = 7.9121.$$

Case 2: NN approximation.

The training and validation data sets are obtained as follows: initial input vectors, $[s, M_1, M_2]^T$, are generated randomly in the expected value ranges ($s \in [0, 1], M_1, M_2 \in [0, 10]$) and normalized to be $y' = [s, m_1, m_2]^T \in [0, 1]^3$; the desired output target $g(s, M_1, M_2)$ is calculated according to the following equation:

$$g(s, M_1, M_2) = \Pr(\varphi(\mathbf{x}) \geq s) = \prod_{i=1}^2 \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) = (1 - s)^{M_1} (1 - s^2)^{M_2}.$$

The approximate objective function, $\hat{U}(M_1, M_2)$, constructed from the approximate system distribution function, $\hat{g}(s, M_1, M_2)$, with Eq. (8), is

$$\hat{U}(M_1, M_2) = 10 \sum_{i=1}^c \left(\alpha_{i+1} \left(1 - \frac{1}{A_i} + \frac{1}{w_{i2}} \ln \frac{A_i}{B_i} \right) \right), \tag{11}$$

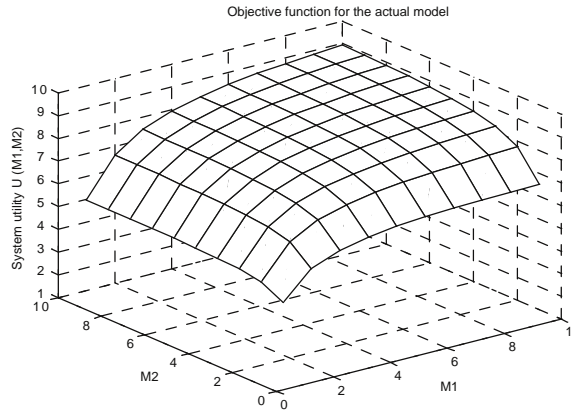


Fig. 3. $U_1(M_1, M_2)$.

where

$$A_i = 1 + e^{-(w_{i1}+w_{i2}+w_{i3}m_1+w_{i4}m_2)},$$

$$B_i = 1 + e^{-(w_{i1}+w_{i3}m_1+w_{i4}m_2)}.$$

The training data set contains 240 pairs of input vector and its corresponding output. The objective of training is that the percent error is smaller than 1%. We achieved this goal when the number of the hidden neurons c is equal to 12 after 2500 training epochs. The actual percent error is 0.80477%. The maximum error for a randomly generated testing set consisting of 50-pair data is 2.0628%. The final optimal design solution is

$$M_1 = 3,$$

$$M_2 = 6,$$

$$\hat{U}(3, 6) = 7.9210.$$

Although the analytical expression of the actual objective function in Eq. (10) obtained analytically and the approximate one in Eq. (11) obtained using the NN approximation are completely different, the actual objective function $U(M_1, M_2)$ and the approximate one $\hat{U}(M_1, M_2)$ shown in Figs. 3 and 4 respectively are almost identical. The final optimal solutions for both methods are the same, i.e., $M_1 = 3, M_2 = 6$. The values of the true objective function and the approximate objective function are very close, i.e., $U(3, 6) = 7.9121, \hat{U}(3, 6) = 7.9210$. In addition, from Fig. 5, the maximum percent error between these two functions is under 3.5%.

4.2. Example 2

In this example, we use a 4-stage, analytically unsolvable parallel-series system in which $N = 4; \mu(s) = 10s; C_{1i} = 3200; C_{2i} = 1700; C_{3i} = 830; C_{4i} = 2500; \text{ and } C_T = 160,000$. The probability

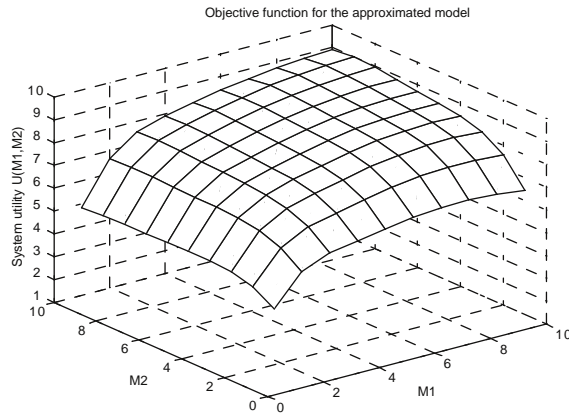


Fig. 4. $\hat{U}_1(M_1, M_2)$.

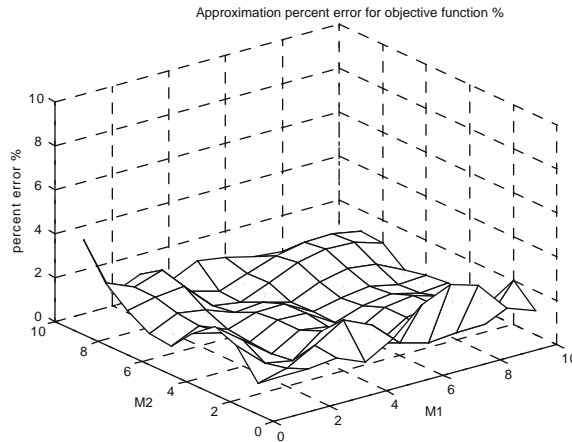


Fig. 5. Percent error between $\hat{U}(M_1, M_2)$ and $U_1(M_1, M_2)$.

density functions of components in the four subsystems are three commonly used distributions: unit distribution, triangular distribution and Beta distribution.

$$f_{1i} = 1; \text{ unit distribution,}$$

$$f_{2i} = 2s; \text{ triangular distribution,}$$

$$f_{3i}(s) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} s^{\alpha-1} (1 - s)^{\beta-1}, \text{ beta distribution where } \alpha = 2, \beta = 3.5,$$

$$f_{4i}(s) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} s^{\alpha-1} (1 - s)^{\beta-1}, \text{ beta distribution where } \alpha = 5, \beta = 2.$$

The density functions of the components in the 4 subsystems are plotted in Fig. 6.

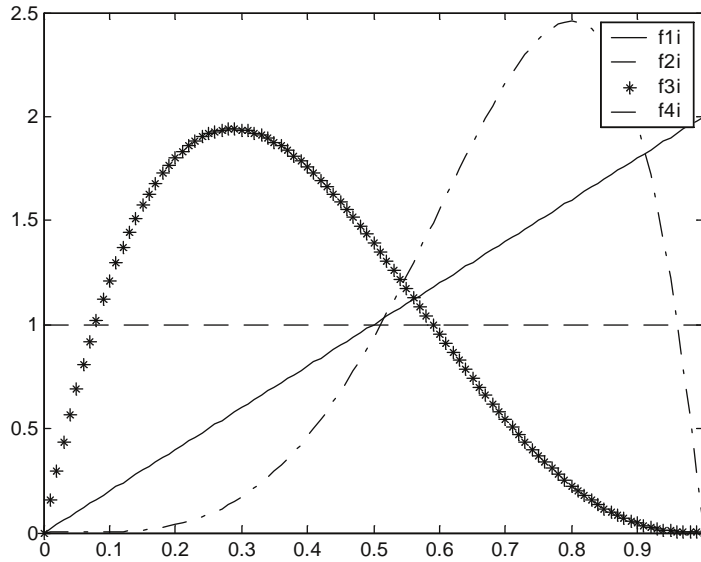


Fig. 6. Component state distribution functions for Example 2.

The desired output targets for NN training, i.e., actual system distribution function, are calculated with

$$\begin{aligned}
 g(s, M_1, M_2, M_3, M_4) &= \prod_{i=1}^4 \left(1 - \prod_{j=1}^{M_i} \left(\int_0^s f_{ij}(t) dt \right) \right) \\
 &= (1 - s)^{M_1} (1 - s^2)^{M_2} (1 - (3.5(1 - s^{4.5}) - 4.5(1 - s^{3.5}) + 0.0635)^{M_3} \\
 &\quad (1 - 6s^5 + 5s^6)^{M_4}.
 \end{aligned} \tag{12}$$

We can see that although Eq. (12) is also complicated, it is calculable. But the objective function $U(M_1, M_2, M_3, M_4)$ constructed from Eq. (6) is too complex and seems impossible to calculate analytically so that the whole optimization design is unsolvable analytically.

On the contrary, it is straightforward to solve this optimization problem using the NN approximation because the expression format of the approximate objective function is the same as in Example 1. In other words, the approximate objective function, $\hat{U}(M_1, M_2, M_3, M_4)$, constructed from the approximate system distribution function is

$$\hat{U}(M_1, M_2, M_3, M_4) = 10 \sum_{i=1}^L \left(\alpha_{i+1} \left(1 - \frac{1}{A_i} + \frac{1}{w_{i2}} \ln \frac{A_i}{B_i} \right) \right),$$

where

$$A_i = 1 + e^{-(w_{i1} + w_{i2} + w_{i3}m_1 + w_{i4}m_2 + w_{i5}m_3 + w_{i6}m_4)},$$

$$B_i = 1 + e^{-(w_{i1} + w_{i3}m_1 + w_{i4}m_2 + w_{i5}m_3 + w_{i6}m_4)}.$$

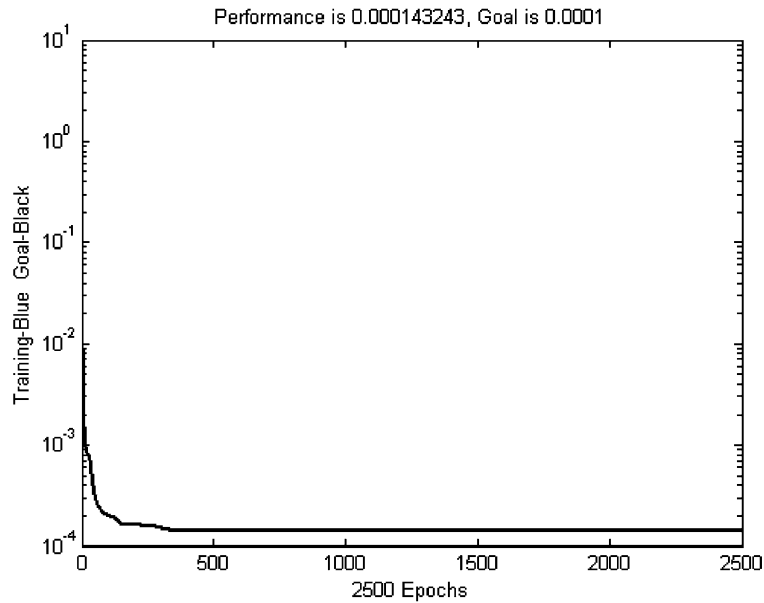


Fig. 7. Training performance versus training epoch for Example 2.

The training data set containing 1600 training pairs is generated randomly. The size of the hidden layer $c = 19$. After 2500 epochs of training iterations, the actual percent training error is 0.9859%. The maximum validation error is 2.4532%. Fig. 7 shows the training performance versus training epochs. After an exhaustive search, the final optimal solution is

$$M_1 = 14,$$

$$M_2 = 9,$$

$$M_3 = 36,$$

$$M_4 = 15$$

and the corresponding approximate utility is $\widehat{U}(12, 7, 35, 32) = 8.4384$.

5. Conclusion and discussions

The continuous-state model for parallel-series system optimal design makes sense in both theory and application. The advantage of a continuous-state model is that it models realities more accurately. However, in some cases, it is very difficult to solve the continuous-state model optimal design problem using classical optimization algorithms because the objective function becomes quite complex. There are also situations wherein explicit system objective function is not available except as a set of experimental data. To overcome these difficulties, we have proposed to use the NN approximation to approximate the objective functions.

Compared to classical direct search optimization methods, another advantage of the NN approximation is that no matter how complicated the system may be, the form of the approximate objective functions is always the same. Specifically, it is always a linear combination of some kind of non-linear functions, for example, sigmoidal functions in this paper. Once the approximate model is available, the subsequent search method and procedure may be identical. This is especially useful and efficient for the cases when the system is large, the state distribution functions of the components in the same subsystem are not identical, and/or there is no explicit system model available except an experimental data set.

References

- [1] Kuo W, Prasad VR, Tillman FA, Huang CL. Optimal reliability design. New York: Cambridge University Press, 2001.
- [2] Misra KB, Sharma U. An efficient approach for multiple criteria redundancy optimization problems. *Microelectronics and Reliability* 1991;31(2):303–21.
- [3] Gen M, Ida K, Lee JU. A computational algorithm for solving 0-1 goal programming with GUB structures and its applications for optimization problems in system reliability. *Electronics and Communication in Japan: Part 3* 1990;73:88–96.
- [4] Gen M, Ida K, Tsujimura Y, Kim CE. Large scale 0-1 fuzzy goal programming and its application to reliability optimization problem. *Computers and Industrial Engineering* 1993;24:539–49.
- [5] Coit DW, Smith AE. Reliability optimization of the parallel-series systems using a genetic algorithm. *IEEE transactions on Reliability* 1996;45(2):254–60.
- [6] Monga A, Zuo MJ. Optimal system design considering maintenance and warranty. *Computers and Operations Research* 1998;25(9):691–705.
- [7] Barlow RE, Wu AS. Coherent systems with multi-state components. *Mathematics of Operations Research* 1978;3(4):275–81.
- [8] Griffith WS. Multi-state reliability models. *Journal of Applied Probability* 1980;17:735–44.
- [9] Xue J, Yang K. Dynamic reliability analysis of coherent multi-state systems. *IEEE Transactions on Reliability* 1995;R-44(4):683–8.
- [10] Huang J, Zuo MJ, Wu Y. Generalized multi-state k-out-of-n:G systems. *IEEE Transactions on Reliability* 2000;49(1):105–11.
- [11] Zuo MJ, Choy LF, Yam RCM. A model for optimal design of multi-state parallel-series systems. 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Edmonton, May 9–12, 1999. p. 1770–3.
- [12] Ross SM. Multivalued state component systems. *The Annals of Probability* 1979;7(2):379–83.
- [13] Block HW, Savits TH. A decomposition for multi-state monotone systems. *Journal of Applied Probability* 1982;19:391–402.
- [14] Cappelle B, Kerre E. Computer assisted reliability analysis: an application of possibilistic reliability theory to a subsystem of a nuclear power plant. *Fuzzy Sets and Systems* 1995;74:103–13.
- [15] Cybenko G. Approximation by superpositions of a sigmoidal function. *Math Control Signals Syst* 1989;2(4):303–14.
- [16] Aourid M, Do XD. NN approximation: application to peak switching over-voltage determination in power systems. *IEEE International Conference on NNs Conference Proceedings* 1995;1:200–4.
- [17] Mathworks, MATLAB, High-Performance Numeric Computation And Visualization Software: Reference Guide. Natick, MA, USA, 1992.
- [18] MATLAB® Version 6.0.0.88 Release 12, September 22, 2000, The MathWorks, Inc.
- [19] Scales LE. Introduction to non-linear optimization. New York: Springer, 1985.
- [20] Sarle WS. Stopped training and other remedies for overfitting. *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics* 1995. p. 352–60.

- [21] Weigend A. On overfitting and the effective number of hidden units. Proceedings of the 1993 Connectionist Models Summer School, 1994. p. 335–42.
- [22] Tetko IV, Livingstone DJ, Luik AI. Neural network studies. 1. Comparison of overfitting and overtraining. *J. Chem. Info. Comp. Sci.* 1995;35:826–33.

Peter X. Liu received the B.S. and M.S. degree in mechanical engineering from Northern Jiaotong University, Beijing, China. He is currently pursuing the Ph.D. degree at the Advanced Robotics and Teleoperation (ART) Laboratory, University of Alberta. His research interests include real-time data transmission in the Internet, Internet based teleoperation, softHaptics, and mobile robotics.

Ming Jian Zuo is Professor in the Department of Mechanical Engineering at the University of Alberta in Canada. He received a B.Sc. degree in Agricultural Engineering from Shandong Institute of Technology, China and M.Sc. and Ph.D. degrees in Industrial Engineering from Iowa State University, USA. His research interests include system reliability analysis, condition based maintenance modeling, and manufacturing systems.

Max Meng received his Ph.D. degree in Electrical and Computer Engineering from the University of Victoria in Canada. He is currently a Professor of Electrical and Computer Engineering and the Director of the Advanced Robotics and Teleoperation (ART) Laboratory at the University of Alberta in Canada. His research expertise is in the area of Robotics, Network Enabled Services, Intelligent and Adaptive Systems, and Human–Machine Interface, with medical, industrial, and military applications. He has published over 100 journal and conference papers. His research team invented the world’s first naturally moving prosthetic “robotic eye” actively controlled using human EOG signals. He is responsible for the introduction of the concept of “SoftHaptics”. He is an editor of the IEEE/ASME Transactions on Mechatronics.