PERGAMON

Computers & Industrial Engineering 36 (1999) 391–407

computers & industrial engineering

www.elsevier.com/locate/compindeng

Computing confidence intervals for stochastic simulation using neural network metamodels

Robert A. Kilmer[a,*], Alice E. Smith[b], Larry J. Shuman[b]

[a]*Business Information Systems, Messiah College, Grantham, PA 17027, USA*
[b]*Department of Industrial Engineering, University of Pittsburgh, 1031 Benedum Hall, Pittsburgh, PA 15261, USA*

Abstract

This paper discusses the use of supervised neural networks as a metamodeling technique for discrete-event, stochastic simulation. An (*s, S*) inventory simulation from the literature is translated into a metamodel through development of parallel neural networks, one estimating expected total cost and one estimating variance of expected total cost. These neural network estimates are used to form confidence intervals, which are compared for coverage to those formed directly by simulation. It is shown that the neural network metamodel is quite competitive in accuracy when compared to the simulation itself and, once trained, can operate in nearly real-time. A comparison of metamodel performance under interpolative versus extrapolative predictions is made.

Keywords: Confidence intervals; Neural networks; Metamodels

1. Introduction

1.1. Simulation and metamodeling

Discrete-event stochastic simulation is used to model many production systems where output cannot be predicted by other means. Inputs to the simulation are the decision variables of the system, while the simulation output is the outcome(s), or response(s), of the system. One of the drawbacks of discrete-event simulation is the amount of computational resources required to

* Corresponding author. Tel.: +1-717-766-2511; fax: +1-717-691-6057.
E-mail addresses: rkilmer@messiah.edu (R.A. Kilmer), aesmith@engrng.pitt.edu (A.E. Smith)

0360-8352/99/$ - see front matter
PII: S0360-8352(99)00139-4

fully explore system responses in terms of expected value and variance. In some situations the magnitude of computational effort will preclude timely and proper decision making. One such situation is selection of optimal decision variable values (*policies*), given response objectives and system constraints, i.e., simulation optimization. Another situation is analysis and decision making in fast paced, rapidly changing venues, such as hospital emergency departments, job shop scheduling or during military operations. In these situations there is simply no time to perform multiple replications for the selected values of the decision variables. To supplement discrete-event simulation models, researchers and practitioners have developed metamodels, which are approximations that perform satisfactorily and are significantly more computationally efficient.

To more explicitly define a simulation metamodel, let X_j denote the value of a factor j influencing the response, Y, of the physical system where $\{X_j | j = 1,2,\ldots, p\}$. The relationship between the response variable Y and the inputs X_j (this can be generalized to a system with more than a single response variable) is:

$$Y = f_1(X_1, X_2, \ldots, X_p) \tag{1}$$

A simulation model f_2 may consider only a subset of the factors of the system itself, $\{X_j | j = 1,2,\ldots, n\}$, where $n \leqslant p$. The response of the simulation Y' is defined as a function f_2 of the remaining input factors and an error term ε_s. The error term, ε_s, is composed of the effects of any excluded or unidentified factors and the modeling error of translating the physical system to the simulation model:

$$Y' = f_2(X_1, X_2, \ldots, X_n) + \varepsilon_s \tag{2}$$

A metamodel f_3 is a further simplification of the physical system, which is deterministic and employs a subset of the factors $\{X_j | j = 1,2,\ldots, m\}$, where $m \leqslant n$

$$Y'' = f_3(X_1, X_2, \ldots, X_m) + \varepsilon_s + \varepsilon_m \tag{3}$$

Y' is the deterministic response of the metamodel, ε_s is as defined above and ε_m is the composition of the error of the effects of any further excluded variables and the metamodeling error of fitting the metamodel to the simulation model (assuming a linear error function).

Simulation metamodeling research has been published since 1970 (e.g., [1–3]), although the descriptor *metamodel* was developed more recently by Kleijnen [4]. A current overview of published research on simulation metamodels for industrial and production applications can be found in Yu and Popplewell [5]. Applications cited include production planning and control, work in progress inventory management, FMS design, order level inventory control, facilities storage design and shop floor control [6–13]. The articles that described these applications concurred in the observation that "metamodels are easier to manage and provide more insight than simulation alone" [5].

The main issues in metamodeling are: (1) the choice of the underlying functional form, (2) the choice of input variables and their corresponding response variable to be used in the metamodel, (3) selection of the sample from the simulation model to construct the metamodel, and (4) validation of the metamodel [14]. Item (2) is specific to the simulated system, where the input(s) and response(s) may embrace the entire simulation, or may include a subset of the

variables and responses. Items (1), (3) and (4) are specific to the metamodel, and involve selection of the appropriate metamodeling technique, construction of the metamodel, and verification and validation of the completed metamodel. The metamodel might model only a local portion of the simulated system (e.g., a selected range of a decision variable) or may encompass the complete simulated system, a global metamodel. Tradeoffs between accuracy and computational expense and between local and global information must be considered when developing a simulation metamodel.

The most popular metamodeling approach in simulation involves the use of parametric polynomial regression models in response surface methods [14]. Following Eq. 3 above, a polynomial of order q is created using the summation of the power functions of each X_m:

$$Y'' = \sum_{j=1}^{m} \sum_{k=1}^{q} \beta_{kj} p_k(X_j) \tag{4}$$

While straightforward to implement, the performance of a polynomial response surface metamodel depends on the applicability of the polynomial functional forms to the simulation response. The main drawback of polynomial response surface metamodels is their lack of flexibility to achieve global fit; power functions are usually iteratively fitted to small regions of interest so many independently developed polynomials are used to describe the entire simulation response surface.

1.2. Applicability of neural networks for metamodeling

Multi-layered perceptron neural networks with nonlinear transfer functions offer universal function approximation capability based wholly on the data itself, i.e., they are purely empirical models that can theoretically mimic any relationship to any degree of precision [15,16]. For development of a metamodel, the ability to universally model any relationship is a significant advantage because it removes the possibility of pre-selecting an incorrect functional form. This means the component of ε_m of Eq. (3) (which comes from the metamodeling error) would be theoretically zero, that is, no error would be incurred in fitting the metamodel since a neural network can exactly mimic any simulation relationship.

There are other aspects of neural networks that are useful for metamodeling. Neural networks are not sensitive to deviations from traditional statistical model assumptions; constant error variance, Gaussian distribution of errors, no multi-collinearity of independent variables, and no autocorrelation. Neural networks can also accommodate a combination of continuous variables and discrete numeric variables (categorical or binary). Additionally, most neural network paradigms are global models, so a single neural network could be developed to model the entire simulation response surface. This differs from polynomial regression metamodeling, where the regression surface is fitted to a locality, i.e., a subset of the response surface. Finally, the parallel architecture provides robustness to data that is incomplete or contains errors, and offers compact, fault tolerant, real time performance when translated from a software platform to a hardware platform, such as a VLSI chip. This last aspect may seem irrelevant to simulation metamodeling, however, there may be hardware uses in specialized

decision making circumstances such as military activities, on-line production changes or space exploration.

While these attributes make neural networks candidates for metamodeling, in practicality, networks are limited in their approximation capability by finite and imperfect data sets, and by stochastic relationships. It is important to note that a trained neural network is a deterministic model, and therefore the simulation relationship modeled is a moment or statistic of the stochastic response, such as $E(Y)$ or $Var(Y)$. The accuracy and precision of a neural model will depend on the quantity and appropriateness of the data set used in training. For stochastic simulation metamodeling, there is computational expense incurred in obtaining each training point (replicated simulation runs at each proposed policy), therefore data sets will not usually be large. Relatively small data sets can lead to imprecision and inaccuracy in neural models, especially when the model is overfitted or used for extrapolative estimations. Neural networks can be easily overfitted because of their many degrees of freedom; neural networks tend to result in low bias, but high variance, models. (See Geman et al. [17] and White [18] for a discussion of the bias/variance dilemma relative to neural networks.) This can result in degradation in performance when the response is for regions of inputs dissimilar from those used to build the neural network model. This degradation may become particularly acute for extrapolation, a problem also inherent in higher order polynomial regression models [14], and any over-parameterized model.

1.3. Prior research in neural network metamodeling

Despite these caveats, neural networks offer a promising alternative to the current localized, polynomial metamodels, and a few attempts have been made to employ them for this purpose. Padgett and Roppel [19] presented a general overview discussion on the potential uses of neural networks for simulation modeling. Fishwick [20] published an attempt to use a neural network as an approximation of a deterministic simulation. His less-than-promising results seem to have primarily been caused by naive use of neural networks and a single experiment. Another use of neural network metamodeling in lieu of a response surface approach for deterministic systems was by Anjum et al. [21]. This paper used two backpropagation neural networks to estimate an unknown function and its derivative, respectively, in order to develop a response surface model, which was used for function optimization. Using a neural network to model simulations was the subject of brief papers by Pierreval [22] and Pierreval and Huntsinger [23]. They modeled two simplified manufacturing shops as stochastic pattern recognition and classification problems (discrete output). Additionally they investigated a continuously valued deterministic simulation system (a coffee percolator) by using a neural approximation (similarly to Fishwick [20] above). Badiru and Sieger [24] successfully used a neural network metamodel to predict future cash flow values for cost estimation when using the initial investment and the interest rate per time period as uniform random variables. Madey and co-workers [25,26] trained a neural network to recommend process settings for a machine tool system. The training data was taken from the inputs and outputs of a stochastic simulation of the tool system. A unique aspect of their work was the selection of training data that represented good choices of input values, i.e., an ad hoc optimization was performed using the simulation, and the results of the optimization provided the training set. Hurrion [27]

developed a metamodel of a stochastic simulation of a coal depot by training a neural network to output the mean and ±99% confidence limits of the time the depot remained open. The neural network was trained on the mean value of nine replications and was tested for interpolation performance. The authors of this paper used a neural network metamodel in an earlier study of a two variable inventory model to estimate expected total cost [28].

1.4. Aspects of modeling a stochastic system

Neural network research has not addressed the problem of modeling explicitly probabilistic relationships, as is found in stochastic simulation, in any depth. Using a neural network to such purpose requires careful consideration of several aspects that are not normally a part of neural network modeling. One aspect is the manner in which to translate a stochastic simulation to a deterministic metamodel. Usually, an expected value of the stochastic response will be modeled, however even this is not entirely straightforward. How many replications, r, should be used in calculating the expected value? Should the number of replications be constant, or should it vary according to some pre-determined criteria? A constant value of r must be set by balancing the computational expense of the replications with the desired accuracy of the estimates. This will be dependent on the exact stochastic characteristics of the simulation and the user will probably want to run some trial replications to gauge an appropriate level of r. If r varies, then it must be used as an input variable to the metamodel. This approach might seem appealing but would require observations using all desired values of r for each and every policy simulated, significantly expanding the tasks of generating the training and validation data sets and building the metamodel. Should other measures besides expected value be included in the metamodel? Candidates include the median, the higher moments and percentiles or fractiles of the response distribution. In stochastic simulation, these statistics can be calculated as by-products of multiple simulation runs. In a neural network, outputs are deterministic, so there is no attempt to reproduce the stochastic behavior of the simulation in the metamodel. (This is true of other metamodeling techniques as well.) However, it is desirable to calculate a measure of variability as well as expected value for any particular system response. An interval estimate of system response can then be used in decision making. When using multiple replications, what is the effect of the distribution of the responses for any particular policy? Do departures from normality and independence affect the metamodel, and if so, how?

A second aspect is the method of data preparation for training. For a stochastic system where responses differ for a single policy (i.e., a single input vector), training could be accomplished through presentation of all data (all replications) or through presentation of the calculated target only, e.g., the expected value. It might seem that these approaches are equivalent, and indeed for regression metamodeling, they are. However neural networks do not inherently use the sample mean for the target when presented with multiple output values for the same set of input values. The theories of neural networks are not yet sufficiently defined to allow for a definitive answer on how training on each replication Y might differ from training on expected value $\bar{Y}$. There are many other factors in neural network training that can be confounding, such as presentation order of the training set, initial weight set and network architecture. The authors postulated that the additional information of including each

replication might improve the knowledge gained by the neural network during training, while incurring virtually no additional computational expense. The primary contributor to computational expense is the number of simulation runs needed to develop the metamodel. The computational expense of actually developing the neural network and then running it is quite small compared to the simulation.

A third aspect is in the validation of the metamodel. How faithfully does the metamodel represent the stochastic simulation? How does the metamodel perform in decision-making tasks, such as establishing confidence intervals of competing policies? How do the precision and accuracy of the metamodel change over the response surface? Is it permissible to use the metamodel for small extrapolations? Extrapolation allows policies that extend beyond the range used in training the neural metamodel, whereas interpolation uses policies only within the ranges used to train the neural network. Extrapolation should be avoided if at all possible for any predictive model, including simulation metamodels. However, there are circumstances where mild extrapolation is called for.

This paper addresses many of the issues listed above and significantly expands the work cited in Section 1.3 by modeling the stochastic system using parallel neural network metamodels for estimation of $\mathrm{Var}(\bar{Y})$, along with $\mathrm{E}(Y)$. Using expected value and variance of the estimate together allows the construction of statistical confidence intervals for system output. This paper compares training using the individual responses, Y, to training using the calculated expected value, $\bar{Y}$. Accuracy and precision of the metamodel over the response surface, including limited extrapolation, is studied in depth, and reported in Section 4. Five test sets with increasing degrees of extrapolation, ranging from pure interpolation to pure extrapolation, were investigated. The issue of replications is not treated experimentally herein, nor is the effect of the distribution of responses other than normal, but both are important open research issues.

2. The inventory simulation

The production system selected for study was taken from the experimental design and optimization chapter of Law and Kelton [29]. The system is a lot size–reorder point system, an (s, S) inventory system where s = "reorder point" quantity and S = "order up to" quantity. The inventory system is representative of inventory systems for a single product. The decision variable is how many items to obtain from the supplier in each of the next n time periods in order to minimize the total cost, Y, of the inventory system. A stationary (s, S) policy is used to decide how much to order at the beginning of each time period, e.g., given that I is the inventory at the beginning of the month, then the amount to order for that time period, Z, is determined by:

$$Z = \begin{cases} S - I & \text{if } I < s \\ 0 & \text{if } I \geqslant s \end{cases} \tag{5}$$

The parameters of the simulation are the following:

[#] Input variable to Law and Kelton's original simulation.

*Input variable to this simulation and the metamodel.
+ Random variable.
@ Output variable.
$X^{\#*}$ = reorder point,
X_2 = order up to quantity,
$X_3^{\#*} = X_2 - X_1$ = reorder quantity,
X_4^* = setup cost of order,
X_5 = stockout/backorder cost,
$X_6^* = X_4/X_5$ (parameterization of stockout/backorder cost),
R_1^+ = size of demands, $\sim D = \{1$ with probability $\frac{1}{6}$, 2 with probability $\frac{1}{3}$, 3 with probability $\frac{1}{3}$, 4 with probability $\frac{1}{6}$
R_2^+ = time between demands, $\sim$exp ($\beta = 0.1$ time period),
R_3^+ = time lag for delivery, $\sim U[0.5, 1.0]$,
C_1 = time between order decisions, $\equiv 1$,
C_2 = initial inventory level, $\equiv 60$,
C_3 = incremental cost per item ordered, $\equiv 3$,
C_4 = holding cost per item, $\equiv 1$ per time period,
n = time horizon for analysis purposes, $\equiv 120$ per time periods,
$Y^@$ = total cost of the inventory policy.

The inventory system computer simulation was written in the SIMAN simulation language and verified against the original Law and Kelton [29] simulation, which held all input parameters fixed except for X_1, reorder point, and X_3, reorder quantity. To be consistent with Law and Kelton [29], ten replications at each policy were used to estimate the system response, Y, total cost. All constants (C_i), random variables (R_i) and n were taken directly from Law and Kelton [29]. This paper considers a more complex version of the simulation by allowing X_4 (setup cost) and X_6 (stockout cost) as input variables, along with X_1 and X_3.[1] The addition of X_4 and X_6 as input variables was made because of the uncertainty concerning these costs. Therefore the metamodel developed could be used to optimize the decision variables, X_1 and X_3, given altering values of X_4 and X_6.

3. The neural network metamodels

The training set of a supervised neural network consists of a group of size m training pairs, where each training pair consists of a value for each input variable (a policy) and a target value for each response variable. The target value is the estimate from the simulation itself. For neural network metamodeling, the inherent assumption is that the simulation output is the correct answer, and thus any deviation of the simulation from the actual system is disregarded. For the research reported in this paper, the number of replications, r, per set of input values was kept at the same value chosen by Law and Kelton [29], viz., ten, and two target outputs were used. The first was $E(Y)$ and the second was $\mathrm{Var}(\bar{Y})$, both over r replications.

[1] Law and Kelton [29] used setup cost = 32 and stockout cost = 5, both constants.

The selection of values for the candidate policies is problem dependent. This paper uses a uniform discretized lattice over the allowable ranges of the four input variables; however more sophisticated methods such as *k-p* designs [29], Latin hypercube sampling [30] or Bayesian techniques [31] may well improve neural network metamodels. The training data was designed

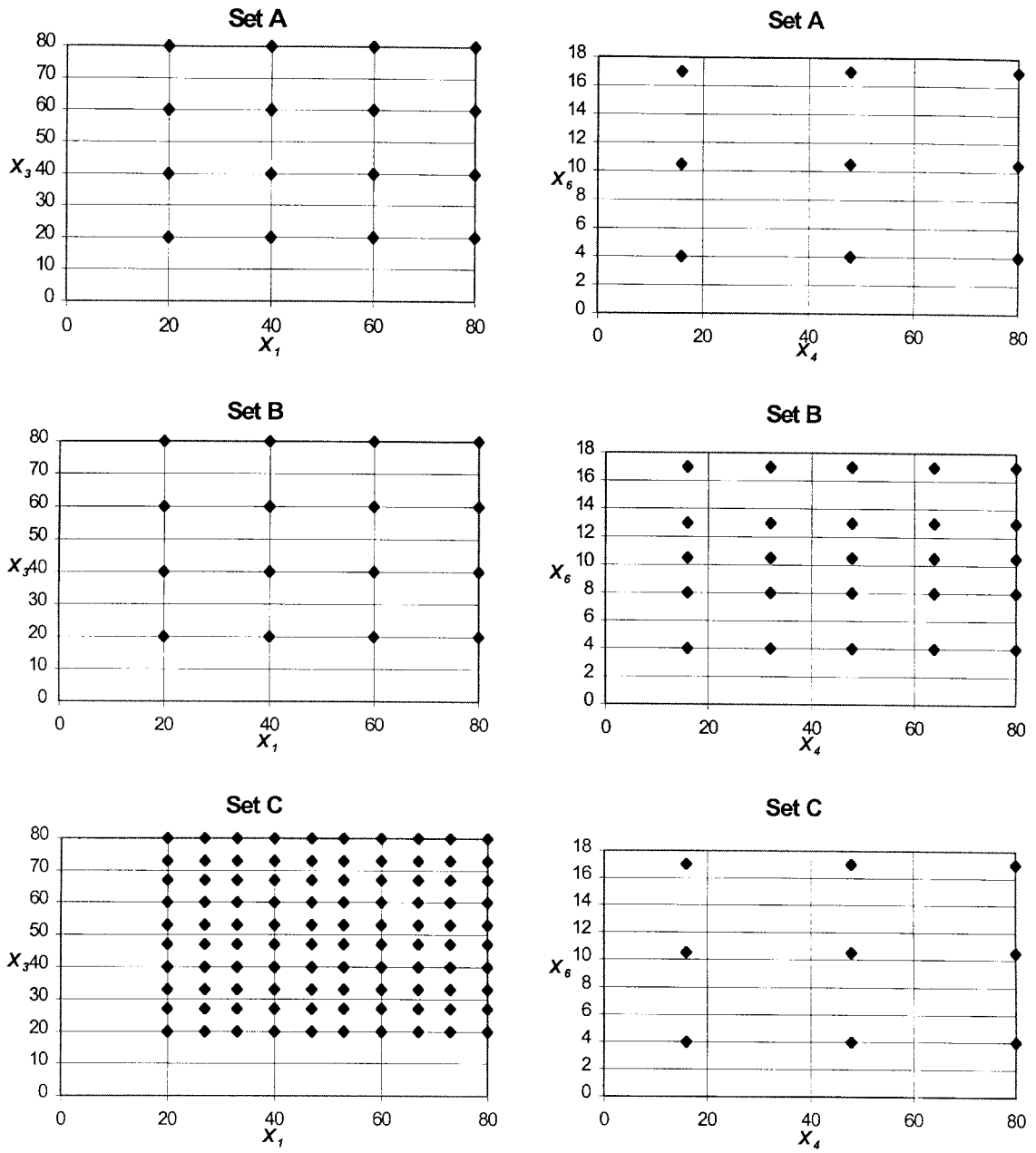

Fig. 1. Training sets A ($n = 144$), B ($n = 400$) and C ($n = 900$) for parameters X_1, X_3, X_4 and X_6.

using various combinations of the input variable settings as shown in Fig. 1. Thirteen training sets, ranging from 16 to 2500 training pairs, were investigated but results are reported for only three representative ones. (See Kilmer [32] for complete details on results of all training sets.) These training sets are labeled A, B and C, and include 144, 400 and 900 training pairs, respectively. The training sets differed in the number of observations in each, not in the ranges of the variables. Two approaches for training a neural network for $E(Y)$ are: (1) using $\bar{Y}$ calculated over r replications for the target and (2) using Y of each of the r replications as targets. This replication training (2) requires a training set size of $m \times r$, whereas the mean value training (1) requires a training set size of only m. For the neural network estimate of $\mathrm{Var}(\bar{Y})$, the target was $\frac{s^2(Y)}{r}$.

The test set was constructed as shown in Fig. 2. There are 900 ($5 \times 5 \times 6 \times 6$) points in the

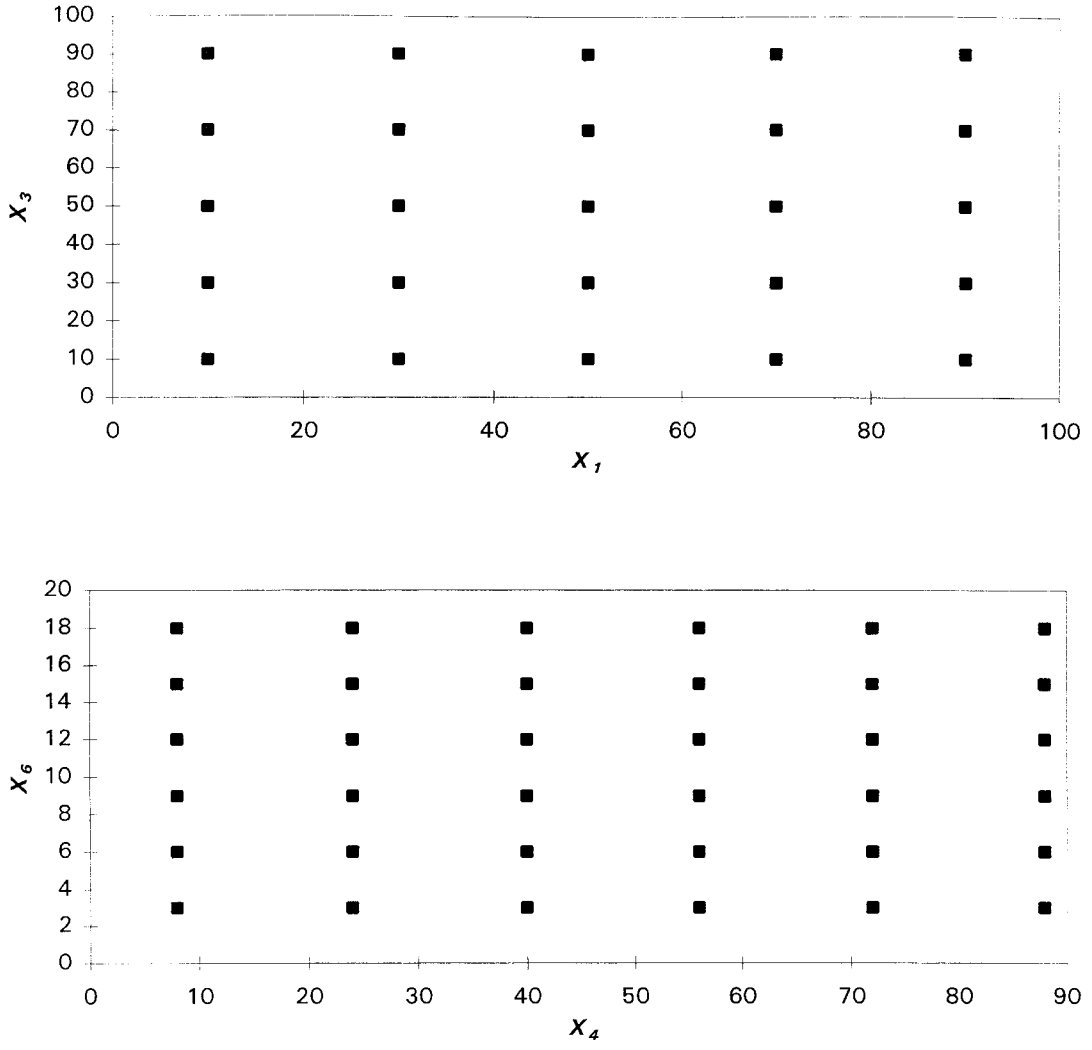

Fig. 2. Total testing set ($n = 900$).

entire test set. Again, the test set was chosen from a uniform discretized lattice over the ranges each input variable, however it also extended beyond the training set ranges. This was done purposely to investigate neural network metamodel performance for extrapolation as well as interpolation. Because it was desired to ensure the integrity of the targets of the test set, each testing pair had 10 sets of 10 replications run (100 total replications). The "correct" output was $\overline{Ymacr}$ of the 10 sets for $\mathrm{E}(Y)$. For $\mathrm{Var}(\bar{Y})$,

$$\frac{\sum_{i=1}^{10}\left(\frac{s_i^2(Y)}{r}\right)}{10}$$

was used as the "correct" answer. Each testing pair can be categorized in terms of the number of input variables whose values are beyond the range of the training sets, i.e., in the extrapolation range. This divides the test set into five groups ranging from wholly interpolative (test set 0) to wholly extrapolative (test set 4), as shown in Table 1. Test sets 1, 2 and 3 contained observations that were partially extrapolations and partially interpolations relative to the training sets. For test set 1, one value of the input parameters was extrapolated beyond the training set, while for test sets 2 and 3, two and three values of the input parameters, respectively, were extrapolations.

Rather than form one neural network with simultaneous predictions of $\mathrm{E}(Y)$ and $\mathrm{Var}(\bar{Y})$, a parallel network architecture, as shown in Fig. 3, was used. This allowed training to be conducted separately and independently for each output. In neural network training, multiple output variables can interfere with the learning of each, and this parallel architecture alleviates that undesirable phenomenon. After experimentation, an architecture of four input neurons, two hidden layers of five neurons each, and a single output neuron was selected for each network. The network architecture chosen was the smallest that could adequately learn the relationship to minimize the probability of overfitting. A traditional backpropagation training algorithm [33] with a smoothing factor and a unipolar sigmoidal transfer function was used, and networks were trained to a normalized error tolerance of 2% or to a maximum of 750 epochs (passes through the training set). The networks were evaluated after each epoch, and the one that performed best on the training set was saved as the final trained network; the final network was generally trained within the first 50 epochs. After experiencing problems with consistent undershoot, the normalization range was expanded to $\pm 10\%$ beyond the range of

Table 1
Subdivisions of test sets

Test set	Number of testing pairs
All internal—pure interpolation (0)	144
One external value (1)	336
Two external values (2)	292
Three external values (3)	112
Four external values—pure extrapolation (4)	16
Total testing pairs	900

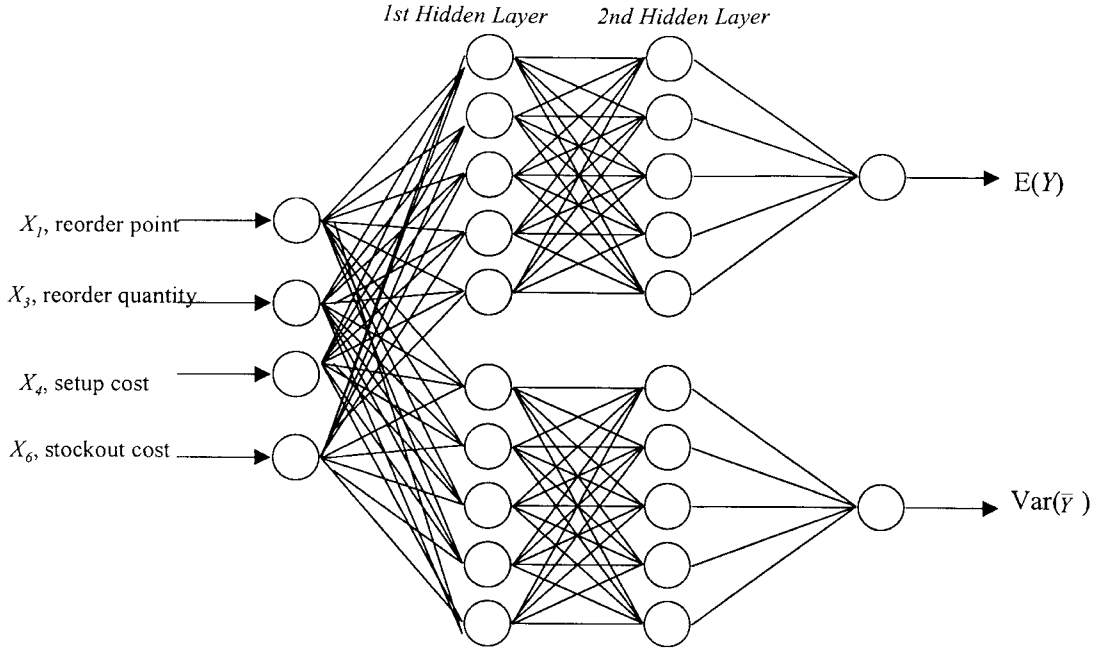

Fig. 3. Parallel neural network metamodel to estimate E(Y) and Var($\bar{Y}$).

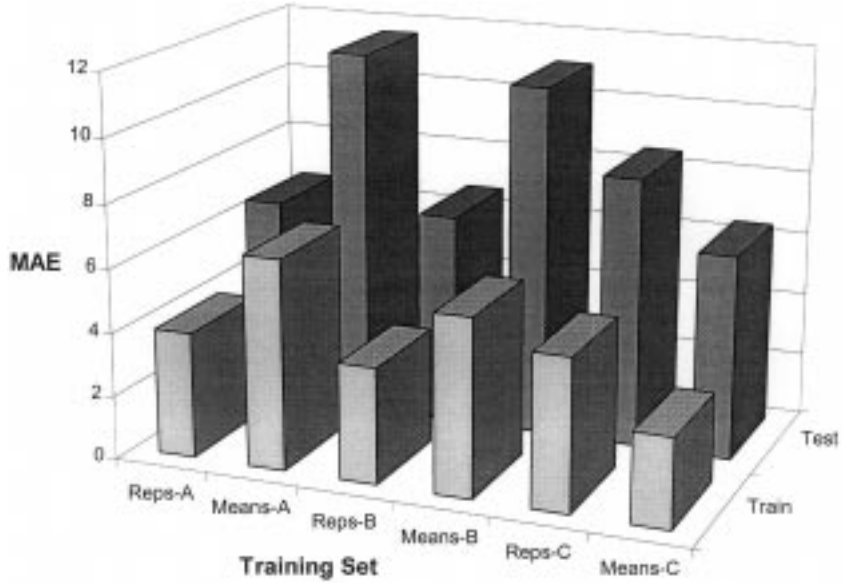

Fig. 4. Mean absolute error for neural networks estimating E(Y) using training sets A, B and C and the entire test set (n = 900), where ''Reps'' are neural networks trained on the responses of each replication and ''Means'' are neural networks trained on the average response over the ten replications.

the training data for both the inputs and the outputs. Undershoot is a typical neural network behavior when using a sigmoid transfer function which biases the output to be less extreme than desired and expanding the normalization range is a typical solution. Normalizing data is necessary for proper handling by the sigmoidal transfer function, whose range of activation is between 0 and 1. Eq. (6) was used to normalize to the expanded range:

$$\text{norm } X_{ij} = \left(\frac{X_{ij} - (1.1 \cdot X_{j,\ \min})}{1.1 \cdot (X_{j,\ \max} - X_{j,\ \min})} \right) \tag{6}$$

4. Results

4.1. Prediction of expected total cost

As postulated, there were fundamental differences in the performance of the networks trained on the single value $\bar{Y}$ versus the r values of Y. The replication networks were more consistent in their ability to accurately predict $E(Y)$ for the test set, except for networks trained on the largest data set (set C), as shown in Fig. 4. Y ranged in value from 88 to 512; the test set mean absolute error (MAE) for training sets A and B (replications) was about 2.2%. An encouraging result for replication training is that networks trained on smaller data sets perform as accurately on the test data (i.e., generalization) as the networks trained on larger data sets. A reduction in training set size requires less simulation runs by a factor of each training pair times r. For example, training set A required 75,600 fewer simulation runs than training set C, however for replication training, the performance on the entire 900 testing pairs was nearly equivalent.

The issue of Y versus $\bar{Y}$ training deserves further attention. The arithmetic mean, $\bar{Y}$, from r replications gives equal probability mass $(1/r)$ to each Y. A replication representing an outlier will strongly influence the mean. The neural network does not necessarily mimic mean value behavior. Myopic gradient descent (the backpropagation training algorithm) will move to an error surface location without necessarily assigning equal probability mass to each observation. The network may find weights which better reflect $E(Y)$ by ignoring (or partly ignoring) an extreme Y. Certainly, replication training gives more information to the network, since none of the simulation output data is discarded. This extra information is more important for a relatively sparse training set, where each new training pair imparts more information to the model (taking an information theoretic view) than it would from a larger training set.

Another validation criteria to examine is the degradation in accuracy from the training set to the testing set. This is an indication of the ability of the network to generalize. While all networks decrease in accuracy for the test sets (as is expected), the networks trained on replications degrade to a lesser degree. The performance on the test set is indicative of the performance to be expected during operation of the metamodel for optimization, "what if" analysis, model aggregation, or for whatever task it is put to use. The performance of a neural network on the training set signifies little, except poor performance may indicate under-parameterization (too few weights) or under-training (too few iterations through the training set). Good performance on the training set may give little indication of generalization ability.

4.2. Predicting the variance of expected total cost

Fig. 5 gives the training and entire test set results for the variance neural networks. The range of $\hat{\mathrm{V}}\mathrm{ar}(\bar{Y})$ was from 0.03 to 48.93; the test set MAE of all training sets was about 2.3%. The predictions did not degrade much from the training set to the testing set, indicating good generalization ability. Another observation from Fig. 5 is that for generalization, a smaller training set was just as effective as a larger training set. This was an encouraging result, in that despite the large variation in $\hat{\mathrm{V}}\mathrm{ar}(\bar{Y})$ the neural network metamodel could adequately learn the relationship with a small data set.

4.3. Forming confidence intervals

To evaluate the overall performance of the neural network metamodels, the neural network estimates of $\mathrm{E}(Y)$ and $\mathrm{Var}(\bar{Y})$ were combined to form confidence intervals. Confidence intervals are important to determine the suitability of the neural network metamodel for typical tasks, because interval estimates are much more useful than point estimates for decision-making. For comparison, confidence intervals were also constructed for each testing pair directly from additional simulation data using ten replications. Both intervals took the form of:

$$\text{Interval} = \mathrm{E}(Y) \pm t_{\alpha/2,\ r-1} * \sqrt{\hat{\mathrm{V}}\mathrm{ar}(\bar{Y})} \tag{7}$$

where $\mathrm{E}(Y)$ and $\hat{\mathrm{V}}\mathrm{ar}(\bar{Y})$ were obtained either through neural network prediction or by r replications of the simulation, and $t_{\alpha/2,r-1}$ is the Student t distribution value for a given level

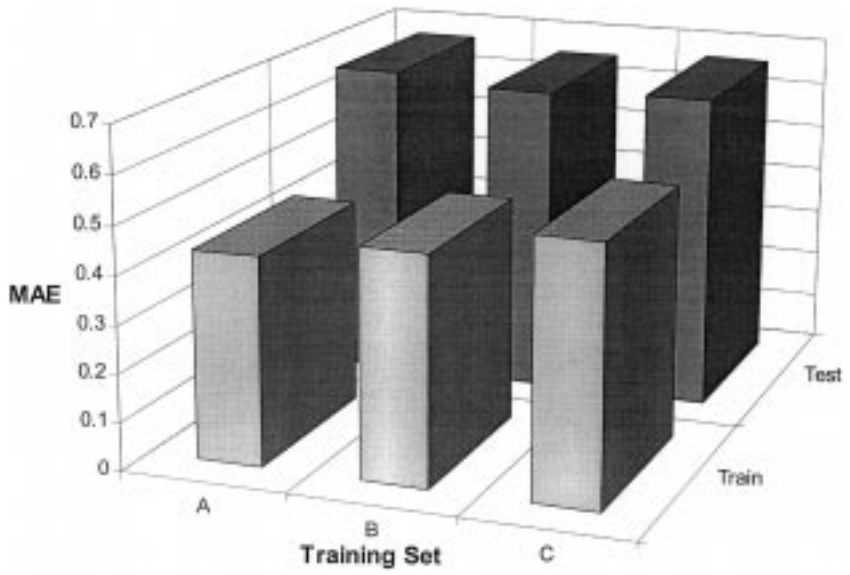

Fig. 5. Mean absolute error for neural networks estimating $\mathrm{Var}(\bar{Y})$ using training sets A, B and C and the entire test set (n = 900).

of significance (α). The normality and independence of the responses, Y, were verified [32] so that the necessary statistical assumptions held.

The coverage obtained directly from the simulation was compared with the neural network coverage by generating 100 additional simulation replications for each testing pair, and then tallying the number of these which fell within the bounds of each interval. For an ideal confidence interval, a count of replications equal to the confidence level would fall within the interval, with equal counts falling on either side of the interval. For example, a 90% confidence interval ($\alpha = 0.10$) should include 90 of the 100 replications within its bounds, with 5 replications falling above the interval and 5 replications falling below the interval, for each testing pair.

While the neural network confidence intervals were not as accurate as those generated directly by the simulation, for interpolation they were quite reasonable. The goodness of the neural network intervals was more dependent on the quality of the neural prediction of $E(Y)$, than that of the prediction of $Var(\bar{Y})$. The networks that did poorly for estimation of total cost also produced poor confidence intervals, as might be expected. In general, these were the networks trained on $\bar{Y}$, rather than on replications of Y. Table 2 shows the coverage of the 100 replications averaged over the 144 testing pairs of the wholly interpolative test set (test set 0) for 90, 95 and 99% confidence levels for both neural network confidence intervals and for confidence intervals obtained directly from the simulation.

Two further observations from Table 2 can be made. First, the neural network intervals are not likely to be correctly centered, that is, they are biased upwards or downwards. Since the confidence intervals themselves are symmetric, this is an effect of the prediction of $E(Y)$ being either high or low. Second, the neural network confidence intervals improve as α decreases (and the interval widens). This happens because the estimates of the neural network for total cost are close to the target. As the interval widens, the neural network metamodel becomes very similar to the simulation itself for constructing interval estimates.

To examine the effect of extrapolation, Table 3 shows the results of training set A using replication training. Test sets 0–4 results are shown for the network and for the simulation at a

Table 2
Confidence interval coverage for wholly interpolative test set (0)

		Neural nets—Y replications			Neural nets—$\bar{Y}$			Simulation intervals		
Train set	Confidence level	Low	Interval	High	Low	Interval	High	Low	Interval	High
A	90	1.4	83.0	15.6	39.9	57.4	2.7	5.6	88.9	5.5
A	95	0.4	91.3	8.3	29.3	69.5	1.2	2.7	94.3	3.0
A	99	0.0	98.9	1.1	10.8	89.1	0.1	0.6	98.8	0.7
B	90	4.6	82.3	13.1	64.0	33.8	2.2	5.6	88.9	5.5
B	95	2.1	90.6	7.3	55.8	43.0	1.2	2.7	94.3	3.0
B	99	0.2	98.6	1.2	34.6	65.2	0.2	0.6	98.8	0.7
C	90	12.1	83.2	4.7	3.0	89.2	7.8	5.6	88.9	5.5
C	95	6.0	91.5	2.5	1.2	95.6	3.2	2.7	94.3	3.0
C	99	0.7	99.0	0.3	0.1	99.7	0.2	0.6	98.8	0.7

Table 3
95% Confidence interval results for training set A using replications

	Neural network			Simulation		
Test set	Low	Interval	High	Low	Interval	High
0	0.4	91.3	8.3	2.7	94.3	3.0
1	3.0	81.0	16.0	2.6	93.5	3.8
2	7.2	67.3	25.5	3.1	93.9	3.1
3	13.3	51.4	35.3	2.8	93.5	3.7
4	18.3	37.3	44.0	1.8	94.9	3.4
Ideal	2.5	95.0	2.5	2.5	95.0	2.5

95% confidence level. Of course, since the simulation is generated directly, there is no extrapolation. However, the neural network estimates uniformly degrade as more extrapolation is required. This highlights the imprudence of using metamodels for extrapolation, especially when multiple simulation inputs are extrapolated and the metamodeling technique has many free parameters.

5. Conclusions

Discrete-event stochastic simulation is used to model many production environments. However, the model itself is not the motivation for the development of these simulations, rather it using the model to make decisions concerning the production system. Decision-making using simulation models can be time consuming because of the computational effort of running the simulation many times before a decision can be made. This has motivated the field of simulation metamodeling, with recent interest in expanding beyond low order, localized polynomial models. There are fundamental reasons for using a neural network approach to simulation metamodeling. These include universal approximation ability which eliminates the search for the most appropriate metamodeling technique and error due to functional assumptions, the ability to simultaneously include continuous and discrete variables without screening for multi-collinearity or autocorrelation, and model robustness to parameter selection (e.g., architecture, training rate) and sample data. Further computational motivations include operation in real time in hardware format and extremely fast software operation, with additional speed increase possible through the use of parallel computers.

The neural network approach has several significant drawbacks. The first is that a neural network metamodel is a ''black box''. Weights cannot readily be interpreted and the user must accept the neural metamodel ''as is''. A ''black box'' is acceptable in many situations, but it does necessitate a complete validation of the model prior to use. Due to the nature of neural networks, this validation must also be empirical and include data not used to construct the model. To minimize the number of simulation runs needed to build and verify a neural network metamodel the use of statistical resampling techniques, such as the bootstrap or grouped cross validation, may be beneficial. See Twomey and Smith [34] for a discussion of

these techniques adapted for neural networks. Overfitting of neural networks is a common pitfall where the model does extremely well on the data used to construct it, but has poor generalization capability.

For the inventory study problem, training on the individual replications, Y, yielded more precise predictions of expected values for smaller data sets at the expense of longer training times (the size of the training set for replications was ten times the size of the training set for $\bar{Y}$). This study also indicated model robustness to large decreases in the size of the training set. Combining estimates of $E(Y)$ with $Var(\bar{Y})$ enables calculation of metamodel confidence intervals, a novel contribution of this research. This allows decision-makers to consider uncertainty when using the metamodel, just as would be done when using the simulation model directly. Statistics other than $\hat{V}ar(\bar{Y})$ could have been used in the metamodel, e.g., percentiles or the higher moments. While the accuracy of the parallel neural metamodel on interpolative predictions was encouraging, clearly more investigation needs to be done on the trade-offs of precision and computational effort. A study by the authors on a larger simulation of a hospital emergency department using the techniques developed in this paper confirmed the general findings [35]. Other important areas for further research are using variable r and consideration of non-normal responses.

Acknowledgement

Alice E. Smith gratefully acknowledges financial support from the US National Science Foundation CAREER grant DMI-9502134.

References

[1] Mihram GA. An efficient procedure for locating the optimal simulation response. In: Proceedings of the Fourth Conference on the Applications of Simulation, 1970. p. 154–61.
[2] Racite MP, Lawlor FD. Techniques for the development of empirical subsystem models. In: Proceedings of the 1972 Summer Simulation Conference, 1972. p. 155–62.
[3] Biles WE. A gradient-regression search procedure for simulation experimentation. In: Proceedings of the 1974 Winter Simulation Conference, 1974. p. 490–7.
[4] Kleijnen JPC. Statistical tools for simulation practitioners. New York: Marcel Dekker, 1987.
[5] Yu B, Popplewell K. Metamodels in manufacturing: a review. International Journal of Production Research 1994;32:787–96.
[6] Friedman LW. The multivariate metamodel in queuing system simulation. Computers and Industrial Engineering 1989;16/2:329–37.
[7] Ghosh JB. Evaluating multiprocessor performance: an exercise in metamodeling. Simulation 1988;51/2:70–4.
[8] Kleijnen JPC. Experimental design and regression analysis in simulation: an FMS case study. European Journal of Operational Research 1988;33:257–61.
[9] Li L, Cochran JK. Metamodels of production line transient behaviour for sudden machine breakdowns. International Journal of Production Research 1990;28/10:1791–806.
[10] Madu CN. Simulation in manufacturing: a regression metamodel approach. Computers and Industrial Engineering 1990;18/3:381–9.
[11] Madu CN, Chanin MN. A regression metamodel of a maintenance float problem with Erlang-2 failure distribution. International Journal of Production Research 1992;30/4:871–85.

[12] Rotmans J, Vrieze OJ. Metamodelling and experimental design: case study of the Greenhouse effect. European Journal of Operational Research 1990;47:317–29.
[13] Starbird SA. A metamodel specification for a tomato processing plant. Journal of the Operational Research Society 1990;41/3:229–40.
[14] Barton RR. Metamodels for simulation input–output relations. In: Proceedings of the 1992 Winter Simulation Conference, 1992. p. 289–99.
[15] Funahashi K. On the approximate realization of continuous mappings by neural networks. Neural Networks 1989;2:183–92.
[16] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks 1989;2:359–66.
[17] Geman S, Bienenstock E, Doursat R. Neural networks and the bias/variance dilemma. Neural Computation 1992;4:1–58.
[18] White H. Learning in artificial neural networks: a statistical perspective. Neural Computation 1989;1:425–64.
[19] Padgett ML, Roppel TA. Neural networks and simulation: modeling for applications. Simulation 1992;58:295–305.
[20] Fishwick PA. Neural network models in simulation: a comparison with traditional modeling approaches. In: Proceedings of the 1989 Winter Simulation Conference, 1989. p. 702–10.
[21] Anjum MF, Tasadduq I, Al-Sultan K. Response surface methodology: A neural network approach. European Journal of Operational Research 1997;101:65–73.
[22] Pierreval H. Training a neural network by simulation for dispatching problems. In: Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering, 1992. p. 332–6.
[23] Pierreval H, Huntsinger RC. An investigation on neural network capabilities as simulation metamodels. In: Proceedings of the 1992 Summer Computer Simulation Conference, 1992. p. 413–7.
[24] Badiru AB, Sieger DB. Neural network as a simulation metamodel in economic analysis of risky projects. European Journal of Operational Research 1998;105:130–42.
[25] Madey GR, Weinroth J. Neural networks and general purpose simulation theory. In: Proceedings of the International Joint Conference on Neural Networks II, 1990. p. 647–50.
[26] Madey GR, Weinroth J, Shah V. Integration of neurocomputing and system simulation for modeling continuous improvement systems in manufacturing. Journal of Intelligent Manufacturing 1992;3:193–204.
[27] Hurrion RD. Using a neural network to enhance the decision making quality of a visual interactive simulation model. Journal of the Operational Research Society 1992;43:333–41.
[28] Kilmer RA, Smith AE. Using artificial neural networks to approximate a discrete event stochastic simulation model. In: Dagli CH, Burke LI, Fernandez BR, Ghosh J, editors. Intelligent engineering systems through artificial neural networks, vol. 3. New York: ASME Press, 1993. p. 631–6.
[29] Law A, Kelton D. Simulation modeling and analysis. New York: McGraw-Hill, 1991.
[30] Lunani M, Sudjianto A, Johnston PL. Generating efficient training samples for neural networks using Latin hypercube sampling. In: Dagli CH, Akay M, Chen CLP, Fernandez BR, Ghosh J, editors. Intelligent systems through artificial neural networks, vol. 5. New York: ASME Press, 1995. p. 209–14.
[31] Belue LM, Bauer Jr KW, Ruck DM. Selecting optimal experiments for multiple output multilayer perceptrons. Neural Computation 1997;9:161–83.
[32] Kilmer RA 1994 Artificial neural network metamodels of stochastic computer simulations. Ph.D. Dissertation, University of Pittsburgh.
[33] Werbos PJ 1974 Beyond Regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. Thesis, Harvard University.
[34] Twomey JM, Smith AE. Bias and variance of validation methods for function approximation neural networks under conditions of sparse data. IEEE Transactions on Systems, Man, and Cybernetics, Part C 1998;28/3:417–30.
[35] Kilmer RA, Smith AE, Shuman LJ. An emergency department simulation and a neural network metamodel. Journal of the Society for Health Systems 1997;5/3:63–79.