Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Simulation Modelling Practice and Theory 12 (2004) 559–578

SIMULATION MODELLING PRACTICE AND THEORY

www.elsevier.com/locate/simpat

A comparison of experimental designs in the development of a neural network simulation metamodel

Fasihul M. Alam *, Ken R. McNaught, Trevor J. Ringrose

Applied Mathematics and OR Group, Department of Engineering Systems, Cranfield University, RMCS Shrivenham, Swindon SN6 8LA, UK

Received 31 October 2002; revised 1 May 2003; accepted for publication 22 October 2003
Available online 25 August 2004

Abstract

In this case study, we investigate the effects of experimental design on the development of artificial neural networks as simulation metamodels. A simple, deterministic combat model developed within the paradigm of system dynamics provides the underlying simulation. The neural network metamodels are developed using six different experimental design approaches. These include a traditional full factorial design, a random sampling design, a central composite design, a modified Latin Hypercube design and designs supplemented with domain knowledge. The results from this case study show how much impact the experimental design chosen for the neural network training set can have on the predictive accuracy achieved by the metamodel. We compare the networks in terms of various performance measures. The neural network developed from the modified Latin Hypercube design supplemented with domain knowledge produces the best performance, outperforming networks developed from other designs of the same size.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Latin Hypercube design; Artificial neural network; Metamodel; Statistical experimental design

* Corresponding author.
E-mail address: m.f.alam@rmcs.cranfield.ac.uk (F.M. Alam).

1569-190X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2003.10.006

1. Introduction

Today, simulation is a popular tool for the analysis and/or design of existing or proposed complex systems. This popularity is partly due to its flexibility, and to its ability to model real-world systems in some detail, which, in turn, leads simulation to be used as a decision support tool in supervising and controlling the underlying system. Though simulation models require fewer restrictive assumptions than mathematical models when representing complex, dynamic systems, these models themselves are usually fairly complex and of relatively high dimensionality. That is, the performance of a simulation model mostly depends on a large number of parameters or factors that act and interact in a complex manner. However, with simulation modelling, the relationships between the design parameters and their resulting performance measures are not explicitly known. Therefore, simulation modelling becomes a trial-and-error process in which a set of input factors is used to predict a set of output performance measures. If the desired performances are achieved, a good system design has been attained, otherwise the process is repeated until a satisfactory set of performance measures is obtained. Unfortunately, the iterative nature of this process can result in both high computing costs and difficulties in interpretation and prediction of the results. In order to overcome these problems, researchers sometimes turn to a simpler, auxiliary model, or *metamodel*, to facilitate an understanding of the relationships between a system's inputs and its outputs. One of the main objectives of a simulation metamodel is to accurately represent the input–output relationships over wide ranges of the parameter space, while being computationally more efficient than the underlying simulation model [1]. This paper further investigates the methodology for developing artificial neural network (ANN) based simulation metamodels, which is not yet well established. A simple, deterministic combat model, developed by Coyle [2] within the paradigm of system dynamics is the underlying simulation model used in the study. Data sets required for six different experimental design approaches are generated from the simulation model and a series of neural network metamodels is then developed from these training sets. The results suggest that experimental design has a significant role to play in developing neural network simulation metamodels.

2. ANNs and their applicability to simulation metamodelling

2.1. ANN architecture

The network is organised in three distinct layers: the input layer, the hidden layer(s) and the output layer. A hidden layer is used to help in extracting higher-level features and to facilitate generalisation of outputs if the relationship between the input and output variables is non-linear. For a given input vector, it generates the output vector by a forward pass. The data are fed to the network at the input layer, and propagated with weights and activation functions to the output layer to provide the response. After presenting the sets of inputs and associated outputs, the network is able to 'learn' the relationships between them by changing the weights of its connections. Then, the mean squared error (MSE), the difference between the network output vector and the known target vector, is computed and back-propagated through the ANN to modify the weights for the entire network, a process referred to as *training*. Learning can be of three types—supervised, unsupervised or reinforcement. One of the most popular methods practised for supervised training of neural networks is the back-propagation training algorithm [3]. The neural networks that are trained by this method are called multilayer feed-forward networks. Those described in this paper use the back-propagation training algorithm because of its outstanding track record in successfully solving a variety of application problems [4]. Multilayer perceptron neural networks are able to learn the behaviour of a system from a training set consisting of examples of the form:

$$\langle \text{input}_1, \text{input}_2, \ldots, \text{input}_p \rangle \Rightarrow \langle \text{output}_1, \text{output}_2, \ldots, \text{output}_q \rangle.$$

Once the network has been trained according to the assigned learning rule, it is able to compute output values associated with new input vectors. Hence, neural networks are simply a set of techniques for iteratively fitting non-linear (usually multivariate) functions. They are therefore often useful in similar situations to regression models, splines, etc.

2.2. Applicability of ANNs to simulation metamodelling

In recent years, computer simulation 'optimisation' has often been conducted via response surface methodologies (RSM) using parametric regression model approximations of the simulation model. The regression model approach in RSM had been used successfully for performing sensitivity analyses within a limited region of the parameter space and determining solutions satisfying various constraints [5]. However, this parametric approach has not been used to perform global approximation of a simulation model because of its inability to provide a globally accurate fit to the response functions. Myers et al. [6] stated: "There appears to be some need for the development of non-parametric techniques in RSM. Most of our analytical procedures depend on a model. The use of model-free techniques would avoid the assumption of model accuracy or low-order polynomial approximation and, in particular, the imposed symmetry associated with a second-degree polynomial". One possible non-parametric approach is to use artificial neural networks. As Padgett and Roppel [7] commented "Neural networks in all categories address the need for rapid computation, robustness, and adaptability. Neural models require fewer assumptions and less precise information about the systems modelled than do some more traditional techniques."

Kilmer [8] mentioned some basic ways in which artificial neural networks could be used to aid in the analysis of combat simulation models. Using ANNs as direct models of physical phenomena is one way to take advantage of their inherently parallel processing nature. One example given by Hedgepeth and Jacobs [9] considers the use of historical combat results to train an ANN. Furthermore, ANNs can be used to learn decision rules within combat models, with an example being given by Morrison

[10], who used the combat simulation model CASTFOREM (Combined Arms and Support Task Force Evaluation Model). After training with the input–output vectors from many simulation runs, ANNs can be used in prediction, optimisation, sensitivity analysis and as a decision tool of the underlying simulation model. This latter approach essentially involves employing ANNs as combat simulation metamodels. Fig. 1 shows a diagram of artificial neural network simulation metamodel development.

Neural networks have also been used in business, industry and science, especially for purposes such as pattern classification, financial analysis and optimisation [11]. There have been several articles on direct uses of neural networks in simulation. Chryssolouris et al. [12] used neural networks and simulation modelling to design a job shop with the objective of estimating the number of machines required at each work centre. Chryssolouris et al. [13] carried out a similar study to determine the operational policies needed to achieve certain performance levels in terms of mean task cost, mean task flow time and mean task tardiness for workloads at four different work centres. Padgett and Roppel [7] presented a general discussion of the effective uses of neural networks in simulation modelling. Mollaghasemi et al. [14]

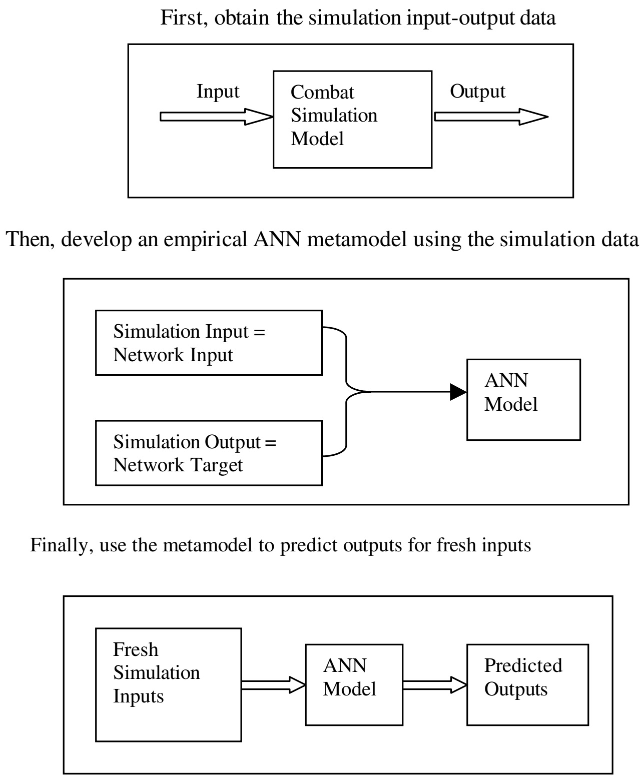

First, obtain the simulation input-output data
Input
Combat Simulation Model
Output
Then, develop an empirical ANN metamodel using the simulation data
Simulation Input = Network Input
Simulation Output = Network Target
ANN Model
Finally, use the metamodel to predict outputs for fresh inputs
Fresh Simulation Inputs
ANN Model
Predicted Outputs

Fig. 1. Process of developing ANN metamodels from a combat simulation model.

applied a neural network in conjunction with simulation modelling to eliminate the trial–error process involved in test operations of a major semiconductor manufacturing plant. For a given set of desired performance measures, i.e. cycle time, work-in-progress, and utilisation of three different testers, the neural network suggests a suitable design of scheduling rules, and the number of each type of tester needed to achieve management's goal. In an early study, Fishwick [15] compared neural networks with traditional approaches, i.e. linear regression and response surface methods, and concluded that the former is inadequate to represent system characteristics. By contrast, later studies focusing on neural networks as a simulation metamodelling approach gave quite encouraging results. Pierreval and Huntsinger [16] indicated the potential use of neural networks as a metamodelling tool for discrete-event and continuous simulation models. From a discrete event job-shop simulation, they describe neural networks as a suitable metamodelling technique if a sufficient number of training cases is available. In his early study on neural networks, Hurrion [17] developed a neural network metamodel from a stochastic simulation of a coal depot. The network is being trained to predict the mean and the 99% confidence interval of the time the depot remains open. He also suggested that a neural network combined with a visual interactive simulation (VIS) can be used as a decision support tool. Badiru and Seiger [18] used a neural network as a simulation metamodel for economic analysis of risky projects. They apply a neural network model to predict the potential returns from investment projects with stochastic parameters, e.g. initial investment, the rate of return, and the investment period. The metamodel developed can analyse the performance of potential future projects without re-running the time-consuming simulation. The authors carried out a similar study in 1998 [19]. Experimenting with a simple investment project, they showed that good predictive capability can be achieved by linking a conventional simulation with a neural network. Kilmer and Smith [20] and Kilmer et al. [21] experimented with an inventory problem, and also suggested neural networks as metamodels for discrete-event simulation analysis. Neural networks perform better than first-order and second-order linear regression models when compared with these two traditional approaches [20]. Anjum et al. [22] mentioned the use of neural network modelling in deterministic simulations rather than a response surface approach. Experimenting with three example problems: an M/M/s queue, a coal depot system and a flow shop system, Hurrion [23] showed that metamodels developed using a neural network produce more accurate responses than regression metamodels. These metamodels can be further employed as a Visual Interactive Meta Simulation technique to analyse the VIS model. Hurrion and Birgil [24] identified several advantages of using a neural network as a simulation metamodel, and have shown that a randomised experimental design can be a valid and practical approach for the development of metamodels.

3. System dynamics combat simulation model

System dynamics (SD) has been suggested as a suitable high-level simulation methodology for representing the dynamics of combat at an aggregate level. This case study reflects that line of approach to the development of a simple, deterministic combat model [2]. The two forces are traditionally referred to as Blue and Red. The scenario is that both sides have frontline forces and others in reserve some distance away. For both sides, if reserves are advanced to the front, there is a delay before their arrival, and due to limitations in transport capacities, there is also a maximum rate of dispatching reserve forces. Each side inflicts combat losses on the other dependent on the rate at which each combatant fires, the number of combatants in action and the effectiveness of fire.

For this case study, a simple SD combat model modified from Coyle [2] is developed within the MATLAB environment, which enables us to generate a large amount of data in a very short time frame. At the centre of the parameter space in which we want to use the metamodel, we have considered an 'even' battle between two forces. Hence, at this point the inputs of the model are set to an approximate parity level, i.e. a combination of values for which both forces gradually tend to zero in a pyrrhic draw. So, from the model one can easily see and analyse how sensitive the simulation output is to a small change in the levels of different factors (inputs) near this central parity point. The approximate central parity combination of levels of four different factors considered in our experiment is as follows:

Factor	Level
Blue force (BLUE)	100
Red force (RED)	300
Blue transport capacity (BTC)	23
Red transport capacity (RTC)	17

Levels of reserve forces are fixed at the following values: Blue reserve = 60, Red reserve = 150 for the approximate parity combination. Individual Blue effectiveness = 1.2/min and individual Red effectiveness = 0.4/min, defined to be the product of individual firing rate and SSKP (single shot kill probability). We have chosen the loss exchange ratio (LER) of the two forces as our simulation output (response variable), defined as

$$\text{LER} = \text{Red Casualties}/\text{Blue Casualties}.$$

4. Experimental design and data set

In order to develop the neural network metamodels, we have considered four input variables (factors) i.e., Blue force (BLUE), Red force (RED), Blue transport capacity (BTC) and Red transport capacity (RTC) from the simulation model. Let these factors be denoted by X_1, X_2, X_3 and X_4, respectively. A data set was generated in conjunction with each experimental design in order to train the networks. Since the main objective of this paper is to investigate the effect of the experimental design approach on the metamodel's predictive accuracy, we have adopted a variety of sampling techniques:

a. three-level full-factorial design (FFD),
b. random sampling design (RSD),
c. random sampling design supplemented with domain knowledge (RSD + DK),
d. central composite design (CCD),
e. a modified Latin Hypercube design (LHD), and
f. a modified Latin Hypercube design supplemented with domain knowledge (LHD + DK)

to generate training data sets for six possible experimental design approaches.

First, we consider a three-level *full-factorial design* (FFD) for four factors that results in $3^4 = 81$ different configurations. Levels (values) for each factor are selected by increasing and decreasing the level of the factor about its central parity value. The following table shows the levels selected for the four factors:

Second, a *random sampling design* (RSD) is considered to generate two different training data sets for both RSD and RSD + DK approaches. The first one that consists of 81 configurations is based on a 'standard' RSD, where each value of each factor is sampled uniformly from within the design envelope (mentioned in Table 1) of the four factors. For the second random sample (from RSD + DK), first 51 configurations are generated using the above-mentioned 'standard' RSD and then the RSD is supplemented with domain knowledge. As mentioned before, for an 'even' battle between two forces considered there exist some approximate parity configurations for which LER values are close to 3. These parity configurations, including the central parity one, are close to the diagonal line in Fig. 2. We shall call this the parity-diagonal line. Fig. 2 is a projection of the design space on to the two force size factors, RED and BLUE, showing an example of a parity-diagonal line joining 7 approximate parity configurations. The further from the parity-diagonal a point is, the higher the advantage to that side. The region close to this parity-diagonal line of the design cube seems to be the most sensitive, and needs experimentation whether the proposed metamodels developed from such a response surface can approximate accurately. This led us to add a further 30 simulation configurations, generated from the region close to the approximate parity-diagonal line, with the previous 51 to make a sample of size 81 for RSD + DK approach.

Table 1
Maximum, central parity and minimum level for the four factors

Factors	Levels		
	−1	0 (central parity)	+1
BLUE	75	100	125
RED	225	300	375
BTC	16	23	30
RTC	12	17	22

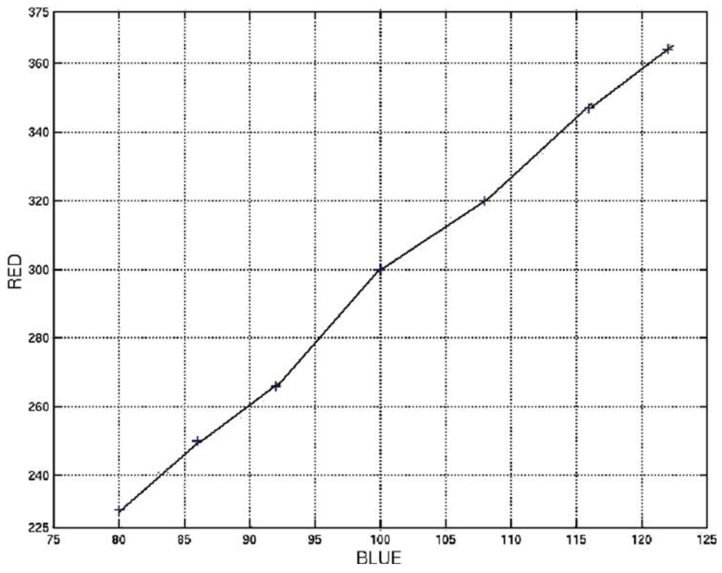


Fig. 2. An approximate parity-diagonal line from a two-dimensional projection of the design cube.

Third, a *central composite design* (CCD) is also employed to draw a sensible comparison among all the possible experimental design approaches. The central composite designs having five different levels (Table 2) of each input factor require far fewer runs than the three-level full-factorial designs. A central composite design has three parts: a cubic portion of two-level factorial design (2^4 configurations for four input factors), a central point of the experiment $(0,0,0,0)$ and an axial design of eight configurations $((\pm a,0,0,0)$, $(0,\pm a,0,0)$, $(0,0,\pm a,0)$, $(0,0,0,\pm a))$ in standardised values. Using this design we have only $2^k + 2k + 1$ or 25 configurations for $k = 4$ input factors. So, a further 56 configurations are being generated using domain knowledge from the response surface to make a sample of size 81. Out of the 56, 30 configurations remain the same as in approach 2 (not to allow bias in comparison) and the remaining 26 are selected randomly from the region close to the approximate parity-diagonal line. Details of the CC design will be found in Box and Draper [5].

Table 2
Five different levels of the factors used to make the CCD

Levels	BLUE	RED	BTC	RTC
$-a$	63	188	12	9
-1	75	225	16	12
0	100	300	23	17
+1	125	375	30	22
$+a$	137	412	34	25

Finally, we consider a modified *Latin Hypercube design* (LHD). Initially, a modified Latin Hypercube sample (LHS) of size 81 is generated using the MAXIMIN-distance criterion (details of the candidate design are given in the following section). A second LHS of size 51 is also generated using the same criterion, which is supplemented by domain knowledge making an equivalent sample of size 81 to the other design approaches.

A random sample of 81 simulation configurations is used as a validation set that has been kept fixed for all four approaches. Data sets are generated in line with the above various approaches so that a direct comparison among them can be made. A further data set (test set) of 10,000 random configurations is sampled uniformly from the design space for independent evaluation of all the neural networks developed from the experiment.

4.1. Developing a modified-latin hypercube design

Random sampling of design points, as in approach 2, is a very common method of producing training data in ANNs, but is problematic when experimenting with small sample sizes. The main drawback of this design is that it can neither ensure a balanced sample within the domain of interest nor can guarantee sampling input levels of each variable proportionately. A conventional factorial design (full or fractional) approach may be a popular choice for someone needs to fill-up this gap. These designs, however, have been frequently used for fitting smooth or linear response surface, and include either a two/three level full or fractional factorial design. In a situation, where the system involves many factors, and each factor has multiple levels (more than 3) i.e. if the response surface is non-linear, these designs can be very expensive and inefficient. So, to take advantage of the non-linearity of ANN, a highly efficient sampling technique called the Latin Hypercube design (LHD) is employed.

An $n \times m$ discrete Latin Hypercube is a design for m factors in n runs, defined as an $n \times m$ design matrix, each column of which is a random permutation of $\{1, 2, \ldots, n\}$. Hence each factor appears once at each of n equally spaced levels. Since their inception by McKay et al. [30] in 1979, Latin Hypercube designs have been applied in many computer experiments. Generally, an LHD permutation is selected randomly. Therefore, how well a particular LHD covers the design space is variable. As a result, poor prediction of the responses is possible. To counteract this possibility, there is a need to use systematic LHDs based on various optimality criteria, such as the entropy-based design criterion [27], integrated mean squared error criterion under a spatial model (IMSE) [28] and a maximin distance criterion [29]. Our approach adopts the maximum minimum distance (MAXIMIN distance) criterion i.e., maximising the minimum distance between points among the configurations. Consider a Latin Hypercube sample (LHS) of size N. To make a proportional sample, the range of each input variable x_i $(i = 1, 2, \ldots, m)$ is divided into N strata, x_{ji} $(j = 1, 2, \ldots, N)$ of equal marginal probability $1/N$, so that representation of all segments of the range is ensured for each input variable. Algorithms described by Lunani et al. [31] are used to generate an LHS using the Maximin distance criterion:

1. Specify the number of input factors, m, the size of the LHS, N, and the number of candidate LHSs to be considered, T.
2. Create N strata for each of the input variables.
3. Generate LHS.
   3.1. For each input variable sample once from each stratum, x_{ji}, where $j = 1,2,\ldots,N$ and $i = 1,2,\ldots,m$ by either random sample within strata or fixed sample within strata.
   3.2. Randomly permute the order of the elements of each column.
4. Calculate $d_{\min}$ the smallest distance between any pair of design points.
   4.1. Calculate inter-point distances as follows:

   $$d(k) = \sum_{i=1}^{m} |x_{ji} - x_{li}|$$

   for $j,l = 1,2,\ldots,N$; $j \neq l$; where $k = 1,2,\ldots,N(N-1)/2$.
   4.2. Derive the minimum distance: $d_{\min} = \min_k(d(k))$.
   4.3. Calculate the number of $d(k)$ equal to $d_{\min}$:

   $$\#d_{\min} = \sum_k \delta_k \quad \text{where } \delta_k = 1 \text{ if } d(k) = d_{\min}, \qquad \delta_k = 0 \text{ if } d(k) > d_{\min}.$$

5. Repeat steps 3 and 4 for T times and select the LHS that maximises $d_{\min}$. In the event of a tie, chose the one which minimises $\#d_{\min}$.

The above algorithms are straightforward for an experiment where each input variable has the same number of levels (or strata). Since our experiment involves discrete input variables with an unequal number of possible values and therefore levels, e.g. X_1 has 51, X_3 has 15, X_4 has 11 possible values, we had to modify the above algorithms to generate the required sample. Suppose, for X_3, we considered a random permutation of 15 so that a single level can occur at least three times while generating an LHS of size 51. As mentioned before, a modified-LHS of size 81 (Fig. 3) is generated by employing the above-mentioned criterion. Fig. 4 shows the second modified-LHS that is supplemented by domain knowledge i.e. the same 30 configurations as used in the RSD + DK design are added to the original LHS to make a sample of size 81 for this design approach.

The approaches employed here are summarised as follows:

1. The training set is made up of the $3^4 = 81$ simulation configurations specified by the combination of factor levels in Table 1.
2. a. The training set is made up of 81 simulation configurations, based on a uniform random sample generated from within the design envelope specified by Table 1.

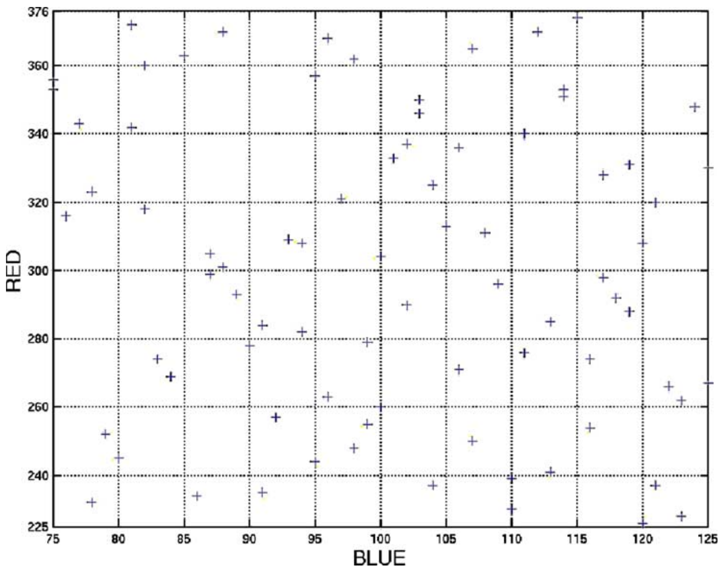


Fig. 3. Training data from a modified-LHD (81 configurations).

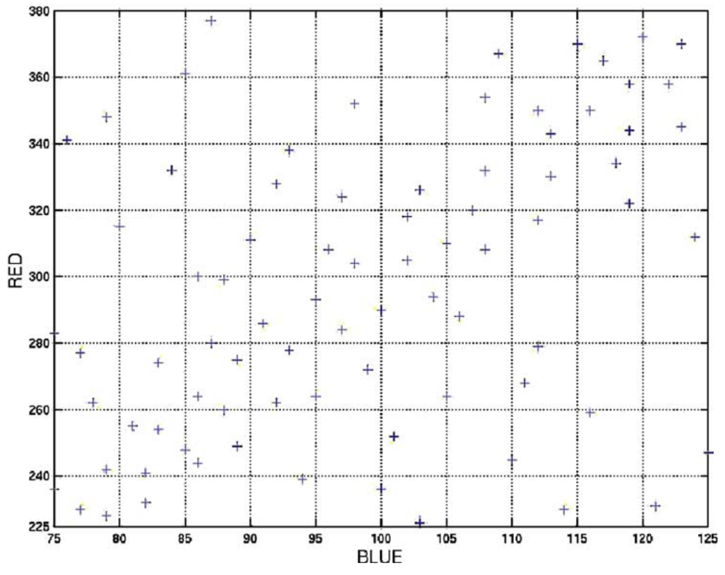


Fig. 4. Training data from a modified-LHD supplemented by domain knowledge.

b. The training set is made up of 81 simulation configurations, based on a uniform random sample of size 51 generated from within the design envelope specified by Table 1 and then supplemented by 30 configurations generated from domain knowledge.

3. The training set is made up of 81 simulation configurations, based on a central composite design supplying 25 configurations specified by Table 2 and further supplemented by 56 configurations based on domain knowledge.
4. a. The training set is made up of 81 simulation configurations, based on a modified-LHD developed using the MAXIMIN distance criterion.
   b. The training set is made up of 81 simulation configurations, based on a modified-LHD of size 51 that is further supplemented by 30 configurations generated from domain knowledge.

Each approach uses the same validation set, consisting of 81 uniformly randomly sampled simulation configurations, and the same test set, consisting of 10,000 uniformly randomly sampled simulation configurations.

5. Development of the neural network metamodels

A three-layer, feed-forward architecture has been employed in our study. Our aim is to train the network to recognise a pattern from the system so that the network is able to predict output values given fresh inputs. For each approach, the data set is split as above into training, validation and test sets. Prior to training the neural network, we pre-process all records by a linear scaling to a range of -1 to $+1$, which is standard practice to make the network's learning task easier. Four input nodes are required to represent the input factors and one output node to represent the single response considered, loss exchange ratio. Each approach has been tested with a different number of nodes, e.g. 5, 7, 10 and 12 at the hidden layer in order to get a valid and suitable neural network. Selection of the number of hidden nodes is based on trial-and-error and some simple rules of thumb. In our experiment, the best performing network observed across all four approaches had five nodes at the hidden layer. However, the number of training epochs, that is the number of processing runs through the complete training data, is limited to 200, giving the maximum training time of the network. Several trials were conducted for each network to find the architecture that minimised the MSE between the target LER values and the LER values predicted by the network. The validation set is used to prevent over-training of the network. Once the MSE of the validation set begins to increase substantially then training can be stopped. Swingler [25] emphasises three criteria, and suggests stopping training if one or more of these have been observed in the network: (1) the average training error has reached a predetermined target value; (2) the average training error no longer falls, or falls by an insignificant amount; (3) the average independent test error starts to increase, indicating the onset of over-fitting. It may be mentioned here that the MSE for the test set should be the same order of magnitude as for the validation set.

Performances of the networks from each of the approaches are evaluated with 10,000 test cases that have not previously been seen by the network. The output generated from the network for these test data is compared with the corresponding target values. Since approach 2 uses a random sampling design, it is necessary that the network developed from this data set should be representative. In practice, an experimenter would usually only have one random sample but the performance of the resulting network depends on how ‘good’ that random sample is. So, in order to obtain representative networks for approach 2, we generate a set of 25 random samples for each of the two separate cases. Hence, for the first case, each of the 25 random samples consists of 81 simulation configurations as described above, as a training set. Similarly, for the second case, each of the 25 samples consists of 51 random configurations, further supplemented by 30 configurations utilising domain knowledge (the same additional 30 configurations have been used for all 25 random samples) to make a design of size 81. Then we develop a neural network for each of the 25 random training sets, and consider the network with median performance measures as the representative network for each of the two separate cases from this approach.

Many authors (e.g., Kleijnen [26]) use the relative prediction error (RPE) as a performance measure for metamodels of deterministic simulations, which is defined as

$$\mathrm{RPE} = \frac{\widehat{Y}_r}{Y_r},$$

where Y_r is the known target value (simulation response) from the independent test data set, and $\widehat{Y}_r$ is the corresponding network output or prediction. Generally, of course, we reject metamodels for which the RPEs are too far from one, but this is a subjective decision and context dependent. Relative prediction errors (RPE) for each metamodel were derived in conjunction with the test data set. In Table 3, percentage distribution of relative prediction error (PDRPE) shows the percentage of test data points for which the RPE falls between 0.8 and 1.2 (20%) and between 0.9 and 1.1 (10%). Obviously, a metamodel is considered to have good predictive ability if the RPEs are close to one. Another measure of performance is the mean squared error of prediction (MSEP), defined as

$$\mathrm{MSEP} = \frac{1}{N}\sum (Y_r - \widehat{Y}_r)^2 \quad (\text{where } N = 10,000)$$

Table 3
Evaluation of NN metamodels developed from various experimental design approaches

Neural network metamodels	MSEP	PDRPE (within 20% error)	PDRPE (within 10% error)	MAPD
Approach 1 (FFD)	0.1457	94	74	7.29
Approach 2 (RSD)				
Complete RSD	0.0812	96	76	7.17
RSD suppl. by DK	0.1152	97	74	7.25
Approach 3 (CCD)				
Suppl. by DK	0.2947	86	57	10.63
Approach 4 (LHD)				
Modified-LHD	0.0451	99	86	5.61
Modified-LHD suppl. by DK	0.0401	99.9	95	4.00

evaluated with the independent test cases after post-processing. The prime advantage of the MSEP lies in its ability to incorporate a measure of both the variance and square of the bias of the prediction errors. The mean absolute percentage deviation (MAPD), which is defined as

$$\text{MAPD} = \frac{1}{N}\sum |[\widehat{Y}_r - Y_r]/Y_r|,$$

is also considered to compare the relative performance of the networks developed from the various approaches. Depending on the degree of precision desired for a particular problem, a reasonable value for each of these measures may vary.

6. Results and discussion

Table 3 presents the performance of the best networks developed from the six different experimental designs. To evaluate the overall performance of the networks, the neural network metamodels have been compared in terms of three predictive indicators: MSEP, PDRPE (for both within 20% error and 10% error) and MAPD. It is repeated here that for both cases of the RSD approach, results of all three performance measures are the median values from the 25 networks developed from 25 different random samples. In order to compare approaches 1, 3 and 4 fairly with a network developed from a 'typical' random sample, the figures quoted for the RSD approach are the median performance measures from the set of 25. In the case of the LHD approach, we considered several modified-LHSs generated using the above mentioned MAXIMIN criterion, and developed the best network for each of the modified-LHSs. We observed much more consistent performances for all the networks developed using the modified-LHD approach, and so it was not con-

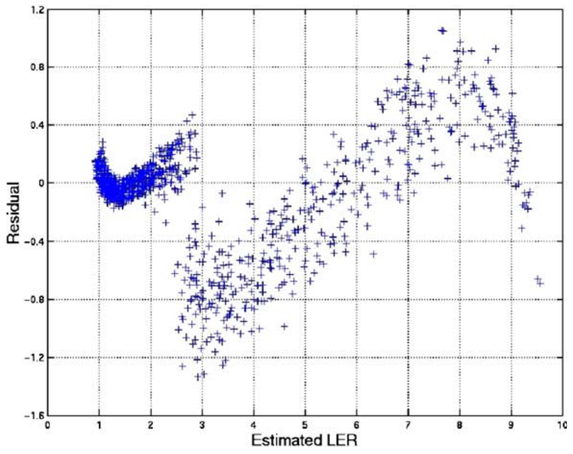
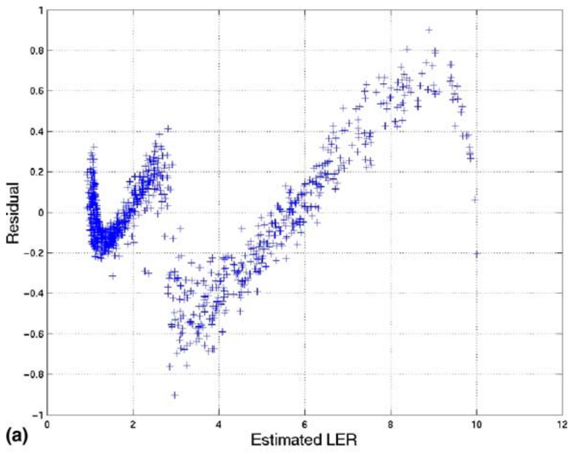


Fig. 5. Sample residuals obtained from network using a FFD.

sidered necessary to generate multiple samples in the way that we did for the RSD approach.

The table shows quite good results for five of the six neural network metamodels with the only exception being approach 3. It is observed that all of the performance measures are best for the final approach (modified-LHD, supplemented with domain knowledge). The percentage of test data points for which the RPE falls between 0.9 and 1.1 (i.e. the relative error is within 10%) ranges from 57% for the CCD approach to 95% for the modified-LHD + DK approach. Similarly, the MAPD ranges from 10.63% to 4.00%, respectively, for these same two approaches. It is clear from the table that a random sampling design (approach 2) can produce better performance than both full factorial and central composite designs for metamodel development, whether the design (RSD) is supplemented by the domain knowledge from the



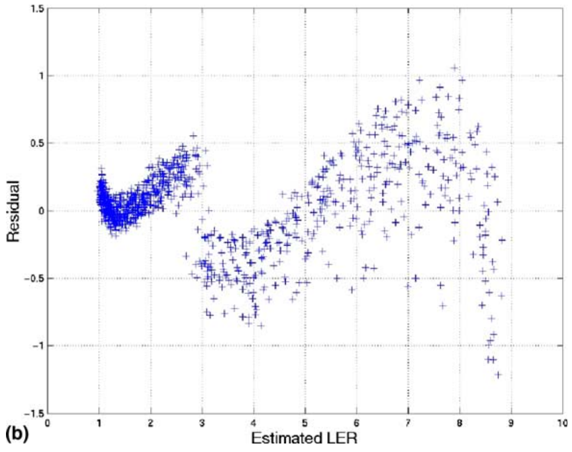


Fig. 6. (a) Sample residuals obtained from using a RSD (without domain knowledge). (b) Sample residuals obtained from network using a RSD + DK.

response surface or not. Studies by Pierreval and Huntsinger [16], Hurrion [23] and Hurrion and Birgil [24] also support this, where they compared only factorial design and randomised design approaches to metamodel development. However, the best predictive metamodels are observed for networks developed from the modified Latin Hypercube design approaches. Furthermore, even an LHS without using any domain knowledge can produce better predictive accuracy than the other approaches (Table 3). Similar results have also been observed from Lunani et al. [31]. They showed that the generalisation capability, measured in terms of 'mean squared prediction errors (MSPE)', was better for the network developed using LHS than those developed using both factorial design and random sampling.

The NN metamodels developed here using training data from both central composite and full-factorial designs perform poorly in terms of approximating the true response function. Due to the limited number of input levels involved in the FFD approach, the configurations are not capable of fully capturing the non-linear behaviour of the response surface. One of the possibilities why CCD performs badly here is because the approach did not cover much of the design space while generating the training data set for the approach. As mentioned in Section 4, most of the training data (56 out of 81) for the CCD approach are generated from a particular region of the design cube using domain knowledge. This does not seem to be a good balance. On the other hand, a random sampling design can be highly variable in how it covers the whole design space, causing the neural networks to perform badly in the region where training data are sparse. The results obtained from these experiments suggest that neural network metamodel, developed from a modified Latin Hypercube design approach produces a more accurate and efficient metamodel than those developed from similar sized full-factorial, central composite or random sampling designs.

A graphical visualisation showing the residuals obtained from the six approaches can be viewed in Figs. 5–8. Figs. 5–8 display the residual-plots against predicted LER for six random samples of size 1000 (out of 10,000 test cases) for the six respec-

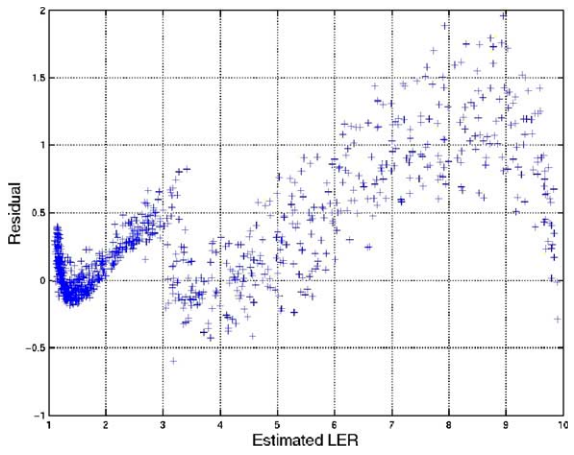
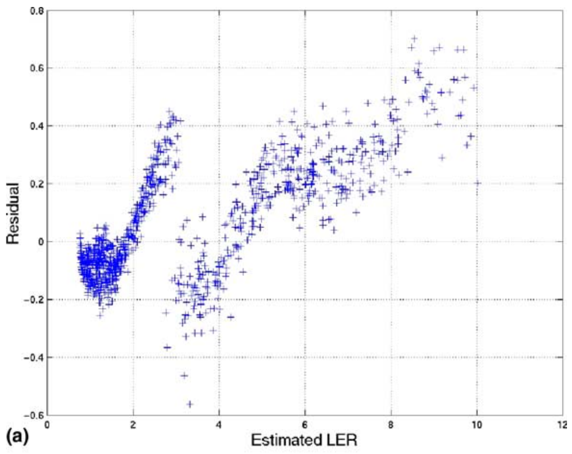


Fig. 7. Sample residuals obtained from using a CCD + DK.



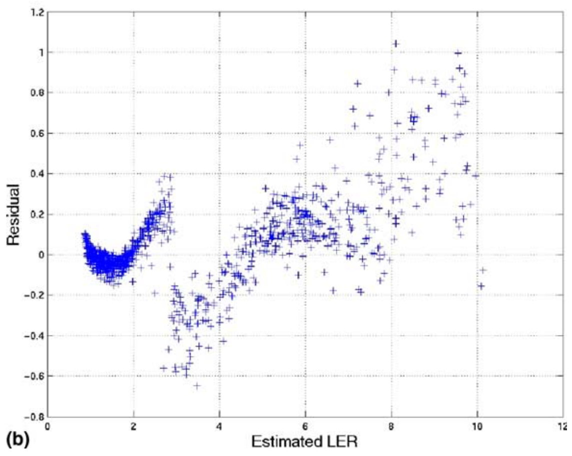


Fig. 8. (a) Sample residuals obtained from network using a modified-LHD (without domain knowledge). (b) Sample residuals obtained from network using a modified-LHD + DK.

tive approaches. For both random sampling cases, RSD and (RSD + DK), the networks having median performances are shown.

It is observed that the overall predictive accuracy measured with respect to the test cases is reasonable for most of the approaches. However, as was already clear from the table, the best predictions are obtained from the network developed using the modified-LHD supplemented with domain knowledge. For most of the sample test configurations, residuals ($\widehat{Y}_r - Y_r$) are close to zero (Fig. 8(b)). Of course there are no distributional results for the residuals here, but clearly $\widehat{Y}_r - Y_r$ will have a larger variance for larger $\widehat{Y}_r$. Hence although we cannot produce standardised residuals as in regression, we can say that the larger spread of residuals for larger $\widehat{Y}_r$ is as expected. All of the metamodels are most sensitive for test cases selected from the region close to the approximate parity diagonal line (Figs. 5–8). That is, a relatively large deviation of the residuals has been observed from all six networks when predicting test points taking LER values approximately between 2.5 and 4.0. LER values calculated for approximate parity points are around 3.0. All of the residual plots exhibit a clear pattern, indicating a systematic bias which the neural networks are not currently taking advantage of. Further work is needed to investigate this.

7. Conclusions

The primary purpose of this study was to investigate the effects of experimental design on the predictive accuracy of artificial neural network metamodels. We suggest that the generalisation capability of an ANN is always sensitive to the training samples, generally, derived from different experimental designs. This paper shows that a modified-Latin Hypercube design, supplemented by domain knowledge, could be an effective and robust method for the development of neural network simulation metamodels. This approach suggests better predictive ability than both conventional (e.g., full factorial and central composite designs) and random sampling design approaches. The LHD is able to generate highly efficient training samples because of its unique behaviour; capturing most non-linearity from the design space while using a relatively small number of configurations. The main drawback with the random sampling design lies in the fact that the experimenter does not have any control over the coverage of the design space and the resulting networks can vary widely in their performance. But the need for a carefully planned experiment has always been recommended during the early stages of a simulation project. The traditional advice to simulation practitioners to employ standard experimental designs may be appropriate when the response surface is relatively smooth and so polynomial regression models, or similar, are appropriate. Such advice is likely to be inappropriate, however, when the response surface is not so well behaved and a more complicated non-linear metamodel is required. The criteria required for factorial designs to perform well are not satisfied for the underlying combat simulation model, with the response surface having a ridge-like feature in the parity-diagonal line of the design cube.

References

[1] R.A. Kilmer, A.E. Smith, L.J. Shuman, An emergency department simulation and a neural network metamodel, Journal of the Society for Health Systems 5 (1997) 63–79.

[2] R.G. Coyle, System Dynamics Modelling: A Practical Approach, Chapman and Hall, London, 1996.

[3] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representation by error propagationParallel Distributed Processing: Exploration in the Microstructure of Cognition, vol. I, MIT Press, Cambridge, Massachusetts, 1986, pp. 318–362.

[4] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan Publishing Company, New York, 1994.

[5] G.E.P. Box, N.R. Draper, Empirical Model Building and Response Surfaces, John Wiley Sons, New York, 1987.

[6] R. Myers, A. Khuri, W. Carter, Response surface methodology: 1966–1988, Technometrics 31 (1989) 137–157.
[7] M. Padgett, T.A. Roppel, Neural networks and simulation: modeling for applications, Simulation 58 (1992) 295–305.
[8] R.A. Kilmer, Applications of artificial neural networks to combat simulations, Mathematical and Computer Modelling 23 (1996) 91–99.
[9] W.O. Hedgepeth, D.A. Jacobs, The application of backpropagation neural networks to predict the results of combat, in: Proceedings of the IEEE International Conference on Neural Networks, Dayton OH, October 17–19, IEEE, Piscataway, NJ, 1994, pp. 199–203.
[10] J.D. Morrison, A neural network model that supports real-time learning of temporal relationships in complex engineering domains, Simulation 59 (1992) 152–163.
[11] B. Widrow, D.E. Rumelhart, M.A. Lehr, Neural networks: applications in industry, business and science, Communication of the ACM 37 (1994) 93–105.
[12] G. Chryssolouris, M. Lee, J. Pierce, M. Domroese, Use of neural networks for the design of manufacturing systems, Manufacturing Review 3 (1990) 187–194.
[13] G. Chryssolouris, M. Lee, M. Domroese, Use of neural networks in determining operational policies for manufacturing systems, Journal of Manufacturing Systems 10 (1990) 166–175.
[14] M. Mollaghasemi, K. LeCroy, M. Georgiopoulos, Application of neural networks and simulation modeling in manufacturing system design, Interfaces 28 (1998) 100–114.
[15] P.A. Fishwick, Neural network models in simulation: a comparison with traditional modeling approaches, in: E.A. MacNair, K.J. Musselman, P. Heidelberger (Eds.), Proceedings of the 1989 Winter Simulation Conference, 1989, pp. 702–710.
[16] H. Pierreval, R.C. Huntsinger, An investigation on neural network capabilities as simulation metamodels, in: Proceedings of the 1992 Summer Computer Simulation Conference, Society for Computer Simulation, San Diego, CA, 1992, pp. 413–417.
[17] R.D. Hurrion, Using a neural network to enhance the decision making quality of a visual interactive simulation model, Journal of the Operational Research Society 43 (1992) 333–341.
[18] A.B. Badiru, D.B. Seiger, Neural network as a simulation metamodel in economic analysis of risky projects, Technical Report (Department of Industrial Engineering, University of Oklahoma), 1993.
[19] A.B. Badiru, D.B. Seiger, Neural network as a simulation metamodel in economic analysis of risky projects, European Journal of Operational Research 105 (1998) 130–142.
[20] R.A. Kilmer, A.E. Smith, Using artificial neural networks to approximate a discrete event stochastic simulation model, in: C.H. Dagli, L.I. Burke, B.R. Fernandez, J. Ghosh (Eds.), Intelligent Engineering Systems Through Artificial Neural Networks, vol. 3, ASME Press, New York, 1993, pp. 631–636.
[21] R.A. Kilmer, A.E. Smith, L.J. Shuman, Neural networks as a metamodelling technique for discrete event stochastic simulation, in: C.H. Dagli, B.R. Fernandez, J. Gosh, R.T. Kumara (Eds.), Intelligent Engineering Systems Through Artificial Neural Networks, vol. 4, ASME Press, New York, 1994, pp. 1141–1146.
[22] M.F. Anjum, I. Tasadduq, K. Al-Sultan, Response surface methodology: a neural network approach, European Journal of Operational Research 101 (1997) 65–73.
[23] R.D. Hurrion, Visual interactive meta-simulation using neural networks, International Transactions in Operational Research 5 (1998) 261–271.
[24] R.D. Hurrion, S. Birgil, A comparison of factorial and random experimental design methods for the development of regression and neural network simulation metamodels, Journal of the Operational Research Society 50 (1999) 1018–1033.
[25] K. Swingler, Applying Neural Networks: A Practical Guide, Academic Press, London, 1996.
[26] J.P.C. Kleijnen, Statistical Tools for Simulation Practitioners, M Dekker, New York, 1987.
[27] C. Currin, T.J. Mitchell, M.D. Morris, D. Ylvisaker, Bayesian prediction of deterministic functions with application to the design and analysis of computer experiments, Journal of the American Statistical Association 86 (1991) 953–963.
[28] J. Sacks, S.B. Schiller, W.J. Welch, Design for computer experiments, Technometrics 31 (1989) 41–47.

[29] M.E. Johnson, L.M. Moore, D. Ylvisaker, Minimax and maximin distance designs, Journal of Statistical Planning and Inference 26 (1990) 131–148.

[30] M.D. McKay, W.J. Conover, R.J. Beckman, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (1979) 239–245.

[31] M. Lunani, A. Sudjianto, P.L. Johnston, Generating efficient training samples for neural networks using Latin Hypercube samplingIntelligent Engineering Systems Through Artificial Neural Networks, vol. 5, ASME Press, 1995, pp. 209–214.