JKAU: Eng. Sci., **vol. 15 no. 1**, pp. 125-138 (1425 A.H./ 2004 A.D.)

Approximation of a Discrete Event Stochastic Simulation Using an Evolutionary Artificial Neural Network

HINDI A. AL-HINDI

Associate Professor of IS, College of Business and Economics, King Saud University, Al-Qasseem Branch, Saudi Arabia
E-mail: hahindi@ksu.edu.sa

ABSTRACT. A computer simulation model may be regarded as a mapping function bietween a set of input and output variables. Although simulation models are very popular experimentation tools, in many cases they are computationally expensive. Hence, it would be essential to have fast, accurate approximation of computer simulation. This paper examines the use of an evolutionary artificial neural network for approximating a lot size – reorder point inventory system simulation. The proposed approach was compared with a Backpropagation trained neural network and multiple linear regression models.

KEYWORDS: Simulation metamodels, Evolutionary ANN

1. Introduction

Simulation based analysis tools are finding increased use to explore different design alternatives. Inputs to the simulation model are the decision variables of the system, while the outputs are the outcomes of the system. One of the shortcomings of discrete event simulation is the amount of computational resources required to fully explore system responses [1, 2]. This is true especially in situations where the decision making is fast and there is no time to perform multiple replications for the selected values of the decision variables. To overcome limitations of discrete event simulation models, researchers and practitioners have developed metamodels, which are approximations to the simulation model and more computationally efficient [3, 4]. The objective of the metamodel is to accurately reproduce the simulation over wide ranges of decision variables and reduce computation time. The metamodel can be used as

a real time decision support to determine the best alternative to solve the problem.

This paper discusses the use of an evolutionary neural network as a metamodeling technique for discrete event, stochastic simulation. An (s, S) inventory system from the literature is represented by a metamodel using an evolutionary neural network to estimate the expected total cost. The performance of the proposed metamodeling approach was compared to that of existing approaches that included a neural network trained with Backpropagation algorithm and regression models. It is shown that the evolutionary neural network metamodel is quite competitive in accuracy when compared to the simulation itself and outperforms other methods, but it requires more training time.

2. Simulation Metamodels

A simulation metamodel is an approximation model, which provides a model of the simulation model [5]. To define a stochastic simulation metamodel [6, 7], let X_j, j=1,2, . . ., n denote the variables influencing the response, Y, of the physical system. The unknown relationship between Y and X_j may be written as:

$$Y = f(X_1, . \ . \ ., \ X_n) + \gamma \tag{1}$$

where γ is a random noise, with zero mean and is independent from the inputs X_j. The predictor of Y is given by:

$$\hat{Y} = E(Y \mid X_1, ..., X_n) = f(X_1, . \ . \ ., \ X_n) \tag{2}$$

A simulation model usually includes a set of the input variables as follows:

$$Y' = g(X_1, . \ . \ ., \ X_p) + \delta \tag{3}$$

where $p \le n$ and δ is random noise with zero mean and is independent of the inputs X_j, and represents the random fluctuation of the simulation model. The predictor for the simulation model is given by:

$$\hat{Y}' = g(X_1, . \ . \ ., \ X_p) \tag{4}$$

The amount $\hat{Y}'-\hat{Y}=\varepsilon_s$ denotes the error term accounting for the simulation error, resulting from the excluded variables and model specification.

A metamodel, *h*, is a further simplification of the simulation and it can be written as:

$$Y''=h(X_1,\ldots,X_m)+\tau \tag{5}$$

where $m \le p \le n$ and τ is a random error. Since the metamodel is adjusted from the simulation model, $\hat{Y}'-Y''=\varepsilon_m$ is an additional error accounting for the metamodeling error resulting from the further excluded variables and the metamodeling error of fitting the metamodel to the simulation model.

The specification of the metamodel is the identification of the parameters of $h(\cdot)$. The metamodel can be viewed as a simplified model of the simulation model acting as a surrogate for the study of the physical system. The metamodeling error is therefore the sum of the three error terms, $Y-Y''=\varepsilon_m+\varepsilon_s+\gamma$.

There are several techniques that have been proposed as metamodeling tools. Parametric regression models are widely used as surrogate models in simulation [8, 9]. While easy to implement, the performance of a regression metamodel depends on the applicability of the regression functional form to the simulation response. In addition, regression models assume that the errors are identical and normally distributed and violation of these assumptions impacts the reliability of the model.

Artificial neural networks (ANNs) have received an increasing attention as metamodeling tools. They offer universal function approximation capability that can mimic any functional relationship with a high degree of precision [10]. For development of a metamodel, the ability to accurately model any functional relationship is critical because it removes the possibility of specifying an incorrect functional form. This means the component ε_m which comes from the metamodeling error would be close to zero that is no error would be incurred in fitting the metamodel since a neural network can exactly mimic any simulation relationship [7].

3. Models Description

The inventory examined in this paper has been described in Law and Kelton [11]. This system is an (*s, S*) inventory system where *s* = reorder point

and S = order quantity and d=S-s is the reorder quantity. A stationary (s, S) policy is used to decide how much to order at the beginning of each time period. Given that I is the inventory at the beginning of each period, then the amount to order for that time period Z is determined by:

$$Z = \begin{cases} S - I & if \quad I < s \\ 0 & if \quad I \geq s \end{cases} \tag{6}$$

Law and Kelton [11] held all input parameters fixed except for s and d and used the annual cost, *tc*, to build a response surface for the inventory system. The total cost includes holding cost, order cost and shortage cost. This paper used the same inputs and output to develop the proposed metamodel. The following are the parameters of the system as described in Law and Kelton [11]:

$$D \text{ (Demand)} = \begin{cases} 1 \text{ with probability } 1/6 \\ 2 \text{ with probability } 2/6 \\ 3 \text{ with probability } 2/6 \\ 4 \text{ with probability } 1/6 \end{cases}$$

t_d (time between demands) exp(1/λ=0.1 month)
I = inventory level (I_0 =60)
s = reorder point

d = reorder quantity
t_0 = time between order decisions = 1 month
t_L = time lag for delivery (U[0.5 to 1] month)
k = setup cost
i = incremental cost per item ordered = 3.0
h = holding cost per item= 1.0 / month
u = underage cost = 5.0 / month
n = time horizon = 120 month

This system has been previously investigated. Klimer and Smith [12] and Kilmer et al. [13] used artificial neural networks trained with Backpropagation algorithm to construct empirical metamodels of computer simulations. In these two studies, basic simulation tasks, such as prediction and comparison of alternatives, were performed with neural approximations of a computer simulation.

In this study, data was obtained from computer simulation in order to build a fast empirical model of the system. The inventory system simulation was written in the C language and verified against the Kilmer and Smith [12] simulation, which held all input parameters fixed except for reorder point s and reorder quantity d. The data used to develop the approximation models consist of 420 data points constructed with all possible combinations of 21 equally spaced values of s between 0 and 100 (i.e., 0, 5, 10, . . . , 100) and 20 equally spaced values of d between 5 and 100 (i.e., 5, 10, 15, . . ., 100). Simulation was run for each combination of s and d to obtain the corresponding total cost, *tc*, in a similar fashion as was done by Kilmer and Smith [12]. Each simulation run resulted in one value of the response variable, total cost, and this response depends on the values of the input variables, reorder point, *s*, and reorder quantity, *d*.

4. Evolutionary Neural Networks

Evolutionary artificial neural networks [14, 15], include genetic algorithm (GA) or other evolutionary algorithms in their operations, which are used to perform various tasks such as weight training, architecture design, input selection and initial weights selection. The combination of neural networks and evolutionary computation has been intensively investigated in the past years. The main goal of these studies is to determine the topology and training of a neural network by an evolutionary algorithm that represents a neural network structure and weights as individual genes. To name a few of these works: [16], [17], [18] and [19].

Weight training in ANNs is usually formulated as minimization of an error function between target outputs and actual outputs and iteratively adjusting the weights. Deterministic training algorithms like Backpropagation (BP) [20] and Conjugate Gradient Backpropagation [21] are widely used in ANN training. These algorithms are based on gradient descent search. They often get trapped in a local minima of the error function and they are inefficient in finding a global minima.

The use of stochastic algorithms is another alternative for ANN training [22], and they are less sensitive to local minima. Nevertheless, they are generally slow compared to the fastest versions of gradient based algorithms, since they perform global searches and implement probabilistic search operators [23]. These algorithms include the use of Genetic Algorithm (GA) [24], and evolutionary algorithms in ANN training. The points in the search space to be examined are selected on the basis of probabilistic criteria. Therefore, stochastic algorithms are less vulnerable to the local minima problem.

It is more advantageous to apply a hybrid training algorithm, which combines the advantages of both deterministic and stochastic approaches. The stochastic algorithm is used to define the initial weights of the neural network while the deterministic algorithm is used to perform local search. Examples of these hybrid training algorithms include the combination of GA and BP algorithms [25, 26] and the combination of simulated annealing and Conjugate Gradient [27].

The proposed evolutionary neural network was trained using differential evolution algorithm [28, 29]. The evolutionary algorithm is used to derive the connection weights and biases of the neural network, which are represented as individuals in the population of the algorithm. Each differential evolution chromosome stores a weight configuration that is improved from one generation

to another. One important feature of evolutionary algorithm is its population based search strategy, where individuals in a population compete and exchange information with each other to improve the solution of the problem.

5. Differential Evolution Algorithm

Differential Evolution (DE) [28] is an improved version of GA for faster optimization. Unlike simple GA that uses binary coding for representing problem parameters, DE uses real coding representation. DE has several advantages that include its simple structure and speed. Price and Storn [28] gave the main principles of DE with single strategy, then, they suggested ten different strategies of DE [29]. The selection of the strategy and key parameter values are critical to the performance of DE. These control parameters are: population size (*NP*), crossover constant (CR) and the scaling factor (F).

There are several variants of DE. The following briefly describes the DE/rand/1/bin scheme. More detail descriptions are given in [28-31]. DE operates on a population, P_G, of candidate solutions, not just a single solution. It maintains a population of constant size that consists of NP real valued vectors $X_{i,G}$, where i indexes the population and G is the generation to which the population belongs.

$$P_G = (X_{1,G}, \ldots, X_{NP,G}) \qquad G = 0, \ldots, G_{\max} \tag{7}$$

Each vector $X_{i,G}$ contains D real parameters (chromosomes):

$$X_{i,G} = (x_{1,i,G}, \ldots, x_{D,i,G}) \qquad i = 1, \ldots, NP; \;\; G = 0, \ldots, G_{\max} \tag{8}$$

The values of the initial population, P_G=0 , are initialized with random values

$$x_{j,i,0} = rand_j[0,1] \qquad i = 1, \ldots, NP; \;\; j = 1, \ldots, D \tag{9}$$

where rand$_j$[0,1] denotes a uniformly distributed random value within the range [0.0,1.0] that is chosen for each j.

Vectors in the current population, P_G, are randomly combined to create candidate vectors for the next generation P_{G+1}. The population of candidates, or trial vectors $U_{i,\,G+1}$ is generated as follows:

$$u_{j,i,G+1} = \begin{cases} x_{j,r3,G} + F.(x_{j,r1,G} - x_{j,r2,G}) & \text{if} \quad \text{rand}_j[0,1] \le CR \vee j = k \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{10}$$

where,

i=1, . . ., NP; j=1, . . ., D

$k \in \{1,\ldots,D\}$; *random parameter index chosen once for each* i

$r_1, r_2, r_3 \in \{1,\ldots,NP\}$; randomly selected with $r_1 \neq r_2 \neq r_3 \neq i$

$CR \in [0,1]$, $F \in (0,1+]$

The randomly chosen indexes, r_1, r_2 and r_3 are different from each other and also different from the running index, i. New random integer values r_1, r_2 and r_3 are chosen for each individual i. The index, k, refers to a randomly selected chromosome which is used to ensure that each individual trial vector, $U_{i,G+1}$, differs from its counterpart in the previous generation, $X_{i,G}$, by at least one parameter. A new random integer value is assigned to k prior to the construction of each trial vector i.

F, NP and *CR* are *DE* control parameters and they remain constant during the search process. F is a real-valued factor in the range [0.0,1+] that scales the differential variations. NP is the size of the population. CR is the crossover factor in the range [0,1] that controls the probability that a trial vector parameter $u_{j,i,G+1}$ will come from a randomly chosen, mutated vector $x_{j,r3,G} + F.(x_{j,r1,G} - x_{j,r2,G})$ instead of from the current vector, $X_{j,i,G}$. Usually suitable values for *NP, F* and *CR* are obtained by trial-and-error using different values of these parameters.

The population for the next generation, P_{G+1}, is selected from the current population and the trial population U_{G+1} according to the following rule:

$$X_{i,G+1} = \begin{cases} U_{i,G+1} & \text{if} \quad f(U_{i,G+1}) \leq f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \tag{11}$$

where $f(\cdot)$ is the fitness (cost) function of the optimization process. Hence, each individual of the current population is compared with its counterpart in the trial population. Assuming that the objective function is to be minimized, the vector with the lowest objective function value wins a place in the next generation population. Therefore, all individuals in the next generation are as good or better than their counterparts in the current population.

6. Results

Two regression models were estimated using the 420 simulation data points. The first is first order regression model and the second is second order regression model, which are typically used in simulation metamodeling.

Three ANN models were constructed using the same data: i) an artificial neural trained with the Conjugate Backpropagation training algorithm, ii) an evolutionary neural network trained with differential evolution, and iii) an evolutionary neural network trained with hybrid training algorithm that combines Conjugate BP and differential evolution algorithm. The DE algorithm for ANN training was implemented using MATLAB scripts [29], whereas the Conjugate BP was implemented using the Neural Networks Toolbox from MATLAB®. All the ANN models had two inputs nodes, one corresponding to s and the other corresponding to *d* and one output node corresponding to *tc*. In addition, each network has one hidden layer with eight nodes, which were selected after several experiments with a different number of nodes.

The differential evolution algorithm requires the specification of several parameters that include the size of the population (NP), crossover factor (CR) and the scaling factor (F). In neural network training, the size of the population is the number of the connection weights and biases. In a neural network with two input nodes, eight hidden nodes and one output node, there are 31 connection weights and biases representing members of the population that will go through the crossover and mutation operations during the training process. The values of CR and F are selected by trial-and-error and after several experiments, it appeared that CR=0.5 and F=0.8 are appropriate values.

Each neural network was trained for a total of 5000 epochs. This number of epochs was found to be the number at which there was no improvement in the performance of any of the three neural networks. Training was evaluated using the regularized performance function [32], which takes the following form:

$$E_{reg} = \frac{e^T e}{n} + \alpha \frac{w^T w}{k} \tag{12}$$

where $\frac{e^T e}{n}$ is the mean square error, $\frac{w^T w}{k}$ is the mean square of the network weights and biases and α is a regularization factor. The size of the training data is n and the number of connection weights and biases is k. Using the regularized performance function can improve the generalization ability of the neural network [32]. Training was performed using a PC Pentium IV 2000 MHz with

256MB RAM. Table (1) shows some of the training statistics for the three training algorithms. Initial E_{reg} is the value of the regularized performance function at the beginning of training (epoch=0) using the initial connection weights and minimum E_{reg} is the value of the regularized performance function at the end of training (epoch=5000), which represents the minimum value of E_{reg} that can be achieved by the training algorithm.

Table 1: Training statistics.

Training algorithm	Initial E_{reg}	Minimum E_{reg}	Training time (minutes)
conjugate Backpropagation	23344.5	146.86	1.05
differential evolution	22603.8	88.95	19.13
hybrid training	88.95	53.23	21.46

The hybrid training algorithm achieved the lowest E_{reg} value followed by the differential evolution algorithm then the conjugate Backpropagation. The conjugate Backpropagation algorithm took only 1.05 minute to run the specified 5000 epochs whereas the evolutionary algorithm took 19.13 minutes and the hybrid algorithm took 21.46 minutes. Training a neural network using the evolutionary algorithm requires more time compared to the conjugate Backpropagation. In the evolutionary algorithm, the population consists of the network connection weights and biases. During the training process, these weights are updated using crossover and mutation to derive a better set of weights, which takes more time compared to the Backpropagation algorithm where weights are updated using a direct gradient descent procedure.

The evolutionary ANN with differential evolution training algorithm was retrained with a training goal set to E_{reg} value equals to 146.86, which is the lowest E_{reg} value achieved by the conjugate Backpropagation in 5000 epochs. The evolutionary ANN converged to the desired error value after 804 epochs. This is a clear indication that although the evolutionary ANN requires more training time, it is less susceptible to fall into a local minima compared with a feed-forward ANN with backpropagation training algorithm.

Table 2: Regression models results.

Model	Regression equation	R^2	MAE
First order	tc= 116.6+0.59s+0.096d	0.55	10.93
Second order	tc=170.5-1.09s-1.01d+0.008sd+0.012s^2+0.007d^2	0.85	6.51

Table 3: ANNs results.

Model	Training algorithm	R^2	MAE
Feed-forward ANN	conjugate Backpropagation	0.75	6.64
Evolutionary ANN	differential evolution	0.88	6.49
Evolutionary ANN	hybrid training	0.94	4.53

Two evaluation criteria were used to evaluate the proposed approaches for simulation metamodeling: the determinant coefficient (R^2) and mean absolute error (MAE) . The values of these criteria for the statistical models are shown in Table 2 whereas Table 3 shows these values for the neural networks. The results of the study show that: i) the evolutionary neural network outperforms the statistical models and the feed-forward neural network trained with Backpropagation algorithm, ii) the evolutionary neural network takes more time in training, however, it requires fewer number of epochs to reach the same performance as the feed-forward neural network, iii) the evolutionary neural network is less susceptible to fall into local minima compared to the feed-forward neural network, and iv) the neural network trained with hybrid training algorithm outperforms all other models. The combination of evolutionary algorithm and Backpropagation is more efficient than using evolutionary training alone. In this case, the weights and biases derived by evolutionary algorithm training are used as initial values for the Backpropagation algorithm.

The response surfaces given in Figure 1 permit a visual comparison of the models considered in the study. Figure 1a plots the response surface of the 420 simulation data points. Figures 1b through 1f plot the response surfaces of first order regression, second order regression, a neural network with BP training, an evolutionary ANN with DE training and an evolutionary ANN with hybrid training respectively that were constructed using the 420 data points. The best one from the ANN or the regression would be a response surface close to 1a. These plots show that the response surface of the evolutionary ANN with hybrid training (1f) is closer than other models to approximating the computer simulation. The second closest is the response surface of the evolutionary ANN trained with DE.

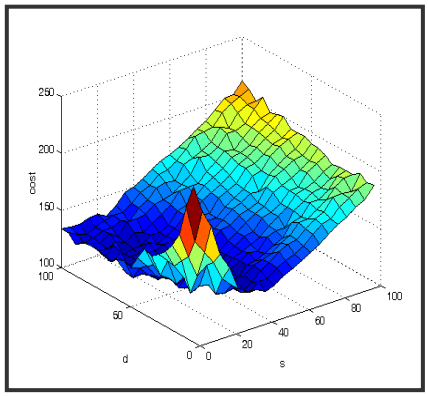

Fig. 1a: Direct simulation 420 points.

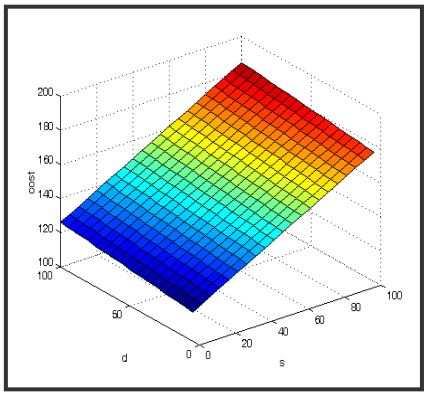

Fig. 1b: First order regression.

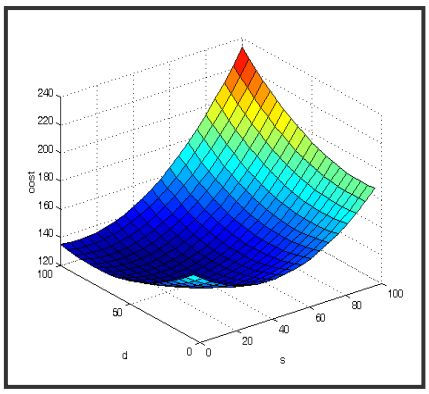

Fig. 1c. Second order regression.

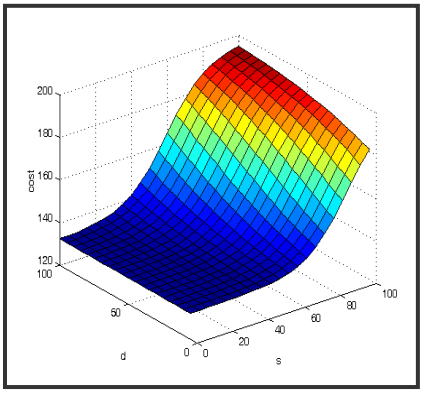

Fig. 1d. ANN with BP training.

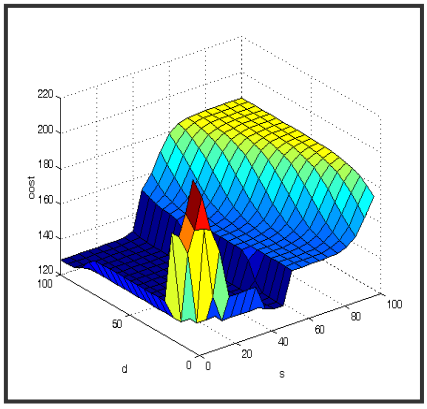

Fig. 1e. ANN with DE training.

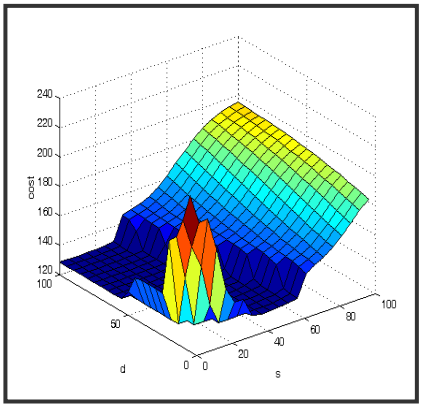

Fig. 1f. ANN with hybrid training.

7. Conclusion

Decision making using simulation models can be time consuming because of the computational efforts of running the simulation many times before a decision can be made. This has motivated the field of simulation metamodeling, intended to build approximations of simulation models. There are fundamental reasons for using a neural network approach to simulation metamodeling, which include universal approximation ability.

This paper proposed a new model for simulation metamodeling and compared its performance to available models. The results of the study demonstrated the capabilities of evolutionary ANNs in building simulation metamodels. The evolutionary algorithm can be used by itself to train the evolutionary neural network or in combination with the Backpropagation algorithm. In either case, it will perform better than statistical models or a neural network trained with Backpropagation.

References

[1] **Barton, R. R.**, Simulation Metamodels, *Proceedings of the 30th Conference on Winter Simulation*, Washington, DC., 167-176 (1998).

[2] **Barton, R. R.**, Metamodels for Simulation Input-Output Relations, *Proceedings of the 1992 Winter Simulation Conference*, **J. J. Swain, D. Goldsman, R. C. Crain** and **J. R. Wilson** (eds), 289-299 (1992).

[3] **Giunta, A. A.** and **Watson, L. T.**, A Comparison of Approximation Modeling Techniques: Polynomial Versus Interpolating Models, *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Design*, St. Louis, MO, 392-404 (1998).

[4] **Jin, R., Chen, W.** and **Simpson, T. W.**, Comparative Studies of Metamodeling Techniques under Multiple Modeling Criteria, *Journal of Structural and Multidisciplinary Optimization*, **23**, 1, 1-13 (2001).

[5] **Kleijnen, J. P. C.**, *Statistical Tools for Simulation Practitioners,* Marcel Dekker, New York (1987).

[6] **Aussem A., Hill D.**, Wedding Connectionist and Algorithmic Modeling: Towards Forecasting Caulerpa Taxifolia Development in North-Western Mediterranean Sea, *Ecological Modeling*, **120**, 225-236 (1999).

[7] **Kilmer, Robert A. Smith, Alice,** and **Shuman, Larry J.**, Computing Confidence Intervals for Stochastic Simulation Using Neural Networks Metamodels, *Computers and Industrial Engineering*, **36**, 391-407 (1999).

[8] **Myers, R. H.** and **Montgomery, D. C.**, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley and Sons, New York (1995).

[9] **Venter, G. Haftka, R. T.** and **Starnes, J. H. Jr.**, Construction of Response Surfaces for Design Optimization Applications, *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, WA, AIAA, **1**, 548-564 (1996).

[10] **Hornik, K, Stinchcombe, M,** and **White, H.**, Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, **2**, 359-366 (1989).

[11] **Law, A.** and **Kelton, D.**, *Simulation Modeling and Analysis*. McGraw Hill, New York (1991).

[12] **Kilmer, Robert A.** and **Smith, Alice**, Using Artificial Neural Networks to Approximate a Discrete Event Stochastic Simulation Model, *Intelligent Engineering Systems Through Artificial Neural Networks*, **3**, 631-636 (1993).
[13] **Kilmer, R. A., Smith, A. E.** and **Shunman, L. J.**, Using Neural Network Metamodels to Develop Prediction Intervals for Discrete Event Simulation, *Intelligent Engineering Systems Through Artificial Neural Networks*, **4**, 1141-1146 (1994).
[14] **Yao, X.**, A Review of Evolutionary Artificial Neural Networks, *International Journal of Intelligent Systems*, **8**, 4, 539-567 (1993).
[15] **Yao, X.**, Evolutionary Artificial Neural Networks, *International Journal of Neural Systems*, **4**, 3, 203-222 (1993).
[16] **Whitley, D. , Starkweather, T.** and **Bogart, C.**, Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity, *Parallel Computing*, **14**, 347-361 (1990).
[17] **Yao, X.** and **Liu, Y.**, Towards Designing Artificial Neural Networks by Evolution, *Applied Mathematics and Computation*, **91**, 1, 83-90 (1998).
[18] **Fogel, D. B., Fogel, L. J.** and **Porto, V. W.**, Evolving Neural Networks, *Biological Cybernetics*, **63**, 487-493 (1990).
[19] **Yao, X** and **Liu., Y.**, Evolutionary Artificial Neural Networks that Learn and Generalise Well, *Proceedings of the 1996 IEEE International Conference on Neural Networks*, Washington DC, 159-164 (1996).
[20] **Rumelhart, D. E., Hinton, G. E.** and **Williams, R. J.**, Learning Internal Representations by Error Propagation, *In Parallel Distributed Processing,* **Rumelhart, D. E.** and **Mcclelland, J. L**. (Eds), **1**, MIT Press, Cambridge, 318-362 (1986).
[21] **Moller, M. F.**, A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, *Neural Networks*, **6**, 525-533 (1993).
[22] **Yao, X.** and **Liu. Y.**, A New Evolutionary System for Evolving Artificial Neural Networks, *IEEE Transactions on Neural Networks*, **8**, **3**, 694-713 (1997).
[23] **Yao, X.**, Evolutionary artificial neural Networks, *Encyclopedia of Computer Science and Technology*, **33**, 137-170 (1995).
[24] **Goldberg, D. E.**, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts (1989).
[25] **Belew, R. K., McInerny, J.** and **Schraudolph, N. N.**, Evolving Networks: Using the Genetic Algorithm With Connectionist Learning, *Proceedings of the Second Conference on Artificial Life,* **Langton, C.G.; Taylor, C. ; Farmer, J. D.** and **Rasmussen, S**. (Eds)*,* 511-548, (1991).
[26] **Kinnebrock, W.**, Accelerating the Standard Backpropagation Method Using a Genetic Approach, *Neurocomputing*, **6**, 583-588 (1994).
[27] **Masters, T.**, *Advanced Algorithms for Neural Networks: a C++ Sourcebook*, John Wiley & Sons, New York (1995).
[28] **Price, K.** and **Storn, R.**, Differential Evolution – A Simple Evolution Strategy for Fast Optimization, *Dr. Dobb's Journal*, **22**, **4**, 18-24, and 78 (1997).
[29] **Price, K.** and **Storn, R.**, *Web Site of DE*, March 2003. URL: http://www.ICSI.Berkeley.edu/~Storn/code.html
[30] **Storn, R.**, System Design by Constraint Adaptation and Differential Evolution, *IEEE Trans. on Evolutionary Computation*, **3**, **1**, 22-34 (1999).
[31] **Storn, R.** and **Price, K.**, Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization*, **11**, 341-359 (1997).
[32] **Chen, D** and **Hagan, M.T.**, Optimal Use of Regularization and Cross-Validation in Neural Network Modeling, Available via URL: http://hagan.ecen.ceat.okstate.edu/chen_hag_nn99.pdf

تقريب محاكاة الأحداث العشوائية الوثابة باستخدام شبكة عصبية اصطناعية تطورية

هندي بن عبدالله الهندي

كلية الاقتصاد والإدارة – جامعة الملك سعود

فرع القصيم – المملكة العربية السعودية

المستخلص: تهدف المحاكاة باستخدام الحاسب الآلي إلى إيجاد العلاقة بين مجموعة من المدخلات والمخرجات. وعلى الرغم من شيوع نماذج المحاكاة كأدوات تجريبية إلا أنها في كثير من الحالات تتطلب وقتًا طويلاً للحصول على النتائج المطلوبة لذلك من الضروري استخدام تقريب لنماذج المحاكاة. ويهدف هذا البحث إلى استخدام شبكة عصبية اصطناعية تطورية لتقريب محاكاة نموذج مخزون عشوائي. الشبكة العصبية الاصطناعية التطورية المقترحة لتقريب المحاكاة تم مقارنتها مع شبكة عصبية اصطناعية تقليدية تم تدريبها باستخدام أسلوب الانتجاه للخلف. كما تم مقارنة الشبكة العصبية المقترحة مع نماذج الانحدار الخطية وذلك للتأكد من كفاءتها.