Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

MODEL ABSTRACTION FOR DISCRETE EVENT SYSTEMS USING NEURAL NETWORKS AND SENSITIVITY INFORMATION

Christos G. Panayiotou
Christos G. Cassandras

Department of Manufacturing Engineering
Boston University
Boston, MA 02215, U.S.A.

Wei-Bo Gong

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003, U.S.A.

ABSTRACT

Simulation is one of the most powerful tools for modeling and evaluating the performance of complex systems, however, it is computationally slow. One approach to overcome this limitation is to develop a "metamodel". In other words, generate a "surrogate" model of the original system that accurately captures the relationships between input and output, yet it is computationally more efficient than simulation. Neural networks (NN) are known to be good function approximators and thus make good metamodel candidates. During training, a NN is presented with several input/output pairs, and is expected to learn the functional relationship between inputs and outputs of the simulation model. So, a trained net can *predict* the output for inputs other than the ones presented during training. This ability of NNs to generalize depends on the number of training pairs used. In general, a large number of such pairs is required and, since they are obtained through simulation, the metamodel development is slow. In DES simulation it is often possible to use perturbation analysis to also obtain sensitivity information with respect to various input parameters. In this paper, we investigate the use of sensitivity information to reduce the simulation effort required for training a NN metamodel.

1 INTRODUCTION

Simulation is arguably the most versatile and general-purpose tool available today for modeling complex systems such as Discrete Event Systems (DES). It can be used for performance evaluation, system design, decision making, and planning. Such applications typically involve the use of simulation to answer a multitude of "what-if" questions under various scenaria, each corresponding to different parameters, designs or decisions. However, simulation is notoriously time consuming. For complex systems, performance evaluation under a single set of input parameters can take several minutes even hours. As a result, it is impractical (if at all feasible) to perform any parametric study of system performance, especially for systems with a large parameter space. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach, even with supercomputers.

One alternative for achieving the required speedup is through "metamodeling". In this framework, any simulator is viewed as a function that maps any vector of input parameters $\mathbf{x} = [x_1, \cdots, x_N]$ to a set of M performance metrics of interest $\mathbf{y} = [y_1, \cdots, y_M]$, that is

$$\mathbf{y} = \Phi(\mathbf{x}). \tag{1}$$

Since the evaluation of the function $\Phi(\cdot)$ is generally complex and time consuming, metamodeling seeks a much simpler and computationally more efficient "surrogate model" $\hat{\Phi}(\cdot)$ such that

$$\hat{\Phi}(\mathbf{x}) \approx \Phi(\mathbf{x}). \tag{2}$$

for all input parameters of interest. A typical approach for building $\hat{\Phi}(\cdot)$ is to use simulation to obtain a training set of Q input-output pairs $\{(\mathbf{x}_i, \mathbf{y}_i),\ i = 1, \cdots, Q\}$, and try to determine a function that captures the input-output relationship that generated the Q samples. Of course, the expectation is that the surrogate model will be such that (2) will hold not only for the training pairs, but also for *any* $\mathbf{x}$ in some domain of interest. This ability of the surrogate model to produce a reasonable response to an input that is not included in the training set is referred to as *generalization*.

Several authors have addressed simulation metamodeling. Zeimer and Tew (1995) used polynomial fitting to develop a metamodel for a Tactical Electronic Reconnaissance Simulation Model (TERSM) that estimates the number of ground-based radar sites detected by a reconnaissance aircraft as a function of its flying mission. Other approaches have used statistical analysis. Santos and Nova (1999) used least squares estimation for non-linear metamodel estimation while Cheng (1999) used regression using Bayesian methods. Neural networks (NN) are generally known as good function approximators and thus make good candidates for surrogate functions. Jablunovsky et al. (2000) have used a back-propagation neural net to capture the behaviour of a Command and Control (C2) network. In some of our earlier work (Cassandras et al. 1998), we used a Cascade Correlation NN (Fahlman and Lebiere 1990) to generate metamodels for the TERSM mentioned above and for an Aircraft Refueling and Maintenance System (ARMS).

Neural network metamodels, though versatile, generally require a large number of training pairs before they acquire good generalization capabilities. This implies that many simulation runs are necessary to build a metamodel. To address this problem, in our earlier work (Cassandras et al. 1998) we also proposed the use of *Concurrent Estimation* (Cassandras and Panayiotou 1999) as a possible way of collecting more training pairs from a single simulation run. In this work we propose a different approach where we use sensitivity (derivative) information to train the neural network.

More specifically, it is by now well-documented in the literature that the nature of sample paths of Discrete Event Systems (DES) can be exploited so as to extract a significant amount of information, beyond merely an estimate of a performance measure $\Phi(\mathbf{x})$. It has been shown that observing a sample path under some parameter value x allows us to efficiently obtain estimates of derivatives of the form $d\Phi/dx$ which are in many cases unbiased and strongly consistent (e.g., see (Cassandras 1993; Glasserman 1991; Ho and Cao 1991) where Infinitesimal Perturbation Analysis (IPA) and its extensions are described). The question that arises then is how the sensitivity information, made available by PA, can be used to construct metamodels. In this paper we recognize that since (2) must hold for all $\mathbf{x}$ in the domain of interest, the partial derivatives of the two functions with respect to any x_i should also be equal. As a result, we modify the standard back-propagation algorithm to also use this information as explained in Section 4.

To our knowledge, this approach of using sensitivity information to reduce the training sample size is new. For the "classification" problem (as opposed to the "function approximation" problem we investigate in this paper) several authors have addressed the issue of determining the minimum training sample size for a neural network to have good generalization properties. Baum and Haussler (1989) found that for a network with N nodes and W weights, the number of randomly selected samples required to achieve correct classification for at least $(1 - \frac{\epsilon}{2})$ fraction of the test examples is $m \geq O\left(\frac{W}{\epsilon} \log \frac{N}{\epsilon}\right)$. Mehrotra et al. (1991) found a tighter bound assuming that the training samples are chosen close to the cluster boundaries. The generalization performance of practical algorithms is also the focal point in Takahashi and Gu (1998). These authors use the so called "ill-disposed" algorithm to derive a probability distribution that allows them to determine a more realistic bound on the sample size as well as the average generalization error.

This paper is organized as follows. In the next section we present the notation that we will use in the sequel. In Section 3 we briefly describe the standard back-propagation neural network and in Section 4 we modify the algorithm to include any available sensitivity information. In Section 5 we demonstrate the potential advantages of the approach with two numerical examples. Finally we close with conclusions and future plans in Section 6.

2 NOTATION

For the purposes of this paper, we assume a 3-layer neural network where

M	Number of units (neurons) in the output layer.
H	Number of units in the hidden layer.
N	Number of units in the input layer.
y_k	Output of the kth unit of the output layer, $k = 1, \cdots, M$.
z_j	Output of jth hidden unit, $j = 0, \cdots, H$ ($z_0 = 1$ corresponds to the bias input of output layer units).
x_i	Input of the ith unit, $i = 0, \cdots, N$ ($x_0 = 1$ corresponds to the bias input of hidden layer units).
$\mathbf{t}_p$	$= [t_k]_p$. Target output given an input vector $\mathbf{x}_p$, where $k = 1, \cdots, M$, $p = 1, \cdots, Q$ and Q is the number of training pairs.
d_{ki}	Sensitivity of kth network output with respect to its ith input, $d_{ki} = \frac{\partial y_k}{\partial x_i}$.
$\mathbf{S}_p$	$= [s_{ki}]_p$ Target sensitivity (d_{ki}) given an input vector $\mathbf{x}_p$. (For DES, we assume that this information is obtained through some Perturbation Analysis (PA) technique).
$f(\cdot)$	Activation function of output layer units.
$g(\cdot)$	Activation function of hidden layer units.

3 BACK-PROPAGATION NEURAL NET

The activation of the kth output unit of a standard, three layer back-propagation neural network (BPNN) as a function of the input $\mathbf{x} = [x_1, \cdots, x_n]$ is given by:

$$y_k = f\left(y_k^{IN}\right) \quad (3)$$

$$y_k^{IN} = \sum_{j=0}^{H} w_{jk} z_j \quad (4)$$

$$z_j = g\left(z_j^{IN}\right) \quad (5)$$

$$z_j^{IN} = \sum_{i=0}^{N} u_{ij} x_i \quad (6)$$

where, w_{jk} is the weight from the jth hidden unit to the input of the kth output unit and u_{ij} is the weight from the ith input to the jth hidden unit.

The learning procedure of the back-propagation neural network is based on minimizing the sum of squared errors. That is, minimize an error function of the form:

$$E = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2. \tag{7}$$

The minimization is done by gradient descent methods, where back-propagation involves the chain rule to back-propagate errors from the network's outputs to each of the network's weights, see (Fausett 1994) for details. Next, we investigate a possible way of utilizing the sensitivity information that can be obtained through some PA technique.

4 DERIVATIVE BACK-PROPAGATION NEURAL NETWORKS

If equation (2) is supposed to hold for all $\mathbf{x}$ in the domain of interest of $\mathbf{x}$, then it is reasonable to require that:

$$\frac{\partial\hat{\Phi}(\mathbf{x})}{\partial x_i} \approx \frac{\partial\Phi(\mathbf{x})}{\partial x_i}, \quad i = 1, \cdots, N. \tag{8}$$

In other words, the sensitivity of the neural net output with respect to each one of its inputs should be approximately equal to the sensitivity of the simulation model with respect to the same inputs. Though complex, it is possible to determine the neural network's sensitivity with respect to its inputs using calculus. Also, for several discrete event systems, the model's sensitivity with respect to its input parameters can be calculated using some perturbation analysis technique. Thus, the main idea behind our approach is to adapt (7) to account not only for the error in the output value, but for the error in the sensitivity as well. Therefore, the neural net training objective function becomes

$$E = \frac{\alpha}{2}\sum_{k=1}^{M}(t_k - y_k)^2 + \frac{\beta}{2}\sum_{k=1}^{M}\sum_{i=1}^{N}(s_{ki} - d_{ki})^2. \tag{9}$$

The first term is the usual error term used in standard back-propagation neural networks. The second term, is the error in the sensitivity of the neural net compared to the sensitivity of the model. Finally, $0 \le \alpha \le 1$ and $\beta = 1 - \alpha$ are weighting factors that determine the significance associated with the derivative error. Note that if $\beta = 0$, then we get the standard back-propagation algorithm.

Next, if we are interested in minimizing the error function of (9), we need to determine the neural network's sensitivity with respect to its inputs, d_{ki}. For the 3-layer network we consider in this paper, this is done by the chain rule of differentiation as shown below:

$$\begin{aligned} d_{ki} = \frac{\partial y_k}{\partial x_i} &= f'\left(y_k^{IN}\right)\frac{\partial y_k^{IN}}{\partial x_i} \\ &= f'\left(y_k^{IN}\right)\sum_{j=1}^{H} w_{jk}\frac{\partial z_j}{\partial x_i} \\ &= f'\left(y_k^{IN}\right)\sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right)\frac{\partial z_j^{IN}}{\partial x_i} \\ &= f'\left(y_k^{IN}\right)\sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right) u_{ij} \end{aligned} \tag{10}$$

Subsequently, if we want to use gradient based techniques to minimize the error function of (9) we need $\frac{\partial E}{\partial w_{jk}}$ and $\frac{\partial E}{\partial u_{ij}}$ for all i, j, k which are derived next, through repetitive use of the chain rule of differentiation.

$$\begin{aligned} \frac{\partial E}{\partial w_{JK}} &= -\alpha(t_K - y_K)\frac{\partial y_k}{\partial w_{JK}} - \beta\sum_{i=1}^{N}\left[e_{Ki}\frac{\partial d_{Ki}}{\partial w_{JK}}\right] \\ &= -\alpha(t_K - y_K) f'\left(y_K^{IN}\right)\frac{\partial y_K^{IN}}{w_{JK}} \\ &\quad -\beta\sum_{i=1}^{N}\left[e_{Ki}\left(f''\left(y_k^{IN}\right)\frac{\partial y_K^{IN}}{\partial w_{JK}} S(K,i)\right.\right. \\ &\quad \left.\left. + f'\left(y_K^{IN}\right) g'\left(z_J^{IN}\right) u_{iJ}\right)\right] \\ &= -\alpha(t_K - y_K) f'\left(y_K^{IN}\right) z_J \\ &\quad -\beta\sum_{i=1}^{N}\left[e_{Ki}\left(f''\left(y_K^{IN}\right) z_J S(K,i)\right.\right. \\ &\quad \left.\left. + f'\left(y_K^{IN}\right) g'\left(z_J^{IN}\right) u_{iJ}\right)\right] \end{aligned} \tag{11}$$

where

$$e_{ki} = (s_{ki} - d_{ki})$$

and

$$S(k,i) = \sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right) u_{ij}.$$

Similarly,

$$\begin{aligned} &\frac{\partial E}{\partial u_{IJ}} = \\ &= -\alpha\sum_{k=1}^{M}(t_k - y_k)\frac{\partial y_k}{\partial u_{IJ}} - \beta\sum_{k=1}^{M}\sum_{i=1}^{N}\left[e_{ki}\frac{\partial d_{ki}}{\partial u_{IJ}}\right] \end{aligned}$$

$$
\begin{aligned}
&= -\alpha \sum_{k=1}^{M} (t_k - y_k) f'\left(y_k^{IN}\right) \frac{\partial y_k^{IN}}{\partial u_{IJ}} \\
&\quad -\beta \sum_{k=1}^{M}\sum_{i=1}^{N} \Bigg[e_{ki} \Bigg(f''\left(y_k^{IN}\right) \frac{\partial y_k^{IN}}{\partial u_{IJ}} S(k,i) \\
&\qquad + f'\left(y_k^{IN}\right) \sum_{j=1}^{H} w_{jk} \frac{\partial}{\partial u_{IJ}} \left(g'\left(z_j^{IN}\right) u_{ij} \right) \Bigg) \Bigg] \\
&= -\alpha \sum_{k=1}^{M} (t_k - y_k) f'\left(y_k^{IN}\right) w_{Jk} \frac{\partial z_J}{\partial u_{IJ}} \\
&\quad -\beta \sum_{k=1}^{M}\sum_{i=1}^{N} \Bigg[e_{ki} \Bigg(f''\left(y_k^{IN}\right) w_{Jk} g'\left(z_J^{IN}\right) \frac{\partial z_J^{IN}}{\partial u_{IJ}} S(k,i) \\
&\qquad + f'\left(y_k^{IN}\right) w_{Jk} \cdot \\
&\qquad \left(g''\left(z_J^{IN}\right) \frac{\partial z_J^{IN}}{\partial u_{IJ}} u_{iJ} + g'\left(z_J^{IN}\right) \frac{\partial u_{IJ}}{\partial u_{IJ}} \right) \Bigg) \Bigg] \\
&= -\alpha \sum_{k=1}^{M} (t_k - y_k) f'\left(y_k^{IN}\right) w_{Jk} g'\left(z_J^{IN}\right) x_I \\
&\quad -\beta \sum_{k=1}^{M}\sum_{i=1}^{N} \Big[e_{ki} \Big(f''\left(y_k^{IN}\right) w_{Jk} g'\left(z_J^{IN}\right) x_I S(k,i) \\
&\qquad + f'\left(y_k^{IN}\right) w_{Jk} \left(g''\left(z_J^{IN}\right) u_{iJ} x_I + g'\left(z_J^{IN}\right) \right) \Big) \Big]
\end{aligned}
\tag{12}
$$

Note that these expressions get considerably simpler when the activation function of the output layer units are linear. In this case, $f(x) = x$, $f'(x) = 1$, and $f''(x) = 0$. Therefore,

$$
\begin{aligned}
\frac{\partial E}{\partial w_{JK}} &= -\alpha (t_K - y_K) z_J \\
&\quad -\beta g'\left(z_J^{IN}\right) \sum_{i=1}^{N} (s_{Ki} - d_{Ki}) u_{iJ}
\end{aligned}
\tag{13}
$$

and

$$
\begin{aligned}
\frac{\partial E}{\partial u_{IJ}} &= -\alpha g'\left(z_J^{IN}\right) x_I \sum_{k=1}^{M} (t_k - y_k) w_{Jk} \\
&\quad -\beta \sum_{k=1}^{M}\sum_{i=1}^{N} e_{ki} w_{Jk} \cdot \\
&\qquad \left(g''\left(z_J^{IN}\right) u_{iJ} x_I + g'\left(z_J^{IN}\right) \right)
\end{aligned}
\tag{14}
$$

Finally, the weight updates at every iteration t are given by

$$
w_{jk}(t+1) = w_{jk}(t) - \gamma \frac{\partial E}{\partial w_{jk}} \tag{15}
$$

and

$$
u_{ij}(t+1) = u_{ij}(t) - \gamma \frac{\partial E}{\partial u_{ij}} \tag{16}
$$

for all $i = 0, \cdots, N$, $j = 0, \cdots, H$, $k = 1, \cdots, M$. Where γ is the learning rate and $\frac{\partial E}{\partial w_{jk}}$ and $\frac{\partial E}{\partial u_{ij}}$ are given by (11) and (12) respectively (Note that $i = 0$ corresponds to the bias input of a neuron). In the sequel, this will be referred to as the Derivative Back-propagation Neural Network (DBPNN). At this point, it is worth pointing out that apart from the learning rate γ one needs to determine the weight to be given to the derivative error $\beta = 1 - \alpha$. This is an important factor that may affect the convergence of the algorithm as discussed later in the paper.

5 NUMERICAL RESULTS

In this section we present some preliminary results that show the potential benefit of using the DBPNN training algorithm described in equations (15) and (16).

In our first experiment we try to approximate the function $y = x^2$ in the interval $[-10, 10]$. For this experiment, we use a neural network with 20 hidden units. First, we used just three input-output pairs $\{(-10, 100), (0, 0), (10, 100)\}$ to train a standard back-propagation neural network (BPNN). Subsequently, we added the derivatives at these three points and used the information to train a network using DBPNN. For this experiment we used $\beta = 0.1$. The outputs of the two networks as well as the target output function are shown in Figure 1. As seen in the figure, the DBPNN approximates the target function much better than the standard BPNN. Also shown in the figure is the absolute value of the approximation error of each network for every value of x in $[-10, 10]$ which also demonstrates the benefit of DBPNN.

In order to compare the generalization ability of each network, we integrate the area under the error curve of each network and plot it in Figure 2 as a function of the number of points used during the training of the two networks. As seen in the figure, DBPNN achieves much better generalization than standard back-propagation neural net, especially when the number of training points is small.

Next we consider an $M/M/1$ queueing network where we are interested in the average time that customers spend in the system S, as a function of the traffic intensity ρ. For this system, the IPA algorithm for determining $\frac{dS}{d\rho}$ is given in (Cassandras 1993). Figure 3 shows the approximations generated by BPNN and DBPNN when both networks have 20 hidden units and are trained with only 5 train points

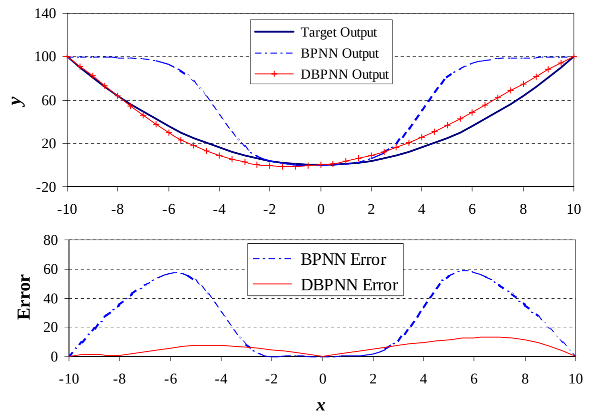

Figure 1: Approximation of the Function $y = x^2$ Using NNs

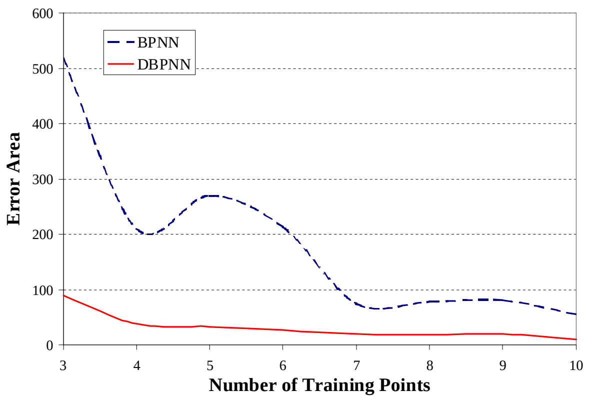

Figure 2: Area Under the Absolute Error of the NN Approximations for $y = x^2$

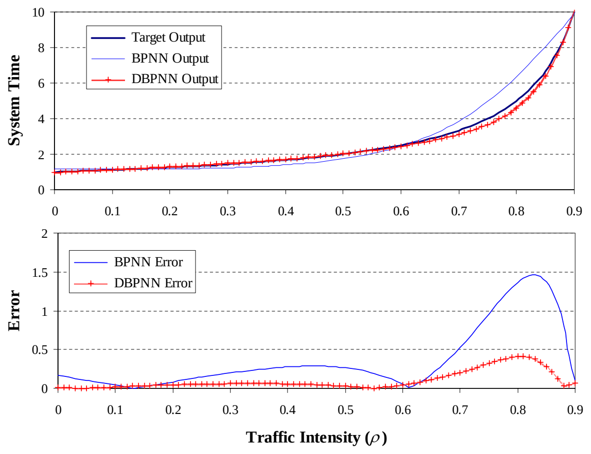

Figure 3: Approximation of the Average System Time in an $M/M/1$ Queueing System

and for DBPNN $\beta = 0.01$. As seen in the figure, DBPNN again achieves a much better generalization than the standard back-propagation network.

Finally, Figure 4 shows that area under the generalization error for the two networks. Again, DBPNN achieves a much better generalization than BPNN especially for a small number of training points.

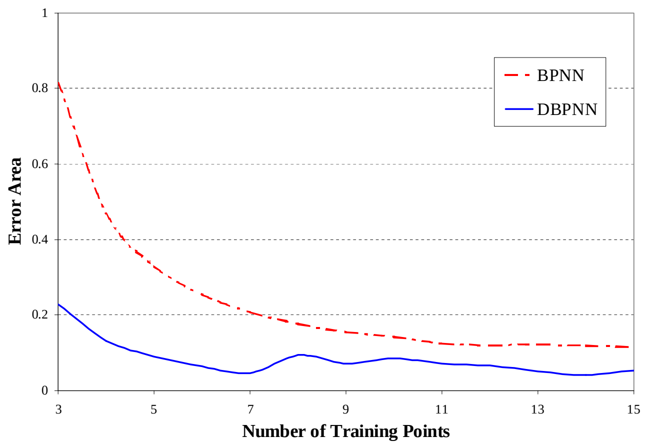

Figure 4: Area Under the Absolute Error of the NN Approximations for the Average System Time in an $M/M/1$ System

Note that for the second experiment we have set the β parameter to a very small value ($\beta = 0.01$). The reason is that as ρ approaches 1, the system time goes asymptotically to infinity and therefore the derivative at this point becomes very large. As a result, the derivative error for this point dominates the entire error function, so, the neural network in its effort to minimize the total error, it approximates *only* that derivative well, but does not approximate well the remaining points. Furthermore, we point out that other factors may also play a role in the value of the β parameter. For example, if we have noisy estimates of the derivatives, it may be preferable to also set β to a small value in order to avoid noise from taking over the output of the neural network. More research is required to determined how β is set.

6 CONCLUSIONS

When dealing with complex systems, simulation is usually the only alternative for performance evaluation however, it is notoriously slow, thus the need for "metamodels". Neural networks are considered as good function approximators thus make good metamodel candidates. However, if a neural net is to adequately learn the functional relationship between the inputs and outputs of a simulation model it requires a significant number of input/output pairs. Since such information can only be obtained through simulation, it means that the training phase of the neural network will be long. In this paper, we investigate the use of sensitivity information in the training of back-propagation neural network. Some preliminary results indicate that the use of sensitivity information can significantly reduce the number of training input/output pairs required which in turn implies that the metamodel development phase will be expedited.

The use of sensitivity information during training of neural networks creates several issues that need to be addressed and are part of our future plans. First, the addition of the derivative error in (9) makes the training objective function more complex. One problem that has been observed during our experiments is that this addition may create several local minima and as a result convergence issues may arise. Furthermore, the importance associated with the derivative errors (i.e., parameter β) needs to be further investigated. As mentioned earlier, this parameter may be critical to the quality of the approximation as well as the convergence of the training algorithm.

ACKNOWLEDGMENTS

This work was supported by AFRL under Contracts F30602-99-C-0056 and F30602-99-C-0057, by AFOSR under contract F49620-98-1-0387 and by ERPI/DoD under contract WO8333-03.

REFERENCES

Baum, E. B. and D. Haussler. 1989. What size net gives valid generalization. *Neural Computation* 1(1): 151–160.

Cassandras, C. G. 1993. *Discrete event systems, modeling and performance analysis*. IRWIN.

Cassandras, C. G., W.-B. Gong, C. Liu, C. G. Panayiotou, and D. L. Pepyne. 1998. Simulation driven metamodeling of complex systems using neural networks. In *Proceedings of SPIE*, 218–229.

Cassandras, C. G. and C. G. Panayiotou. 1999. Concurrent sample path analysis of discrete event systems. *Journal of Discrete Event Dynamic Systems* 9(2): 171–195.

Cheng, R. C. H. 1999. Regression metamodeling in simulation using Bayesian methods. In *Proceedings of Winter Simulation Conference*, 330–335.

Fahlman, S. and C. Lebiere. 1990. The cascade-correlation learning architecture. Technical report, Carnegie Mellon University CMU-CS-90-100.

Fausett, L. 1994. *Fundamentals of neural networks: Architectures, algorithms and applications*. New Jersey: Prentice Hall.

Glasserman, P. 1991. *Gradient Estimation via Perturbation Analysis*. Boston: Kluwer.

Ho, Y. and X. Cao. 1991. *Perturbation Analysis of Discrete Event Systems*. Boston: Kluwer.

Jablunovsky, G., C. Dorman, and P. Yaworsky. 2000. A neural network sub-model as an abstraction tool: Relating network performance to combat outcome. In *Proceedings of SPIE*, Orlando, Florida.

Mehrotra, K. G., C. K. Mohan, and S. Ranka. 1991. Bounds on the number of samples needed for neural learning. *IEEE Transactions on Neural Networks* 2(6): 548–558.

Santos, M. I. R. d. and A. M. O. P. Nova. 1999. The main issues in nonlinear simulation metamodel estimation. In *Proceedings of Winter Simulation Conference*, 502–509.

Takahashi, H. and H. Gu. 1998. A tight bound on concept learning. *IEEE Transactions on Neural Networks* 9(6): 1191–1202.

Zeimer, M. A. and J. D. Tew. 1995. Metamodel applications using tersm. In *Proceedings of Winter Simulation Conference*, 1421–1428.

AUTHOR BIOGRAPHIES

CHRISTOS G. PANAYIOTOU is a Research Associate at the Manufacturing Engineering Department, Boston University. He received a B.S. and a Ph.D. in Electrical Engineering, both from the University of Massachusetts at Amherst in 1994 and 1999 respectively. In 1999, he also received an MBA from the Eisenberg School of Management at the aforementioned University. His research interests include optimization and control of discrete-event systems, computer simulation, computer communication networks, manufacturing systems and transportation networks. His email and web addresses are respectively `<panayiot@bu.edu>` and `<http://control.bu.edu/panayiot>.`

CHRISTOS G. CASSANDRAS is Professor of Manufacturing Engineering and Professor of Electrical and Computer Engineering at Boston University. He received degrees from Yale University (B.S., 1977), Stanford University (M.S.E.E., 1978), and Harvard University (S.M., 1979; Ph.D., 1982). In 1982-84 he was with ITP Boston, Inc. where he worked on the design of automated manufacturing systems. In 1984-1996 he was a faculty member at the Department of Electrical and Computer Engineering, University of Massachusetts/Amherst. He specializes in the areas of discrete event systems, stochastic optimization, and computer simulation, with applications to computer networks, manufacturing systems, and transportation systems. He has published over 150 papers in these areas, and two textbooks. He has guest-edited several technical journal issues and serves on several editorial boards. Dr. Cassandras is currently Editor-in-Chief of the Transactions on Automatic Control and has served as Editor for Technical Notes and Correspondence and Associate Editor. He is a member of the CSS Board of Governors, chaired the CSS Technical Committee on Control Theory, and served as Program Chair of various conferences. He is the recipient of the 1999 Harold Chestnut Prize (IFAC Best Control Engineering Textbook) for Discrete Event Systems: Modeling and Performance Analysis, a 1991 Lilly Fellowship, a member

of Phi Beta Kappa and Tau Beta Pi, and a Fellow of the IEEE. His email and web addresses are `<cgc@bu.edu>` and `<http://vita.bu.edu/cgc>`.

WEI-BO GONG is Professor of Electrical and Computer Engineering at the University of Massachusetts/Amherst. He received SM and PhD degrees from Harvard University. He has been a mathematical software consultant for the Lotus Development Corporation and a consultant for modeling the air traffic control systems for the MIT Lincoln Laboratory. His research interests include simulation, approximation, hierarchical modeling, optimization and information fusion in discrete event systems. His research is supported by NSF, ARO, AFOSR and Rome Lab. His publications are on the subjects of perturbation analysis, integer and stochastic optimization, queueing systems, modeling and control of communication networks, rational approximation for stochastic discrete event systems and general discrete event system theory. In 1997 he received the IEEE Transactions on Automatic Control George Axelby Outstanding Paper Award, was elected an IEEE Fellow in 1999. He is on the Editorial Board of Journal of Optimization Theory and Application, and has been an Associate Editor for IEEE Transactions on Automatic Control and IEEE Transactions on Robotics and Automation. His email and web addresses are `<gong@ece.umass.edu>` and `<http://www-unix.ecs.umass.edu/%7egong/>`.