Genetic Algorithm based Input Selection for a Neural Network Function Approximator with Applications to SSME Health Monitoring

Charles C. Peck and Atam P. Dhawan
Dept. of Electrical and Computer Engineering
University of Cincinnati Cincinnati, OH 45221

Claudia M. Meyer
Sverdrup Technology, Inc.
NASA Lewis Research Center Group
Brook Park, OH 44142

ABSTRACT

A genetic algorithm is used to select the inputs to a neural network function approximator. In the application considered, modeling critical parameters of the Space Shuttle Main Engine (SSME), the functional relationship between measured parameters is unknown and complex. Furthermore, the number of possible input parameters is quite large. Many approaches have been used for input selection, but they are either subjective or do not consider the complex multivariate relationships between parameters. Due the optimization and space searching capabities of genetic algorithms they were employed in this paper to systematize the input selection process. The results suggest that the genetic algorithm can generate parameter lists of high quality without the explicit use of problem domain knowledge. Suggestions for improving the performance of the input selection process are also provided.

I. INTRODUCTION

There is considerable interest within the space industry in improving the fault detection and isolation capabilities of rocket engine condition monitoring and post-test diagnostic processing systems. This requires developing accurate models of engine parameters based on other parameters measured from the engine. Developing accurate models is particularly difficult due to the highly complex, non-linear nature of rocket engines, the limited suite of measured parameters, and the large variability of behavior among engines of the same design.

It has been shown that neural networks with one hidden layer can uniformly approximate any continuous function [1, 2, 3]. Furthermore, neural networks are well-suited for problems in which the exact relationships between inputs and outputs are complex or unknown [4, 1]. These conclusions may be applied to dynamical systems if the system state is sufficiently represented in the inputs of the neural network. For these reasons, feedforward neural networks have been used to model critical parameters of the Space Shuttle Main Engine (SSME) during the start-up transient and they have been shown to be quite effective [4].

A task that is critical to the success of neural network modeling of complex, dynamical systems such as the SSME is the choice of the input parameters. There are several constraints that complicate this task. First, while the instrumentation of the SSME is extensive, it is not complete. Therefore, it is unlikely that it will be possible to completely describe any subsystem input or output. Second, as was discussed above, it is necessary to provide enough state information to model the desired parameter. Finally, it is not practical to use a large number of inputs for a number of reasons. First, a time window of each input parameter is typically used in order to provide time dependent information. The size of the window multiplies the number inputs to the network. For example, if 10 parameters are chosen as the network inputs and a time window of the past ten values is used, then the effective number of inputs to the network is 100. Another reason that the input set should be restricted is that large networks are difficult to train. Finally, the input set should

be small if the system is to be used for real-time modeling.

A number of ad hoc approaches have been proposed or used for input selection. These include the use of characteristic equations, engine schematic analysis, correlations between candidate input parameters and the modeled parameter, and expert advice. These methods are highly subjective or they do not adequately measure the multivariate dependencies present in the system. For these reasons, a systematic approach for input selection is desired.

The choice of inputs may be modeled as an optimization problem where the space of possible solutions is quite large. In fact, roughly 500 sensors are used for monitoring during test firings of the SSME. This represents approximately 2^{500} distinct input sets. Since an exhaustive search is clearly not possible, an alternative search method is required.

Genetic algorithms are well suited for searching in a large parameter space [5, 6]. Through the use of seeding (the process of providing an initial set of possible solutions), genetic algorithms search from a set of solutions or starting points, rather than a single starting point. Genetic algorithms are not derivative based, thus they can search spaces where methods such as conjugate descent fail. They work with both discrete and continuous parameters, and explore and exploit the parameter space [7]. Furthermore, through the use of elitism (a variant method in which the best solution of a generation is promoted unaltered to the next generation), a genetic algorithm can be guaranteed to perform at least as well the methods used to seed or initialize it. For these reasons, a genetic algorithm was used in this paper to select the inputs to a neural network used for SSME parameter modeling during the start-up transient.

This paper will first present the design issues and methodology applied to the selection of SSME input parameters. A presentation and discussion of results will follow. The paper will conclude with the conclusions and ideas for future work.

II. DESIGN ISSUES AND METHODOLOGY

The design issues range from those applicable to all genetic algorithms and multi-layered perceptron neural networks to those specific to this particular problem of SSME parameter approximation.

There are two fundamental design requirements for applying genetic algorithms: encoding candidate solutions onto binary strings, and developing a fitness function. In this case, encoding candidate solutions onto binary strings is trivial since a single bit is sufficient to indicate whether a particular parameter is to be included in the network input set. Accordingly, the string, or *chromosome*, has one bit for every candidate engine parameter. To reduce the size of the search space, redundant sensor measurements were eliminated and those parameters believed to be nearly independent of the modeled parameter were not included in the candidate parameter set. This reduced the size of the candidate parameter set to 49 parameters. Before discussing the development of the fitness function, it should be noted that in the genetic algorithm used, the smaller the fitness function value, the better the evaluated solution is considered to be.

The choice of a fitness function is somewhat more complicated than the string encoding. Recall that the primary function of the genetic algorithm is to produce input sets that enable neural network function approximators to accurately learn and generalize the relationships between the modeled parameter and the input parameters. One way to do this is to make the fitness function proportional to the neural network training error. Adding the input set size constraint to the fitness function could be done simply by multiplying the training error by the number of parameters selected. This results in a very strong constraint, however. The strength of the size constraint can be controlled by adding a constant to the number of parameters selected. A small offset created in this manner yields a strong size constraint, whereas a large offset yields a weak one. The fitness function may be further tweaked by squaring the size constraint term. This increases the strength of the constraint as the number of parameters increases.

The additional need to minimize the number of inputs to the network and the disparity in the size between heuristically and randomly selected seeding sets are primarily responsible for the added complexity of the fitness function. The heuristically selected seeding sets consist of approximately 10 parameters, while the randomly selected seeding sets consist of approximately 25 parameters. If the two seeding sets were approximately the same size, an offset could be chosen that would yield the desired input set size at the

end of the evolution process. This size disparity, however, results in either a strongly biased choice of input parameters or it results in input sets that are too large. To see this, consider the use of an offset sufficient to reduce the randomly selected seeding sets to a target size of 8 parameters. Due to the size disparity, the heuristically selected seeding sets would have considerably lower fitness function values and would thus dominate in successive generations. Conversely, the use of an offset that does not significantly favor the heuristically selected seedings may not significantly reduce the size of the parameter lists.

For the work presented in this paper, generation dependent offsets were used to avoid biasing the results while ensuring satisfaction of the size constraint. Initially, the offset was set very high to allow the candidate solutions to compete primarily on the basis of the training error. As the genetic algorithm proceeded, the size constraint was made progressively stronger. By the end of the genetic algorithm the offset was small, yielding a strong bias for shorter lists. This change of offset with respect to the generation will be referred to as an offset progression. Two offset progressions were used: one yielding a generally weak size constraint, and another yielding a generally strong size constraint. The offset progression yielding the weaker size constraint ranged from 71 initially to 14 over 20 generations. The other ranged from 45 initially to 7 over 20 generations. The resulting fitness functions are shown in Equations 1 and 2, respectively:

$$f = \frac{(c + 71 - 3G)^2}{(71 - 3G)^2} \times \text{Training Error}, \tag{1}$$

$$f = \frac{(c + 45 - 2G)^2}{(45 - 2G)^2} \times \text{Training Error}, \tag{2}$$

where f is the fitness function value, c is the number of parameters in the candidate input list, and G, which ranges from 0 to 19, is the generation number.

To ensure robustness and resistance to domination by "Super Individuals" (i.e., non-optimal solutions that are significantly more fit than other solutions early in the evolution process), the evolutionary process was designed to run in two stages. In the first stage, three populations were independently evolved. These populations were used to seed a second evolutionary stage. In the first stage, fitness functions with weaker size constraints were used. This favors lower training error. In the second stage, the fitness function with the stronger size constraint was used.

To further increase diversity within the "gene pool," the fitness functions in two of the first stage genetic algorithms were varied to favor either early or late convergence of the neural network training error. The method used to implement these biases exploits the observation that the training error consistently remained on a high plateau before falling precipitously, as shown in Figure 1. Since oscillations and unusual patterns in the training error were not observed, integration of the area bounded by the error curve and a bounding rectangle could be performed. To favor early convergence, the fitness function in Equation 1 was multiplied by the area of integration normalized by the area of the bounding box. If A, B, and C denote the normalized areas of their corresponding regions in Figure 1, the shape dependent fitness term is A for the early training error curve and $A + B$ for the late training error curve. To favor late convergence, the normalized area of integration is first subtracted from 1 before multiplying Equation 1. This corresponds to a shape dependent fitness term of $B + C$ for the early training error curve and C for the late training error curve.

As described above, the fitness function evaluation involves creating a neural network, training it, and evaluating its performance. This is computationally expensive and time consuming. To limit the cost of performing this operation, the QuickProp learning algorithm was used [8]. Furthermore, the network was trained only as far as necessary to distinguish it from other networks with different input configurations. It was determined empirically that 100 epochs is sufficient. According to the analysis provided in [8], this should be comparable to 1000 epochs of training with standard backpropagation. Finally, the neural networks were presented with a time window of 5 past values instead of the 10 past values used in [4].

Another important design consideration is that the training error of a network is a noisy fitness evaluation function. The weight initialization can have a significant effect on the performance of a network. Thus, to avoid biasing the fitness of a particular candidate set of inputs as either too good or too bad, the fitness of each candidate input set was evaluated every generation in which it was present.

III. RESULTS AND DISCUSSION

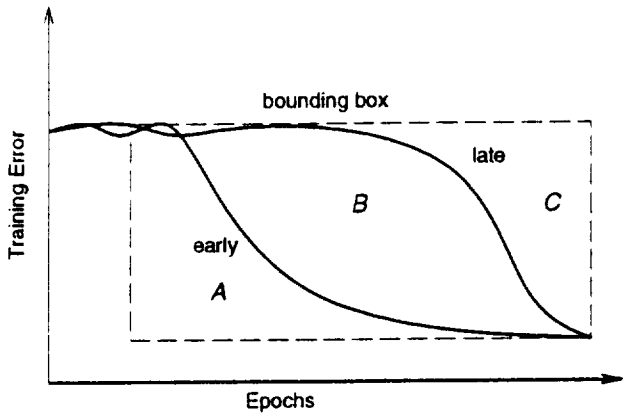

Figure 1: Early versus Late Training Error Convergence

Table 1: Parameter Lists

Parameter List	*Number of PIDs*	*Parameters*
GA-1	6	21 58 209 734 951 1050
GA-2	7	21 58 209 327 734 951 1058
GA-3	8	21 52 58 209 327 734 951 1050
REF	9	40 42 59 231 480 1205 1212 O/Cs OPBs

The fundamental output of the system described above consists of candidate parameter lists. The three parameter lists with the best fitness values are presented in Table 1. These three lists are labeled GA-1–GA-3. An additional list, labeled REF, is also presented for the purpose of comparison. This "reference" list has been modified from the one presented in [4] to exclude autoregressive information.

The parameter that was modeled is the SSME's High Pressure Oxidizer Turbine (HPOT) discharge temperature, which has a Parameter IDentification number (PID) of 233. Descriptions of this PID and the others included in the four lists described above are provided in Table 2.

To evaluate the performance of the parameter lists produced by the genetic algorithms, feedforward neural networks were fully trained using these lists and the reference list. The resulting networks were then used to approximate PID 233 using measured parameters from 12 actual SSME test firings. Four of the test firings were used for training the networks and eight were used to validate the resulting models. The results, as represented by the mean squared error (MSE), the normalized MSE, and the maximum percent error, are shown in Tables 3, 4, 5, and 6. A summary of these results is presented in Tables 7 and 8. The results are divided into two groups: one presenting the aggregate performance of the networks on the training data (Table 7), and the other presenting the aggregate performance of the networks on the validation data (Table 8).

It is clear from the results that the parameter list GA-1 has the worst error performance of the four lists. This is compensated by the fact that this is the shortest parameter list. Even though the error performance of this parameter list is the worst, it is still close to the performance of the other lists, including the reference list.

The parameter lists GA-2 and GA-3 outperformed the reference list on the training data and performed only slightly worse than the reference list on the validation data. Due to the large standard deviations of validation data error, the differences in the error means cannot be considered statistically significant.

Table 2: Parameter Descriptions

PID	*Description*
21	Main Combustion Chamber Oxidizer Injection Temperature
40	Oxidizer Preburner Oxidizer Valve Actuator Position
42	Fuel Preburner Oxidizer Valve Actuator Position
52	High Pressure Fuel Pump Discharge Pressure
58	Fuel Preburner Chamber Pressure
59	Preburner Boost Pump Discharge Pressure
209	High Pressure Oxidizer Pump Inlet Pressure
231	High Pressure Fuel Turbine Discharge Temperature
233†	High Pressure Oxidizer Turbine Discharge Temperature
327	High Pressure Oxidizer Pump BalCav
480	Oxidizer Preburner Chamber Pressure
734	Low Pressure Oxidizer Pump Speed
951	High Pressure Oxidizer Pump Pressure SL DR
1050	Oxidizer Tank Discharge Temperature
1058	Engine Oxidizer Inlet Temperature
1205	FAC Fuel Flow
1212	FAC Oxidizer Flow
O/Cs	Dummy Parameter indicating Open/Closed Loop Operation
OPBs	Dummy Parameter indicating Oxidizer Preburner Prime Time

† the modeled parameter

Table 3: Error Statistics from Parameter List GA-1

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	3.787033	0.000322	2.2330
B1060	T	14.743364	0.001223	4.8150
B1061	V	20.168583	0.001657	10.4348
B1062	V	34.029559	0.002832	9.6225
B1063	V	39.671779	0.003301	6.9063
B1066	V	30.608499	0.002532	7.5330
B1067	V	42.103255	0.003498	9.2189
B1070	T	11.699498	0.000945	3.1922
B1071	V	63.607371	0.005154	20.8187
B1072	V	23.816642	0.001898	8.3420
B1075	V	20.268258	0.001669	10.0018
B1077	T	12.931541	0.001045	5.3681

Table 4: Error Statistics from Parameter List GA-2

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	3.341027	0.000284	2.0253
B1060	T	6.059692	0.000503	3.1339
B1061	V	19.080619	0.001568	5.9461
B1062	V	37.601837	0.003129	9.9597
B1063	V	35.212338	0.002930	6.6999
B1066	V	33.799122	0.002796	7.3425
B1067	V	36.724494	0.003051	7.9440
B1070	T	10.692421	0.000864	3.6021
B1071	V	48.479267	0.003929	15.9965
B1072	V	17.781945	0.001417	5.1814
B1075	V	35.017457	0.002884	11.4959
B1077	T	7.973934	0.000644	2.6040

Table 5: Error Statistics from Parameter List GA-3

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	4.015642	0.000341	1.7042
B1060	T	6.114787	0.000507	2.5343
B1061	V	20.477665	0.001682	6.2484
B1062	V	40.542411	0.003374	10.5837
B1063	V	38.320758	0.003188	7.1349
B1066	V	38.782970	0.003208	8.8021
B1067	V	39.245907	0.003261	8.4381
B1070	T	10.516996	0.000850	3.2462
B1071	V	53.008396	0.004296	18.9413
B1072	V	17.990471	0.001434	4.7040
B1075	V	33.172788	0.002732	12.0369
B1077	T	6.889614	0.000557	2.3684

Table 6: Error Statistics from Parameter List REF

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	6.652181	0.000565	3.8462
B1060	T	7.375382	0.000612	3.0370
B1061	V	22.370471	0.001838	4.8509
B1062	V	23.747774	0.001976	7.2832
B1063	V	28.076726	0.002336	7.9618
B1066	V	16.538060	0.001368	7.6115
B1067	V	20.482848	0.001702	6.6011
B1070	T	6.588053	0.000532	3.8668
B1071	V	50.654580	0.004105	11.0878
B1072	V	42.897089	0.003419	6.7544
B1075	V	25.213499	0.002077	9.4449
B1077	T	7.809484	0.000631	4.5456

Table 7: Summary of Parameter List Performance on Training Data

Parm. List	*MSE*		*NMSE*		*Max.*	
	μ	σ	μ	σ	μ	σ
GA-1	10.790359	4.833357	0.000884	0.000392	3.902086	1.445958
GA-2	7.016768	3.101272	0.000574	0.000244	2.841333	0.679829
GA-3	6.884260	2.709112	0.000564	0.000212	2.463281	0.633292
REF	7.106275	0.589258	0.000585	0.000045	3.823920	0.617108

Table 8: Summary of Parameter List Performance on Validation Data

Parm. List	*MSE*		*NMSE*		*Max.*	
	μ	σ	μ	σ	μ	σ
GA-1	34.284241	14.485731	0.002818	0.001180	10.359750	4.397353
GA-2	32.962132	10.068242	0.002713	0.000832	8.820735	3.563816
GA-3	35.192673	11.349328	0.002897	0.000937	9.611175	4.430940
REF	28.747631	11.808645	0.002353	0.000932	7.699442	1.889502

It should be noted that the heuristically chosen parameter lists that were used to seed the genetic algorithms were outperformed early in the process by genetic algorithm generated parameter lists. While the behavior and results of the genetic algorithm were certainly affected by the heuristically chosen parameter sets, the guidance provided by these sets did not appear to be strong.

IV. CONCLUSIONS AND FUTURE WORK

The results indicate that the error performance of the genetic algorithm generated parameter lists was roughly the same as that of the reference list. Furthermore, in all cases, the genetic algorithm generated parameter lists were smaller than the reference list. Thus, the genetic algorithm was able to systematically generate parameter lists that performed well without the explicit use of problem domain knowledge.

Many improvements for the input selection process have been envisaged. One may, for example, modify the fitness evaluation function to be dependent on the error on a validation set instead on the training. This would favor parameter lists that yield networks with superior generalizing capabilities instead of lists that yield networks capable of rapid learning. As an extension, the fitness function could be made a function of the training error, the validation error, and the generation. In this manner, learning capability could be favored early in evolution and generalization could be favored later.

As demonstrated by the GA-1 list, smaller size can be overemphasized compared to the error performance. Instead of favoring a parameter list of the smallest size, a list of a particular size could be favored. This would favor the inclusion of sufficient information while discouraging the use of parameters that do not significantly improve the error performance. For this particular application, a size of 10 would be reasonable.

V. ACKNOWLEDGMENTS

The public domain genetic algorithm GENESIS Version 5.0, written by John J. Grefenstette, was used for the work described in this paper. Furthermore, the fitness evaluation function is a highly modified and optimized derivative of Terry Regier's implementation of the QuickProp training algorithm.

VI. REFERENCES

[1] S. Chen, S. A. Billings, and P. M. Grant. Non-linear systems identification using neural networks. Research Report 370, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, August 1989.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[3] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[4] Claudia M. Meyer and William A. Maul. The application of neural networks to the ssme startup transient. Number 2530 in 91. AIAA, July 1991.

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[6] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[7] David J. Powell, Michael M. Skolnick, and Siu Shing Tong. *Handbook of Genetic Algorithms*, chapter 20, pages 312–331. Van Nostrand Reinhold, New York, 1991.

[8] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo, CA, June 1988. Carnegie Mellon University, Morgan Kaufmann Publishers.