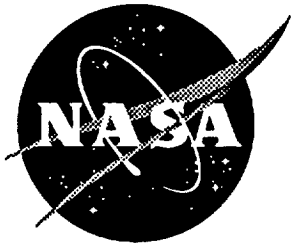
NASA Technical Memorandum 110154



Reducing Neural Network Training Time With Parallel Processing

James L. Rogers, Jr.
Langley Research Center, Hampton, Virginia

William J. LaMarsh II
Computer Sciences Corporation, Hampton, Virginia

February 1995

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

REDUCING NEURAL NETWORK TRAINING TIME WITH PARALLEL PROCESSING

James L. Rogers
NASA Langley Research Center

William J. LaMarsh II
Computer Sciences Corporation

ABSTRACT

Obtaining optimal solutions for engineering design problems is often expensive because the process typically requires numerous iterations involving analysis and optimization programs. Previous research has shown that a near optimum solution can be obtained in less time by simulating a slow, expensive analysis with a fast, inexpensive neural network. A new approach has been developed to further reduce this time. This approach decomposes a large neural network into many smaller neural networks that can be trained in parallel. Guidelines are developed to avoid some of the pitfalls when training smaller neural networks in parallel. These guidelines allow the engineer: to determine the number of nodes on the hidden layer of the smaller neural networks; to choose the initial training weights; and to select a network configuration that will capture the interactions among the smaller neural networks. This paper presents results describing how these guidelines are developed.

INTRODUCTION

Analysis programs used to solve engineering design problems are often computationally expensive. Numerous iterations between the analysis program and an optimization program are typically required to obtain an optimal solution. This process can become prohibitive because of the computer time required for convergence. Therefore, any new techniques that could significantly reduce the required computer time in solving a complex design problem would be beneficial.

One promising new technique is the simulation of a slow, expensive analysis program with a fast, inexpensive neural network. A

backpropagation neural network was selected for this simulation. Backpropagation neural networks were originally introduced in 1969 (Bryson and Ho, 1969). Only a brief introduction to neural networks is provided in this paper; other references can provide more background if necessary (Jones and Hoskins, 1987; Lippmann, 1987).

Recently, engineers have applied neural networks to structural mechanics problems (Rehak et al, 1989; VanLuchene and Roufei, 1990; Berke and Hajela, 1991; and Swift and Batill, 1991). For example, Berke and Hajela (1991) found optimum designs of trusses with a neural network. The input data consisted of lengths and heights for the truss; the output data consisted of the optimized bar areas and the total weight of the truss. After the neural network was trained, new optimal truss designs were found by propagating different sets of input data through the neural network. VanLuchene and Roufei (1990) applied a neural network to simulate the structural analysis of a simply supported rectangular plate. The analysis predicted the location and magnitude of the maximum moment. In their conclusions, the authors suggested additional study in the use of neural networks to solve civil-engineering optimization problems.

To address this issue, a new tool called NETS/PROSSS (Rogers and LaMarsh, 1991), which couples a backpropagation neural network called NETS (Baffes, 1989) to an optimization system called the Programming System for Structural Synthesis (PROSSS) was developed (Sobieszczanski-Sobieski and Bhat, 1979; Rogers et al, 1981; and Rogers, 1982). The PROSSS was developed in 1979 to provide a system for coupling analysis and optimization and is used for the reference optimization process for this project. Although this system was designed to handle any type of analysis program, most of the work to date has evolved around coupling a structural analysis program with an optimization program. In NETS/PROSSS, the neural network simulates the structural analysis program.

A trade-off must be considered in deciding whether to simulate structural analysis with a neural network in an optimization process. This trade-off centers around the amount of time spent in obtaining the optimum design. First, the engineer estimates the time required for a single analysis. The analysis portion of the optimization process typically executes sequentially. Because the analyses required to develop training data for the neural network are independent, they can be executed in parallel if more than one computer is available.

Therefore, because of parallelization, the execution of analysis programs to generate training data for neural networks may require less time to generate the same amount of analysis results. Although training the neural network accumulates time without yielding analysis results; once a neural network has been trained, it yields analysis results with an insignificant time penalty. At some point, a decision must be made whether time should be spent training the neural network to simulate the analysis program or simply executing the reference optimization process with the analysis program.

From previous research (Rogers, 1994), guidelines have been developed for designing and training a neural network to simulate a structural analysis program in an optimization process. The previous research is reviewed in the background section and the guidelines are summarized there. By following these guidelines, the number of analyses required to develop the training data and the time required to train the neural network was reduced. The resulting neural network proved to be faster than the reference optimization process by providing a reasonable approximation of the analysis results and achieving a near optimum design in less time. However, further reductions in training time may be obtained by taking advantage of parallel processing to develop the training data and to train the neural network. After the background section, this paper presents an approach by which a large neural network can be decomposed into several smaller neural networks that can be trained in parallel.

BACKGROUND

In a previous research project (Rogers and LaMarsh, 1992), NETS/PROSSS was applied to the optimization of a simple cantilever-beam problem. The motivation for this project was to determine the feasibility of using this new tool in an optimization process. The NETS/PROSSS approach followed the same convergence path as PROSSS in the optimization process; however, NETS/PROSSS (including the training time for the neural network) converged in approximately one-third the time required for PROSSS. These results were encouraging: NETS/PROSSS is a feasible optimization tool, a neural network can successfully simulate a structural analysis program, and an optimum design can be achieved in less time.

Although this project was successful, a concern existed that the finite-element model was too simple, which simplified this simulation of a structural analysis program by a neural network. As

a result, a second project (Rogers, 1994) was designed in which the complexity of the finite-element model was increased by adding a support and moving the load from the end to the middle of the beam. The placement of the design variables was changed, and the initial design variables (heights) were halved. In the previous project, only two methods were tried for selecting training data. For the second project, two additional methods, one based on statistical selection and the other on random selection, were also tested.

In addition, Carpenter and Barthelemy (1992) questioned whether an underdetermined neural network could achieve an adequate approximation. The number of "unknowns" in a neural network depends on the number of weights, the number of hidden layers, and the number of nodes on the hidden and output layers. The number of "knowns" depends on the number of nodes in the output layer and the number of training pairs. This yields the relation:

(output nodes * training pairs) = (input nodes * hidden nodes) + (hidden nodes * output nodes) + hidden nodes + output nodes.

One hidden layer was deemed sufficient for this simulation. Therefore, to reduce the time needed to design and train a neural network for the adequate simulation of a structural analysis program, two questions were addressed. What is the best way to select training data and what is the appropriate number of nodes on the hidden layer? A complete analysis must be executed to generate one training pair; as a result, these questions must be answered because they pertain to the number of training pairs required for an adequate approximation of the design space.

The second project (Rogers, 1994) addressed these two questions by trying four different methods for selecting training data. Three neural networks, each with a different number of nodes on the hidden layer, were trained with each of these four sets of training pairs. The output from these 12 combinations were compared to a reference optimization, and the best combination was selected for further processing to determine whether this technique could yield an optimal design in less time. All work was performed on a Sun Sparc2 workstation, which requires 3 min. per analysis execution, compared with the previous project in which the analysis was executed on a Dec MicroVax and required 20 min. per execution.

Based on these results, the following guidelines were developed for using neural networks to simulate structural analysis in the optimization process. More testing is needed to determine if this process is viable with other analysis programs.

1. Create a neural network with the sum of the nodes from the input and output layers on the hidden layer.

2. Build a hypercube around the initial design and add points at the upper and lower bounds of the design variables. For n design variables, $2*n + 3$ design combinations will result.

3. Perform analyses for the design combinations in step 2 to create training pairs.

4. Train the neural network and save the neural network weights.

5. Run NETS/PROSSS to find an approximate optimum, start from the initial design with the saved weights from step 4.

6. Build a hypercube around the approximate optimum design. This yields an additional $2*n + 1$ design combinations.

7. Perform analyses for the design combinations in step 6 to create additional training pairs.

8. Use both sets of training pairs ($4*n + 4$ total) and the saved weights from step 4 to train the network, and again save the weights from this training.

9. Execute NETS/PROSSS with the final design from step 5 as the initial design and the saved weights from step 8 to improve the design.

10. To check the results, restart PROSSS with the final design from step 9 as the initial design.

A total of 198 min. was required to reach an optimum solution with the reference optimization and structural analyses. By following these guidelines, the optimum solution was obtained in 159 minutes with a neural network to simulate the structural analyses. In this current project, an attempt is made to further reduce this time by

taking advantage of parallelization in developing the training pairs and training the neural networks.

THE TEST PROBLEM

The test problem is to optimize the shape of a beam to minimize the weight (the objective function) and satisfy stress constraints. A beam with 3000 degrees-of-freedom (DOF) 1025 joints, and 640 three-dimensional solid brick elements is used for the finite-element model (fig. 1).

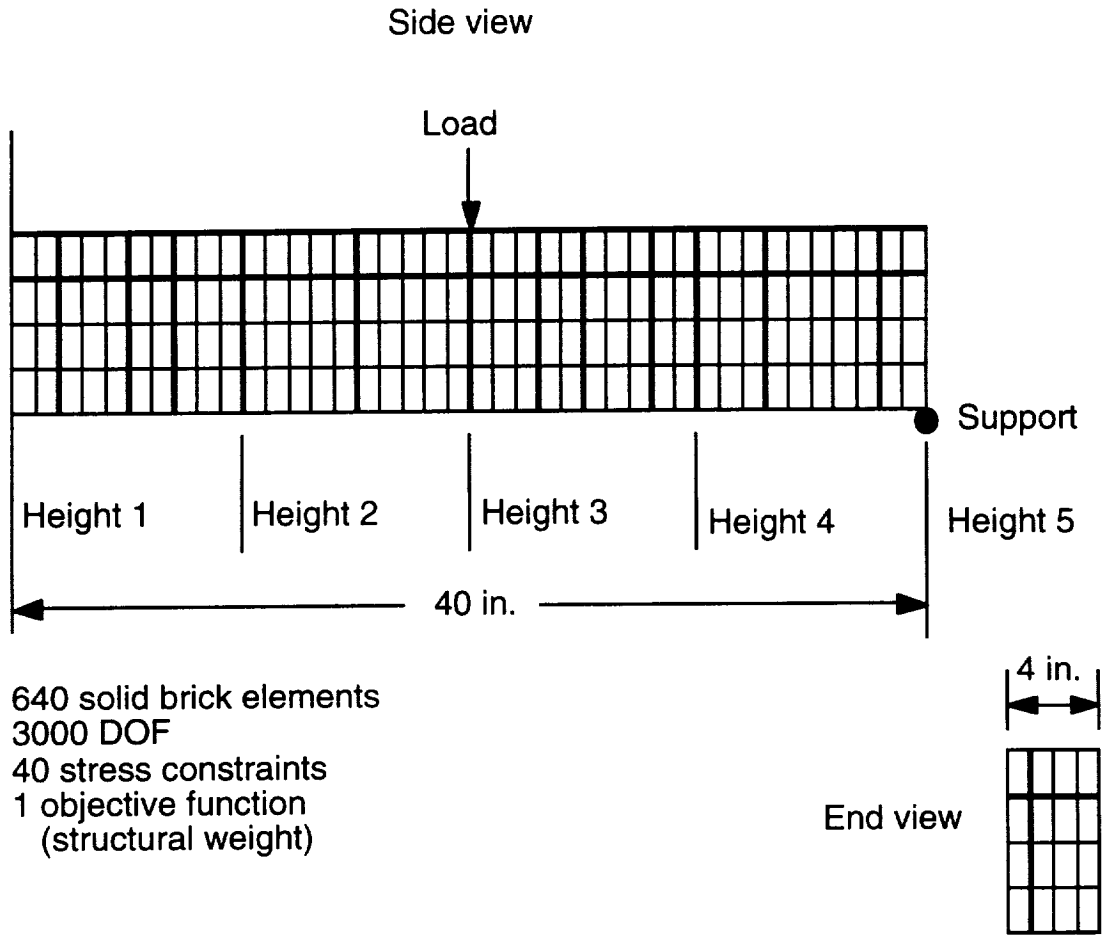


Figure 1. Initial design for beam test problem.

The beam is 40 in. long, 4 in. wide and begins with a height of 4 in.. The problem has five design variables that determine the shape of the beam by specifying the heights of the elements. Each design variable controls the height of a block of 160 elements. Forty cumulative stress constraints (Barthelemy and Riley, 1988) exist, one

for each of 40 stations (16 elements per station) along the beam. The beam is made of a material with a modulus of elasticity of $35.9*10^6$ psi, a weight per unit volume of 0.283 lb/in^3 , and an allowable stress of $150*10^3$ psi. A total load of 10,000 lb is applied in the z direction and distributed at the 25 joints in the section at the middle of the beam (for details see Rogers and LaMarsh, 1992). A support exists at the end of the beam.

REFERENCE OPTIMIZATION PROCESS

The PROSSS iterates between the structural analysis program and the optimizer. Based on current design variables, the structural analysis program computes the stresses (which are converted into constraints) and the objective function (weight); the optimizer uses the current design variables, the stress constraints, and the objective function to arrive at a new design (updated design variables). This iterative process continues until it converges to an optimum design (fig. 2). Convergence usually means that the objective function has not changed more than some given tolerance within the last three optimization cycles.

Several options are available in PROSSS to accomplish the optimization process. Because the objective function for this problem is linear and no analytical gradients are available, the PROSSS option with a piecewise linear analysis in which the gradients are computed by finite differencing that is external to the optimizer is selected. This option requires six structural analyses to be executed in each optimization cycle; each analysis requiring about 3 min. of computing time.

For the initial design, all design variables are set at 4 in.. The reference optimization process requires 198 min. and 11 cycles to converge; each cycle (6 analyses) requires 18 min. of computing time. For each of the first 5 cycles, the design variables are limited to a maximum change of 30 percent (move limits of 30 percent) to speed up convergence. The maximum change is decreased to 5 percent for the last 6 cycles.

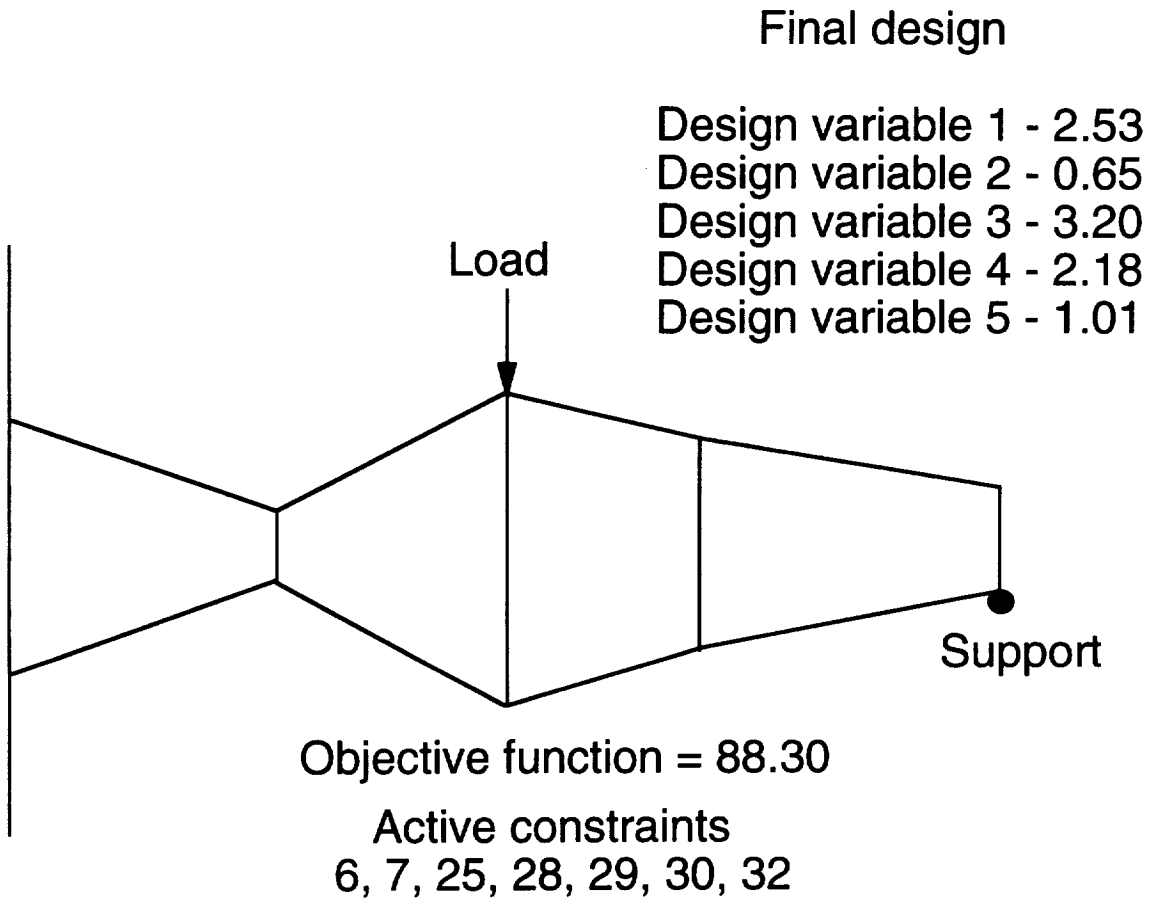


Figure 2. Final design of beam test problem (exaggerated vertical scale).

NEURAL NETWORK OPTIMIZATION PROCESS

A flowchart of the entire optimization system (in general terms) in which a neural network is used to simulate an analysis program is shown in figure 3. A training pair is a set of known input data (in this case the design variables) with the known output results (values of the constraints and the objective function). Several sets of design variables are chosen to represent the design space. To determine the output results for a given set of inputs, the structural analysis program must be executed for each set to create the training data for the neural network. Both the input and output data in the training pairs must be scaled. The neural network is trained by processing the training pairs through NETS to determine the weights that represent a relationship between the input and the output data. These weights are saved after they have been computed. New input

data can now be propagated through the neural network and used with the weights obtained from the training to approximate new output data.

The weights, along with the initial design variables, are input to NETS/PROSSS, which iterates between NETS and the optimizer. NETS simulates the structural analysis by computing the objective function and the stress constraints from the design variables and the set of weights; the optimizer then uses this data to compute a new set of design variables. The iteration continues until convergence to a near optimal design, based on a predefined convergence criterion, is realized. This revised set of design variables can serve as an initial design in PROSSS that iterates between the structural analysis program and the optimizer until a final optimal design is obtained.

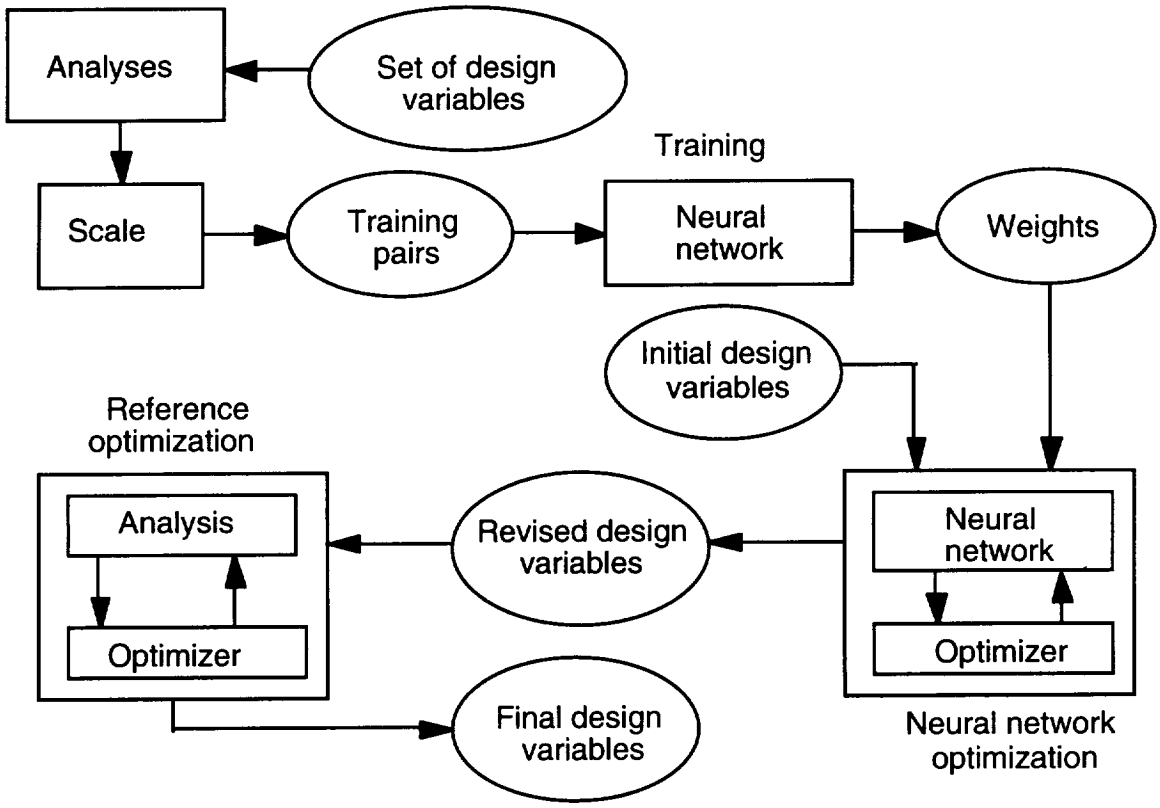


Figure 3. Optimization process with neural network simulation.

DECOMPOSING THE NEURAL NETWORK

In this paper, a large neural network is decomposed into many smaller, independent neural networks that can be trained in parallel.

Parallel training was performed on different workstations rather than a multiprocessor computer. The first step was to determine the number of nodes on the hidden layer of the smaller neural networks. Then, several methods for choosing the initial training weights were tested. Finally, a different network configuration was tested to capture the interactions among the smaller networks. Results are presented to indicate those methods that were successful and those that were not. From these results, additional guidelines are developed to avoid some of the pitfalls in training neural networks to simulate structural analysis in an optimization process.

Hidden-Layer Nodes

The original network had a 5-46-41 configuration representing 5 design variables as input, 40 stress constraints and an objective function as output, and 46 nodes on the hidden layer. This network was decomposed into 41 smaller neural networks: 1 for each of the 40 constraints and 1 for the objective function (the output nodes). Each neural network had 5 input nodes representing the design variables. The number of nodes to be placed on the hidden layer had to be determined.

The first choice was a 5-6-1 configuration. Seven networks were trained to either a root mean square (RMS) error of 0.001 or 100,000 cycles (an arbitrarily chosen cutoff point), whichever occurred first. All networks had 24 training pairs developed from the guidelines presented above. The results are shown in figure 4.

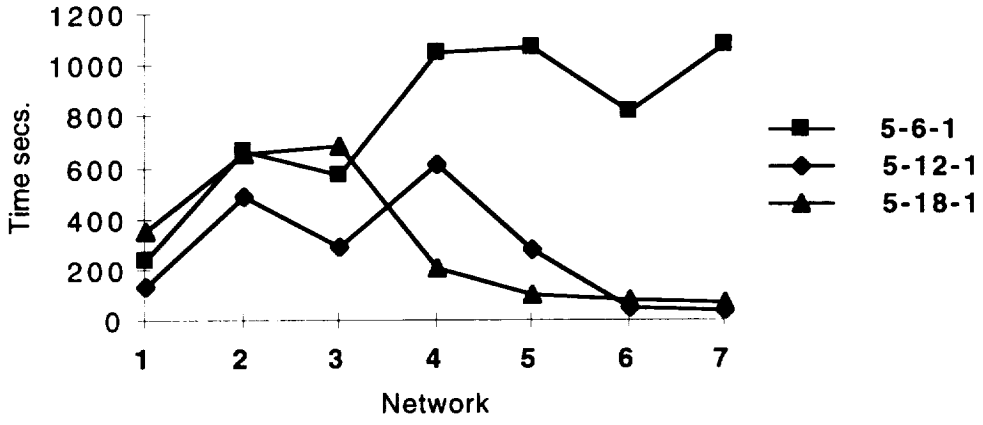


Figure 4. Training times for neural network configurations.

Networks 1, 2, and 3 used training data from the objective function, and constraints 1 and 2 respectively; random weights were used as the starting point. Constraints 1 and 2 were arbitrarily selected for training. After training, the 3 sets of weights were saved.

Next, networks 4 and 5 used training data representing constraints 1 and 2 respectively. To determine if the training time might be reduced by using consistent starting weights, these two networks were trained beginning with the weights saved from training the objective function (network 1). These trained sets of weights were also saved.

Finally, networks 6 and 7 used training data representing constraint 2. To determine if starting with weights from a constraint network might reduce training time even more, these two networks were trained beginning with the weights saved from the trainings of the constraint 1 network (networks 2 and 4). Three of the 5-6-1 networks (networks 4, 5, and 7) did not train to an RMS of 0.001 within 100,000 cycles.

Next, a 5-12-1 configuration was trained with the same procedures as before. All networks trained within 100,000 cycles (0.019 sec/cycle) and in less overall time than the 5-6-1 configuration. For comparison, the 5-46-41 configuration takes 0.368 sec/cycle for 24 training pairs.

Finally, a 5-18-1 configuration was trained; a law of diminishing returns appears to exist. Although each network was trained in less than 100,000 cycles (0.027 sec/cycle), the overall time was worse than for the 5-12-1 network.

The original relation between "knowns" and "unknowns":

(output nodes * training pairs) = (input nodes * hidden nodes) + (hidden nodes * output nodes) + hidden nodes + output nodes

can be rearranged to determine the number of hidden nodes.

hidden nodes = (output nodes * (training pairs - 1))/(input nodes + output nodes + 1).

Much of the complexity of the problem has been reduced by decomposing the large neural network into many smaller neural

networks with a single output node. Therefore the relation may be rewritten as:

hidden nodes = (training pairs - 1)/(input nodes + 2) = 3.29 for this problem.

But this relation does not hold for our current problem which requires 12 nodes on the hidden layer to train properly. A new factor is needed because of the decomposition. Thus the relation is modified to be:

hidden nodes = factor * ((training pairs - 1)/(input nodes + 2)).

By solving using data from the current network, the factor is found to be approximately (4/output nodes). This relation can be used as a good guess to determine the number of nodes on the hidden layer. More testing is required to determine the generality of this relation.

The 5-12-1 network was selected for more testing.

Training the Neural Network

Now that the training pairs have been created and the neural network configuration defined, the next question that must be resolved is how to train the network in terms of initial weights. Five different methods were tried because a good set of initial starting weights can decrease training time significantly.

(1) For the first method, the network for constraint 1 was trained using random starting weights. The weights from this network were then used as the starting point to determine the weights in the other 39 constraint networks in parallel.

(2) For the second method, the network for the objective function was trained using random starting weights. The weights from this network were then used as the starting point to determine the weights for the 40 constraint networks in parallel. This method was chosen to determine if the objective function would produce weights that model the entire model and, therefore, produce good starting points to train the constraints in parallel.

(3) For the third method, the constraint networks were trained using the weights from the previous constraint network as the starting

point. The network for constraint 1 was trained by starting with random weights, the network for constraint 2 was trained by starting with the weights saved from the network for constraint 1, and so on. This method takes advantage of the engineer's knowledge about the problem but loses the advantage of parallel training.

(4) The fourth method began training of all networks with random weights. Starting with random weights would yield different times for each training, therefore this is just a representative time for the method. This method also has the advantage of parallel training.

(5) The fifth method incorporated parallel training with the engineer's knowledge of the problem. The model was divided into 4 sections. A network for a constraint toward the middle of each section was trained by starting with random weights. The other 9 constraint networks were trained by starting with the weights of the network already trained in that section.

The time required to train each network with the 5 different methods is shown in figure 5.

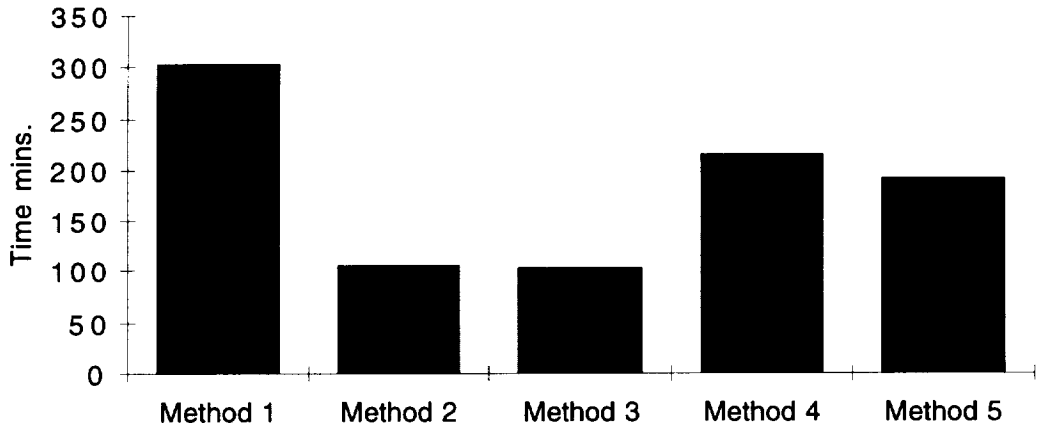


Figure 5. Training times for different methods of selecting starting weights.

When compared against the reference optimization time of 198 min., the figure shows that methods 1, 4, and 5 took too long to train: 304, 215, and 191 min. respectively. However, if they could be trained in parallel, this time could be reduced significantly. Although method 3 takes the shortest time to train (103 min.), it cannot take advantage of parallelization. Method 2 is only slightly slower (107 min.) than

method 3; however the networks can be trained in parallel which would reduce this time significantly. The choice of method 2 over method 4 (random weights) also provides consistency in training times.

Therefore, the weights from method 2 were chosen to test with NETS/PROSSS, which was modified to handle data to and from more than one neural network. The design variables were not close to those found in the reference run; and after restarting with PROSSS, it was found that the approximation was not acceptable. Although the use of the objective function weights helped to train the constraint networks faster, a portion of the problem interactions was lost in the decoupling into separate networks.

Capturing Neural Network Interactions

A different network configuration was created to determine if the interactions among the smaller neural networks could be retained. Instead of a separate network for the objective function, the objective function was added as an output for each of the 40 constraint networks. This resulted in a 5-10-2 network configuration with 40 different networks. The number of nodes (10) on the hidden layer was determined from the relation:

hidden nodes = ((4/output nodes) * output nodes * (training pairs - 1))/(input nodes + output nodes + 1).

This yields approximately the same number of unknowns as the 5-12-1 network. The following steps were used to train and test this configuration.

(a) The 13 training pairs from the hypercube built around the initial design were used to train the 40 networks. This took about 22 min.; however, this step could have been accomplished in parallel. The maximum training time for a single network was less than 2 min. The weights were saved; and NETS/PROSSS was executed with this configuration to obtain an approximate design.

(b) A new, large hypercube was built around the approximate design from step (a), and 11 additional training pairs were developed. The 40 neural networks were then retrained with 24 training pairs. The retraining, starting with the weights saved from

step (a), took 519 min. (weights were again saved). The maximum training time for a network was approximately 30 min.

(c) Training pairs were developed the same as in (b). The 40 neural networks were then retrained with 24 training pairs. The retraining, starting with random weights, took 448 min. (weights were again saved). This indicates that the "knowledge" from the previous training with 13 training pairs did not save time in training even though the 13 training pairs make up part of the 24 training pairs used here. The maximum training time for a network was about 30 min. As before, this training could be done in parallel to reduce the time.

(d) A 5-46-41 neural network was trained with these 24 training pairs to compare results. Weights were again saved.

(e) The NETS/PROSSS was executed with each of these 3 sets (from steps b, c, and d) of saved weights, and PROSSS was restarted from the approximate designs with move limits of 30 percent and run for 5 iterations. The approximate designs from NETS/PROSSS are shown in figure 6.

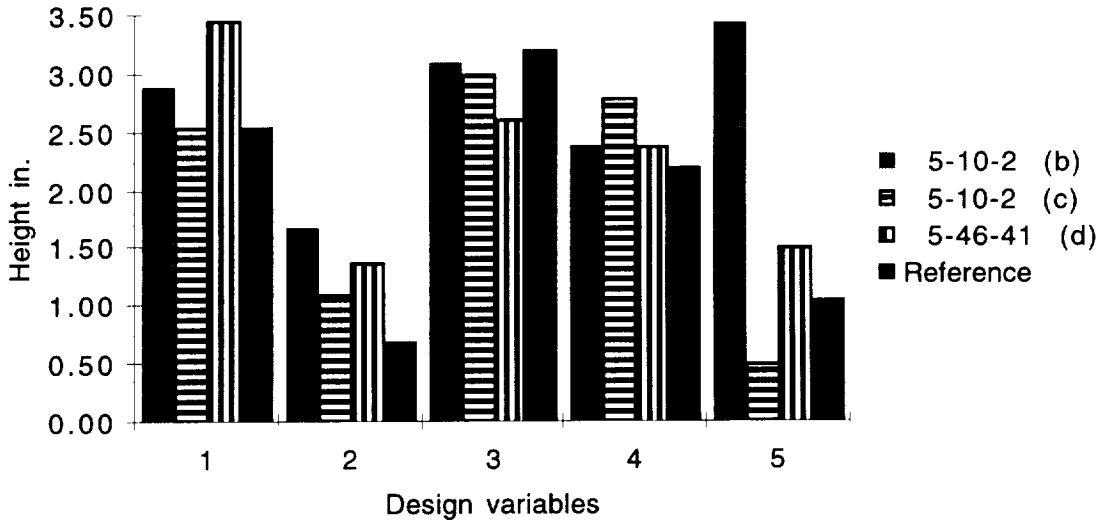


Figure 6. Comparison of design variables.

In the figure, these design variable approximations do not appear to be acceptable in comparison with the reference run. After PROSSS is restarted for 5 cycles, the final designs are shown in figure 7.

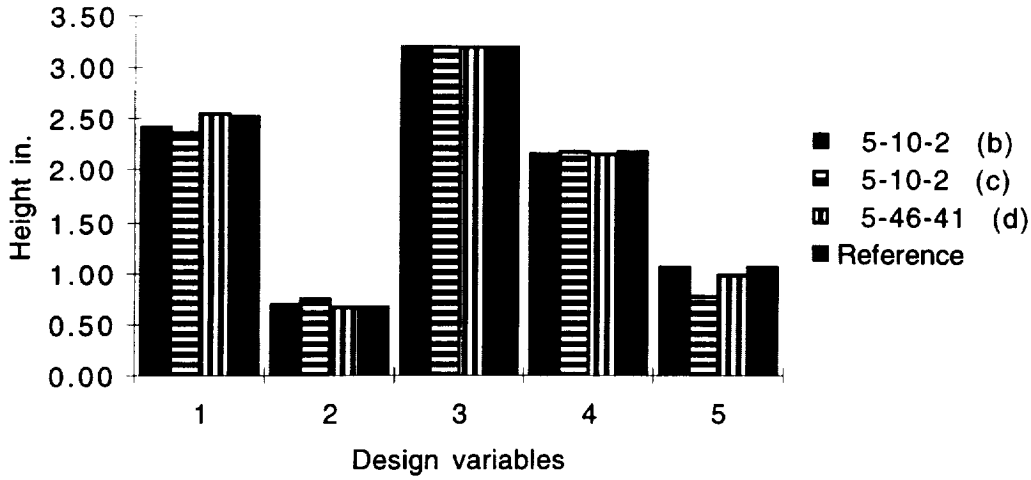


Figure 7. Comparison of design variables after restart.

Although the designs in figure 7 and their respective objective functions (88.85, 87.47, 88.41) are close to the reference (88.30), significant differences exist in the constraints. Run (c) has 16 active constraints and 7 violated constraints; run (d) has 18 active constraints and 1 violated constraint.

Run (b) is the closest approximation to the reference run. The 7 active constraints for the reference run are 6, 7, 25, 28, 29, 30, and 32; the 4 active constraints for run (b) are 6, 25, 28, and 29. Neither has any violated constraints. Again, method (b) which involves the construction of 2 hypercubes yields the best results.

Therefore, if the most efficient use is made of parallelization in both the analysis (all analyses executed at the same time on different workstations) and the training (all networks trained at the same time on different workstations), then, a reasonable optimal solution for this problem could be found in a much shorter time (see below). The times in the training steps represent the maximum time required to train a single network at each step.

Analyses for initial design	3 min.
Training in step (a)	2 min.
Analyses for approximate design	3 min.
Training steps (b) or (c)	30 min.
Total time	38 min.

These results represent a significant improvement in time when compared both with the reference time of 198 min. and the time required for optimization using neural networks without parallelization of 159 min.

CONCLUDING REMARKS

If a neural network is to be a viable simulator for structural analysis in an optimization process, then a meaningful optimal design must be obtained in less time. The time needed to create the training pairs and to train the neural network must be significantly less than the time required to execute the optimization process with analysis.

Guidelines have been developed to create a neural network that can simulate structural analysis in the optimization process and obtain a meaningful optimal design in less time. For the example selected, the hypercube method for selecting the training pairs appears to give the best approximation of the design space, particularly when it is used twice. The first time, the hypercube represents the entire design space around the initial design. The neural network is trained with this data and an approximate optimum is found with NETS/PROSSS. The second hypercube is built to represent the design space around this approximate optimum. The neural network is then trained with both sets of training pairs, and NETS/PROSSS is again executed to find a better approximation of the optimal design.

The time required to obtain a reasonable optimal design can be further reduced by decomposing a single, large neural network into many smaller neural networks that can be trained in parallel. When the smaller neural networks are configured, the number of nodes on the hidden layer must be large enough to capture the essence of the function. Based on the results from this project, it appears that a slightly underdetermined neural network can provide an adequate approximation of the results for a structural analysis program. A good initial guess for the number of nodes on the hidden layer appears to be given by:

hidden nodes = ((4/output nodes) * output nodes * (training pairs - 1))/(input nodes + output nodes + 1).

The output layer should contain a node that captures the interactions among the various smaller neural networks; otherwise, these interactions may be lost.

These guidelines, coupled with those from previous research, were used to obtain a reasonable near optimal design for the test problem. If verification is required, then the optimization process with structural analysis can be restarted from this design and should quickly converge to an optimum solution. The guidelines offered in this paper are strictly applicable to the test problem. More general applicability to other engineering design problems remains to be demonstrated.

REFERENCES

Baffes, P. T. (1989). NETS 2.0 User's Guide. LSC-23366, NASA Lyndon B. Johnson Space Center.

Barthelemy, J.-F. M., and Riley, M. F. (1988). Improved Multilevel Optimization Approach for the Design of Complex Engineering Systems, AIAA Journal, **26**, No. (3), 353-360.

Berke, L., and Hajela, P. (1991). Applications of Neural Nets in Structural Optimization, Presented at the NATO/AGARD Advanced Study Institute on "Optimization of Large Structural Systems', Berchtesgaden, Germany.

Bryson, A. E., and Ho, Y. C. (1969). Applied Optimal Control, Blaisdell.

Carpenter, W. C. and Barthelemy, J.-F. M. (1992). Comparison of Polynomial Approximations and Artificial Neural nets for response Surfaces in Engineering Optimization. Submitted to the 33rd SDM Conference in Dallas, TX.

Jones, W. P., and Hoskins, J. (1987). Back-Propagation, *BYTE Magazine*, October, 155-162.

Lippmann, R. P. (1987). An Introduction to Computing with Neural Nets, *IEEE ASSP Magazine,* April, 4-22.

Mason, R. L., Gunst, R. F., and Hess J. L. (1989). Statistical Design and Analysis of Experiments, John Wiley and Sons.

Rehak, D. R., Thewalt, C. R., and Doo, L. L. (1989). Neural Network Approaches in Structural Mechanics Computations,

Computer Utilization in Structural Engineering, Ed. J. J. Nelson, Jr., ASCE Proceedings from Structural Congress.

Rogers, J. L. Jr., Sobieszczanski-Sobieski, J., and Bhat, R. B. (1981). An Implementation of the Programming Structural Synthesis System (PROSSS). NASA TM 83180.

Rogers, J. L. Jr. (1982). Combining Analysis with Optimization at Langley Research Center - An Evolutionary Process. "*Proceedings of the Second International ASME Computers in Engineering Conference*, **3** , 83-91, San Diego, CA.

Rogers, J. L. Jr., and LaMarsh, W. J. II. (1991). User's Guide to NETS/PROSSS. NASA TM 104166.

Rogers, J. L., and LaMarsh W. J. II. (1992). Application of a Neural Network to Simulate Analysis in an Optimization Process, Proceedings of the Artificial Intelligence in Design '92 Conference, Pittsburgh, PA, pp. 739-754.

Rogers, J. L. (1994). Simulating Structural Analysis with Neural Network. ASCE Journal of Computing in Civil Engineering, Vol. 8, No. 2, April 1994, pp. 252-265.

Sobieszczanski-Sobieski, J., and Bhat, R. B. (1979) Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled By a Computer Operating System, *A Collection of Technical Papers on Structures - AIAA / ASME / ASCE / AHS 20th SDM Conference,* 20-71, AIAA Paper No. 79- 0723.

Swift, R. A., and Batill, S. M. (1991). Application of Neural Networks to Preliminary Design, *A Collection of Technical Papers on Structures - AIAA /ASME/ASCE/AHS 32nd SDM Conference,* AIAA Paper No. 91-1038.

VanLuchene, R. D., and Roufei, S. (1990) Neural Networks in Structural Engineering, *Microcomputers in Civil Engineering* **5**, 207-215.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1995	Technical Memorandum

4. TITLE AND SUBTITLE
Reducing Neural Network Training Time with Parallel Processing

5. FUNDING NUMBERS
505-63-50-12

6. AUTHOR(S)
James L. Rogers, Jr.
William J. LaMarsh II

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, Virginia 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA TM-110154

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category - 61

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Obtaining optimal solutions for engineering design problems is often expensive because the process typically requires numerous iterations involving analysis and optimization programs. Previous research has shown that a near optimum solution can be obtained in less time by simulating a slow, expensive analysis with a fast, inexpensive neural network. A new approach has been developed to further reduce this time. This approach decomposes a large neural network into many smaller neural networks that can be trained in parallel. Guidelines are developed to avoid some of the pitfalls when training smaller neural networks in parallel. These guidelines allow the engineer: to determine the number of nodes on the hidden layer of the smaller neural networks; to choose the initial training weights; and to select a network configuration that will capture the interactions among the smaller neural networks. This paper presents results describing how these guidelines are developed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Neural Network, Parallel Processing, Optimization	20
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	