Parallel Neural Network Training

Ed P. Andert Jr. and Thomas J. Bartolac
Conceptual Software Systems Inc.
P.O. Box 727, Yorba Linda, CA 92686-0727, U.S.A.
Phone: (714) 996-2935, Fax: (714) 572-1950
email: andert@orion.oac.uci.edu

Introduction

Connectionist approaches in artificial intelligence can benefit from implementations that take advantage of massive parallelism. The network topologies encountered in connectionist work often involve such massive parallelism. This paper discusses additional parallelism utilized prior to the actual network implementation. Parallelism is used to facilitate aggressive neural network training algorithms that enhance the accuracy of the resulting network. The computational expense of the training approach to training is minimized through the use of massive parallelism. The training algorithms are being implemented in a prototype for a challenging sensor processing application.

Our neural network training approach produces networks that are superior to the limited accuracy of typical connectionist applications. The approach addresses function complexity, network capacity, and training algorithm aggressiveness. Symbolic computing tools are used to develop algebraic representations for the gradient and Hessian of the least-squares cost functions for feed-forward network topologies, with fully-connected and locally-connected layers. These representations are then transformed into block-structured form. They are then integrated with a full-Newton network training algorithm and executed on vector/parallel computers.

Gradient-based Training

Most researchers train with the "back-propagation" algorithm [McClelland 86], a simplified version of gradient descent. The name comes from the concept that as a given input flows forward through the network, the resulting output is compared with the desired output, and that error is propagated backward through the network, generating the changes to the network's weights as it progresses.

One advantage of back-prop is that it only requires the calculation of the training cost function gradient. The layered structure of the network allows the gradient to be expanded by the chain rule method, resulting in each node being able to compute "its own" partial derivatives and weight changes based solely on local information, such as the error propagated to it from the previous layer.

Unfortunately, back-prop is not efficient. It is notoriously slow, especially as the solution is approached, due to the fact that the gradient vanishes at the solution. Superior methods exist, such as conjugate gradient, which still only require the cost function gradient (and a few additional global values) to be computed. These methods converge more quickly, but inevitably slow down as the solution is approached.

This slow-down is more than a matter of patience, however. It also indicates a lack of "aggressiveness". In attempting to learn a complicated mapping, an unaggressive training algorithm may cause the network to learn only the more obvious features of the mapping. In this case, the network will never achieve an arbitrarily high accuracy in approximating the mapping, no matter how complicated the network may be. This is similar to Simpson's method for numerical integration, in which decreasing the step size achieves an increase in accuracy only to a certain limit, after which further step size decreases only causes round-off error to corrupt the solution.

Hessian-based Training

More aggressive training algorithms rely on the Hessian of the cost function. This allows the network to learn the more subtle features of a complicated mapping, since the Hessian is able to represent the more subtle relationships between pairs of weights in the network. The full-Newton method directly calculates the Hessian, while quasi-Newton methods approximate the Hessian, or its inverse. These algorithms converge more quickly, especially as the solution is approached, since the Hessian does not vanish at the solution.

Far from the solution, the Hessian is not always positive-definite, and the Newton step cannot be assumed to be a descent direction. For this reason it is necessary to modify the full-Newton method by adjusting the Hessian until it is positive-definite. Different methods for doing this have been developed [Dennis 83], all with the goal of minimizing the amount by which the Hessian is perturbed. The Hessian can be perturbed by a very small amount, which varies as the training progresses, using a technique that we have developed.

In exchange for the aggressiveness and superior convergence rates, these algorithms are more computationally expensive per iteration. Furthermore, the expense grows with network size at a rate faster than that for gradient-based techniques, typically by a factor of N, for a network having N weights.

A further hindrance for full-Newton methods is the need to explicitly calculate the Hessian. Since approximating it numerically is risky for complicated cost functions, researchers must derive the analytic expressions for each of the terms in the Hessian. There are of order $2L^2$ algebraically unique terms in the Hessian for a network with L adjacently-connected hidden layers.

Finally, computing a Hessian requires global information, since each element refers to a different pair of weights. Therefore a Hessian cannot be mapped easily onto the existing topology of its network, as can be done for a gradient, and so typically it is calculated "off-line".

These disadvantages have deterred many from Hessian-based approaches, except for the simplest and smallest of networks. This, in turn, has limited the application of neural networks as solution estimators for difficult function evaluations or inverse problems.

Two technologies exist that can remove the limitations to Hessian-based learning algorithms. These are vector/parallel computer architectures and block-structured algorithms.

The technologies make Hessian-based training algorithms a more practical consideration. The increase in their computational expense with network size is reduced by the commensurate increase in efficiency with which they can be calculated on vector/parallel architectures. The overall growth, of order N^3 for a network of N weights, must still be paid, but now the researcher can choose to have N, or even N^2 of the cost, paid in computer hardware, with the remainder paid in execution time.

Performance Advantages

High-performance computers that offer vector-based architectures or parallel computing units promise faster execution for problems that are highly vectorizable or parallelizable. The evaluation of a Hessian is a good example of such a problem. Ideally, by providing an N-fold increase in the computational hardware, a problem could be solved N times faster.

In reality, however, most problems cannot be completely vectorized or parallelized, and the remaining serial portion degrades the overall performance, leading to a case of diminishing returns, as described by Amdahl's Law.

Furthermore, computer technology is such that the computing units can typically perform their operations on two operands faster than the memory and bus can supply the operands or digest the result. This has led to a change in strategy in the design of compute-bound algorithms. Now the goal is to perform many operations on a small set of data, repeating across the data set, rather than the previous method of performing a small number of operations across the entire data set, repeating for the required list of operations.

In this way the overhead cost of moving the data can be amortized over many operations, and with clever cache architectures, can be hidden entirely. This approach leads to the data set being partitioned into blocks, and hence the name "block-structured algorithms". Preliminary results indicate that block-structured algorithms more closely achieve the ideal of a linear speed-up [Gallivan 90].

Recently LAPACK was released [Anderson 92], a block-structured version of the LINPACK and EISPACK libraries. These portable routines make calls to BLAS3 (Basic Linear Algebra System), and take into account the effects of high-performance architectures, such as vector registers, parallel computing units, and cached memory structures.

The BLAS3 routines are matrix-matrix operations, which represent the largest ratio of computation to data movement for fundamental operations. They complement the earlier BLAS1 and BLAS2 (vector-vector and vector-matrix operations), and all are written in forms optimized for each specific computer architecture, often in the native assembly language.

The combination of these two technologies makes Hessian-based training algorithms a more practical consideration. The increase in their computational expense with network size can be efficiently offset by an increase in the amount of computer hardware hosting the training algorithm.

But in addition to simply deriving the Hessians, they must be cast into block-structured form, to take maximum advantage of the vector/parallel architectures they will be evaluated on. Whereas the standard approach to defining the size of a block is to match it with the vector length in the computer's vector registers or the number of units that can compute in parallel, here the layered nature of the network suggests one would choose block sizes according to the sizes of the network's layers. This will lead to each block being, in general, rectangular in shape and of different sizes (see figure 1).

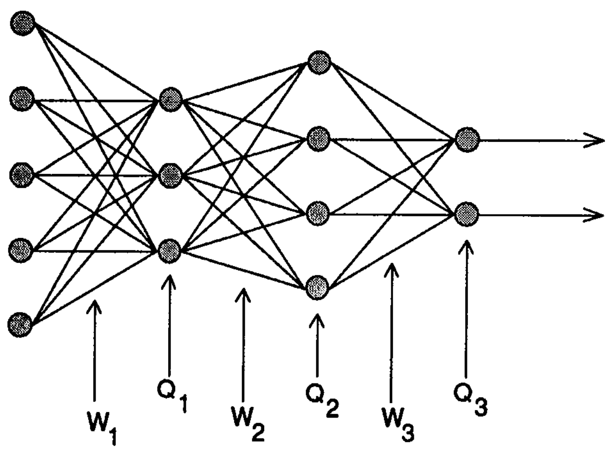

Figure 1a: *A typical 3-layer feed forward network*

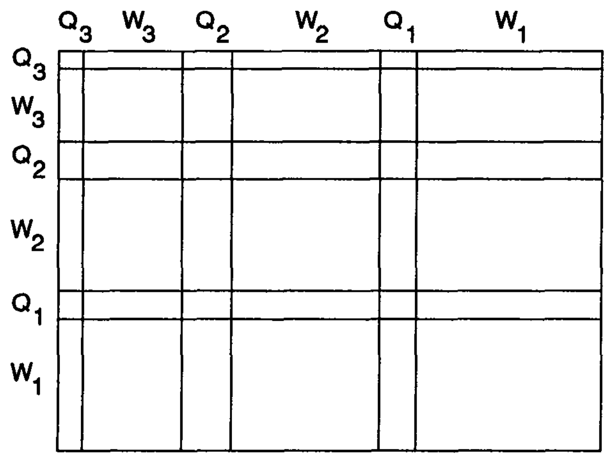

***Figure 1b**: The block structure of the Hessian for the above network showing relative block sizes.*

When these block sizes are smaller than the computer's vector length (actually pipeline depth) or number of parallel units, there will be some loss of efficiency. However, for large networks, with many nodes per layer, the situation will be reversed, and each block in the Hessian will be partitioned at execution time into a number of smaller, uniformly-sized sub-blocks, as dictated by the "size" of the computer. For improving the training for large networks, the latter case will prevail.

Note that since a gradient is a one-dimensional object, it does not lend itself to a block-structured form, and at best can be represented in terms of vector-vector and vector-matrix operations. Therefore, gradient-based training algorithms will enjoy some degree of speed-up when executed on vector or parallel architectures, but since algorithms based on BLAS1 and BLAS2 routines cannot completely hide the cost of moving data, their speed-up will still be less than that of block-structured Hessian-based algorithms.

Preliminary Network Training Performance Results for a Sensor Processing Application

The approach for aggressive network training was applied to a sensor processing application. This application was a prototype achieving encouraging preliminary results. The application involves separating closely-spaced objects for missile defense system focal-plane sensor data processing. The target ideally produces a single point of illumination, but in reality the point is blurred somewhat by the optics of the sensor. This causes the point to illuminate several detectors (blur circle). As a result, the subsequent data processing must reconstitute the original point's amplitude and location on the basis of the signals actually received from the detectors illuminated by the point's blur circle. Occasionally, two point-targets are in close proximity, and their blur circles overlap on the focal plane. For this "closely-spaced objects" (CSO) problem, the task of reconstituting the original two points' locations and intensities is much more challenging. It is difficult to develop accurate estimators based on simple linear combinations of the detector returns, due to the complexity of patterns possible with overlapping blur circles.

The CSO problem is an example of an inverse function approximation problem. It is possible to train a feed-forward neural network to map from detector return space to blur circle parameter space. Its input is the actual detector return values, and its output is the corresponding blur circle parameters.

Figure 2 shows a typical network training session. It shows training set and testing set cost function values vs. number of epochs in the training exercise. This network has 8 input units, 15 hidden units, and six output units. The training and testing sets had 378 and 7497 members, respectively. Note the sudden drop in the cost function values at certain iterations. This is an example of how the aggressive full-Newton training algorithm quickly takes advantage of encountered opportunities to drastically improve the network's level of training.

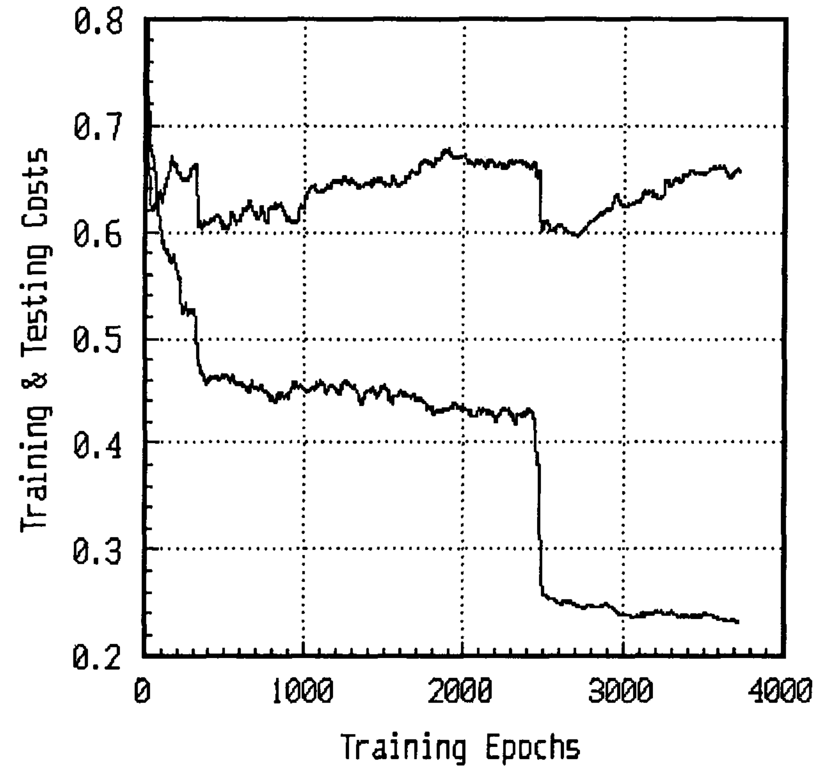

***Figure 2:** Network training session with 15 hidden units sampled at a density of 3.*

Figure 3 shows the results of training networks with different numbers of hidden units on the one-parameter version of the CSO problem. For a given network size, as the number of examples of the function are used in the training set, the training set cost function increases (the problem is getting harder to learn), and the testing set cost function values reduce (there is a more careful specification of the function), asymptotically approaching each other as the function specification becomes sufficiently dense.

For different network sizes, the cost function value at which the training and testing sets meet is different, with larger nets resulting in lower values. Furthermore, with increased network size, the function to be learned must be sampled to a higher density before the training and testing set cost function values meet. Both of these effects would be expected, since a network with a larger hidden layer has a greater capacity for learning a complicated mapping to a better accuracy.

Figure 4 shows a similar plot for networks simultaneously trained on the six CSO parameters. These networks had 5, 10, and 15 hidden units. Note the same basic shape to these curves as in the previous figure. The smallest network reaches its limit at about 0.41, for a sampling density of 4. The medium-sized network achieves about 0.37 for a density of 5, and the largest network, although not yet in its asymptotic limit, does suggest it will continue the trend in asymptotic cost function value and sampling density.

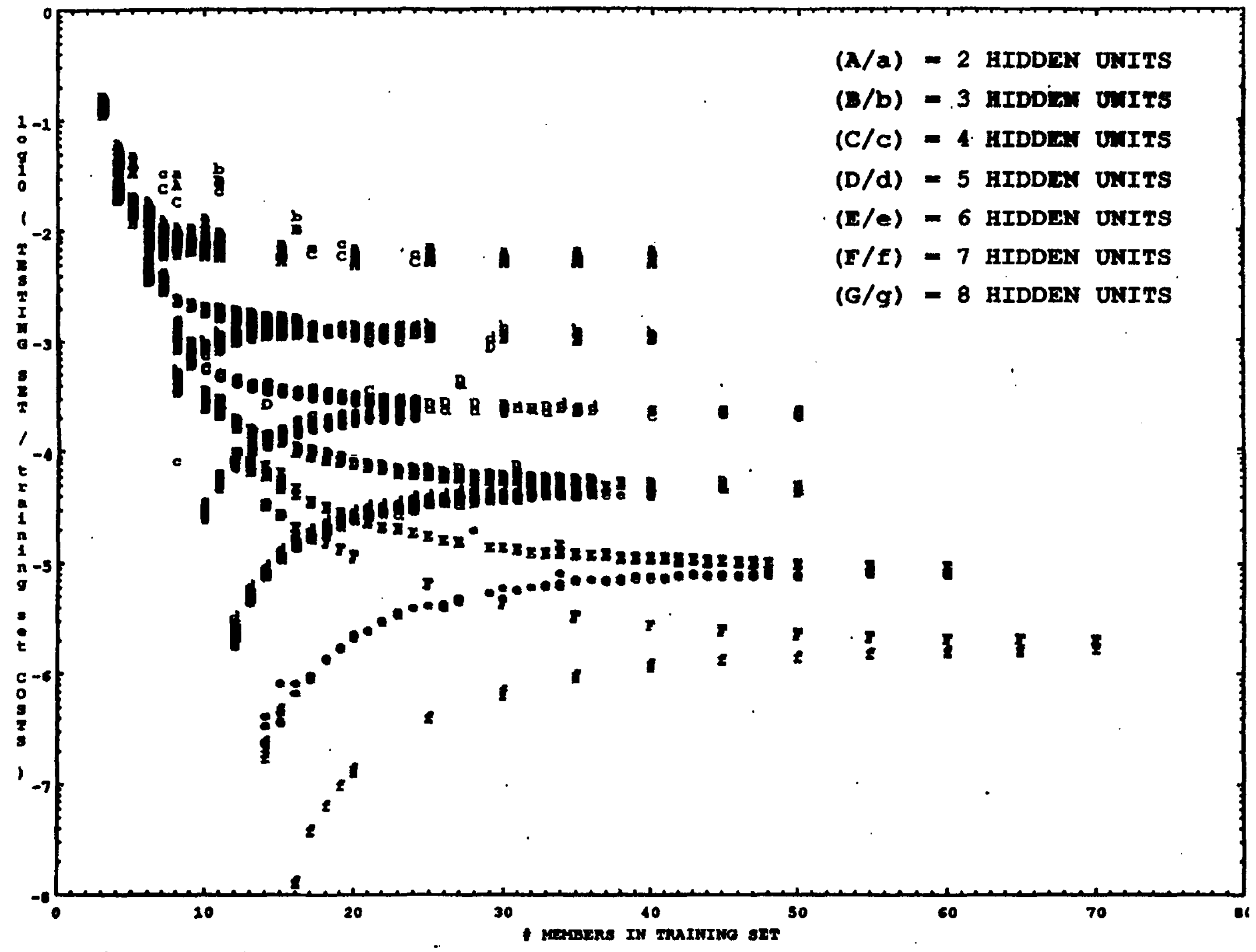

Figure 3: The results of training networks with differing numbers of hidden units on the one-parameter version of the CSO problem.

These results were accomplished through the development of a full-Newton training algorithm for the two-layer feed-forward neural network. It was written in double-precision FORTRAN, and was developed to execute on an Intel i860XR pipelined processor (11 LINPACK DP megaFLOPS), implemented as an add-in board to a 486-based PC. The two major components of the execution time taken by the training program were the calculation of the Hessian, and several matrix-manipulation algorithms. The training algorithm was able to utilize the i860 pipeline and cache by applying the blocking techniques discussed earlier when calculating the Hessian, and utilized several LAPACK routines for the matrix calculations.

The training algorithm readily lends itself to parallelization. The calculation of the Hessian, which is performed over the entire training set, can be spread across several processors, each dedicated to a portion of the training set. The auxiliary matrix calculations implemented in LAPACK routines can be parallelized at the DO-loop level, given a compiler (e.g., Alliant) that can recognize such structures. We anticipate investigating these opportunities in the near future.

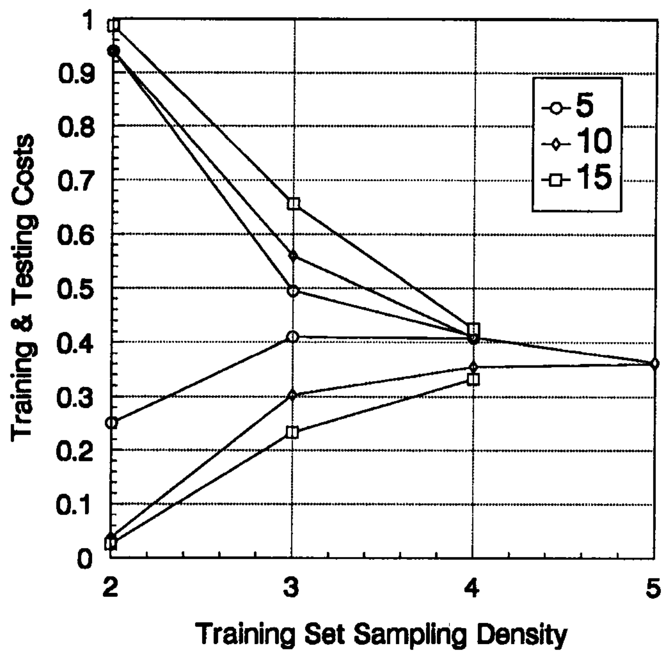

Figure 4: Network performance vs. problem size.

References

[Anderson 92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen; LAPACK User's Guide; *Society for Industrial and Applied Mathematics*, Philadelphia, 1992.

[Dennis 83] Dennis, J. Jr., and Schnabel, R. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall.

[Gallivan 90] Gallivan, K., et al. (1990). *Parallel Algorithms for Matrix Computations*. the Society for Industrial and Applied Mathematics.

[McClelland 86] McClelland, J., Rumelhart, D., and the PDP Research Group. (1986). *Parallel Distributed Processing; Explorations in the Microstructure of Cognition*. MIT Press.