Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second

Dean A. Pomerleau
George L. Gusciora
David S. Touretzky
H.T. Kung

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract:

We describe a fast back-propagation algorithm for a linear array of processors. Results of an implementation of this algorithm on Warp[1], a ten processor, programmable systolic array computer, are reviewed and compared with back-propagation implementations on other machines. Our current Warp simulator is about 8 times faster at simulating the NETtalk text-to-speech network than the fastest back-propagation simulator previously reported in the literature. This fast simulator on Warp is being used routinely in a road recognition experiment for robot navigation at Carnegie Mellon. Our results indicate that linear systolic array machines can be efficient neural network simulators. Planned extensions and improvements to our current algorithm are discussed.

1. Introduction

The application of neural networks to speech recognition [14], complex signal prediction [7], robotic control [3], image processing [9], and natural language processing [8] requires huge amounts of computing resources. Networks typically learn to perform these tasks by iterative error-minimization. Research on scaling up these learning algorithms has been hampered by the slow speeds of today's neural network simulators. One option to increase simulator power is to use a parallel machine. This paper describes a parallel implementation of the back-propagation learning algorithm [10] on the Warp systolic array computer. First we describe the architecture of the Warp and how back-propagation can be mapped onto such an architecture. Then we compare the performance of the Warp back-propagation simulator with that of other simulators. The Warp is eight times faster at learning the NETtalk text-to-speech task [11] than Blelloch and Rosenberg's implementation of NETtalk on the Connection Machine model CM-1 [2]. Finally, planned extensions and improvements to our current algorithm are discussed.

---

[1] Warp is a service mark of Carnegie Mellon University.

2. Warp Architecture

The workhorse in the Warp machine is a programmable systolic array, called "Warp array." The Warp array is composed of a linear array of processors, called "cells" (see Figure 1). Each cell has a peak computation rate of 10 million floating-point operations per second (10 MFLOPS), giving the current 10-cell machine a peak rate of 100 MFLOPS. More precisely, each cell has a 5 MFLOPS adder chip, a 5 MFLOPS multiplier chip and a 10 MIPS integer ALU. Each is equipped with 32K words of fast access data memory and can be programmed separately. Each cell is also capable of very high bandwidth communication (80 Mbytes/sec) with its left and right neighbors in the linear array. Communication between adjacent cells can be conducted in parallel over two independent channels: a left-to-right X channel and a bidirectional Y channel.

The Warp array is not the only place where parallelism exists in the Warp machine. Two cluster processors, an interface unit, and a support processor situated between the Warp array and the host act in concert to provide efficient I/O between the processor array and the large cluster memory. The interface unit is a single board processor which connects the cluster processors to the Warp array. It contains a 1024-word queue for inputting and outputting data. This ensures that a stream of input data is always available to the Warp array. The cluster processors are designed to efficiently transfer data between the interface unit and the cluster memory, which can hold up to 39 Mbytes of data without extending the cabinet. Normally, one cluster processor feeds data into the Warp array while the other transfers data from the array back to cluster memory. These functions are software switchable via the interface unit, so either cluster processor can talk to either end of the array for input or output. The support processor serves to synchronize communication between the cluster processors and the host, a Sun 3/160.

Warp programs are written in a high level Pascal-like language called W2, which is supported by an optimizing compiler written in Common Lisp. W2 programs are developed in a sophisticated, Lisp-based programming environment supporting interactive program development and debugging. A C or Lisp program can call a W2 program from any UNIX host on the local area network.

Warp was originally developed at Carnegie Mellon with support from DARPA, and it is now manufactured and sold commercially by General Electric. A complete ten-cell system costs about $350,000. For more details on systolic machines and Warp in particular, see [4,5].

3. Initial Warp Back-Propagation Algorithm: Network Partitioning

Since we started working on the neural network simulation on Warp in September 1987, we have devised several algorithms. Our first back-propagation algorithm implemented on Warp was based on network partitioning. Each cell in the 10-cell Warp array simulated one tenth of the units in every layer. Cells contained the weights for all the connections originating from the units they simulated. This partitioning scheme is illustrated in Figure 2.

During the forward pass of this back-propagation algorithm, each cell calculated the contribution its layer i units made to the net input of units in layer $i+1$. Each cell also received the partial net inputs calculated by its left neighbor for all units in layer $i+1$. Each cell then sent to its right neighbor the summed net input for all units in layer $i+1$. At the end of this sending and summing process, the tenth cell contained the total net input for all units in layer $i+1$. The tenth cell then

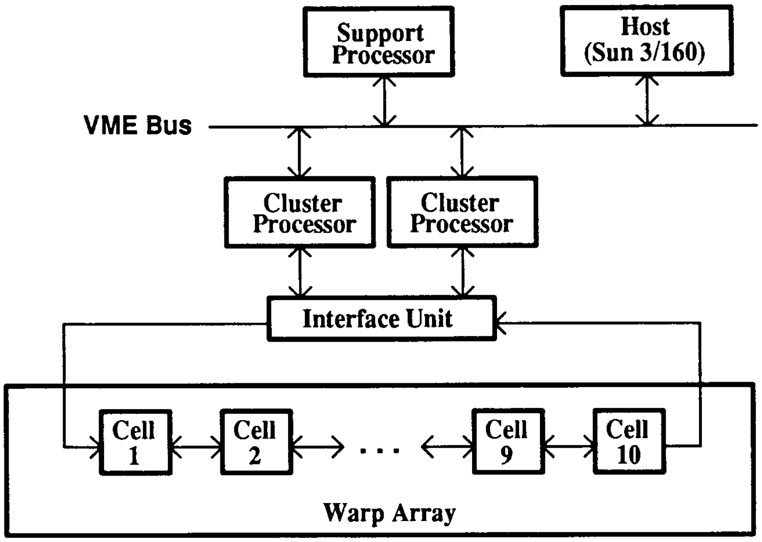

Figure 1: The Warp architecture.

sent the net inputs for each unit of the $i+1$st layer back to the cell that simulated it. Finally, each cell calculated the new activation level for its layer $i+1$ units. The process was then repeated for the next higher layer.

During the backward pass, each cell calculated the error signal for the units it simulated in level i. These error values were broadcast to all cells, so every cell contained the error value for all units in layer i. Each cell used these error values and the activation levels of the units it simulated in layer $i-1$ to calculate the weight changes to the connections originating at its $i-1$st layer units. This information was also used by each cell to calculate error values for the units in layer $i-1$. The process was then repeated to propagate error backward through the next layer.

4. Improved Warp Back-Propagation Algorithm: Data Partitioning

After the above algorithm was implemented and operational on Warp, we became more ambitious. We wanted to simulate larger and arbitrarily connected networks with faster speed. The above algorithm used the cells to store all weights. This imposed two limitations. First, the total number of weights was limited by the total size of the cell memories in the Warp array. Second, because of the additional memory access each cell needed to obtain the weights, the cell's execution speed was limited by the cell memory bandwidth. In addition, the network partition scheme used by the above algorithm made load balancing difficult for arbitrarily connected networks.

In March 1988 we started implementing an improved algorithm, which exploits the fact that weight-updates are usually small compared to the magnitude of the weight. Thus, we can run several simulations with fixed weights before updating them. Consequently several simulations can be performed in parallel on different training patterns. Whereas the initial algorithm partitioned the network into ten disjoint pieces, the improved algorithm uses a data partitioning approach where each cell simulates the entire network on a different set of training patterns. The first nine cells perform the forward and backward pass for a number of input/output pattern pairs, while the

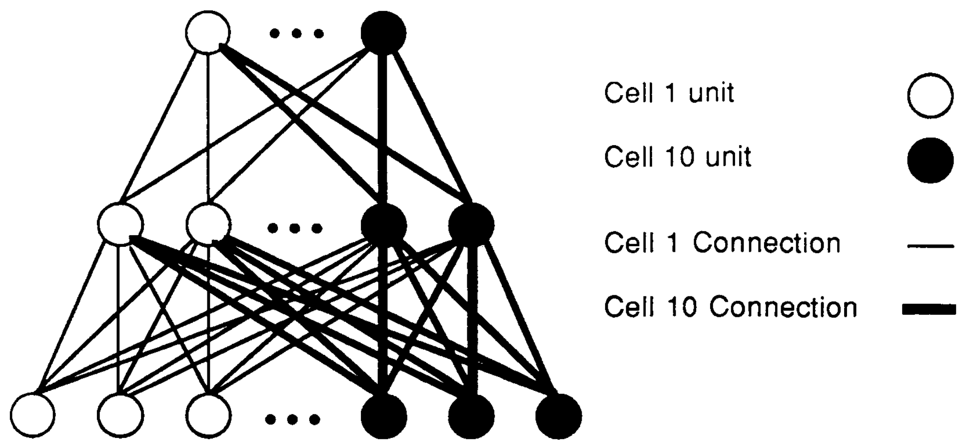

Figure 2: Data distribution for the network partitioning algorithm.

tenth cell is reserved for computing weight updates. Weights are no longer partitioned among the cells' local memories; instead they are pumped through the array from the cluster memory, which can presently hold up to 39 Mbytes. See Figure 3. Cells store only the activation levels of the units. This allows us to simulate much larger networks. Since each cell has multiple training patterns, each weight sent in to the cell is used multiple times, one for each training pattern. For these operations no memory access is needed for accessing the weight, as it can be stored in a register. Thus the local memory bandwidth no longer represents a performance bottleneck; the execution rate of each cell is substantially increased. Furthermore, this improved scheme is automatically load balanced for any arbitrarily connected network, because each cell simulates exactly the same network.

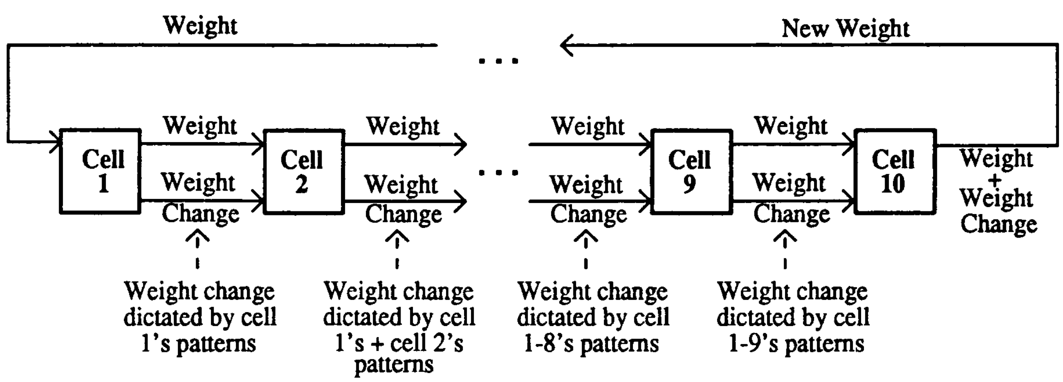

Figure 3: Information flow in the data partitioning algorithm.

In more detail, each of the first nine cells initially reads in a set of input patterns (currently ten per cell) from the host. In the forward pass, the weights from the ith layer units to $i+1$st layer units are sent in from the host one after another. As a cell receives a weight, it sends it on to the next cell and also uses it to calculate the contribution of the corresponding ith layer unit to the $i+1$st layer unit in each of the ten patterns it is simulating. After all the ith to $i+1$st layer weights are passed through the cells, each cell contains the net input to the $i+1$st layer units for the ten patterns it is simulating. Each cell performs the sigmoid calculation to determine the activation level for the $i+1$st layer units in its ten patterns. This procedure is repeated to propagate activation from each layer to the next higher.

To propagate error during the backward pass, each of the first nine cells reads in its ten output patterns from the host and then calculates the error values for the output units for each pattern. The weights between the output layer (layer i) and the last hidden layer (layer $i-1$) are again pumped through the first nine cells. Each cell uses the error values for the ith layer and the incoming weights to calculate the error values for the $i-1$st layer units in its ten patterns. Each cell also uses to the ith layer error values and the activation levels for the $i-1$st layer units to calculate the changes to the weights between these layers. These weight change values are summed across the ten patterns on each cell, then summed across the nine cells simulating patterns and finally added to the current weights by the tenth cell. The same procedure is used to propagate error backwards from the last hidden layer to subsequent lower layers.

5. Speed Comparisons with Other Machines

Speed measurements for the Warp back-propagation simulator have been performed using Sejnowski and Rosenberg's NETtalk task as a benchmark [11]. NETtalk learns to transform written English text to a phonetic representation; thus, it "learns to read aloud." The network architecture consists of three layers. The input layer has 203 units divided into 7 groups of 29; the hidden layer has 60 units, and the output layer has 26 units. During training, each input group represents a single character from among 29 possibilities (26 letters plus 3 punctuation symbols.) The 26 output units, which stand for phonetic features, form a distributed representation of a phoneme. Following Blelloch and Rosenberg [2], the 60 hidden units are fully interconnected to both the input and output units, meaning there are a total of 12,180 connections between the input and hidden layers, 1560 between the hidden and output layers, and 86 connections to the true unit for thresholds. The total number of connections is thus 13,826.

Our current implementation performs a single learning trial (one forward and one backward pass) in 0.82 milliseconds. This corresponds to a simulation rate of approximately 17 million connections per second (MCPS). This includes the computation time and all the overheads, i.e., the time required to read in the input pattern, propagate activation forward through the network, read in the correct output pattern, propagate the error signal backward through the network, compute weight changes and accumulate error values for each unit. Time spent in each of these phases is listed below.

- Forward pass 40.3% - time spent propagating activation forward through the network.
- Backward pass 50.6% - time spent propagating error backward through the network.
- Other 9.1% - time spent reading in the input and output activations, computing error statistics and actually updating weights (done concurrently on the last cell).

Table 1 illustrates the speed of the Warp back-propagation simulator in comparison with simulators for other machines. The Warp simulator is 340 times faster than Sejnowski and Rosenberg's original implementation of NETtalk on a Ridge 32, which ran at 0.05 MCPS and required 275 msec per trial [12]. The Ridge is said to be comparable in performance to a Vax-11/780 with FPA [11]. After the initial loading of the weights, a 16K processor CM-1 Connection Machine can simulate

Machine	MCPS	Relative Speed
Ridge 32	0.05	1
Convex C-1	1.8	36
16K CM-1	2.6	52
Warp	17.0	340

Table 1: Simulator speed comparison for various machines. Speed for Convex C-1 is from [6].

this network at a speed of approximately 2.6 MCPS in the steady-state [2]. Counting the initial startup I/O, the total time required by the CM-1 implementation is about 8 minutes [1]. Since the total time required by the Warp implementation is under 1 minute, Warp is eight times faster.

We must make two cautionary notes about our comparison with the Connection Machine. First, the reported speed is for a CM-1 model because there is currently no back propagation simulator for the more powerful CM-2. Also, the performance figure of 2.6 MCPS assumes a 16K processor machine instead of the 64K maximum, because more than 16K processors can't be used effectively for the NETtalk task. In the current CM back-propagation implementation, each processor simulates a weight and there are fewer than 16K weights in this network. For a 64K processor machine on a larger network, Blelloch and Rosenberg indicate their CM-1 simulator can run at 13 MCPS. A ten-cell Warp is still about 25 percent faster. The Warp architecture is trivial to extend to include more cells. A 20-cell Warp is currently being assembled, which should yield about 32 MCPS on networks as small as NETtalk, since the speedup of the Warp back-propagation algorithm is approximately linear in the number of processors provided that the network is large enough.

6. Limitations and On-going Work

The only major limitation to the current algorithm is that we always simulate fully interconnected networks. As pointed out before, our algorithm can potentially simulate general networks with any number of layers and arbitrary connectivity – with no significant performance sacrifice. The basic idea is that for each floating-point weight that is sent into the Warp array, two integers are sent in pointing to the source and destination unit for the connection. Since we are storing several input patterns in the cell's memory, we can perform many floating-point operations with each weight while the integer ALU deals with the weight indices. Thus, execution time will not be limited by I/O bandwidth. This new simulator capable of handling arbitrary connectivity is being implemented.

In addition to increasing simulator flexibility, current work with the Warp includes an exploration of neural network road following algorithms for the Autonomous Land Vehicle project. The NAVLAB, a modified Chevy van with a Warp, a video camera, and a laser rangefinder fitted inside, serves as a testbed for the project [13]. Our Warp back-propagation simulator speed makes it possible for simulating the large network necessary for this vision application in less than a day. The three-layered network currently being used for the road following task contains 1125 input units representing gray-scale image and depth values, 84 hidden units, and 126 output units representing road position information. Each layer is fully connected to the one above, so the network has a total of 105,084 connections. During training, we have been routinely cycling through 40 epochs of approximately 50,000 different simulated road images. Our current simulator runs at about 17 MCPS on this network, which corresponds to approximately 3.5 hours of Warp CPU time for training. Including time for generating the simulated roads and downloading them to the Warp host, the network requires about 5 hours to train, which is easily accomplished overnight.

7. Conclusions

Understanding the nature of an application and the architecture of the underlying machine is important in deriving an efficient mapping between the two. By exploiting the fact that weight updating need not be done frequently, we devised the data partitioning algorithm, which for the Warp architecture is much more efficient and flexible than the network partitioning algorithm. Several architectural features of Warp have contributed to its effectiveness in the data-partitioning implementation. High-bandwidth inter-cell communication allowed for the weights to be stored in cluster memory and pumped through the array when they were needed. Both a floating-point adder and multiplier enabled the computationally expensive parts in the algorithm to be done quickly and with full resource utilization. Because each cell has its separate program control, cells do not have to execute the same program at the same time; this allows for example the tenth cell to execute a different program from the other cells. Finally, an optimizing Warp compiler produces efficient code and eliminates the need to program in assembly language to achieve peak speed.

We can conclude that a one-dimensional systolic array can be very cost-effective for fast simulations of neural networks. A second generation Warp computer, called iWarp, is currently being co-developed by Carnegie Mellon and Intel. Each cell in the iWarp will be a custom VLSI chip capable of 20 MFLOPS. With a total 72 cells, iWarp will have a peak rate of 1.44 GFLOPS. Thus, upon its scheduled completion in 1990 the iWarp should provide at least another order of magnitude speedup for neural network simulations.

Acknowledgements

We are grateful to Guy Blelloch and Charles Rosenberg for their helpful correspondence. We also thank Bernd Bruegge for enhancements to the Warp Programming Environment made at our request, and Monica Lam for assistance with the W2 compiler.

This research was supported by the Office of Naval Research under Contracts N00014-87-K-0385, N00014-87-K-0533 and N00014-86-K-0678, by National Science Foundation Grant EET-8716324, and by the Defense Advanced Research Projects Agency (DOD) monitored by the Space and Naval Warfare Systems Command under Contract N00039-87-C-0251.

[1] Blelloch, G. (1987) Personal communication.

[2] Blelloch, G., Rosenberg C.R. (1987) Network learning on the Connection Machine. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy.

[3] Jorgensen, C.C. (1987) Neural network representation for sensor graphs in autonomous robot

path planning. *Proceedings of the IEEE First International Conference on Neural Networks, Vol VI*, pp. 507-515, San Diego, CA.

[4] Kung, H.T. (1982) Why Systolic Architectures? *IEEE Computer* **15**(1) pp. 37-42.

[5] Annaratone, M., Arnould, E., Gross, T., Kung, H.T., Lam, M., Menzilcioglu, O., Webb, J.A. (1987) The Warp Computer: Architecture, Implementation, and Performance. IEEE Transactions on Computers, December 1987, pp. 1523 - 1538.

[6] Lang, K. Personal communication, 1988. Carnegie Mellon University.

[7] Lapedes, A., and Farber, R. (1987) Nonlinear signal processing using neural networks: prediction and signal processing. To appear.

[8] McClelland, J.L. and Kawamoto, A.H. (1986) Mechanisms of sentence processing: assigning roles to constituents of sentences. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. II: Psychological and Biological Models* pp. 272-325, Bradford Books/MIT Press, Cambridge, MA.

[9] Mesrobian, E. and Skrzypek, J. (1987) Discrimination of natural textures: A neural network architecture. *Proceedings of the IEEE First International Conference on Neural Networks, Vol. VI*, pp. 247-258, San Diego, CA.

[10] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations* pp. 318-362, Bradford Books/MIT Press, Cambridge, MA.

[11] Sejnowski, T. J., and Rosenberg, C. R. (1987) Parallel networks that learn to pronounce English text. *Complex Systems* **1**(1):145-168.

[12] Sejnowski, T. J. (1988) Personal communication.

[13] Thorpe, C., Herbert, M., Kanade, T., Shafer S. et. al. (1987) Vision and navigation for the Carnegie-Mellon Navlab *Annual Reviews of Computer Science Vol. II,* pp. 521-556.

[14] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1988) Phoneme recognition: neural networks vs. hidden Markov models. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing,* New York, NY.