A revised version of this appears in: Journal of Aerospace Computing, Information, and Communication, Vol. 5, No. 1, Jan. 2008.

AIAA Paper No. 2005-7168

Scalable Massively Parallel Artificial Neural Networks

Lyle N. Long [*] and Ankur Gupta [†]
The Pennsylvania State University, University Park, PA 16802

There is renewed interest in computational intelligence, due to advances in algorithms, neuroscience, and computer hardware. In addition there is enormous interest in autonomous vehicles (air, ground, and sea) and robotics, which need significant onboard intelligence. Work in this area could not only lead to better understanding of the human brain but also very useful engineering applications. The functioning of the human brain is not well understood, but enormous progress has been made in understanding it and, in particular, the neocortex. There are many reasons to develop models of the brain. Artificial Neural Networks (ANN), one type of model, can be very effective for pattern recognition, function approximation, scientific classification, control, and the analysis of time series data. ANNs often use the back-propagation algorithm for training, and can require large training times especially for large networks, but there are many other types of ANNs. Once the network is trained for a particular problem, however, it can produce results in a very short time. Parallelization of ANNs could drastically reduce the training time. An object-oriented, massively-parallel ANN (Artificial Neural Network) software package SPANN (Scalable Parallel Artificial Neural Network) has been developed and is described here. MPI was used to parallelize the C++ code. Only the neurons on the edges of the domains were involved in communication, in order to reduce the communication costs and maintain scalability. The back-propagation algorithm was used to train the network. In preliminary tests, the software was used to identify character sets. The code correctly identified all the characters when adequate training was used in the network. The code was run on up to 500 Intel Itanium processors with 25,000 neurons and more than 2 billion neuron weights. Various comparisons in training time, forward propagation time, and error reduction were also made.

Nomenclature

x_i	=	Input to the i^{th} Neuron
y_i	=	Activity of the i^{th} Neuron
E	=	Net Error of the Artificial Neural Network
d_i	=	Desired output of the i^{th} Neuron
w_{ij}	=	Weight for the i^{th} input of the j^{th} neuron
w_{ij}'	=	Previous weight for the i^{th} input of the j^{th} neuron
w_{ij}''	=	Next to previous weight for the i^{th} input of the j^{th} neuron
η	=	Learning rate
α	=	Momentum Factor
e_j	=	Error term for the j^{th} Neuron

I. Introduction

Artificial Neural Networks (ANN) [1-4] have been used for many complex tasks such as stock prediction and nonlinear function approximations. The neural network methodology is free from the algorithmic complexity that is usually associated with these tasks. The same methodology can be used in a variety of applications. ANNs are designed to loosely mimic the human brain and consist of large networks of artificial neurons. These neurons have two or more input ports and one output port. Generally each input port is assigned a weight, and also the change of weight (delta-weight), to speed up the convergence. The output of a neuron is the weighted sum of the

[*] Prof., Aerospace Engineering and Director, Inst. for Computational Science, lnl@psu.edu, Assoc. Fellow AIAA
[†] Graduate Research Assistant, Mechanical Engineering, azg139@psu.edu

inputs. A transfer function is generally applied to the output depending on the desired behavior of the ANN. For example the sigmoid function is generally used when the output varies continuously but not linearly with input. The "learning" in ANNs occurs through iteratively modifying the input weights of each neuron, and typically uses the back-propagation algorithm [5-8]. Training massively parallel neural networks that run on large parallel computers can be quite time consuming since they do not scale well.

Clearly ANNs are crude approximations to human neural systems, which are briefly discussed later in this paper. In an ANN, however, the mathematical operations required for each neuron are very simple, and we can therefore use millions or billions of them in computer programs. The human brain, however, has approximately 100 billion neurons [25] and each one has roughly 1,000 synapses. If we assume each synapse requires one byte, this amounts to roughly 10^{14} bytes of data (100 terabytes). The human brain is also capable of roughly 10^{14} to 10^{16} instructions per second [25, 26]. This is far from being possible to simulate using traditional serial computers, but is not far from current massively parallel computers. For comparison purposes, a lizard and a mouse have processing power of roughly 10^{9} and 10^{11} instructions per second, respectively [26]. Even though the individual human neurons "compute" relatively slowly compared to a digital computer, the brain is a massively parallel device with enormous memory capacity.

There are many different types of ANNs (spiking [27], multi-layer [29], recurrent [29], ART [26], etc.), some are more biologically plausible than others. ANNs are also important in attempts to model the brain, but they are also important for practical engineering applications. In modeling the human brain we should try to use biologically plausible networks [11-16] (e.g. spiking networks or ART), but for engineering applications we simply want efficient software and algorithms.

Figure 1 shows the present state of technology in terms of memory and speed [26], along with data on some living creatures. Currently, one of the largest parallel computers is the NASA SGI Columbia machine [28], with 10,240 Itanium 2 processors (1.5 GHz) which has sustained $4*10^{13}$ floating point operations per second and has $2*10^{13}$ bytes of memory. This places it in a position as shown in the figure. IBM has developed two BlueGene computers that are even larger and faster than Columbia [24]. Current massively parallel computers do offer the hope of delivering significant "intelligence." In addition, Moore's law states that computer power doubles roughly every 18 months, so computers could be 1,000 times larger and faster in 15 years, which means they might approach or exceed the capacity of the human brain fairly soon. Even though computers are approaching the size and speed of mammal brains, there is the additional problem of machine learning. It would not be trivial to teach these machines to know what a human knows (but once this is accomplished it would be easy to transfer this knowledge to other computers!). Learning in humans is not necessarily fast either, since it typically takes about 18 (or more) years to train a human to function in modern society.

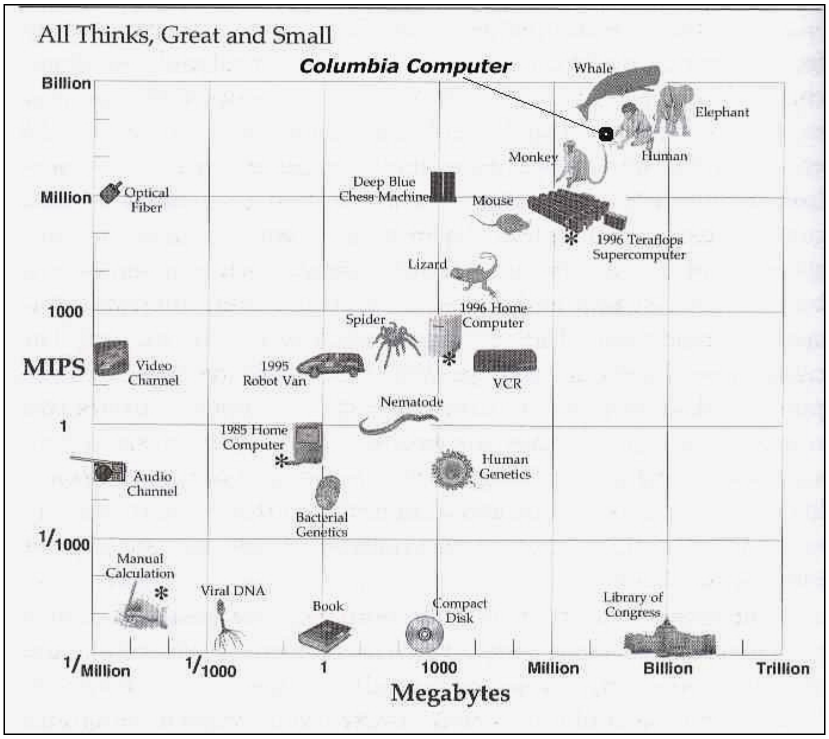


Figure 1. Present status of technology (from Moravec [26]).

The work described here is *not* an attempt to accurately model the brain, it is simply an attempt to implement ANNs on massively parallel computers.

This may be useful for both a first step towards software models of the human brain and also for developing useful engineering tools (e.g. for pattern recognition). The connections and flow of information in the brain are much more complex than a simple feed-forward network.

ANN's require a large amount of memory and computational time to train. Figure 2 shows the convergence of a serial ANN having 1000 neurons (using one hidden layer) for a period of 2,000 iterations. It uses the back-propagation algorithm [5, 6, 30] to adjust the weights. The time required to train this network (per iteration) was about 0.05 seconds using a 1.5 GHz Pentium 4 processor. The memory required for this network was roughly 1 MB (each weight was stored as a 4-byte floating point number). For comparison purposes, the time taken per iteration for a network having 30,000 neurons was about 50 seconds. Also, the amount of memory required by such a network was about 0.9 GB. Thus, serial ANNs have severe limitations in terms of memory and training time required. And it is well known that they do not scale well.

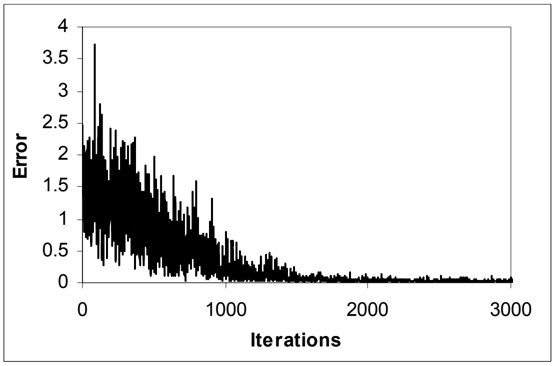


Figure 2. Absolute Error against number of training iterations.

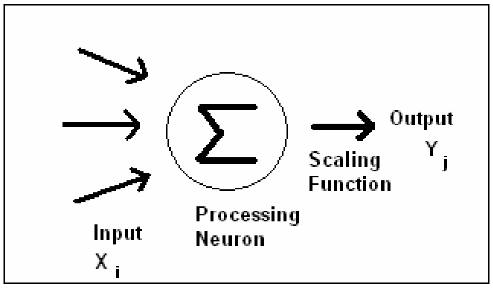


Figure 3. A processing unit or Neuron.

II. Back-propagation Algorithm for Neural Networks

This section presents briefly the back-propagation algorithm for multi-layer neural networks. Figure 3 shows how a neuron calculates its activity using the inputs. A neuron in the output layer computes its activity using equations (1) and (2). Here, f(x) is the scaling function which is generally the sigmoid function. A unipolar sigmoid function varies from 0 to 1, and is calculated by Eq. (3). A bipolar sigmoid function varies from -1 to 1, and is given by Eq. (4). The gain here is 1.0. After calculating the activity of the neurons, the network computes its error, given by Equation (5).

$$x_j = \sum y_i W_{ij} \qquad (1)$$

$$y_j = f(x_j) \qquad (2)$$

$$f(x) = 1/(1+e^{-x}) \qquad (3)$$

$$f(x) = \frac{2}{1+e^{-x_j}} - 1 \qquad (4)$$

$$E = 0.5 \sum (y_i - d_i)^2 \qquad (5)$$

For these simple artificial neurons, the inputs and outputs sort of represent dendrites and axons, respectively. The complicated chemical processes that occur in the synaptic spaces are crudely approximated by the weights. The learning or training here amounts to setting the weights using back-propagation. Back-propagation is not a very biologically-plausible method for learning, but it is a useful technique.

Back-propagation starts at the output layer with Equations (6) and (7). The learning rate, η, applies a scaling factor to the adjustment to the old weight. If the factor is set to a large value, then the neural network may learn quickly, provided that the variation in the training set is not large. Usually, it is better to set the factor to a small value initially and later increase it if the learning seems slow. We typically use a value of η=0.1.

The momentum factor α basically allows a change to the weights to persist for a number of adjustment cycles. This can improve the learning in some situations, by helping to smooth out unusual conditions in the training set. We typically use a value of α =0.5.

$$w_{ij} = w_{ij}' + (1-\alpha)\eta e_j x_i + \alpha(w_{ij}' - w_{ij}'') \tag{6}$$

$$e_j = y_j(1-y_j)(d_j - y_j) \tag{7}$$

Once the error terms are computed and weights are adjusted for the output layer, the values are recorded and the previous layer is adjusted. The same weight adjustment process, determined by Equation (6), is followed, but the error term is computed by Equation (8).

$$e_j = y_j(1-y_j)\sum e_k w_{jk}' \tag{8}$$

In contrast to Eq. (7), in this equation, the difference between the desired output and the actual output is replaced by the sum of the error terms for each neuron, *k*, in the layer immediately succeeding the layer being processed times the respective pre-adjustment weights. There is no exact criteria for how many weights are required for a neural network, but there are some rules of thumb. For example [29], for a network with N_i input units, a lower bound for the number of hidden units would be $Log_2(N_i)$, other sources [30] have suggested using the average of the number of inputs and number of outputs.

It is well known that feed-forward networks with back-propagation do not scale well. A single feed-forward and back-propagation operation requires O(N) operations [29], where N is the number of adjustable weights. There is no exact formula for the amount of training required for a neural network, it depends on the problem and the network architecture. Barron [31] shows that the error in a feed-forward ANN can be O(1/N). There is also a rule of thumb [29] that states that the size of the training set, T, should satisfy the following relation:

$$T = O(N / \varepsilon) \tag{9}$$

Where ε denotes the fraction of classification errors permitted. This means that for a case where 10% error rate is allowed, the number of training examples should be 10 times the number of weights. In the tests performed so far, the networks here appear to require far fewer than the above might indicate. It is important to point out though that the feed-forward operation is O(N), so if we can use large parallel computers to train neural networks then we can copy the weights to a serial computer and put the network to use.

III. Software

We have developed ANN software, called SPANN (Scalable Parallel Artificial Neural Network), which runs on massively parallel computers and uses the back-propagation training algorithm. An object oriented (C++) approach is used to model the neural network. The software implements several C++ objects, such as Neuron, Layer, and Network. The software can have any number of layers and each layer can have any number of neurons. Each neuron stores information such as weights, delta-weight for momentum, error and its output. Figure 4 shows a UML type diagram of the data and methods associated with each of the classes (objects). As one would expect, the Neuron object has output, error, and an array of weights. The Layer objects have arrays of Neurons, and the Network objects have arrays of layers. The object oriented approach [32, 33] allows one to develop maintainable code. The Message Passing Interface (MPI) library [34, 35] is used for parallelization. Neurons in each layer are distributed roughly equally to all the processors, however all the inputs are fed to each processor. Each layer in a processor has two ghost neurons at the boundaries. Error, Outputs, or Weights of the boundary neurons are communicated and are stored in the ghost Neurons in the neighboring processors, whenever required. The main MPI routines used are:

- MPI_Bcast to broadcast the input variables to other processors
- MPI_Allreduce to get the errors at all processors when required
- MPI_Gather, MPI_Gatherv to get the final combined output onto the master processor
- MPI_SendRecv to send and receive data in ghost nodes
- MPI_Barrier to synchronize processors

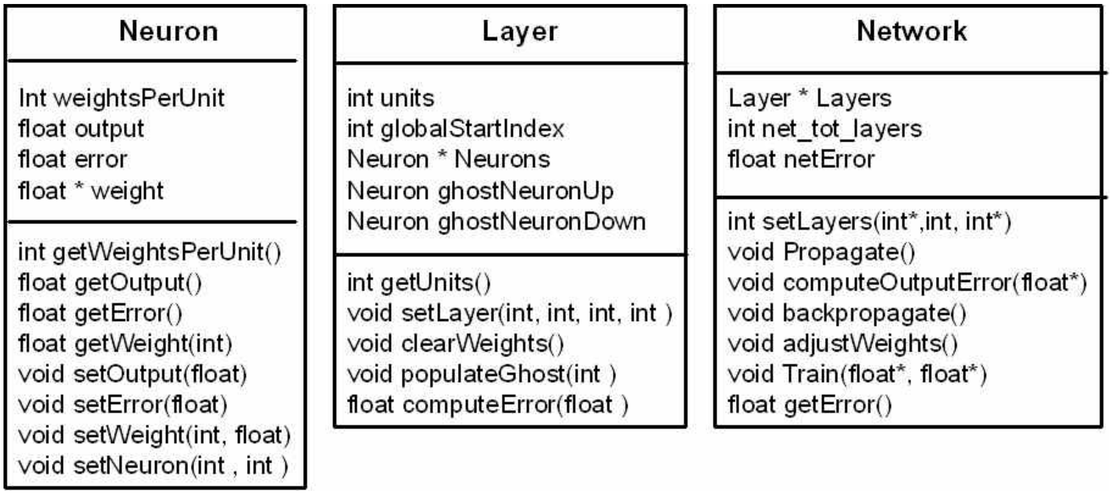


Figure 4. A UML type diagram of the important classes, data, and methods used in the code.

An important issue in the parallel implementation of ANNs is the communication load due to the nature of the feed-forward and back-propagation algorithms traditionally used. Conventional feed-forward algorithms have every neuron connected to all the neurons in the previous layer. This approach results in huge communication costs which would affect the scalability, and it is not biologically plausible either. In our approach, only values at the boundaries of the processor domains are communicated thus reducing the communication cost significantly. Figure 5 shows the communication involved in a layer for a 3-processor (P0, P1, and P2) simulation, with 12 inputs, 2 hidden layers with 9 hidden units each and 6 outputs. It also shows an example of how the ghost neurons are used at the edges of the processor domains. The figure shows all the inputs connected to all the neurons in the first hidden layer of processor P0, similar connections feed into all the other processors. Each of these connections stores the value of the weight and delta-weight for momentum. This approach also loosely mimics the structure of the connectivity in the human brain. The neurons are distributed among the processors such that each processor has about the same load. The network is trained using the back-propagation algorithm.

In order to better understand the parallel ANN, it might be useful to discuss how the number of weights varies with number of neurons. First, for a traditional ANN with 1 hidden layer, the total number of weights in the network can be computed from:

$$Total\ Weights_{serial} = (\ N_{in} +\ N_{out}\ )\ N_{hidden}$$

where N_{in}, N_{out}, and N_{hidden} are the number of neurons in the input, output, and hidden layer, respectively. The CPU time for one forward/backward propagation (for the serial case) varies linearly with the total number of weights. For the parallel ANN used here, the total number of weights varies according to:

$$Total\ Weights_{parallel} = N_{in}\, N_{hidden} + \left((L-3)\, N_{hidden} + N_{out}\right)\frac{N_{hidden}}{N_{proc}} + 4\left(L-1\right)N_{proc}$$

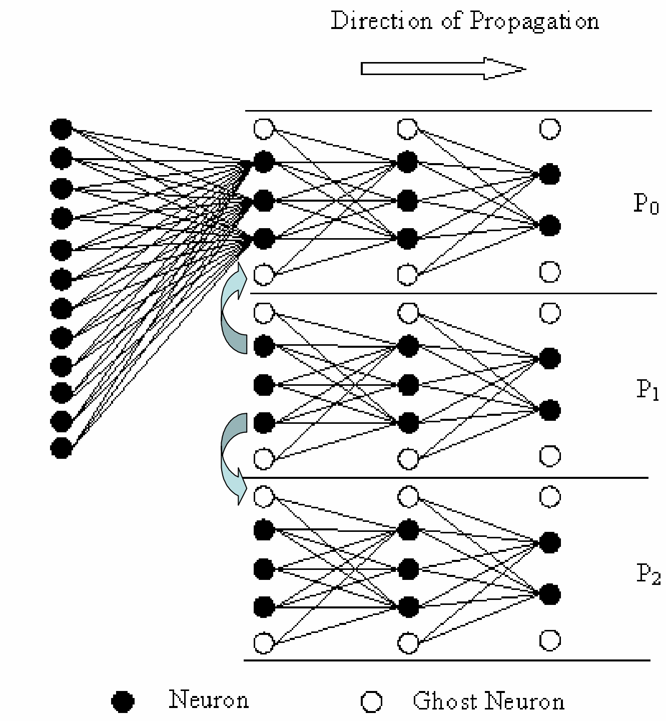


Figure 5. Neuron connections and communication in a two processor system.

where L is the number of layers (including input and output) and N_{proc} is the number of processors being use. All of the inputs (N_{in}) are fed into every processor, but each processor has N_{hidden}/N_{proc} hidden neurons/layer and N_{out}/N_{proc} output neurons. The number of weights per processor is given by:

$$Weights / \Pr ocessor_{parallel} = \frac{N_{in}\, N_{hidden}}{N_{proc}} + \left((L-3)\, N_{hidden} + N_{out}\right)\frac{N_{hidden}}{N_{proc}^{2}} + 4\left(L-1\right)$$

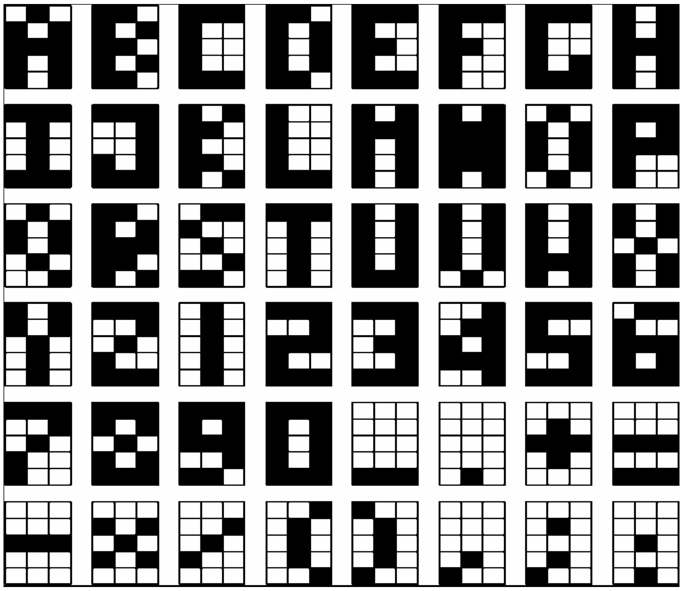


Figure 6. Character set used for testing.

For example, for the serial ANN, if N_{in} = 200, N_{out} = 48, and N_{hidden} = 125 then the total number of weights is 31,000. As an example for the parallel case, if N_{in} = 200, N_{out} = 48, N_{hidden} = 120, L = 6, and N_{proc} = 8 then the total number of weights would be 30,280 and each processor would have 3785 weights. This shows how the two different networks could be configured to have roughly the same number of total weights. The CPU time on the parallel computer will vary linearly with the weights/processor, plus there will be inter-processor communication costs.

In order to test the software, a character set, shown in Figure 6, consisting of 48 characters was used [36]. Each character is represented by an array of 3x5 pixels. Each pixel has a value of 1 or -1 according to whether it is on or off. Thus, input for each character is represented by a vector of 15 real numbers. The output for each character is a vector of length 48, with all but one value being zeros. The size of the input could be increased by increasing the resolution of the characters. Thus, for a resolution of R, each pixel in the original character set was replaced by R X R identical pixels. For the serial case, the output remained the same for any resolution. As the parallel code required at least two neurons per processor in a single layer, the output was padded with zeros whenever required.

IV. Results

Several serial runs were made for comparison with the parallel code. The serial runs were performed on an IBM P640 RS6000 Server [37]. Table 1 shows the training time required for the serial case as the resolution of the characters is increased. Most of the characters were recognized correctly. Figure 7 shows the percentage of the characters correctly recognized compared to the training iterations for different values of total weights in the network. It is observed that there seems to be an optimal value of weights for the network, which gives the most correct character recognitions for lesser number of iterations. This is due to the fact that the network is unable to "learn" due to under fitting when there are less than the required numbers of weights. Also, when the number of weights is much larger than the required, the network tends to remember rather than "learn", due to over fitting. It should be noted that what seems to be true for this problem might not hold for other neural nets or other applications, and generalization can only be made after proper verification. The parallel code has been tested using roughly the same number of weights as in the serial case. These results were obtained from a 160-processor Beowulf computing cluster [38]. Figure 8 shows the percentage of the characters correctly recognized against the training iterations for different values of total weights in the network.

Table 1. Training time required for the serial ANN

Resolution	Total Weights	Iterations	Training time (sec)
1	1890	37824	5.4
2	3240	64800	13.1
4	8640	172800	76
5	12690	253776	157
8	30240	604800	842
9	37890	757776	1313

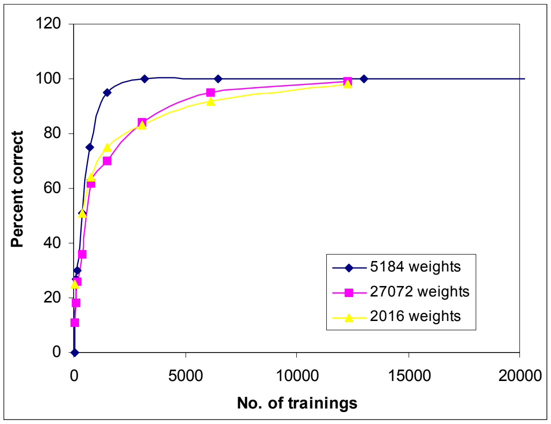


Figure 7. Percentage of correct characters recognized against number of training iterations for different weights in a serial ANN.

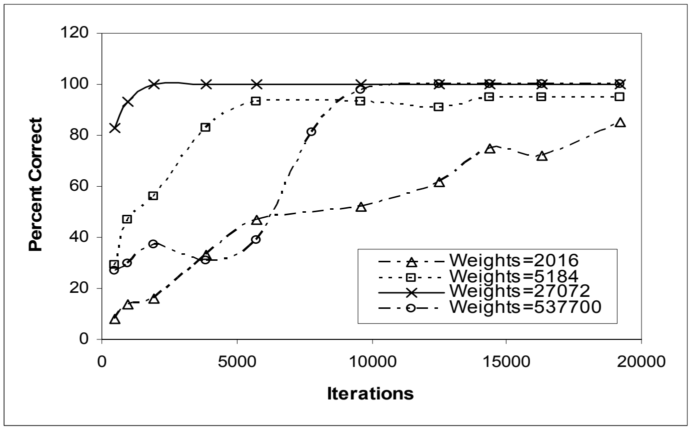


Figure 8. Percentage of correct characters recognized against number of training iterations for different weights in parallel ANN. Number of processors was 4.

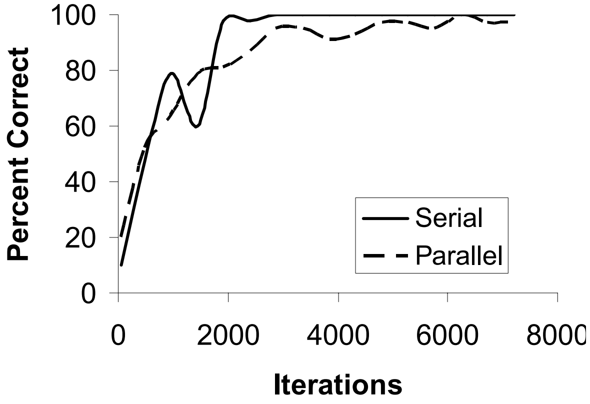


Figure 9. Convergence of serial and parallel codes.

More results, comparing the convergence of the serial and parallel codes, are shown in Figure 9. For this case, both networks had roughly 12,000 weights and the number of inputs was 135. The serial case used a 135-70-48 network, with a learning rate of 0.1 and momentum factor of 0.5. The parallel case used 8 processors and a 135-76-76-76-48 network, with a learning rate of 0.7 and momentum factor of 0.7. The convergence of the two networks is similar, even though the network structures are quite different.

Figures 10 and 11 show the performance results of SPANN on a large parallel computer. In order to maintain parallel efficiency, the number of neurons per layer was scaled linearly with the number of processors. The runs were performed on the NASA SGI Columbia computer [28]. The resolution of the characters was 1.0 for these runs. Figure 10 shows that the training time is essentially constant when the number of neurons is scaled linearly with the number of processors. All of these runs used the same number of training intervals. In practice, the larger networks may require more training for the same level of accuracy. Figure 11 shows that the time taken for a single forward propagation step also remains essentially constant as the number of neurons are scaled linearly with processors. For 64000 neurons per layer (using 6 layers) on 500 processors, the total memory required was about 0.2GB/processor (each weight was stored as a 4-byte floating point number). The memory required by a single neuron for this case was about 1KB. The largest case (on 500 processors) used more than 24 million neuron weights. This was run for 290,000 training sets, but it would need to run longer to converge.

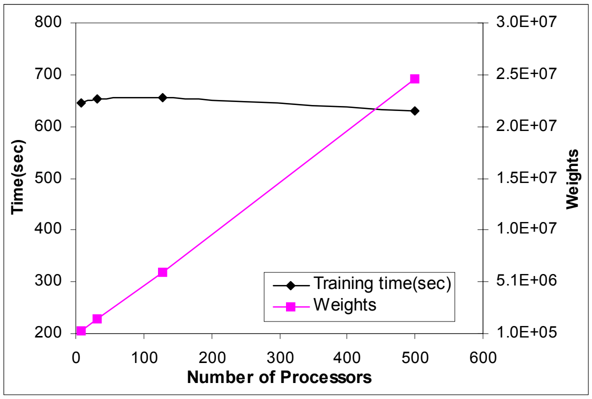


Figure 10. Training time taken as the problem size was increased linearly with the number of processors (290,000 training set s).

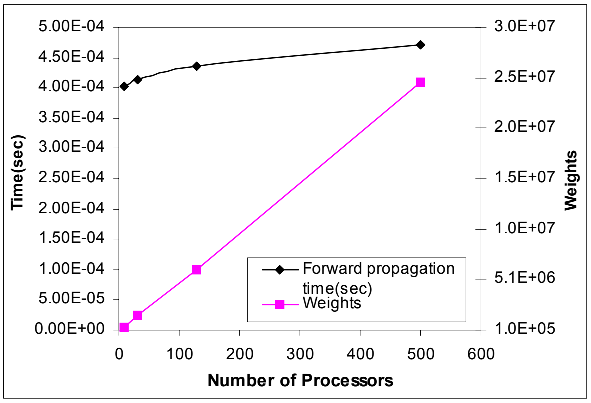


Figure 11. Time taken for one forward propagation as the problem size increases linearly with the number of processors. (290,000 training sets)

In order to show the scalability of the code in another manner, Table 2 shows results for cases where the number of inputs was scaled with the number of processors. All of these cases used six hidden layers, and were run on NASA's Columbia computer (1.5 GHz Itanium2 processors) using the Intel C++ compiler (vers. 7.1) with the options: -ftz -ipo -IPF_fltacc -IPF_fma –O3. Note that Columbia has 1.9GB memory/processor, so these cases were using a small fraction of the total memory. A single feed-forward operation on the largest case required only 0.25 seconds.

.

Table 2. SPANN results for scaling number of inputs of neural network with number of processors.

Processors	Inputs	Neurons	Neurons Per Hidden Layer	Weights	Percent Correct	Memory Used (GB)	CPU Time (sec.)
16	37,500	1584	256	9,613,376	100 %	0.08	246
64	150,000	6272	1024	153,652,480	100 %	1.20	2489
500	600,000	25,000	4000	2,400,106,384	89 %	19.0	6238

Biology of Human Brain

Since ANNs are designed to loosely mimic the human brain, it is useful to have some understanding of the structure of the human brain. This section gives a brief overview of the biology of the human brain, just for completeness.

It is believed that the neocortex -- the wrinkled outer layer of the brain -- is primarily responsible for intelligence. It is believed that the neocortex has a columnar structure and has six layers [13]. Figure 12 shows the columnar structure of the neocortex [13]. Figure 13 shows a possible model of this columnar structure [13]. In the approach used here the ANN has a columnar structure also, and we typically use six layers. LeDoux [9], describes the neocortex as the area where the essential aspects of intelligence (e.g. perception, language, imagination, mathematics, art, music, and planning) occur. References [10-16] also discuss the architecture of the human brain. The neocortex is the most recently evolved part of the brain, and is often believed to be responsible for the evolution of intelligence and language. They exist primarily in mammals, but appear somewhat in birds and reptiles. The human neocortical sheet consists of six layers and a large number of tightly packed nerve cells, or neurons. It is estimated that a typical human neocortex contains about 100 billion neurons (worms, ants, and rats have roughly 300, 50 thousand, 50 million, respectively).

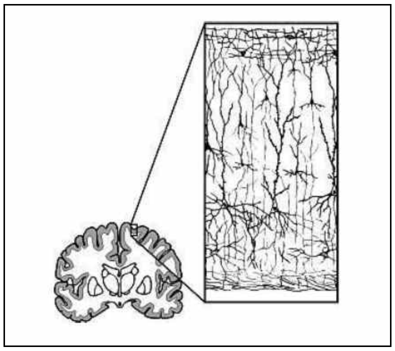


Figure 12. Columnar structure of the neocortex (from Dean [13]).

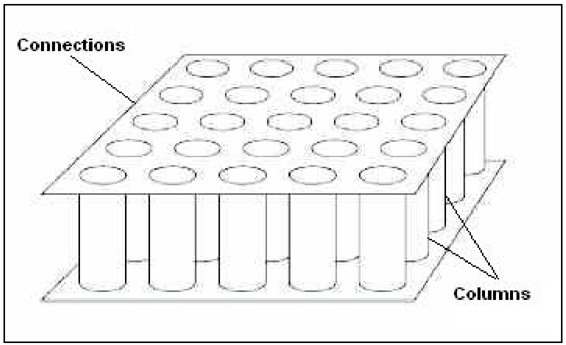


Figure 13. Model of the columnar structure of the neocortex (from Dean [13]).

The neurons have branching, wire-like structures called dendrites as shown in Figure 14. These usually branch out somewhat like tree branches and receive input into the neuron from other nerve cells. The neuron also has axons, the part of the neuron that is specialized to carry messages away from the neurons and to relay messages to other cells. Near its end, the axon divides into many fine branches which end in close proximity to the dendrites of another neuron, forming small connections called synapses. An average neuron forms approximately 1,000 synapses with other neurons. Electrical signals travel down the dendrites and axons, but the signals that travel across the synapses are primarily through chemical effects.

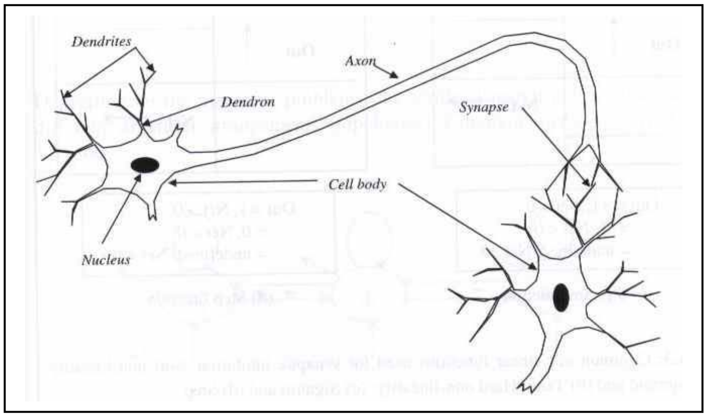
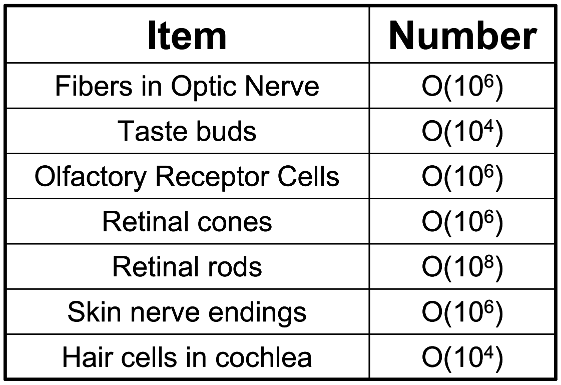


Figure 14. Schematic of a biological neuron (from Konar [17]).

Biological synapses can be inhibitory or excitatory. Glutamate (excitatory) and GABA (gamma-aminobutyric acid) (inhibitory) are the primary neural transmitters in the synapses [9]. The human ability to hear, remember, fear, or desire are all related to glutamate and GABA in the synapses. Human memory is a product of the synapses, and the process is often referred to as Hebbian learning [18]. Synaptic changes early in life are often referred to as "plasticity" and later in life are called "learning." The organization of the synapses is a combination of heredity and learning. Another interesting fact is that we are never really aware of brain processing, but only of the consequences of the processing. LeDoux [9] states that "we are our synapses" and that "Consciousness can be thought of as the product of underlying cognitive processes." Dennett [19] describes consciousness as "… a von Neumannesque virtual machine implemented in the parallel architecture of a brain that was not designed for any such activities." LeDoux also describes how human brains learn in parallel, somewhat like the old Connection Machine computers from Thinking Machines Corporation [20], which were very effective at solving some large scale problems [21-23].

In addition to considering the processing power of the human brain, one should also consider the input/output features of it. According to Mumford [15], all input to the cortex, except for olfactory, goes through the thalamus. The thalamus sort of acts like a 7th layer of the cortex. The output from the brain is more complicated; some of it bypasses the thalamus and goes to the spinal cord. Other outputs go directly to controlling eye motion. And a third type of output goes to the basal ganglia. Figure 15 summarizes the enormous amount of sensor inputs in humans, which are very approximate estimates. If computer models of the human brain are eventually developed, they will need to have an enormous number of inputs and outputs, as well as neurons. This will require massively parallel and scalable ANNs.



Item	Number
Fibers in Optic Nerve	O(10^6)
Taste buds	O(10^4)
Olfactory Receptor Cells	O(10^6)
Retinal cones	O(10^6)
Retinal rods	O(10^8)
Skin nerve endings	O(10^6)
Hair cells in cochlea	O(10^4)

Figure 15. Approximate values for human sensory inputs [faculty.washington.edu/chudler/facts.html].

V. Conclusions

Traditional ANN's on serial computers which are trained using back-propagation do not scale well. For massively parallel ANNs it is not practical to have the neurons at each level connected to all the neurons on the previous level, since this would require an enormous amount of inter-processor communication. In the approach described here the neural network is constructed in a columnar manner, which loosely mimics the connections in the human brain. In the future we could have layers of neurons in a two-dimensional or three-dimensional grid, and distribute them to processors in a checkerboard fashion. The software developed here is scalable and permits the use of billions of weights or synapses. The approach used here also allows an ANN to be trained on a massively parallel computer and then used on small serial computers.

Acknowledgments

The authors would like to thank the NASA Advanced Supercomputing Division (NAS) for the use of the Columbia supercomputer [28].

References

1. Seiffert, U., "Artificial Neural Networks on Massively Parallel Computer Hardware," *ESANN'2002 proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium)*, 24-26 April 2002.
2. Takefuji, Y., *Neural network parallel computing*, Kluwer Academic Publishers Norwell, MA, 1999.
3. Hassoun, M. H., *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, 1995.
4. White, D.A. and Sofge, D.A., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches,*Van Nostrand Reinhold, June 1, 1992.
5. Werbos, P., *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, John Wiley & Sons, New York, 1994.
6. Werbos, P.J., *Backpropagation: Basics and New Development, The Handbook of Brain Theory and Neural Networks*, MIT Press, First Edition, 1995.
7. Mitchell, T. M., *Machine Learning*, McGraw-Hill, NY, 1997.
8. Rumelhart, D.E. and McClelland, J.L., *Parallel Distributed Processing*, MIT Press, 1986.
9. LeDoux, J., *Synaptic Self*, Penguin, New York, 2002.
10. Hawkins, J., and Blakeslee, S., *On Intelligence*, Times Books, New York, 2004.
11. Carpenter, G.A. and Grossberg, S., *Adaptive resonance theory, In M.A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks*, Second Edition, Cambridge, MA: MIT Press, 87-90, 2003.
12. Taylor, J., "Neural networks of the brain: Their analysis and relation to brain images," *Proceedings of the IEEE International Joint Conference on Neural Networks*, Montreal, 2005.
13. Dean, T., "A Computational Model of the Cerebral Cortex", *AAAI Conference*, Pittsburgh, 2005
14. Mountcastle, V.B., "Introduction to the special issue on computation in cortical columns," *Cerebral Cortex*, Vol. 13, No. 1, 2003.
15. Mumford, D., "On the computational architecture of the neocortex I: The role of the thalamo-cortical loop," *Biological Cybernetics*, Vol. 65, 1991.
16. Mumford, D., "On the computational architecture of the neocortex II: The role of cortico-cortical loops", Biological Cybernetics, Vol. 66, 1992.
17. Konar, A., *Computational Intelligence*, Springer, New York, 2005.
18. Hebb, D.O., *The Organization of Behavior*, Wiley, New York, 1949.
19. Dennett, D.C., *Consciousness Explained*, Back Bay, Boston, 1991.
20. Hillis, D*., The Connection Machine*, MIT Press, 1989.
21. L. N. Long, M. Khan, and H.T. Sharp, "A Massively Parallel Euler/Navier-Stokes Method," AIAA Journal, Vol. 29, No. 4, April, 1991.
22. Ozyoruk and L. N. Long, "A New Efficient Algorithm for Computational Aeroacoustics on Massively Parallel Computers," Journal of Computational Physics, Vol. 125, No. 1, pp. 135-149, April, 1996.
23. D. Lewin, "Supercomputer '93 Showcases Parallel Computing Finalists," Computers in Physics, Vol. 7, no. 6, 1993.
24. http://www.research.ibm.com/bluegene/
25. Kurzweil, R., *The Age of Spiritual Machines*, Penguin, NY, 1999.
26. Moravec, H., *Robot: mere machine to transcendent mind*, Oxford University Press, November 1998.
27. Gerstner, W. And Kistler, W.M*., Spiking Neuron Models*, Cambridge Univ. Press, Cambridge, 2002.
28. http://www.nas.nasa.gov/Resources/Systems/columbia.html
29. Haykin, S., *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Prentice-Hall, 1999.
30. http://www.statsoft.com/textbook/stneunet.html
31. Barron, A.R., "Universal Approximation Bounds for Superpositions of Sigmoidal Functions," IEEE Transactions on Information Theory, Vol. 39, 1993.
32. Yang, D., *C++ and Object-Oriented Numeric Computing for Scientists and Engineers*, Springer, New York, 2000.
33. Shtern, V*., Core C++: A Software Engineering Approach*, Prentice-Hall, NJ, 2000.
34. Kumar, V., Grama, A., Gupta, A., and Karypis, G., *Introduction to Parallel Computing*, San Francisco: Benjamin Cummings / Addison Wesley, 2002.
35. The MPI Forum On-line, http://www.mpi-forum.org.
36. Al-Alaoui, M. A., Mouci, R., Mansour, M. M., Ferzli, R., "A Cloning Approach to Classifier Training", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, Vol. 32, No. 6, November 2002.
37. http://gears.aset.psu.edu/hpc/systems/rs6k/
38. http://www.cse.psu.edu/mufasa/