

PARALLELIZATION STRATEGIES ANALYSIS FOR SUPERVISED LEARNING

Snezhina Yanakieva

Technical University of Varna, Computer Science and Engineering Department, 9010, 1 Studentska Street, Varna, Bulgaria

Abstract

The algorithms for supervised learning in artificial neural networks (ANN) require time and high computational power. As these algorithms gain popularity in a variety of domains, it is critical for them to run fast. Following a brief survey of the different dimensions of parallelism in ANN this paper analyses the performance comparison between different parallelization techniques to show the advantages and disadvantages of these strategies.

Keywords: Parallel Neural Network, Multilayer Perceptron, Back-propagation, Bulk Synchronous Parallelism, Computer cluster

1. INTRODUCTION

Artificial neural networks (ANN) attempt to replicate, in simplified form, the way a human brain works. "Typically, the human brain consists of approximately 10^{11} neurons, each with an average of $10^3 - 10^4$ connections. It is believed that the immense computing power of the brain is the result of the parallel and distributed computing performed by these neurons"[1]. The biological models that inspire ANN systems are massively parallel with many simple biological cells working together to solve problems. Therefore, the implementation of massively parallel computational models on sequential machines is highly inefficient, thus it natural to develop algorithms that can be implemented on parallel computers.

2. MULTILAYER PERCEPTRON

The most attractive characteristic of an ANN is its ability to learn. A neural network is trained so that a set of inputs produces the desired (or at least a consistent) set of outputs. During the supervised learning, a set of training examples are presented that comprise the input vectors and the desired output vectors. Training is performed until the neural network learns to associate each input vector to its corresponding and desired output vector. Examples of ANN trained using supervised learning are the perceptron and the multilayer perceptron (MLP). An MLP consists of an input layer, at least one "hidden" layer, and one output layer, the neurons from each layer being fully connected (in some cases, partially connected) to the neurons from the next layer.



Fig. 1. multilayer perceptron (MLP) with one hidden layer

2.1. Back-propagation algorithm

A type of multilayer perceptron is the back-propagation neural network. The back-propagation model is used as a pattern classifier or a feature detector because it can recognize different patterns presented to its inputs.

The network topology is such that each neuron in a layer receives input from every neuron of the previous layer. Each neuron computes a weighted sum of all its inputs. Then a nonlinear activation function is applied to the sum resulting in an activation value of the neuron.

In the first stage of the algorithm, the input values are propagated forward through the network to compute the output values. Then the output vector of the network is compared with a target vector, provided by a supervisor, resulting in an error vector.

In the second stage, the values of the error vector are propagated back through the neural network. This pass attempts to correct any error made concerning the desired output. Each hidden layer then computes the contribution of its neurons to this error. This happens when each neuron in a layer calculates its from a weighted sum of the error of the succeeding layer. The weights for each neuron are adjusted in proportion to that neuron's error.

Back-Propagation Training Algorithm:

1 Forward stage:

1.1 Apply an input and its desired output vector.

1.2 Propagate the input signals forward through the network and compute the output.

1.3 Calculate the error vector.

2. Backward stage:

2.1 Propagate the error backward.

2.2 Adjust the weight between layers.

2.3 Repeat from 1.1.

The training population of input pairs is called an epoch. Training takes place over several epochs until the average error for the training population falls below a threshold. The training stops at this point and the neural network has converged. Once convergence has been achieved, the training is considered successful based on the network's ability to correctly generalize unseen input vectors.

3. DIMENSIONS OF PARALLELISM

To realize an implementation on a parallel computer, parts of the algorithm that can be run in parallel must be identified. Exploration of the smallest computational units reveals the different dimensions of parallelism[2].

3.1. Training session parallelism

Starting different training sessions on different processing elements describes training session parallelism. Every session may have its starting values for the weights and learning rates. The usage of parallel machines with complex control makes possible the training of different-sized neural networks at the same time.

3.2. Training example parallelism

This approach works by using the large number of examples as the source of parallelism. A subset of the examples is given to each of the processors and trains using the standard sequential technique. When the processor has processed its examples, the weight corrections are summed up across all processors which gives a global correction to the weights.

Training example parallelism does not cause communication overhead. Therefore it provides an almost linear speedup with the number of processors. However, this method demands that the processors have enough memory to store the activation values of all the nodes in a network.

3.3. Layer parallelism

In a multilayer network calculations can be pipelined allowing more than one training to go through the network at the same time.

3.4. Node parallelism

Node parallelism (neuron parallelism) uses the natural parallelism within the neural network. In the purest case, each processing element is responsible for computing the activation of a single neuron. This form of parallelism means that the calculations are viewed as matrix operations and each row of the matrix map is let onto a processor.

The activation values are sent forward after all nodes in a layer have been calculated. This means that the maximum parallelism is available for the widest mode layer. If this dimension of parallelism is fully utilized, the layers with fewer nodes cannot fully utilize the computational resource.

In practice, it is difficult to determine the best way to realize parallelism because of the number of different ways in which the work of training can be divided into concurrently-executable pieces, the communication costs, and the absence of realistic analytic ways to predict the performance of parallel architectures.

Training session parallelism and layer parallelism were considered unsuitable for this paper analysis for two reasons: they do not provide a general-purpose solution for neural network training and are not specifically studies of parallel algorithms.

This paper examines the performance differences between exemplar and node parallelism using theoretical BSP cost model and a practical implementation on a cluster computer.

4. ANALYZATION STRATEGIES

This paper analyses the performance of exemplar and node parallelization techniques for supervised learning by comparing the results from BSP (Bulk Synchronous Parallelism) cost model and cluster computer implementation.

4.1. Bulk Synchronous Parallelism

Bulk Synchronous Parallelism (BSP) is a new parallel computation model designed for communication-intensive applications. The predictions of its cost model are usually within a few percent range from the actual performance.

BSP's advantages are:

1. Separating communication from synchronization.
2. Communication is global. This makes the congestion visible enough to be minimized by the system.

A BSP is an abstract machine that consists of a set of processor-memory pairs connected by a communication topology. A BSP program is a sequential composition of supersteps. Each superstep is a parallel element in which each processor executes the same code. The superstep consists of three phases:

1. Local computation in each processor.
2. Global communication among processors.
3. A barrier synchronization, after which all communicated data gets stored in local memories.

The cost of a superstep is computed from two architectural parameters. The first one g measures the ability of the communication network to deliver continuous data. The maximum number of incoming

or outgoing messages for a superstep is denoted by h . The ability of a communication network to deliver data is captured by the parameter g , defined such that it takes time hg for a processor to deliver h messages of size 1. The cost of communication is dominated by the time needed to cross the boundary between processors and network. The second parameter l measures the time needed for barrier synchronization.

$$\text{MAX}_{\text{processes}} w_i + \text{MAX}_{\text{processes}} h_i g + l$$

If we consider a superstep in which processor i takes w_i instructions for its local computation and sends and receives h_i words the cost of the superstep is the sum of the costs of three phases (computation costs + communication costs + number of supersteps). [3]

This formula is not only theoretical approximation, but it has shown to be very accurate across a wide range of architectures and applications[4].

4.2. Computer cluster implementation

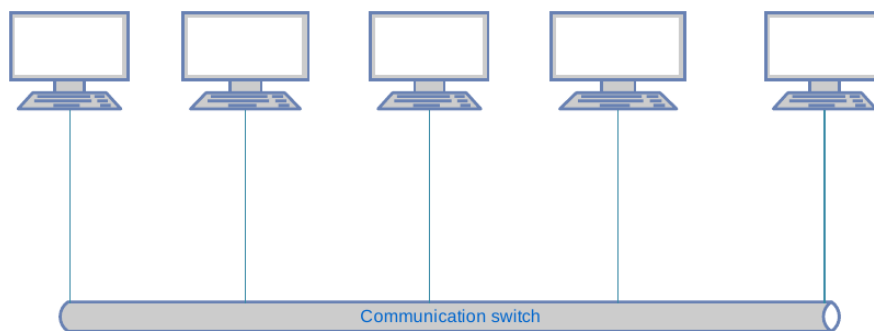


Fig. 2. Cluster computer architecture

An alternative to multiprocessor parallel computers is a cluster computer built using a network of workstations. A computer cluster is a collection of computers that work together as a single system. The cluster environment differs from a shared memory parallel computer in the communicational costs but has the advantage over multiprocessor parallel systems in that they are relatively inexpensive to construct and provide a highly scalable parallel platform.

5. BSP COST ANALYSIS

R.O. Rogers and D.B. Skillicorn [3] examined the relative cost of three parallelization techniques for supervised learning using multilayer perceptron. They observed that the number of weights W that have to be updated to learn one exemplar is:

$$W = N^2(L - 1),$$

where N is the number of input neurons, L is the number of layers.

5.1. Exemplar parallelism

The learning is divided into two phases:

1. Each processor learns independently from a subset of exemplars.
2. The global computation.

The implementation happens using two BSP Cost analysis supersteps.

First superstep requires each processor to calculate W/p weight updates for every exemplar, which makes the computation cost:

$$AEW / p,$$

where A is the number of operations it takes to update each weight.

At the end of the superstep, each processing element passes the weight correction to one processor. Where the global computation (second superstep) is done. The cost of the communication is:

$$(p - 1) Wg,$$

where p is the number of processors.

Experimental results, shown by J.M.D.Hill and D.B. Skillcorn, merging corrections using a tree perform poorly on modern architectures because the synchronization performance is poor. The faster method is likely to be a BSP barrier synchronization. The barrier method of synchronization for a group of threads or processes means any thread or process must stop at a point and cannot proceed until all other threads or processes reach that barrier.

Every calculation performed by the processor that computes the final weight correction from the corrections received from all of the other processors takes time - (p - 1)W.

The overall BSP cost of exemplar parallel implementation is:

$$\left[\frac{AEW}{p} + (p - 1)W \right] + [(p - 1)W]g + 2l$$

5.2. Node parallelism

For node parallelism to each processor are assigned N/p neurons. Inter-processor communication is needed when an error or activation has to be transferred between neurons assigned to different processors. The processors can be arranged in rows. These rows pass information in both directions – activation upwards and error downwards. Each communication forms a superstep. J.M.D.Hill and D.B. Skillcorn calculate the actual number of supersteps to be :

$$(E - 1) + 2(p - 1).$$

And the computation required for each superstep as:

$$AW / p.$$

At the beginning of each superstep every processor in a row sends values to each processor in a subsequent row. This communication requires to send and receive a proportionate share of the weights W, which makes the communication volume per superstep:

$$2W / p.$$

If we set z as the number of supersteps: $z = (E - 1) + 2(p - 1)$.

This makes the total BSP cost of Node parallelism:

$$[AW / p]z + [2W / p]zg + zl$$

5.3. Comparison of Exemplar parallelism and Node parallelism

Both methods have almost the same computation time, except that the exemplar parallelism performs additional post-processing. It is the communication costs that significantly differ. Exemplar parallelism is the clear winner when the number of supersteps is taken into account, especially considering that the cost of a barrier synchronization (l) is large in today's systems. The differences are even bigger as the number of exemplars increases.

6. CLUSTER COMPUTER IMPLEMENTATION

M. Pethick, M. Liddle, P. Werstein, and Z. Huang implement the two strategies on a cluster computer in order to determine their scalability when the size of the network, the training population and the number of processors vary[7].

Their tests are made using three-layered neural networks with neurons organized as $N \times 3/4N \times 1/2N$, where values of N are 250, 500, 1000, or 2000. This configuration is chosen because it provided a regular pattern for several different neural network sizes, thus ensuring that network layout would not impact the scalability results. The training pairs are in the range of 0.1 – 0.9, tested using training populations containing 100, 1000, 10000, or 20000 training pairs. Each of the tests was conducted out on a cluster computer with 1, 2, 4, 8, 16, or 32 processors. Each test lasts for 50 epochs. The results show the speedup between the time taken to complete the core algorithm in the simulated environment and the time taken to complete the core algorithm by a cluster computer with 1 processor.

6.1. Exemplar parallelism performance results

The exemplar parallel implementation presents a strong correlation between the network performance and the dataset size. For every network, the speedup grows with the dataset. However, the results show a lower correlation between the performance and network size. The exemplar parallel realization shows that the speedup on 16 processors starts to drop when the network size increases to 2000.

The results look like this because the data set size defines the amount of computation done by each process for an epoch, at the same time the size of the network defines the amount of data that is sent. Therefore the larger the dataset the higher the computation/communication ratio, which makes the speed up higher. When the network size reaches a threshold the communication overhead is large enough to overshadow the benefit due to parallelization of the network and the speedup starts to drop.

Dataset size	5000	10000	20000
Speedup	Between 6 and 8 times	12 times	Between 12 and 14 times

Table 1. Exemplar parallelism performance based on dataset size

Neural network size	500	1000	2000
Speedup	More than 14 times	More than 14 times	Between 12 and 14 times

Table 2. Exemplar parallelism performance based on network size

6.2. Node parallelism performance results

The node parallel implementation shows a strong correlation between performance and the size of the neural network. For a given dataset size the speedup rises steadily with the increased size of the network. With the best performance being speedup of 5.99 times on 16 processors. For the tested range of neural network sizes, the speedup of the node parallelism version keeps growing steadily and reaches up to 16.36 on 32 processors.

The experimental results present only a very limited correlation between performance and dataset size. For a given size of the network, the speedups are almost the same for different dataset sizes such as 1000 and 10000, although there is a slight trend for the speedup on 32 processors when the dataset size grows to 10000.

Dataset size	5000	10000	20000
Speedup	Between 10 and 12 times	Between 10 and 12 times	Between 10 and 12 times

Table 3. Node parallelism performance based on dataset size

Neural network size	500	1000	2000
Speedup	Between 4 and 6 times	Between 4 and 6 times	6 times

Table 4. Node parallelism performance based on network size

6.3. Comparison of Exemplar parallelism and Node parallelism

The impact of the size of the network is higher on the exemplar parallel implementation than the impact of the data set size is on the node parallel implementation. The exemplar parallelism appears to perform slightly worse on the neural network size 2000 than the size 1000. This is generally expected as the neural network size determines the amount of network traffic and its communication load is more evenly distributed.

The worst-case performance for node parallelism is when the network size is very small, while for exemplar parallelism the worst-case performance is when the dataset size is very small. In those cases, both applications show a significant slowdown as the number of processors increases.

7. CONCLUSION

BSP cost model shows the exemplar parallelism as the preferred technique, but the practical realization on a cluster computer shows that the choice of which strategy to use will be driven primarily by the type of learning required. For small datasets or very large networks, node parallelism manages to outperform exemplar parallelism. With large datasets, the exemplar parallel strategy usually outperforms node parallelism, especially with smaller network sizes.

REFERENCES

1. Rumelhart, & E., David & McClelland, James & L., James. (1986). Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations.
2. T. Nordstrom and B. Svensson, "Using and designing massively parallel computers for artificial neural networks," Journal of Parallel and Distributed Computing, vol. 14, no. 3, pp. 260–285, 1992.
3. Rogers, R. & Skillicorn, David. (1997). Strategies for Parallelizing Supervised and Unsupervised Learning in Artificial Neural Networks Using the BSP Cost Model.
4. Jonathan M. D. Hill, Paul I Crumpton, and David A. Burgess. The theory, practice, and a tool for BSP performance prediction. In EuroPar'96, volume 1124 of LNCS, pages 697{705. Springer-Verlag, August 1996.
5. S. Becker and M. Plumbley. Unsupervised neural network learning procedures for feature extraction and classification. Journal of Applied Intelligence, 6:1{21, 1996.
6. M. Besch and H.W. Pohl. How to simulate artificial neural networks on large scale parallel computers exploiting data parallelism and object orientation. Technical Report TR-94022, GMD FIRST Real World Computing Laboratory, November 1994.
7. Pethick, Mark & Liddle, Michael & Werstein, Paul & Huang, Zhiyi. (2003). Parallelization of a Backpropagation Neural Network on a Cluster Computer. 15.

8. M. Besch and H.W. Pohl. Flexible data parallel training of neural networks using MIMD computers. In Third Euromicro Workshop on Parallel and Distributed Processing, January 1995.
9. P. Frasconi, M. Gori, and G. Soda. Daphne: Data parallelism neural network simulator. International Journal of Modern Physics C, 1992.
10. S. Haykin. Neural Networks: A Comprehensive Foundation. Prentice-Hall, 1994.
11. J.M.D. Hill and D.B. Skillicorn. Practical barrier synchronisation. Technical Report TR-96-16, Oxford University Computing Laboratory, October 1996. 9
12. Jonathan M. D. Hill, Paul I Crumpton, and David A. Burgess. The theory, practice, and a tool for BSP performance prediction. In EuroPar'96, volume 1124 of LNCS, pages 697{705. Springer-Verlag, August 1996.
13. D. Rumelhart, G. Hinton, and R. Williams, Learning internal representations by error backpropagation, in D. Rumelhart, J. McClelland and The PDP Research Group (Ed.) Parallel Distributed Processing: Explorations into the Microstructure of Cognition, 1 (Cambridge: The MIT Press, 1986) 318–362.
14. M. Quoy, S. Moga, P. Gaussier, and A. Revel, Parallelization of neural networks using PVM, in Lecture Notes in Computer Science, 1908 (Berlin: Springer Verlag, 2000) 289–3
15. M. Misra, Parallel environments for implementing neural networks, Neural Computing Surveys, 1, 1997, 48–60.
16. Almeida, L. D. Backpropagation in perceptrons with feedback. In NATO ASI Series: Neural Computers. Neuss, Federal Republic of Germany, 1987.
17. Arbib, M. A. Metaphorical Brain 2: An Introduction to Schema Theory and Neural Networks. Wiley-Interscience, New York, 1989.
18. Arbib, M. A. Schemas and neural network for sixth generation computing. J. Parallel Distrib. Comput. 6, 2 (1989), 185'--216.
19. Asogawa, M., Nishi M., and Seo, Y. Network learning on the supercomputer. In Proc. 36th IPCJ Meeting, 1988, pp. 2321--'2322. [In Japanese]
20. Barash, S.C., and Eshera, M.A. The systolic array neurocomputer: Fine-grained parallelism at the synaptic level. In First International Joint Conference on Neural Networks, 1989.