International Journal of Artificial Intelligence and Neural Networks – IJAINN
Volume 4 : Issue 3 [ISSN 2250 – 3749]
Publication Date : 30 September, 2014

Performance Optimization of Levenberg-Marquardt Algorithm with Parallelization

Nirmal Lourdh Rayan S.
Computer Science and Engineering,
Christ University Faculty of Engineering
Bangalore, India

K. Balachandran
Computer Science and Engineering,
Christ University Faculty of Engineering
Bangalore, India

***Abstract*—Mathematical Optimization refers to finding the minimum or maximum value from a desired set of outcomes. This paper discusses about optimization in two levels. Levenberg-Marquardt is used for back propagation to minimize non-linear least square error using curve fitting. This minimization involves functional optimization to reduce error in neural network (NN) classification. The second level of optimization is on improving the performance of Levenberg-Marquardt algorithm (LMA) by using divide and conquer methods to parallelize computation.**

We make use of Fork/Join framework in Java which uses divide and conquer technique to split a task into many elementary subtasks and executing them in parallel. Additionally, the Fork/Join architecture uses work-stealing algorithm to effectively utilize the worker threads that have completed their tasks to steal tasks from other threads that are still busy.

We have used standard UCI Machine Learning Repository dataset called Million Song Dataset for constructing the neural network. The target output will be the year of song's publication and the input vector consists of the metadata and characteristics of audio (song).

The effective speedup achieved for varying data sizes are estimated by comparing the performance of traditional LMA with parallelized LMA. We also study the rate of improvement in performance when the input data sample size is varied from 100 to 1,00,000. We have achieved over 300% steady gain in performance using thread level parallelism on LMA in a single workstation.

Keywords—Levenberg-Marquardt; Back Propagation; Neural Networks; Optimization, Fitting, Parallelization; Fork/Join; Divide and Conquer; Java.

I. INTRODUCTION

The **brain** is a highly complex, nonlinear, and parallel computer (information-processing system). It is an organ that serves as the center of the nervous system in all vertebrate and most invertebrate animals. In neuroscience, a *biological neural network* (also called a neural pathway) is a series of interconnected neurons whose activation defines a recognizable linear pathway. Several axon terminals form an interface through which neurons interact with their neighbors connected via synapses to dendrites on other neurons. The structure of a biological neuron is given in Fig. 1. This forms a network of neurons. If the sum of the input signals into one neuron surpasses a certain threshold, the neuron sends an action potential (AP) at the axon hillock and transmits this electrical signal along the axon.

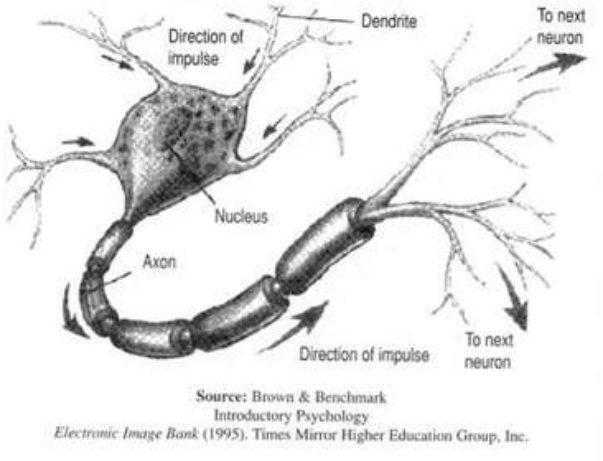

Fig 1. Structure of a Neuron.

A. Artificial Neural Network

An *artificial neural network* is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. Neural network system is currently sethe most active subject in artificial intelligence research. It is also the achievement created by human manufacturing, scientific research, artificial intelligence and computer technology [5].

Perceptron: It is an algorithm for supervised classification of an input into one of several possible non-binary outputs in neural networks. Perceptron is a type of linear classifier that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Fig. 2 depicts a simple perceptron. It consists of three layers: input, hidden and output. The core functionality performed on the input data to get the desired output can be subdivided into two units: the summation unit and activation unit. The summation unit gives the sum of product of weights

and their corresponding input signal strength. A multi-layer perceptron (MLP) is a weighted directional bipartite graph. MLP is created by LMA for classifying Million Song Dataset from UCI Machine Learning Repository using error back propagation. The second unit consists of the activation function which uses a threshold to estimate the output of a neural network. Different types of activation functions exist such as Linear Threshold (as shown in Fig. 2), Step function, Sigmoid function, Multi-quadratic function, Gaussian function, Tanh function etc.

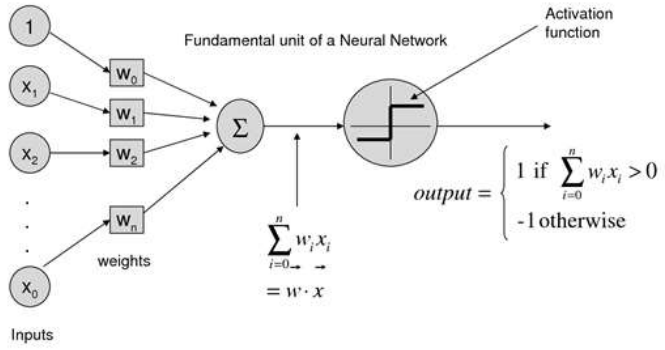

Fig 2. Perceptron with Linear Threshold Activation Unit

B. Levenberg-Marquardt Algorithm

The **Levenberg-Marquardt algorithm** (Referred as LM henceforth) is an iterative technique that locates a local minimum of a multivariate function that is expressed as the sum of squares of various non-linear, real-valued functions. LMA has become a standard technique for nonlinear least-square problems, vastly adopted in various disciplines for handling data-fitting applications.

LM can be thought of as a combination of steepest descent and the Gauss-Newton method.

- The LM algorithm behaves like a steepest descent method when the current solution is far from a local minimum: slow, but guaranteed to converge.
- It becomes a Gauss-Newton method and exhibits fast convergence when the current solution is close to the local minimum [7].

In our research, an optimized Levenberg-Marquardt algorithm is implemented, by harnessing thread level parallelism in Java, to enhance performance of training a neural network.

C. Fork-Join Framework

The Fork/Join architecture, as shown in Fig. 3, is introduced in Java SE 7 to make use of all the processing cores available in a single workstation to perform execution of tasks in parallel instead of sequential. It is an implementation of the ExecutorService interface that helps in effectively sharing workload among multiple processors. This framework is designed by Java for work that can be broken into smaller pieces recursively.

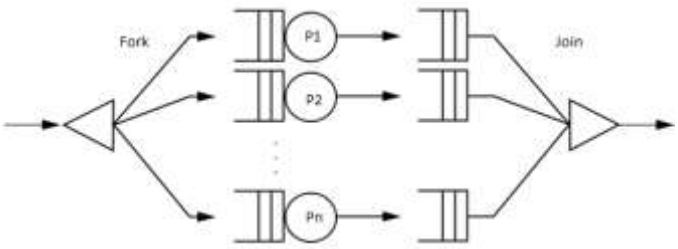

Fig 3. Fork-Join Architecture displaying task queue for each processor

D. Description of Terms

Neurons: *A cell capable of transmitting nerve impulses; a nerve cell.*
Pathway: *A chain of nerve fibers along which impulses normally travel.*
Axon: *The long process of a nerve fiber that conducts impulses away from the body of the nerve.*
Synapse: *The link between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter.*
Dendrites: *The short branched extension of a nerve cell, along which impulses received from other cells at synapses are transmitted to the cell body.*
Minima: *The minimum of a function is the smallest value that the function takes at a point either within a given neighborhood or on the function domain in its entirety (global or absolute extremum).*
Speedup: *It refers to how much a parallel algorithm is faster than a corresponding sequential algorithm.*
Synaptic Weight: *It refers to the strength or amplitude of a connection between two nodes*

II. Related Work

A thorough understanding of the current research in the field of Neural Networks is necessary to identify areas that require improvements. Prior researches indicate promising outcomes in terms of enhancing LM algorithm in space and time domain however, there are only a few efforts on optimization of LMA.

Lera et. al. [1] present a quasi-local Levenberg-Marquardt algorithm for neural network training. The main idea is to consider these neighborhoods as independent learning units. By doing this, they applied LM reduced to one neighborhood in each step of the algorithm, decreasing the number of operations and the memory required. Thus, a more local method is obtained, meaning that, in order to adapt a given neuron's weights we only need information about its neighborhood. They prove that these neighborhoods significantly decrease memory and time requirements imposed by the Levenberg-Marquardt method.

Zhao et. al. compare the quality of various BP algorithms in neural network toolbox of MATLAB and prove the superiority of trainlm. Especially for medium-sized networks, trainlm has a fast convergence, but the different algorithm should be selected in the different network [2].

Zhang et. al. adopt LMA to achieve Gaussian fitting on multiple GPUs (Graphics Processing Units). Since the algorithm involves plenty of matrix operations, it is appropriate to take advantage of GPU to deal with this parallel problem. They achieve quick Gaussian fitting on massive numbers of particle images taken by an ultra-microscope using multiple GPUs and show that near proportional growth of computing speed can be reached as the number of GPUs are increased. They analyzed the LMA and found that there exists a large number of matrix operations, including matrix multiplication, addition and calculating the Jacobian matrix. Therefore, each block in GPU was assigned to fit a curve, and the threads in each block adopt appropriate parallel methods, including reduction algorithms. Parallel computing was applied in both task-level and instruction-level for better improvement of curve fitting [3].

Reynaldi et. al. train finite element based on neural network using Back propagation and Levenberg-Marquardt algorithm. The purpose was to solve differential equation and inverse problem of differential equation. They formulate hybrid finite element neural network using both back propagation algorithm and Levenberg-Marquardt algorithm for solving inverse problem [4].

Jian-rong Li applied and modeled Levenberg-Marquardt algorithm to establish a neural network model for predicting the damage of the oil and gas layers to protect the layers and provide effective assistance. The neural network constructed had a maximum error of 0.022249% and accuracy above 99%. The technology will be an innovation in the damage assessment of oil and gas layer, for it has perfected the layers monitoring and applied neural network model into predicting damage [5].

A research on whether Levenberg-Marquardt is the most efficient optimization algorithm for implementing bundle adjustment was conducted by Lourakis et. al. Bundle Adjustment (BA) is often used as the last step of many feature-based 3D reconstruction algorithms. BA is typically the most time consuming computation in such algorithms [7].

Gavin P.H describe the Levenberg-Marquardt method for nonlinear least square curve-fitting problems. Least square problems arise when fitting a parameterized function to a set of measured data points by minimizing the sum of the squares of the errors between the data points and the function. Nonlinear least square problems arise when the function is not linear in the parameters. Least square methods (nonlinear) involve an iterative improvement to parameter values in order to reduce the sum of the squares of the errors between the function and the measured data points. [8]

A. *Current Systems*

The current Machine Learning algorithms are implemented in sequential programming environment using Java (translated from FORTRAN - Numerical Recipes in FORTRAN). Existing Levenberg-Marquardt algorithm is programmed sequentially to function in a single processing environment. Zhang et. al. adopt LMA to achieve Gaussian Fitting using multiple GPUs. The complex matrix operations were parallelized of increase the performance.

B. *Limitations of Existing Systems*

Although researchers have tried implementing LM algorithm in distributed environment, there is a lack to parallelization techniques that make use of the resources offered by a single workstation.

C. *Proposed System*

The multi-core environment available with computers these days allow us to make use of programming paradigms that make use of divide and rule techniques for work sharing and load balancing. Additionally, the work-stealing algorithm within Fork/Join framework allows us to re-use existing threads that have completed their tasks instead of creating new threads for handling new tasks. We also achieve this with very minimal synchronization cost. Our research discusses the implementation of LM algorithm in a commodity workstation consisting of multi-cores processing units, RAM and cache memory.

III. DATASET

We have used standard dataset available for download for academic research in UCI Machine Learning Repository.

The Million Song Dataset (or MSD) is a freely-available collection of audio features and metadata for a million contemporary popular music tracks.

The core purposes of MSD include:

- Providing a reference dataset for evaluating research
- Encouraging research on algorithms that scale to commercial sizes
- Acts as a shortcut alternative to creating a large dataset with APIs (e.g. The Echo Nest's)
- Helping new researchers get started in the MIR field

The dataset is provided by The Echo Nest. The core of the dataset is the feature analysis and metadata for one million songs. The dataset only includes the derived features or meta-data not any audio.

In this paper, we have not performed data pre-processing on the input dataset before training the neural network. This step has been ignored, although it is of prime importance in building a neural network because our main focus is not to create the most accurate neural network but instead to create neural network with and without parallelization to estimate the speedup gained. Therefore, we have used all the 90 input attribute vectors without performing feature subset selection.

IV. METHODOLOGY

Fork/Join parallelism is used for obtaining good parallel performance and is among the simplest and most effective design techniques. Fork/join algorithms are parallel implementations of divide−and−conquer algorithms which takes the typical form:

```
Result solve (Problem problem) {
if (problem is small)
    directly solve problem
else {
    split problem into independent parts
    fork new subtasks to solve each part
    join all subtasks
    compose result from subresults
    }
}
```

The fork operation starts a new parallel fork/join subtask. On the other hand, the join() operation causes the current task not to proceed until the forked subtask has finished. Like other divide−and−conquer algorithms, fork/join algorithms also are nearly always recursive, continuously splitting subtasks until they are small enough to solve using short, simple sequential methods.

There are two specific methods featured in ForkJoinTask objects:

- The **fork()** method allows a ForkJoinTask to be planned for asynchronous execution. This splits the parent task into subtasks and allows a new ForkJoinTask to be launched from an existing one.
- The **join()** method allows a ForkJoinTask object to wait for the completion of another one.

V. Experimental Setup

A modification to the LMA is applied in our experiment. The original algorithm is described in Numerical Recipes in FORTRAN, 2nd edition, p. 676-679, ISBN 0-521-43064X, 1992. The Java version of LMA package version 1.2 is made available by J. Holopainen and it is free for non-commercial use. We have used Eclipse IDE for development and debugging LMA. For monitoring performance, we use jConsole plugin for Java. However, we have given explicit instructions to run the code in command prompt.

A. Evaluation Metrics

The overall performance of individual classifier is measured by

$$Accuracy = \frac{\#\ of\ correctly\ labeled\ songs}{\#\ of\ all\ songs\ in\ the\ test\ dataset} \quad (1)$$

In the test dataset, let E be the set of songs with year prediction e, E' be the set of songs which are classified as year e by the NN classifier, then we define the precision on years e as:

$$Precision\,(e) = \frac{|E \cap E'|}{|E'|} \quad (2)$$

the recall on e as:

$$Recall\,(e) = \frac{|E \cap E'|}{|E'|} \quad (3)$$

and the F-measure on e as:

$$F - Score = \frac{2 * precision * recall}{precision + recall} \quad (4)$$

which is a generalization of F_β-Score below:

$$F_\beta\text{-}Score = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall} \quad (5)$$

B. Example: 100 songs dataset

Attributes: 100

Kappa Statistics: 0.25

Limitation: The dataset is too small to obtain a good neural network classifier. The neural network constructed has only 31% accuracy in correctly classifying the output. However, it can be used for the purpose of demonstrating the confusion matrix and other measures of NN.

Table 1 shows the measures calculated from the confusion matrix obtained after training the neural network with 100 songs. The area of interest or area under the curve is obtained by plotting a graph using TP Rate and FP Rate. This is called the Receiver Operating Characteristic (ROC) curve.

TABLE I. DETAILED ACCURACY BY CLASS FOR 100 SONGS DATASET

	TP Rate	FP Rate	Precision	Recall	F-Measure	Class
	0	0	0	0.7209	0	1987
	0	0	0	0.7209	0	1989
	0.1	0.1	1	0.6976	0.8218	1992
	0	0	0	0.7209	0	1995
	0	0	0	0.7209	0	1996
	0.6666	0.1818	0.667	0.6744	0.6706	1997
	0.125	0.5	0.125	0.6976	0.212	1998
	0.125	0.1	0.125	0.6976	0.212	1999
	0.4615	0.4285	0.4615	0.5813	0.5145	2000
	0.3333	0.1667	0.3333	0.6744	0.4461	2001
	0	0	0	0.7209	0	2002
	0.75	0.2307	0.75	0.6511	0.697	2003
	0	0	0	0.7209	0	2004

	0.5	0.1	0.5	0.6976	0.5824	2005
	0	0	0	0.7209	0	2006
	0.2857	0.2857	0.2857	0.6744	0.4013	2007
	0.2667	0.4	0.2667	0.6279	0.3743	2008
	0.8889	0.3478	0.8889	0.5348	0.6678	2009
	0	0	0	0.7209	0	2010
Weighted Average	**0.2369**	**0.1495**	**0.2843**	**0.5752**	**0.2947**	

VI. RESULTS

TABLE II. EXECUTION TIME OF LMA WITH AND WITHOUT PARALLELIZATION FOR VARYING DATASIZE

Sl. No.	Data Size (MSD)	Execution Time of LMA (ms.)	Execution Time of LMA with Parallelization (ms.)	Speedup
1	100	261.76	464.11	0.564
2	500	1165.55	1280.77	0.91
3	1,000	2303.14	2385.65	0.9654
4	5,000	13145.76	6697.56	1.9627
5	10,000	39368.64	17164.32	2.2936
6	50,000	342924.76	142553.47	2.4055
7	1,00,000	1231735.58	368185.11	3.3454

Table II above describes the number of samples taken for training the neural network, time taken by LM algorithm in the sequential implementation, time taken by LM algorithm in parallelized implementation and the speedup achieved.

VII. EFFECT OF INCREASING THE TRAINING DATA

It is noteworthy to observe that the traditional LMA outperforms parallelized LMA when the sample dataset size is 100 to 1,000 although there is a gradual increase in the performance of parallelized LMA. However, when the sample dataset size is increased to 5000 the fork/join LMA gives almost twice the gain in performance. The effect of increasing the training data from 5,000 to 1,00,000 shows a steady rise in performance in the parallelized LMA with respect to serial LMA and obtain a speedup of 3.3454. This implies that the parallelized algorithm works 334.54% faster than the original LMA. To be sure, we ran the programs 100 times for each sample size in both original LMA and parallelized LMA mode. We calculated the representational average execution time in each mode for different sample sizes.

VIII. DISCUSSION

Clearly we have two different scenarios:

1) When the sample size is less (between 100 to 1000), the original LMA outperforms parallelized LMA and
2) When the sample size is more (over 1000), the parallelized LMA outperforms original LMA drastically as the size increases.

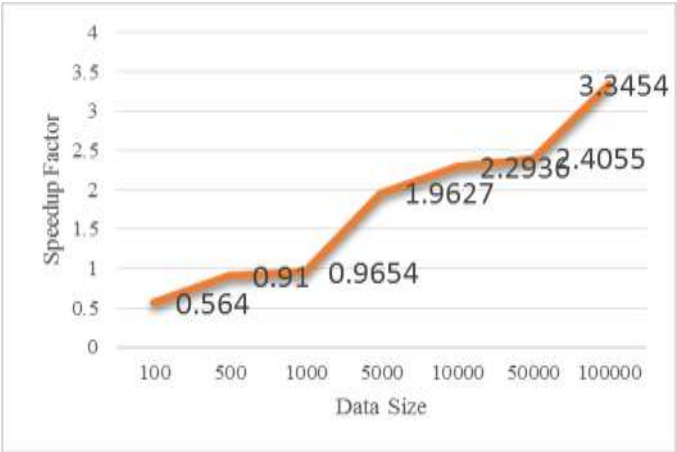

Fig 4. Change in speedup with change in data size for parallelized LMA

The first scenario occurs because the overhead involved in creating the thread pool is more in parallelized LMA with respect to the size of the dataset. The scale of the dataset is too small to hide the cost of thread initialization, and destruction, etc.[3]

The second scenario clearly depicts the effective utilization of parallel processing using threads where heavy computational load is distributed among threads. Internally, Fork/Join architecture implements work-stealing algorithm to distribute workload to different threads that have completed but can be used for executing other tasks before destruction. This avoids in creation of few threads by reusing existing threads that steal work from other de-queue. The computational cost involved in calculating Chi2 value for each input vector array element is cumbersome when the input data size becomes massive. The partial derivative for each input vector value is calculated by a thread (divide phase) and a cumulative result of the partial derivatives of each thread is obtained later (conquer phase).

Although sometimes neural networks are trained using small datasets, most-often they require large input data values to

accurately train a model. It has been estimated that Levenberg-Marquardt algorithm is by far the fastest back propagation algorithm existing for curve fitting problems in neural networks. In our research, we have parallelized Levenberg-Marquardt algorithm to optimize a neural network even faster as the data size grows. We have implemented this algorithm in Fork/Join architecture in Java to allow cross platform compatibility.

IX. Conclusion

We have provided the entire code with embedded documentation for parallelized Levenberg-Marquardt algorithm using Java's Fork/Join Framework.

http://sourceforge.net/projects/parallelizedlma/files/lma/lma%20with%20threading.rar/download

This project is freely available to the research community for academic purposes and to promote further research in this domain.

In our research, we have achieved over 300% steady performance gain by using thread level parallelization to optimize and train a neural network using Levenberg-Marquardt algorithm. The inherent problem of back-propagation requires additional time to train a model and usually it would require a large dataset. Levenberg-Marquardt optimization gives a solution to finding the minimum least squared error in a curve fitting plane. We try to estimate the global minima using this algorithm. Since most of the time is consumed in calculating the partial derivatives in the CalculateChi2 function, we have implemented thread level parallelization that uses Divide and Conquer technique to fork() subtasks and eventually obtain the final result by performing join() operations.

References

[1] G. Lera, M. Pinzolas, "A quasi-local Levenberg-Marquardt algorithm for neural network training", in Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on (Volume:3), pp. 2242–2246 vol.3.

[2] Z. Zhao, H. Xin, Y. Ren, X. Guo, "Application and Comparison of BP Neural Network Algorithm in MATLAB", in Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on (Volume:1), pp. 590–593.

[3] L. Zhang, Y. Zhao, K. Hou, "The Research of Levenberg-Marquardt Algorithm in Curve Fittings on Multiple GPUs," in Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, pp. 1355–1360.

[4] A. Reynaldi, S. Lukas, H. Margaretha, "Backpropagation and Levenberg-Marquardt Algorithm for Training Finite Element Neural Network," in Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium, pp. 89-84.

[5] J. Li, "The application and modeling of the Levenberg-Marquardt algorithm," published in e-Business and Information System Security (EBISS), 2010 2nd International Conference, pp. 1-3.

[6] G. Lera and M. Pinzolas, "Neighborhood Based Levenberg–Marquardt Algorithm for Neural Network Training," in Neural Networks, IEEE Transactions on (Volume:13 , Issue: 5), Sep 2002, pp. 1200–1203.

[7] M.I.A. Lourakis and A. A. Argyros, "Is Levenberg-Marquardt the Most Efficient Optimization Algorithm for Implementing Bundle Adjustment?," Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference (Volume:2), pp. 1526–18531 Vol2, 2005.

[8] H.P. Gavin, "The Levenberg-Marquardt method for nonlinear least squares curve-?tting problems," Duke University. 2013.

[9] K. Levenberg, "A method for the solution of certain problems in least squares, Quart. Appl. Math., 1944, Vol. 2, pp. 164–168.

[10] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," SIAM J. Appl. Math., 1963, Vol. 11, pp. 431–441.

[11] M.I.A. Lourakis, "A brief description of the Levenberg-Marquardt algorithm implemented by" levmar, Technical Report, Institute of Computer Science, Foundation for Research and Technology- Hellas, 2005.

[12] Ranganathan A., "The Levenberg-Marquardt Algorithm", 2004.

[13] Demuth H., Beale M., Hagan M., "Neural Network Toolbox 6 – User Guide," The MathWorks, Inc.

[14] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011)