THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

GPU Implementation of a Deep Learning Network for Financial Prediction

Robin Kumar
Department of Computer science and Engineering, Guru Nanak Dev University, Amritsar, India
Amandeep Kaur Cheema
Department of Computer science and Engineering, Guru Nanak Dev University, Amritsar, India

Abstract:
Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. Neural network is the well-known branch of machine learning & it has been used extensively by researchers for prediction of data and the prediction accuracy depends upon fine tuning of particular financial data. In this paper neural networks have been used for financial prediction and to improve the performance, Graphics Processing Units have been used. They have emerged as important players in the transition of the computing industry from sequential to multi- and many-core computing. In this paper OPENCL and APARAPI have been used in Java to implement & analyse the same.

Keywords: *Neural Networks, GPU, Financial Prediction, 0pencl, APARAPI*

1. Introduction

Human brain has more than 86 billion neurons in on average. Neural networks mimic the human brain. Machine learning or artificial intelligence is a very big field in which neural networks are responsible for creating a learning network like brain.
Financial forecasting, as one of the most common financial investment decision making problems, is the process of estimating future business performance by mapping the input and output data in order to discover the patterns that govern the observed movements. Given that financial data is highly time-variant and non-linear, predicting the future value of the financial variable is a challenging task. Evolutionary computation is frequently applied to the problem of financial investing because of its potential capability to model dynamic and non-linear relationships in data [12]. The biggest challenge for financial professionals and researchers is the existence of uncertainty and risk. While gambling with investment choices is not an option for most, proper research and planning can greatly reduce risk and guide investors to take the right steps. Artificial neural networks gained great popularity in the financial field, for their ability to deal with uncertainty and handle noisy data [1]. Neural network architecture is suitable for financial data but reaching a good design and fast execution is a big challenge. For the fast execution Graphical processing unit is being used due to its parallel architecture. One of the best advantages of GPU over the Central Processing Unit (CPU) is its lower cost to create parallel threads on blocks due to its efficient hardware implementation, whilst the CPU incurs an overhead to switch to another program. For this reason, the GPU hardware architecture allows the improvement of computational performance in massive data scenarios. It is worth noting that the GPU seems to be a natural candidate for massive processing of financial high-frequency data on forecasting applications [4]. This paper is organized as follows:
Section II shows the different types of learning methods. Also, it gives the information about neural networks concepts. Section III gives the detail about methodology, GPU parallel architecture and OPENCL, APARAPI. Concluding remarks and future directions are reported in Section V.

2. Type of Learning

Neural networks are a class of machine learning systems that were originally inspired by the architecture of the brain. Although none of the models are true to the information processing mechanisms of nature, they all borrow the basic idea: a system of interconnected units that can become activated due to some combination of input stimulation.
The learning process can be separated distinctly into two categories: supervised and unsupervised.

2.1. Supervised learning

Supervised learning uses a data set that has associated labels given to each input. That is, the “right answer” is given to the system during learning so the parameters defining the system are adjusted towards this solution.

2.2. Unsupervised learning

Unsupervised learning only needs the empirical data itself to extract some useful information. Since no labels are needed, no human input is required which allows unsupervised learning to use much larger data sets or a continuous stream of input data.

In financial data the concept of supervised learning was used. The financial data is the data which actually change according to time. The data values are taken to find the change percentage. After finding the change percentage first two value are taken as the input and third one as the relative output of input until the end of data. Let "Data" be an array of change percentage.

Input 1	Input 2	Output
Data[1]	Data[2]	Data[3]
Data[2]	Data[3]	Data[4]
Data[3]	Data[4]	Data[5]
Data[4]	Data[5]	Data[6]
Data[5]	Data[6]	Data[7]
Data[6]	Data[7]	Data[8]
Data[7]	Data[8]	Data[9]

Table 1: Layout of Fetched Data

3. Methodology

A typical NN consists of multiple neurons organized in a layered fashion and connected to each other forming an inter-dependent network. There are two types of NNs, feed-forward and recurrent (feedback) NNs [1]. But feed forward neural network has been used. Figure 1 shows a typical 3 layered feed-forward network.

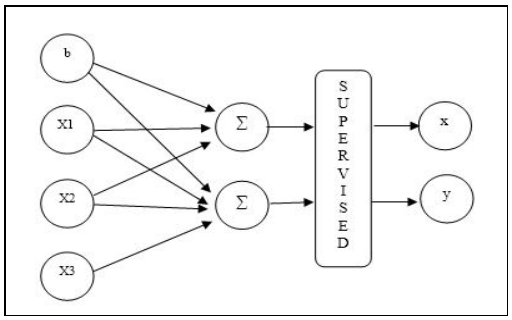


Figure 1: Neural Network Design

Feed forward neural network has three layers: input, hidden and output. A graphical representation of the FFN can be seen in figure 1.

Each layer of the FFN consists of a fixed set of neurons. All neurons in a layer are connected to all neurons in neighbouring layers. Each connection has an associated weight that models the mutual influence between the neurons. Each input vector is associated with bias.

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where

$f(x)=K\left(\sum_i w_i g_i(x)\right)$

Where K(commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions g_i as simply a vector

$g=(g_1,g_2,\ldots,g_n)$.

In this real world example supervised financial data has been used & the resultant data has been taken as the learning agent of non-liner equation.

f(x)= w1x1+w2x2+b

f(x)= w1x1+w2x2+w3x3+b

3.1. Neuron Weight Adjustment

This corrective procedure is called back propagation (hence the name of the neural network) and it is applied continuously and repetitively for each set of inputs and corresponding set of outputs produced in response to the inputs. This procedure continues so long as the individual or total errors in the responses exceed a specified level or until there are no measurable errors. At this point, the neural network has learned the training material and the training process can be stopped and the neural network can be used to produce responses to new input data. Back propagation starts at the output layer with the following equations:

$$w_{ij} = w'_{ij} + LR \cdot e_j \cdot X_i$$

[Eqn 3]

$$e_j = Y_j \cdot (1 - Y_j) \cdot (d_j - Y_j)$$

[Eqn 4]

For the ith input of the jth neuron in the output layer, the weight w_{ij} is adjusted by adding to the previous weight value, w'_{ij}, a term determined by the product of a learning rate, LR, an error term, e_j, and the value of the ith input, X_i. The error term, e_j, for the jth

neuron is determined by the product of the actual output, Y_j, its complement, 1 - Y_j, and the difference between the desired output, d_j, and the actual output.

Once the error terms are computed and weights are adjusted for the output layer, the values are recorded and the next layer back is adjusted. The same weight adjustment process, determined by Equation 3, is followed, but the error term is generated by a slightly modified version of Equation 4.

This modification is:

$$e_j = Y_j \cdot (1 - Y_j) \cdot \sum (e_k \cdot w'_{jk}) \quad \text{[Equ 5]}$$

The learning rate, LR, applies a greater or lesser portion of the respective adjustment to the old weight. If the factor is set to a large value, then the neural network may learn more quickly, but if there is a large variability in the input set then the network may not learn very well or at all. In real terms, setting the learning rate to a large value is analogous to giving a child a spanking, but that is inappropriate and counter-productive to learning if the offense is so simple as forgetting to tie their shoelaces. Usually, it is better to set the factor to a small value and edge it upward if the learning rate seems slow.

4. Graphics Processing Units (GPUS)

Graphics Processing Units (GPUs) have been used as graphic processor engines in gaming industry due to their powerful capability and parallel characteristics. The graphics pipeline is well-suited for parallelism attaining high performances in matrix and vector operations as for instance when dealing with 2D and 3D graphics. Their enormous computational potential has led to an explosion of research to leverage GPUs for general-purpose computation on GPUs (GPGPU). The increase of their programmability featuring floating point arithmetic makes them suitable for data high-demanding real time applications. This is especially important with machine learning algorithms such as neural networks which are often complex, placing high demands on memory and computing resources and CPUs are simply not powerful enough to solve them quickly for use in interactive applications. Example of applications of GPU computing spawn a broad range of areas such as Bioinformatics, Medical Imaging, Linear Algebra and many other. Unlike CPUs which use the paradigm SISD (Single Instruction Single Data), GPUs are optimized to perform floating-point operations (on large data sets) using the paradigm Single Instruction Multiple Data (SIMD). Subsequently, this architectural difference between both platforms leads to more complex programming tasks. To cope with this complexity NVIDIA developed a parallel technology namely CUDA (Compute United Device Architecture) which provides a programming model for its GPUs with an adequate API for non-graphics applications using standard ANSI C, extended with keywords that designate data-parallel functions. The CUDA programming model is supported by an architecture built around a scalable array of multi-threaded Streaming Multiprocessors (SMs), over the past few years not only GPU evolved but also the programming model and programming tools able to allow the development of applications of this nature. The trend is to develop high parallel computers and to concentrate on adding cores rather than increase the capacity of single thread performance ones.

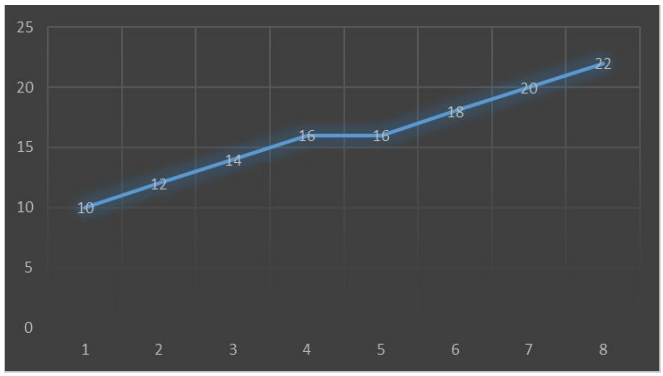
Most neural networks recognition systems have two major components: training and classification. The system is trained using a large number of labelled patterns with relevant features. The training is often iterative and proceeds until the error on a small set of testing patterns is sufficiently low. However, the training process is computational intensive and time consuming especially when a large number of training patterns are involved. The classification accuracy of unknown patterns often depends on the effort spent on training and most applications settle down for a suitable trade-off. Using new training data to improve the performance is uncommon due to the large computational effort. On the contrary, the parallelism of a GPU is fully utilized by accumulating a lot of input feature vectors and weight vectors, then converting the many inner-product operations into one matrix operation.

5. OPENCL And APARAPI

General-Purpose Computing on Graphics Processing Units (GPGPU, rarely GPGP or GP²U) is the utilization of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU)[15]. OpenCL™ is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices. OpenCL (Open Computing Language) greatly improves speed and responsiveness for a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software [13]. Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors

APARAPI is a java application programming interfaces. APARAPI allows Java developers to take advantage of the computing power of GPU and APU devices by executing data parallel code fragments on the GPU rather than being confined to the local CPU. It does this by converting Java byte code to OpenCL at runtime and executing on the GPU, if for any reason APARAPI can't execute on the GPU, it will execute in a Java thread pool [14].

6. Implementation Result



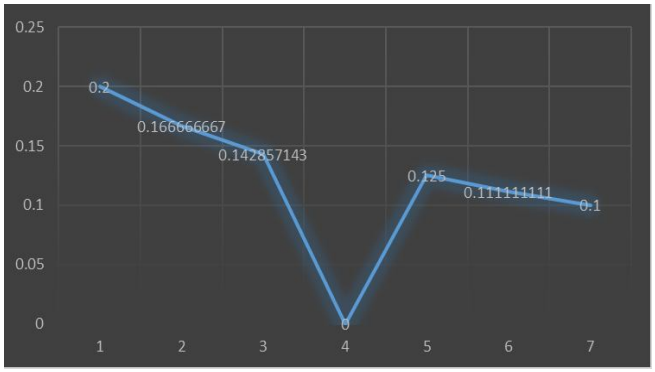


Figure 2: Financial Data Graph

Figure 3: Change in Percentage of Financial Data

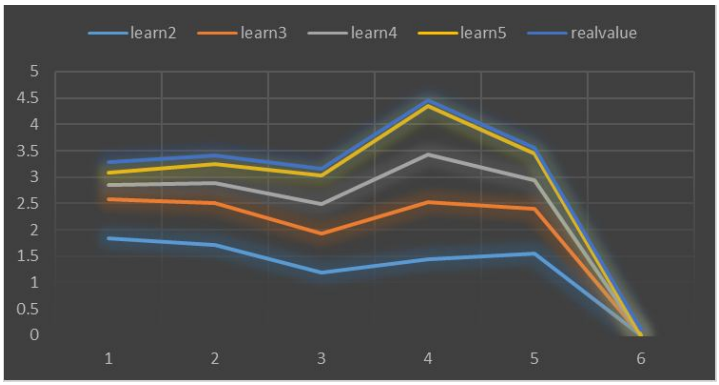


Figure 4: Learning Graph

Learning Rate	0.5
Error	0.1 to -0.1
Execution Time	193 ms
Predictive Change %	0.514008365445363
Final Adjusted Weight	[1.0,-3.2100571943459824,-0.762678127459943]

Table 2: Actual Results

7. Conclusion

Processing a large amount of data puts a lot of load on the CPU and it already has a lot to handle but if we utilize the cores of GPU, it can be done quite easily and in a faster manner. OpenCL & CUDA were used earlier but they are not easy to learn, whereas using the APARAPI in JAVA, the simple JAVA code is converged to OpenCL code at runtime hence making it quite easy to program. It even makes the execution possible through the CPU just in case GPU is not available by using JAVA thread pool hence making it as a more reliable option for the programming and execution of the task. The results of financial prediction of data through this technique of implementation reflect how efficiently and faster are the processing power of GPU as compared to the techniques used earlier which just involved the CPUs.

8. References

1. Lasfer, A.; El-Baz, H.; Zualkernan, I., "Neural Network design parameters for forecasting financial time series," Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on , vol., no., pp.1,4, 28-30 April 2013
2. Yong Liu; Yeming Xiao; Li Wang; Jielin Pan; Yonghong Yan, "Parallel implementation of neural networks training on graphic processing unit," Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference on , vol., no., pp.1571,1574, 16-18 Oct. 2012
3. Bako, L.; Kolcsar, A.; Brassai, S.; Marton, L.; Losonczi, L., "Neuromorphic Neural Network Parallelization on CUDA Compatible GPU for EEG Signal Classification," Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on , vol., no., pp.359,364, 14-16 Nov. 2012
4. Arce, P.; Maureira, C.; Bonvallet, R.; Fernández, C., "Forecasting High Frequency Financial Time Series Using Parallel FFN with CUDA and ZeroMQ," Information and Telecommunication Technologies (APSITT), 2012 9th Asia-Pacific Symposium on , vol., no., pp.1,5, 5-9 Nov. 2012

5. Nabiyouni, M.; Aghamirzaie, D., "A Highly Parallel Multi-class Pattern Classification on GPU," Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on , vol., no., pp.148,155, 13-16 May 2012
6. Ribeiro, B.; Lopes, N.; Silva, C., "High-performance bankruptcy prediction model using Graphics Processing Units," Neural Networks (IJCNN), The 2010 International Joint Conference on , vol., no., pp.1,7, 18-23 July 2010
7. Shian-Chang Huang; Tung-Kuang Wu, "Credit quality assessments using manifold based semi-supervised discriminant analysis and support vector machines," Natural Computation (ICNC), 2011 Seventh International Conference on , vol.4, no., pp.2037,2041, 26-28 July 2011
8. Behbood, V.; Jie Lu; Guangquan Zhang, "Fuzzy Refinement Domain Adaptation for Long Term Prediction in Banking Ecosystem," Industrial Informatics, IEEE Transactions on , vol.10, no.2, pp.1637,1646, May 2014
9. Mager, J.; Paasche, U.; Sick, B., "Forecasting financial time series with support vector machines based on dynamic kernels," Soft Computing in Industrial Applications, 2008. SMCia '08. IEEE Conference on , vol., no., pp.252,257, 25-27 June 2008
10. Behbood, V.; Jie Lu; Guangquan Zhang, "Long term bank failure prediction using Fuzzy Refinement-based Transductive Transfer learning," Fuzzy Systems (FUZZ), 2011 IEEE International Conference on , vol., no., pp.2676,2683, 27-30 June 2011
11. de Souza, L.V.; Pozo, A. T R; da Rosa, J.M.C.; Neto, A.C., "The boosting technique using correlation coefficient to improve time series forecasting accuracy," Evolutionary Computation, 2007. CEC 2007. IEEE Congress on , vol., no., pp.1288,1295, 25-28 Sept. 2007
12. Jie Du; Rada, R., "Knowledge-guided genetic algorithm for financial forecasting," Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on , vol., no., pp.1,8, 29-30 March 2012
13. http://www.khronos.org/opencl
14. http://code.google.com/p/aparapi/
15. http://www.wikipedia.com