Procedia Computer Science

Volume 53, 2015, Pages 327–334

2015 INNS Conference on Big Data

A MapReduce Cortical Algorithms Implementation for Unsupervised Learning of Big Data

Nadine Hajj[1], Yara Rizk[2], and Mariette Awad[3]

[1] American University of Beirut, Beirut, Lebanon
njh05@aub.edu.lb
[2] American University of Beirut, Beirut, Lebanon
yar01@aub.edu.lb
[3] American University of Beirut, Beirut, Lebanon
mariette.awad@aub.edu.lb

Abstract

In the big data era, the need for fast robust machine learning techniques is rapidly increasing. Deep network architectures such as cortical algorithms are challenged by big data problems which result in lengthy and complex training. In this paper, we present a distributed cortical algorithm implementation for the unsupervised learning of big data based on a combined node-data parallelization scheme. A data sparsity measure is used to divide the data before distributing the columns in the network over many computing nodes based on the MapReduce framework. Experimental results on multiple datasets showed an average speedup of 8.1× compared to serial implementations.

Keywords: MapReduce, Cortical Algorithms, Big Data, Unsupervised Learning

1 Introduction

The exponential growth in today's data sources exposed traditional machine learning (ML) techniques to poor scalability, loss in robustness and redundancy. A powerful algorithm capable of extracting hidden structures from large datasets in hence a necessity.

Cortical algorithms (CA) are a biologically inspired model proposed in [1] and further developed in [6], [8],[7], trained in a supervised fashion, and have achieved superior generalization on a wide array of problems [7, 2]. However, they suffer from expensive training, limiting their usage to relatively small, labeled datasets. To leverage CA's advantages while improving scalability, a parallel CA implementation for unsupervised learning of big data is proposed based on a combined node and data parallelism approach using Google's MapReduce framework. Although a previous supervised parallel CA implementation has been proposed in [12], achieving up to 48× speedup on GPUs (668 cores), the required hardware resources are enormous. Instead, the proposed approach implements an unsupervised CA training algorithm to cluster data in a MapReduce framework on Matlab and run on limited hardware resources.

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.310

Next, a survey of neural network (NN) architectures for big data learning is presented in Section 2. Section 3 details the proposed clustering and distributed implementation. Section 4 presents the experimental results before concluding in Section 5.

2 Literature Review

The distributed implementation of ML algorithms has been widely proposed for big data environments. Node or data parallelization of NN architectures are commonly used. The former distributes the network over computing nodes, exploiting existing parallelism in the algorithm or resolving variable dependency [15],[3]. Such approaches are more efficient when dealing with complex network structure but suffer with big data since all the training data must be available to all nodes [4]. The latter distributes the training data over nodes to update the network parameters in parallel but deploys the entire network on all nodes [10], reducing the efficiency of systems with complex networks such as CA.

The only work on parallel CA implementation [12], to the best of our knowledge, form a cluster of GPUs and map each column to a CUDA-thread and each hypercolumn to a thread block or Cooperative Thread-Array (CTA). 14 streaming multiprocessors (448 cores) and 128 thread per CTA implementation lead to approximately 23× speedup compared to a single threaded CPU C++ implementation on an Intel i7 processor for the supervised learning of the MNIST database. Pipelining the propagation of activation signals between consecutive layers and using a heterogeneous GPGPU configuration, an association of Intel i7 processor, a GTX 280 GPU and a Fermi C2050 GPU (688 cores) resulted in 48× speedup.

3 Proposed Solution

In this section, we present a short primer on CA before detailing our proposed approach to adapt CA for unsupervised learning on big data problems using a MapReduce based parallel implementation.

3.1 Background information

CA are six-layered structures, composed of cortical columns (a group of neurons sharing the same input) and connected through vertical connections between columns of consecutive layers and horizontal inhibiting connections between columns within the same layer. Training CA includes random initialization of weights, an unsupervised feed-forward training phase to identify particular unique features of the input data through random firing and repeated exposure and a supervised feedback phase to correct pattern misclassification. A detailed description and mathematical formulation of the model can be found in [5].

3.2 Unsupervised clustering using CA

The proposed method uses hierarchical representations generated by CA manifested in the comparable firing schemes observed at the output of the network for similar patterns. Our clustering algorithm is as follows:

1. Random initialization: the weights of the network are randomly initialized to small values.

2. Feedforward learning and initial clustering: for every training data instance, the weights of the firing columns are strengthened while inhibiting non-firing columns. The first instance is assigned to a random cluster. Subsequent data points are assigned to a new or the closest centroid, based on the network output.

3. Weight fine tuning and cluster verification: if the distance separating two seeds is less than a predefined threshold, the corresponding clusters are merged. Conversely, if the maximum within-cluster distance (the distance separating the farthest two points in a cluster) is greater than a threshold, the cluster is split into two with each point being assigned to the nearest cluster centroid. Steps 2 and 3 are repeated until convergence of the objective function given in (1),where X denotes an instance of the input data belonging to a cluster C_i of centroid s_i, K the number of clusters, and $d(a, b)$ the distance between points a and b..

$$E_K = \frac{min_{k \neq h}(s_k, s_h)}{max_k max_{X_{i,j} \epsilon C_k} d(X_i, X_j)} \tag{1}$$

3.3 Distributed CA

To accelerate training, a combined node-data parallelization approach splits the training data over an association of computing nodes, and divides the network over these nodes. Using this architecture, a trade-off between overhead and efficiency is attained, by avoiding the storage of the entire set over all nodes and by distributing the network over parallel computing units. Assuming the total number of workers is $L = N \times M$, our approach splits the data into N subsets processed by N arrays of M CA sub-networks.

Instead of randomly distributing the data,we propose to divide the data into equal blocks along the dimension with the smallest Gini index, a normalized measure of spreadness [17].In essence, a feature with a large index has all its instances concentrated in one component, conversely a feature with a small index has its instances spread over the range of values. With this data distribution method, we assume that the instances are spread evenly across the chosen dimension; hence, the resultant distribution has a balanced aspect guaranteeing comparable execution times for parallel computing nodes. Given a feature vector of length n, denoted by $x = [x_1, \ldots, x_i, \ldots, x_n]$, Gini's index is computed in (2).

$$G(x) = 1 - \frac{2}{||x||_1} \sum_{i=1}^{n} x_i \left(\frac{n - i + 0.5}{n} \right) \tag{2}$$

Next, CA networks are distributed over multiple nodes, by parallelizing the computations of the cortical columns within a layer given the output of the previous layer and the concerned column's current stateusing a cascade of Map/Reduce phases as shown in Figure 1, were used to parallelize the feedforward training phase. Columns in a layer are mapped to a map function; its input is the output of the previous layer, and its output is the column output that will be propagated to the reduce phase. Columns are divided into blocks and assigned to a worker with neighbouring columns placed in the same block since they share the same input. Columns in a layer are executed in parallel, and layers are updated sequentially.

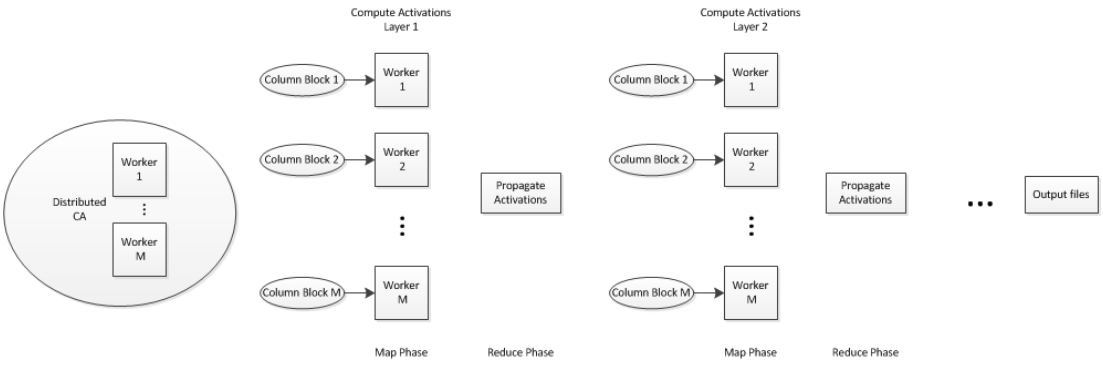

Figure 1: Distributed CA

4 Experimental Results

The proposed technique is tested using different experimental setups, detailed below, on a collection of databases summarized in table 1. For each experiment, three distance measures are used, (a) Euclidean (b) absolute value and (c) KL divergence.

1. Data parallelization: The data is split according to Gini's index; the entire network is deployed on each of the individual workers.
2. Node parallelization: With the entire data made available to all workers, the network is distributed among computing nodes
3. Data-Node parallelization: The proposed combined scheme is implemented.
4. Random Data-Node parallelization: A combined data-node parallelization approach is adopted with the data subsets randomly distributed.

The adopted CA network has 6 layers, with 1000 20-node columns in the first layer and reducing this number by half between consecutive layers. Table 2 summarizes the speedup obtained relative to a single threaded Matlab implementation. The parallel implementation is deployed on a cluster of 4 Intel Xeon E5-2603 @ 1.8 GHz 4-core CPUs with 94 GB of on board memory run on Red Hat Enterprise Linux Server. In experiments 3 and 4, the data is divided into 4 subsets, each processed by a processor; the network is distributed over 4 cores.

4.1 Speedup evaluation

The best speedup was obtained by the proposed algorithm 10.2× compared to 8.5× for data parallelization, 7.1× for node parallelization and 8.5× for random data-node parallelization. Balanced data cuts increased the speedup from 6.6× for random cuts to 8.1×. The additional preprocessing to obtain the data splits was compensated by the balanced aspects of the load; random sampling resulted in decreased performance due to the unbalanced subsets.

Looking at the effect of the distance measure, the Euclidean distance is the most computationally expensive and resulted in the least speedup, whereas the absolute value measure achieved the highest speedup. However, the Euclidean distance has superior clustering performance compared to the absolute value measure due to the penalization of large within-cluster

Table 1: Database Description

Dataset	Number of Attributes	Number of Instances	Size
Japanese Vowel [14]	12	640	1.2 MB
Dorothea [11]	100,000	1,950	10 MB
MNIST [9]	784	70,000	11 MB
Microsoft Anonymous Web Data [14]	294	37,711	1.6 MB
Forest Cover Type [14]	54	581,012	75.2 MB
YouTube Multiview Video Games [11]	3,231,961	2,396,130	234 MB
Bag of Words [11]	100,000	8,000,000	1.9 GB
Higgs [11]	28	11,000,000	2.6 GB

Table 2: Speedup Comparison

Experiment		URL Reputation	Bag of Words	YouTube Multiview Video Games	MNIST
1	a	6.7	5.2	5.9	6.5
	b	8.5	7.7	7.8	8.1
	c	6.2	7.7	6.7	7
2	a	4.2	4.9	3.9	5
	b	6.2	6.2	5.9	7.1
	c	5	5.3	4.9	6.1
3	a	5.7	6.7	7.2	6.8
	b	9.8	10.1	9	10.2
	c	6.3	8.9	8	8.6
4	a	5.7	7.5	4.6	8.5
	b	6.4	8.3	7.5	6.5
	c	5	5.8	5.3	8.5

distances resulting in more compact clusters. Although the KL divergence achieves a lower speedup than the absolute value distance, it attains a better clustering performance; this is due to the fact that the KL divergence is an information metric indicating the degree of representation of the centroid to the corresponding cluster.

4.2 Clustering evaluation

The performance of our proposed clustering algorithm is evaluated by reporting on the value of the objective function obtained using the settings of experiment 3 of the previous section in addition to the modularity and silhouette index, two well known metrics for evaluating clustering performance [13], [16]. Results are summarized in Table 3; the objective function values were normalized between 0 and 1, for ease of comparison. Furthermore, the recognition rate on three labeled datasets, obtained from the UCI KDD archive [14], are summarized in Table 4 and show the merit of the proposed clustering algorithm, achieving recognition rates between 87.1 and 90.1%.

4.3 Effect of architecture

The effect of the choice of the number of data splits N and the number of workers per CA implementation M is studied on three datasets from the UCI ML repository [11]. The speedup

Table 3: Performance comparison

Performance Measure	Distance	URL Reputation	Bag of words	Youtube multiview	MNIST
Objective Function	Euclidean	0.62	0.65	0.58	0.99
	Absolute Value	0.41	0.57	0.39	0.78
	KL Divergence	0.83	0.94	0.8	0.96
Modularity Index	Euclidean	0.78	0.56	0.64	0.93
	Absolute Value	0.63	0.47	0.55	0.91
	KL Divergence	0.89	0.86	0.78	0.92
Silhouette Index	Euclidean	0.61	0.74	0.57	0.88
	Absolute Value	0.54	0.66	0.52	0.85
	KL Divergence	0.73	0.82	0.68	0.91

Table 4: Recognition rates (%) when using three different distance measures

Dataset	Euclidean	Absolute value	KL Divergence
Forest Cover Type	85.4	83.2	87.1
Japanese Vowel	89.6	85.8	86.7
Microsoft Anonymous Web Data	90.1	88.3	89.8

obtained compared to a single threaded implementation are shown in Table 5 for values of $N\epsilon[1, 16]$. Increasing the number of data splits results in an increase in the speedup up to a value of $N\epsilon[4, 8]$, after which the overhead of the reads and writes to memory becomes more expensive than the gain from computational parallelization. In the performed experiments, the total number of workers is fixed, equal to 16. Node parallelism is performed when $N = 1$ and data parallelism is performed when $N = 16$; points in between correspond to different combinations of (N, M) with $L = N \times M = 16$. The drop in performance observed after a value of $N = 8$ is expected as higher values suffer from the limitations of data parallelism, i.e. deploying the entire network on one worker or a small number of workers does not perform well on complex networks like CA.

5 Conclusion

In this paper, we presented a novel distributed implementation of CA for clustering problems based on a combined data-node parallelization approach using the MapReduce framework. Our parallelization algorithm divides the data across computing nodes using the Gini index and divides the network over the nodes as well. Experimental results on multiple publicly available datasets showed that the proposed method performs better than a node or data division strategy alone and achieves a speedup of up to 8.1× compared to a single threaded implementation.

Acknowledgments

This work was partly funded by the University Research Board Grant at the American University of Beirut.

Table 5: Architecture speedup comparison

(N, M)	Database	Distance Measure			Average
		Euclidean	Absolute value	KL Divergence	
$(1, 16)$	Higgs	4.5	6.2	5.3	5.3
	Bag of Words	7.5	8.3	5.8	7.2
	Dorothea	7.6	8.5	6	7.4
	Average	6.5	7.7	5.7	6.6
$(2, 8)$	Higgs	5.5	7.6	7.1	6.7
	Bag of Words	7.9	8.3	6.3	7.5
	Dorothea	7.9	8.9	6.4	7.7
	Average	7.1	8.3	6.6	7.3
$(4, 4)$	Higgs	6.6	9.7	8.2	8.2
	Bag of Words	6.7	10.1	8.9	8.6
	Dorothea	6.8	10.2	8.9	8.6
	Average	6.7	10	8.7	8.5
$(8, 2)$	Higgs	6.3	8.6	7.5	7.5
	Bag of Words	7.2	10.9	9.6	9.2
	Dorothea	7.2	10.9	9.1	9.1
	Average	6.9	10.1	8.7	8.6
$(16, 1)$	Higgs	6.1	7.5	6.4	6.7
	Bag of Words	5.2	7.7	7.7	6.9
	Dorothea	7.2	10.8	9	9
	Average	6.2	8.7	7.7	7.5

References

[1] Gerald M Edelman and Vernon B Mountcastle. *The mindful brain: Cortical organization and the group-selective theory of higher brain function*. Massachusetts Institute of Technology Pr, 1978.

[2] Elia Formisano, Federico De Martino, Milene Bonte, and Rainer Goebel. " who" is saying" what"? brain-based decoding of human voice and speech. *Science*, 322(5903):970–973, 2008.

[3] K Ganeshamoorthy and DN Ranasinghe. On the performance of parallel neural network implementations on distributed memory architectures. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th International Symposium on*, pages 90–97. IEEE, 2008.

[4] Rong Gu, Furao Shen, and Yihua Huang. A parallel computing platform for training large scale neural networks. In *Big Data, 2013 IEEE International Conference on*, pages 376–384. IEEE, 2013.

[5] Nadine Hajj and Mariette Awad. Weighted entropy cortical algorithms for isolated arabic speech recognition. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013.

[6] Atif Hashmi and Mikko H Lipasti. Discovering cortical algorithms. In *IJCCI (ICFC-ICNC)*, pages 196–204, 2010.

[7] Atif Hashmi, Andrew Nere, James Jamal Thomas, and Mikko Lipasti. A case for neuromorphic ISAs. *ACM SIGPLAN Notices*, 47(4):145–158, 2012.

[8] Atif G Hashmi and Mikko H Lipasti. Cortical columns: Building blocks for intelligent systems. In *Computational Intelligence for Multimedia Signal and Vision Processing, 2009. CIMSVP'09. Symposium on*, pages 21–28. IEEE, 2009.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied

to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[11] M. Lichman. UCI machine learning repository, 2013.

[12] Andrew Nere, Atif Hashmi, and Mikko Lipasti. Profiling heterogeneous multi-GPU systems to accelerate cortically inspired learning algorithms. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 International*, pages 906–920. IEEE, 2011.

[13] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[14] University of California Irvine. UCI KDD archive, 1999.

[15] Mark Pethick, Michael Liddle, Paul Werstein, and Zhiyi Huang. Parallelization of a backpropagation neural network on a cluster computer. In *International conference on parallel and distributed computing and systems (PDCS 2003)*, 2003.

[16] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, et al. *Introduction to data mining*, volume 1. Pearson Addison Wesley Boston, 2006.

[17] Dornoosh Zonoobi, Ashraf A Kassim, and Yedatore V Venkatesh. Gini index as sparsity measure for signal reconstruction from compressive samples. *Selected Topics in Signal Processing, IEEE Journal of*, 5(5):927–932, 2011.