Large-scale Artificial Neural Network: MapReduce-based Deep Learning

Kairan Sun, Xu Wei, Gengtao Jia, Risheng Wang, and Ruizhi Li
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida 32611

Abstract—Faced with continuously increasing scale of data, original back-propagation neural network based machine learning algorithm presents two non-trivial challenges: huge amount of data makes it difficult to maintain both efficiency and accuracy; redundant data aggravates the system workload. This project is mainly focused on the solution to the issues above, combining deep learning algorithm with cloud computing platform to deal with large-scale data. A MapReduce-based handwriting character recognizer will be designed in this project to verify the efficiency improvement this mechanism will achieve on training handwritten digits. Careful discussion and experiment will be developed to illustrate how deep learning algorithm works to train handwritten digits data, how MapReduce is implemented on deep learning neural network, and why this combination accelerates computation. Besides performance, the scalability and robustness will be mentioned in this report as well. Our system comes with two demonstration software that visually illustrates our handwritten digit recognition/encoding application.

Index Terms—Neural Network, MapReduce, Machine Learning, Deep learning.

I. INTRODUCTION

Classification of data patterns generated by handwriting characters, photograph pixels, audio signals has long been a popular topic in machine learning field. We used to take advantages of the non-volatile feature of computer memory so as to create huge database to afford data searching and pattern matching mechanism. However, we are expecting machines behave more closely to human beings and instead of merely matching mechanism. However, the process described above cannot be implemented on any individual computation node because with tremendously increasing number of data on Internet, the incredible complexity of computation would lead to unbounded execution time, thus in turn making the whole mechanism infeasible. Despite the performance, few disks can afford that enormous amount of data, which is usually hundreds of
gigabytes or even terabytes. On the other hand, conventional
data mining algorithms for classification are not suitable for
cloud computing algorithms either since the weight processing of
each layer in back-propagation neural network are dependent
on other layers, and in this way, MapReduce is helpless when
dividing the entire back-propagation algorithm into mappers
and reducers. This problem is carefully reviewed in [2].
Even though supplied with cluster computing resources, the
iteration cycle keeps the same and it can hardly improve the
performance of ANN.

In our project, we implement the latest achievement of
neural network algorithm, deep learning [4], to accelerate the
performance of back-propagation neural network. Inspiration
comes from the progress we learn to recognize objects. Before
studying an unfamiliar object, we are already informed its
shape: whether it is a line, a circle, a triangle, or a rectangle
and so on. Instead of learning a totally strange object, we learn
the representation of the combination of several recognized
objects. The fundamental concept of deep learning is that an
observation can be represented in multiple ways, but certain
representations make it easier to learn tasks of interest from
examples. A many-layered neural network could be effectively
pre-trained one layer at a time, treating each layer in turn as
an unsupervised Restricted Boltzmann Machine, then followed
by supervised back-propagation fine-tuning.

Integrating back-propagation in non-linear deep learning
network, our cluster-computing framework is built on top of
Amazon Web Service. The aim is quite simple: distribute the
complex, large-scale computation to clusters of computers,
ensuring parallelized cluster computing.

II. RELATED WORK

Of course, we are not the first ones who tried to combine
parallel methods of neural network with the cluster computing.
Most of existing work achieves parallelization by mapping
the nodes in network to the nodes in computing cluster,
like [5]–[7]. However, these algorithms are not suitable for
MapReduce. In MapReduce structure, users cannot specifically
control a certain node in a cluster. Instead, MapReduce can
only assign the mapper’s and reducers’ jobs as a whole, so
the algorithms above are not implementable in this cloud
computing environment. Moreover, the scale of these work is
relatively small, and lack of elasticity. Mapping the network
nodes to different computing nodes will inevitably increase the
I/O cost. In most cases, neural network problem are dealing
with big data, which requires large amount of I/O operations.
As the result, I/O cost is the major cost in the distributed
computing environment of existing methods.

Chu et al. [8] and Liu et al. [9] both proposed their own
algorithm that used back-propagation algorithm to train a
three-layer neural network based on MapReduce. They did
the supervised learning (which is the only kind of learning
back-propagation can do) to classify the input data into two
categories and did some experiments on multi-core environ-
ment. However, They did not adopt the latest neural network
achievement, deep learning, and therefore still suffered from
the problem of low learning efficiency.

III. SYSTEM ARCHITECTURE

A. Main Strategy

In detail, we implement deep learning algorithm [4] to train
the input data, where a MapReduce programming model is
made use of to parallelize the computation. The MapReduce
job consists of the mapper and reducer functions, of which
the mapper function will extract key/value pairs from input
data and transfer them into a list of intermediate key/value
pairs, and reducer function merges these intermediate values
corresponding to the same key generated from mapper function
to produce output values. In the case of Machine Learning,
the input value will be the data from a certain object that a
machine is going to “learn”, and in our project, the objects are
data sets extracted from handwriting characters. The interme-
diate key/value pair will be the weights, in order for a machine
to determine whether it has acknowledged the object correctly.
Reducer function uses these weights to compute the so-called
“acknowledgment” of the machine of an intended object [10].
If there exits an intolerable difference between the precision
of training set and expected precision, the MapReduce job will
loop until an acceptable result has worked out.

1) Difficulties: As mentioned before, the main problem
is the huge amount of data waiting to be trained. Although
deep learning learns from representation, it is not capable of
eliminating similarity and noisiness contained in data sets, and
this will terribly affect machine learning [11]. Thus the most
straightforward consequence is that the MapReduce job would
loop many times so that it is hard to provide a satisfying
precision, or even it cannot produce an output. It is necessary,
however, to implement a mechanism to somehow improve
the efficiency of the procedure of neural network. One idea
is to remove the similar items by using diversity-based data
sampling method. It can be applied to MapReduce program-
ing model, too, where the frequency of input data will be
counted, and those duplicated data is eliminated. Another way

![Fig. 1. Architecture of ANN [3].](image)
is to somehow train the data so that the machine would find a certain pattern corresponded with the data set. Both are applicable to develop the efficiency of back-propagation.

B. Modules and Subsystems

Our system is made up of three different parts, the Deep learning implemented by Java, the Cloud Computing by Hadoop, and Demonstration software implanted by Matlab. Such design can be well illustrated in figure 2.

Fig. 2. Modules and Subsystems of the Design.

IV. DEEP LEARNING

A. Pre-training

The motivation to insert pre-training step before fine-tuning is the unacceptable inefficiency of back propagation when converting a high-dimensional data into low-dimensional codes. When we have multiple layers, large initial weights always witnesses poor local minima, while small initial weights will lead to tiny gradients in early layers, making it infeasible to train the whole back propagation system with many hidden layers [12].

Therefore, we have to make the initial weights close to the solution. The idea is simple. Think about two pictures of number zero and number one. Instead of jumping to the final step of recognizing the number, we let machine accomplish the task step by step, where it first looks for the internal pattern of two pictures, namely, recognizing a circle and a stick, then encodes the circle and the stick to zero and one respectively. Practically, however, it is not easy because this process requires a very different type of algorithm that learns one layer of features at a time.

We make advantage of restricted Boltzmann machine (RBM) to realize pre-training, where an ensemble of binary vectors can be modelled using a two-layer network, in which stochastic, binary pixels are connected to stochastic, binary feature detectors using symmetrically weighted connections. Pre-training involves learning a stack of RBMs, each of which has one layer of feature detectors, and it is sort of recursive process where the learned feature activations of one RBM are used as “data” for training the next RBM in the stack. For each RBM, we are not going to evaluate the weights between two layers for infinite times in order for the efficiency. Instead we get the modified weights result from two rounds of RBM calculation, and the result of this recursive execution is unrolled to create a deep auto encoder. Note that this does not hurt the precision quite much because of the mathematics proof by Geoffrey E. Hinton. Thanks to the largely increased number of machines involved in, the time cost for training will decrease in inverse proportion, too. Figure 3 give us a directly illustration of what RBM is.

One more point to be noticed is that we are adjusting the weights of paths based on the average variations of a single weight resulting from the modification made by a batch of training items. The reason is updating the value of weights once a single training item is extremely slow and inapplicable for MapReduce method.

B. Fine-tuning

This is the step where we get zero and one from a circle and a stick. Pre-training initializes the weights so that they are close to the solution we want, but they are not the answer. We train the weights using back propagation [13] algorithm based on MapReduce method in multiple layer neural network [14]. Back propagation and its improved methods are applied in mobile data processing. Such applications can be found in [15]–[17]. Note that here the trained data is not randomized weights but well-initialized weights, making it easier to get the solution. Every mapper receives one training item and then computes all update value of the weights. Then each reducer gathers update-values for one weight and calculating the average. The input value of mapper is the input item while the input key is empty. On the other hand, the value of the reducer is the difference between original weight and updated weight, and the corresponding key is the weight. Similar to pre-training, variations of each weight should be updated batch by batch for the sake of efficiency.

Fig. 3. Algorithm Flow Char for Restricted Boltzmann Machine (RBM).
C. Precision Refinement

This step is still in discussion. We are going to use Adaboosting method to refine the result of data training. Basically, a neural network model after training is regarded as a classifier. Unfortunately, the precision of a classifier is vulnerable to the noisy and the large-scale of the mobile data. Such learning is called weak learning, and we need to improve the performance of the classifier.

The general concepts of the Adaboosting method can be summarized as: get one weak classifier from part of the training set; get more using different parts of the training set sampled out by the features of the former one; assemble them. This is just the basis and we are not going to talk much about the Adaboosting method here because we do not have a good idea about it now. Our main focus at present is implementing the pre-training step, using deep learning, and realizing the MapReduce based back-propagation algorithm.

V. Cloud Computing Based on MapReduce

A. Choice of Cloud Computing Platform

One of the most severe disadvantages of deep learning is the long training time. Such time-consuming problem prohibit trained machine from quickly equipped with accurate neural network data and accomplish required task. To solve such problem, applying cloud computing algorithm to machine learning is a good idea.

Among all the MapReduce platform, Hadoop [19] is a good choice for our project. It is an open source software for data storage as well as large scale processing of data-sets on clusters of commodity hardware [20]. The main modules that compose Hadoop framework are Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop Yarn, and Hadoop MapReduce [21]. Hadoop Common contains all the utilities and libraries that required by all the other Hadoop modules. HDFS is a distributed file-system that containing data on the clustering machines. It can provide highly aggregated bandwidth for both masters and slaves across the commodity. Hadoop Yarn is a resource-management platform. It is responsible for arranging computing resources in commodity through which the users’ applications are scheduled. Hadoop MapReduce is the most important part here. It provides a programming model for large scale data.

In this project, we use Amazon Web Service EC2 platform to achieve our MapReduce algorithm. It is a collection of remote computing services that provides a cloud computing platform. Based on a physical srver farm, it can provides customer faster and cheaper large computing capacity.

B. MapReduce Structure Design

In order to implement MapReduce to RBM, we apply an algorithm that can be well illustrated in figure. For the sake of easy calculation, all the weights coming from paths between nodes are allocated in a matrix. For each mapper task, one training item is sent to a mapper. Their outputs are matrices of variables of weight resulted through the RBM algorithm. A unique ID can identify every element in the output matrix of mapping tasks, which is considered as the key of the reducing task. For each reducer task, a reducer accepts the ID and the value of a certain element of the matrix as key value pair. Since each reducer will only receive variable value of one particular ID, it can sum up all the results provided by each training item and find the final update of that element.

C. Pseudo-code for Algorithm

Our MapReduce-based deep learning algorithm contains 6 Java classes. A MapReduce driver class (DeepLearningDriver.java), two mappers and two reducers for RBM training and forward propagation tasks (RBMMapper.java, RBMReducer.java, PropMapper.java, and PropReducer.java), and a class that implements all the matrix operations (Matrix.java).

We present our algorithms using Pseudo-codes listed below. First, Algorithm 1 describes the driver of deep learning MapReduce program. It contains two MapReduce structures: RBM and forward propagation. It is the scheduler for the MapReduce tasks.

Algorithm 2 describes the mapper of restricted Boltzmann machine (RBM) training part in deep learning MapReduce program. Each mapper only trains the weights for one iteration using one test case. So, in order to train the weights of the whole neural network, the MapReduce program needs to execute \((maxEpoch \times numLayers)\) times. It contains six parts: configure() reads all the configurations and distributed cache from outside; initialize() parse the input strings into parameters, and initialize parameters for algorithm; getpositive() does the positive phase of RBM training; getnegative() does the negative phase of RBM training; update() computes the update of weights using previous results and parameters; map() implements the mapper. It outputs the original key and updated value pair as the intermediate data. In the pseudo-
Algorithm 1 The Driver for MapReduce-based Deep Learning algorithm

Initialization:
The user provides the input file location, output file location, max iteration number (maxEpoch), number of layers (numLayer), number of nodes in each layer (numNodes[layer]) via input arguments.

Iteration:
1. for layer ← 1 to numLayers − 1 do
2.   numVis ← numNodes[layer − 1];
3.   numHid ← numNodes[layer];
4.   Weights[layer] ← RandomizeWeights();
5.   iter ← 1;
6.   for iter ≤ Max Iteration Times do
7.     JobRBM ← Initialized MapReduce Job for RBM;
8.     Store Weights(layer) into → file system: FS;
9.     Give config data: numVis → JobRBM;
10.    Give config data: numHid → JobRBM;
11.   Assign Distributed Cache: FS → JobRBM;
// Each Mapper has a full copy of weights
12.   Start JobRBM :
13.   input: the network input for current layer;
   // (# of Mappers: num of cases)
14.   output: Weights_update ← update for every weight;
   // (# of Reducers: numVis × numHid)
15.   Update Weights(layer);
16.   for all weight ∈ Weights(layer) do
17.     weight ← weight + weight_update;
18.   end for
19.   iter++;  
20. end for
21. Jobprop ← Initialized MapReduce Job for propagate;
22. Store Weights(current) into → file system: FS;
23. Give config data: numVis → Jobprop;
24. Give config data: numHid → Jobprop;
25. Assign Distributed Cache: FS → Jobprop;
26. Start Jobprop :
27. input: the network input for current layer;
28. output: the network output for current layer;
29. end for
30. return
The final Weights(last_layer) is the trained result we want.

Algorithm 2 The mapper of restricted Boltzmann machine (RBM) training part in deep learning MapReduce program.

Initialization:
Input of the mapper is one training case from network input. Also, there are arguments like numVis, numHid and Weight(current) past in via configurations or distributed cache.

Iteration:
1: initialize();
2: getposphase();
3: getnegphase();
4: update();
5: for i ← 0 to numVis − 1 do
6: for j ← 0 to numHid − 1 do
7: output (key, value) pair:
   ⟨Weight_{ID}, Weight_{Update}⟩
8: end for
9: end for
10: return
Each mapper outputs its update of Weights() according to its training case.

Algorithm 3 The reducer of restricted Boltzmann machine (RBM) training part in deep learning MapReduce program.

Initialization:
Input of the reducer is the intermediate data output by the mappers from the same weight ID.

Iteration:
1: sum ← 0;
2: for all Weight_update ∈ same Weight_{ID} do
3:   sum ← sum + Weight_{update};
4: end for
5: output (key, value) pair:
   ⟨Weight_{ID}, sum⟩
6: return
Each reducer outputs the overall update of Weights().

this MapReduce program is numLayer − 1. It contains four parts: configure() reads all the configurations and distributed cache from outside; initialize() parses the input strings into parameters, and initializes parameters for algorithm; prop2nextLayer() computes the forward propagation algorithm; map() implements the mapper. It outputs the original key and updated value pair. We also skip the algorithm details here, and just focus on the structure.

The reducer of forward propagation part is just output the intermediate data as final output, so it is omitted.

VI. PERFORMANCE AND RESULTS

In the performance evaluation section, extensive experiments have been conducted. The experimental results prove the efficiency, and scalability of our proposed MapReduce on neural network method. This method can be applied over large-scale realistic data on the cloud-computing platform of AWS.
Algorithm 4: The mapper of forward propagation part in deep learning MapReduce program.

Initialization:
Input of the mapper is one training case from network input. Also, there are arguments like numVis, numHid and Weight(current) past in via configurations or distributed cache.

Iteration:
1: initialize();
2: prop2nextLayer();
3: Initialize string update ← ""
4: for i ← 0 to numHid − 1 do
5:   update ← update + Caseupdate + " ";
6: end for
7: output ⟨key, value⟩ pair:
   ⟨CaseID, update⟩
8: return

Each mapper output the forward-propagated training case Caseupdate.

Because the goals of our project consist of two parts: deep learning on neural network and MapReduce-based parallelized computing, three sets of experiments have been conducted: objective performance evaluation of deep learning algorithm, experiments on speed-up gained from MapReduce, and subjective performance evaluation of demo applications.

A. Performance Experiment Set-up

We use the hand-written digit training and testing cases from the on-line database available in [22]. It has a training set of 60,000 examples, and a test set of 10,000 examples. And all of them are labeled.

The AWS cloud-computing platform is built up by multiple EC2 instances. Up to 32 EC2 nodes have been used in these experiments for comparing the performance of experiments in different running instances. Massive input data and intermediate results are stored in the distributed cache offered by the platform. The raw input data is in the size of 300 megabytes and stored distributed across the platform. Each EC2 instances has Intel 64 bit CPU, 16 GB memories and high network performance. Moreover each node can be boosted up to 4 virtual CPU with the performance triple increased. Open source framework Hadoop is adopted by AWS for the distributed architecture of those nodes.

Amazon EC2 instances provide a number of additional features to deploy, manage, and scale our applications. Multiple storage options based on our requirements can be choose. Details about how EC2 instances work can be found in figure 5.

B. Result Evaluations

In the first part, we conduct the objective performance evaluation of deep learning algorithm. Error rate of our system is the primary concern, therefore we conducting several experiments to measure the error rate after several iterations. Figure 6 shows the number of training error and testing error as the iteration number increases, for unsupervised learning of hand-written digits. As we can see from figure 6 the error rate for reconstruction of hand-written digits decreases significantly after several iterations, for both training cases and testing cases. This is achieved by the application of RBM algorithm and back-propagation algorithm.

(a) Training Error v.s. iteration time curve (b) Testing Error v.s. iteration time curve

Fig. 6. Train and test error v.s. iteration time curve for unsupervised learning.

Figure 7 shows the number of misclassification in training and testing as the number of iterations increases, for supervised learning of hand-written digits recognition. As we can see in figure 7 (a), the training error soon reaches 0 after several iterations. However, in figure 7 (b), it shows that the misclassification rate of testing cases increases as number of iterations increases. This is a sign of over-fitting problem, which is clearly discussed in [23]. Over-fitting is inevitable in supervised learning and classification problems.

In the second part, we conduct the experiments on speed-up gained from MapReduce. The efficiency and scalability of our MapReduce method is tested on cloud clusters with 2,4,8,16,and 32 nodes. The test input size is 300 MB. From our result, we can clearly conclude that the running time of tasks is linearly dependent on the aspects of nodes numbers in the cluster. The slightly mismatch between result and our anticipation is because of the overhead of system architecture. Our result proves MapReduce have an excellent speed-up performance. Figure 8 shows the running time has an inverse
C. GUI Demo by MATLAB

To further refine our project, we implement two GUI demo software inherited from MATLAB GUI package. GUIs also known as graphical user interfaces will provide a type of point-and-click control of the software application we designed. By implementing a GUI interface, users are easier to understand the performance of our system. The reason why we choose to use MATLAB GUIs is MATLAB apps are self-contained programs with GUI front ends that automate a test. MATLAB GUIs usually includes controls such as menus, toolbars, buttons, and sliders.

There are two demos: figure 9 shows the screen shots of demo software for supervised learning (hand-written digit recognition) results, and figure 10 shows the screen shots of the demo software for unsupervised learning (hand-written digit auto-encoding and auto-decoding) results.

As figure 9 shown, to use the software, there are three steps:(a) step is to import the corresponding supervised learning weights file by selecting “File → Open”; (b) step is to draw a digit with mouse on left box; (c) step is to click the “recognize” button and the recognition result shows on the right.

As figure 10 shown, to use the software, there are four steps:(a) step is to import the corresponding unsupervised learning weights file by selecting “File → Open”; (b) step is to draw a digit with mouse on left box; (c) step is to click encode button and the code shows in the middle; (d) step is to click decode button and the reconstruction shows on the right. Note that the picture is 28 × 28 dimensions, and the code in the middle only has 30 dimensions, so the compress rate is 30 ÷ 784 = 0.357.

VII. CONCLUSIONS

In general, we successfully designed and implemented a MapReduce neural network algorithm on large scale of data which running on top of Amazon Web Service platform. Great improvements on efficiency and accuracy have been achieved by our system, due to the facts that our system is running on distributed file platform. We also observe that the running time of processing data is decreasing as the number of nodes increasing, almost in an inverse linear mode. The slightly mismatch between experiments and theory is because of the
system overhead when larger number of nodes is introduced into the system. We are convinced that with the application of MapReduce, neural network is capable of successfully recognizing handwriting digits with great efficiency and accuracy. In the future we are planning to apply our algorithm on more complex input model including face recognition, speech recognition and human language processing.

APPENDIX

The code repository can be found on Github at: https://github.com/sunkairan/MapReduce-Based-Deep-Learning

ACKNOWLEDGEMENT

The authors thank Dr. Andy Li and his teaching assistances for their suggestions that significantly improved the quality of the paper. The authors also thank Dr. Dapeng Oliver Wu and his team for providing deep discussions about our research details. The computing cluster resources used by this project is supported by the course of EEL-6935: Cloud Computing and Storage.

REFERENCES

[1] J. Hoskins and D. Himmelblau, “Process control via artificial neural networks and reinforcement learning,” Computers & chemical engineering, vol. 16, no. 4, pp. 241–251, 1992.
[2] M. T. Hagan, H. B. Demuth, M. H. Beale et al., Neural network design. Pws Pub. Boston, 1996.
[3] D. E. Rumelhart and J. L. McClelland, “Parallel distributed processing: explorations in the microstructure of cognition. volume 1. foundations,” 1986.
[4] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” Neural computation, vol. 18, no. 7, pp. 1527–1554, 2006.
[5] M. Pethick, M. Liddle, P. Werstein, and Z. Huang, “Parallelization of a backpropagation neural network on a cluster computer,” in International conference on parallel and distributed computing and systems (PDCS 2003), 2003.
[6] K. Ganeshamoorthy and D. Ranasinghe, “On the performance of parallel neural network implementations on distributed memory architectures,” in Cluster Computing and the Grid, 2008. CCGRID’08. 8th IEEE International Symposium on. IEEE, 2008, pp. 90–97.
[7] S. Suresh, S. Omkar, and V. Mani, “Parallel implementation of backpropagation algorithm in networks of workstations,” Parallel and Distributed Systems, IEEE Transactions on, vol. 16, no. 1, pp. 24–34, 2005.
[8] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, “Map-reduce for machine learning on multicore,” Advances in neural information processing systems, vol. 19, p. 281, 2007.
[9] Z. Liu, H. Li, and G. Miao, “Mapreduce-based backpropagation neural network over large scale mobile data,” in Natural Computation (ICNC), 2010 Sixth International Conference on, vol. 4. IEEE, 2010, pp. 1726–1730.
[10] D. Gillick, A. Faria, and J. DeNero, “Mapreduce: Distributed computing for machine learning,” Berkley (December 18, 2006), 2006.
[11] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, “Systemml: Declarative machine learning on mapreduce,” in Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011, pp. 231–242.
[12] L. G. Valiant, “A theory of the learnable,” Communications of the ACM, vol. 27, no. 11, pp. 1134–1142, 1984.
[13] S. Haykin, Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.
[14] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, 1989, pp. 593–605.
[15] M. C. Mozer, R. Wolniewicz, D. B. Grimes, E. Johnson, and H. Kaushansky, “Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry,” Neural Networks, IEEE Transactions on, vol. 11, no. 3, pp. 690–696, 2000.
[16] J. Ferreira, M. B. Vellasco, M. A. C. Pacheco, R. Carlos, and H. Barbosa, “Data mining techniques on the evaluation of wireless churn.” in ESANN, 2004, pp. 483–488.
[17] H. Hwang, T. Jung, and E. Suh, “An ltv model and customer segmentation based on customer value: a case study on the wireless telecommunication industry,” Expert systems with applications, vol. 26, no. 2, pp. 181–188, 2004.
[18] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, “Boosting the margin: A new explanation for the effectiveness of voting methods,” The annals of statistics, vol. 26, no. 5, pp. 1651–1686, 1998.
[19] Apache™. (2013) Hadoop®. [Online]. Available: http://hadoop.apache.org
[20] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” Journal of Internet Services and Applications, vol. 1, no. 1, pp. 7–18, 2010.
[21] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[22] C. C. Yann LeCun and C. J. Burges. (2013) The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[23] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern classification. John Wiley & Sons, 2012.