# On Distributed Deep Network for Processing Large-Scale Sets of Complex Data

Qin Chao, Gao Xiao-guang
School of Electronics and Information
Northwestern Polytechnical University
Xi'an, P.R.China
qinchaoaiziji@163.com

Chen Da-qing
School of Engineering
London South Bank University
London SE1 0AA, England
chend@lsbu.ac.uk

*Abstract*—**Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with hundreds of parameters using distributed CPU cores. We have developed Bagging-Down SGD algorithm to solve the distributing problems. Bagging-Down SGD introduces the parameter server adding on the several model replicas, and separates the updating and the training computing to accelerate the whole system. We have successfully used our system to train a distributed deep network, and achieve state-of-the-art performance on MINIST, a visual handwriting font library. We show that these techniques dramatically accelerate the training of this kind of distributed deep network.**

***Keywords-component; Distributed Deep Network; Bagging-Down SGD; Parameter server***

## I. INTRODUCTION

As a cutting-edge disruptive technology, deep learning has attracted a significant research attention. Since Hinton introduced a layer-wise training algorithm for multilayered networks, the theory and applications of deep learning have been greatly developed. State-of-the-art of this kind of machine learning has been reported in several domains, ranging from speech recognition [2, 3], visual object recognition [4, 5], to text processing [6, 7].Some big companies invested money and personnel to build their own deep network and achieved outstanding results [8]. Google announced its own distributed deep network framework "Tensorflow", which has a more flexible programming model than others [29].

It has been testified that increasing the scale of deep learning, with respect to the number of training examples, can drastically improve ultimate classification accuracy [4, 5, 9]. These results have led to a surge of interest in scaling up the training and inference algorithms used for these models and in improving applicable optimization procedures [9, 10]. The use of GPUs [1, 2, 3, 11] is a significant advance in recent years that makes the training of modestly sized deep networks practical. A known limitation of the GPU approach is that the training speed-up is small when the model does not fit in GPU memory (typically less than 6 gigabytes). To use a GPU effectively, researchers often reduce the size of the data or parameters so that CPU-to-GPU transfers are not a significant bottleneck [11].

While data and parameter reduction work well for small problems (e.g. acoustic modeling for speech recognition), they are less attractive for problems with a large number of examples and dimensions (e.g., high-resolution images).

Our Contribution: In this paper, we describe an efficient approach: Bagging-Down SGD, an asynchronous stochastic gradient descent procedure which leverages learning rates and supports a large number of model replicas, to assure the learning rate and accuracy while processing large scale of data.

The remainder of this paper is organized as four parts: We begin with a description of previous work in Section 2 and propose the Bagging-Down SGD (Stochastic gradient descent) algorithm in Section 3. And we conclude with experiments in Section 4.

## II. RELATED WORK

In recent years data sets have been growing dramatically fast. In order to deal with large-scale of datasets, a great many authors have explored scaling up machine learning algorithms through parallelization and distribution [12, 13, 14, 15, 16, 17, 18]. Within this area, some groups have relaxed synchronization requirements, exploring delayed gradient updates for convex problems [13, 18]. This can accelerate computationally intensive problems whenever gradient computations are relative expensive while sharing a common parameter vector [13]. In parallel, other groups working on problems with sparse gradients (problems where only a tiny fraction of the coordinates of the gradient vector are non-zero for any given training example) have explored lock-less asynchronous stochastic gradient descent on shared-memory architectures (i.e. single machines) [6, 19]. But the entire network needs to wait the slowest machine to training.

In the context of deep learning, most work has focused on training relatively small models on a single machine (e.g., Theano [20]). Suggestions for scaling up deep learning include the use of a farm of GPUs to train a collection of many small models and subsequently averaging their predictions [21], or modifying standard deep networks to make them inherently more parallelizable [22].

For special cases where one layer dominates computation, distributing computation has been considered in that one layer and replicating computation in the remaining layers [6]. But in the general case where many layers of the model are computationally intensive, full model parallelism [23] is required.

We are interested in an approach that allow the use of a cluster of machines asynchronously computing gradients, but without requiring that the problem be either convex or sparse. And our focus is distributing deep learning techniques in the direction of training, those with a few hundred parameters, but without introducing restrictions on the form of the model.

## III. A DISTRIBUTED OPTIMIZATION ALGORITHM

In order to train a large model in a reasonable amount of time, we need to parallelize computation not only within a single instance of the model, but to distribute training across multiple model instances. To be specific, we need to consider three Elements to combine those single models：

- The structure of each single model.
- The input data of each single model.
- The problem of speed allocation.

### *A. The Bagging-Down SGD algorithm*

Stochastic gradient descent (SGD) is perhaps the most commonly used optimization procedure for training deep neural networks [4, 26, 27]. Unfortunately, the traditional formulation of SGD is inherently sequential, making it impractical to apply to very large data sets where the time required to move through the data in an entirely serial fashion is prohibitive. For synchronous SGD (standard SGD), if one machine fails, the entire training process is delayed waiting the slowest machine finish training.

To apply SGD to large data sets and avoid the problem of synchronous SGD, we introduce Bagging-Down SGD, a variant of asynchronous stochastic gradient descent that uses multiple replicas of a single model. The processing of the algorithm is as Figure 1. The basic approach is as follows: we divide the training data $\mathcal{L} = \{(y_n, x_n), \mathrm{n} = 1 \ldots, \mathrm{N}\}$, where y denotes the output and the x denotes the input data, into several subsets $\mathcal{L}_\mathrm{k} = \{(\mathrm{y_n}, \mathrm{x_n}), \mathrm{n} = \mathrm{k_1} \ldots \mathrm{k_m}\}$ as the input datasets of each single model. The models interact with a centralized parameter server for model updating. Each model replica asks the parameter server service for an updated copy of its model parameter $\mathrm{W}'$, and provides the rate of parameter changing $\Delta\mathrm{W}$. The parameter server needs to compute the final rate of parameter changing after it receives $\Delta\mathrm{W}$ from model replicas. The algorithm of this parameter server is shown as Figure 2. Bagging and Speed Monitor collect all the rates $\mathrm{W}'$ and create a sequence $(\Delta\mathrm{W}_1 \ldots \ldots \Delta\mathrm{W}_\infty)$. The monitor extracts the first T parameters, which is used to compute the final $\mathrm{W}'$. After computing, the monitor deletes those $\mathrm{W}'$ to get a updated result in the next step. The monitor can solve the problem of the whole speed depending on the slowest one.

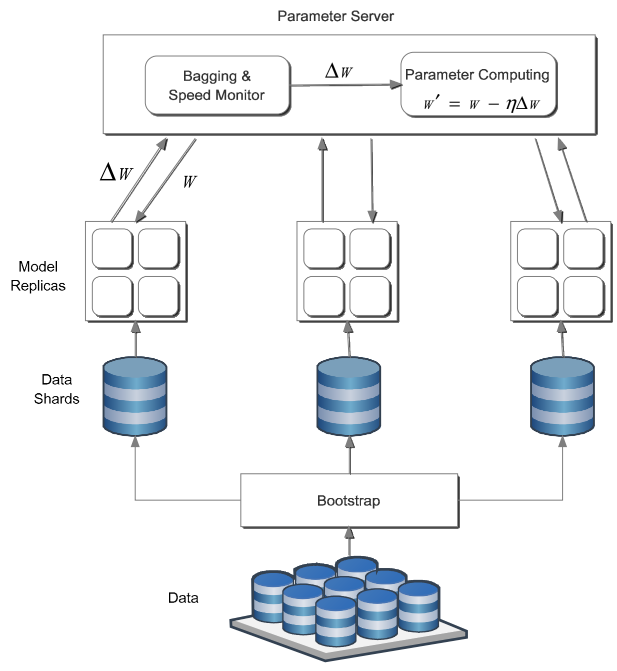

Figure 1. Bagging-Down SGD

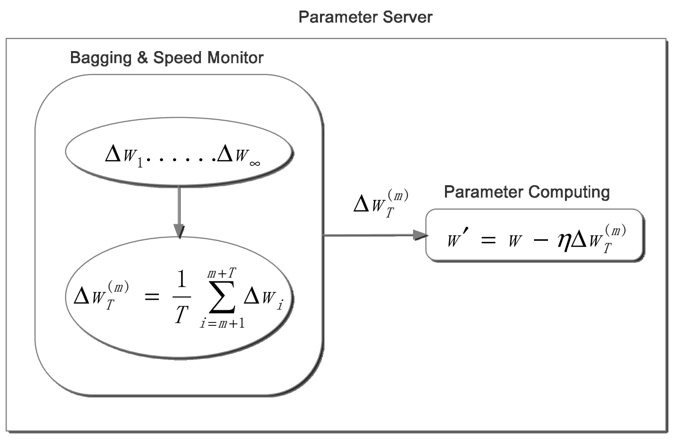
Parameter Server
Bagging & Speed Monitor
$\Delta w_1 \ldots\ldots \Delta w_\infty$
$\Delta w_T^{(m)} = \frac{1}{T}\sum_{i=m+1}^{m+T} \Delta w_i$
$\Delta w_T^{(m)}$
Parameter Computing
$w' = w - \eta\Delta w_T^{(m)}$

Figure 2. Parameter Monitor

In each step of the training, each single model require the newest $\mathrm{W}'$. In the algorithm, every model replica conducts computing independently and the communication with the input dataset is simple. Each model will never affect the training of others; as such it could reduce the communication-load and reducing the whole speed of the network.

Bagging-Down SGD is more robust to machines failures than synchronous SGD. For synchronous SGD, if one machine fails, the entire training process is delayed; whereas for Bagging-Down SGD, if one machine in a model replica fails, the other model replicas continue processing their training data and updating the model parameters via the parameter servers. On the other hand, the multiple forms of asynchronous processing in Bagging-Down SGD introduce a great deal of additional stochasticity in the optimization procedure. Most obviously, a model replica is almost certainly computing its gradients based on a set of parameters that are slightly out of date, in that some other model replica will likely have updated the parameters on the parameter server in the meantime. But there are several other sources of stochasticity beyond this: Because the parameter server shards act independently, there is no guarantee that at any given moment the parameters on each shard of the parameter server have undergone the same number of updates, or that the updates were applied in the same order. Moreover, because the model replicas are permitted to fetch parameters and push gradients in separate threads, there may be additional subtle inconsistencies in the timestamps of parameters. There is little theoretical grounding for the safety of these operations for nonconvex problems, but in practice

One technique that we have found to greatly increase the robustness of Bagging-Down SGD is the use of η (learning rate).

Rather than using a single fixed learning rate on the parameter sever (η in Figure 1), we use a separate adaptive learning rate for each parameter. Let $\eta_{i,k}$ be the learning rate of the i-th parameter at iteration k, then we set:

$$\eta_{i,k} = \Upsilon \Big/ \sqrt{\sum_{j=1}^{K} \Delta W_{i,j}^{\ 2}} \qquad (1)$$

The value of ϒ, the constant scaling factor for all learning rates, is generally larger (perhaps by an order of magnitude) than the best fixed learning rate. The changing extends the maximum number of model replicas that can productively work simultaneously, and combines with a practice of "warmstarting" model training with only a single model replica before unleashing the other replicas. This has virtually eliminated stability concerns in training deep networks using Bagging-Down SGD.

## B. *The Validation of Bagging and Speed Monitor.*

We assume that the $(\Delta W, x)$ sampled from distribution P independently. $\Delta\varphi_A(x) = E_{\mathcal{L}}\Delta\varphi(x,\mathcal{L})$ is the observation vector representing the final ΔW.

$$E_{\mathcal{L}}\big(\Delta W - \Delta\varphi(x,\mathcal{L})\big)^2 = \Delta W^2 - 2\Delta W\Delta\varphi(x,\mathcal{L}) + E_{\mathcal{L}}\Delta\varphi^2(x,\mathcal{L}) \quad (2)$$

We set $E_{\mathcal{L}}\Delta\varphi(x,\mathcal{L}) = \Delta\varphi_A(x)$ and use the inequality of $EZ^2 \geq (EZ)^2$ and we get:

$$E_{\mathcal{L}}\big(\Delta W - \Delta\varphi(x,\mathcal{L})\big)^2 \geq \big(\Delta W - \Delta\varphi_A(x)\big)^2 \qquad (3)$$

From the inequality (3), we can find that the mean square error in the Bagging-down algorithm is smaller than in others in most cases.

## IV. EXPERIMENTS

### A. *Parameter Designing*

Because the aim of this paper is to prove the superiority of the proposed distributed algorithm, we have chosen a simple single model in deep learning. In our experiment, we selected the AutoEncoder as the structure of each model replica. The advantage of AutoEncoder is that we could get less hidden layers and use more hidden nodes.

To facilitate the training of very large deep networks, we use a framework named DisDeepNet as Figure 3 which supports distributed computation in neural networks and layered graphical models with four machines. The messages should be passed during the upward and downward phases of the computation. The framework automatically parallelizes computation in each machine using all available cores, and manages communication, synchronization and data transfer between machines during both training.

We use the MINIST font library as the input data of the whole system. We set reconstruction error as:

$$L_H(x,z) = -\sum_{k=1}^{d}[x_k log z_k + (1 - x_k)\log(1 - z_k)] \qquad (4)$$

Each model replica has 28*28 input nodes, also 28*28 output nodes because of the AutoEncoder structure, and 500 hidden nodes.

In the distributed platform, we select the value of T in the bagging and speed monitor as the argument. We have designed six groups which T ranges in the array (0, 1, 2, 4, 6, 8). Note that if T=0 it represents the situation of no distributed training.

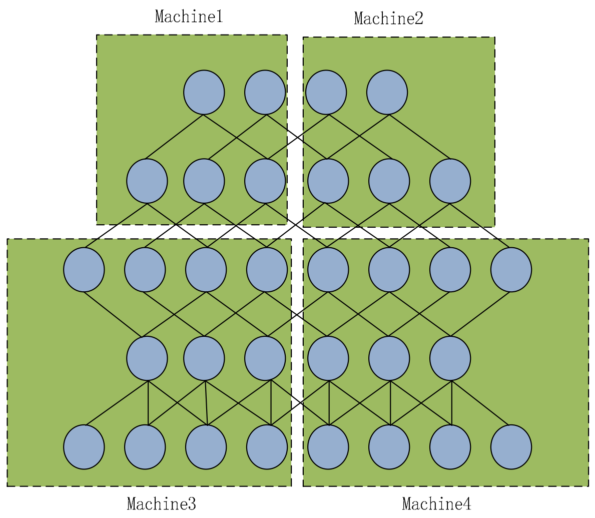

Figure 3. Conception of DisDeepNet

### B. *The experimental results*

Figure 4 shows the value of the reconstruction error in each step. It can be seen from the results that the model optimization gets better with the increase of T. And the rate of reconstruction error change is faster with a higher T.

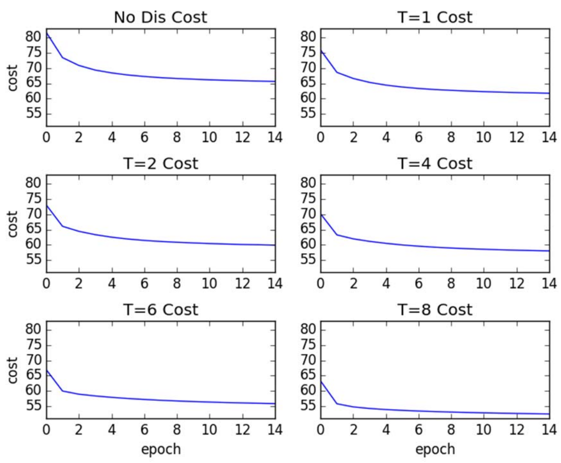

Figure 4. Reconstruction Error

Figure 5 shows the changing of training time. The curve get stable after T = 4.

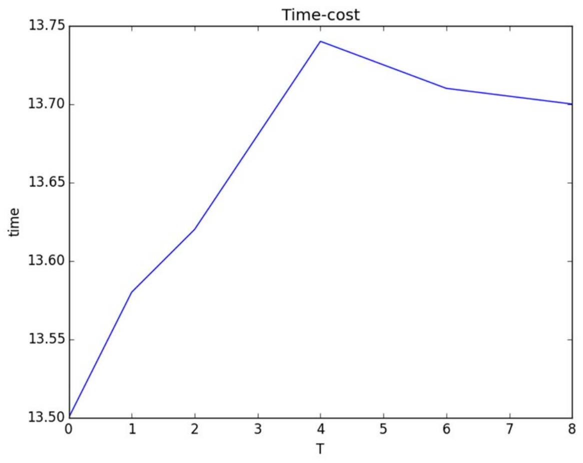

Figure 5. Time cost

## V. Conclusions

In section 4.2, we proved the validation of the proposed algorithm. From the experiment we have shown the rate of the reconstruction error is more remarkable using a higher T, and changing the parameter T in the Bagging and Speed Monitor is remarkably effective.

We can tell that the values of time-cost are different. The final cost is increased with T ranging from 0 to 4. It is not difficult to think of the communication-load as the main cause for this. Also, the speed of updating in parameter server is fast, which makes the fact that different machines get different value and the slower machine does the same job as the faster machine did several steps back. We conclude that the T in parameter server can reduce the communication-load and gives the whole system better speed allocation.

At the moment, the algorithm is offline; however we could train online datasets because we divide the entire input dataset using a sampling method. And we inferred to add more machines to training in order to get faster training speed and lower reconstruction error.

## References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [1]. Where appropriate, include the name(s) of editors of referenced books. The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first . . ."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] Hinton G E, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets[J]. Neural computation, 2006, 18(7): 1527-1554.

[2] Dahl G E, Yu D, Deng L, et al. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition[J]. Audio, Speech, and Language Processing, IEEE Transactions on, 2012, 20(1): 30-42.

[3] Hinton G, Deng L, Yu D, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups[J]. Signal Processing Magazine, IEEE, 2012, 29(6): 82-97.

[4] Claudiu Ciresan D, Meier U, Gambardella L M, et al. Deep big simple neural nets excel on handwritten digit recognition[J]. arXiv preprint arXiv:1003.0358, 2010.

[5] Coates A, Ng A Y, Lee H. An analysis of single-layer networks in unsupervised feature learning[C]//International conference on artificial intelligence and statistics. 2011: 215-223.

[6] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. The Journal of Machine Learning Research, 2003, 3: 1137-1155.

[7] Collobert R, Weston J. A unified architecture for natural language processing: Deep neural networks with multitask learning[C]//Proceedings of the 25th international conference on Machine learning. ACM, 2008: 160-167.

[8] Bengio Y. Deep learning of representations: Looking forward[M]//Statistical Language and Speech Processing. Springer Berlin Heidelberg, 2013: 1-37.

[9] Ngiam J, Coates A, Lahiri A, et al. On optimization methods for deep learning[C]//Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011: 265-272.

[10] Martens J. Deep learning via Hessian-free optimization[C]//Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010: 735-742.

[11] Raina R, Madhavan A, Ng A Y. Large-scale deep unsupervised learning using graphics processors[C]//Proceedings of the 26th annual international conference on machine learning. ACM, 2009: 873-880.

[12] Shi Q, Petterson J, Dror G, et al. Hash kernels[C]//International Conference on Artificial Intelligence and Statistics. 2009: 496-503.

[13] Langford J, Smola A, Zinkevich M. Slow learners are fast[J]. arXiv preprint arXiv:0911.0491, 2009.

[14] Mcdonald R, Mohri M, Silberman N, et al. Efficient large-scale distributed training of conditional maximum entropy models[C]//Advances in Neural Information Processing Systems. 2009: 1231-1239.

[15] McDonald R, Hall K, Mann G. Distributed training strategies for the structured perceptron[C]//Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2010: 456-464.

[16] Zinkevich M, Weimer M, Li L, et al. Parallelized stochastic gradient descent[C]//Advances in neural information processing systems. 2010: 2595-2603.

[17] Agarwal A, Chapelle O, Dudík M, et al. A reliable effective terascale linear learning system[J]. The Journal of Machine Learning Research, 2014, 15(1): 1111-1133.

[18] Agarwal A, Duchi J C. Distributed delayed stochastic optimization[C]//Advances in Neural Information Processing Systems. 2011: 873-881.

[19] Niu F, Recht B, Re C, et al. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent[J]. Nips, 2011:693-701.

[20] Bergstra J, Breuleux O, Bastien F, et al. Theano: a CPU and GPU math expression compiler[C]//Proceedings of the Python for scientific computing conference (SciPy). 2010, 4: 3.

[21] Ciresan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification[C]//Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012: 3642-3649.

[22] Deng L, Yu D, Platt J. Scalable stacking and learning for building deep architectures[C]//Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012: 2133-2136.

[23] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images[J]. 2009.

[24] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[25] Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.

[26] Bottou L. Stochastic gradient learning in neural networks[J]. Proceedings of Neuro-Nımes, 1991, 91(8).

[27] LeCun Y A, Bottou L, Orr G B, et al. Efficient backprop[M]//Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012: 9-48.

[28] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization[J]. The Journal of Machine Learning Research, 2011, 12: 2121-2159.

[29] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015[J]. Software available from tensorflow. org.

[30] Beltramelli T, Risi S. Deep-Spying: Spying using Smartwatch and Deep Learning[J]. arXiv preprint arXiv:1512.05616, 2015.

[31] Nalisnick E, Ravi S. Infinite Dimensional Word Embeddings[J]. arXiv preprint arXiv:1511.05392, 2015.

[32] Pan X, Papailiopoulos D, Oymak S, et al. Parallel Correlation Clustering on Big Graphs[C]//Advances in Neural Information Processing Systems. 2015: 82-90.

[33] Bengio Y. How auto-encoders could provide credit assignment in deep networks via target propagation[J]. arXiv preprint arXiv:1407.7906, 2014.

[34] Grosse R B, Maddison C J, Salakhutdinov R R. Annealing between distributions by averaging moments[C]//Advances in Neural Information Processing Systems. 2013: 2769-2777.

[35] Längkvist M, Karlsson L, Loutfi A. A review of unsupervised feature learning and deep learning for time-series modeling[J]. Pattern Recognition Letters, 2014, 42: 11-24.