Performance of Training Sparse Deep Neural Networks on GPUs

Jianzong Wang*, Zhangcheng Huang*, Lingwei Kong*, Jing Xiao*, Pengyu Wang†, Lu Zhang†, Chao Li†
*Ping An Technology (Shenzhen) Co., Ltd
{wangjianzong347, huangzhangcheng624, konglingwei630, xiaojing661}@pingan.com.cn
†Shanghai Jiao Tong University
{wpybtw, luzhang}@sjtu.edu.cn, lichao@cs.sjtu.edu.cn

***Abstract*—Deep neural networks have revolutionized the field of machine learning by dramatically improving the state-of-the-art in various domains. The sizes of deep neural networks (DNNs) are rapidly outgrowing the capacity of hardware to fast store and train them. Over the past few decades, researches have explored the prospect of sparse DNNs before, during, and after training by pruning edges from the underlying topology. After the above operation, the generated neural network is known as a sparse neural network. More recent works have demonstrated the remarkable results that certain sparse DNNs can train to the same precision as dense DNNs at lower runtime and storage cost.**

Although existing methods ease the situation that high demand for computation resources severely hinders the deployment of large-scale DNNs in resource-constrained devices, DNNs can be trained at a faster speed and lower cost. In this work, we propose a Fine-tune Structured Sparsity Learning (FSSL) method to regularize the structures of DNNs and accelerate the training of DNNs. FSSL can: (1) learn a compact structure from large sparse DNN to reduce computation cost; (2) obtain a hardware-friendly to accelerate the DNNs evaluation efficiently. Experimental results of the training time and the compression rate show that superior performance and efficiency than the Matlab example code. These speedups are about twice speedups of non-structured sparsity.

***Index Terms*—sparse neural networks, sparse matrices, graph analysis, GPU Computing**

I. Introduction

Driven by the availability of massive data and the computational capability to process it, deep learning has recently emerged as a critical tool for solving complex problems across a wide range of domains, including image recognition [1], speech processing [2]–[4], natural language processing [5], language translation [6], and autonomous vehicles [7]. However, deep Neural Networks (DNNs) [8], [9] have many parameters, which leads to problems related to storage, computation, and energy cost.

Thus, there is significant interest in exploring the effectiveness of sparse DNN representations where many of the weight values are zero. As a result, methods for efficiently processing them are often critical to the performance of many applications. Some methods are based on decomposition and factorization [10]. These methods can preserve the regular dense computation structure of the original models, thus can achieve both compression and acceleration on general-purpose processors. Pruning serves as another effective method to reduce significantly the number of parameters with no loss of accuracy [11].

In this paper, we show that this redundancy makes it possible to notably reduce the amount of computation required to process Sparse matrix-vector multiplication (SpMV) [12], [13] operations, by sparse decompositions of the sparse DNNs. Two-Stage decompositions are applied to explore the inter-channel and intrachannel redundancy of the sparse DNNs. We first perform an initial decomposition based on a graph structure to transform the sparse matrix to a CSR format [14], then calculate the sparse matrix multiplication based on a fine-tune cuSparse library [15]. In the fine-tuning phase, we optimize the network training error, the sparsity of sparse DNNs, as well as the processing time together by minimizing a fused loss function. Surprisingly high sparsity can be achieved in our model. We can zero out more than 90% of the sparse parameters of the deep network in short processing time while keeping high accuracy.

The organization of the rest of this paper is as follows. Section II provides information on our approaches and the relevant computations of CSR format. Section III describes the experimental setup and evaluation criteria used in the Challenge. Section IV presents the experimental results and discusses the reason for each outcome. Section V summarizes the work and describes future directions.

II. Approach

A. Problem Definition

- Sparse Matrix and Sparse DNN
  In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense.
- Compressed Structures
  The sparsity of the data has led to the development of several data representations, which are common for both problem formulations [16]: Compressed Sparse Row (CSR), Coordinate (COO), Compressed Sparse Column (CSC), and ELL (Ellpack). Unlike a dense adjacency matrix, which may be potentially filled with 0-values,

Corresponding Author: Jianzong Wang, jzwang@188.com

This paper is supported by National Key Research and Development Program of China under grant No. 2018YFB1003503.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

these formats avoid storing these trivial values. As such, these data-structures are cost-effective in terms of memory. Next, we will introduce our sparse matrix transform approach thoroughly.

B. SPARSE MATRIX TRANSFORM

The massive parallelism of graphics processing units (GPUs) offers tremendous performance in many high-performance computing applications. While dense linear algebra readily maps to such platforms, harnessing this potential for sparse matrix computations presents additional challenges. Given its role in iterative methods for solving sparse linear systems and eigenvalue problems, sparse matrix-vector multiplication (SpMV) is of singular importance in sparse linear algebra. Alexander et al. [17] have implemented data structures and algorithms for SpMV that are efficiently implemented on the CUDA platform for the fine-grained parallel architecture of the GPU. Based on that, we transformed the sparse DNN into a CSR so that given the memory-bound nature of SpMV, we can emphasize memory bandwidth efficiency and compact storage formats.

As the standard representation of a sparse matrix is a structure where non-zero elements are linked in rows and columns, we adopt a general graph structure corresponding to this representation especially.

C. Format the CSR Matrix

The compressed sparse row (CSR) format is perhaps the most popular general-purpose sparse matrix representation. CSR explicitly stores column indices and nonzero values in arrays $column\ indices$ and $values$. A third array of row pointers, $row\ offsets$, allows the CSR format to represent rows of varying length. Figure. 1 show the mechanism.

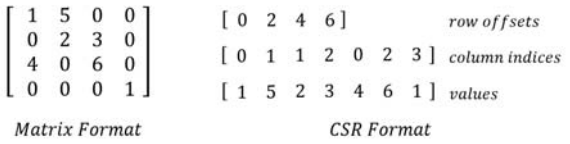

Fig. 1. The schematic diagram of CSR.

Sparsity helps reduce the computational complexity of deep neural networks by skipping zeros. Taking advantage of sparsity is listed as a high priority in the next generation DNN accelerators. Next, we will focus on the operation of sparse matrix multiplication based on the CSR format.

D. Sparse Matrix Multiplication

In our problem settings, we focus on the sparse-sparse matrix multiplication problem $C = A \times B$. A $\in \Re^{m \times k}$ is a sparse weight input and B $\in \Re^{k \times n}$ is a sparse MNIST input. A and B are divided into the CSR structure in the same way as the sparse matrix case described above. Now, we consider the multiplication of one row strip and one column strip

$$\widetilde{C} = \widetilde{A} \times \widetilde{B}, \widetilde{A} \in \Re^{1024 \times k}, \widetilde{B} \in \Re^{k \times 1024}, \widetilde{C} \in \Re^{1024 \times 1024}$$

. For any matrix M, let m_i be the i^{th} row of M and m_j be the j^{th} column of M. The matrix multiplication can be represented as

$$\overline{c}_{*,j} = \sum_{i=1}^{k} \overline{a}_{*,i} \overline{b}_{i,j}, 1 \leq j \leq 1024 \quad (1)$$

In which every $a_{*,j}$ and $b_{*,j}$ are held in three CSR vectors. Since both $\widetilde{A}$,$\widetilde{B}$ are sparse in our case, we need to have the knowledge of the locations of non-zero elements and skip the zeros. To avoid indirect memory access, we propose to encode the structure of $\widetilde{C}$ into the program.

E. Sparse Matrix Multiplication in CuSparse

There are two main considerations on designing an efficient matrix multiplication algorithm: (a) Taking advantage of the CSR format for higher computing throughput; (b) Maximally utilizing the CSR format to skip the zeros for memory reduction. We implemented two versions of baseline (with/without buffer) by leveraging CusparseDcsrgemm and cusparseDcsrgemm2 API [15]. CuCparseDcsrgemm does not need to allocate a buffer to store intermediate data while CuSparseDcsrgemm2 needs a buffer. The usage of the buffer may limit the size of data to compute.

We also observe that a large number of rows in result matrix could be all-zero. So we further implement the $filter$ operation before the calculation in each thread block. Explicitly, the filter will skip rows that are all zeros for better hardware resource utilization.

III. EXPERIMENTS

In this section, we first describe the synthetic DNNs used in the Challenge then provide the specifics of the input feature dataset based on MNIST images. Secondly we discuss the relevant metrics for the Challenge. Then we lay out the experimental results and relevant explanations.

A. Synthetic NEURAL NETWORK DATA

Since large sparse DNNs are difficult to obtain from real data, we must simulate data with desired topological properties. The Graph Challenge [18] proposed a scheme of formating the synthetic neural network data. The RadiX-Net synthetic sparse DNN generator [19] is an efficient approach of generating pre-determined DNNs. Different base of the approach of numerations are used by RadiX-Net for the specific connectivity of the pre-determined DNNs. Combining with the Kronecker products, the DNNs can be overspread to large scaled DNNs. We can take 2 as an exapmle.

The Graph Challenge produced different Sparase DNNs with various RadiX-Net parameters from shallow to deeper neurons, which are shown in Table. III-A. The scale of the produced large sparse DNNs are shown in Table. III, and the total number of connections in the largest Sparase DNN is 32x1920x65536. This approach of expanding neural scale can remain the property in original DNNs.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

TABLE I
FOUR PRE-DETERMINED DNNS GENERATED UNDER THE STANDARD OF THE GRAPH CHALLENGE [18].

Number of Layers	Number of Neurons per Layer	Connections per Layer	Density	Bias
6	1024	32	0.03	-0.30
8	4096	32	0.008	-0.35
10	16384	32	0.002	-0.40
12	65536	32	0.0005	-0.45

TABLE II
DIFFERENT SCALE OF THE DNNS DESCRIBED BY THE TOTAL NUMBER OF CONNECTIONS IN THE SPARSE DNN GRAPH CHALLENGE [18].

Neurons \ Layers	***120***	***480***	***1920***
1024	32x120x1024	32x480x1024	32x1920x1024
4096	32x120x4096	32x480x4096	32x1920x4096
16384	32x120x16384	32x480x16384	32x1920x16384
65536	32x120x65536	32x480x65536	32x1920x65536

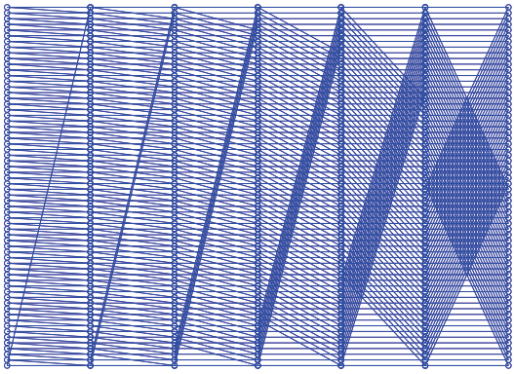

Fig. 2. The structure produced under the Graph Challenge with 6 layer, 1024 neurons per layer and 32 connections per layer. Figure reproduced from [18].

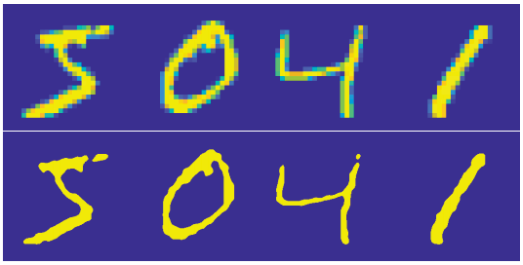

Fig. 3. The original 28x28 pixel images and the 256x256 resampled images in the 60,000 handwritten digits MNIST dataset . Figure reproduced from [18].

B. INPUT DATA SET

The input data required to implement sparse DNNs Graph Challenge is design as the MNIST dataset. MNIST [20] consists 60,000 28x28-pixel handwritten digits and it is a famous databose for people to try machine learning techniques and has been applied in many DNN training approaches. Graph Challenge uses the dataset with 28x28 resolution images, which can be reproduced to different size as shown in (Figure. 3). The final dataset are images with 1024 (32x32-pixel) neurons, 4096 (64x64-pixel) neurons, 16384 (128x128) neurons and 65536 (256x256-pixel) neurons. The thresholded image data contains values of 0 or 1 in each pixel, so that the non-zero values are written in the style of .tsv files.

C. Evaluation Methodology

Correctness Correctness is evaluated by comparing the reported categories with the ground truth categories provided. Loss Function of the MNIST Classification: Cross entropy is used as a loss function for image classification:

$$H(p,q) = -\sum_{x} p(x) log q(x) \tag{2}$$

where p is the target output and q is the probability output of the networks. Performance

- Total number of non-zero connections in the given DNN;
- Execution time: Total time required to perform DNN inference;
- Rate: Measures the throughput of the implementation as the ratio of the number of inputs (e.g., number of MNIST images) times the number of connections in the DNN divided by the execution time;
- Processor: Number and type of processors used in the computation.

D. Computer Information

The following section of the results are based on the machines listed in Table III.

TABLE III
EXPERIMENT COMPUTER CORRESPONDING TO THE MACHINE MEMORY, CPU AND GPU INFORMATION.

Computer Memory	***CPU***	***GPU***
64G	Intel Core i7 6700K	GeForce GTX 1080
256G	Intel Xeon E7-8860 v4	Tesla V100-PCIE

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

TABLE IV
EXPERIMENTAL RESULTS BASED ON THE CUSPARSE WITH BUFFER

Number of Nuerons Per Layer	Number of Layers	Time Performance (/s)	
		1080	v100
1024	120	9.81046	2.78748
	480	35.1718	9.78644
	1920	135.981	39.3201
4096	120	55.9797	14.5041
	480	208.488	56.5171
	1920	815.878	228.183
16384	120	/[1]	99.4049
	480	/[1]	403.006
	1920	/[1]	1610.5

[1] means not enough memory.

TABLE V
TIME RESULTS BASED ON THE CUSPARSE WITHOUT BUFFER

Number of Nuerons Per Layer	Number of Layers	Time Performance (/s)	
		1080	v100
1024	120	7.2919	3.90375
	480	26.1885	14.6403
	1920	101.04	57.7488
4096	120	90.2148	49.9786
	480	345.722	197.274
	1920	1372.07	788.416
16384	120	/[1]	601.705
	480	/[1]	2420.39
	1920	/[1]	9737.52

[1] means not enough memory.

IV. RESULTS

The results for the various computational experiments are presented from Table III to VIII based on multiple synthetic sparse deep neural network data. From these results, we can see that:

- From the comprehensive observation and analysis of these six tables, we can find that the same approach exhibits completely different performance on different devices and data sets.
- Table IV and VII show that as the number of neurons per layer and the number of layers increase, training time increases and inference rate decreases. At the same time, it's easy to discover that the growth multiples of the number of layers and time cost increase nonlinearly at 120 and 480, but at 480 and 1920, the growth of them is almost linear such as $35.1718 \times 4 \approx 135.981$. Although time and rate are closely related, the characteristics of rate and time are quite different. When the number of neurons per layer is fixed, the relationship between the number of layers and time presents a simple growth relationship. When the number of layers is fixed, the time grows semi-exponentially with the number of neurons per layer, such as $e^{9.81046/2} \approx 55.9797$.
- Table V-VIII and Table VI-IX have unique characteristics different from Table IV-VII in detail, but their overall trend is the same, the difference lies in the numerical value rather than the rule. Compared with Table IV-VII, the value of Table V-VIII is lower than the ideal value, while Table VI-IX is higher than the ideal value. For example, $7.2919 \times 4 > 26.1885$ and $810.675 \times 4 > 3253.21$.
- Based on the results of the above experiments, the overall performance of method Cusparse with Filter is the best, followed by method Cusparse with Buffer and method Cusparse without Buffer is the worst. The results when the number of neurons per layer = 1024, the hardware device is 1080 are noteworthy. The reason for this is that this combination is suitable for the calculation of a small amount of data.

TABLE VI
TIME RESULTS BASED ON THE CUSPARSE WITH FILTER

Number of Nuerons Per Layer	Number of Layers	Time Performance (/s)	
		1080	v100
1024	120	9.70645	2.75839
	480	36.2174	9.7138
	1920	139.968	38.2703
4096	120	51.5869	13.9857
	480	200.405	54.1294
	1920	810.675	223.583
16384	120	3253.21	98.1582
	480	4781.61	399.274
	1920	/[1]	1588.71

[1] means not enough memory.

TABLE VII
RATE RESULTS BASED ON THE CUSPARSE WITH BUFFER

Number of Nuerons Per Layer	Number of Layers	Rate [1]	
		1080	v100
1024	120	2.40488e+10	8.43365e+10
	480	2.68317e+10	9.64312e+10
	1920	2.77603e+10	9.60037e+10
4096	120	1.68582e+10	6.50565e+10
	480	1.81059e+10	6.67917e+10
	1920	1.85071e+10	6.61727e+10
16384	120	/ [2]	3.79747e+10
	480	/ [2]	3.74672e+10
	1920	/ [2]	3.75027e+10

[1] inputs*connections/time
[2] means not enough memory.

V. CONCLUSIONS

Deep neural networks has revolutionized the field of machine learning by dramatically improving the state-of-the-art in various domains. However the sizes of deep neural network architectures put increasing strain on the hardware needed to implement them. In the interest of reduced storage and runtime costs, much research over the past decade has focused on the sparsification of these neural networks.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

TABLE VIII
RATE RESULTS BASED ON THE CUSPARSE WITH BUFFER

Number of Nuerons Per Layer	Number of Layers	Rate [1]	
		1080	v100
1024	120	3.2355e+10	6.04366e+10
	480	3.60356e+10	6.44601e+10
	1920	3.73602e+10	6.53671e+10
4096	120	1.04608e+10	1.88824e+10
	480	1.09188e+10	1.91352e+10
	1920	1.10049e+10	1.91517e+10
16384	120	/ [2]	6.27363e+9
	480	/ [2]	6.22304e+9
	1920	/ [2]	6.20261e+9

[1] inputs*connections/time
[2] means not enough memory.

TABLE IX
RATE RESULTS BASED ON THE CUSPARSE WITH BUFFER

Number of Nuerons Per Layer	Number of Layers	Rate [1]	
		1080	v100
1024	120	2.43065e+10	8.56385e+10
	480	2.6057e+10	8.92383e+10
	1920	2.69695e+10	9.06312e+10
4096	120	4.70905e+9	6.71254e+10
	480	4.65646e+9	6.69231e+10
	1920	4.64142e+9	6.60102e+10
16384	120	/ 7.89457e+8	3.92834e+10
	480	/ [2]	3.83412e+10
	1920	/ [2]	3.82711e+10

[1] inputs*connections/time
[2] means not enough memory.

In this paper, we propose the Fine-tune Structured Sparsity Learning (FSSL) method to fully take advantage of the CSR format data to encode the large sparse matrix; then utilize the CSR format to calculate the sparse matrix-vector multiplication(spmv). We have developed customized techniques (with/without buffer) for sparse matrix multiplication implementation. Additionally, an filter operation is developed to reduces memory cost furthermore. Finally we test our two-step SpMV model with the MNIST handwritten dataset under the basis of Graph Challenge and the results show superior performance and efficiency for training sparse deep neural networks.

VI. ACKNOWLEDGEMENT

This paper is supported by National Key Research and Development Program of China (No. 2018YFB1003503). Corresponding author is Jianzong Wang from Ping An Technology (Shenzhen) Co., Ltd.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] G. Hickok and D. Poeppel, "The cortical organization of speech processing," *Nature reviews neuroscience*, vol. 8, no. 5, p. 393, 2007.

[3] R. J. Zatorre, A. C. Evans, E. Meyer, and A. Gjedde, "Lateralization of phonetic and pitch discrimination in speech processing," *Science*, vol. 256, no. 5058, pp. 846–849, 1992.

[4] A. Gray and J. Markel, "Distance measures for speech processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 5, pp. 380–391, 1976.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[6] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, 2007, pp. 177–180.

[7] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[10] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, 2016.

[11] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 1135–1143.

[12] R. Li and Y. Saad, "Gpu-accelerated preconditioned iterative linear solvers," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 443–466, 2013.

[13] S. Yan, C. Li, Y. Zhang, and H. Zhou, "yaspmv: yet another spmv framework on gpus," in *Acm Sigplan Notices*, vol. 49, no. 8. ACM, 2014, pp. 107–118.

[14] J. L. Greathouse and M. Daga, "Efficient sparse matrix-vector multiplication on gpus using the csr storage format," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 769–780.

[15] M. Naumov, "Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas," *Nvidia white paper*, 2011.

[16] F. Busato, O. Green, N. Bombieri, and D. A. Bader, "Hornet: An efficient data structure for dynamic sparse graphs and matrices on gpus," in *2018 IEEE High Performance Extreme Computing Conference, HPEC 2018, Waltham, MA, USA, September 25-27, 2018*, 2018, pp. 1–7.

[17] A. Monakov and A. Avetisyan, "Implementing blocked sparse matrix-vector multiplication on NVIDIA gpus," in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 9th International Workshop, SAMOS 2009, Samos, Greece, July 20-23, 2009. Proceedings*, 2009, pp. 289–297.

[18] V. G. M. J. L. M. R. R. Jeremy Kepner, Simon Alford and S. Samsi1, "Sparse deep neural network graph challenge," 2019.

[19] R. A. Robinett and J. Kepner, "Radix-net: Structured sparse matrices for deep neural networks," *CoRR*, vol. abs/1905.00416, 2019.

[20] F. Busato, O. Green, N. Bombieri, and D. A. Bader, "Sparse deep neural network graph challenge," 2019, pp. 1–7.