International Journal of Future Generation Communication and Networking
Vol. 13, No. 3, (2020), pp. 1234–1239

Parallelization of Deep Convolutional Neural Network

Umesh Chavan[1], Dinesh Kulkarni[2]

[1]*Department IT, Walchand College of Engineering,Sangli*
[2]*Department IT, Walchand College of Engineering,Sangli*

Abstract

Convolutional Neural Network (CNN) in Deep Learning (DL) has been has been achieving success in various objectives of pattern recognizing and classification. Training CNN model in an acceptable time is necessary. This is computationally intensive task. We have analyzed speedup in training of complex DL model using single core CPU and GPU. In the distributed setting, we studied weight update algorithms. We analyzed performance characteristics using our own designed DL model for facial expression recognition task. The novelty of this effort is to demonstrate performance acceleration in distributed DL frame-work. Analysis and study show that DL training performance improved over six times speedup for the own model.

Keywords: *Convolutional Neural Network, GPU, Deep Learning.*

1. Introduction

CNN has shown better classification accuracies in various applications of computer vision. In various applications like computer vision, CNN is found to be more effective [1]. Learning capability can be processed in better using Deeper network [2].Learning of CNN is mostly carried out using SGD algorithms. Parallelization CNN is comparatively challenging due to inherent sequential nature of algorithm. In this approach input data set is divided as mini-batch. Minimizing cost by updating parameters of the model using gradients is the goal in training of CNN model [3].Commonly used optimization methods for optimizing SGD: Adam, AdaMax, Adagrad, Adadelta. Model parameters are updated using gradients at each iteration on mini-batches of input data set. Across mini-batches there is data depend on the parameters. There are millions of parameters in typical CNN model. This requires a large amount of data to learn these parameters of the model. This results into slow training of the model. Typically, it takes order of days of training the CNN model. One of the successful models -VGG took order of days to train on a single core CPU. It accelerates speed to 1/c on a single machine where there are c cores in the CPU. It typically reduces to in order of hours for training. But the bottleneck is due to large datasets it is practically difficult to store whole data set on a single machine. Parallelism and distribution of model and training can run much faster. This can significantly reduce the end to end training of CNN.

2. Convolutional Neural Network

CNN is a special instance of ANN in which the connectivity pattern between its neurons is inspired by the organization of animal visual cortex. The visual cortex has small regions of cells in the brain fires only when exposed to vertical, horizontal and diagonal edges. CNN has Convolutional, ReLU, Pooling and fully connected layers [4].

A. SGD CNN Training Algorithm

Back propagation algorithm is adapted in training of CNN model [5]. The training consists of two phase's forward and backward propagation forward propagation the activations are calculated using input images. The output at each layer is propagated in forward direction. In backward propagation, errors are calculated based on the activation at the last layer. The error propagation is in backward direction. The errors and activations are calculated using

$$a_n^l = \sigma\left(\sum_{i=0}^{w-1}(w_i^l).w_i^l.a_{n+i}^{l-1} + b_n^l\right) \quad (1)$$

$$e_n^l = \sum_{i=0}^{w-1} w_i^l\, w_i^{l+1}.e_{n-i}^{l+1} \quad (2)$$

$$a_n^l = \sigma\left(\sum_{i=0}^{w-1}(w_i^l)w_i^l.a_{n+i}^{l-1} + b_n^l\right) \quad (3)$$

$$e_n^l = \sum_{i=0}^{w-1} w_i^l\, w_i^{l+1}.e_{n-i}^{l+1} \quad (4)$$

ISSN: 2233-7857 IJFGCN

Algorithm 1: SGD -CNN Training algorithm
(T: Number of mini-batches , N: Number of Layers in CNN,)

1: **for** mini-batch *t = 0,…T-1* **do**
2: $\Delta\theta = 0$
3: Get the t^{th} mini-batch, D^t
4: **for** layer *l = 0,… N-1* **do**
5: compute A^l based on D^t (activation).
6: end for
7: **for** layer *l = N- 1,…0* **do**
8: compute E^l (error).
9: compute $\Delta\theta^l$ (weight Gradients).
10: end for
11: **for** layer *l = 0; …N*-1 **do**
12: Update parameters, B^l and θ^l.
13: end for
14: **end for**

Algorithm 1 presents training of CNN. The input data set is partitioned into **T** total number of mini-batches. CNN is trained on subsets of data sets of mini-batches. Training on single mini-batch is known as one iteration. mster loop in 1 has a major data-dependency. This is limitation to the scalability of SGD training. The overall accuracy of this algorithm is $O\ (N\,T\,BK^2)$. T is the number of mini-batches, N is the total number of layers in model, B is size of each mini-batch. K is the largest number of neurons among all the layers in CNN model.

3. Parallelization of CNN

A. Distributed SGD

Parallelization and distribution of SGD computation across nodes/machines and multiple cores are the approaches can be applied. In multi-core processing the CNN model and data are stored on a single node. Here we are assuming that whole model and data can fit into a memory of single node having multiple cores. These cores will process on multiple data set at once in each layer or in another approach multiple cores performs SGD on mini-batches in parallel. We can use GPU for computational intensive tasks like matrix vector multiplication in algorithm. It can be possible to use both mulit-core as well as GPU where all cores share GPU and computationally intensive task can handle by the GPU. When practically it is not possible to fit a data set or model on single node, it will be better to split data set or model across multiple nodes/machines. In data parallelism approach data is distributed across multiple nodes. In model parallelism model can be split across nodes.in parallel SGD algorithm 4 the first step is to shuffle the input data set so as to each node in the cluster of nodes will have a representative subset of it.

B. GPU

Usually the first layer in CNN is convolution layer. In convolution operation a filter known as kernel is sided over an image [8]. This does an element-wise multiplication it has an overlap. Sum of this product will be assigned to the pixel under processing which is center to the block under processing. The convolution operations extract features from the input images like edges. In typical CNN which have many layers of convolution and each layer have multiple stacks of kernels. This overall convolution operation for network will require more computation time if computed with CPU i.e. in sequential execution. Therefore, an important way to improve the performance of the whole network is to reduce the run-time of convolution. The CPU handles all the complicated logic part of this process. Algorithm 2 and 3 gives the difference in processing mechanism with CPU and GPU respectively. Iterations in convolutions are concurrently processed in multi-threaded architecture of GPU. Multi-GPU platforms are widely adopted to speed up DNN training through parallel execution [9].

4. Analysis of Parallel SGD Algorithm

Computation cost and communication time analysis is carried out in this section. This is apart from cost of shuffling data sets.

Algorithm 2:CPU convolution (each point in the image)	**Algorithm 3: GPU convolution (have many cores, each core corresponds to point in the image)**
1: for i in output row: do 2: for j in output column: do 3: for k in output layer: do 2: do convolution 5: end for 6: end for 7: end for	1: for every thread do 2: do convolution 3: end for

The computation time for convolution of image of size N x N and size of kernel m x m is $t_c(2m^2 + 1)A = \pi r^2$ where t_c is average time for computation operation. Convolution is the most expensive operation in CNN. m^2 additions and m^2 multiplications require for each pixel in the image. The computation time for convolution with several kernels K and all kernels of size $n\,x\,m$ is $t_c(2m^2+1)N^2K$ .In parallel convolution approaches data is decomposed into subsets and distributed among P processes. Each process is responsible for subset of size N/p. The computation time for each process is

$$\mathrm{T}_{comp} = \frac{1}{p}T_1 = \frac{t_c(2m^2+1)N^2K}{p} \quad (5)$$

The processor communication cost is:

$$\mathrm{T}_{comp} = 2K(t_s + tbSN\frac{m-1}{2}) \quad (6)$$

Algorithm 4: Parallel SGD (parameters, data, k)

1: Shuffle the data set on all nodes in the cluster so that each node has a representative subset of original data set
2: **for** each node $i \in 1::k$ **do**
3: di $<-$ SGD (parameters; data)
4: **end for**
5: Aggregate from all machines d <- $(\frac{1}{k})$ 1 $[d_i]$ and ***return d***.

t_s denotes message startup time and tb represents transfer time per byte. S is total number of messages needing to be sent concurrently. Network topology, algorithm and communication pattern determines parameter S. The execution Time: $\mathrm{T}_{parallel} = t_c\frac{(2m^2+1)N^2K}{p} + 2t_sK + t_bK\mathrm{SN(m-1)}$ (7)

From above equations speedup S is: $\frac{\mathrm{p}}{1+\frac{2t_s s+t_b SN(m-1)}{t_c(2m^2+1)N^2P}}$ (8)

Computation Cost: The data set is distributed over nodes. Each node has data set on which SGD run locally. The cost of computation for SGD at each node is $o(n\,log\frac{1}{\sigma})$ where, n is the size of parameters. Therefore computation is $O(\frac{n}{q}logk) + O(n\,log\frac{1}{\sigma})$ with q nodes.

Communication Cost: Locally on each node SGD is computed. There is no communication cost require for SGD method. Once all the parameters are updated for each node locally, it's necessary to perform all to one communication messaging to master node where it will be averaged.

The communication cost will be: $O(\frac{kn}{B}) + O(LK)$. For broadcasting parameters the communication cost in the last step is i.e. computing average (all to one) $O(kp)$ as $pk(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots)$. Therefore, the total communication cost is $O(N\ k) + O(pk)$. Communication Time: The communication cost is $O\ (p\ log\ k)$. There are $log(k)$ rounds of parallel communication over the cluster. This requires $O(p)$ communication time. Convex loss function is strongly used in Parallel SGD which gives a unique minimum. To improve accuracy shuffling data sets between nodes at each run of SGD is carried out. The communication time will be $O(N)$ where $N = (D/K)$. the total communication cost will be $O(kN)$.

4. Results

The configuration of machine we used to perform analysis is machine having Processor: AMD Processor having 4 Core(s) CPU. The GPU card used is NVIDIAs GTX520 GeForce, 48 cores with compute capability. Table I summaries the specifications of GPU used in this research.

A. GPU speedup for convolution

There are two major computation phases' forward computation and backward computation. We observed the propagation time for these phases for fully connected layer as well convolutional layer for different sizes of image data. The result is presented in Fig. 1.

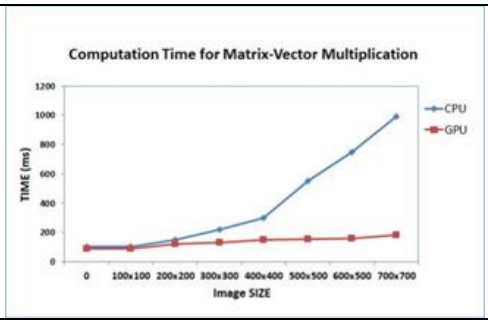

Fig. 1. Computation Time of CPU and GPU for Matrix-Vector Multiplication

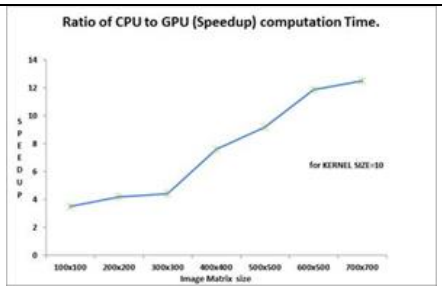

Fig. 2. Ratio of CPU to GPU (speedup)

In our second experiment the speedup achieved using GPU for different image sizes are observed and result is presented in Fig. 2.

B. GPU speedup on CNN

For this experiment, we designed the CNN model which has one fully connected layer and one convolutional layer. We observed processing time of the network with variants of network parameters. These parameters are: number of neurons in a layer, count of filters and size, etc. The input image size is 28x28 keeping the output size fixed. The architecture of CNN model is represented in format; e.g. 5-50-100 architecture where 10 indicate kernel size is of 5x5 for first layer, 50 numbers of neurons in convolution layer. 100 number of neurons in fully connected layer and 10 indicates total number of neurons in the last layer. The observations for learning of CNNs with CPU and GPU are presented in Fig. 3.

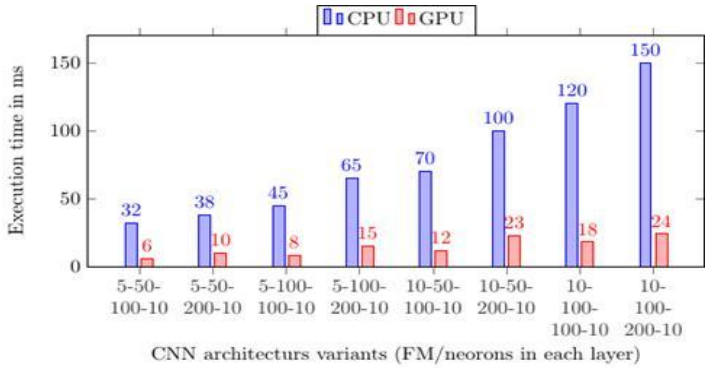

Fig.3 Training time comparison for CNN models with CPU and GPU

C. Speedup with Distributed Computing

The benchmarking of multi core distributed computing was done on WCE node of WCE private cloud. The cloud cluster allows up to 64 cores to work in parallel. However as WCE is busy that we could only 16 cores available. The benchmarking used test cases with 2 to 16 cores. We noted the time require for training on node. The Fig. 4 shows the result of it

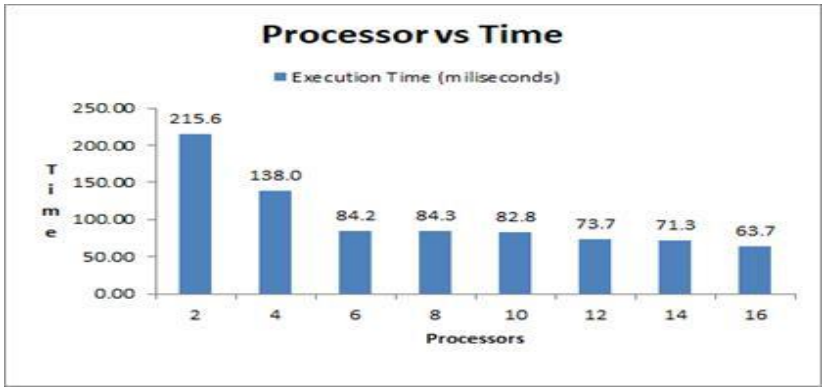

Fig. 4. Execution time with number of nodes

5. Analysis

In this section we will make performance analysis of CNN training methods in convolution layers. In table I we compared the speed of several characteristics. All parallel methods are better than serial methods. It proves the effectiveness of using GPU in experiment. Other observation is that storing input to shared memory consumes almost same time as naive parallel methods. This is due to size of kernel is much smaller than input image. The challenges with GPU computation are that fast memories are very limited. The idea is to store input in shared memory and convolution filters in global memory of GPU. Naive parallel convolution is 600times better than serial method. Compared to naive parallel in kernel+memory copy method 2 times speedup is achieved. The speedup compared is shown in Table II.

Layers	Serial	naive	Input in Shared memory	Input & filter In shared memory	Advanced block setting
Layer1	10033	4.07	3.99	4.82	3.89
Layer2	2205	9.27	8.95	8.02	5.08
Layer3	798	4.27	5.23	7.84	3.99
Layer4	908	6.06	6.86	11.21	5.55
Layer5	605	5.28	5.48	5.70	4.62
Total	**14549**	**29.47**	**30.51**	**37.59**	**23.11**

TABLE I. Time consumed in Convolutional Layers.

Layers	naive	Input in Shared memory	Input & filter In shared memory	Advanced block setting
speed compared to serial	826	630	529	701
Speed compared to naive parallel	/	0.96	0.82	1.26

TABLE II. SPEED COMAPRED TO PARALLEL METHOS WITH NAIVE METHOD IN CL

6. Conclusion

The paper focus is on Parallelization of Deep Convolutional Neural Network. The main objective is to improve performance training model based on Deep Convolutional Neural Network using parallel framework. The paper discussed time in milliseconds to measure the performance. The experiment is carried out with Facial expression recognition image data. In the proposed system; a facial expression recognition system has been introducing using Machine Learning Techniques such as classification using convolutional neural network algorithms. It takes large amount of data and millions of parameters to learn CNN. To improve the accuracy and efficiency of the network, it can achieve by making the model more complex and bigger in size. With growing sizes of model and larger data sets, it can become possible to extract complex features such as facial features for emotion classification tasks. But it can become costly due to more computation time for training. This task can be accelerated using ways to parallelize and distribute training phase. We have done analysis on speedup of convolutional layer of CNN. In distributed environment we split the model across different cores or data parallelism and analyze the cost of computation and communication cost. The focus of future work is to analyze method on system having multiple GPU cards

References

[1] 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.
[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le et al., “Large scale distributed deep networks,” in Advances in neural information processing systems, 2012, pp. 12231231.
[3] S. Ruder, “An overview of gradient descent optimization algorithms,” arXiv preprint arXiv:1609.04747, 2016.
[4] J.Schmidhuber, ”Deep learning in neural networks: An overview,” Neural networks, vol. 61, pp. 85117, 2015.
[5] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng,” On optimization methods for deep learning, “in Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011.
[6] Vishakh Hegde and Sheema Usmani, “Parallel and Distributed Deep Learning,”, 2016.
[7] Sunwoo Lee, Dipendra Jha, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao, “Parallel Deep Convolutional Neural Network Training by Exploiting the Overlapping of Computation and Communication,” IEEE 24th International Conference on High Performance Computing, 2017.
[8] Mouna Afif, Yahia Said, Mohamed Atri,” Efficient 2D Convolution Filters Implementations on Graphics Processing Unit Using NVIDIA CUDA”, I.J. Image, Graphics and Signal Processing, 2018.
[9] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri,V., Devanur, N. R., Ganger, G. R., and Gibbons, P. B. Pipedream: Fast and efficient pipeline parallel DNN training. CoRR, 2018.