Reducing the training time of deep learning models using synchronous SGD and large batch size

Salah Eddine Loukili
Faculty of Science and Technology, Laboratory VTE
Hassan First University of Settat
Settat, Morocco
s.loukili@uhp.ac.ma

Abdellah Ezzati
Faculty of Science and Technology, Laboratory VTE
Hassan First University of Settat
Settat, Morocco
abdellah.ezzati@uhp.ac.ma

Said Ben Alla
Faculty of Science and Technology, Laboratory VTE
Hassan First University of Settat
Settat, Morocco
said.ben.alla@uhp.ac.ma

Brahim Zraibi
Faculty of Science and Technology, Laboratory LAMSAD
Hassan First University of Settat
Settat, Morocco
b.zraibi@uhp.ac.ma

2022 International Conference on Intelligent Systems and Computer Vision (ISCV) | 978-1-6654-9558-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCV54655.2022.9806117

***Abstract*— Recently, deep learning research has demonstrated that being able to train big models improves performance substantially. In this work, we consider the problem of training a deep neural network with millions of parameters using multiple CPU cores. On a single machine with a modern CPU platform, training a benchmark dataset of Dogs vs Cats can take up to hours; however, distributing training across numerous machines has been seen to dramatically reduce this time. The current state of the art for a modern distributed training framework is presented in this study, which covers the many methods and strategies utilized to distribute training. We concentrate on synchronous versions of distributed Stochastic Gradient Descent, different All Reduce gradient aggregation algorithms, and best practices for achieving higher throughput and reduced latency, such as gradient compression and large batch sizes. We show that using the same approaches, we can train a smaller deep network for an image classification problem in a shorter time. Although we focus on and report on the effectiveness of these approaches when used to train convolutional neural networks, the underlying methods may be used to train any gradient-based machine-learning algorithm.**

Keywords—deep learning, distributed training, machine learning, convolutional neural network.

I. INTRODUCTION

Recently, in a wide range of applications, including speech recognition, computer vision, text processing, and natural language processing, deep learning has outperformed classical Machine Learning models in creating models to address complicated problems. Despite significant progress in customizing neural networks designs, there is still one major drawback: training big NNs is memory and time intensive. The training of NNs in a distributed way is one answer to this problem. The purpose of distributed deep learning systems (DDLS) is to scale out the training of big models by combining the resources of several separate computers. As a result, several of the DDLS presented in the literature use various ways to implement distributed model training [1]. Training times have increased substantially as models and datasets have become more sophisticated, sometimes weeks or even months on a single GPU. To address this issue, two techniques proposed by many researchers for scaling out big deep learning workloads are model and data parallelism. Model parallelism seeks to transfer model execution stages onto cluster hardware, whereas data-parallel methods treat collaborative model training as a concurrency/synchronization challenge [1]. The main idea behind data parallelism is to enhance the overall sample throughput rate by duplicating the model over several computers and performing backpropagation in parallel to acquire more information about the loss function more quickly. It is achieved in the following way. Each cluster node begins by downloading the current model. Then, utilizing its parallel data assignment, each node executes backpropagation. Finally, the various results are combined and merged to create a new model [2].

II. DISTRIBUTED TRAINING ALGORITHMS

In data parallelism, and for a distributed setting, distributed SGD algorithms can be roughly classified into two variants synchronous and asynchronous [1,3].

Asynchronous SGD is a distributed gradient descent algorithm that allows multiple model replicas to be trained in parallel on different nodes using different data subsets. Each model replica requests global weights from parameter servers, runs a mini batch to calculate gradients, and then sends them back to the parameter server, to finally update the global weights.

In synchronous SGD, the master node aggregates these gradients by averaging them to form the new global set of gradients for the weight update step. These global gradients use the same formula as the single machine SGD to update the local weights of each node, after which nodes can begin processing the next batch of data. Because this entire procedure is analogous to computing a forward pass and backpropagation step on a single machine using a single mini batch of data, synchronous SGD guarantees convergence Research shown in the latest studies that synchronous methods scale and provide better performance than asynchronous methods [1, 3, 4]. For that, in the next experiment we will use the All Reduce SGD, a synchronous SGD variant, and try to improve the training time by scaling the batch size.

III. EXPERIMENT

In this experiment, we will train a CNN model on a modern size dataset on a single machine with multiple cores, simulating a distributed setting. Each time, the batch size (128,

978-1-6654-9558-5/22/$31.00 ©2022 IEEE

256, 512, 1024) will be increased to see how it affects training time and test accuracy.

A. Dataset

Similar to the development of image classification algorithms, in this study our approach is developed and tested on publicly available data. This dataset is provided as a subset of photos from a much larger dataset of 3 million manually annotated photos. The dataset was developed as a partnership between Petfinder.com and Microsoft. The Dogs vs. Cats dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or cat. The dataset contains 25,000 images of dogs and cats. We choose 22,500 images to be used for training and 2500 for testing. Each image in the dataset has a different size, therefore the first step in the preprocessing phase is to resize the images to 256*256 pixels, then we scale the decoded values for each pixel by 1/255 (values for the 3 color channels will be between 0 and 255). The dataset is also divided into batches and we will be using different values for the batch size.

B. Model

Based on various experiments we did in this study, we will be presenting the best combination of architecture / configuration that we have achieved both in terms of accuracy and training time. Table I below explains in detail the chosen model architecture.

TABLE I. MODEL ARCHITECTURE.

Layer	Units/ filters	Activation	Kernel size	Pool size	Dropout rate
Conv2D	32	relu	(3, 3)		0
MaxPooling2D				(2, 2)	
Conv2D	64	relu	(3, 3)		0
MaxPooling2D				(2, 2)	
Conv2D	128	relu	(3, 3)		0
MaxPooling2D				(2, 2)	
Conv2D	128	relu	(3, 3)		0
MaxPooling2D				(2, 2)	
Flatten					
Dense	512	relu			0.25
Dense	256	relu			0.5
Dense	1	sigmoid			0

The model is trained using the RMSprop optimizer with learning rate value of 0.001 and a Binary cross entropy loss function. Accuracy is used to judge the performance of the model.

C. Hardware

In this experiment, we train our model on a virtual machine in the Google Cloud Platform with the following hardware configuration.

TABLE II. HARDWARE CONFIGURATION.

Memory (GB)	**8 - 60 GB**
CPU Platform	**Intel SkyLake**
CPU Cores	**1 - 96**
Tensorflow	**v2.1.0**

IV. RESULTS AND DISCUSSION

The next 4 figures shows the results of 4 experiments where the variable is the batch size, in each experiment, we train the model on the batched dataset and record the training time and test accuracy, then we increase the CPUs count and train it again.

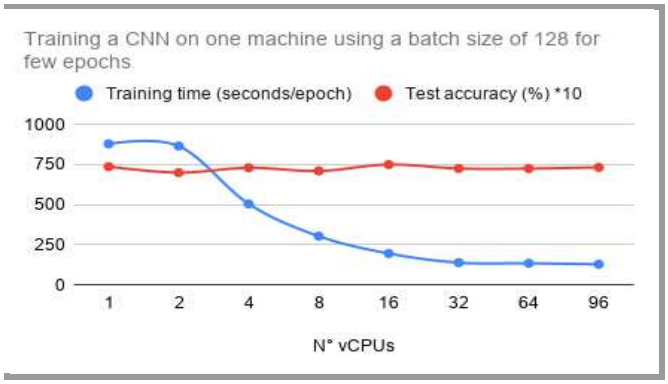

Fig. 1. Training a CNN model with batch size = 128

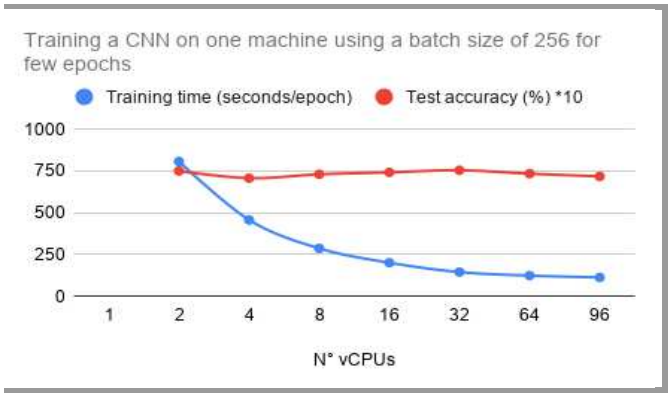

Fig. 2. Training a CNN model with batch size = 256

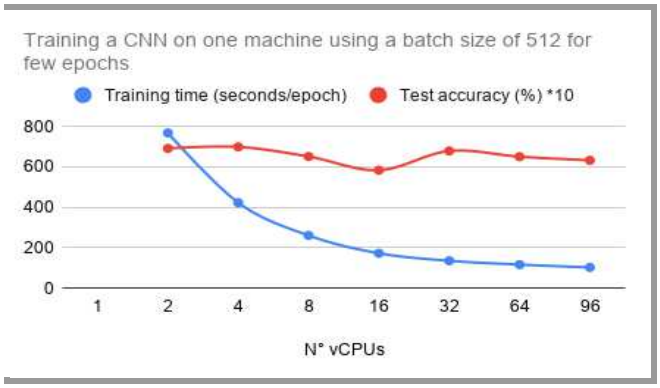

Fig. 3. Training a CNN model with batch size = 512

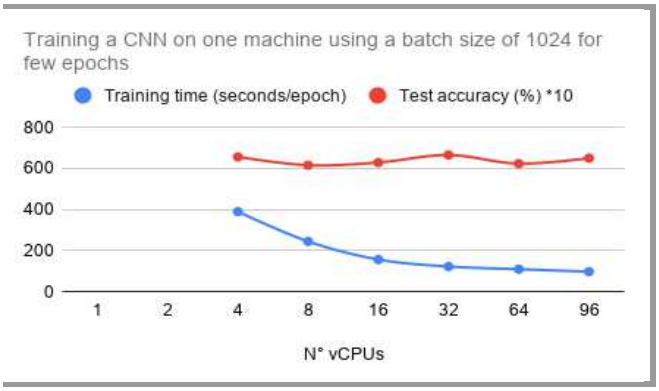

Fig. 4. Training a CNN model with batch size = 1024

In Figure 1, 2, 3 and 4 we trained the model with a batch size of 128, 256, 512 and 1024 on various hardware settings, increasing the number of CPU cores each time (i.e., distributing the model training) and measuring the training time and test accuracy. We notice that the training duration decreases until it reaches a limit. We also notice that the test accuracy did not decrease with time and remained rather constant.

TABLE III. TRAINING A CNN ON ONE MACHINE USING DIFFERENT BATCH SIZES AND MEASURING THE T RAINING TIME IN SECONDS/EPOCH

N° CPUs / Batch size	1	2	4	8	16	32	64	96
128	**878**	**865**	**504**	**303**	**195**	**138**	**134**	**128**
256	**N/A**	**806**	**456**	**286**	**200**	**144**	**123**	**112**
512	**N/A**	**806**	**456**	**286**	**200**	**144**	**123**	**112**
1024 **Increase wrt batch size=128**	**N/A**	**N/A**	**389** **+22.8%**	**244** **+19.4%**	**156** **+20%**	**122** **+11.5%**	**109** **+18.6%**	**97** **+24.2%**

Table III illustrates that increasing the batch size reduces training time while maintaining test accuracy throughout the studies. By just increasing the batch size by a factor of 4, we were able to minimize the training time from 128 seconds per epoch to 97 seconds per epoch (up to **24,2%** gain in training time).

V. CONCLUSION

Data parallelism techniques using asynchronous algorithms have been widely employed to expedite the training of deep learning models. To enhance data throughput while ensuring computing efficiency in each worker, scale up techniques rely on tight hardware integration. Increasing the batch size, on the other hand, may results in a loss in test accuracy, which may be mitigated by a number of recent concepts, such as increasing the learning rate throughout the training process and using a learning rate warm up technique.

REFERENCES

[1] Langer, M., He, Z., Rahayu, W., & Xue, Y. (2020). Distributed training of deep learning models: A taxonomic perspective. IEEE Transactions on Parallel and Distribute d Systems, 31(12), 2802 2818.

[2] Shi, S., Zhou, X., Song, S., Wang, X., Zhu, Z., Huang, X., Jiang, X., Zhou, F., Guo, Z., Xie, L., & others (2021). Towards scalable distributed training of deep learning on public cloud clusters. Proceedings of Machin e L earning and Systems.

[3] Chahal, K., Grover, M., Dey, K., & Shah, R. (2020). A hitchhiker's guide on distributed training of deep neural Journal of Parallel and Distributed Computing, 137, 65 76.

[4] Birnie, C., Jarraya, H., & Hansteen, F. (2021). An introduction to distributed training of deep neural networks for segmentation tasks with large seismic datasets. arXiv preprint arXiv:2102.13003.

[5] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., & others (2012). Large scale distributed deep networks. Advances in neural information processing systems, 25, 1223 12