Convergence time analysis of Asynchronous Distributed Artificial Neural Networks

Mauro D. L. Tosi*
University of Luxembourg
Esch-sur-Alzette, Luxembourg
mauro.dalleluccatosi@uni.lu

Vinu E. Venugopal
University of Luxembourg
Esch-sur-Alzette, Luxembourg
vinu.venugopal@uni.lu

Martin Theobald
University of Luxembourg
Esch-sur-Alzette, Luxembourg
martin.theobald@uni.lu

ABSTRACT

Artificial Neural Networks (ANNs) have drawn academy and industry attention for their ability to represent and solve complex problems. Researchers are studying how to distribute their computation to reduce their training time. However, the most common approaches in this direction are synchronous, letting computational resources sub-utilized. Asynchronous training does not have this drawback but is impacted by *staled* gradient updates, which have not been extended researched yet. Considering this, we experimentally investigate how stale gradients affect the convergence time and loss value of an ANN. In particular, we analyze an asynchronous distributed implementation of a Word2Vec model, in which the impact of staleness is negligible and can be ignored considering the computational speedup we achieve by allowing the staleness.

ACM Reference Format:
Mauro D. L. Tosi, Vinu E. Venugopal, and Martin Theobald. 2022. Convergence time analysis of Asynchronous Distributed Artificial Neural Networks. In *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD) (CODS-COMAD 2022), January 8–10, 2022, Bangalore, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3493700.3493758

1 INTRODUCTION

Artificial Neural Networks (ANNs) have had tremendous success in the recent past, in particular, due to their capability to solve complex problems. Those problems are generally solved using large models coupled with a high volume of training data. Researchers have achieved excellent speedups on the training of large models by considering data parallelism, where a subset of the training data is distributed among multiple compute nodes. However, common model optimizers like Stochastic Gradient Descent (SGD) are sequential in nature. Prior approaches such as [3] perform a *synchronous* parallel computation of SGD on large mini-batches. Nevertheless, a larger mini-batch size can negatively impact the model's ability of generalization [1]; and the synchronous SGD distribution increases the idle time of the compute nodes due to the necessity of synchronization barriers [2].

Another line of research in this direction is to explore *asynchronous* approaches where nodes would asynchronously update themselves, ignoring the potential *staleness* of their updates. Following this notion, Niu *et al.* [2] developed Hogwild, which used asynchronous application of SGD on multicore machines. Later, Zhang, Hsieh, and Akella [6] developed Hogwild++, which explores how to overcome Hogwild's bottleneck of having all worker nodes reading/updating a single central weight matrix. However, these approaches achieve a nearly optimal rate of convergence only when the associated optimization problem is *sparse*. In addition, the impact of updating an ANN with stale gradients on the model's convergence time and loss value is not well studied experimentally.

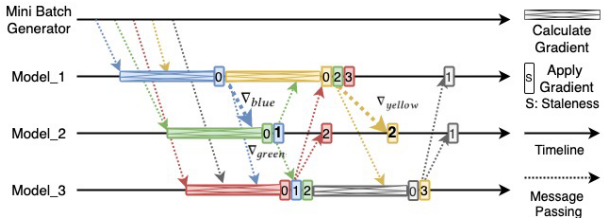

Figure 1: Asynchronous SGD distribution.

In this paper, we consider a distributed asynchronous architecture for training an ANN model that helps us to systematically study the *asynchronicity* and *staleness* notions effectively. This setting allows models to learn from sharded input data simultaneously. It also lets them loosely synchronise themselves without synchronisation barriers, making the training process much faster than the synchronous approach. However, theoretically, the convergence of a model in an asynchronous SGD approach depends on the staleness of model updates. Within the scope of this paper, we define *staleness* in this setting. Then we analyse its impact on the convergence time and loss value of the model and report our observations on this topic based on our experiments with Word2Vec.

2 ASYNCHRONICITY AND STALENESS

We perform the asynchronous distributed training of an ANN model by training n models with different batches of data across N computing nodes. Each model represented by a weight matrix w, would asynchronously exchange the gradient $\nabla_{w,x}$, calculated from the batch of data (x), with the other $n-1$ models. All models are initially assigned with the same weight matrix w_0. Considering that the model updates are gradient summations, thus commutative operations, all the n models would eventually get synchronized. However, the convergence of these models largely relies on the following factors: the degree of staleness of the incoming gradients; and how the models consider those gradients while updating themselves [2].

In an asynchronous setting, we end up updating each model with gradients that were calculated based on a different weight

*Doctoral candidate at the University of Luxembourg

CODS-COMAD 2022, January 8–10, 2022, Bangalore, India

ACM ISBN 978-1-4503-8582-4/22/01.
https://doi.org/10.1145/3493700.3493758

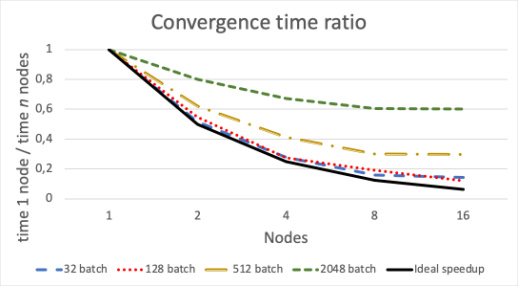

Figure 2: Convergence time ratio using 4 Word2Vec models per node varying the mini-batch sizes on 1 and n nodes.

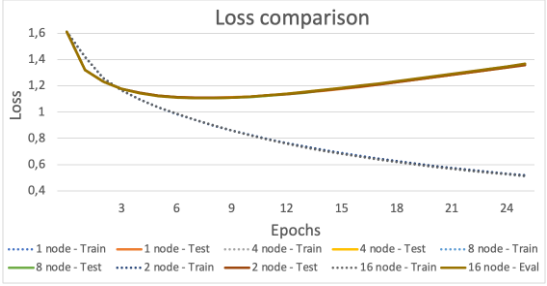

Figure 3: Train and test losses of the distributed Word2Vec usecase using 128 minibatch size on 4 models per node.

matrix. For example, if w_a is the current weight matrix of Model A, then on receiving two new gradients the model is updated as $w'_a = w_a + \nabla_{w_b,x_i} + \nabla_{w_c,x_j}$, where w_b and w_c are weight matrices of two remote models, B & C, respectively, and x_i & x_j are the respective training data they considered. Consequently, the SGD guarantees of convergence to a local minimum are loosened accordingly to the staleness of the gradients used for updating the model.

At Model A, the staleness of a gradient $\nabla_{w,x}$ calculated by Model B can be represented as $S(\nabla_{w,x}, B, A)$. We can define this staleness in terms of two gradient sets: (1) $G_{B \rightsquigarrow \nabla_{w,x}}$, the set of gradients processed by Model B at the time that it calculated the gradient using the batch x; and (2) $G_{\nabla_{w,x} \rightsquigarrow A}$, the set of gradients processed by Model A at the time it applies the gradient. Intuitively, we define the staleness here as the symmetric difference of both gradient sets. That is, $G_{B \rightsquigarrow \nabla_{w,x}} \triangle G_{\nabla_{w,x} \rightsquigarrow A}$. The arrow, $\rightsquigarrow$, denotes if the gradient is an incoming or an outgoing gradient. Fig. 1 shows the distribution of the calculation and application of the gradients, along with their staleness values.

In Fig. 1, we depict a 4 node infrastructure in which the *Mini Batch Generator* (MBG) is a node that generates and asynchronously sends mini batches of training examples to Models 1 to 3. All models are initialised with the same weights. Each model calculates new gradients from the incoming batches and applies the gradients to itself, and incorporate the gradients it receives from the remote models. Take as example $Model_2$ that calculates the green gradient, ∇_{green}, apply it to itself and sends a copy of it to the other models. Then, $Model_2$ applies ∇_{blue} that it received from $Model_1$. Before calculating the blue gradient, $Model_1$ did not apply any gradient to itself, thus, $G_{1 \rightsquigarrow blue} = \{\}$. Also, $Model_2$, immediately before applying ∇_{blue}, had already applied the green gradient to itself, thus $G_{blue \rightsquigarrow 2} = \{\nabla_{green}\}$. Therefore, the staleness of ∇_{blue} when applied to $Model_2$ is 1 because $S(blue, 1, 2) = G_{1 \rightsquigarrow blue} \triangle G_{blue \rightsquigarrow 2} = 1$.

3 CONVERGENCE ANALYSIS & DIRECTIONS

Experimental setting. By following the above-mentioned asynchronicity and staleness notions, we trained a Word2Vec model[1] using a fixed learning rate and 1% of Wikipedia (85% training and 15% test) on an HPC [4] environment programmed using AIR [5], and Tensorflow C API as the NN framework.

Convergence analysis. Fig. 2 shows the reduction in the convergence time of the Word2Vec use case when asynchronously distributing its training. We achieved a near-optimal reduction in the convergence time for smaller mini-batches, whereas the training with larger ones is less time-consuming—the margin of speed up has reduced. In Fig. 3, we show that the losses on the *training* set and the *test* set remain the same while varying the number of nodes. This behaviour is evident from overlapping plots in the figure. While using 1 node (4 models), we had an average staleness of 3. And, while using 16 nodes (64 models), we had an average staleness of 55, which did not impact the model convergence rate. We also ran this same experiment using 32, 512, and 2048 mini-batches, and we observed this same pattern of overlap of the losses. However, while increasing the staleness further, even for sparse problems as Word2Vec, we eventually expect a difference in this trend. In general, these results indicate that the impact of staleness is negligible and can be ignored for sparse problems as Word2Vec considering the speed up achieved by allowing the staleness.

Future directions. We plan to investigate the behaviour of training a Word2Vec model with more nodes until observing the negative effects of the staled gradients. Moreover, we intend to explore how dense networks behave in an asynchronous distributed scenario. At last, we plan to perform these future studies using GPUs to approximate our experiments to current industry implementations.

[1] https://www.tensorflow.org/tutorials/text/word2vec

ACKNOWLEDGMENTS

Funded by the Luxembourg National Research Fund under the PRIDE programme (PRIDE17/12252781).

REFERENCES

[1] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France*. OpenReview.net.

[2] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, Vol. 24. Curran Associates, Inc.

[3] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[4] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. 2014. Management of an Academic HPC Cluster: The UL Experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. IEEE, Bologna, Italy, 959–967.

[5] Vinu E. Venugopal, Martin Theobald, Samira Chaychi, and Amal Tawakuli. 2020. AIR: A Light-Weight Yet High-Performance Dataflow Engine based on Asynchronous Iterative Routing. In *32nd IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2020*. IEEE, 51–58.

[6] Huan Zhang, Cho-Jui Hsieh, and Venkatesh Akella. 2016. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 629–638.