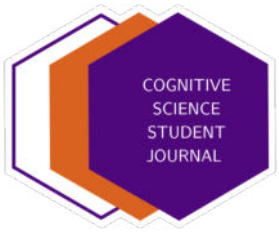

Cognitive Science Student Journal
Osnabrück University

Stochastic Gradient Descent - But Make it Parallel!

Julia Hattendorf

Number 2

Hattendorf, J. (2024). Stochastic Gradient Descent - But Make it Parallel! *Cognitive Science Student Journal 2024*, *2*. 1-6.

This title can be downloaded at:
http://cogsci-journal.uni-osnabrueck.de

Institut für Kognitionswissenschaft
Universität Osnabrück
49069 Osnabrück
Germany
https://www.ikw.uni-osnabrueck.de

Storage and cataloging done by Osnabrück University

Abstract

This literature review aims to provide an introduction to the use of distribution techniques in neural network training, in particular, the application of data parallelization to Stochastic Gradient Descent (SGD). The primary methods for parallelizing SGD can be classified as either asynchronous (AsyncSGD) or synchronous (SyncSGD). Both approaches, as well as their issues and variants, will be explained and compared. Notably, SyncSGD ensures convergence but is slower due to its synchrony, while AsyncSGD offers faster updates but may lead to convergence challenges due to its asynchrony. The choice between the two hinges on the trade-off between speed and convergence stability, making it context-dependent.

1 Introduction

Deep learning is a subcategory of machine learning that involves training artificial neural networks with multiple layers, so-called deep neural networks, to recognize patterns and make predictions from large and complex datasets. To train a network, an error is computed based on its prediction and then used to adjust the network's parameters through backpropagation. However, the training of deep learning models is often time-consuming and resource-intensive due to the handling of big datasets and the calculations involving numerous matrix multiplications. This can be mitigated by dividing the work among multiple workers, e.g., servers, processors, or cores. Such divide-and-conquer strategies are known as distribution techniques. Applying them to the training of neural networks is termed distributed training or distributed learning. In this work, I will focus on the popular training algorithm Stochastic Gradient Descent (SGD) (Ketkar, 2017) and introduce how its workflow can be parallelized. The main approaches to do so can be categorized in a synchronous manner, synchronous SGD (SyncSGD), or in an asynchronous manner, asynchronous SGD (AsyncSGD). SyncSGD and AsyncSGD are introduced on their own, along with their issues and variants, after which I will make a comparison between the two.

2 Background

The following section will introduce two main concepts needed for the understanding of distributed SGD. Firstly, the training algorithm SGD itself, and secondly, how distributed learning works, particularly using the data parallelism approach.

2.1 Introduction to Stochastic Gradient Descent

The main difference between SGD and vanilla gradient descent is the batching of data: While vanilla gradient descent calculates the gradients for the model based on every training example in the dataset, SGD only uses small subsets of the dataset. The subsets are either taken to be single examples or mini-batches of a dataset. These sub-sets are sampled at random, which is why the training algorithm is called "stochastic" (Ketkar, 2017). In a forward pass, the mini-batches are propagated through the network and the error based on the output is calculated. In a backward pass, the error is backpropagated through the model to yield gradients which are used to update model weights. An iteration is defined as a sub-set passing through the forward and backward phases, while an epoch is defined as a forward-backward run over the whole training dataset. The model is trained over a number of epochs until the predefined number of epochs is reached or until the model converges, meaning the performance of the model is no longer changing significantly. This implies that the updates to the model parameters have become sufficiently small, indicating that the algorithm reaches a stable and optimal solution for the model's parameters. Convergence is a critical

aspect in training machine learning models, as it signifies that the model has learned the underlying patterns in the data and is less likely to undergo significant changes with further iterations.

SGD offers multiple advantages compared to vanilla gradient descent. Due to only needing mini-batches, SGD is able to handle large datasets while keeping computational time and memory requirements low. By randomly selecting mini-batches of data, SGD introduces randomness into the optimization process, which can help the algorithm avoid getting stuck in local minima and saddle points (Ketkar, 2017). Although SGD is faster than vanilla gradient descent, its speed is constrained due to processing the data in a serial manner (Dean et al., 2012), which is a problem that can be overcome by introducing data parallelism.

2.2 Introduction to Data Parallelism

In order to gain insight into the parallelization of SGD, an introduction to distributed learning is necessary, with data parallelism (refer to Figure 1) standing out as a popular approach in this domain (Chahal et al., 2020). As the name suggests, the data is divided among worker nodes which will run the algorithm in parallel. Each worker, enumerated with i, is responsible for calculating its own gradients ∇w_i with its model replica. The gradients are then pushed onto a parameter server, where the new global gradients ∇w are calculated, also known as gradient accumulation, and broadcasted back to the workers. Because each worker receives the same gradients, the model's weights remain consistent across workers.

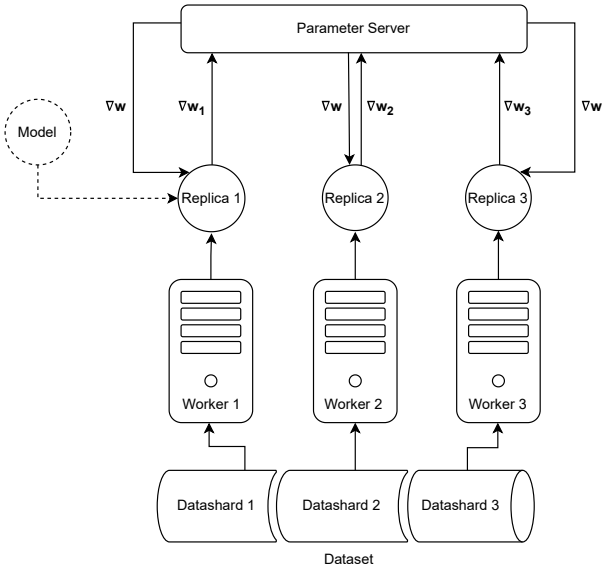

Figure 1: Dataflow in Data Parallelism. Based on Dean et al. (2012).

To summarize, SGD differs from vanilla gradient descent by using random mini-batches of data, offering efficiency in handling large datasets and avoiding local minima. To understand the parallelization of SGD, I introduced data parallelism, which involves distributing data among worker nodes for parallel processing.

3 Synchronous SGD

The following chapter will delve into one of the two ways, we can apply the concept of data parallelism to SGD. In SyncSGD (see Figure 2) a synchronization barrier is introduced into the process by making

the parameter server wait for all workers to complete computing before calculating the new gradients and updating the workers (Chahal et al., 2020).

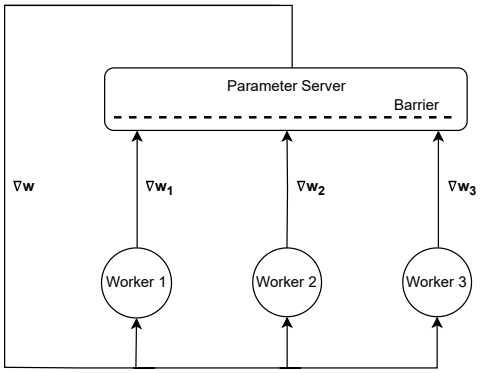

Figure 2: Exchange of local gradients ∇w_i for i workers and global gradients ∇w in SyncSGD. Based on Curci et al. (2021).

3.1 Issues

Since the parameter server has to wait for each worker to finish its computation, the algorithm is running only as fast as the slowest machine, which is known as straggler. The reasons for a machine being a straggler can be hardware-related, e.g., machine crashes (Chen et al., 2017), as well as network-related, e.g., slow network conditions or failed network requests (Chahal et al., 2020).

Additionally, needing all the workers to finish computing before continuing, means that even one worker failing in doing so would hold the whole algorithm. Therefore, the algorithm has a single point of failure (Chahal et al., 2020). Further, a lot of workers trying to communicate with one parameter server simultaneously leads to bandwidth problems (Chahal et al., 2020), which result in delays in transmitting, thus slowing down the convergence of the training algorithm. In case the parameter server fails, the system may also encounter a single point of failure problem. (Chahal et al., 2020)

3.2 Variations

The following variations of SyncSGD try to tackle these issues in different ways.

In order to deal with stragglers, Chen et al. (2017) propose to partition the amount of workers into N main workers and b backup workers. After any N workers have uploaded their gradients to parameter servers, the new global gradients are computed, and the gradients of the remaining b workers are discarded.

The 1-bit algorithm, introduced by Seide et al. (2014), is directly tackling the bandwidth problem by reducing the amount of data that gets transferred between workers and the parameter server. This is achieved through compressing each gradient to a single bit, specifically by quantizing it to its sign. To ensure that the SGD converges it is important to keep track of the quantization errors. These errors are accumulated and added to the gradients of the subsequent mini-batch before compression. Consequently, all gradients are eventually incorporated fully into the parameter server, albeit delayed across iterations of mini-batches.

To sum up, in SyncSGD, the parameter server must wait for all workers to finish computations before updating, causing potential slowdowns due to stragglers. Approaches like partitioning workers or the 1-bit algorithm aim to address these challenges by discarding gradients from straggling workers or compressing them to reduce bandwidth, ensuring convergence more efficiently.

4 Asynchronous SGD

Another way to parallelize SGD is AsyncSGD (see Figure 3). Compared to SyncSGD, AsyncSGD has given the workers a new level of independence. Each worker node asynchronously performs its own gradient updates and periodically synchronizes its parameters with a central parameter server. When one node fails, the remaining nodes can continue processing, removing the straggler problem and the single point of failure among workers provided by SyncSGD.

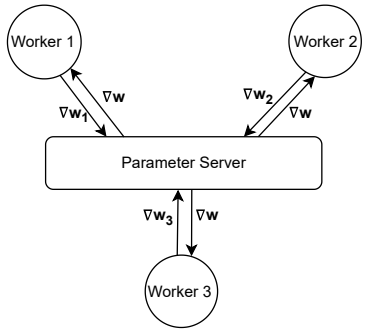

Figure 3: Exchange of local ∇w_i for i workers and global gradients ∇w in AsyncSGD. Based on Curci et al. (2021).

4.1 Issues

While AsyncSGD solves the biggest problem present in SyncSGD, the straggler problem, it introduces novel, potentially more severe issues. Since model updates may have occurred while a worker was computing its gradient, the resulting gradients are frequently computed with outdated parameters and are subsequently known as "stale" gradients. Thus, the convergence of AsyncSGD cannot be guaranteed (Chahal et al., 2020).

Additionally, similar to SyncSGD many workers communicate with a central parameter server, which leads to bandwidth problems (Jin et al., 2016) and a single point of failure.

4.2 Variations

Followingly, novel variations of AsyncSGD addressing the aforementioned limitations of AsyncSGD were proposed.

In order to deal with the stale gradients problem, the staleness-aware AsyncSGD (Zhang et al., 2015) attempts to factor the staleness into the computation itself. The staleness of the gradient is quantized, as the amount of times the weights used to compute said gradient have been updated. The learning rate is then divided by this staleness factor. The authors proved that with their staleness-dependent learning rate, AsyncSGD can converge at the same rate as SyncSGD (Zhang et al., 2015). In the paper by Odena (2016) the issue regarding staleness was further elaborated. The authors criticize that Zhang et al. (2015) do not differentiate between the quality of gradients and, thus, redefine staleness according to moving averages of gradient statistics.

Instead of combating stale gradients, Downpour SGD (Dean et al., 2012) attempts to increase the efficiency of AsyncSGD and addresses the bandwidth problem. The authors propose to divide the parameter server into parameter server shards, which operate independently. This allows for multiple parameter-worker transactions at the same time, and it would support a larger number of workers than a single parameter server. Since the parameter shards are not synced, parameters on a

shard might change more or less than on other shards, introducing additional stochasticity into the process and resulting in potentially inconsistent parameters across servers. The authors acknowledge that they cannot ensure convergence, but they argue that, based on observations, the lowering of requirements for consistency can be quite effective.

5 SyncSGD vs AsyncSGD

Having described SyncSGD and AsynSGD, I will now compare the two approaches of synchronizing SGD regarding their drawback and variations.

SyncSGD's most obvious issue is the straggler problem. While Chen et al. (2017) mitigates the effects of stragglers by introducing backup workers, AsyncSGD solves the problem completely by eliminating the barrier of waiting on all workers. This, however, leads to the problem of stale gradients, as gradients uploaded by a relatively slow worker would be outdated but are still used in the computation, leading to worse overall results. This problem of stale gradients is addressed in AsyncSGD algorithms like staleness-aware AsyncSGD (Odena, 2016; Zhang et al., 2015). It is also notable that SyncSGD has a guarantee to converge while AsnycSGD does not (Odena, 2016), making SyncSGD more reliable.

Both Sync- and AsyncSGD rely on a central parameter server, and therefore, are facing the single point of failure problem. Although, Downpour SGD calls for multiple server shards, and one shard could, in theory, take over the work of another, Chahal et al. (2020) argue that due to the tree-like hierarchy of the servers, the single point of failure problem is not omitted. For the workers, however, AsyncSGD eliminates the single point of failure problem, since the remaining workers continue computing (Dean et al., 2012).

Finally, bandwidth problems are present in both variants, due to the funneling of data from a lot of workers to a single server. These can be solved by lowering the amount of data to transfer, as seen in synchronous 1-bit SGD (Seide et al., 2014), or by introducing multiple server shards that act independently from another, as seen in the asynchronous Downpour SGD (Dean et al., 2012)

To summarize, while SyncSGD guarantees convergence, it faces challenges due to stragglers and a single point of failure among workers. On the other hand, AsyncSGD eliminates these issues, but it introduces the problem of stale gradients and lacks a convergence guarantee.

6 Conclusion

Distributed learning facilitates the training of neural networks, in particular with the effective use of resources and the speed up of neural network training by distributing computation across multiple nodes. To illustrate the application of distributed learning, I chose the training method SGD and the distribution technique data parallelism. After explaining SGD and data parallelism, I showed how data parallelism can be applied to SGD, resulting in a synchronous variant, SyncSGD, and an asynchronous variant, AsyncSGD. Finally, after comparing the drawbacks of both algorithms, it is clear there is no one-fits-all solution. SyncSGD is less efficient than AsyncSGD regarding speed and usage of resources, but is guaranteed to converge. While AsyncSGD's added stochasticity may help avoid local optima, it also runs the risk of making the training algorithm never converge. Although the optimal option varies depending on the specifics of the task, both provide significant speedups over non-parallelized SGD. To conclude, this literature review has provided insight into distributed learning. As the demand for more powerful and scalable machine learning solutions continues to rise, I expect that distributed learning will become essential to push the limits of what is possible in the field of artificial intelligence.

References

Chahal, K. S., Grover, M. S., Dey, K., & Shah, R. R. (2020). A hitchhiker's guide on distributed training of deep neural networks. *Journal of Parallel and Distributed Computing*, *137*, 65–76. https://doi.org/10.1016/j.jpdc.2019.10.004

Chen, J., Pan, X., Monga, R., Bengio, S., & Jozefowicz, R. (2017). *Revisiting distributed synchronous sgd.* arXiv. https://doi.org/10.48550/ARXIV.1604.00981

Curci, S., Mocanu, D. C., & Pechenizkiyi, M. (2021). *Truly sparse neural networks at scale.* arXiv. https://doi.org/10.48550/arXiv.2102.01732

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q., & Ng, A. (2012). Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

Jin, P. H., Yuan, Q., Iandola, F., & Keutzer, K. (2016). *How to scale distributed deep learning?* arXiv. https://doi.org/10.48550/ARXIV.1611.04581

Ketkar, N. (2017). Stochastic gradient descent. In *Deep learning with python: A hands-on introduction* (pp. 113–132). Apress. https://doi.org/10.1007/978-1-4842-2766-4_8

Odena, A. (2016). *Faster asynchronous sgd.* arXiv. https://doi.org/10.48550/arXiv.1601.04033

Seide, F., Fu, H., Droppo, J., Li, G., & Yu, D. (2014). 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. *Proc. Interspeech 2014*, 1058–1062. https://doi.org/10.21437/Interspeech.2014-274

Zhang, W., Gupta, S., Lian, X., & Liu, J. (2015). *Staleness-aware async-sgd for distributed deep learning.* arXiv. https://doi.org/10.48550/arXiv.1511.05950

About the Journal

The 'Cognitive Science Student Journal' aims at giving its readers an insight into current research and cutting-edge topics at our institute from a student perspective as well as students a platform to publish their work. Its editorial board consists of seminar participants and instructors of the Institute of Cognitive Science.

Cognitive Science is taught as an interdisciplinary research field at University Osnabrück, investigating cognition and the mind as a joint research effort of Artificial Intelligence, Neuroscience, Computational Linguistics, Psychology, Neuroinformatics, and Philosophy of Mind.

The journal can be accessed via:
http://cogsci-journal.uni-osnabrueck.de

Find us on social media:
https://www.instagram.com/cogscistudentjournal/
https://www.linkedin.com/company/cognitive-science-student-journal/

Number 2,

Cognitive Science Student Journal 2024