Exploiting data distribution in distributed learning of deep classification models under the parameter server architecture.

Nikodimos Provatas
supervised by Prof. Nectarios Koziris
National Technical University of Athens
Athens, Greece
nprov@cslab.ece.ntua.gr

ABSTRACT

Nowadays, deep learning models are used in a wide range of applications, including classification and recognition tasks. The constant growth on the data size has led to the use of more complex model architectures for creating neural network classifiers. Both the model complexity and the amount of data usually prohibit the training on a single machine, due to time and memory limitations. Thus, distributed learning setups have been proposed to train deep networks, when a vast amount of data is available. One such common setup follows the parameter server approach, where worker tasks compute gradients to update the network stored in the servers, often in a synchronous free manner. However, the lack of synchronization may harm the resulting model quality due to the effect of stale gradients, which are computed based on older model versions. In this PhD research, we aim to explore how asynchronous learning could benefit from data preprocessing tasks revealing hidden traits regarding the data distribution.

1 INTRODUCTION

In the recent years, deep learning has become a widely used part in a variety of applications. For example, in the image processing domain, neural networks are widely used in classification [20] and tagging [32] applications. Deep models are also widely used in other domains as speech recognition [13] or text classification [21].

All the aforementioned applications are actually classification tasks. In order to create accurate classifiers, a wide amount of data shall be used. As we increase the volume of the data available, more complex models are used in order to represent the patterns implied by the data. A common example of complex model architectures proposed are the ResNet [14] and Inception [29] architectures used on the Imagenet [9] dataset.

Both the increase in the model complexity and the amount of data available could prohibit model training in a single machine, since it would take numerous hours or days to create a generic and reliable model. For instance, without the use of accelerators as GPUs, Stanford's Dawnbench [6] took more than 10 days to train a ResNet-152 model on the Imagenet dataset [2]. Thus, there have been proposed distributed architectures that could be used in order to speed up the training process. Depending on what is decided to be distributed, multiple solutions could be adopted. The most common approach, especially when it comes to big data, is to distribute the available data to various workers, following data parallel learning architectures [8]. The most prominent setups of such distributed learning setups are met under the parameter server [30] and the all-reduce [7] architectures. Widely used deep learning systems, as Google TensorFlow [3], Apache MXNet [5] and PyTorch [25] have adopted the concept of distributed learning following one or more the aforementioned architectures.

While distributed learning enables faster deep neural network training, each distributed approach introduces some new issues that might harm either the training speed or the quality of the resulting model. For example, all-reduce approaches introduce synchronization overheads on each training step. On the other hand, while in the case of parameter server architecture such overheads are not met, as workers usually operate in an asynchronous manner [18]. Thus, stale gradients effects may occur, which could either delay model convergence or lead the model's training loss function to diverge [23]. However, we believe that if data are wisely used, such phenomena could be overcome.

In this PhD research, we focus on studying how the distributed deep learning process could benefit from the distribution of the training data, especially under the parameter server architecture. Data preprocessing techniques can be used to obtain an a-priori knowledge of the data domain, which could be beneficial in the training process . We aim to study and propose systematic ways on how the data should be assigned to the available training worker nodes. Furthermore, we will also study whether random or algorithmic access patterns on data are preferable during the training process, focusing on the distributed case. Considering such techniques, we aim for the training to be less sensitive to undesirable effects that appear in asynchronous distributed learning setups.

The rest of this papers is organized in four sections. At first, in Section 2 we refer to any related background knowledge necessary to easily understand the paper. Section 3 follows with a discussion on how former knowledge of the data distribution could benefit the learning process, especially under a parameter server setup. Finally, the paper concludes with Section 4 which outlines the steps that will be followed to complete this research.

2 BACKGROUND

2.1 Optimization Related Preliminaries

In the context of classification problems [12], a neural network with weights represented by the vector $\vec{w}$, is considered to approximate a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that, given an input feature vector $\vec{x}$, could be used to classify it to a category y, i.e. $y = f(\vec{x}; \vec{w})$. Given a set of feature vectors $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ and their corresponding category labels $y_1, y_2, ..., y_n$, the function f used to describe the neural network can be identified by using the appropriate vector $\vec{w}$ derived as the

Proceedings of the VLDB 2021 PhD Workshop, August 16th, 2021. Copenhagen, Denmark.

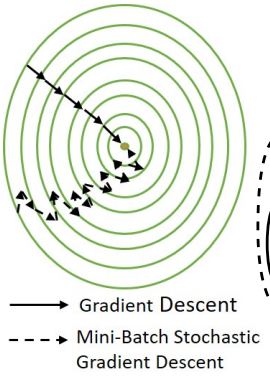

Figure 1: Contour Plot outlining Gradient Descent and Mini-Batch Gradient Descent while converging to a local minimum point.

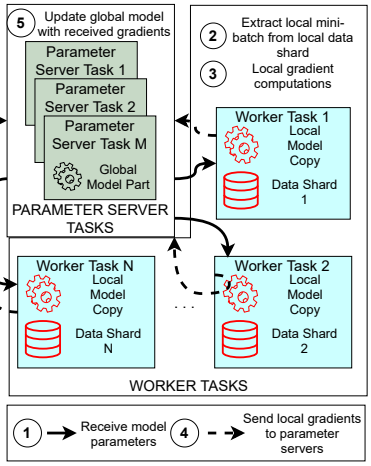

Figure 2: Training under the Parameter Server Architecture

solution of an optimization problem on a loss function L. Gradient Descent is considered to be among the most popular algorithm used in optimization problems [27]. However, considering both the size of deep neural networks and the vast amount of data usually available, Gradient Descent is not preferred, since each iteration will be too slow. Mini-Batch Stochastic Gradient Descent (mini-batch SGD) is used as an alternative instead, which uses only a subset of training examples in each training iteration.

While an iteration of Mini-Batch SGD is faster than the one of GD, it is important to note that it usually needs more iterations to converge. Figure 1 presents a contour plot, which outlines how the gradients move the weights towards the optimization point. Gradient Descent takes into account the whole data distribution in each training step and thus continuously moves towards the optimization point. However, mini-batch SGD computes the gradient of a training step with only B examples, directing the weights to various directions before converging. The aforementioned algorithms cannot guarantee convergence to the global minimization point, as optimization functions in neural network training are non-convex and they may stuck on local minima. Alternatives, as Adam, AdaGrad and others, have been proposed as less vulnerable to such phenomena [27].

2.2 Parameter Server Architecture

Following a data parallel scheme, parameter server architecture [15, 30] introduces two different entities in the learning process: the *workers* and the *parameter servers*. Parameter servers are used to store neural network parameters in a distributed fashion. Workers use local copies of the network and a local part of the data to compute gradients based on some variance of the mini-batch SGD algorithm. While gradients can be aggregated under various synchronization schemes, parameter server usually follows an asynchronous parallel approach for updating the global model in the servers. The steps of training a model under the aforementioned setup are fully depicted in Figure 2.

3 EXPLOITING DATA DISTRIBUTION IN LEARNING

In this section, we will present how we could exploit distribution related traits that could emerge from data preprocessing for the learning process, especially in the distributed case.

3.1 Exploit data distibution in single node training.

As we mentioned in Section 2.1, Mini-Batch SGD does not move the weights of the neural network directly to the minimization point due to the restricted view it has on the data on each iteration. However, we know that Gradient Descent is able to move directly towards the optimization point. A usual approach to overcome such problems when training deep learning models is to randomly shuffle the data before each mini-batch extraction in order to obtain a mini-batch with less correlated data [12].

However, it is reasonable to state the question what will happen if the mini-batch is chosen such that it is actually representative of the whole dataset. Will this either result to a more accurate model or to a faster training process in respect to the random sampling techniques used? Is it important to perform some preprocess to the training data in order to understand their structure and determine the sampling process during the mini-batch selection?

Bengio proposed Curriculum Learning [4] as an approach towards this direction and proved that the training was able to converge to better local minima when he decided to use traits of the data to help the network training process. For instance, in an image classification case, he decided to use only some easily distinguished data at first, and then include more complex images.

In this PhD research, we aim to focus on how to select the mini-batch on each iteration to be representative of the whole data set and boost the network training. As a proof of concept, we clustered each class from the CIFAR-10 [19] dataset to two sub-clusters and chose each mini-batch to include data from each resulting sub-cluster of all classes. In this example, we noticed a 5% improvement in the validation loss and a 2% improvement in the validation error. We further aim to examine whether we could benefit from real-time training metrics in order to select the training examples for each upcoming mini-batch.

3.2 Exploit data distibution in parameter server training.

Stale Gradients Effect. As we stated in the introductory section 1, the parameter server training is usually harmed from the stale gradients effect. Stale gradients occur when a worker computes a gradient update using old model parameters. In 2013, Dutta approached the problem of staleness with proposing an appropriate variable learning rate [10]. In 2017, Jiang also approaches the staleness problem with learning rate techniques in an heterogeneous environment [16]. Moreover, in 2018, Huang proposed FlexPS [15] which facilitated a staleness parameter controlling the aging of the parameter to avoid staleness effects.

Better data assignment to workers. In the research works stated above, algorithmic solutions in the parameter server or learning rate level are proposed to smooth the staleness effect. However,

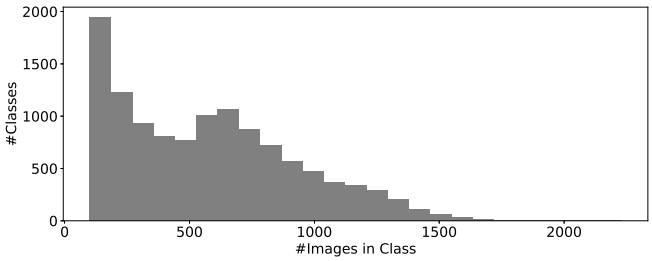

Figure 3: Class Population Histogram of Imagenet data obtained from Flickr

we believe that if the data part that can be accessed from each worker is not representative on the whole dataset, this may further harm the staleness effect, since a common approach is to randomly shard the data to workers. For instance, TensorFlow uses a modular sharding approach based on the training example index to assign it to a worker. To further support this claim, we will discuss an example based on an Imagenet subset with images from Flickr (approx. 60GB size). Figure 3 presents a histogram with the image population in each class. In this figure, we can observe that most of the classes consist of approximately 100 images, while some of them consist of more than 1000 (and even more than 1500 images). Thus, it is possible that a random data assignment approach could not provide a worker with data of some of the less populated classes or bias another towards a highly populated class (data skew on some workers). Having trained on a stale parameter set on some iteration, such worker could direct the weight not towards the direction of the true optimization point, but possibly to another one which will better optimize this part of data, due to lack of knowledge regarding the data space. In this research, we aim to study whether stratification in data sharding to workers and in mini-batch selection per worker (in class or in hidden level according to the distribution) could be facilitated in order to smooth the effects of staleness.

Stratification is widely used when the computing task cannot view entirely the data, as for example in an approximate query processing problem [17]. In the context of learning, it is also used to facilitate learning from heterogeneous databases [26]. Moreover, in a 2020 research [24], hidden stratification appeared to crucially affect the quality of classification models for medical images.

Extracting data distribution related information. Hidden stratification can be used to reveal how the data are organized in the distribution. A common approach to discover hidden patterns in the data distribution is the use of unsupervised learning techniques, as clustering. In the big data context, multiple clustering techniques have been proposed. For instance, in [28] they have designed a clustering framework for big data, which is able to discover multiple distribution types. Others propose approximate and distributed versions of common clustering algorithms, as DBSCAN [22]. Apart from clustering, it would also be efficient to utilize function that cluster together similar points, in the same manner as hash functions do. Towards this approach and in the spirit of Locality Sensitive Hashing, Gao proposed in [11] Data Sensitive Hashing, where he facilitates data distribution to hash together close data points in a high dimensional space.

4 RESEARCH PLAN

Having presented the concept and the ideas behind this PhD reasearch, he have designed a plan that we should follow to conduct this research. The aforementioned plan is outlined below.

- **Measure the effects of mini-batch design in single node training.** The first part of our work includes to propose and study efficient techniques that take into account distribution traits to systematically construct the mini-batches, as representatives of the whole data set, used for neural network training. In case of data sets with numerous classes, as ImageNet, where we can not create representative mini-batches with commonly used size, we plan to randomly omit different parts of the data from each mini-batch. Having experimented with the distribution traits, we further plan to take the real-time training and validation metrics into account when creating the next mini-batch. For instance, in case the model presents large loss metrics in some examples, we could attempt to provide the next mini-batch with more data following an equivalent distribution.
- **Study and evaluate techniques to extract distribution traits from big data sets.** A first approach to identify how data points are organized in the multidimensional space is with the help of clustering algorithms. However, since we want to focus on big data and distributing learning, we have to compare various techniques that could efficiently discover distribution related information, as the DSH one stated earlier. Having studied existing approaches in this problem, we will attempt to design and propose our own method that will efficiently compute any necessary information.
- **Apply distribution related information in data sharding.** In order to create representative data shards, our first goal is to consider class stratification, and identify whether it is able to facilitate the learning process. Moreover, having efficiently discovered any necessary distribution traits, the next part of our research aims to exploit them in shard creation process. The knowledge of stratification and distribution traits derived from the data is expected to further help parameter server training.
- **Design and propose a streaming system for serving mini-batches to workers in the parameter server setup.** Our research will conclude with the design and implementaion of a system that will collect data information and facilitate the distribution information extraction mechanisms to learn how to efficiently prepare mini-batches for the workers to train in the parameter server setup. This system will be created taking into consideration any observations from the steps described above.

4.1 Technologies

Having presented our research plan, we briefly describe state-of-the-art technologies that we plan to use in order to construct our various components.

- Apache Spark [31] is a widely used general purpose big data system. Among other libraries, Spark provides SparkML, which offers some clustering algorithms that we could exploit to identify initial data distribution traits. Moreover,

Spark can easily be used to compute any other interesting metrics that we might need to consider.
- Regarding neural network training, we aim to use Google TensorFlow [3], which also operates under the parameter server architecture.
- For streaming mini-batches we aim to examine the use of Apache Arrow [1], since it optimizes data in a columnar format for CPU and GPU analytical operations. Moreover, TensorFlow can directly read from Arrow streams.

We aim to design the final system as a layer over the training cluster, which will encapsulate all the above technologies. Thus, a user will be able to easily benefit from our system.

4.2 Benchmarking Setup

Having implemented each of our components, we aim to benchmark how they affect the asynchronous distributed training process in terms of speed and resulting training and validation metrics. As a baseline, we will use state-of-the-art neural networks trained under a parameter server setup with the optimal hyper parameters. For instance, we could train ResNet or Inception models using the ImageNet data set in a simple parameter server setup. These baseline models could be compared with the ones trained taking the distribution traits into account.

ACKNOWLEDGMENTS

This research has been co-financed by the European Union and Greek national funds through the Operational Program "Competitiveness, Entrepreneurship and Innovation", under the call RESEARCH – CREATE – INNOVATE (project code:T1EDK-04605).

REFERENCES

[1] [n.d.]. https://arrow.apache.org/
[2] [n.d.]. DAWNBench. https://dawn.cs.stanford.edu/benchmark/ImageNet/train.html
[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, and Michael Isard. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.
[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.
[5] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).
[6] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. *Training* 100, 101 (2017), 102.
[7] Jason Jinquan Dai, Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Cherry Li Zhang, Yan Wan, Zhichao Li, Jiao Wang, Shengsheng Huang, Zhongyuan Wu, Yang Wang, Yuhao Yang, Bowen She, Dongjie Shi, Qi Lu, Kai Huang, and Guoqiong Song. 2019. BigDL: A Distributed Deep Learning Framework for Big Data. In *Proceedings of the ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '19)*. Association for Computing Machinery, New York, NY, USA, 50–60. https://doi.org/10.1145/3357223.3362707
[8] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 1*. 1223–1231.
[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.
[10] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. 2018. Slow and stale gradients can win the race: Error-runtime tradeoffs in distributed SGD. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 803–812.
[11] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. 2014. DSH: data sensitive hashing for high-dimensional k-nnsearch. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1127–1138.
[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[15] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. 2018. FlexPS: Flexible Parallelism Control in Parameter Server Architecture. *Proceedings of the VLDB Endowment* 11, 5 (Jan. 2018), 566–579. https://doi.org/10.1145/3177732.3177734
[16] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 463–478.
[17] Srikanth Kandula, Kukjin Lee, Surajit Chaudhuri, and Marc Friedman. 2019. Experiences with approximating queries in Microsoft's production big-data clusters. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2131–2142.
[18] Evdokia Kassela, Nikodimos Provatas, Ioannis Konstantinou, Avrilia Floratou, and Nectarios Koziris. 2019. General-Purpose vs Specialized Data Analytics Systems: A Game of ML & SQL Thrones. In *2019 IEEE International Conference on Big Data (Big Data)*.
[19] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[21] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 115–124.
[22] Alessandro Lulli, Matteo Dell'Amico, Pietro Michiardi, and Laura Ricci. 2016. NG-DBSCAN: scalable density-based clustering for arbitrary data. *Proceedings of the VLDB Endowment* 10, 3 (2016), 157–168.
[23] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A data system for optimized deep learning model selection. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2159–2173.
[24] Luke Oakden-Rayner, Jared Dunnmon, Gustavo Carneiro, and Christopher Ré. 2020. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. In *Proceedings of the ACM conference on health, inference, and learning*. 151–159.
[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.
[26] Jose Picado, Arash Termehchy, and Sudhanshu Pathak. 2018. Learning efficiently over heterogeneous databases. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2066–2069.
[27] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
[28] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A Framework for Clustering Uncertain Data. *Proceedings of the VLDB Endowment* 8, 12 (Aug. 2015), 1976–1979. https://doi.org/10.14778/2824032.2824115
[29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2818–2826. https://doi.org/10.1109/CVPR.2016.308
[30] Eric P. Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A New Platform for Distributed Machine Learning on Big Data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 1335–1344. https://doi.org/10.1145/2783258.2783323
[31] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65. https://doi.org/10.1145/2934664
[32] Y. Zhang, B. Gong, and M. Shah. 2016. Fast Zero-Shot Image Tagging. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5985–5994. https://doi.org/10.1109/CVPR.2016.644