Towards Faster Distributed Deep Learning Using Data Hashing Techniques

Nikodimos Provatas
Computing Systems Laboratory
National Techincal University of Athens
Athens, Greece
nprov@cslab.ece.ntua.gr

Ioannis Konstantinou
Computing Systems Laboratory
National Techincal University of Athens
Athens, Greece
ikons@cslab.ece.ntua.gr

Nectarios Koziris
Computing Systems Laboratory
National Techincal University of Athens
Athens, Greece
nkoziris@cslab.ece.ntua.gr

***Abstract*—Nowadays, deep learning is a crucial part of a variety of big data applications. Both the vast amount of data and the high complexity of the state-of-the-art neural networks have led to perform the network training in a distributed manner accross clusters. Since synchronization overheads are usually fatal for the training's performance, asynchronous training is usually preferred in such cases. However, this training mode is sensitive to conflicting updates. Such updates most commonly occur when the workers train on a totally different part of the data. To reduce this phenomenon, in this paper, we propose the use of hashing schemes when distributing training data across workers.**

***Index Terms*—deep learning, data hashing, distributed training**

I. Introduction

Deep learning has become an important feature in a variety of big data applications, as in image classification [1] and speech recognition [2] domains. Specifically, deep architectures are growing larger and larger in order to effectively support the vast amount of available data. For example, in 2015, Microsoft proposed the ResNet architecture which provided the lowest error rate in the ImageNet dataset when consisting of 152 different layers [3].

Both the amount of data used and the size of the state-of-the-art neural networks imply to perform the training in a distributed way across clusters. For instance, Google has developed TensorFlow to support distributed deep learning [4]. Multiple architectures have been proposed to distribute the training of neural networks, with the parameter server [5]–[7] being the most common one. In large networks, the layer parameters are split across multiple parameter servers, a technique named *model parallelism* [8]. Thus, communication and synchronization overheads are introduced [9], which crucially hurt the scalability of the training.

However, apart from the layers of a neural network, data can also be distributed between the training workers (*data parallelism*) [8]. This process is depicted in Figure 1 for the parameter server case. In this setup, each worker has a local copy of the model and updates using different gradients, based on their local view of the data. Obviously, synchronization between the various gradient updates is needed to retrieve the same model with a single node training, introducing overheads,

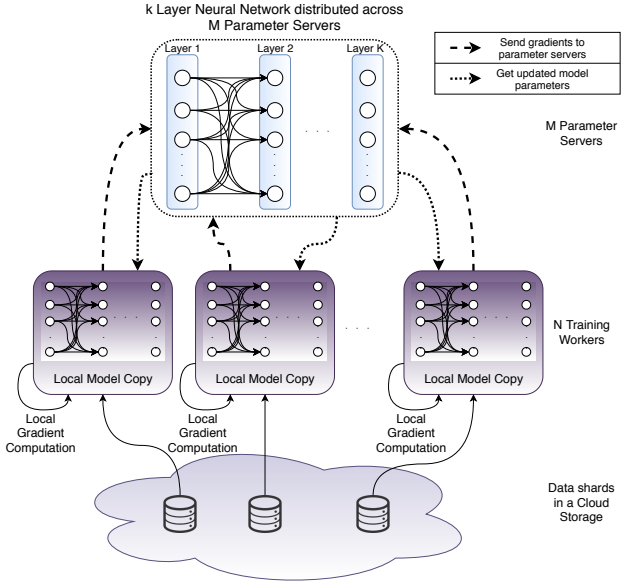

Fig. 1. Distributed training using Data Parallelism in a Parameter Server Architecture.

as stated above. In order to avoid such overheads, asynchronous training is a common workaround. Asynchronous training suggests no synchronization between the available training workers. However, it is sensitive to conflicting and stale parameter updates and is able to converge if the number of concurrent updates is small [10]. Conflicting updates may occur when the workers train utilizing totally different part of the data, since data are distributed across worker in a random or round robin way.

In this paper, the main idea is to distribute data across workers in a systematic way. Since such approach will reduce the amount of conflicting updates while training neural networks in an asynchronous manner, less training epochs will be needed to achieve convergence.

The rest of the paper is organized as follows. First of all, the essential background needed for the reader to understand our approach is presented in Section II. Then, the aforementioned data distribution is further described in Section III. Finally, in Section IV, we outline related work to our problem and the techniques we propose.

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

II. Theoretical Background

One technique to avoid conflicting updates that may occur in asynchronous training is that all workers have a similar view on the dataset. Since each worker utilizes a different local sub-batch for a training iteration, we could provide them with similar sub-batches (sub-batches consisting of similar data points) or split the data in similar shards.

Considering that each data point can be presented as a d-dimensional vector in $\mathbb{R}^d$ and $\|s,t\|$ is a distance between the vectors $s,t \in \mathbb{R}^d$, the ϵ-approximate nearest neighbour search (ϵ-ANN) problem is mathematically formulated in Definition 1 [11].

Definition 1 (ϵ-approximate Nearest Neighbour Search): Given a set $S \subset \mathbb{R}^d$, preprocess the set S to efficiently locate a point $q_0 \in S$, such that for any query point s:

$$\|q_0, s\| \leq (1+\epsilon) \cdot \min_{t \in S} \|s,t\|$$

The ϵ-ANN problem can be generalized to the ϵ-approximate k-Nearest Neighbours (ϵ-kNN) problem. The generalized problem is formulated in Definition 2.

Definition 2 (ϵ-approximate k Nearest Neighbour Search): Given a set $S \subset \mathbb{R}^d$ and any query point $s \in S$, preprocess the set S to efficiently provide a sequence of data points q_1, q_2, ..., $q_k \in S$, s.t. the point q_i is not further from the query point s than $1+\epsilon$ times the distance of s from its i-th nearest neighbour.

One of the state-of-art methods to solve the aforementioned problems in sub-linear time is the Locality Sensitive Hashing (LSH) [12], [13]. LSH uses a family of functions, named LSH family, to hash a set of data points to ensure that similar points will collide with greater probability than dissimilar points. The formulation of an LSH family is provided with the conditions given in Definition 3.

Definition 3 (Locality Sensitive Hashing (LSH) family): A family $\mathcal{H} = \{h : S \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive for a similarity measure D if for any data points $q_1, q_2 \in S$, the following conditions are satisfied:

- if $\|q_1, q_2\| \leq r_1$, then $\mathbb{P}[h(q_1) = h(q_2)] \geq p_1$
- if $\|q_1, q_2\| \geq r_2$, then $\mathbb{P}[h(q_1) = h(q_2)] \leq p_2$

A common problem when using the LSH families is that the probabilities p_1, p_2 used in Definition 3 might not create strict enough conditions to achieve a proper hashing. A common technique to overcome this problem is to concatenate hash functions from a given hash family, as defined in Definition 4. Thus, dissimilar points are more unlikely to be in the same bucket.

Definition 4 (AND-Concatenation of LSH Functions): Given an (r_1, r_2, p_1, p_2)-sensitive LSH family $\mathcal{H} : S \to U$ and a positive integer m, a set of concatenated hash functions $\mathcal{G} : S \to U^m$ can be defined. Specifically, each hash function $g \in \mathcal{G}$ on a data point $p \in S$ is formulated as

$$g(p) = (h_1(p), h_2(p), ..., h_m(p))$$

where $h_1, h_2, ..., h_m$ are randomly chosen from the hash family $\mathcal{H}$ with replacement. Thus, $\mathcal{G}$, satisfies for any data points $q_1, q_2 \in S$ the properties

- if $\|q_1, q_2\| \leq r_1$, then $\mathbb{P}[g(q_1) = g(q_2)] \geq p_1^m$
- if $\|q_1, q_2\| \geq r_2$, then $\mathbb{P}[g(q_1) = g(q_2)] \leq p_2^m$

and is an (r_1, r_2, p_1^m, p_2^m)-sensitive LSH family.

III. Approach

In our approach, we will examine hashing techniques for more effective data distribution, which aims to reduce the conflicting updates noticed in the asynchronous training. First of all, we aim to design hash families, with the properties mentioned in Definition 3 (see Section II). However, our hash families will take into account the data distribution to identify the similarity between two given data points. For this purpose a representative sample from the data will be used. Using the concatenation of functions randomly chosen from this custom LSH-like family, it will be easier to locate the nearest neighbours of a given query data point in terms of the distribution.

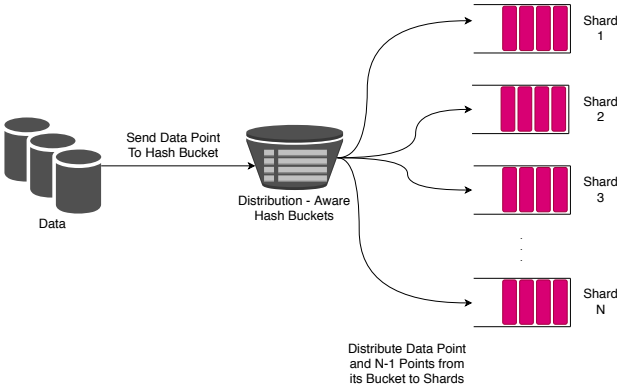

Fig. 2. Data Shard Creation using Hash Buckets.

Given the ability to solve efficiently the ϵ-kNN problem by exploiting LSH hash families, we vision a better data distribution, following the process depicted in Figure 2. Such distribution Figure 2 shows how the data in the cloud will be split into the shards used from the workers to train a neural network. Let a neural network training be performed by a set of N workers. Thus, the data need to be split into N shards. Suppose that a specific data point *A* is chosen to be part of a shard. Performing an ϵ-kNN query on the hash buckets created from the LSH-like functions described in the previous paragraph will provide the similar data points to *A*. As mentioned in Section II, note that these neighbours will be found in sub-linear time to the data sample used to create the buckets. This set of N data points will be distributed across the shards. The process is continued until all the data are split.

By definition the hash family will be designed to take into account the data distribution. Therefore, the workers will have a similar view on the data when training with a specific shard.

Especially if the shards are placed in queues, their first-in first-out operations will ensure that a local sub-batch used from one worker will be equivalent to the one local sub-batch used from another. Thus, conflicting gradient updates will be restrained, rendering the asynchronous training more effective.

In case of vast amount of data, it might be prohibited to split the data beforehand. The method described above could be adapted to be used in the batch creation part. In more detail, when one worker includes one data point in their local sub-batch, they will inform the others for their selection. Thus, the rest workers will be capable of using the hash buckets to select similar data for their local sub-batches.

IV. Related Work

There are numerous other works that integrate hashing techniques with the deep learning domain for efficient training. For example, in [14], the authors reduce the number of computations that occur during the training or testing of a neural network with the use of hashing based techniques. Their hashing approach performs an efficient inner product search on fewer sparse nodes to locate the nodes with the highest activation. Specifically, with using only the 5% of the total number of multiplications, they keep the accuracy within 1% of the one of the original model. In [15], they propose a new neural network archtecture based on hashing. In further detail, they use low-cost hash functions to group network weights, where weights in the same group have the same value. Thus, they manage to reduce the number of parameters of a neural network and therefore the memory footprint of the network. In this paper, we have also proposed the use of hashing to enhance deep learning. However, unlike the aforementioned works, we propose the use of hashing in the data distribution level in order to reduce the training time, by restraining the number of conflicting updates.

Moreover, hashing techniques are used in schemes like LSH in order to answer ϵ-kNN queries in a more efficient manner. A variety of works have generated hash families that take into account the distribution of the data that they are about to hash [16], [17]. In the first work, the propose Data Sensitive Hashing (DSH), which aims to improve hashing functions and families for more efficient response to ϵ-kNN queries using machine learning techniques. Their experiments show that indexing structures are created in comparable time with those of the simple LSH case, while they provide more balanced results. The latter work aims again to cost-effective approximate neighbour queries and leverages techniques from the Principal Component Analysis field to create LSH-like buckets. Their proposed approach is named Distribution-Aware LSH (DLSH). Experiments presented in this paper state that ϵ-kNN queries have better latency and accuracy when answered with DLSH than with other traditional method and are more space-efficient. Inspired from those approaches that take into account the data distribution when applying LSH methodologies for ϵ-kNN queries, we believe that similar approaches should be applied on the distributed deep learning field for more efficient neural network creation.

V. Acknowledgement

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] 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] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[5] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 583–598. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[6] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 703–710, 2010.

[7] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[8] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[9] E. Kassela, N. Provatas, I. Konstantinou, A. Floratou, and N. Koziris, "General-Purpose vs Specialized Data Analytics Systems: A Game of ML & SQL Thrones," in *2019 IEEE International Conference on Big Data (Big Data)*, Dec 2019.

[10] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[11] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[12] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[13] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 10 2011, ch. 3, pp. 80–84.

[14] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 445–454.

[15] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[16] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: data sensitive hashing for high-dimensional k-nnsearch," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1127–1138.

[17] Y. Sun, Y. Hua, X. Liu, S. Cao, and P. Zuo, "DLSH: a distribution-aware LSH scheme for approximate nearest neighbor query in cloud computing," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 242–255.