LITERATURE REVIEW: Evaluating Distributed methods for training Generative Adversarial Networks

Omid Davoudi
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
omiddavodui@cmail.carleton.ca

October 1, 2019

1 Introduction

Deep Learning has opened many possibilities in the world of machine learning. This comes at the cost of requiring extremely large datasets and high computational power. As datasets grow in size, it is getting harder and harder to fit them in the memory of a single machine. Sometimes, different partitions of data are located in different continents. The slowdown in hardware development and network speed increases can also be a major concern for the future of machine learning. Lastly, privacy concerns and regulations could prevent movement of data between different countries or even devices, further complicating the learning process.

One of the most realistic ways to combat those problems is to distribute the training of Deep Learning models among a number of distinct computers. The machines would not necessarily need to be located in one locality and could theoretically be distributed among different continents. The massively parallel nature of artificial neural networks, which form the backbone of deep learning, helps in this regard.

Generative Adversarial Networks (GANs)[1] are a deep learning model that learns the distribution of the input data and outputs data points from that distribution. One of the domains where GANs are popular is the domain of image generation. A properly trained GAN can learn from a dataset of images and then, output new images that, while not found in the dataset, follow the same general rule. For example, a GAN properly trained on dataset containing images of cats will be able to output new cat images.

GANs are generally hard to train.[2] Their structure and competing nature means that convergence is not guaranteed. Even when they do converge, there could be many problems with the output, such as different distributions of data or in severe cases, mode collapse. The additional challenge of having to train these models in a distributed manner could further complicate the situation and possibly prevent convergence.

In this section, a number of approaches for distributing the training of neural networks are introduced. Furthermore, an approach specifically designed for distributed training of GANs is reviewed.

2 Literature Review

Related literature to this field can be classified in three major subgroups:

- Algorithms that try to distribute any machine learning algorithm, including neural networks, over a set of machines.
- Algorithms that try to distribute neural network training itself
- Algorithms and architectures that try to distribute training of a specific class of neural networks.

This section presents the publications based on the above groupings.

2.1 Distribution of general machine learning algorithms

Previous work on methods to distribute machine learning algorithms in general mostly tries to find efficient ways for each machine to access the elements of the dataset over a network. One of the main characteristics of these approaches is that there are very few assumptions about the underlying machine learning algorithm. As a result, they can usually be exploited by broad groups of machine learning algorithms.

As mentioned above, these approaches usually try to help the machine learning algorithm access data over a network. An approach by Li et. al[3] defines a parameter server where each node can access the parameters of the model as well as the training data without worrying about where the data is actually located.

The approach groups the machines into two categories: Server nodes and workers. Server nodes each contain a portion of the model data and they communicate with each other to keep the model consistent across the whole server group. Data is replicated across servers to increase reliability.

As server nodes are not required to contain all of the data at the same time, the system can be scalable even when the parameters exceed the storage capability of any single server node. There is also a server manager node which is designed to organize the servers themselves by checking for node removals, failures or additions. Worker nodes are where the actual computation takes place. worker nodes are separated into worker groups and each node can access or update the parameters of the model by issuing push and pull commands to the server group. A scheduler for each worker group assigns tasks to each worker and monitors the progress, rescheduling the tasks in case of node failure or addition. This system is designed to be model agnostic.

While this approach performs well in clusters with high network bandwidth and low latency, it fails in situations where the network bandwidth between nodes is low or the latency is high. This is because most machine learning algorithms are extremely communications intensive. One example of such setup is the case where some of the nodes are located in different geographical positions. This can be as extreme as having nodes in a different continent. In these cases, the bandwidth between different geographic locations is low and the latency will be higher as the physical distance grows.

One way to address this is by centralizing all of the data into a single data center with high bandwidth between each node. This approach has complications such as regulatory prohibitions, cost of data movement and the need for exceedingly large data centers that can store that large amount of data.

To address these issues, Cano et. al[4] introduced a method to perform some machine learning algorithms in a geo-distributed manner. This approach tries to minimize communications between datacenters by only sending statistical information and estimates between them. The machine learning algorithms that can be geo-distributed this way are the ones that can fit the Statistical Query Model[5].

While the previous approach helped mitigate the problem with cross-datacenter communications, it is limited to only a subset of the machine learning algorithms. To address this, Hsieh et al.[6] introduced a method called Gaia. This method tries to limit the communication between different datacenters without changing the underlying machine learning algorithm. It works by reducing the communication to synchronize the models in different datacenters. Instead, it does so only when the models have sufficiently diverged from each other.

This approach works on the basis that most of the machine learning algorithms iterations do not change the parameter values significantly. Communicating each of these insignificant updates to the central model will use up valuable cross-datacenter network bandwidth. Waiting until significant divergence can be detected in the model before submitting the changes will drastically decrease the communications between different datacenters. Within datacenters themselves, the shortage of network bandwidth is not as profound. This means that within each datacenter, updates could always be synchronized.

2.2 Distribution of neural network training

While general approaches for distributing machine learning tasks have shown to work well in many situations, their all-encompassing nature and the need to apply for many different machine leaning algorithms prevents them from reaching the best possible performance for the task at hand. As neural networks become more and more popular, the need for distribution mechanisms tailored for train these models increases.

One approach is Federated Learning Proposed by McMahan et al.[7]. This approach tries to distribute the training of a neural network among a large number of mobile devices each with a local dataset. To do so, a fraction of the devices are selected and the gradient of the loss over all data contained within each is calculated. These gradients are then sent into a central server. The central server then naively averages the gradients and repeats the process for another subset of the nodes.

Naively averaging these gradients has been shown to have the potential to result it bad models.[8] To combat this, federated learning synchronizes the initial weights of the different models. Empirical results from the paper show that this approach results in relatively good average models, even those that have only been trained on different portions of the dataset.

2.3 Distribution of GAN training

Generative adversarial networks are different in that the consist of two separate networks that are trained in two phases. Previous neural network training distribution schemes are generally working to distribute the training of networks used in classification. The training of GANs on the other hand, is not as straightforward as that of normal classifier neural nets.

One of the more recent suggested methods for training GANs in distributed settings is the MD-GAN architecture.[9] The way this architecture deals with the problem of having two networks is to only synchronize the generator of the GAN. As the generator is usually

the real reason for training a generative adversarial network, having different local discriminators would not be a disadvantage of this approach. On the other hand, it could simplify the training procedure and help with convergence in a model known for being hard to train.

Simply using synchronized generators with local discriminators is not going to guarantee success. The approach assumes that the class distribution among different nodes is similar. It also interchanges the discriminators in fixed intervals. This is needed because otherwise the discriminators will overfit on local data and will result in the generator to diverge.

References

[1] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[2] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which training methods for GANs do actually converge?" In: *arXiv preprint arXiv:1801.04406* (2018).

[3] Mu Li et al. "Scaling Distributed Machine Learning with the Parameter Server". In: *Proceedings of OSDI* (2014), pp. 583–598.

[4] Ignacio Cano et al. "Towards geo-distributed machine learning". In: *arXiv preprint arXiv:1603.09035* (2016).

[5] Michael Kearns. "Efficient noise-tolerant learning from statistical queries". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 983–1006.

[6] Kevin Hsieh et al. "Gaia: Geo-Distributed Machine Learning Approaching {LAN} Speeds". In: *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 2017, pp. 629–647.

[7] H. Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017* 54 (2017). arXiv: arXiv:1602.05629v3.

[8] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems". In: *arXiv preprint arXiv:1412.6544* (2014).

[9] Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. "MD-GAN: Multi-Discriminator Generative Adversarial Networks for Distributed Datasets". In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* ii (2019), pp. 866–877. DOI: 10.1109/ipdps.2019.00095.