Evaluating the MD-GAN architecture for distributed training of Generative Adversarial Networks

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

December 1, 2019

Abstract

Distributed training of Generative Adversarial Networks is a hard task considering the convergence patterns of these models. A method suggested to help is the MD-GAN architecture. The original paper on MD-GAN assumed that the local datasets have an equal class distribution. This assumption is almost always wrong when working with real world datasets. This project evaluated the performance of MD-GAN on imbalanced datasets and found out that this model is indeed vulnerable to different class distribution in local datasets, but the problems start at a skew of around 60%. This means that despite their vulnerability, MD-GAN based models can still function with the majority of real world datasets.

1 Introduction

Machine Learning has been on the rise in the past decade, mainly due to the advent of Deep Neural Networks and Big Data. While the larger amount of data has enabled us to achieve results that were impossible before, utilizing it via the base machine learning algorithms is hard or even sometimes impossible. Real world data could be so large that it could not fit into a single machine. Another problem is the recent rise in the regulations forbidding the movement of data out of national boundaries. Lastly, the processing power required to handle this amount of data might be too large to exist within a single computer. One of the most promising solutions to these problems is Distributed Machine Learning.

Distributed Machine Learning is a set of methods and algorithms used to offload part of the learning task on other computers, usually connected by a network. This way, the resources of other machines can be used to increase the speed of training and sometimes, enable the training on very large datasets. It can also help with abiding by the local laws as nodes could theoretically be scattered around the globe.

Past research has resulted in many different solutions for distributed machine learning. Some of these solutions are general enough to work for all types of machine learning algorithms. Others only apply to specific models and methods. The latter generally have better performance due to using inherit properties of the underlying machine learning model. This project focuses on a specific type of machine learning model called Generative Adversarial Network.

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.

A solution proposed by Hardy et al.[3] is to use local discriminators coupled with a shared generator to learn from the data. This architecture, called MD-GAN, uses much less network bandwidth than naive methods and is also complete in the sense that it does not delay updates until noticeable change in the parameters.

However, MD-GAN is not without its flaws. The original paper assumes identical data distributions in the local datasets of each node. This is not true for most of real world datasets, especially if the data is gathered from different geographical locations. As each node has its own local discriminator, it is possible that each discriminator will converge on the local data distribution. This could prove catastrophic for the generator which relies on the backpropagated gradients from the discriminators to update its weights. Discriminators that do not agree with each other could pull the weights in different directions, destroying the any chance of learning for the generator.

The mentioned vulnerability of MD-GAN could have important consequences on its use in real world situations. As a result, this project aims to evaluate the extend of this vulnerability. In different experiments, different datasets with varying levels of imbalance are given to MD-GAN to train on. The results show that While the vulnerability certainly exists, it does not happen unless very large imbalances of about more than 60% exist. One of the more immediate concerns is increased chance for mode collapse and generally decreased diversity of results. In the end, this shows that MD-GAN can be used for real world data if the distribution differences among local datasets is not too large.

The paper is sectioned as follows: In Section 2, we will review the relevant literature. Section 3 will present the exact definition of the problem. In Section 4, the experiments are presented and Section 5 concludes the paper.

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[4] 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[5] 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[6].

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.[7] 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.[8]. 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.[9] 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.

Another approach is the one proposed by Dean et al.[10] It tries to decrease the number of network calls by applying a modified version of gradient descent called Downpour SGD. This approach, built upon the parameter server framework, does not update the central parameter server every time a local update happens. Worker nodes in turn, do not ask for the updated parameters from the central server either. These changes, coupled with splitting different parameters in different nodes reduces network calls significantly while having a modest impact on model performance. Still, delayed updates could potentially hurt the convergence of the model.

Wen et al. proposed TernGrad[11], a method which tries to decrease network traffic usage by using ternary gradients. Instead of communicating the updates via float vectors, this approach uses ternary vectors of either -1, 0 or 1 to convey direction of the update. Small floating point vectors are also shared to determine an estimate of the magnitude of the change. This approach has been proven to converge on supervised learning tasks and the model performance suffers very little. On the other hand, nothing is known about its behaviour in unsupervised settings like GANs and it loses information by using ternary gradients.

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.

An approach that is used in High Energy Physics[12] is to use Horovod[13]. In this case, each node gets part of the data from a central data storage and calculates the gradients locally. The gradients are then averaged over the workers and every node is then updated. This process is designed for increasing training speed and does not scale well for when some nodes are located in different datacenters. It also requires a central data storage which might not be possible depending on the amount of data.

One of the more recent suggested methods for training GANs in distributed settings is the MD-GAN architecture.[3] 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.

3 Problem Statement

The goal of this paper can be summarized as this: Given an MD-GAN network based on an arbitrary GAN architecture, determine whether imbalanced local datasets have any impact on the convergence and the results of the final model. If there is an impact, give a threshold for the amount of imbalance required before noticeable changes occur.

The reason this is important is that most real world datasets are imbalanced. If local datasets are gathered from different geographical locations, it is almost certain that the data distributions from each of these locations is different. Take the example of medical information of patients. If the data is gathered from around the globe, it is likely that each location has different data distributions due to the fact that each country has a different medical profile. Even regions within a single country might have differences in terms of common diseases and ailments.

As a result, it might be important to see the performance of MD-GAN on nodes with different local data distributions. If any vulnerability is found, it is worthwhile to know how imbalanced the datasets should be before adverse effects become apparent.

4 Experiments

To test the hypothesis, an MD-GAN architecture was implemented over a cluster of 4 machines. One of the machines was the server and the rest were workers. Each machine had 4 CPUs and the training was done using the pytorch framework. The base GAN network was a simple Deep Convolutional GAN with 4 layers in each of discriminator and the generator.

The dataset used was the MNIST handwritten digits dataset. In each experiment, each of the classes were split into 3 separate chunks and given to each of the workers to train on. As a result, no data point was shared between two workers. The split was created so that one of the workers got more data of the same class compared to the other two based on the skew value. One of the classes was always balanced for all of the workers to see the effects of mixed imbalances in the data.

The experiment was done for skew values of 0%, 20%, 40%, 60%, 80% and 100%. A value of 0 means perfect balance and a value of 100 means that a node contains all of the instances of a class while the other two contain none (total imbalance). Samples of the best results after 20 epochs and a sample of the dataset are shown in Figure 1-7.

Figure 1 is a typical example of an experimental evaluation result. Such graphs are ususally created with GnuPlot.

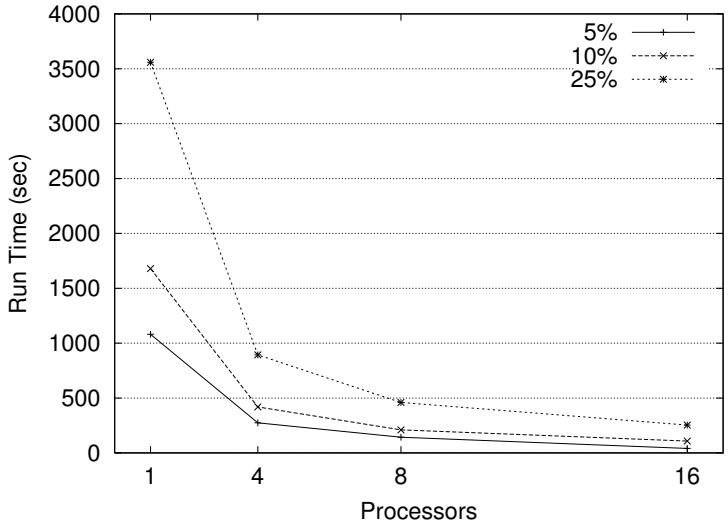

Figure 1: Measured Running Times Of Some Unknown Algorithm Implementation

5 Conclusions

You generally cover three things in the Conclusions section.

1. Conclusions
2. Summary of Contributions
3. Future Research

Conclusions are not a rambling summary of the thesis: they are short, concise statements of the inferences that you have made because of your work. All conclusions should be directly related to the research question.

The Summary of Contributions will be much sought and carefully read by the readers. Here you list the contributions of new knowledge that your paper makes. Of course, the paper itself must substantiate any claims made here. There is often some overlap with the Conclusions, but that's okay.

The Future Research should indicate interesting new problems arising from your work. No paper ever solves everything. In fact, the best research papers lead to new research questions for other researchers to work on.

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] 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.

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

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

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

[7] 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.

[8] 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.

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

[10] Jeffrey Dean et al. "Large scale distributed deep networks". In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

[11] Wei Wen et al. "Terngrad: Ternary gradients to reduce communication in distributed deep learning". In: *Advances in neural information processing systems*. 2017, pp. 1509–1519.

[12] Sofia Vallecorsa et al. "Distributed Training of Generative Adversarial Networks for Fast Detector Simulation". In: *International Conference on High Performance Computing*. Springer. 2018, pp. 487–503.

[13] Alexander Sergeev and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow". In: *CoRR* abs/1802.05799 (2018). arXiv: 1802.05799. URL: http://arxiv.org/abs/1802.05799.