Master Thesis

On Scalable Deep Learning and Parallelizing Gradient Descent

Joeri R. Hermans

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science of Artificial Intelligence

at

Maastricht University
Faculty of Humanities and Sciences
Department of Data Science & Knowledge Engineering
Maastricht, The Netherlands

Preface

This thesis is submitted as a final requirement for the Master of Science degree at the Department of Data Science & Knowledge Engineering of Maastricht University, The Netherlands. The subject of study originally started as a pilot project with Jean-Roch Vlimant, Maurizio Pierini, and Federico Presutti of the EP-UCM group (CMS experiment) at CERN. In order to handle the increased data rates of LHC Run 3 and High Luminosity LHC, the CMS experiment is considering to construct a new architecture for the High Level Trigger based on Deep Neural Networks. However, they would like to significantly decrease the training time of the models as well. This would allow them to tune the neural networks more frequently. As a result, we started to experiment with various state of the art distributed optimization algorithms. Which resulted in the achievements and insights presented in this thesis.

I would like to express my gratitude to several people. First and foremost, I would like to thank my promotors, Gerasimos Spanakis, and Rico Möckel for their expertise and suggestions during my research, which drastically improved the quality of this thesis. Furthermore, I would also like to thank my friends, colleagues and scientists at CERN for their support, feedback, and exchange of ideas during my stay there. It was a very motivating and inspiring time in my life. Especially the support and experience of my CERN supervisors, Zbigniew Baranowski, and Luca Canali, was proven to be invaluable on multiple occasions. I would also like to thank them for giving me the personal freedom to conduct my own research. Finally, I would like to thank my parents and grandparents who always supported me, and who gave me the chance to explore the world in this unique way.

Joeri R. Hermans
Geneva, Switzerland 2016 - 2017

Abstract

Abstract here.

Summary

Summary here.

Contents

Abbreviations and Notation

λ	Communication period, or frequency of commits to the parameter server.
$\nabla f(\theta\ ;\ x)$	Gradient of f with respect to parametrization θ, and input x.
τ	Staleness
$\tilde{\theta}_t$	Center variable, or central parametrization maintained by the parameter server.
$\triangleq$	Is defined as
n	Number of parallel workers.
ADAG	Asynchronous Distributed Adaptive Gradients
ASGD	Asynchronous Stochastic Gradient Descent
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
EASGD	Elastic Averaging Stochastic Gradient Descent
HL-LHC	High Luminosity Large Hadron Collider
LHC	Large Hadron Collider
MNIST	Mixed National Institute of Standards and Technology database
PS	Parameter Server
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

In this chapter we introduce Distributed Deep Learning and the problems surrounding it. A more detailed description of the subject of study is given in Chapter 3. Furthermore, we make the reader more comfortable with the notation and abbreviations used throughout this thesis. Finally, we formally define the problem statement in Section 1.2, and give an outline of this thesis in Section 1.3.

1.1 Distributed Deep Learning, an introduction

Unsupervised feature learning and deep learning has shown that being able to train large models can drastically improve model performance. However, consider the problem of training a deep network with millions, or even billions of parameters. How do we achieve this without waiting for days, or even multiple weeks? Dean et al. propose a different training paradigm which allows us to train and serve a model on multiple physical machines [1]. The authors propose two novel methodologies to accomplish this. Namely, *model parallelism*, introduced in Section 1.1.1, and *data parallelism*, introduced in Section 1.1.2.

In this thesis we study *data parallelism*, since this methodology mainly focuses on the development of distributed optimization algorithms. Whereas, *model parallelism* is mainly an engineering effort because it still follows the traditional optimization scheme, i.e., sequential gradient updates in the case the applied numerical optimizer utilizes a gradient based approach.

1.1.1 Model Parallelism

In *model parallelism*, a single model is distributed over multiple machines [1]. The performance benefits of distributing a deep network across multiple machines mainly depends on the structure of the model. Models with a large number of parameters typically benefit from access to more CPU cores and memory, up to the point where communication costs, i.e., propagation of weight updates and synchronization mechanisms, dominate [1].

Let us start with a simple example in order to illustrate this concept more clearly. Imagine having a perceptron, as depicted in Figure 1.1. In order to parallelize this efficiently, we can view a neural network as a dependency graph, where the goal is to minimize the number of synchronization mechanisms, assuming we have unlimited resources. Furthermore, a synchronization mechanism is only required when a node has more than 1 *variable* dependency. A variable dependency is a dependency which can change in time. For example, a bias would be a *static* dependency, because the value of a bias remains constant over time. In the case for the perceptron shown in Figure 1.1, the parallelization is quite straightforward. The only synchronization mechanism which should be implemented resides in output neuron since $y \triangleq \sigma(\sum_i w_i x_i)$ where σ is the activation function of the output neuron.

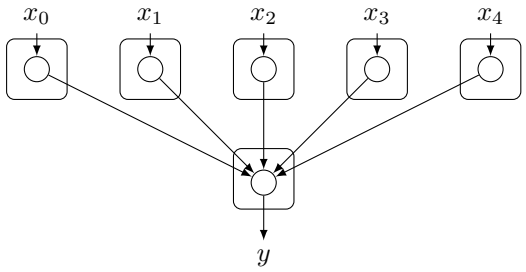

Figure 1.1: A perceptron partitioned using the *model parallelism* paradigm. In this approach every input node is responsible for accepting the input x_i from some source, and multiplying the input with the associated weight w_i. After the multiplication, the result is sent to the node which is responsible for computing y. Of course, this node requires a synchronization mechanism to ensure that the result is consistent. The synchronization mechanism does this by waiting for the results y depends on.

1.1.2 Data Parallelism

In this thesis, we focus our efforts on data parallelism. Data parallelism is an inherently different methodology of optimizing parameters. The general idea is to reduce the training time by having n different workers optimizing a model by processing n different shards (partitions) of the dataset in parallel [1]. In this setting we distribute n model replicas over n processing nodes, i.e., every node (or process) holds one model replica. Then, the workers train their local replica using the assigned data shard. However, it is possible to coordinate the workers in such a way that, together, they will optimize a single objective. There are several approaches to achieve this, and these will be discussed in greater detail in Chapter 3.

Nevertheless, a popular approach to optimize this objective, is to employ a centralized *parameter server* [1, 5, 4]. A parameter server is responsible for the aggregation of model updates, and parameter requests coming from different workers. The distributed learning process starts by partitioning a dataset into n *shards*. Every individual shard will be assigned to a particular worker. Next, a worker will sample mini-batches from its shard in order to train the local model replica. After every mini-batch (or multiple mini-batches), the workers will communicate a variable with the parameter server. This variable is in most implementations the gradient $\nabla f_i(x)$. Finally, the parameter server will integrate this variable by applying a specific UPDATE procedure which knows how to handle this variable. This process repeats itself untill all workers have sampled all mini-batches from their shard. The above high level description is summarized in Figure 1.2.

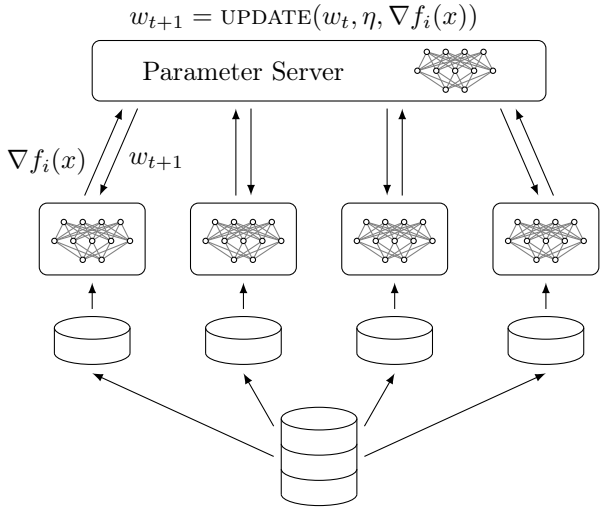

Figure 1.2: Schematic representation of a data parallel approach. In this methodology we spawn n workers (not necessarily on different machines), and assign a data shard (partition) of the dataset to every worker. Using this data shard, a worker i will iterate through all mini-batches to produce a gradient, $\nabla f_i(x)$, for every mini-batch x. Next, $\nabla f_i(x)$ is send to the parameter server, which will incorperate the gradient using an UPDATE mechanism.

1.2 Problem Statement

1.3 Thesis Outline

Chapter 2

Optimization Algorithms

Chapter 3

Distributed Deep Learning

3.1 Synchronous Data Parallelism

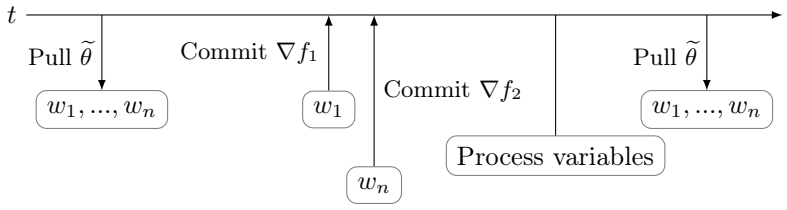

Figure 3.1: Caption here

3.1.1 Model Averaging

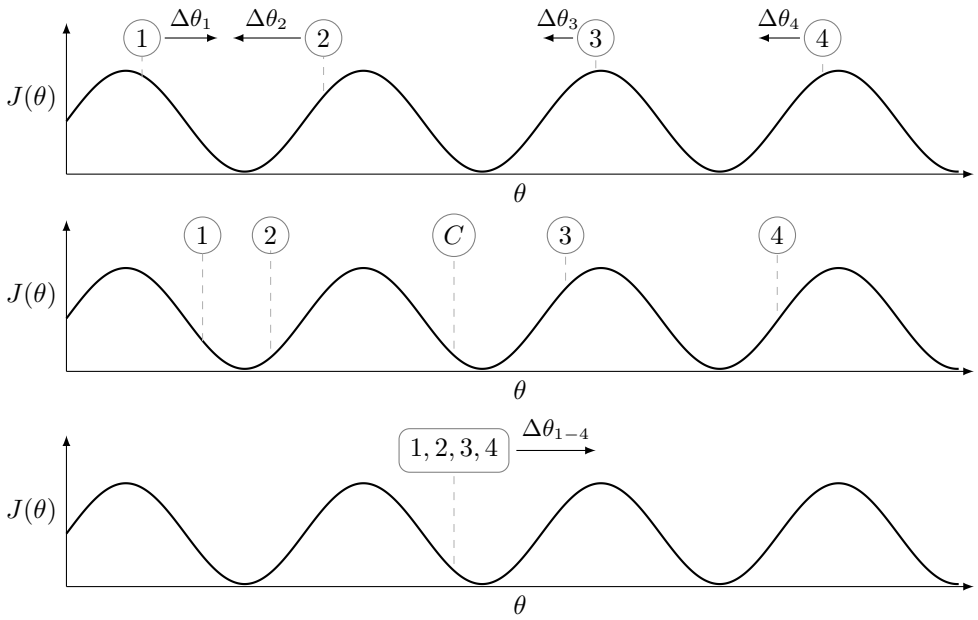

Figure 3.2: Caption here

3.1.2 Elastic Averaging SGD

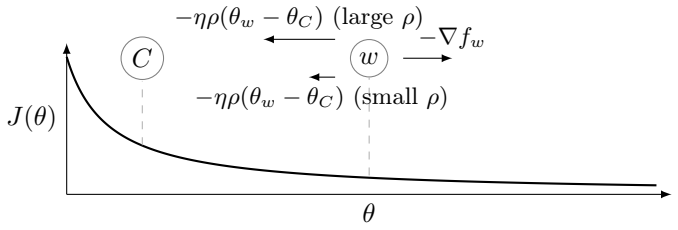

Figure 3.3: EASGD Caption here

3.2 Asynchronous Data Parallelism

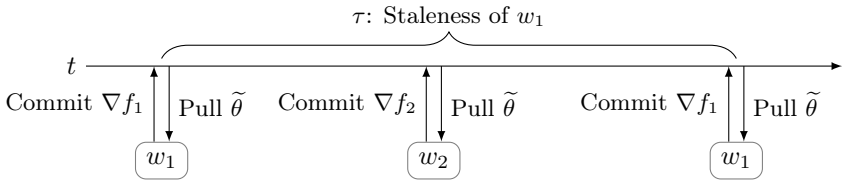

Figure 3.4: Caption here.

3.2.1 Hogwild!

3.2.2 DOWNPOUR

3.2.3 Asynchronous Elastic Averaging SGD

Chapter 4

Accumulated Gradient Normalization

$$\Delta\theta = -\frac{\sum_{i=0}^{\lambda} \eta(t) \frac{1}{m} \sum_{j=0}^{m} \nabla f(\theta_i; x_{ij}; y_{ij})}{\lambda} \tag{4.1}$$

Chapter 5

Asynchronous Distributed Adaptive Gradients

In this chapter we introduce a novel optimizer called ADAG. ADAG, or *Asynchronous Distributed Adaptive Gradients*, is an optimization process designed with data parallel methods in mind. We build upon previous work [1, 2, 3, 5] and incorperate new insights backed up by theory and experimental evidence. We start in Section 5.1 by formalizing the problem setting. In Section 5.2, we summarize previous work on distributed (data parallel) optimization. Section 5.3 will describe our algorithm in detail, supported by intuition and theory. Finally, we experimentally show the effectiveness of our approach in Section 5.4 and give some points for future work in Section 5.5.

5.1 Problem setting

5.2 Previous work

5.3 Algorithm

5.3.1 Update rule

5.4 Experiments

5.4.1 Handwritten digit classification

5.4.2 Higgs event detection

5.4.3 Sensitivity to hyperparameters

5.4.4 Sensitivity to number of parallel workers

5.5 Future work

Chapter 6

Distributed Keras

Chapter 7

Experiments

Chapter 8

Conclusion

Bibliography

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

[2] Stefan Hadjis et al. "Omnivore: An optimizer for multi-device deep learning on cpus and gpus". In: *arXiv preprint arXiv:1606.04487* (2016).

[3] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[4] Benjamin Recht et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent". In: *Advances in Neural Information Processing Systems*. 2011, pp. 693–701.

[5] Sixin Zhang, Anna E Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD". In: *Advances in Neural Information Processing Systems*. 2015, pp. 685–693.

Appendices