Newton-ADMM: A Distributed GPU-Accelerated Optimizer for Non-Convex Problems

Kuntal Ghosh
School of Electrical Engineering and Computer Science (EECS)
University of Ottawa
Ottawa, Canada
akademik.gk@gmail.com

December 20, 2021

Abstract

In machine learning applications, the first-order optimization techniques - such as gradient descent (SGD) and its variants - are extensively utilized. For, their simplicity and they cost less per iteration. However, in distributed environments, they often need a massive number of iterations incurring associated communication costs. On the other hand, Newton-type methods, which have higher per-iteration computation costs, generally need a much smaller number of iterations; thus it reduces the communication costs. To solve the aforementioned issue, Chih-Hao Fang et al - in a recent paper - presented a new distributed optimizer for classification problems, which associates a GPU-accelerated Newton-type solver with the global consensus formulation of Alternating Direction of Method Multipliers (ADMM). For this project work, a serial non-convex solvers named Newton-MR is being incorporated into the proposed distributed framework for non-convex optimization problems arising from Deep Neural Network..

1 Introduction

The critical component of a large amount of machine learning (ML) applications is estimating parameters of a model from a given data set. The process of parameter estimation problem usually convert into one of finding a minima of an appropriately suitably formulated objective or cost function of the machine learning applications. The main challenges in today's machine learning applications dealing with real big data are: a) very large numbers of parameters of the models - which is essentially equivalent to high dimensional optimization problems, b) huge training data set c) low generalization errors in learning models [1]. To address these challenges quite a size amount of efforts in research has been put. The most prevalent utilized optimization technique in machine learning problem is gradient decent and its variants, such as stochastic gradient descent (SGD). Generally gradient algorithms, that only depends on gradient information are referred to as first-order methods. The curvature information in the form of Hessian or its approximations seem to have gained improvements in performance as shown in their execution time, rate of convergence and predictions of the model in the recent research [5,6,25]. The other key challenges in machine learning optimization problems is distributed nature of the huge training data-set. As it is almost impractical to gather the whole training data set on single node or machine. Also, huge training data set cannot be processed serially on a single node due to the lack of resources, privacy of data as data can be imported to or shared with a centralized node, lessening the time for optimization. To answer these key challenges, there is a requirement for the optimization methods that are appropriately tailored to parallel and distributed computing environments. To solve the aforementioned issue, Chih-Hao Fang et al - in a recent paper - presented a new distributed optimizer for classification problems, which associates a GPU-accelerated Newton-type solver with the global consensus formulation of Alternating Direction of Method Multipliers (ADMM). For this project, I have tried to extend their work to non-convex problems generated by deep neural networks by incorporating serial non-convex solvers Newton-MR into their distributed framework.

2 Literature Review

The main reasons why first-order methods such as gradient descent and its variants are usually used in ML applications are the low cost computing cost for per-iterations and the ease of implementation [27, 25]. However, there are drawbacks too. These methods require quite a large number of iterations in order to attain generalization; mainly due to the fact that they are reactive to ill-conditioning problem i.e. it becomes hard to attain generalization. On the other hand, the second-order methods utilize curvature information by means of Hessian matrix - consequently they are not prone to ill-conditioning problem and are not affected by hyper-parameter tuning [23, 6, 28]. However, in order to process the Hessian matrix, they require higher memory and other computation resources. One way to avoid this issue is to use quasi-Newton methods to approximate the Hessian matrix using the history of gradients. But, for approximating the Hessian matrix, the gradients need to be stored and to satisfy the strong Wolfe condition, extra computation cost is added up. Furthermore, these methods are reported to be unstable when used on conjunction with mini-batches [23, 26]. Of late various distributed solvers have been proposed both first-order and second order methods. Given all of these proposed solvers for first order methods have minimum communication overhead, they have higher communication costs for large number of exchanging of messages for each mini-batch and total number of iteration [14, 20, 28, 29]. Second-order methods are proposed to reduce communication cost as well as to improve the convergence rate [18, 3, 22, 30]. To approximate Newton direction, DANE i.e. DynaNewton - Accelerating Newton's [3] and the accelerated inexact of DANE, called as AIDE [22] use Stochastic Variance Reduced Gradient (SVRG)[30] as the subproblem solver. These methods are often affected to the finetuning of SVRG. Another scheme name DiSCO employs distributed Preconditioned Conjugate Gradient (PCG) to approximate the Newton direction. In this scheme the total number of communications among nodes per PCG cal is commensurate to the number of PCG iterations [7]. There is another proposed method named GIANT, which performs conjugate gradient (CG) on each node and approximates the Newton Direction by means of averaging the solution from each CG call [6]. The experimental results have demonstrated that GIANT performs better than DANE, AIDE and DiSCO. An adaptive approach, which is similar to trust-region methods ane adapts dynamically the auxiliary model to compensate for modeling errors proposed by Dunner et al [8] is claimed to outperform GIANT but it does not do well on sparse data sets. Another recently proposed variant name DINGO [18] is derived by optimization of the gradient's norm as a surrogate function. It does not apply any specific form to the underlying functions and it can applied beyond convexity problem - class of non-convex functions such as invex, which treats convexity as a special sub-class. In addition, it supports the arbitrary distribution of the data across the computing environment. However, it can converge to unwanted stationary points if invexity is not present. Alternating Direction Method of Multipliers (ADMM)[2] is a famous choice for distributed environment. It is based on an augmented Lagrangian framework - which tries to solve the global consensus problem. It solves such problem by alternating iterations on primal or dual variables. As a result, it inherits the advantages of the superior convergence properties of the method multipliers and the decomposability of dual ascent. It only needs one round of communication per iteration. However, the choice of the penalty parameter [31] and the local subproblem solvers affect its performance a lot.

3 Problem Statement

Let's consider a finite sum optimization problem of the following form.

$$\min_{\mathbf{x}\in\mathbb{R}^d} F(\mathbf{x}) \triangleq \sum_{i=1}^{n} f_i(\mathbf{x}) + g(\mathbf{x})$$

where each $f(x)_i$ is a smooth convex function and $g(x)$, which is a strictly convex function is a smooth regularizer. From the perspective of machine learning problem, $f(x)_i$ can be considered as loss corresponding to the i^{th} value. The author of the original work, multi-class classification was chosen using soft-max and corss-entropy loss function as an example of finite sum minimizing problem. Let's say a p dimensional feature vector a with corresponding labels or classes b, taken from C classes. In this classifier, the probability of a belonging to a class $c \in \{1, 2, ..., C\}$is given by:

$$\mathbf{Pr}\left(b = c \mid \mathbf{a}, \mathbf{x}_1, \ldots, \mathbf{x}_C\right) = \frac{e^{\langle \mathbf{a}, \mathbf{x}_c \rangle}}{\sum_{c'=1}^{C} e^{\langle \mathbf{a}, \mathbf{x}_{c'} \rangle}}$$

where $x_c \in \mathbb{R}^p$ is the weight vector of corresponding to class c. For a training dataset $\{a_i, b_i\}_{i=1}^{n} \subset \mathbb{R}^p \ X \ \{1, 2, ..., C\}$, the cross-entropy loss function for $x = [x_1; x_2; ...; x_{(C-1)}] \in \mathbb{R}^{(C-1)p}$ can be expressed as

$$F(\mathbf{x}) \triangleq F(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{C-1})$$
$$= \sum_{i=1}^{n} \left(\log \left(1 + \sum_{c'=1}^{C-1} e^{\langle \mathbf{a}_i, \mathbf{x}_{c'} \rangle} \right) - \sum_{c=1}^{C-1} \mathbf{1}(b_i = c) \langle \mathbf{a}_i, \mathbf{x}_c \rangle \right)$$

$$= \sum_{i=1}^{n} \left(M(\mathbf{a}_i) + \log\left(\alpha(\mathbf{a}_i)\right) - \sum_{c=1}^{C-1} \mathbf{1}(b_i = c) \langle \mathbf{a}_i, \mathbf{x}_c \rangle \right)$$

where

$$M(\mathbf{a}) = \max \left\{ 0, \langle \mathbf{a}, \mathbf{x}_1 \rangle, \langle \mathbf{a}, \mathbf{x}_2 \rangle, \ldots, \langle \mathbf{a}, \mathbf{x}_{C-1} \rangle \right\}$$

$$\alpha(\mathbf{a}) := e^{-M(\mathbf{a})} + \sum_{c'=1}^{C-1} e^{\langle \mathbf{a}, \mathbf{x}_{c'} \rangle - M(\mathbf{a})}.$$

After the training a new data instance a is classified as:

$$b = \arg\max \left\{ \left\{ \frac{e^{\langle \mathbf{a}, \mathbf{x}_c \rangle}}{\sum_{c'=1}^{C-1} e^{\langle \mathbf{a}, \mathbf{x}_{c'} \rangle}} \right\}_{c=1}^{C-1}, 1 - \frac{e^{\langle \mathbf{a}, \mathbf{x}_1 \rangle}}{\sum_{c'=1}^{C} e^{\langle \mathbf{a}, \mathbf{x}_{c'} \rangle}} \right\}$$

4 Proposed Solution:

A novel distributed GPU-accelerated Newton type method based on an ADMM framework was proposed in the original paper. The proposed framework has little communication overhead, minimal resource overhead,and superior convergence properties. Let's suppose there are N nodes in a distributed environment and a data set D, which is split among the N nodes such that the following conditions are met.

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \ldots \cup \mathcal{D}_{\mathcal{N}}$$

$$\begin{aligned} \min \sum_{i=1}^{\mathcal{N}} \sum_{j \in \mathcal{D}_i} & f_j(\mathbf{x}_i) + g(\mathbf{z}) \\ \text{s.t.} \quad \mathbf{x}_i - \mathbf{z} &= 0, \quad i = 1, \ldots, \mathcal{N} \end{aligned}$$

where z is a global variable ensuring consensus among x_i's of each nodes. It enforces a consensus among all nodes such that all the local variables x_i at different node agree with global variable z. Furthermore, there is a penalty variable ρ, which acts like the weight on the disagreement measure between $x_i's$ and global consensus variable z. Thus, ADMM iterations can be expressed as follows.

$$\begin{aligned} \mathbf{x}_i^{k+1} &= \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \frac{\rho_i^k}{2} ||\mathbf{z}^k - \mathbf{x}_i + \frac{\mathbf{y}_i^k}{\rho_i^k}||_2^2, \\ \mathbf{z}^{k+1} &= \arg\min_{\mathbf{z}} g(\mathbf{z}) + \sum_{i=1}^{\mathcal{N}} \frac{\rho_i^k}{2} ||\mathbf{z} - \mathbf{x}_i^{k+1} + \frac{\mathbf{y}_i^k}{\rho_i^k}||_2^2, \\ \mathbf{y}_i^{k+1} &= \mathbf{y}_i^k + \rho_i^k (\mathbf{z}^{k+1} - \mathbf{x}_i^{k+1}). \end{aligned}$$

The algorithms of ADMM and Inexact Newton-type method are as follows:

```
Algorithm 1: ADMM method (outer solver)
   Input      : $\mathbf{x}^{(0)}$ (initial iterate), $\mathcal{N}$ (no. of nodes)
   Parameters: $\beta$, $\lambda$ and $\theta < 1$
1  Initialize $\mathbf{z}^0$ to 0
2  Initialize $\mathbf{y}_i^0$ to 0 on all nodes.
3  foreach $k = 0, 1, 2, \ldots$ do
4      Perform Algorithm 2 with, $\mathbf{x}_i^k$, $\mathbf{y}_i^k$, and $\mathbf{z}^k$ on all
         nodes
5      Collect all local $\mathbf{x}_i^{k+1}$
6      Evaluate $\mathbf{z}^{k+1}$ and $\mathbf{y}_i^{k+1}$ using
8      Distribute $\mathbf{z}^{k+1}$ and $\mathbf{y}_i^{k+1}$ to all nodes.
9      Locally, on each node, compute spectral step sizes
         and penalty parameters
10 end
```

```
Algorithm 2: Inexact Newton-type Method
   Input      : $\mathbf{x}^{(0)}$
   Parameters: $0 < \beta, \theta < 1$
1  foreach $k = 0, 1, 2, \ldots$ do
2      Form $\mathbf{g}(\mathbf{x}^{(k)})$ and $\mathbf{H}(\mathbf{x}^{(k)})$
3      if $\|\mathbf{g}(\mathbf{x}^{(k)})\| < \epsilon$ then
4          STOP
5      end
6      Update $\mathbf{x}^{(k+1)}$
7  end
```

Newton-type methods have better convergence rates and are not reactive to ill-conditioned problems. For, such methods use curvature information by means of Hessian matrix. However, both the computational cost and memory footprint will be higher per information of these methods if Hessian matrix is explicitly formed and calculated. To resolve this issue, they proposed a Hessian-free Newton-type method to calculate ${x_i}^{k+1}$. Basically, given a vector $v \in \mathbb{R}^d$, the goal is to compute the Hessian-vector product Hv without explicitly forming the Hessian H. Thus, if

$$h(\mathbf{a}, \mathbf{x}) := \frac{e^{\langle \mathbf{a}, \mathbf{x} \rangle - M(\mathbf{x})}}{\alpha(\mathbf{a})},$$

$$\mathbf{V} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{v}_1 \rangle & \langle \mathbf{a}_1, \mathbf{v}_2 \rangle & \dots & \langle \mathbf{a}_1, \mathbf{v}_{C-1} \rangle \\ \langle \mathbf{a}_2, \mathbf{v}_1 \rangle & \langle \mathbf{a}_2, \mathbf{v}_2 \rangle & \dots & \langle \mathbf{a}_2, \mathbf{v}_{C-1} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{a}_n, \mathbf{v}_1 \rangle & \langle \mathbf{a}_n, \mathbf{v}_2 \rangle & \dots & \langle \mathbf{a}_n, \mathbf{v}_{(C-1)} \rangle \end{bmatrix}_{n \times (C-1)}$$

and

$$\mathbf{W} = \begin{bmatrix} h(\mathbf{a}_1, \mathbf{x}_1) & h(\mathbf{a}_1, \mathbf{x}_2) & \dots & h(\mathbf{a}_1, \mathbf{x}_{C-1}) \\ h(\mathbf{a}_2, \mathbf{x}_1) & h(\mathbf{a}_2, \mathbf{x}_2) & \dots & h(\mathbf{a}_2, \mathbf{x}_{C-1}) \\ \vdots & \vdots & \ddots & \vdots \\ h(\mathbf{a}_n, \mathbf{x}_1) & h(\mathbf{a}_n, \mathbf{x}_2) & \dots & h(\mathbf{a}_n, \mathbf{x}_{C-1}) \end{bmatrix}_{n \times (C-1)}$$

to get

$$\mathbf{Hv} = \text{vec}\left(\mathbf{A}^T \mathbf{U}\right)$$

we compute

$$\mathbf{U} = \mathbf{V} \odot \mathbf{W} - \mathbf{W} \odot \Big(\big((\mathbf{V} \odot \mathbf{W}) \, \mathbf{e} \big) \mathbf{e}^T \Big)$$

where $v = [v_1; v_2; ...v_{C-1}] \in \mathbb{R}^d, v_i \in \mathbb{R}^p, i = 1, 2, ...C-1, e \in \mathbb{R}^{C-1}$ and $A \in \mathbb{R}^{nxp}$ is a row vector corresponding to the i^{th} data point i.e. $A^T = [a_1, a_2, ...a_n]$. To efficiently compute the Hessianvector product Hv, using General Matrix to Matrix Multiplication (GEMM) operations, Pytorch's Basic Linear Algebra Subprograms (BLAS) interface to the GPUs was used.

5 Proposed Modification

Newton-CG and other Newton-type variants are limited to making convexity assumptions. Thus, they are not always applicable to non-convex optimization problems which are commonly generated by Deep Learning models. Therefore, the authors of [11] recently proposed a new method named Newton-MR (Minimu Residual) making two simple modifications of Newton-CG. For non-convex problems, Newton-MR can be plugged into the proposed Newton-ADMM framework in lieu of Inexact Newton-type method. That is making the following changes.

$x_i^{k+1} = x_i^k + \alpha_k p_k$, where $p_k = -[\nabla^2 f(x_i^k)] \nabla f(x_i^k)$ and $0 \leq \alpha \leq 1$

6 Experimental Evaluation

The author of [32] implemented Newton-ADMM using PyTorch/0.3.0.post4 with Message Passing Interface (MPI) back end. They tested the performance of Newton-ADMM on two hardware platforms - one for weak and another for strong scaling. Furthermore, Newton-ADMM was validated using real-world data-sets such as MNIST, CIFAR-10, HIGGS, E18 and was compared with state-of-the-art first-order and second-order optimizer such as AIDE, GIANT, InExact DANE etc. Some of their empirical results are included below. However, I was unable to run their implementation of Newton-ADMM's code - in spite of my painstaking efforts. Hence, I could not empirically verify if Newton-MR will improve the performance of Newton-ADMM.

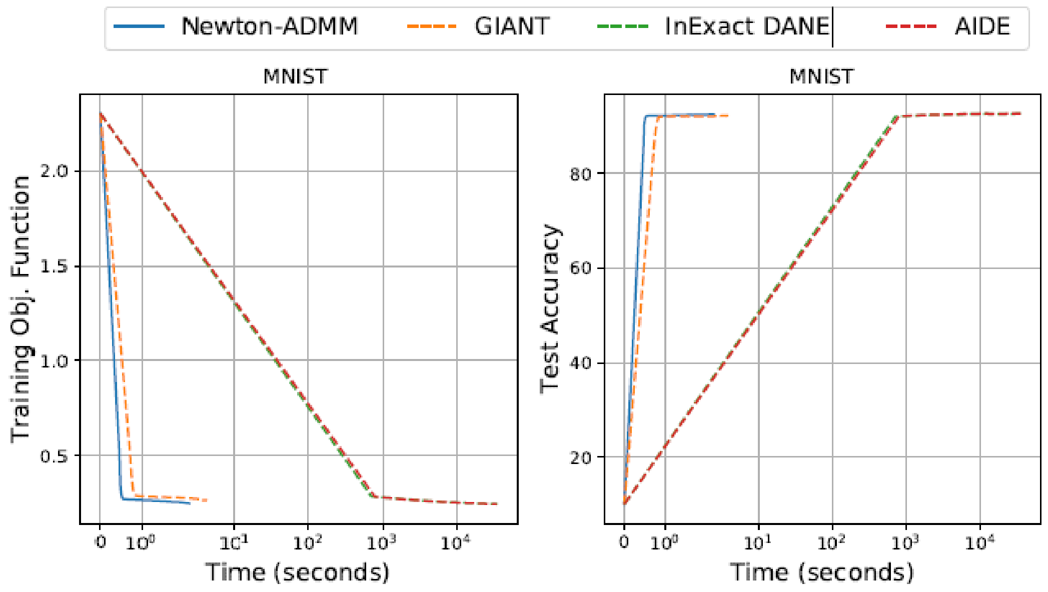

Figure 1: Training and test accuracy comparisons on MNIST

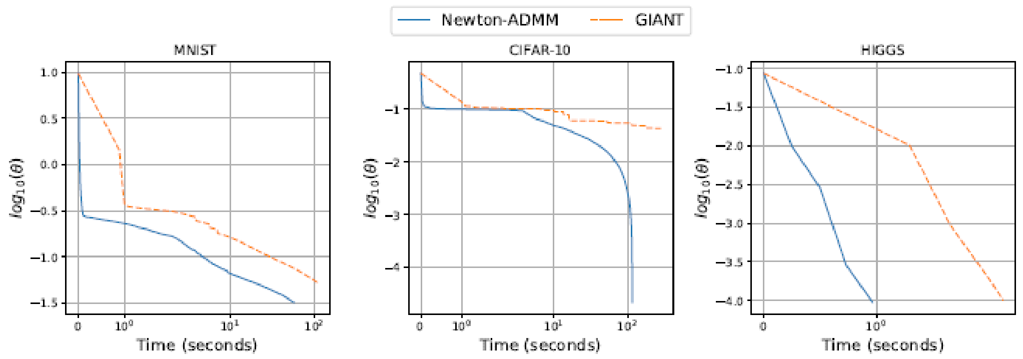

Figure 2: Convergence performances on MNIST

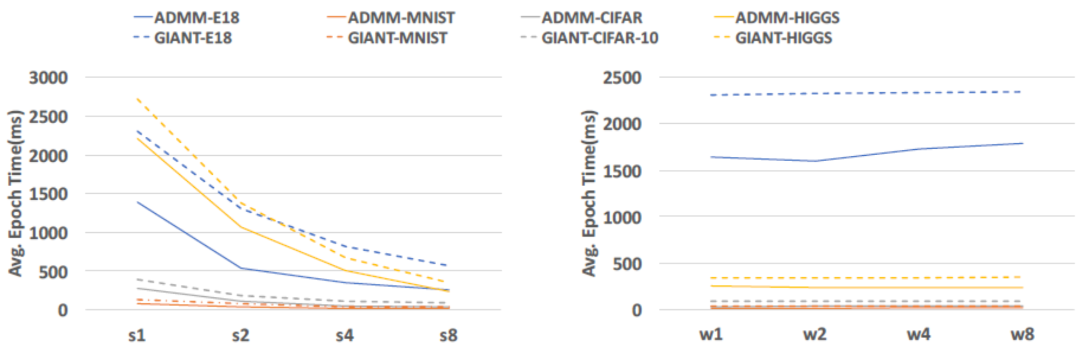

Figure 3: Strong and Weak Scaling for Newton-ADMM and GIANT

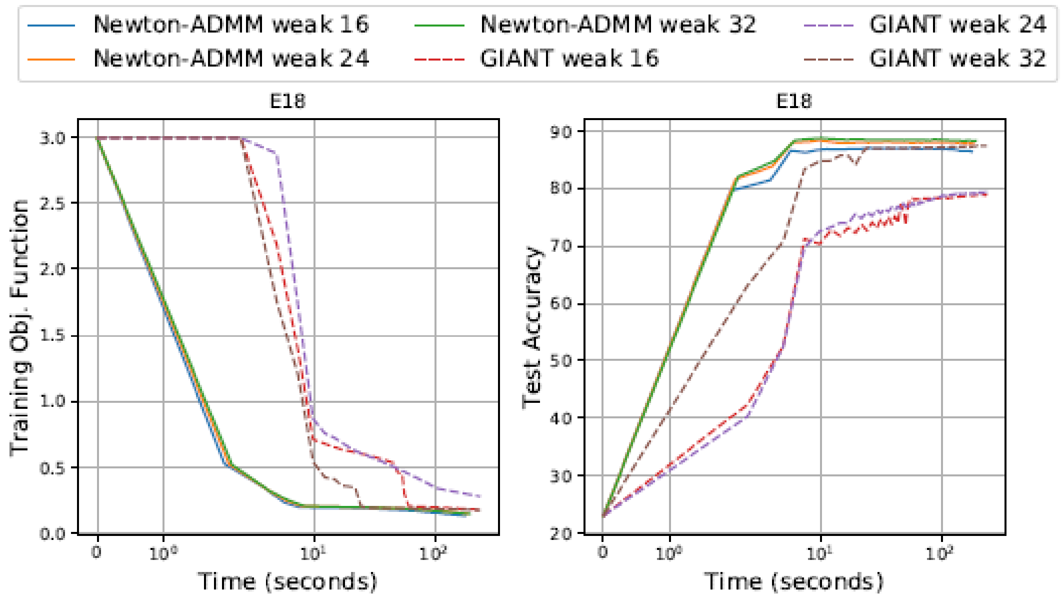

Figure 4: Comparisons of scalability of Newton-ADMM on largest data set (E18)

7 Conclusions

In the project I have tried to show how Newton-MR method can be Incorporated with Newton-ADMM to resolve the issue of non-convex optimization arising from Deep Learning models. I would like to provide the following insights for future works.

1. To find a more GPU efficient way to compute Hessian-vector product than General Matrix to Matrix Multiplication (GEMM)
2. Empirically test if Newton-MR improves the performance of Newton-ADMM
3. Investigates if Newton-ADMM can work well with other Deep Learning models or not.
4. Investigates if other Newton-type optimization methods can be incorporated with Newton-ADMM or not.
5. Examine the performance of Newton-ADMM with larger and higher dimensional data-sets.

8 Acknowledgement

I am infinitely grateful to our professor Dr. Frank Dehne for teaching me parallel computing and clearing all my doubts - else I could not do this project. For, this the first time ever I have been introduced to parallel computing. Furthermore, my sincerest gratitude to Chih-Hao Fang - the principle author of the main paper of this project for helping understand his code. Last but no means least, my supervisor for allowing me to use Compute Canada resources for this project.

References

[1] Fang, C.H., Kylasa, S.B., Roosta, F., Mahoney, M.W. and Grama, A., 2020, November. Newton-ADMM: A distributed GPU-accelerated optimizer for multiclass classification problems. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-12). IEEE.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: Foundations and Trends® in Machine learning 3.1 (2011), pp. 1-122.

[3] Hadi Daneshmand, Aurelien Lucchi, and Thomas Hofmann. "DynaNewton-Accelerating Newton's Method for Machine Learning". In: arXiv preprint arXiv:1605.06561 (2016).

[4] Fred Roosta, Yang Liu, Peng Xu, and Michael W Mahoney. ”Newton-MR: Newton’s Method Without Smoothness or Convexity”. In: arXiv preprint arXiv:1810.00303 (2018).

[5] Sudhir B Kylasa. ”HIGHER ORDER OPTIMIZATION TECHNIQUES FOR MACHINE LEARNING”. PhD thesis. Purdue University Graduate School, 2019.

[6] Farbod Roosta-Khorasani and Michael W Mahoney. Sub-sampled Newton methods. Mathematical Programming, 174(1-2):293326, 2019.

[7] Yuchen Zhang and Xiao Lin. ”DiSCO: Distributed optimization for self-concordant empirical loss”. In: International conference on machine learning. 2015, pp. 362-370.

[8] Celestine D¨unner, Aurelien Lucchi, Matilde Gargiani, An Bian, Thomas Hofmann, and Martin Jaggi. ”A Distributed Second-Order Algorithm You Can Trust”. In: arXiv preprint arXiv:1806.07569 (2018).

[9] L´eon Bottou, Frank E Curtis, and Jorge Nocedal. ”Optimization methods for large-scale machine learning”. In: arXiv preprint arXiv:1606.04838 (2016).

[10] Hadi Daneshmand, Aurelien Lucchi, and Thomas Hofmann. ”DynaNewton-Accelerating Newton’s Method for Machine Learning”. In: arXiv preprint arXiv:1605.06561 (2016).

[11] Fred Roosta, Yang Liu, Peng Xu, and Michael W Mahoney. ”Newton-MR: Newton’s Method Without Smoothness or Convexity”. In: arXiv preprint arXiv:1810.00303 (2018).?

[12] L. Angelani, R. Di Leonardo, G. Ruocco, A. Scala, and F. Sciortino. Saddles in the Energy Landscape Probed by Supercooled Liquids. Physical review letters, 85(25):5356, 2000.

[13] Y. Arjevani, O. Shamir, and R. Shi. Oracle Complexity of Second-Order Methods for Smooth Convex Optimization. Mathematical Programming, pages 134, 2017.?

[14] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. ”Revisiting distributed synchronous SGD”. In: arXiv preprint arXiv:1604.00981 (2016).?

[15] Y. Bengio et al. Learning deep architectures for AI. Foundations and trends R in Machine Learning, 2(1):1127, 2009.?

[16] S. Bellavia, C. Cartis, N. I. M. Gould, B. Morini, and Ph. L. Toint. Convergence of a regularized Euclidean residual algorithm for nonlinear least-squares. SIAM Journal on Numerical Analysis, 48(1):129, 2010.?

[17] Groueix, T., Fisher, M., Kim, V., Russell, B., Aubry, M.: Atlasnet: A papier-m$^a cheapproachtolearning3dsurfacegeneration.In: CVPR$2018(2018)

[18] Rixon Crane and Fred Roosta. "DINGO: Distributed Newton-Type Method for Gradient-Norm Optimization". In: Proceedings of the Advances in Neural Information Processing Systems. Accepted. 2019.?

[19] D. Calvetti, B. Lewis, and L. Reichel. L-curve for the MINRES method. In Advanced Signal Processing Algorithms, Architectures, and Implementations X, volume 4116, pages 385396. International Society for Optics and Photonics, 2000.?

[20] Priya Goyal, Piotr Doll´ar, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, large minibatch SGD: training imagenet in 1 hour". In: arXiv preprint arXiv:1706.02677 (2017).?

[21] K. v. d. Doel and U. Ascher. Adaptive and stochastic algorithms for EIT and DC resistivity problems with piecewise constant solutions and many measurements. SIAM J. Scient. Comput., 34:DOI: 10.1137/110826692 2012?

[22] Sashank J Reddi, Jakub Kone?cn'y, Peter Richt´arik, Barnab´as P´ocz´os, and Alex Smola. "AIDE: Fast and communication efficient distributed optimization". In: arXiv preprint arXiv:1608.06879 (2016).?

[23] B. Kylasa, F. Roosta-Khorasani, M. W. Mahoney, and A. Grama. GPU Accelerated Sub-Sampled Newton's Method. arXiv preprint arXiv:1802.09113. Accepted for publication in the Proceedings of SIAM SDM 2019.?

[24] H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak- Lojasiewicz condition. Joint European Conference on Machine Learning and Knowledge Discovery in Databases,pages 795811, 2016.?

[25] S´ebastien Bubeck et al. "Convex optimization: Algorithms and complexity". In: Foundations and Trends® in Machine Learning 8.3-4 (2015), pp. 231-357

[26] Albert S Berahas, Raghu Bollapragada, and Jorge Nocedal. An Investigation of Newton-Sketch and Subsampled Newton Methods. arXiv preprint arXiv:1705.06211, 2017.

[27] Amir Beck. First-Order Methods in Optimization. Vol. 25. SIAM, 2017

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

[29] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. "How to scale distributed deep learning?" In: arXiv preprint arXiv:1611.04581 (2016).

[30] Shusen Wang, Farbod Roosta-Khorasani, Peng Xu, and Michael W Mahoney. "GIANT: Globally Improved Approximate Newton Method for Distributed Optimization". In: Advances in Neural Information Processing Systems (NIPS). 2018, pp. 2338–2348.

[31] Zheng Xu, Gavin Taylor, Hao Li, Mario Figueiredo, Xiaoming Yuan, and Tom Goldstein. “Adaptive consensus ADMM for distributed optimization”. In: arXiv preprint arXiv:1706.02869 (2017).

[32] Fang, C.H., Kylasa, S.B., Roosta, F., Mahoney, M.W. and Grama, A., 2020, November. Newton-ADMM: A distributed GPU-accelerated optimizer for multiclass classification problems. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-12). IEEE.