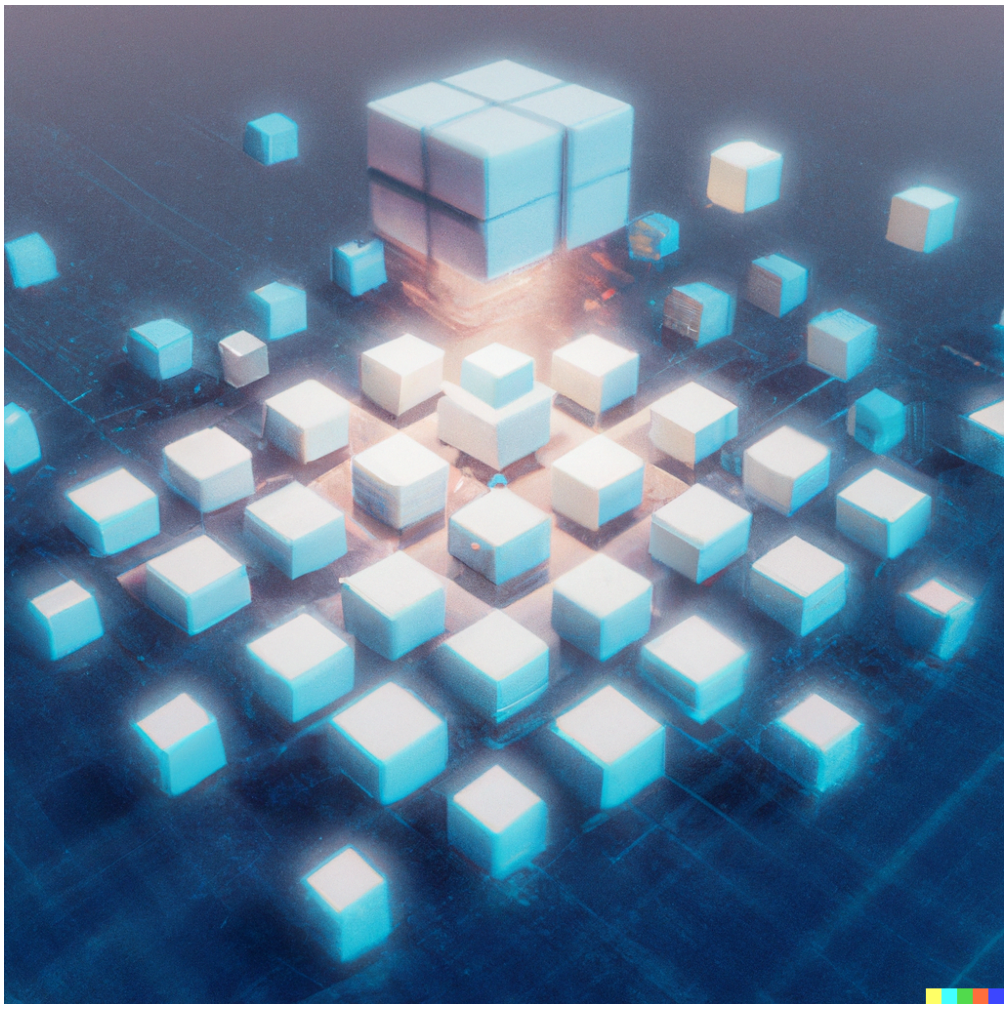

Investigating a Byzantine Resilient Framework for the Adam Optimizer

Master's thesis in Data Science and AI

Basil FABRIS

Department of Mathematical Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

BASIL FABRIS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

Investigating a Byzantine Resilient Framework for the Adam Optimizer
Master Thesis in Mathematics
Basil FABRIS

Supervisor: Sadegh Farhadkhani, Distributed Computing Lab at EPFL
Supervisor: Axel Ringh, Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg
Examiner: Axel Ringh, Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image generated by Dalle-2 with the prompt "Many Artificial Intelligences fusing to create a single one. High Resolution. Voxel art."

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Investigating an Adam-based Byzantine Resilient Framework for Distributed Machine Learning
Master Thesis in Mathematics
Basil FABRIS
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Over the past few years, the utilization of Machine Learning has experienced tremendous growth across various domains, ranging from engineering to marketing. This widespread adoption of Machine Learning has been made possible by advancements in hardware, which have facilitated the training of increasingly large machine learning models. However, these models have given rise to larger datasets and raised concerns regarding data safety and privacy. To address these challenges, Distributed Machine Learning has emerged as a promising solution. By training models locally on participants' devices, Distributed Machine Learning enhances privacy as raw data remains on the respective devices, while also reducing the need for specialized and novel hardware, as most of the computation takes place on participants' devices. Nonetheless, due to the lack of control over participants, Distributed Machine Learning is susceptible to attacks carried out by misbehaving (byzantine) participants. This research introduces two Adam-based optimization frameworks for Distributed Machine Learning. Both frameworks are evaluated through empirical analysis using homogeneous and heterogeneous datasets, and their performance is assessed against multiple state-of-the-art attacks. Additionally, we present preliminary evidence of convergence for DRA (Distributed Robust Adam) on homogeneously distributed data.

Keywords: Machine Learning, Distributed, Byzantine Resilience, Adam Optimisation Algorithm.

Acknowledgements

I am deeply grateful to my supervisors, Sadegh Farhadkhani, PhD candidate at the Distributed Computing Lab at EPFL, and Assistant Professor Axel Ringh, for their support and guidance.

Basil FABRIS, Gothenburg, 06 2023

Nomenclature

Algorithm

α	Learning rate
β	Exponential decay rate
$\hat{m}$	Bias corrected first moment
$\hat{v}$	Bias corrected second moment
$\mathcal{F}(\cdot)$	Aggregation function
$\mathcal{L}(\theta)$	Loss on the complete data set
m	First moment
m^*	Aggregated first moment
q	Descent direction
v	Second moment
v^*	Aggregated second moment

Global

$[d]$	Range from 0 to d
$\mathcal{B}$	Set of Byzantine workers
$\mathcal{H}$	Set of honest worker
$\mathcal{W}$	Set of worker
θ	Model parameter vector
d	Number of model parameters
f	Number of malicious worker
h	Arbitrary honest worker i.e. $h \in \mathcal{H}$
M	Set of moments received by the server
n	Number of workers
t	Arbitrary time step i.e. $t \in [T]$
w	Arbitrary worker i.e. $w \in \mathcal{W}$

Proof

g	Gradient on a data point or batch

Contents

Nomenclature **viii**

1 Background **1**
1.1 Traditional Machine Learning . . . 1
1.1.1 Supervised Machine Learning . . . 1
1.1.2 Optimisation Algorithms . . . 2
1.2 Distributed Machine Learning . . . 3
1.2.1 RESAM . . . 5
1.2.2 Aggregation Methods . . . 5
1.2.2.1 Byzantine resilient aggregation . . . 5
1.2.2.2 Pre-aggregation . . . 7
1.2.3 Byzantine Behaviors . . . 8

2 Background **11**
2.1 Traditional Machine Learning . . . 11
2.1.1 Supervised Machine Learning . . . 11
2.1.2 Optimisation Algorithms . . . 12
2.2 Distributed Machine Learning . . . 13
2.2.1 RESAM . . . 15
2.2.2 Aggregation Methods . . . 15
2.2.2.1 Byzantine resilient aggregation . . . 15
2.2.2.2 Pre-aggregation . . . 18
2.2.3 Byzantine Behaviors . . . 18

3 Adam-based Distributed Optimization Algorithms **21**
3.1 DRA and MRA . . . 21
3.2 DRA Convergence on Homogeneous Data . . . 23
3.2.1 Lemmas . . . 25
3.2.2 Proof of Theorem 3.2.1 . . . 28
3.2.3 Open Problems . . . 33
3.2.3.1 Upper bound on ξ_t . . . 33
3.2.3.2 Bounding the difference in descent direction between two workers . . . 33

4 Numerical Experiments and Results **35**
4.1 MNIST . . . 35
4.1.1 Setup . . . 35

4.1.2 Results and Discussion 37
4.1.2.1 Homogeneously Distributed Data 37
4.1.2.2 Heterogeneously Distributed Data 40
4.2 CIFAR-10 42
4.2.1 Setup 42
4.2.2 Results and Discussion 44
4.2.2.1 Homogeneously Distributed Data 44
4.2.2.2 Heterogeneously Distributed Data 45

5 Conclusion **47**

Bibliography **49**

A Appendix 1 **I**

1
Background

Before describing our frameworks, some background needs to be covered. The first section describes how traditional or centralized machine learning works. Then, section 2.2 describes Distributed Machine Learning before delving into RESAM [1] and the different parts needed to make the framework byzantine resilient.

1.1 Traditional Machine Learning

While there are many approaches to training a machine learning model (supervised, unsupervised, self-supervised, etc), supervised is arguably the easiest to work with, especially if we already have access to the data. Hence, this work focuses on this method. Therefore the following section introduces supervised learning and then explains how we can minimise the error using an optimisation algorithm.

1.1.1 Supervised Machine Learning

Machine Learning is essentially the estimation of an unknown function $M : X \to Y$. Many approaches exist, the easiest being supervised learning. This technique relies on a labelled set where each data point $x_i \in X$ matches an output $y_i \in Y$. The models' output is expressed as follows $M(x_i) = y_i + \mu_i$ where μ_i denotes random noise.

For example, in a task of cats and dogs image classification, the data point x would be a picture of a cat or a dog, and the corresponding label y is 0 if the image is a cat and 1 if it is a dog. Another example is the estimation of a company's stock price. A data point x could be historical prices and the performance of similar stocks, and the matched value y is the stock price at an arbitrary time.

The most popular machine learning approach to these problems is Neural Networks or Deep Learning. The simplest component of a neural network is the perceptron

$$output = \sigma(W^T x + b), \tag{1.1}$$

where σ is the activation function, W is the weight matrix and b the bias vector. Without going into too much detail, the activation function introduces non-linearity. Nowadays, many perceptrons are stacked in layers to form much larger models. W and b are the trainable or learnable parameters we will finetune such that the output matches y_i as well as possible. We group all the trainable parameters in θ to

facilitate the notation.

However, minimizing the error requires measuring it. Many approaches exist to compare the prediction $M_\theta(x_i)$ and the actual value y_i such as the 1- and 2-norm. We aggregate the measures of all the distances to compute the loss function $\mathcal{L}(\theta)$,

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{(x,y)\in X\times Y} d(M_\theta(x), y), \tag{1.2}$$

where n is the number of data samples we have and d is the distance between $M_\theta(x)$ and y, measured using the Euclidean distance for example. Now that we have the loss function, we can finetune the parameters θ to reduce the model's error. In other words, we need to find θ that minimizes the loss. Many techniques exist and are further discussed in the following section.

1.1.2 Optimisation Algorithms

The finetuning of θ is usually an iterative process which stops once the loop reaches some type of stopping criterion, such as the maximum number of rounds. Due to the possibly large number of parameters, computing the Hessian at each iteration requires significant computational power, possibly slowing the training session down. Therefore, machine learning relies on first-order methods as they avoid computing the Hessian, thereby saving memory and computational power. Arguably, the most known first-order optimisation algorithm is Gradient Descent:

$$\theta_t = \theta_{t-1} - \alpha\nabla\mathcal{L}(\theta_{t-1})$$

where $\nabla\mathcal{L}(\theta_t)$ is the gradient over the entire data. However, the ever-growing data sets made it very expensive to compute the gradient on the entire set. A solution is Stochastic Gradient Descent; the idea is to split the training set into stratified chunks, called batches, and perform an update on each batch. This method reduces the memory required since we only use a chunk of the data, and the model prediction happens faster since it has fewer data points to predict. SGD being iterative, it performs an update on each chunk, also called a batch. Once the algorithm used each batch once, the algorithm performed an epoch. The algorithm goes back to the first batch and performs a new epoch. Denoting the loss computed on the batch corresponding to time step t by $\mathcal{L}_t(\theta_{t-1})$, we have

$$\theta_t = \theta_{t-1} - \alpha\nabla\mathcal{L}_t(\theta_{t-1}).$$

The growth of the Machine Learning field led to the development of various optimisation algorithms such as RMSProp [2] and Adam [3]. Both use moving averages called momentums. RMSProp solves the vanishing and exploding gradient problem introduced by complex functions such as Neural Networks, by dividing the gradients by the square root of the second momentum,

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla\mathcal{L}_t(\theta_{t-1})\right)^2. \tag{1.3}$$

The update formula for RMSProp is as follows

$$\theta_t = \theta_{t-1} - \frac{\alpha}{v_t}\nabla\mathcal{L}_t(\theta_{t-1}).$$

The division scales small gradients up and large ones down. Adam is an extension of RMSProp as it uses the first momentum,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\mathcal{L}_t(\theta_{t-1}), \tag{1.4}$$

instead of the gradient in the numerator. The Adam algorithm is described in Algorithm 10.

Algorithm 1 Adam as described in [3]: g_t^2 stands for the element-wise square, ϵ is usually 10^{-10}

Require: Learning rate: α, Exponential decay rates: $\beta_1, \beta_2 \in [0, 1)$, Objective function $\mathcal{L}$, Initial model parameters: θ_0

$m_0 := 0$
$v_0 := 0$
$t := 0$
while θ_t not converged **do**
    $t := t + 1$
    $g_t := \nabla\mathcal{L}_t(\theta_{t-1})$
    $m_t := \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$
    $v_t := \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$
    $\hat{m}_t := m_t/(1-\beta_1^t)$
    $\hat{v}_t := v_t/(1-\beta_2^t)$
    $\theta_t := \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$
end while
return θ_t

Computing $\hat{m}_t$ and $\hat{v}_t$ corrects the bias toward zero introduced by initializing m_0 and v_0 to zero.

1.2 Distributed Machine Learning

The increasing data set sizes and model complexity led to scalability bottlenecks due to the limited computational power and memory a single hardware device (CPU, GPU) has. Distributed Machine Learning solves this bottleneck by leveraging the computational power of various computational devices to compute an update vector. Thanks to this, Distributed Machine Learning became the main approach to training large models.

The ability to leverage the power from multiple devices inspired the fields to develop methods where each participant trains a model cutting the need for data gathering. The most known approach is Federated Learning [4]. Federated Learning works in rounds where each participant trains a model locally on a batch or epoch and

sends the updated model to the server. The server aggregates the models, using FedAvg [5] for example, and sends the model back to the participants who perform another update. Another approach is to aggregate descent directions, instead of model weights, and have the server update the model. Figure 2.1 shows an iteration of this approach. The server receives all the descent directions (step 1), aggregates them (step 2), updates the model (step 3) and sends the new version of the model to the workers (step 3).

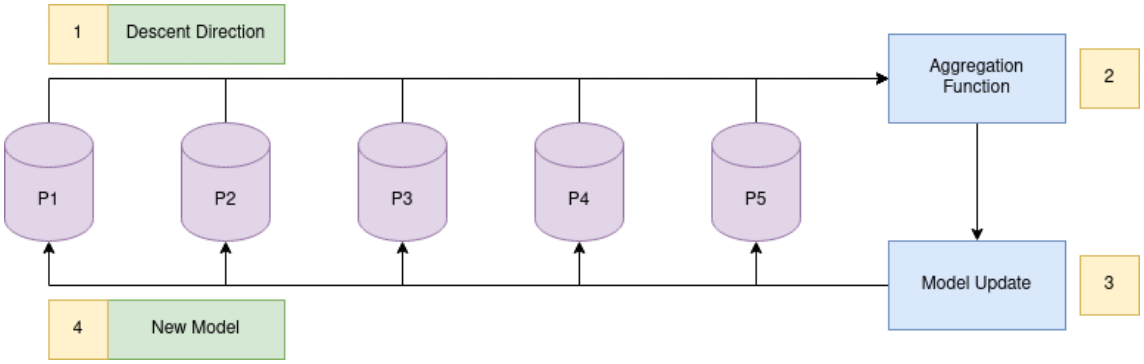

Figure 1.1: An iteration in a Distributed Machine Learning workflow. The purple blocks represent the participants, the blue blocks the work performed by the server, the yellow blocks the steps, and the green blocks show what is shared. In the first step, the workers send the locally computed descent direction. The server aggregates them in Step 2, updates the model in Step 3, and sends the new model to each worker in Step 4.

While it is tempting to use the average to aggregate the descent directions, this measure is unsafe as it is outlier sensitive. Instead, one would use a byzantine resilient aggregation method, i.e. able to counter byzantine descent directions. This is where the RESAM (RESilient Averaging of Momentums) [1] framework comes in.

1.2.1 RESAM

RESAM leverages the first moment and a Byzantine aggregation function to train a model even in the presence of Byzantine participants. The first reduces the variance in locally computed directions. Hence, independently of the impact of the byzantine nodes on a given update, the local gradients are guaranteed to approximately follow the previous trajectory. The latter aims at approximating the descent direction by aggregating the locally computed descent directions, even in the presence of byzantine participants. Such an aggregation function is denoted by $\mathcal{F}(\cdot)$.

Algorithm 2 RESAM

Require: Initial model parameters $\theta_1^{[d]}$, Set of workers $\mathcal{W}$, Number of steps T, Learning rates: $\{\alpha_1, ..., \alpha_t\}$, Exponential decay rates: $\beta \in [0, 1)$, Aggregation Function: $\mathcal{F}(\cdot)$, $\forall w \in \mathcal{W} : m_0^{(w)} = 1$

for each t = 1,...,T **do**

1. Send θ_t to each worker
2. Each worker computes $m_t^{(w)} = \beta m_{t-1}^{(w)} + (1-\beta) g_t^{(w)}$ and sends it to the server. $m_t^{(w)}$ might an arbitrary value in case w is byzantine.
3. The server computes θ_{t+1}: $\theta_{t+1} = \theta_t - \alpha_t \mathcal{F}(m_t^{(1)}, ..., m_t^{(n)})$

end for

return uniformly randomly chosen $\{\theta_0, ..., \theta_T\}$

1.2.2 Aggregation Methods

We distinguish two types of aggregation methods; byzantine resilient aggregation and pre-aggregation. As its name suggests, the first type aims at making the framework byzantine resilient. The second mitigates the impact of heterogeneous data.

1.2.2.1 Byzantine resilient aggregation

The simplest method one can think of is the unweighted average. However, it is sensitive to extreme values. Hence an attacker could send a very large or small m_t to impact the training phase. Therefore, we need more sophisticated methods. [1] introduces the notion of (f, ϵ)-Resilient averaging measuring the byzantine resilience of aggregation functions. It is defined as follows:

Definition 1.2.1. (f, ϵ)***-Resilient averaging***
For a total of n workers of which f are byzantine such that $f < n$ and a real value $\lambda \leq 0$, an aggreation function $\mathcal{F}(\cdot)$ is called (f, ϵ)-Resilient averaging if for any collection of n vectors $x_1, ..., x_n$ and any set $S \subset \{1, ..., n\}$ of size $n - f$:

$$||\mathcal{F}(x_1, ..., x_n) - \bar{x}_S|| \leq \lambda \max_{i,j \in S} ||x_i - x_j|| \tag{1.5}$$

Where $\bar{x}_S := \frac{1}{|S|} \sum_{i \in S} x_i$ and $|S|$ is the cardinality of S

In our case, S denotes the set of honest workers. The value of λ is the resilience coefficient. Using this definition, the authors of [1] computed the byzantine resilience

of various aggregation methods enabling us to select the most byzantine-resilient aggregation functions.

Coordinate-Wise Trimmed mean (CWTM) With $\lambda = \frac{f}{n-f}\min(2\sqrt{n-f}, \sqrt{d}) \geq \frac{f}{n-f}$, Coordinate-Wise Trimmed mean [6] is the most resilient aggregation function investigated in [1].The algorithm ascendingly sorts the parameters coordinate-wise and averages the parameters in the range $[f, n-f]$. Let M_t denote the set of all the moments received from the workers at time t and m^* the output from the aggregation methods which the server will use to compute the update.

Algorithm 3 Coordinate-Wise Trimmed Mean at a coordinate $i \in [d]$

Require: M_t

$M_{new,i} = sort(M_{t,i})$

$m^*_{t,i} = \frac{1}{n-2f} \sum_{q \in [f, n-f]} m_{new,i,q}$

return $m^*_{t,i}$

While it still performs averaging, sorting beforehand and selecting a range in the middle ensures extreme values are filtered out. The main downside is that it does not scale well due to the need to sort all the vectors element-wise at each iteration.

Coordinate Wise Median (CWM) This methods comes from [6] and has a resilience coefficient of $\lambda = \frac{n}{2(n-f)}\min(2\sqrt{n-f}, \sqrt{d})$. As its name suggests, the server selects the median at each coordinate. The advantage is that the median is less susceptible to outliers and, since we assume $f < \frac{n}{2}$, otherwise it is impossible for the algorithm to converge, we are very likely to select the descent direction/moment from an honest worker.

Algorithm 4 Coordinate-Wise Median at a coordinate $i \in [d]$

Require: M_t

$m^*_{t,i} = \text{median}(M_{t,i})$

return $m^*_{t,i}$

Minimum Diameter Averaging (MDA) With a resilience score of $\frac{2f}{n-f}$, Minimum Diameter Averaging [7] only differs from the optimal λ by a constant factor making it an ideal candidate for our framework. The rule averages a subset $S_t \subset M_t$ containing $n-f$ vectors such that S_t has the smallest diameter. Using the same notation as in [1]:

Algorithm 5 Minimum Diameter Averaging

Require: M_t

$S = \underset{\substack{S\subset\{1,\dots,n\}\\|S|=n-f}}{\arg\min} \left\{\max_{i,j\in S} ||m_t^{(i)} - m_t^{(j)}||\right\}$

$m^* = \frac{1}{n-2f} \sum_{k\in S} m_t^{(k)}$

return m_t^*

Mean Around Median (MEAMED) The last aggregation rule we consider is Mean Around Median [8] which has a resilience score of $\frac{2f}{n-f}\min(2\sqrt{n-f}, \sqrt{d})$. Its name is also very descriptive, as the server averages the $n-f$ points closest to the median.

Algorithm 6 Mean around Median at a coordinate $i \in [d]$

Require: M_t

$med_{t,i} = median(P_{t,i})$

$C = \underset{\substack{m_{t,i}\in M_{t,i}\\|C|=n-f}}{\arg\min} |med_{t,i} - m_{t,i}|$

$m^*_{t,i} = \frac{1}{n-f} \sum_{m_{t_i}\in C} m_{t,i}$

return $m^*_{t,i}$

The intuition is similar to CWTM. The median is assumed to come from a non-byzantine participant. It should be relatively close to the values of the other non-byzantine workers. Taking their average provides a better approximation of the real gradient.

1.2.2.2 Pre-aggregation

Due to the lack of control over the participants and their data, it is safe to assume that the data distribution differs between the workers. A solution is bucketing [9]; the descent directions are randomly placed into an arbitrary number of buckets before being averaged, and the result is passed to a byzantine resilient aggregation function. However, a recent method called Nearest Neighbour Mixing (NNM) [10] proved to outperform bucketing, making it a good candidate pre-aggregation method for our framework. For each $m_t^{(w)}$, we find the $n-f$ closest vectors in M_t. Then we compute the average between them.

Algorithm 7 NNM

Require: $\{m_t^{(1)}, ..., m_t^{(n)}\}$
1: Initialize: M_{new}
2: **for each** $w \in \mathcal{W}$ **do**
3: $\quad Y = \underset{\substack{|Y|=n-f \\ x \in \{m_t^{(1)}, ..., m_t^{(n)}\}}}{\arg\min} \quad (|m_t^{(w)} - x|)$
4: $\quad M_{new,w} = \frac{1}{n-f} \sum_{y \in Y} y$
5: **end for**
6: **return** M_{new}

Now that we described all methods to counter Byzantine participants, we have to discuss and detail what they are and how they can attack our training loop.

1.2.3 Byzantine Behaviors

Several state-of-the-art attacks can be launched against distributed learning systems. For example, Empire [11] and Little [12] are two such attacks one can use to disrupt the training process.

Little Recall that the core assumption for CWTM is that byzantine gradients are either very large or small. Hence by sorting them and discarding the smaller and larger elements, the aggregation function should filter the byzantine descent directions out. However, if one has access to multiple workers computing legitimate descent directions, we could use these to estimate the ones computed by the honest worker. Thereby enabling the attacker to increase or decrease the byzantine directions as much as possible while still remaining in the range CWTM uses. Figure 2.2 shows five (two byzantine (red) and three honest (green)) sorted descent directions. The big circle represents the directions selected by CWTM. CWTM averages two green dots close to each other and a red dot placed right at the limit, leading to the red dot having a significant impact on the average.

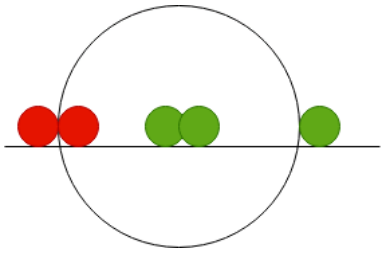

Figure 1.2: Example of how little impacts CWTM. The two red dots are byzantine while the green are honest. The large circle shows the descent direction that will be taken into account.

To estimate the diameter of the large circle, Little computes the coordinate-wise standard deviation and subtracts it from the descent directions. Assuming an at-

tacker controls a set of workers $\mathcal{B} \subset \mathcal{W}$ and each of these workers compute genuine descent directions m_t grouped into B_t, we have the following algorithm.

Algorithm 8 Little

Require: $B_t \subset \{m_t^{(1)}, ..., m_t^{(n)}\}$
1: **for each** $i \in [d]$ **do**
2: $\sigma_i = std(B_{t,i})$
3: **for each** $m_{t,i} \in B_{t,i}$ **do**
4: $m_{t,i} := m_{t,i} - \sigma_i$
5: **end for**
6: **end for**
7: **return** P_t

Each worker then sends the new descent directions to the server.

Empire Another intentional attack is Empire [11]. This attack is less sophisticated than Little as it simply multiplies each m_t it has access to by -0.9.

Algorithm 9 Empire

Require: $B_t \subset \{m_t^{(1)}, ..., m_t^{(n)}\}$
1: **for each** $k \in |P_t|$ **do**
2: $B_{t,k} := -0.9 \cdot B_{t,k}$
3: Send $B_{t,k}$ to the server
4: **end for**

Sign Flip Presented in [13], this algorithm is similar to Empire, each byzantine worker multiplies its moment by -1. The algorithm is the same as Empire, we only change -0.9 to -1 in Algorithm 18.

Label Flip Originating from the same article as Sign Flip [13], it simulated an attacker trying to implement a backdoor in our model by changing the labels, for example, changing 0 to 1 which might "confuse" the model. In the case of the MNIST data set, the worst case would be to change all of the labels i.e. for a data and label pair $(x, y)_i$ where $y \in \{0, ..., 9\}$, the algorithms uses $(x, y)_i := (x, 9 - y)_i$ such as to change all of the labels.

Infinity As its name suggests, this attack sends vectors made out of infinity instead of momentum. The byzantine participants either send negative or positive infinity.

2
Background

Before describing our frameworks, some background needs to be covered. The first section describes how traditional or centralized machine learning works. Then, section 2.2 describes Distributed Machine Learning before delving into RESAM [1] and the different parts needed to make the framework byzantine resilient.

2.1 Traditional Machine Learning

While there are many approaches to training a machine learning model (supervised, unsupervised, self-supervised, etc), supervised is arguably the easiest to work with, especially if we already have access to the data. Hence, this work focuses on this method. Therefore the following section introduces supervised learning and then explains how we can minimise the error using an optimisation algorithm.

2.1.1 Supervised Machine Learning

Machine Learning is essentially the estimation of an unknown function $M : X \to Y$. Many approaches exist, the easiest being supervised learning. This technique relies on a labelled set where each data point $x_i \in X$ matches an output $y_i \in Y$. The models' output is expressed as follows $M(x_i) = y_i + \mu_i$ where μ_i denotes random noise.

For example, in a task of cats and dogs image classification, the data point x would be a picture of a cat or a dog, and the corresponding label y is 0 if the image is a cat and 1 if it is a dog. Another example is the estimation of a company's stock price. A data point x could be historical prices and the performance of similar stocks, and the matched value y is the stock price at an arbitrary time.

The most popular machine learning approach to these problems is Neural Networks or Deep Learning. The simplest component of a neural network is the perceptron

$$output = \sigma(W^T x + b), \tag{2.1}$$

where σ is the activation function, W is the weight matrix and b the bias vector. Without going into too much detail, the activation function introduces non-linearity. Nowadays, many perceptrons are stacked in layers to form much larger models. W and b are the trainable or learnable parameters we will finetune such that the output matches y_i as well as possible. We group all the trainable parameters in θ to

facilitate the notation.

However, minimizing the error requires measuring it. Many approaches exist to compare the prediction $M_\theta(x_i)$ and the actual value y_i such as the 1- and 2-norm. We aggregate the measures of all the distances to compute the loss function $\mathcal{L}(\theta)$,

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{(x,y)\in X\times Y} d(M_\theta, (x), y), \tag{2.2}$$

where n is the number of data samples we have. Now that we have the loss function, we can finetune the parameters θ to reduce the model's error. In other words, we need to find θ that minimizes the loss. Many techniques exist and are further discussed in the following section.

2.1.2 Optimisation Algorithms

The finetuning of θ is usually an iterative process which stops once the loop reaches some type of stopping criterion, such as the maximum number of rounds. Due to the possibly large number of parameters, computing the Hessian at each iteration requires significant computational power, possibly slowing the training session down. Therefore, machine learning relies on first-order methods as they avoid computing the Hessian, thereby saving memory and computational power. Arguably, the most known first-order optimisation algorithm is Gradient Descent:

$$\theta_t = \theta_{t-1} - \alpha\nabla\mathcal{L}(\theta_{t-1})$$

where $\nabla\mathcal{L}(\theta_t)$ is the gradient over the entire data. However, the ever-growing data sets made it very expensive to compute the gradient on the entire set. A solution is Stochastic Gradient Descent; the idea is to split the training set into stratified chunks, called batches, and perform an update on each batch. This method reduces the memory required since we only use a chunk of the data, and the model prediction happens faster since it has fewer data points to predict. SGD being iterative, it performs an update on each chunk, also called a batch. Once the algorithm used each batch once, the algorithm performed an epoch. The algorithm goes back to the first batch and performs a new epoch. Denoting the loss computed on the batch corresponding to time step t by $\mathcal{L}_t(\theta_{t-1})$, we have

$$\theta_t = \theta_{t-1} - \alpha\nabla\mathcal{L}_t(\theta_{t-1}).$$

The growth of the Machine Learning field led to the development of various optimisation algorithms such as RMSProp [2] and Adam [3]. Both use moving averages called momentums. RMSProp solves the vanishing and exploding gradient problem introduced by complex functions such as Neural Networks, by dividing the gradients by the square root of the second momentum,

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla\mathcal{L}_t(\theta_{t-1})\right)^2, \tag{2.3}$$

The update formula for RMSProp is as follows

$$\theta_t = \theta_{t-1} - \frac{\alpha}{v_t}\nabla\mathcal{L}_t(\theta_{t-1}).$$

The division scales small gradients up and large ones down. Adam is an extension of RMSProp as it uses the first momentum,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla\mathcal{L}_t(\theta_{t-1}), \tag{2.4}$$

instead of the gradient in the numerator. The Adam algorithm is described in Algorithm 10.

Algorithm 10 Adam as described in [3]: g_t^2 stands for the element-wise square, ϵ is usually 10^{-10}

Require: Learning rate: α, Exponential decay rates: $\beta_1, \beta_2 \in [0, 1)$, Objective function $\mathcal{L}$, Initial model parameters: θ_0

$m_0 := 0$
$v_0 := 0$
$t := 0$
while θ_t not converged **do**
    $t := t + 1$
    $g_t := \nabla\mathcal{L}_t(\theta_{t-1})$
    $m_t := \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t := \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    $\hat{m}_t := m_t / (1 - \beta_1^t)$
    $\hat{v}_t := v_t / (1 - \beta_2^t)$
    $\theta_t := \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
end while
return θ_t

Computing $\hat{m}_t$ and $\hat{v}_t$ corrects the bias toward zero introduced by initializing m_0 and v_0 to zero.

2.2 Distributed Machine Learning

The increasing data set sizes and model complexity led to scalability bottlenecks due to the limited computational power and memory a single hardware device (CPU, GPU) has. Distributed Machine Learning solves this bottleneck by leveraging the computational power of various computational devices to compute an update vector. Thanks to this, Distributed Machine Learning became the main approach to training large models.

The ability to leverage the power from multiple devices inspired the fields to develop methods where each participant trains a model cutting the need for data gathering. The most known approach is Federated Learning [4]. Federated Learning works in rounds where each participant trains a model locally on a batch or epoch and sends the updated model to the server. The server aggregates the models, using FedAvg [5] for example, and sends the model back to the participants who perform another update. Another approach is to aggregate descent directions, instead of model weights, and have the server update the model. Figure 2.1 shows an iteration

of this approach. The server receives all the descent directions (step 1), aggregates them (step 2), updates the model (step 3) and sends the new version of the model to the workers (step 3).

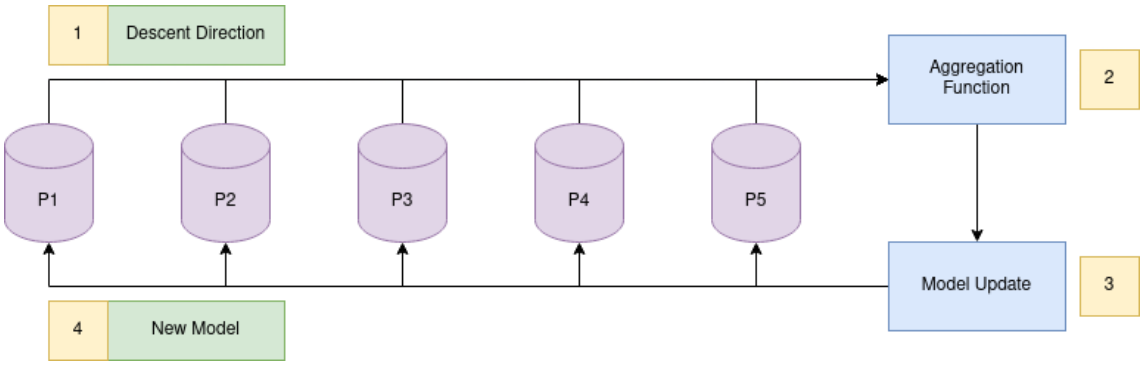

Figure 2.1: An iteration in a Distributed Machine Learning workflow. The purple blocks represent the participants, the blue blocks the work performed by the server, the yellow blocks the steps, and the green blocks show what is shared. In the first step, the workers send the locally computed descent direction. The server aggregates them in Step 2, updates the model in Step 3, and sends the new model to each worker in Step 4.

While it is tempting to use the average to aggregate the descent directions, this measure is unsafe as it is outlier sensitive. Instead, one would use a byzantine resilient aggregation method, i.e. able to counter byzantine descent directions. This is where the RESAM (RESilient Averaging of Momentums) [1] framework comes in.

2.2.1 RESAM

RESAM leverages the first moment and a Byzantine aggregation function to train a model even in the presence of Byzantine participants. The first reduces the variance in locally computed directions. Hence, independently of the impact of the byzantine nodes on a given update, the local gradients are guaranteed to approximately follow the previous trajectory. The latter aims at approximating the descent direction by aggregating the locally computed descent directions, even in the presence of byzantine participants. Such an aggregation function is denoted by $\mathcal{F}(\cdot)$.

Algorithm 11 RESAM

Require: Initial model parameters $\theta_1^{[d]}$, Set of workers $\mathcal{W}$, Number of steps T, Learning rates: $\{\alpha_1, ..., \alpha_t\}$, Exponential decay rates: $\beta \in [0, 1)$, Aggregation Function: $\mathcal{F}(\cdot)$, $\forall w \in \mathcal{W} : m_0^{(w)} = 1$

for each t = 1,...,T **do**

1. Send θ_t to each worker
2. Each worker computes $m_t^{(w)} = \beta m_{t-1}^{(w)} + (1-\beta) g_t^{(w)}$ and sends it to the server. $m_t^{(w)}$ might an arbitrary value in case w is byzantine.
3. The server computes θ_{t+1}: $\theta_{t+1} = \theta_t - \alpha_t \mathcal{F}(m_t^{(1)}, ..., m_t^{(n)})$

end for

return uniformly randomly chosen $\{\theta_0, ..., \theta_T\}$

2.2.2 Aggregation Methods

We distinguish two types of aggregation methods; byzantine resilient aggregation and pre-aggregation. As its name suggests, the first type aims at making the framework byzantine resilient. The second mitigates the impact of heterogeneous data.

2.2.2.1 Byzantine resilient aggregation

The simplest method one can think of is the unweighted average. However, it is sensitive to extreme values. Hence an attacker could send a very large or small m_t to impact the training phase. Therefore, we need more sophisticated methods. [1] introduces the notion of (f, ϵ)-Resilient averaging measuring the byzantine resilience of aggregation functions. It is defined as follows:

Definition 2.2.1. ***(f, ϵ)-Resilient averaging***
For a total of n workers of which f are byzantine such that $f < n$ and a real value $\lambda \leq 0$, an aggreation function $\mathcal{F}(\cdot)$ is called (f, ϵ)-Resilient averaging if for any collection of n vectors $x_1, ..., x_n$ and any set $S \subset \{1, ..., n\}$ of size $n - f$:

$$||\mathcal{F}(x_1, ..., x_n) - \bar{x}_S|| \leq \lambda \max_{i,j \in S} ||x_i - x_j|| \tag{2.5}$$

Where $\bar{x}_S := \frac{1}{|S|} \sum_{i \in S} x_i$ and $|S|$ is the cardinality of S

In our case, S denotes the set of honest workers. The value of λ is the resilience coefficient. Using this definition, the authors of [1] computed the byzantine resilience

of various aggregation methods enabling us to select the most byzantine-resilient aggregation functions.

Coordinate-Wise Trimmed mean (CWTM) With $\lambda = \frac{f}{n-f}\min(2\sqrt{n-f}, \sqrt{d}) \geq \frac{f}{n-f}$, Coordinate-Wise Trimmed mean [6] is the most resilient aggregation function investigated in [1].The algorithm ascendingly sorts the parameters coordinate-wise and averages the parameters in the range $[f, n-f]$. Let M_t denote the set of all the moments received from the workers at time t, and m^* denotes the output from the aggregation methods which the server will use to perform the update. For a given coordinate i in the set of coordinates $[d]$, the algorithm for CWTM is described as follows.

Algorithm 12 Coordinate-Wise Trimmed Mean at a coordinate $i \in [d]$

Require: M_t

1: $M_{new,i} = sort(M_{t,i})$

2: $m^*_{t,i} = \frac{1}{n-2f} \sum_{q \in [f, n-f]} m_{new,i,q}$

3: **return** $m^*_{t,i}$

While it still performs averaging, sorting beforehand and selecting a range in the middle ensures extreme values are filtered out. The main downside is that it does not scale well due to the need to sort all the vectors element-wise at each iteration.

Coordinate Wise Median (CWM) This methods comes from [6] and has a resilience coefficient of $\lambda = \frac{n}{2(n-f)}\min(2\sqrt{n-f}, \sqrt{d})$. As its name suggests, the server selects the median at each coordinate. The advantage is that the median is less susceptible to outliers and, since we assume $f < \frac{n}{2}$, otherwise it is impossible for the algorithm to converge, we are very likely to select the descent direction/moment from an honest worker.

Algorithm 13 Coordinate-Wise Median at a coordinate $i \in [d]$

Require: M_t

1: $m^*_{t,i} = \text{median}(M_{t,i})$

2: **return** $m^*_{t,i}$

Minimum Diameter Averaging (MDA) With a resilience score of $\frac{2f}{n-f}$, Minimum Diameter Averaging [7] only differs from the optimal λ by a constant factor making it an ideal candidate for our framework. The rule averages a subset $S_t \subset M_t$ containing $n-f$ vectors such that S_t has the smallest diameter. Using the same notation as in [1]:

Algorithm 14 Minimum Diameter Averaging

Require: M_t

1: $S = \underset{\substack{S\subset\{1,\dots,n\}\\|S|=n-f}}{\arg\min} \left\{\max_{i,j\in S} ||m_t^{(i)} - m_t^{(j)}||\right\}$

2: $m^* = \frac{1}{n-2f} \sum_{k\in S} m_t^{(k)}$

3: **return** m_t^*

Mean Around Median (MEAMED) The last aggregation rule we consider is Mean Around Median [8] which has a resilience score of $\frac{2f}{n-f}\min(2\sqrt{n-f}, \sqrt{d})$. Its name is also very descriptive, as the server averages the $n-f$ points closest to the median.

Algorithm 15 Mean around Median at a coordinate $i \in [d]$

Require: M_t

1: $med_{t,i} = median(P_{t,i})$

2: $C = \underset{\substack{m_{t,i}\in M_{t,i}\\|C|=n-f}}{\arg\min} |med_{t,i} - m_{t,i}|$

3: $m_{t,i}^* = \frac{1}{n-f} \sum_{m_{t_i}\in C} m_{t,i}$

4: **return** $m_{t,i}^*$

The intuition is similar to CWTM. The median is assumed to come from a non-byzantine participant. It should be relatively close to the values of the other non-byzantine workers. Taking their average provides a better approximation of the real gradient.

2.2.2.2 Pre-aggregation

Due to the lack of control over the participants and their data, it is safe to assume that the data distribution differs between the workers. A solution is bucketing [9]; the descent directions are randomly placed into an arbitrary number of buckets before being averaged, and the result is passed to a byzantine resilient aggregation function. However, a recent method called Nearest Neighbour Mixing (NNM) [10] proved to outperform bucketing, making it a good candidate pre-aggregation method for our framework. For each $m_t^{(w)}$, we find the $n-f$ closest vectors in M_t. Then we compute the average between them.

Algorithm 16 NNM

Require: $\{m_t^{(1)}, ..., m_t^{(n)}\}$
1: Initialize: M_{new}
2: **for each** $w \in \mathcal{W}$ **do**
3: $\quad Y = \underset{\substack{|Y|=n-f \\ x \in \{m_t^{(1)}, ..., m_t^{(n)}\}}}{\arg\min} (|m_t^{(w)} - x|)$
4: $\quad M_{new,w} = \frac{1}{n-f} \sum_{y \in Y} m_t^{(y)}$
5: **end for**
6: **return** M_{new}

Now that we described all methods to counter Byzantine participants, we have to discuss and detail what they are and how they can attack our training loop.

2.2.3 Byzantine Behaviors

Several state-of-the-art attacks can be launched against distributed learning systems. For example, Empire [11] and Little [12] are two such attacks one can use to disrupt the training process.

Little Recall that the core assumption for CWTM is that byzantine gradients are either very large or small. Hence by sorting them and discarding the smaller and larger elements, the aggregation function should filter the byzantine descent directions out. However, if one has access to multiple workers computing legitimate descent directions, we could use these to estimate the ones computed by the honest worker. Thereby enabling the attacker to increase or decrease the byzantine directions as much as possible while still remaining in the range CWTM uses. Figure 2.2 shows five (two byzantine (red) and three honest (green)) sorted descent directions. The big circle represents the directions selected by CWTM. CWTM averages two green dots close to each other and a red dot placed right at the limit, leading to the red dot having a significant impact on the average.

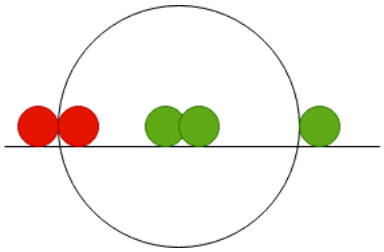

Figure 2.2: Example of how little impacts CWTM. The two red dots are byzantine while the green are honest. The large circle shows the descent direction that will be taken into account.

To estimate the diameter of the large circle, Little computes the coordinate-wise standard deviation and subtracts it from the descent directions. Assuming an attacker controls a set of workers $\mathcal{B} \subset \mathcal{W}$ and each of these workers computes genuine descent directions m_t grouped into B_t, we have the following algorithm.

Algorithm 17 Little

Require: $B_t \subset \{m_t^{(1)}, ..., m_t^{(n)}\}$
1: **for each** $i \in [d]$ **do**
2: $\quad \sigma_i = std(B_{t,i})$
3: $\quad$ **for each** $m_{t,i} \in B_{t,i}$ **do**
4: $\quad\quad m_{t,i} := m_{t,i} - \sigma_i$
5: $\quad$ **end for**
6: **end for**
7: **return** P_t

Each worker then sends the new descent directions to the server.

Empire Another intentional attack is Empire [11]. This attack is less sofisticated than Little as it simply multiplies each m_t it has access to by -0.9.

Algorithm 18 Empire

Require: $B_t \subset \{m_t^{(1)}, ..., m_t^{(n)}\}$
1: **for each** $k \in |P_t|$ **do**
2: $\quad B_{t,k} := -0.9 \cdot B_{t,k}$
3: $\quad$ Send $B_{t,k}$ to the server
4: **end for**

Sign Flip Presented in [13], this algorithm is similar to Empire, each byzantine worker multiplies its moment by -1. The algorithm is the same as Empire, we only change -0.9 to -1 in Algorithm 18.

Label Flip Originating from the same article as Sign Flip [13], it simulated an attacker trying to implement a backdoor in our model by changing the labels, for

example, changing 0 to 1 which might "confuse" the model. In the case of the MNIST data set, the worst case would be to change all of the labels i.e. for a data and label pair $(x, y)_i$ where $y \in \{0, ..., 9\}$, the algorithms uses $(x, y)_i := (x, 9 - y)_i$ such as to change all of the labels.

Infinity As its name suggests, this attack sends vectors made out of infinity instead of momentum. The byzantine participants either send negative or positive infinity.

3

Adam-based Distributed Optimization Algorithms

3.1 DRA and MRA

This work studies two new algorithms; *D*escent Direction Resilient Adam (*D*RA) and *M*oment Resilient Adam (*M*RA). Similarly to RESAM [1], a server receives momentums or descent directions from the workers, aggregates them using a Byzantine resilient aggregation method and applies the update before sending the updated model to the workers. The main difference is that, in the case of MRA, the server receives the first and second momentum, while for DRA, the worker sends descent directions, as computed in Adam. Introducing

$$q_t^{(h)} = \frac{m_t^{(h)}}{\sqrt{\epsilon + v_t^{(h)}}}, \tag{3.1}$$

i.e. the descent direction computed by an honest worker $h \in \mathcal{H} \subset \mathcal{W}$. Recall the set $\mathcal{B} \subset \mathcal{W}$ containing all byzantine workers, we describe DRA as follows:

Algorithm 19 DRA

Require: Initial model parameters $\theta_1^{[d]}$; Set of workers $\mathcal{W}$; Number of steps T, $\alpha = 0.001$; Exponential decay rates: $\beta_1 = 0.99$ and $\beta_2 = 0.999$; Aggregation Function: $\mathcal{F}(\cdot)$; $\forall w \in \mathcal{W} : m_0^{(w)} = 0$

for each t = 1,...,T **do**

1. Send θ_t to each worker
2. Each honest worker h computes $q_t^{(h)}$ as described in Algorithm 10 and sends it to the server. If the worker is byzantine, the descent direction, $q_t^{(w)}$ for $w \in \mathcal{B}$, might be arbitrary depending on the attack.
3. The server computes θ_{t+1}: $\theta_{t+1} = \theta_t - \alpha_t \mathcal{F}(q_t^{(1)}, ..., q_t^{(n)})$

end for

return θ_T

In the case of MRA, the server has to aggregate both momentums sparely i.e. it performs $\mathcal{F}(m_t^{(1)}, ..., m_t^{(n)})$ and $\mathcal{F}(v_t^{(1)}, ..., v_t^{(n)})$, doubling the work needed. Note that while we use the same aggregation methods for the first and second moment, one could technically use two different ones. However, due to the finetuning work it

would require, we decided to use the same aggregation function for both. The following algorithm describes MRA.

Algorithm 20 MRA

Require: Initial model parameters $\theta_1^{[d]}$, Set of workers $\mathcal{W}$, Number of steps T, $\alpha = 0.001$, Exponential decay rates: $\beta_1 = 0.99$ and $\beta_2 = 0.999$, Aggregation Function: $\mathcal{F}(\cdot)$, $\forall w \in \mathcal{W} : m_0^{(w)} = 0$

for each t = 1,...,T **do**

1. Send θ_t to each worker
2. Each honest worker h computes $\hat{m}_t^{(h)}$ and $\hat{v}_t^{(h)}$ as described in Algorithm 10 and sends it to the server. If $w \in \mathcal{B}$, the momentum $\hat{m}_t^{(w)}$ and $\hat{v}_t^{(w)}$ might be arbitrary depending on the attack.
3. The server computes the update as follows

$$
\begin{aligned}
m_t^* &= \mathcal{F}(\hat{m}_t^{(1)}, ..., \hat{m}_t^{(n)}) \\
v_t^* &= \mathcal{F}(\hat{v}_t^{(1)}, ..., \hat{v}_t^{(n)}) \\
\theta_t &:= \theta_{t-1} - \alpha \cdot \frac{m_t^*}{\sqrt{v_t^* + \epsilon}}
\end{aligned}
$$

end for
return θ_T

However, it is not sufficient to describe algorithms. Indeed one also has to prove they converge. As we will see in Chapter 4, DRA outperforms MRA in our experiments. Hence we started with the convergence proof for DRA. In order to prove the convergence of DRA, we assume the data is homogeneously distributed. This assumption enables us to adapt the proof presented in [14]. The key step is extending [14, Lemma A.2], which proves that the descent direction computed on a single batch is valid, to use the aggregation function $\mathcal{F}$. Once this step done, we could adapt [14, Proof A.6] which is a bound on the expectation of the 2-norm of the gradient of the loss function at any location θ_τ, $\mathbb{E}\left[||\nabla \mathcal{L}(\theta_\tau)||^2\right]$, where τ denotes an arbitrary index in $[T]$ such that

$$
\forall t \in \mathbb{N}, \quad t < T, \quad \mathbb{P}(\tau = t) \propto 1 - \beta_1^{T-t}. \tag{3.2}
$$

Unfortunately, this extension introduced new terms to bound. Finding such bounds turned out to be more complex than expected, resulting in a convergence proof containing two unbounded terms for which we, at the moment, have no informative upper bound.

3.2 DRA Convergence on Homogeneous Data

First, recall that our set of workers $\mathcal{W}$ contains a total of n workers, of which f are byzantine and $n-f$ are honest. The set of honest nodes is denoted by $\mathcal{H}$. Since we use MRA, we use $\mathcal{F}_t$ to denote the output from $\mathcal{F}(q_t^{(1)}, ..., q_t^{(n)})$. $\mathcal{L}$ denotes the loss function and θ_t^d the d dimensional model parameters at time $t \in [T]$. $G_t = \nabla\mathcal{L}(\theta_{t-1})$ denotes the gradient of the loss function at time t. Furthermore, the local data is randomly split into batches, hence, $g_t^{(h)} = \nabla\mathcal{L}_t^{(h)}(\theta_{t-1})$ denotes the stochastic gradient computed by worker h. Note that we do not necessarily perform one epoch, batches can be used multiple times so t refers to the batch used at time t. Furthermore, for any participant $w \in \mathcal{W}$, let

$$q_t^{(w)} = \frac{m_t^{(w)}}{\sqrt{\epsilon + v_t^{(w)}}} \quad \text{and} \quad Q_t^{(w)} = \frac{g_t^{(w)}}{\sqrt{\epsilon + v_t^{(w)}}}$$

denote the descent direction with and without the first momentum in the numerator, respectively. [14] introduces

$$\tilde{v}_{t,k} = \beta_2^k v_{t-k} + \mathbb{E}\left[\sum_{j=n-k+1}^{t} \beta_2^{t-j} g_j^2\right].$$

We will also need the mean of the descent directions computed by the honest workers. This is computed as follows

$$\bar{q}_t = \frac{1}{n-f} \sum_{h \in \mathcal{H}} q_t^{(h)}.$$

Lastly, we rewrite the steps performed locally such as to facilitate the proof. We first update the computation of both momentums:

$$m_t^{(w)} = \beta_1 m_{t-1}^{(w)} + (1-\beta_1) g_t^{(w)} = (1-\beta_1) \sum_{k=1}^{t} \beta_1^{t-k} g_k^{(w)}$$

$$v_t^{(w)} = \beta_2 v_{t-1}^{(w)} + (1-\beta_2)(g_t^{(w)})^2 = (1-\beta_2) \sum_{k=1}^{t} \beta_2^{t-k} (g_k^{(w)})^2.$$

We move $(1-\beta_1)$ and $(1-\beta_2)$ to the learning rate. Furthermore, both bias correction terms can also be moved to the learning rate. As shown in [14], dropping the corrective term for m_t has no observable effect on the training process and is equivalent to a learning-rate warm-up. Furthermore, dropping this term makes α_n monotonically increasing, simplifying the proof. Also, as shown in [14], the corrective term for m_t converges much faster to its limit value.

With these changes, the exact algorithm for a worker w is

$$m_t^{(w)} = \beta_1 m_{t-1}^{(w)} + g_t^{(w)} \tag{3.3}$$

$$v_t^{(w)} = \beta_2 v_{t-1}^{(w)} + (g_t^{(w)})^2 \tag{3.4}$$

$$q_t^{(w)} = \frac{m_t^{(i)}}{\sqrt{\epsilon + v_t^{(i)}}}, \tag{3.5}$$

and the only difference at the server level is the learning rate:

$$\alpha_t = \alpha \cdot (1-\beta_1)\sqrt{\frac{1-\beta_2^t}{1-\beta_2}}. \tag{3.6}$$

In order to prove convergence, we need a few assumptions. The first one concerns the bounds of the locally computed gradients and comes from [1]. Since the data is assumed to be homogeneously distributed across the participants, we assume these gradients to be unbiased with bounded variance. Which is mathematically expressed as follows

Assumption 1: $\mathcal{L}$ is bounded below by $\mathcal{L}_* \geq -\infty$ i.e.

$$\forall \theta \in \mathbb{R}^d \quad \mathcal{L}(\theta) \geq \mathcal{L}_*. \tag{3.7}$$

Assumption 2: l_∞ of the stochastic gradients computed by any honest worker is almost surely bounded

$$\forall \theta \in \mathbb{R}^d \quad ||\nabla \mathcal{L}^{(h)}(\theta)||_\infty \leq R - \sqrt{\epsilon} \quad R \geq \epsilon. \tag{3.8}$$

Assumption 3: The gradient of $\mathcal{L}$ is L-Lipschitz-continuous w.r.t 2-norm

$$\forall \theta_i, \theta_j \in \mathbb{R}^d \quad ||\nabla \mathcal{L}(\theta_i) - \nabla \mathcal{L}(\theta_j)||_2 \leq L||\theta_i - \theta_j||. \tag{3.9}$$

Theorem 3.2.1. *(**DRA convergence on Homogeneous data**) Given an homogeneously distributed data set, the hyper-parameters $0 \leq \beta_1 < \beta_2 < 1$, α as in (3.6) with $\alpha > 0$, τ defined by (3.2) and assumptions 3.7, 3.8, 3.9, $T > \frac{\beta}{1-\beta_1}$, T is the last iteration, we have*

$$\begin{aligned}\mathbb{E}[||\nabla \mathcal{L}(\theta_\tau)||^2] \leq 2R\frac{\mathcal{L}(\theta_0) - \mathcal{L}_*}{\alpha \tilde{T}} + \frac{2R\alpha_T}{\alpha\tilde{T}} \sum_{t=1}^{T} \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] \\ + \frac{2R\alpha_T^2 L}{\alpha\tilde{T}} \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{E}{\tilde{T}}\left(\ln\left(1 + \frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2)\right),\end{aligned}$$

with

$$E = \frac{2\alpha RLd(1-\beta_1)}{(1-\beta_1/\beta_2)(1-\beta_2)} + \frac{2\alpha^2 L^2 \beta_1 d}{(1-\beta_1)^{3/2}(1-\beta_1/\beta_2)} + \frac{12R^2 d\sqrt{1-\beta_1}}{\sqrt{1-\beta_2}(1-\beta_1/\beta_2)^{3/2}},$$

and $\tilde{T} = T - \frac{\beta_1}{1-\beta_1}$

3.2.1 Lemmas

Since the proof is a distributed version of the one presented in [14], we make extensive use of their lemmas and properties.

Lemma 3.2.2 (**Adaptive update with momentum approximately follows a descent direction** [14, Lemma A.1])**.** *Given $\theta_0 \in \mathbb{R}^d$, the iterate defined by the simplified algorithm, Equation* (3.3) *to Equation* (3.5) *with learning rate* (3.6)*, for $(\alpha_t)_{t\in T}$ and under assumption 1, 2 and 3 with $0 \leq \beta_1 < \beta_2 \leq 1$, for $t \in T$ and an honest worker $h \in \mathcal{H}$ we have:*

$$\mathbb{E}\left[\sum_{i\in[d]} G_{t,i} q_{t,i}^{(h)}\right] \geq \frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=1}^{t-1} \beta_1^k \mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon + \tilde{v}_{t,k+1,i}^{(h)}}}\right]\right)$$
$$- \frac{\alpha_n^2 L^2}{4R}\sqrt{1-\beta_1}\left(\sum_{l=1}^{t-1}||q_{n-l}^{(h)}||^2 \sum_{k=l}^{t-1}\beta_1^k\sqrt{k}\right) - \frac{3R}{\sqrt{1-\beta_1}}\left(\sum_{k=0}^{t-1}\left(\frac{\beta_1}{\beta_2}\right)^k \sqrt{k+1}||Q_{t-k}^{(h)}||^2\right) \tag{3.10}$$

Since the expectation of the inner product of the gradient G_t and the descent direction $q_{t,i}^{(h)}$ is lower bounded by a positive term, $q_{t,i}^{(h)}$ is an approximation of G_t. Hence, the descent directions computed by honest workers approximately follow the gradient G_t. However, due to the distributed nature of our optimisation algorithm, we do not have a single direction but rather an aggregation of descent directions provided by honest and byzantine participants. Hence, we need to ensure $\mathcal{F}_t$ approximately follows a descent direction. Since $\mathcal{F}_t$ can be an arbitrary aggregation function, it is easier to rely on the upper bound of the difference between the aggregation function and $\bar{q}_t$. This is exactly what definition 2.2.1 computes. Introducing ξ_t as the difference between $\mathcal{F}_t$ and $\bar{q}_t$, i.e. $\xi_t = \mathcal{F}_t - \bar{q}_t$, we have $\mathcal{F}_t = \xi_t + \bar{q}_t$. For resilience, by definition 2.2.1, we are interested in a bound of the form

$$||\xi_t|| = ||\mathcal{F}_t - \bar{q}_t|| \leq \lambda \max_{i,j\in S} ||q_t^{(i)} - q_t^{(j)}||.$$

ξ is the difference between the aggregation function and the mean of the directions computed by the honest workers. Upper bounding ξ removes the need to compute a convergence proof for each aggregation function as we use a difference instead of the output of the aggregation function. Using this definition to bound the deviation

between G and $\mathcal{F}$ yields

$$\begin{aligned}\mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\mathcal{F}_{t,i}\right] &= \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}(\xi_t + \bar{q}_t)\right] \\ &= \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] + \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\frac{1}{n-f}\sum_{h\in\mathcal{H}} q_{t,i}^{(h)}\right] \\ &= \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] + \mathbb{E}\left[\sum_{i\in[d]} \frac{1}{n-f}\sum_{h\in\mathcal{H}} G_{t,i}q_{t,i}^{(h)}\right] \\ &= \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] + \frac{1}{n-f}\sum_{h\in\mathcal{H}}\mathbb{E}\left[\sum_{i\in[d]} G_{t,i}q_{t,i}^{(h)}\right]\end{aligned} \tag{3.11}$$

By Lemma 3.2.2, we have a bound for the second term in the addition. We now need a bound on the first term. However, computing ξ_t became much more challenging than expected. Unfortunately, within the scope of this master's thesis, we have not been able to compute the ξ meaning that we cannot find such a bound. An explanation will be provided in section 3.4.2. Instead, we will assume that such a bound exists, and continue with the rest of the proof. Hence we obtain the following lemma.

Lemma 3.2.3 (**Aggregation of the updates with momentum approximately follows a descent direction**). *Given $\theta_0 \in \mathbb{R}^d$, the iterate defined by the simplified algorithm, Equation* (3.3) *to Equation* (3.5) *with learning rate* (3.6)*, for $(\alpha_t)_{t\in T}$ and under assumption 1, 2 and 3 with $0 \leq \beta_1 < \beta_2 \leq 1$, for $t \in T$ and an honest worker $h \in \mathcal{H}$ we have:*

$$\begin{aligned}\mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\mathcal{F}_{t,i}^{(h)}\right] \geq\ & \mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] + \frac{1}{n-f}\sum_{h\in\mathcal{H}}\frac{1}{2}\left(\sum_{i\in[d]}\sum_{k=1}^{t-1}\beta_1^k\mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon + \tilde{v}_{t,k+1,i}^{(h)}}}\right]\right) \\ & - \frac{\alpha_n^2 L^2}{4R}\sqrt{1-\beta_1}\frac{1}{n-f}\sum_{h\in\mathcal{H}}\left(\sum_{l=1}^{t-1}||q_{n-l}^{(h)}||^2\sum_{k=l}^{t-1}\beta_1^k\sqrt{k}\right) \\ & - \frac{3R}{\sqrt{1-\beta_1}}\frac{1}{n-f}\sum_{h\in\mathcal{H}}\left(\sum_{k=0}^{t-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}||Q_{t-k}^{(h)}||^2\right)\end{aligned} \tag{3.12}$$

The following three lemmas come from [14] and do not need any adaptation.

Lemma 3.2.4 (**Sum of ratios of the square of a decayed sum and a decayed sum of square** [14, Lemma A.2]). *Assuming $0 \leq \beta_1 < \beta_2 \leq 1$, and given a sequence of real numbers $(a_t)_{t\in T}$. For an honest worker $h \in \mathcal{H}$, let $b_t^{(h)} = \sum_{l=1}^t \beta_2^{t-l}(a_l^{(h)})^2$ and $c_t^{(h)} = \sum_{l=1}^t \beta_2^{t-l}a_l^{(h)}$, then*

$$\sum_{l=1}^t \frac{\left(c_l^{(h)}\right)^2}{\epsilon + b_l} \leq \frac{1}{(1-\beta_1)(1-\beta_1/\beta_2)}\left(\ln\left(1 + \frac{b_t^{(h)}}{\epsilon}\right) - t\ln(\beta_2)\right)$$

Lemma 3.2.5 (**Sum of geometric term times a square root** [14, Lemma A.3])**.** *Given* $0 < a < 1$ *and* Q *a real integer, we have*

$$\sum_{q=0}^{Q-1} a^q \sqrt{q+1} \leq \frac{2}{(1-a)^{3/2}}$$

Lemma 3.2.6 (**Sum of a geometric term times roughly a power 3/2** [14, Lemma A.4])**.** *Let* $0 < a < 1$ *and* L *a real integer, we have*

$$\sum_{l=1}^{L} a^l \sqrt{l}(l+1) \leq \frac{4a}{(1-a)^{5/2}} \tag{3.13}$$

3.2.2 Proof of Theorem 3.2.1

This section is devoted to the proof of theorem 3.2.1.

Proof of Theorem 3.2.1. Given an iteration $t \in \mathbb{N}^*$ and using the smoothness of $\mathcal{L}$ defined in Assumption 3, the descent lemma states:

$$\mathcal{L}(\theta_t) \leq \mathcal{L}(\theta_{t-1}) + \langle \nabla \mathcal{L}(\theta_{t-1}), \theta_t - \theta_{t-1} \rangle + \frac{L}{2} ||\theta_t - \theta_{t-1}||^2,$$

for the traditional centralized problem, i.e. one worker and one descent direction q_t. Noting that $\theta_t - \theta_{t-1} = -\alpha_n q_t$, we have

$$\mathcal{L}(\theta_t) \leq \mathcal{L}(\theta_{t-1}) - \alpha_n \langle \nabla \mathcal{L}(\theta_{t-1}), q_t \rangle + \frac{\alpha_n^2 L}{2} ||q_t||^2. \tag{3.14}$$

However, we are in a distributed environment where the descent direction is computed using an aggregation method $\mathcal{F}$. Introducing this function into (3.14) gives,

$$\mathcal{L}(\theta_t) \leq \mathcal{L}(\theta_{t-1}) - \alpha_n G_t^T \mathcal{F}_t + \frac{\alpha_t^2 L}{2} ||\mathcal{F}_t||^2. \tag{3.15}$$

Taking the expectation on both sides of this inequality gives

$$\mathbb{E}[\mathcal{L}(\theta_t)] \leq \mathbb{E}[\mathcal{L}(\theta_{t-1})] - \alpha_t \mathbb{E}[G_t^T \mathcal{F}_t] + \frac{\alpha_t^2 L}{2} \mathbb{E}\left[||\mathcal{F}_t||^2\right],$$

and applying Lemma 3.3.2 to upper bound $\mathbb{E}[G_t^T \mathcal{F}_t]$, we obtain

$$\begin{aligned}
\mathbb{E}[\mathcal{L}(\theta_t)] \leq \mathbb{E}[\mathcal{L}(\theta_{t-1})] &+ \frac{\alpha_t^2 L}{2} \mathbb{E}\left[||\mathcal{F}_t||^2\right] + \alpha_t \mathbb{E}\left[\sum_{i \in [d]} G_{t,i} \xi_{t,i}\right] \\
&- \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{\alpha_t}{2} \left(\sum_{i \in [d]} \sum_{k=1}^{t-1} \beta_1^k \mathbb{E}\left[\frac{G_{t-k,i}^2}{\sqrt{\epsilon + \tilde{v}_{t,k+1,i}^{(h)}}} \right] \right) \\
&+ \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{\alpha_t^3 L^2}{4R} \sqrt{1-\beta_1} \left(\sum_{l=1}^{t-1} \mathbb{E}\left[||q_{n-l}^{(h)}||^2\right] \sum_{k=l}^{t-1} \beta_1^k \sqrt{k} \right) \\
&+ \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{3\alpha_t R}{\sqrt{1-\beta_1}} \left(\sum_{k=0}^{t-1} \left(\frac{\beta_1}{\beta_2} \right)^k \sqrt{k+1} \mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right] \right).
\end{aligned}$$

Because of the bound on the ∞-norm of the stochastic gradients, for any $t \in T$ and coordinate $i \in [d]$, $\sqrt{\epsilon + \tilde{v}_{t,k+1,i}} \leq R\sqrt{\sum_{l=0}^{t-1} \beta_2^l}$. Introducing $\Omega_t = \sqrt{\sum_{l=0}^{t-1} \beta_2^l}$,

$$\begin{aligned}
\mathbb{E}[\mathcal{L}(\theta_t)] \leq \mathbb{E}[\mathcal{L}(\theta_{t-1})] &+ \frac{\alpha_t^2 L}{2} \mathbb{E}\left[||\mathcal{F}_t||^2\right] + \alpha_t \mathbb{E}\left[\sum_{i \in [d]} G_t \xi_{t,i}\right] \\
&- \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{\alpha_t}{2R\Omega_t} \left(\sum_{i \in [d]} \sum_{k=1}^{t-1} \beta_1^k \mathbb{E}\left[||G_{t-k,i}||^2\right] \right) \\
&+ \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{\alpha_t^3 L^2}{4R} \sqrt{1-\beta_1} \left(\sum_{l=1}^{t-1} \mathbb{E}\left[||q_{n-l}^{(h)}||^2\right] \sum_{k=l}^{t-1} \beta_1^k \sqrt{k} \right) \\
&+ \frac{1}{n-f} \sum_{h \in \mathcal{H}} \frac{3\alpha_t R}{\sqrt{1-\beta_1}} \left(\sum_{k=0}^{t-1} \left(\frac{\beta_1}{\beta_2} \right)^k \sqrt{k+1} \mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right] \right).
\end{aligned}$$

Now, we introduce the summation over all $t \in [T]$. Note that α_t is non-decreasing, hence using α_T instead of α_t maintains the upper bound. Furthermore, using Assumption 3 gives $\mathcal{L}(\theta_t) \geq \mathcal{L}_*$. Applying these observations gives

$$\begin{aligned}
&\frac{1}{n-f}\sum_{h\in\mathcal{H}}\underbrace{\frac{1}{2R}\sum_{t=1}^{T}\frac{\alpha_t}{\Omega_t}\sum_{k=0}^{t-1}\beta_1^k\mathbb{E}\left[||G_{t-k}||^2\right]}_{A} \leq \mathcal{L}(\theta_0) - \mathcal{L}_* + \alpha_T\sum_{t=1}^{T}\mathbb{E}\left[\sum_{i\in[d]}G_{t,i}\xi_{t,i}\right] \\
&+ \underbrace{\frac{\alpha_T^2 L}{2}\sum_{t=1}^{T}\mathbb{E}\left[||\mathcal{F}_t||^2\right]}_{B} \\
&+ \frac{1}{n-f}\sum_{h\in\mathcal{H}}\underbrace{\frac{\alpha_T^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{t=1}^{T}\left(\sum_{l=1}^{t-1}E\left[||q_{t-l}^{(h)}||^2\right]\sum_{k=l}^{t-1}\beta_1^k\sqrt{k}\right)}_{C} \\
&+ \frac{1}{n-f}\sum_{h\in\mathcal{H}}\underbrace{\frac{3\alpha_T R}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\left(\sum_{k=0}^{t-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right]\right)}_{D}. \qquad (3.16)
\end{aligned}$$

We need a lower bound on A and an upper bound on B, C, D. We will start by simplifying A. Simplifying the notation of A with $l = t - k$ we get

$$\begin{aligned}
A &= \frac{1}{2R}\sum_{t=1}^{T}\frac{\alpha_t}{\Omega_t}\sum_{l=1}^{t}\beta_1^{t-l}\mathbb{E}[||G_l||^2] &(3.17)\\
&= \frac{\alpha(1-\beta_1)}{2R}\sum_{t=1}^{T}\mathbb{E}[||G_t||^2]\sum_{l=t}^{T}\beta_1^{l-t} &(3.18)\\
&= \frac{\alpha}{2R}\sum_{t=1}^{T}(1-\beta_1^{T-t+1})\mathbb{E}[||G_t||^2] &(3.19)\\
&= \frac{\alpha}{2R}\sum_{t=1}^{T}(1-\beta_1^{T-t+1})\mathbb{E}[||\nabla\mathcal{L}(\theta_{t-1})||^2] &(3.20)\\
&= \frac{\alpha}{2R}\sum_{t=0}^{T-1}(1-\beta_1^{T-t})\mathbb{E}[||\nabla\mathcal{L}(\theta_t)||^2] &(3.21)
\end{aligned}$$

More specifically, we get (3.18) from (3.17) by using the definition $\alpha_t = (1-\beta_1)\Omega_n\alpha$, and we get (3.21) from (3.20) by shifting the summation indices and starting the summation from $t = 0$.

Noting that

$$\sum_{t=0}^{T-1}(1-\beta_1^{T-t}) = T - \beta_1\frac{1-\beta_1^T}{1-\beta_1},$$

and taking the distribution of τ, $P(\tau = t) \propto 1 - \beta_1^{T-t}$, we have

$$1-\beta_1^{T-t} = 1 - \beta_1^{T-t}\frac{T-\beta_1\frac{1-\beta_1^T}{1-\beta_1}}{T-\beta_1\frac{1-\beta_1^T}{1-\beta_1}} = \left(T-\beta_1\frac{1-\beta_1^T}{1-\beta_1}\right)P(\tau=t).$$

Since $T - \beta_1 \frac{1-\beta_1^T}{1-\beta_1} \geq T - \frac{\beta_1}{1-\beta_1} = \tilde{T}$, we have

$$A \geq \frac{\alpha \tilde{T}}{2R} \mathbb{E}[||\nabla \mathcal{L}(\theta_\tau)||^2].$$

Moving to B, we first expand $\mathbb{E}\left[||\mathcal{F}||^2\right]$

$$\begin{aligned} \mathbb{E}\left[||\mathcal{F}||^2\right] &= \mathbb{E}\left[||\xi_t + \bar{q}_t||^2\right] \\ &\leq 2\mathbb{E}\left[||\xi_t||\right] + 2\mathbb{E}\left[||\bar{q}_t||^2\right], \end{aligned} \tag{3.22}$$

since $||\bar{q}_t||^2 = ||\frac{1}{n-f} \sum_{h \in \mathcal{H}} q_t^{(h)}||^2 \leq \frac{1}{n-f} \sum_{h \in \mathcal{H}} ||q_t^{(h)}||^2$, we can rewrite (3.22) as

$$\begin{aligned} \mathbb{E}[||\mathcal{F}_t||^2] &\leq 2\mathbb{E}[||\xi_t||^2] + 2\mathbb{E}\left[\frac{1}{n-f} \sum_{h \in \mathcal{H}} ||q_t^{(h)}||^2\right] \\ &= 2\mathbb{E}[||\xi_t||^2] + \frac{2}{n-f} \sum_{h \in \mathcal{H}} \mathbb{E}\left[||q_t^{(h)}||^2\right]. \end{aligned} \tag{3.23}$$

Applying Lemma 3.2.4 to the right-hand side of the sum yields

$$\begin{aligned} B &\leq \alpha_T^2 L \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{2}{n-f} \sum_{h \in \mathcal{H}} \frac{\alpha_T^2 L}{2(1-\beta_1)(1-\beta_1/\beta_2)} \sum_{i \in [d]} \left(\ln\left(1 + \frac{v_{T,i}^{(h)}}{\epsilon}\right) - T \ln(\beta_2) \right) \\ &= \alpha_T^2 L \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{\alpha_T^2 L}{(n-f)(1-\beta_1)(1-\beta_1/\beta_2)} \sum_{h \in \mathcal{H}} \sum_{i \in [d]} \left(\ln\left(1 + \frac{v_{T,i}^{(h)}}{\epsilon}\right) - T \ln(\beta_2) \right). \end{aligned} \tag{3.24}$$

Noting that $v_T^{(h)} = \sum_{t=1}^{T} \beta_2^{T-t} \left(g_t^{(h)}\right)^2 \leq \sum_{t=1}^{T} \beta_2^{T-t} R^2$, due to assumption 2 and taking R^2 out of the sum we are left with $\sum_{t=1}^{T} \beta_2^{T-t} = \frac{1}{1-\beta_2}$. Hence $v_T^{(h)} \leq \frac{R^2}{1-\beta_2}$, yielding

$$\sum_{i \in [d]} \left(\ln\left(1 + \frac{v_{T,i}^{(h)}}{\epsilon}\right) - T \ln(\beta_2) \right) \leq d \left(\ln\left(1 + \frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T \ln(\beta_2) \right). \tag{3.25}$$

Applying this to (3.24), we get

$$\begin{aligned} B &= \alpha_T^2 L \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{\alpha_T^2 L}{(n-f)(1-\beta_1)(1-\beta_1/\beta_2)} \sum_{h \in \mathcal{H}} \sum_{i \in [d]} \left(\ln\left(1 + \frac{v_{T,i}^{(h)}}{\epsilon}\right) - T \ln(\beta_2) \right) \\ &\leq \alpha_T^2 L \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{\alpha_T^2 L}{(n-f)(1-\beta_1)(1-\beta_1/\beta_2)} \sum_{h \in \mathcal{H}} d \left(\ln\left(1 + \frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T \ln(\beta_2) \right) \end{aligned} \tag{3.26}$$

$$= \alpha_T^2 L \sum_{t=1}^{T} \mathbb{E}[||\xi_t||^2] + \frac{\alpha_T^2 L d}{(1-\beta_1)(1-\beta_1/\beta_2)} \left(\ln\left(1 + \frac{R^2}{\epsilon(1-\beta_2)}\right) - T \ln(\beta_2) \right). \tag{3.27}$$

Due to v_T^h being upper bounded by a constant in (3.26) making the terms involved in the summation constant, we can remove the mean over $\mathcal{H}$ to get (3.27).

Moving to C and factorizing the sum gives,

$$C = \frac{\alpha_T^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{t=1}^{T}\sum_{l=1}^{t}\mathbb{E}\left[||q_l^{(h)}||^2\right]\sum_{k=t-l}^{t-1}\beta_1^k\sqrt{k} \tag{3.28}$$

$$= \frac{\alpha_T^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{t=1}^{T}\mathbb{E}\left[||q_t^{(h)}||^2\right]\sum_{l=t}^{T}\sum_{k=l-t}^{l-1}\beta_1^k\sqrt{k} \tag{3.29}$$

$$= \frac{\alpha_T^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{t=1}^{T}\mathbb{E}\left[||q_t^{(h)}||^2\right]\sum_{k=0}^{T-1}\beta_1^k\sqrt{k}\sum_{l=t}^{t+k}1. \tag{3.30}$$

These steps are basically factorizing the sums. (3.29) factorizes the expectations since (3.28) generates multiple terms with the same expectations. The factorization gives (3.31), and we can now apply Lemma 3.2.5 to get

$$C = \frac{\alpha_T^3 L^2}{4R}\sqrt{1-\beta_1}\sum_{t=1}^{T}\mathbb{E}\left[||q_t^{(h)}||^2\right]\sum_{k=0}^{T-1}\beta_1^k\sqrt{k}(k+1) \tag{3.31}$$

$$\leq \frac{\alpha_T^3 L^2}{R}\sum_{t=1}^{T}\mathbb{E}\left[||q_t^{(h)}||^2\right]\frac{\beta_1}{(1-\beta_1)^2}. \tag{3.32}$$

We can now apply Lemma 3.2.4,

$$C \leq \frac{\alpha_T^3 L^2 \beta_1}{R(1-\beta_1)^3(1-\beta_1/\beta_2)}\sum_{i\in[d]}\left(\ln\left(1+\frac{v_{T,i}^{(h)}}{\epsilon}\right)-t\ln(\beta_2)\right)$$

and use the identity (3.25) as used in B, to obtain a constant

$$C \leq \frac{\alpha_T^3 L^2 d\beta_1}{R(1-\beta_1)^3(1-\beta_1/\beta_2)}\left(\ln\left(1+\frac{R^2}{\epsilon(1-\beta_2)}\right)-t\ln(\beta_2)\right) \tag{3.33}$$

Moving to D, we start by simplifying the equation

$$D \leq \frac{3\alpha_T R}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\left(\sum_{k=0}^{t-1}\left(\frac{\beta_1}{\beta_2}\right)^k\sqrt{k+1}\mathbb{E}\left[||Q_t^{(h)}||^2\right]\right) \tag{3.34}$$

$$= \frac{3\alpha_T R}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right]\sum_{k=t}^{T}\left(\frac{\beta_1}{\beta_2}\right)^{t-k}\sqrt{1+t-k} \tag{3.35}$$

$$\leq \frac{6\alpha_T R}{\sqrt{1-\beta_1}}\sum_{t=1}^{T}\mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right]\frac{1}{(1-\beta_1/\beta_2)^{3/2}}. \tag{3.36}$$

At (3.34) we take $\mathbb{E}\left[||Q_{t-k}^{(h)}||^2\right]$ out as it does not depend on k. Furthermore, we operate an index change enabling us to use lemma 3.2.5 to get (3.36).

Then, applying Lemma 3.2.4 with $\beta_1 = 0$ on $\mathbb{E}\left[||Q_t^{(h)}||^2\right]$

$$D \leq \frac{6\alpha_T R}{\sqrt{1-\beta_1}(1-\beta_1/\beta_2)^{3/2}} \sum_{i\in[d]} \left(\ln\left(1+\frac{v_{T,i}^{(h)}}{\epsilon}\right) - t\ln(\beta_2)\right) \tag{3.37}$$

Here again we use 3.25 to get,

$$D \leq \frac{6\alpha_T Rd}{\sqrt{1-\beta_1}(1-\beta_1/\beta_2)^{3/2}} \left(\ln\left(1+\frac{R^2}{\epsilon(1-\beta_2)}\right) - t\ln(\beta_2)\right). \tag{3.38}$$

Plugging (3.2.2), (3.27), (3.33) and (3.38) into (3.16), noting that we can remove the average over honest worker for B, C and D, since they are constants and using Definition (3.6) for α_t, gives

$$\begin{aligned}
\frac{\alpha\tilde{T}}{2R}\mathbb{E}[||\nabla\mathcal{L}(\theta_\tau)||^2] \leq \mathcal{L}(\theta_0) - \mathcal{L}_* &+ \alpha_T \sum_{t=1}^{T}\mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] + \alpha_T^2 L\sum_{t=1}^{T}\mathbb{E}[||\xi_t||^2] \\
&+ \frac{\alpha_T^2 L}{(1-\beta_1)(1-\beta_1/\beta_2)}\sum_{i\in[d]}\left(\ln\left(1+\frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2)\right) \\
&+ \frac{\alpha_T^3 L^2\beta_1 d}{R(1-\beta_1)^3(1-\beta_1/\beta_2)}\left(\ln\left(1+\frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2)\right) \\
&+ \frac{6\alpha_T Rd}{\sqrt{1-\beta_1}(1-\beta_1/\beta_2)^{3/2}}\left(\ln\left(1+\frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2)\right).
\end{aligned}$$

Rearranging terms, we get the following theorem

$$\begin{aligned}
\mathbb{E}[||\nabla\mathcal{L}(\theta_\tau)||^2] \leq 2R\frac{\mathcal{L}(\theta_0)-\mathcal{L}_*}{\alpha\tilde{T}} &+ \frac{2R\alpha_T}{\alpha\tilde{T}}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{i\in[d]} G_{t,i}\xi_{t,i}\right] \\
&+ \frac{2R\alpha_T^2 L}{\alpha\tilde{T}}\sum_{t=1}^{T}\mathbb{E}[||\xi_t||^2] + \frac{E}{\tilde{T}}\left(\ln\left(1+\frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2)\right),
\end{aligned}$$

with

$$E = \frac{2\alpha RLd(1-\beta_1)}{(1-\beta_1/\beta_2)(1-\beta_2)} + \frac{2\alpha^2 L^2\beta_1 d}{(1-\beta_1)^{3/2}(1-\beta_1/\beta_2)} + \frac{12R^2 d\sqrt{1-\beta_1}}{\sqrt{1-\beta_2}(1-\beta_1/\beta_2)^{3/2}}.$$

This completes the proof. □

The bound differs from the centralized proof in two points. The first is the first part of the sum E, i.e. the coefficient corresponding to B. Indeed, in the distributed proof, this value is twice larger than the corresponding coefficient in the original proof. The second is the addition of two terms relying on ξ_t. These represent the error or the difference between $\bar{q}_t$ and $\mathcal{F}$. Since bounding ξ_t turned out to not be trivial, the following section details our approach up to where we got stuck.

3.2.3 Open Problems

The real challenge of the proof turned out to be the upper bound on ξ_t. The following subsection presents how we tried to bound it and where we got stuck. However, the steps we followed seemed natural and might be a hint for someone trying to bound these terms.

3.2.3.1 Upper bound on ξ_t

From definition 2.5 we have

$$||\xi_t||^2 = ||\mathcal{F}_t - \bar{q}_t||^2 \leq \lambda^2 \max_{i,j \in \mathcal{H}} ||q_t^{(i)} - q_t^{(j)}||^2. \tag{3.39}$$

[1, Lemma 2] shows that

$$\max_{i,j \in \mathcal{H}} ||q_t^{(i)} - q_t^{(j)}||^2 \leq 4 \sum_{i \in \mathcal{H}} ||q_t^{(i)} - \bar{q}_t||^2.$$

Knowing that

$$\begin{aligned} \frac{2}{n-f} \sum_{i \in \mathcal{H}} ||q_t^{(i)} - \bar{q}_t||^2 &= \frac{1}{(n-f)^2} \sum_{i,j \in \mathcal{H}} ||q_t^{(i)} - q_t^{(j)}||^2 \\ \Rightarrow \sum_{i \in \mathcal{H}} ||q_t^{(i)} - \bar{q}_t||^2 &= \frac{1}{2(n-f)} \sum_{i,j \in \mathcal{H}} ||q_t^{(i)} - q_t^{(j)}||^2, \end{aligned}$$

we can rewrite the upper bound as follows

$$||\mathcal{E}_t||^2 \leq \frac{2\lambda^2}{(n-f)} \sum_{i,j \in \mathcal{H}} ||q_t^{(i)} - q_t^{(j)}||^2. \tag{3.40}$$

We now need to bound $||q_t^{(i)} - q_t^{(j)}||^2$ i.e. the difference between two descent directions computed by honest workers.

3.2.3.2 Bounding the difference in descent direction between two workers

Starting with the two norms of the difference between the update vectors provided by honest worker i and j at an arbitrary step $t \in [T]$,

$$||q_t^{(i)} - q_t^{(j)}||^2 = ||q_t^{(i)}||^2 + ||q_t^{(j)}||^2 - 2\langle q_t^{(i)}, q_t^{(j)} \rangle \tag{3.41}$$

Introducing the sum over $[T]$, we get

$$\begin{aligned} ||q_t^{(i)} - q_t^{(j)}||^2 &\leq \sum_{t=1}^{T} ||q_t^{(i)} - q_t^{(j)}||^2 \\ &= \sum_{k=1}^{T} \left(||q_t^{(i)}||^2 + ||q_t^{(j)}||^2 - 2\langle q_t^{(i)}, q_t^{(j)} \rangle \right) \\ &= \underbrace{\sum_{t=1}^{T} ||q_t^{(i)}||^2}_{A} + \underbrace{\sum_{t=1}^{t} ||q_t^{(j)}||^2}_{B} - 2 \underbrace{\sum_{t=1}^{T} \langle q_t^{(i)}, q_t^{(j)} \rangle}_{C}. \end{aligned}$$

We can now apply Lemma 3.2.6 to upper bound A,

$$A \leq \sum_{i\in[d]} \frac{1}{(1-\beta_1)(1-\beta_1/\beta_2)} \left(\ln\left(1 + \frac{v_{T,i}^{(i)}}{\epsilon}\right) - T\ln(\beta_2) \right), \tag{3.42}$$

and B,

$$B \leq \sum_{i\in[d]} \frac{1}{(1-\beta_1)(1-\beta_1/\beta_2)} \left(\ln\left(1 + \frac{v_{T,i}^{(j)}}{\epsilon}\right) - T\ln\beta_2 \right). \tag{3.43}$$

Furthermore, using the same trick as in (3.25), we get

$$A + B \leq 2d \left(\ln\left(1 + \frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2) \right). \tag{3.44}$$

Hence (3.41) becomes

$$\sum_t^T ||q_t^{(i)} - q_t^{(j)}||^2 \leq 2d \left(\ln\left(1 + \frac{R_T^2}{\epsilon(1-\beta_2)}\right) - T\ln(\beta_2) \right) - 2\sum_{t=1}^T \langle q_t^{(i)}, q_t^{(j)} \rangle. \tag{3.45}$$

Now, the issue arise in bounding $\langle q_t^{(i)}, q_t^{(j)} \rangle$. One would search for a lower bound on $\sum_{t=1}^T \langle q_t^{(i)}, q_t^{(j)} \rangle$, however, we could not find such a bound on the inner product or the descent direction. The complexity relies on finding a lower bound that does not violate the current inequality. The challenge comes from the numerator of q_t possibly being negative making the computation of the lower bound non-trivial.

4

Numerical Experiments and Results

While the convergence proof only concerns DRA and homogeneously distributed data, we will also investigate the potential use of DRA and MRA on heterogeneously distributed data. Furthermore, these frameworks will only be evaluated on image-recognizing tasks as they are simple to work with while requiring relatively simple models. Hence, SGD, RESAM, DRA and MRA are evaluated on MNIST and CIFAR-10, both homogeneously and heterogeneously distributed among the workers. The experiments, results and discussion are grouped in sections corresponding to the data sets.

4.1 MNIST

We will start by discussing the data and models used during the experiments. Then we will present and discuss the results.

4.1.1 Setup

Data The data sets follow the same structure as in [1] since these are the setups where their algorithm performed the best. In the case of the homogeneous data set, MNIST is split into 15 chunks, each containing 3610 data points, i.e. 361 copies of each digit.

Forming heterogeneous data sets requires more work. As done in [15], we used the Dirichlet distribution to generate the concentration of each label in each subset. The Probability Density Function is as follows:

$$f(x_1, ..., x_n; \alpha_1, ..., \alpha_n) = \frac{1}{B(\alpha)} \prod_{i=1}^{n} x_i^{\alpha_i - 1}.$$

where B is the multivariate beta function and is a normalizing constant, and $\alpha = (\alpha_1, ..., \alpha_n)$ contains the concentration parameters. The advantage of this probability distribution is that, given concentration parameters and a uniform prior, it assigns a probability to each such that the sum is 1, i.e. given a $1 \times n$ concentration vector α, the distribution outputs a $1 \times n$ vector whose sum is 1. As the parameters reach ∞, the assigned probabilities become $1/n$. On the other hand, a concentration parameter close to 0 yields one entry becoming one and the others zero. In our

case, α contains concentration parameters of 0.1. Furthermore, we generate the concentration for each class, i.e. the distribution receives a 1×17 vector containing 0.1. Computing the concentrations for each class enables us to maximise the use of the data while creating heterogeneous local sets. (4.1) shows the local set size and the concentration of the data points. As we can see, there is a big difference in label concentration and data set size.

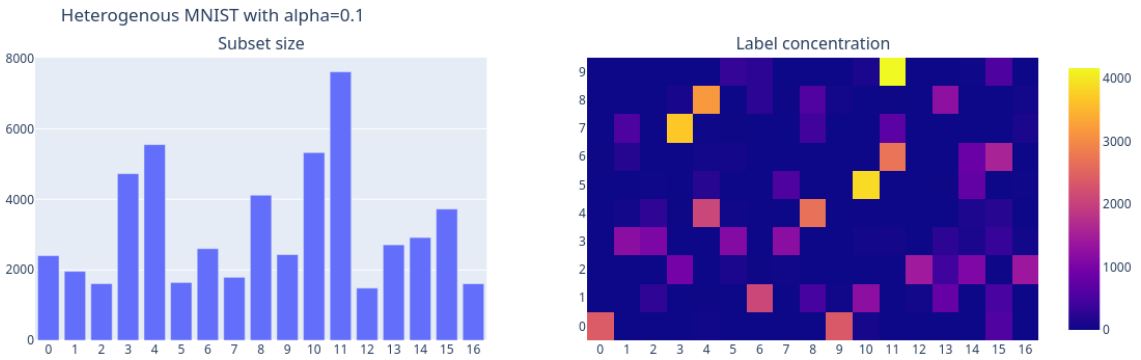

Figure 4.1: The plot on the left shows the size of the 15 heterogeneous MNIST data sets obtained by using $\alpha = 0.1$. The image on the left shows the class concentration in each subset. We see that subsets 3 and 10 are among the most unbalanced as they have mostly 7-labelled and 5-labelled images, respectively.

Model The task being image classification, we use the convolutional neural network as described in [1]. C denotes a 2-D convolutional layer with 20 channels, a kernel size of 5, 0 paddings, and stride 1. MP denotes a 2D max pool layer with a kernel size of 2. $L(\#outputs)$ represents a fully-connected linear layer with the number of nodes between the parentheses, and S stands for log-softmax. The architecture is as follows:

$$C \text{ - ReLU - } MP \text{ - } C \text{ - ReLU - } MP \text{ - } L(500) \text{ - ReLU - } L(500) \text{ - } S$$

The following figure is a visualization of the architecture:

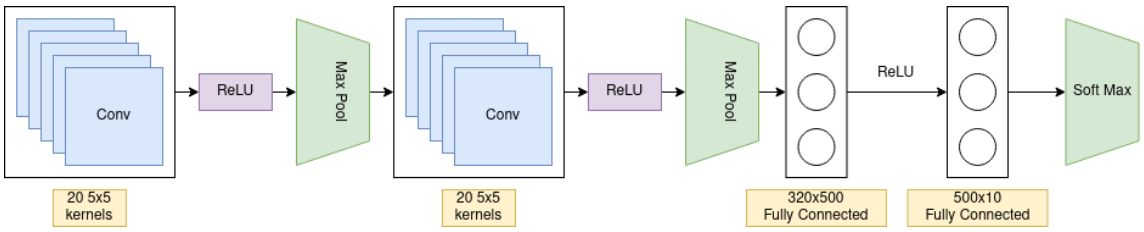

Figure 4.2: Figure of the architecture for the MNIST model. The blue blocks represent the convolutional layers, in purple the activation function, the greens are max-pool or softmax, and the white parts correspond to the fully connected layers.

Hyper parameters For homogeneously distributed MNIST, every model uses the same parameters as [1]; a clipping parameter of 2, l_2-regularization with a factor of 10^{-4}, a batch size of 25 and Cross Entropy as loss function. SGD and RESAM

use a learning rate of 0.75 on Homogeneously distributed data and $\frac{0.75}{1+\lfloor \frac{t}{50} \rfloor}$ for the heterogeneous case. $\beta_1 = 0.99$ for RESAM. Adam uses the same parameters on both distributions; learning rate= 0.001, $\beta_1 = 0.99$ and $\beta_2 = 0.999$.

4.1.2 Results and Discussion

Now that the experimental setup is detailed, we move to the results. First, we present them for the homogeneously distributed data sets and then for the heterogeneously distributed data sets. Then we will discuss them. The number of honest and byzantine workers depends on the data distribution. In the case of homogeneously distributed data, we have ten honest workers and five malicious ones, there are 13 honest workers and four byzantine ones when the data is heterogeneously distributed.

4.1.2.1 Homogeneously Distributed Data

As we can see in Figure 4.3, all methods converge when the pool of participants is byzantine-free. This is expected, it is the same as taking multiple batches per update on a traditional machine-learning training loop. While RESAM with CWTM performs the best (accuracy of 0.968), it is very closely followed by SGD with AVG (0.9675). Intuitively, one would assume that AVG would perform the best as it takes the entire set of information into account. However, while the gradients might be similar due to the data distribution, the local moments might differ from each other in the long run leading to CWTM being better for RESAM. Our two optimisation algorithms perm the worst, DRA reached an accuracy of 0.951 with CWTM and MRA is the last with an accuracy of 0.9485 with CWM.

Figure 4.3: The columns show the test accuracy computed after each training round on homogeneously distributed data without byzantine participants for each optimisation algorithm (SGD, RESAM, DRA, MRA). Each line in the subplot corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. All four optimization algorithms converge independently from the aggregation function. SGD converges the fastest and CWTM seem to converge much faster when using RESAM, DRA and MRA compared to the other aggregation functions.

Figure 4.4 presents the accuracy of the test set at each iteration. First, SGD is the most severely hit by Little. CWTM is the only aggregation method able to mitigate the attack but stays at a test accuracy of 0.4, the other aggregation methods spike then go to 0. The issue comes from the use of gradients. Indeed, the local update direction depends on the model and data, if the model starts to get flawed, the workers have no way to correct the update direction. On the other hand, the use of moments reduces the variance and ensures the updated directions remain valid. Thereby reducing the impact of a model updated with descent directions provided by byzantine workers. This explains the good performance of the moment-based optimization algorithms.

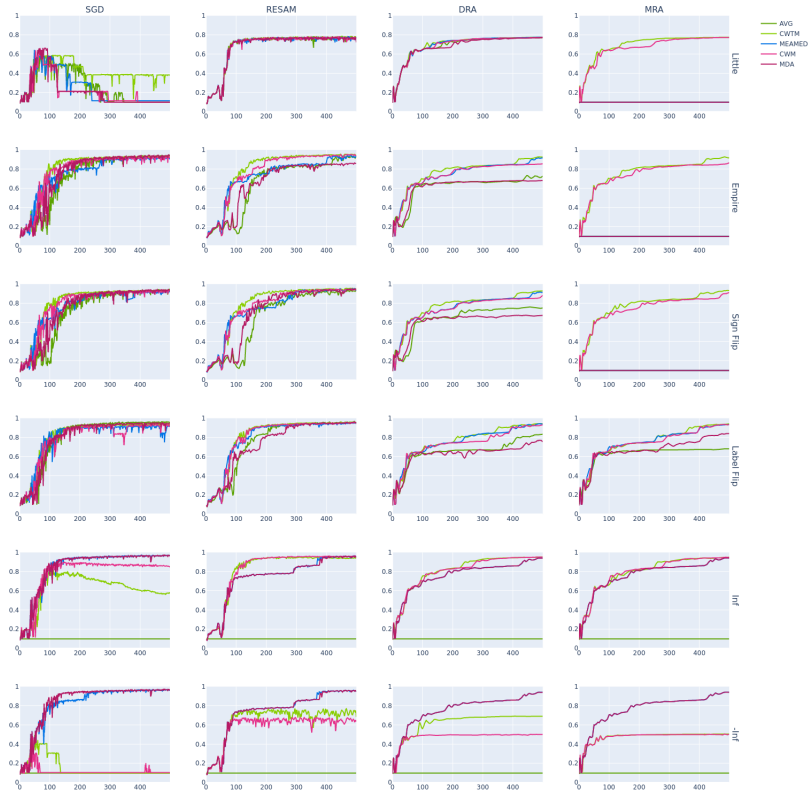

Figure 4.4: The columns show the test accuracy computed after each training round on homogeneously distributed data. The columns represent the optimisation algorithm (SGD, RESAM, DRA, MRA) while the rows correspond to the attacks; Little, Empire, Sign Flip, Label Flip, Infinity and -Infinity. Each line in the subplots corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. There are two notable behaviours, Little prevents convergence for SGD, and CWTM and CWM are the only effective aggregation methods for MRA against attacks on the momentums.

However, we see that MDA, MEAMED and AVG do not mitigate gradient-based attacks (Little, Empire, Label Flip) for MRA. As we will see this is a recurring theme and gets worse when the data is heterogeneously distributed. One of the reasons is quite simple, the second moments are supposed to be non-negative but an attack might make some negative. Applying the average might create negative elements

in the moments which yields a NaN when computing the square root. Hence, the model output's NaN's, and cannot be trained anymore. For MDA, we might have a negative value very close to 0 which leads to a sign change, the same logic applies to MEAMED where a negative value might be close enough to the median leading to it being incorporated into the mean. Hence in the three cases, we have at least one NaN in the first update effectively stopping the ability to improve the model. This is actually an unused advantage of the optimizer since we can easily detect some byzantine workers and eject them from the pool of workers. Nevertheless, we decided to not use it since MRA already requires the aggregation of 2 moments limiting the scalability of the method. Adding the necessity to go through all the moments and delete the ones containing negative elements adds twice the work.

To conclude this section, we will have a look at Table 4.1 which presents the accuracy of the last update. As we can see, RESAM outperforms all the other optimization algorithms against engineered attacks although it is closely followed by DRA (Difference of 0.0015) and MRA (Difference of 0.0025) on Little. For Empire, the gap is a bit larger, SGD with average is the second best. SGD outperforms the other optimizers on Sign Flip using the average, and both infinity attacks using MDA. As we can see, CWTM is the dominating aggregation function for RESAM, DRA and MRA.

	Little	Empire	Sign Flip	Label Flip	Inf	-Inf
SGD	0.3825 CWTM	0.945 AVG	0.948 AVG	0.9625 AVG	0.9725 MDA	0.9725 MDA
RESAM	0.777 CWTM	0.948 CWTM	0.951 CWTM	0.958 MDA	0.958 CWM	0.956 MEAMED
DRA	0.7755 CWTM	0.9265 CWTM	0.9295 CWTM	0.944 CWTM	0.9515 CWTM	0.9405 MEAMED
MRA	0.7745 CWTM	0.918 CWTM	0.9325 CWTM	0.942 CWTM	0.9495 CWTM	0.941 MEAMED

Table 4.1: Table showing the highest test accuracy and aggregation function after the last update for each optimisation algorithm (row) and attack (column). The values highlighted in green correspond to the best performer while the red corresponds to the worst. RESAM and SGD dominate DRA and MRA on Empire, Sign Flip and Label Flip but DRA and MRA are very close to the best-performing optimisation algorithm, RESAM, against Little.

4.1.2.2 Heterogeneously Distributed Data

A very important element in this section is the pre-aggregation method. Recall from Section 2.2.2.2 that we aggregate each momentum and descent direction with the Y closets momentum and direction, respectively, computed by other workers, before applying the byzantine aggregation. As in the previous section, Figure 4.5 shows the test accuracy of each optimisation algorithm with each aggregation method in a byzantine-free training loop. SGD performs the best, with an accuracy of 0.93 and CWTM. RESAM reaches an accuracy of 0.917 with CWTM as an aggregation function. Then comes DRA with MDA and an accuracy of 0.89, and MRA is the worst performer with 0.853 with CWTM. Here again, CWTM is the most effective aggregation method of the three algorithms.

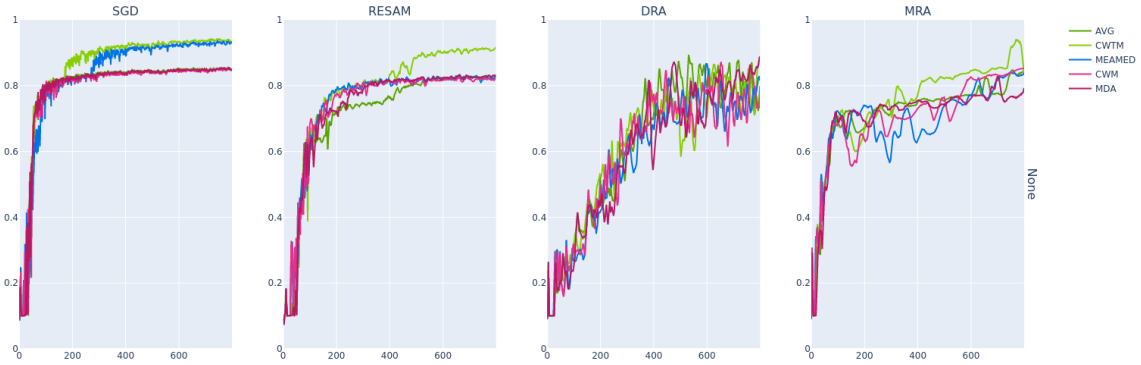

Figure 4.5: The columns show the test accuracy computed after each training round on heterogeneously distributed data without byzantine participants for each optimisation algorithm (SGD, RESAM, DRA, MRA). Each line in the subplot corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. SGD and RESAM have a nice stable curve while DRA and MRA spikes a lot. MRA reaches a top of 0.94 but drops to 0.853

From Figure 4.6 we see that MRA suffers the most as it simply does not converge when the attacks directly influence the momentums. The explanation is rather simple because Little, Empire and Sign Flip introduce negative second momentums, the pre-aggregation creates a domino effect by introducing negative elements in momentums originating from honest workers making it impossible for CWM and CWTM to filter them out as they were doing it in the homogeneous case. Hence, we end up with NaN's in each update vector independently from the byzantine resilient aggregation function. On the other hand, DRA performs well against little, reaching 0.804 test accuracy with CWTM. MEAMED works well for MRA while MDA, CW and AVG reach a peak before decreasing. However, MRA has a very slow convergence against Empire and Sign Flip. Due to nnm and the infinity being very far away, the vectors are very similar leading to similar updates independently from the aggregation method. Furthermore, due to the pre-aggregation, the two infinity-based attacks have the same effect on the training loop.

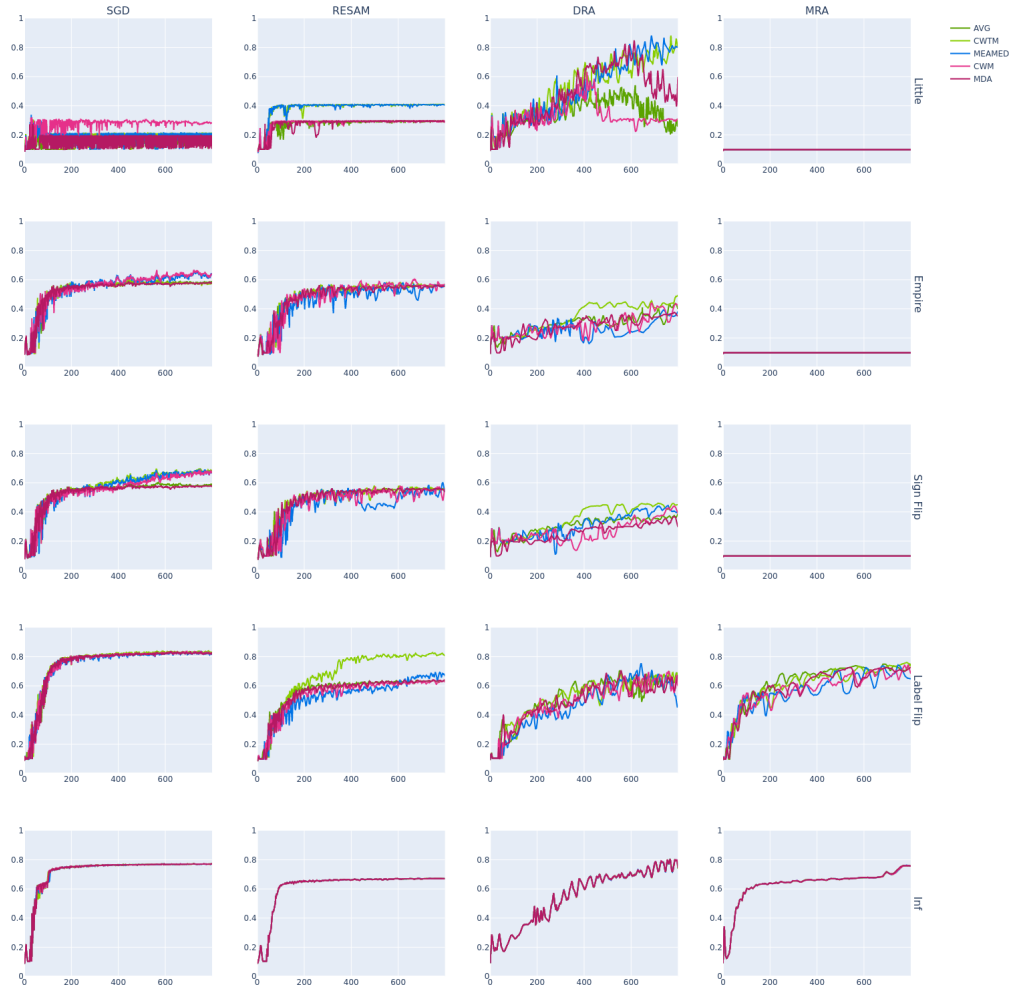

Figure 4.6: The columns show the test accuracy computed after each training round on heterogeneously distributed data. The columns represent the optimisation algorithm (SGD, RESAM, DRA, MRA) while the rows correspond to the attacks; Little, Empire, Sign Flip, Label Flip, Infinity and -Infinity. Each line in the subplots corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. MRA does not converge when attacked by Little, Empire and Sign Flip. DRA performs well against Little.

As we can see in the following table, except against Little, SGD is the best performer. This is most likely due to the data distribution. Indeed as explained earlier, momentums reduce the variance in descent direction. However, the local data sets being unbalanced require "flexible" gradients that will best minimize the current batch. Take the extreme case where the first batch contains only images of ones, but the next batch does not contain any sample of 1, then the gradients are tied around the gradient from the first batch and need some time to "overcome" this limit. In the case of SGD, each gradient points toward the best direction to minimize the error on the current batch independently from the previous ones. Where moments were an advantage on homogeneous data, it turns out to be an issue for heterogeneously distributed data. Here again, the table is dominated by CWM and CWTM.

	Little	Empire	Sign Flip	Label Flip	Inf
SGD	0.282 CWM	0.639 CWM	0.6885 CWTM	0.824 CWTM	0.772 CWTM
RESAM	0.4095 CWTM	0.5685 CWM	0.5625 CWTM	0.806 CWTM	0.671 AVG
DRA	0.8035 CWTM	0.484 CWTM	0.4505 CWTM	0.6765 MDA	0.759 CWTM
MRA	0.098 AVG	0.098 AVG	0.098 AVG	0.747 AVG	0.758 MDA

Table 4.2: Table showing the highest test accuracy and aggregation function after the last update for each optimisation algorithm (row) and attack (column). The values highlighted in green correspond to the best performer while the red corresponds to the worst.

4.2 CIFAR-10

RESAM and SGD outperformed MRA and DRA in most situations. Since MNIST is a simple data set requiring a trivial model, we decided to test our algorithms on CIFAR-10 since this data set requires a larger and more complex model. It remains an image recognition task relying on convolutional neural networks but introduces new layers, such as batch norm and dropout.

4.2.1 Setup

Data For the homogeneous data set, CIFAR-10 is homogeneously divided among 15 workers with each containing 330 data points from each class i.e. each worker has 3300 samples in total. Each subset is divided into batches of size 25.

Here as well, the Dirichlet distribution is used to form the heterogeneously distributed data sets. Due to CIFAR-10 containing more data samples than MNIST, the difference in subset size is more extreme than for MNIST, as we can observe in Figure (4.7).

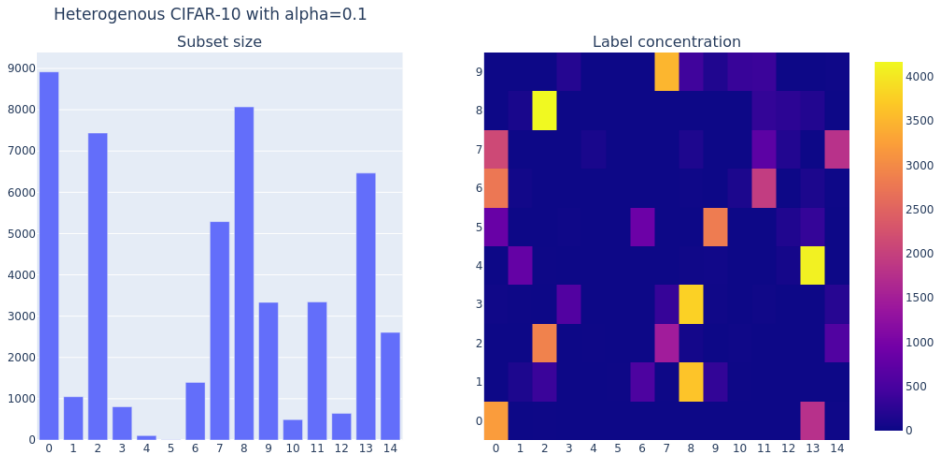

Figure 4.7: This plot presents the subset sizes and subsets class distributions on CIFAR-10 with $\alpha = 0.1$. Similarly to the previous plots, the left plot shows the subset sizes and the right one presents the subsets class distribution. CIFAR-10 being a much larger data set, we observe a larger difference in subset size.

Model Here again we perform image classification but CIFAR-10 requires a larger and more complicated model. Indeed we will use batch-normalization represented by B and dropout with fixed probability 0.25 denoted by D. Lastly, $C(\#channels)$ denotes a 2D convolutional layer with kernel size 3, 1 padding and 1 stride. With this notation, the architecture of the network can be described as,

(3,32×32)-C(64)-R-B-C(64)-R-B-M-D-C(128)-R-B-C(128)-R-B-M-D-L(128)-R-D-L(10)-S,

Figure (4.8) is a visualization of the architecture.

Hyper parameters For homogeneously distributed data, CIFAR-10 uses the same parameters as [1]; a clipping parameter of 2, l_2-regularization with a factor of 10^{-4}, a batch size of 50 and Cross Entropy as loss function. Adam uses the same parameters on both data sets, learning rate= 0.001, $\beta_1 = 0.99$ and $\beta_2 = 0.999$. RESAM uses $\beta_1 = 0.99$ and has the same learning rate as SGD, 0.25. For Heterogeneous data, the only difference is the learning rate for SGD which we set to $\frac{0.75}{1+\lfloor \frac{t}{50} \rfloor}$. In the case of RESAM, [10] runs performs 2000 iterations, but due to computational constraints, we will use 1500 and uses a learning rate of 0.25.

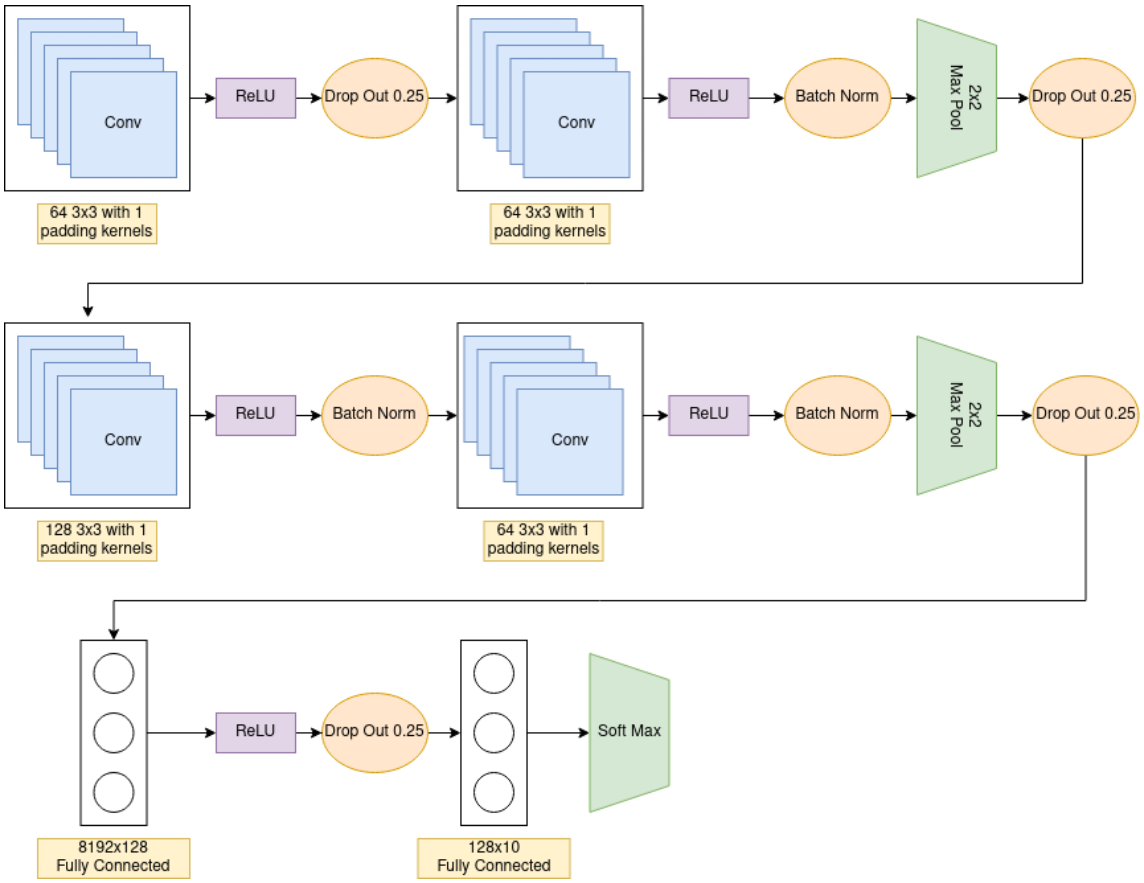

Figure 4.8: Similarly to Figure (4.2), the blue blocks represent the convolutional layers, in purple the activation function, the greens are max-pool or softmax and the white parts correspond to the fully connected layers. Besides more layers, we also have Drop Out and Batch Norm components in orange.

4.2.2 Results and Discussion

In both homogeneously and homogeneously distributed data sets, the models train for 1500 rounds. In the case of homogeneous data, we used ten honest workers and five malicious ones. For heterogeneous data, we have 11 honest workers and four malicious ones to keep the same ratio as the best-performing situation in [10].

4.2.2.1 Homogeneously Distributed Data

Interestingly, when we have no attacks, AVG is the best-performing aggregation method for SGD, RESAM and DRA leading to 0.703, 0.747, 0.74 test accuracy, respectively. MRA performs the best with a test accuracy of 0.758 and CWTM as an aggregation method. Figure A.1 in the appendix presents the plot of test accuracies at each iteration. Moving to the attacks, we immediately note in figure 4.9 that SGD and RESAM really struggle against Little, with a test accuracy of 0.1725 and 0.538, respectively. Then comes MRA with 0.574 and DRA is the best with 0.6125, both use CWTM.

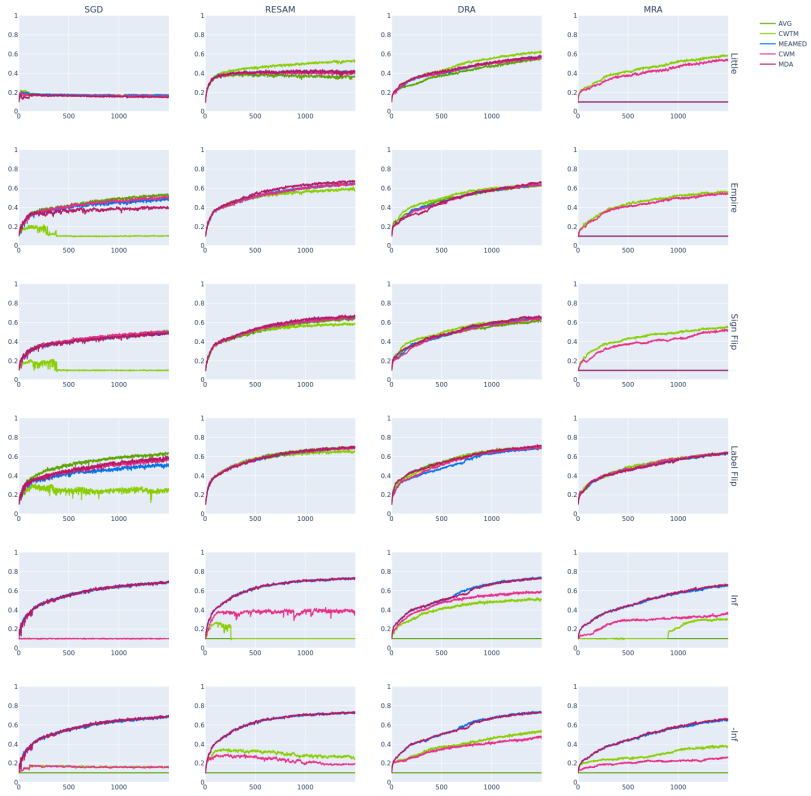

Figure 4.9: The columns show the test accuracy computed after each training round on homogeneously distributed data. The columns represent the optimisation algorithm (SGD, RESAM, DRA, MRA) while the rows correspond to the attacks; Little, Empire, Sign Flip, Label Flip, Infinity and -Infinity. Each line in the subplots corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. Little is again the worst attack, SGD tops at 0.175 and RESAM even decreases. DRA and MRA are the best performers but MRA suffers the same flaw as MNIST

Looking at the following table, we note that DRA is the best performer twice and is never the worst performer. This is an improvement compared to homogeneous MNIST where it never was the best and was the worst once. Furthermore, SGD is the worst performer against any attack except the infinity ones. Furthermore, while RESAM and DRA with CWTM perform the best against little, MDA is the best aggeration function against Empire, Sign Flip and Label Flip with MDA.

	Little	Empire	Sign Flip	Label Flip	Inf	-Inf
SGD	0.1725 MEAMED	0.5495 AVG	0.5105 AVG	0.6315 AVG	0.6915 MEAMED	0.686 MEAMED
RESAM	0.538 CWTM	0.6705 MDA	0.671 MDA	0.707 MDA	0.7385 MEAMED	0.742 MEAMED
DRA	0.6125 CWTM	0.6625 MDA	0.659 MDA	0.7105 MDA	0.727 MDA	0.725 MDA
MRA	0.574 CWTM	0.559 CWTM	0.5495 CWTM	0.6425 CWM	0.6575 MDA	0.6575 MDA

Table 4.3: Table showing the highest test accuracy and aggregation function when trained on homogeneously distributed data after the last update for each optimisation algorithm (row) and attack (column). The values highlighted in green correspond to the best performer while the red corresponds to the worst

Furthermore, we note that RESAM and DRA performance are closer on CIFAR10 than MNIST against Empire. Indeed, DRA accuracy on MNIST is 0.0245 smaller than RESAM while the gap is 0.008 on CIFAR10. The same is observed for Sign Flip as the gap is 0.012 on CIFAR10 but 0.0315 on MNIST.

4.2.2.2 Heterogeneously Distributed Data

Moving on to heterogeneous data without attacks, MRA is the best performer with a final test accuracy of 0.6115 using CWTM, followed by RESAM (0.5935) also with CWTM. The worst performer is DRA with an accuracy of 0.57 and CWTM as an aggregation method.

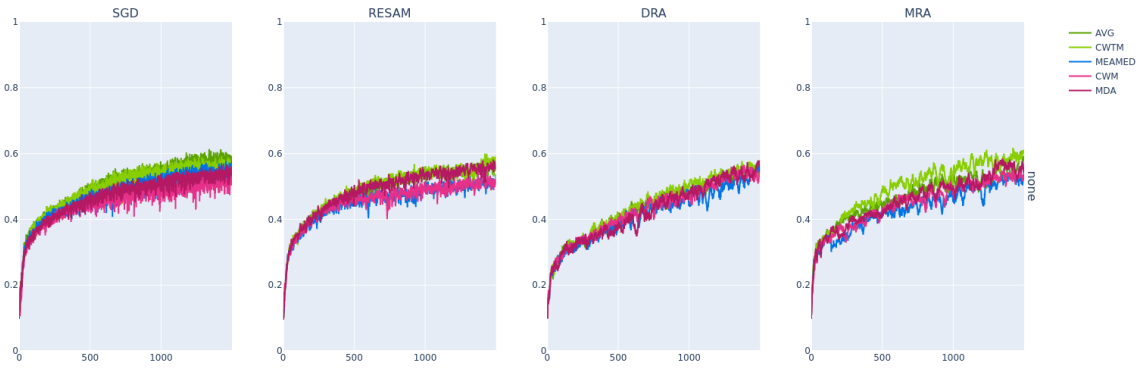

Figure 4.10: The columns show the test accuracy computed after each training round on heterogeneously distributed data without byzantine participants for each optimisation algorithm (SGD, RESAM, DRA, MRA). Each line in the subplot corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red.

When we add the attacks, DRA and RESAM perform very similarly. While DRA with MEAMED outperforms RESAM on Little, DRA's accuracy is 0.0015 points

lower than RESAM against Empire, 0.035 against Sign Flip, 0.003 against Label Flip and 0.0075 against both infinity attacks. Furthermore, MRA performs especially well on Label Flip.

	Little	Empire	Sign Flip	Label Flip	Inf	-Inf
SGD	0.1565 CWM	0.32 CWM	0.324 MEAMED	0.4075 MEAMED	0.522 CWM	0.522 CWM
RESAM	0.2135 CWTM	0.3025 CWM	0.3145 CWM	0.4265 CWTM	0.4795 MDA	0.4795 MEAMED
DRA	0.3845 MEAMED	0.301 MDA	0.311 MDA	0.4235 CWM	0.472 MEAMED	0.472 MEAMED
MRA	0.1 AVG	0.1 AVG	0.1 AVG	0.469 CWTM	0.503 CWM	0.503 CWM

Table 4.4: Table showing the highest test accuracy and aggregation function when trained on heterogeneously distributed data after the last update for each optimisation algorithm (row) and attack (column). The values highlighted in green correspond to the best performer while the red corresponds to the worst

Figure A.2 shows the accuracies after each update for heterogeneously CIFAR-10 with the attacks.

5
Conclusion

We presented and investigated two new Adam based frameworks for distributed machine learning. RESAM performed the best on the homogeneous MNIST data set, but DRA performed better on homogeneous CIFAR10 against Little and Label Flip and performed similarly to RESAM against Empire. Besides the need to aggregate two momentums, MRA has a big flaw due to its sensitivity to negative moments. While detecting negative moments can help us filter out byzantine nodes, it introduces new computational steps limiting the scalability of the algorithm.

Concerning heterogeneously distributed data, DRA performs especially well against Little. However, it performs poorly compared to RESAM on the MNIST data set. Similarly to homogeneously distributed data, DRA performs close to RESAM and outperforms it against Little. On the other hand, MRA won't converge when the attacks create negative entries in the second moment. However, it is the best performer on CIFAR10 against Label Flip.

From the experiments, DRA seems to be a promising framework as it performs very closely to RESAM on a more complex data set and model. While MRA performed well in some situations, its sensitivity to the sign of the second moment and the work needed (perform the aggregation twice and remove the negative sign in the second moment), it does not seem to be a good algorithm in practice compared to DRA which performs better and requires close to half the work. In this thesis, we also started on the convergence proof for DRA on homogeneously distributed data. However, in order to complete the proof, one still has to find the informative upper bound on terms containing the difference between the aggregation function and the mean of the directions computed by the honest workers, i.e. ξ.

However, due to time and hardware constraints, we had to limit the number of rounds. It would be worth running the experiments for longer runs as it might yield a smaller gap between RESAM and DRA. Furthermore, as mentioned in the introduction, this method removes the data gathering part but that does not mean Distributed Machine Learning is private. Hence, it would be interesting to study the behavior of DRA coupled with privacy enhancing methods.

Bibliography

[1] S. Farhadkhani, R. Guerraoui, N. Gupta, R. Pinot, and J. Stephan, "Byzantine Machine Learning Made Easy by Resilient Averaging of Momentums," in *International Conference on Machine Learning*, 2022.

[2] T. Tieleman and G. Hinton, "Lecture 6.5-RMSProp, coursera: Neural networks for machine learning," *University of Toronto, Technical Report*, vol. 6, 2012.

[3] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[4] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and Open Problems in Federated Learning," *CoRR*, vol. abs/1912.04977, 2019.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[6] P. Rousseeuw, "Multivariate Estimation With High Breakdown Point," *Mathematical Statistics and Applications Vol. B*, pp. 283–297, 1985.

[7] R. Guerraoui, S. Rouault *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3521–3530.

[8] C. Xie, O. Koyejo, and I. Gupta, "Generalized byzantine-tolerant SGD," *CoRR*, vol. abs/1802.10116, 2018. [Online]. Available: http://arxiv.org/abs/1802.10116

[9] M. J. Sai Praneeth Karimireddy, Lie He, "Byzantine-Robust Learning on Heterogeneous Datasets via Bucketing," *International Conference on Learning Representations*, 2022.

[10] Y. Allouah, S. Farhadkhani, R. Guerraoui, N. Gupta, R. Pinot, and J. Stephan, "Fixing by Mixing: A Recipe for Optimal Byzantine ML under Heterogeneity," 2023. [Online]. Available: https://arxiv.org/abs/2302.01772

[11] C. Xie, S. Koyejo, and I. Gupta, "Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, ser. Proceedings of Machine Learning Research, R. P. Adams and V. Gogate, Eds., vol. 115. PMLR, 22–25 Jul 2020, pp. 261–270. [Online]. Available: https://proceedings.mlr.press/v115/xie20a.html

[12] M. Baruch, G. Baruch, and Y. Goldberg, "A Little Is Enough: Circumventing Defenses For Distributed Learning," in *Neural Information Processing Systems*, 2019.

[13] Z. Allen-Zhu, F. Ebrahimian, J. Li, and D. Alistarh, "Byzantine-resilient non-convex stochastic gradient descent," *arXiv preprint arXiv:2012.14368*, 2020.

[14] A. Défossez, L. Bottou, F. Bach, and N. Usunier, "A Simple Convergence Proof of Adam and Adagrad," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://openreview.net/forum?id=ZPQhzTSWA7

[15] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

A

Appendix 1

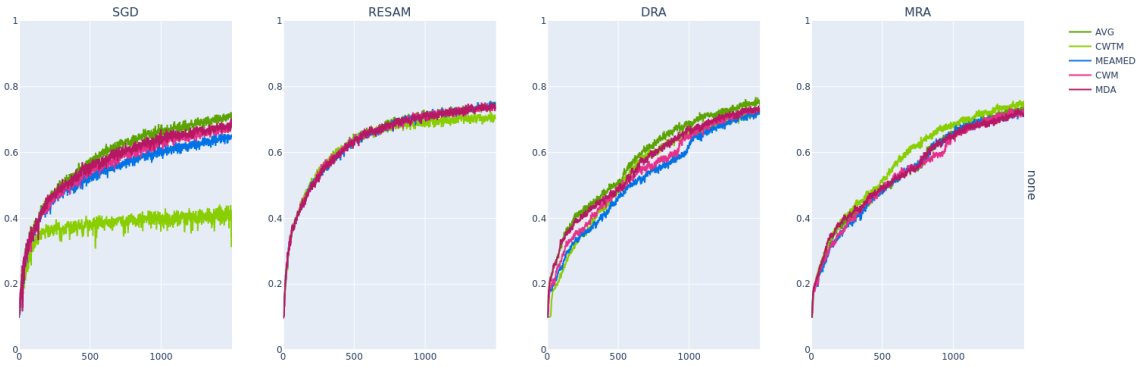

Figure A.1: The columns show the test accuracy computed after each training round on homogeneously distributed CIFAR-10 without byzantine participants for each optimisation algorithm (SGD, RESAM, DRA, MRA). Each line in the subplot corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red.

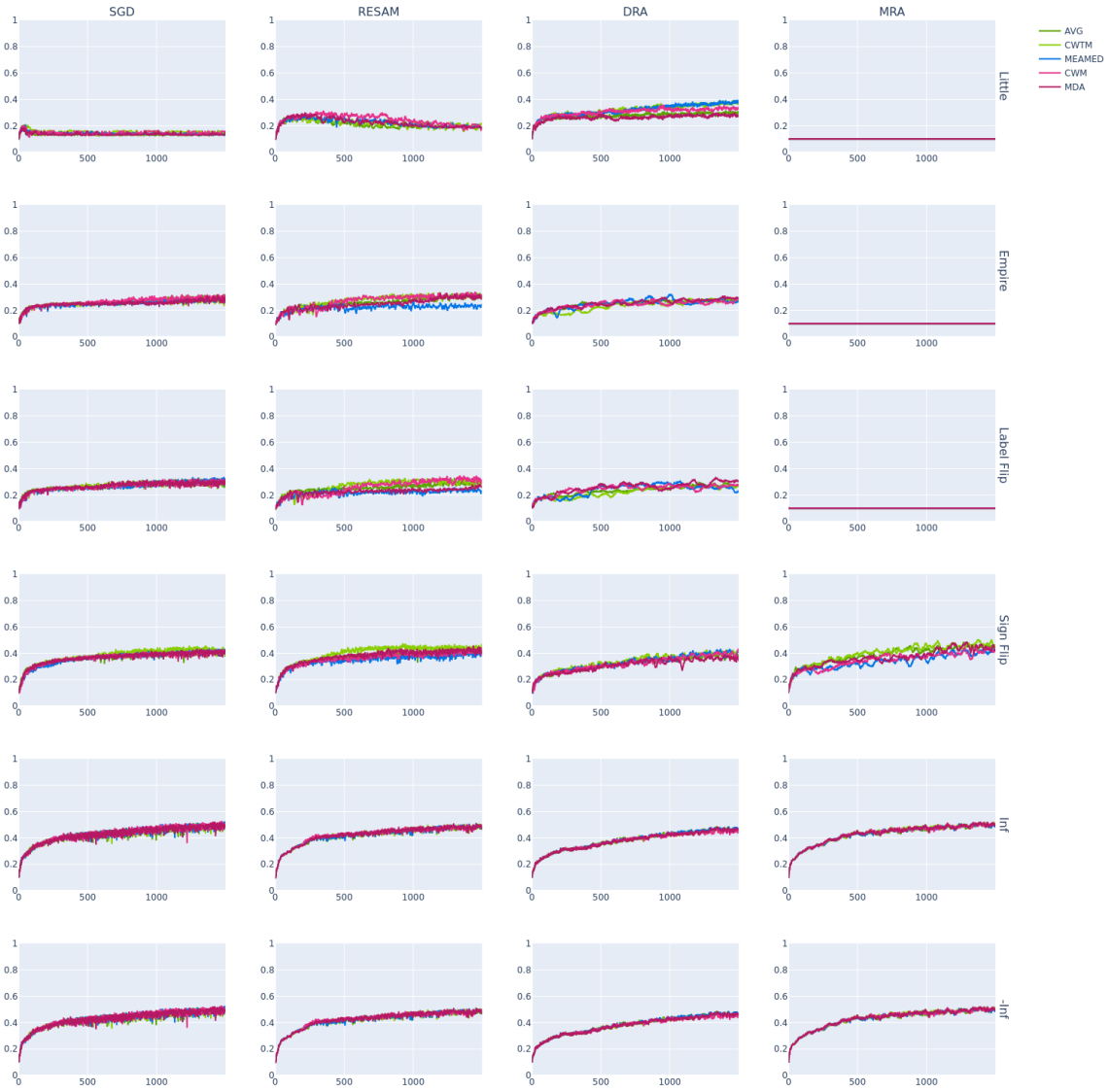

Figure A.2: The columns show the test accuracy computed after each training round on homogeneously distributed CIFAR-10 data. The columns represent the optimisation algorithm (SGD, RESAM, DRA, MRA) while the rows correspond to the attacks; Little, Empire, Sign Flip, Label Flip, Infinity and -Infinity. Each line in the subplots corresponds to an aggregation function; AVG in dark green, CWTM in light green, MEAMED in blue, CWM in pink and MDA in red. Plot of the test accuracies after each update. Here again MRA won't converge when the gradients are alterated

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY