AdaMM vs Z0-AdaMM: Convergence Analysis and Minima Shape

Cyrille Pittet, Elia Fantini, Kieran Vaudaux
EPFL, Switzerland

***Abstract*—The adaptive momentum method (AdaMM) is a first-order optimisation method that is increasingly used to solve deep learning problems. However, like any first-order (FO) method, it is only usable when the gradient is computable. When this is not the case, a zero-order (ZO) version of the method can be used. This paper proposes to investigate the convergence of a ZO version, ZO-AdaMM, of the AdaMM method. We have shown that there is a slowdown of the order of $\mathcal{O}(\sqrt{d})$ in the convergence of the ZO method compared to the FO method. Moreover, we have managed to obtain reasonable performances with our ZO method and have proposed improvement ways to obtain better performances. Finally, we have highlighted the convergence to different minima for the ZO and FO methods.**

I. Introduction

Increasingly, signal processing, machine learning (ML) and deep learning (DL) involve tackling complex optimisation problems that are difficult to solve analytically. To solve these problems many optimisation algorithms have been developed and introduced during the last decades. The majority of these algorithms rely on the fact that the first order gradient information is easily accessible since the closed form of the gradient of the function can generally be computed easily for many problems. Nevertheless, the development of gradient-free optimisation methods has become increasingly important for solving many machine learning problems for which the explicit expression of the gradients is expensive or unobtainable. Optimisation for these types of problems falls into the category of zero-order (ZO) optimisation over black-box models. The idea behind zero-order optimisation methods is to mimic first-order optimisation methods by approximating the gradient with finite difference schemes along random directions. Approximating the gradient indubitably slows down the convergence of zero-order algorithms compared to first-order algorithms. Indeed, many articles ([1]) have already discussed this subject and have highlighted a slowing down of the convergence of zero-order methods compared to first-order methods, of the order of $\mathcal{O}(\sqrt{d})$ where d is the number of variables to be optimized.

This article proposes to study, at first, the emergence of this $\mathcal{O}(\sqrt{d})$ bound in the framework of the Zero-Order Adaptative Momentum Method (ZO-AdaMM) which is a gradient-free variant of the AdaMM, itself being a modified version of the Adam algorithm. Following this, this paper proposes to focus on the shape of the minima reached by these first order and zero order methods. These two elements are studied in the context of training several networks of varying sizes on the MNIST dataset. All the experiments, the results, as well as the implementation from scratch of the zero and first order methods are available on our GitHub.

II. Zero-Order Gradient Estimation

This section provides an overview of the way to estimate the gradient used in this paper, as well as potential improvements that could be considered in a follow-up to this paper.

Suppose here that we want to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$. A natural way of estimating the gradient proposed in the article [1] is to use the forward differences of f:

$$\hat{\nabla} f(\mathbf{x}) := \frac{\phi(d)}{\mu} \left(f(\mathbf{x} + \mu \mathbf{u}) - f(\mathbf{x}) \right) \mathbf{u} \tag{1}$$

where $\mathbf{u} \sim p$ is a random direction drawn from a uniform distribution on the unit sphere of $\mathbb{R}^d$, $p \sim \mathcal{U}(\mathcal{S}(0,1))$ (other distributions can be used, but we will focus on this one), $\mu > 0$ is a perturbation radius and $\phi(d)$ is a certain dimension-dependent factor related to the distribution p, in our case $\phi(d) = d$, ([2]).

For a fixed number of optimisation parameters, the expected approximation error of the gradient, $\mathbb{E}\left\{ \|\hat{\nabla} f(\mathbf{x}) - \nabla f(\mathbf{x})\|_2^2 \right\}$, gets better as the smoothing parameter μ tend to 0, as show in [1]. Nevertheless, in practice this result can be compromised by the noise generated by the computer's approximation errors when μ becomes too small. Moreover, contrary to the first-order stochastic gradient, the zero-order gradient estimate suffer of a dimension dependent variance which increases as $\mathcal{O}(d) \|\nabla f(\mathbf{x})\|_2^2$. A common way to reduce the variance of the zero-order gradient is to approximate the gradient on a $i.i.d.$ mini-batch of random direction sample from $\mathcal{U}(\mathcal{S}(0,1))$:

$$\hat{\nabla} f(\mathbf{x}) := \frac{\phi(d)}{\mu} \frac{1}{b} \sum_{i=1}^{b} \left(f(\mathbf{x} + \mu \mathbf{u}_i) - f(\mathbf{x}) \right) \mathbf{u}_i \tag{2}$$

For reasons of simplicity of implementation and computation time, the experiments in this paper were carried out by estimating the gradient at a point $\mathbf{x}$ using a mini-batch of a single random direction, i.e. using the Eq.1. But, as often used in deep learning model optimization, the gradient is often computed for mini-batches of inputs $\{\mathbf{x}_i\}_{i=1}^n$. Here, the estimation of the gradient on a mini-batch will be performed by drawing a single random direction for all the batch.

$$\hat{\nabla} f(\{\mathbf{x}_i\}_{i=1}^n) := \frac{\phi(d)}{\mu n} \left[\sum_{i=1}^{n} \left(f(\mathbf{x}_i + \mu \mathbf{u}) - f(\mathbf{x}_i) \right) \right] \mathbf{u} \tag{3}$$

III. AdaMM algorithm

Although zero-order methods can be applied to all first-order optimisation algorithms, this article will focus on the AdaMM ([1, 3]), with the algorithm available in the appendix A. In our case, we do not have any constraints in the parameters space, so we have $\mathcal{X} = \mathbb{R}^d$ and the Mahalanobis projection is then the identity.

IV. Experiments

This section provides results and insights on the following questions :

- How do the two methods' performances compare?

- Can we observe the theoretical $\mathcal{O}(\sqrt{d})$ decrease in performance with ZO-AdaMM?
- Do the two methods lead to convergence towards similar minima ?

A. Setting

In this paper we consider the classification task on the MNIST dataset. The models used are simple modular convolutional neural networks whose architecture is shown in figure 1. We found empirically that the default hyperparameters

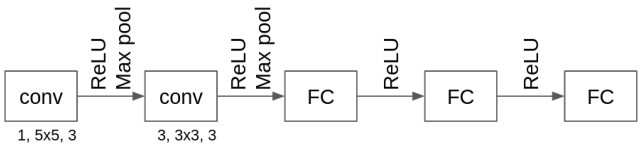

Fig. 1: Modular model used in experiments. The size of the two convolutional layers are fixed whereas the number of neurons in the following 3 fully connected layers can be changed. Models considered ranges from 1400 parameters up to 2'806'751 parameters.

for both AdaMM and ZO-AdaMM worked best, namely we set $\beta_1 = 0.9$, $\beta_2 = 0.999$. The learning rate is decreased according to a plateau detection in the test loss. We also empirically found that decreasing the smoothing parameter μ of the ZO-AdaMM worked well, therefore it is decreased with the same schedule as the learning rate. Both are first initialized to 10^{-3}. To have reproducible results, we use a normal Xavier initialization with a given seed for the weights and initialize the biases to 0.01. Each model is trained 10 times (each with a different seed) over 50 epochs with a batch size of 128.

All the results are reproducible as follows :

- Train the models : run the "experiments" notebook
- Reproduce the plots : run the "analysis" notebook

B. Performance comparison and $\mathcal{O}(\sqrt{d})$ bound analysis

Due to the nature of the zeroth order method, which relies only on estimations of the gradient to optimize the model's parameters, we could not expect ZO-AdaMM to reach a performance similar to the first order counterpart. On the other hand, while AdaMM optimizer reached an accuracy of 0.99, ZO-AdaMM also managed to reach almost 0.90 in the experiments with bigger models. From figure 2 we can see that augmenting the model's size doesn't deteriorate accuracy, instead it correctly improves the train loss, showing that the gradient estimation works even when the optimization landscape has much higher dimensions. It is worth noting that our way of scaling up the model's size is trivial and a more complex method, such as the one proposed by EfficientNet's paper [4] would probably lead to a higher performance-scale ratio. Theoretically, the convergence rate of zeroth-order (ZO) AdaMM for both convex and nonconvex optimization is roughly a factor of $O(\sqrt{d})$ worse than that of the first-order (FO) AdaMM algorithm, where d is the problem size (number of the model's parameters). This means that we should observe:

$$Loss_{ZO} \approx Loss_{FO} \cdot \sqrt{d} \cdot k \rightarrow k \approx \frac{Loss_{ZO}}{Loss_{FO} \cdot \sqrt{d}}$$

Indeed, we managed to analyse such theoretical bound in practice. As shown in figure 3, the k value gets almost constant after $\sim$ 25 epochs, $\sim$ 5 with the smallest model, since the

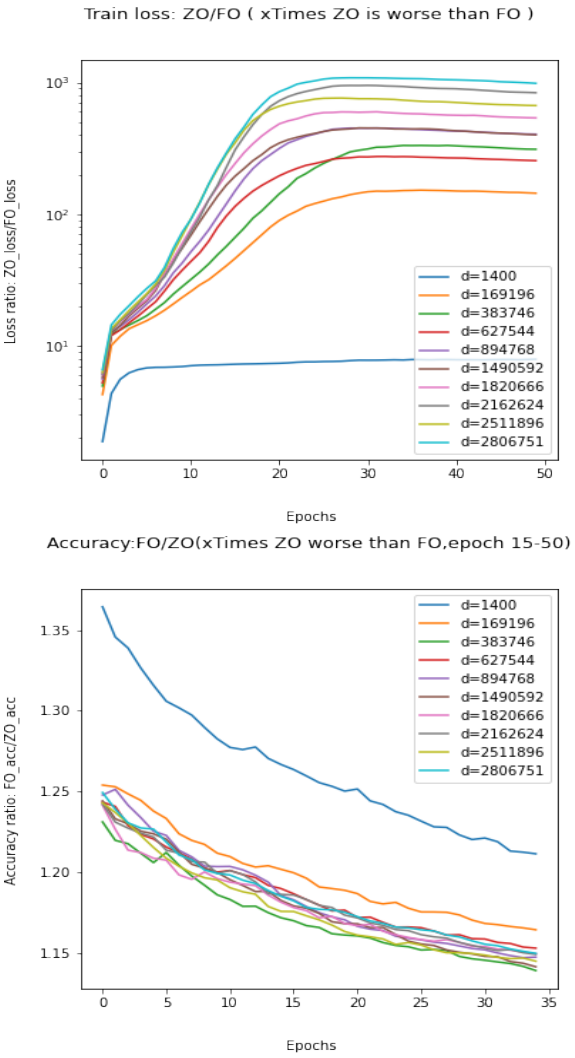

Fig. 2: ZO vs FO AdaMM: train loss and accuracy ratio

learning stabilizes sooner. Most importantly, the value of k is almost the same across all scales, showing that the theoretical is confirmed on practice. On the other hand, the observed $k \approx 0.5$ is quite high: considering the number of parameters of modern neural networks (hundreds of billions) and the corresponding $\sqrt{d} \cdot k$ value, the decrease in performance would likely be too big to apply a zero order method.

Finally, regarding the computation time of the two methods, ZO-AdaMM goes from an average of $\sim$ $17s$ per epoch for the smallest models until $\sim$ $28s$ for the biggest one, while AdaMM remains always around $\sim 12s$ per epoch. The difference between the two is solely due to the computation of the gradient's estimate, which gets slower as the number of dimensions of the parameters' space gets higher.

C. Similar minima

Since the only difference between the zeroth order and first order versions of AdaMM resides in the way we compute the gradient, i.e. estimate it with finite difference on a random direction and compute it exactly respectively, both methods might tend to converge to similar minima. To investigate this we recorded the weights of each layer of the smallest model (1400 parameters) after each epoch, trained with both optimizers.

To see if convolutional layers could get similar results, we printed the final filters, obtained after 50 epochs, as images, but we couldn't see any significant similarity, neither a kernel

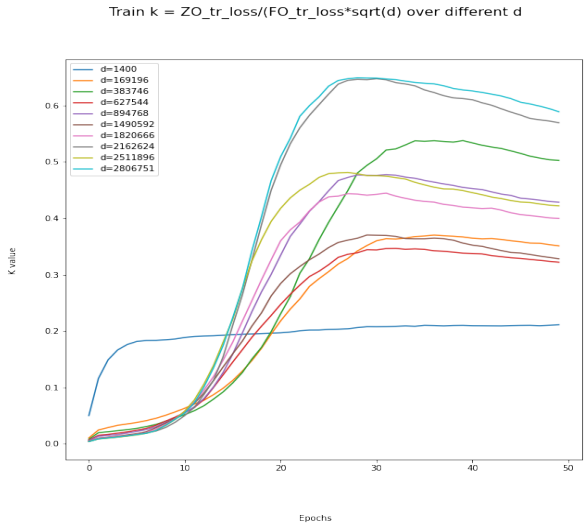

Fig. 3: Analysis of k value trough epochs and across different model's scales

similar to a Gabor filter, which is typical in the first layer of CNN for image classification. Hence, we tried to project the weights of each individual filter into a 2-dimensional space using the t-SNE algorithm, which tries to preserve in low dimensions the neighborhood of the data points in high dimensions. In figure 5 is shown the result of such computation.

While weights start in the same place (due to being initialized identically), the weights obtained from AdaMM and ZO-AdaMM go directly in very different directions. During the first epochs, the first order (FO) method is able to make much larger steps compared to the zeroth order (ZO) one. This is the advantage of computing the exact gradient, giving the direction of maximum decrease (exact in the sense that it is not estimated but computed explicitly for a given mini-batch) compared to the ZO estimation which only gives an estimation of the gradient in a random direction. The gradients of the FO method tend to be more aligned than the ones of ZO method. This allows AdaMM to build momentum in the first moment estimation ($\mathbf{m_t}$ in algorithm 1), hence resulting in larger weights updates. A way to improve on this aspect for the ZO method could be to sample multiple random directions, estimate the gradient along each of these directions for each inputs of the batch, and then take the average on the batch and the randoms directions, which gives us a combination of Eq.2 and Eq.3.

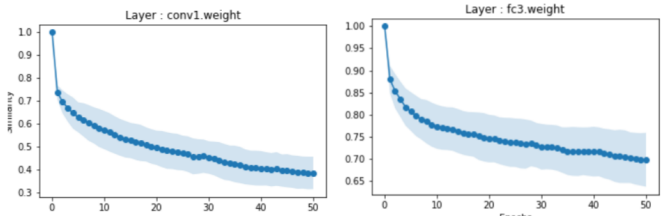

Fig. 4: Cosine similarities between models' parameters using AdaMM and ZO-AdaMM. The curves are the mean $+/-$ standard deviation over 10 experiments.

We also compared both models' weights from different layers with cosine similarity, most significant results are shown in figure 4, while the complete analysis is shown in appendix figure 6. Again, we can see that convolutional layers are very different. The similarity for the last fully connected layer is higher (≈ 0.7): this might suggest that even if convolutions extract different features, the final division of space through hyperplanes is similar, but not identical, which is consistent with the difference in performance of the two models.

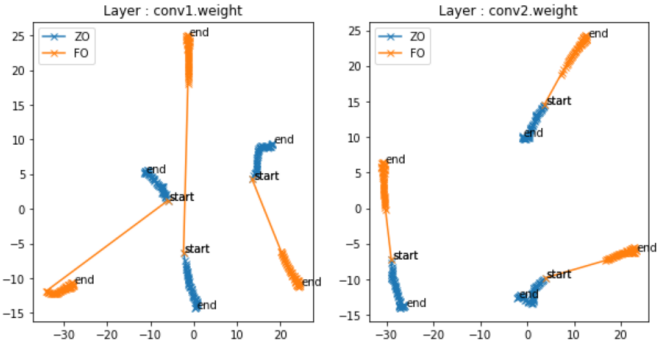

Fig. 5: Projection of the 3 convolutional filter (without bias) for each of the two convolutional layers (single experiment).

V. Conclusions

In this paper, we first proposed to compare the overall performance of the ZO and FO versions of the AdaMM algorithm. Although it achieves acceptable accuracies, the ZO version still suffers from a certain slowdown compared to the FO version. This is due to the fact that our ZO version allows the parameters to move in a single direction which is certainly biased with respect to the exact gradient. However, the accuracy continued to increase slowly and the train loss to decrease, so we think we would achieve better results even if it would take some time. Then, we were interested in the appearance of this theoretical bound of $\mathcal{O}(\sqrt{d})$. According to our experiments, this theoretical bound does indeed seem to occur, which would make it difficult to use our method for larger and more complex models. Finally, we are interested in the shape of the minima reached by each of the two methods. Although the fact that the parameters converge to quite distinct minima, deducing that one of these minima is "better" than the other is a complicated subject. Nevertheless, the fact that for equal training losses, the FO method has a much higher accuracies than the ZO method, leads us to believe that the ZO method converges to a local minimum and the FO to a global minimum or at least a "better" local minimum.

The major improvement of our method lies in reducing the bias of the gradient estimation by using a mini-batch of random directions as in Eq.2. We believe that this would allow us to have a better chance of converging this time to the minima reached by the FO method and also to accelerate the convergence, which so far remains rather slow as we explore the parameter space one direction at a time.

References

[1] Xiangyi Chen et al. "ZO-AdaMM: Zeroth-Order Adaptive Momentum Method for Black-Box Optimization". In: (2019).

[2] Abraham Flaxman, Adam Tauman Kalai, and H. Brendan McMahan. "Online convex optimization in the bandit setting: gradient descent without a gradient". In: (2004).

[3] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: (2019).

[4] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: (2019).

APPENDIX A
ADAMM ALGORITHM

Algorithm 1: AdaMM

Input: $\mathbf{x}_0 \in \mathcal{X}$, step sizes $\{\alpha_t\}_{t=1}^T$ and $\beta_1, \beta_2 \in (0, 1]$

for $t = 0, ..., T$ **do**

$\mathbf{g}_t = \nabla f(\mathbf{x_t})$

$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$

$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t^2$

$\hat{\mathbf{v}}_t = \max(\hat{\mathbf{v}}_{t-1}, \mathbf{v}_t)$, $\hat{\mathbf{V}}_t = \text{diag}(\hat{\mathbf{v}}_t)$

$\mathbf{x_{t+1}} = \Pi_{\mathcal{X}, \hat{\mathbf{V}}_t^{-\frac{1}{2}}}(\mathbf{x_t} - \alpha_t \hat{\mathbf{V}}_t^{-\frac{1}{2}} \mathbf{m}_t)$

end

where $\Pi_{\mathcal{X},\mathbf{V}}(\mathbf{a})$ denotes the projection operation under Mahalanobis distance with respect to $\mathbf{H}$, .i.e $\arg\min_{\mathbf{x}\in\mathcal{X}} \|\mathbf{H}^{-\frac{1}{2}}(\mathbf{x} - \mathbf{a})\|_2^2$. The zero order version of Adamm (ZO-AdaMM) simply replaces the gradient $\nabla f(x_t)$ by the estimate of eq. 3.

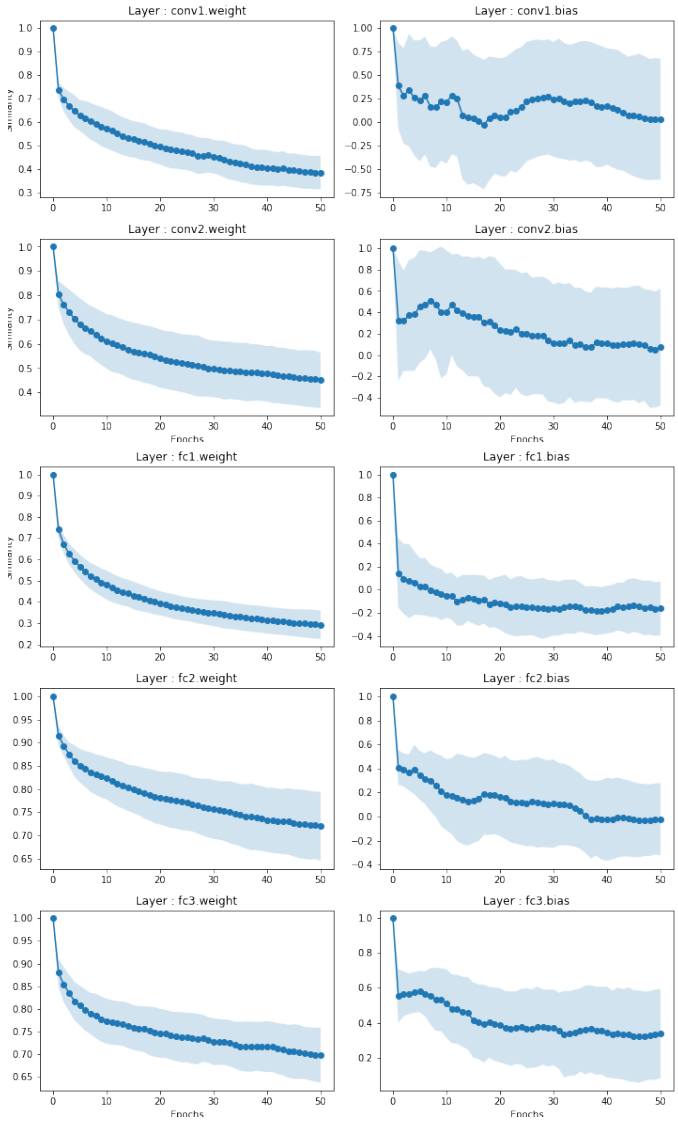

Fig. 6: Cosine similarities between models' parameters using AdaMM and ZO-AdaMM. The curves are the mean $+/-$ standard deviation over 10 experiments.