Discussion on the viability of a modern Second Order Method in Non-Convex Optimization training a Deep Convolutional Neural Network

Raphaël Attias, Riccardo Cadei, Miloš Novaković
Department of Computer Science, EPFL, Switzerland

I. Introduction

Second order algorithms are among the most powerful optimization algorithms with superior convergence properties as compared to first order methods such as SGD and Adam. However computing or approximating the curvature matrix (Hessian) can be very expensive both in per-iteration computation time and memory cost. In this paper, we analyze the convenience in using a state-of-the-art Second Order Method (ADAHESSIAN [1]) in Non-Convex Optimization training a Deep Convolutional Neural Network (RESNET18 [2]) on MNIST data-set comparing with traditional First Order Methods. In fact almost all the theoretical results of these methods cannot be extended to Non-Convex optimization and we have to limit to experimental comparisons. Advantages and disadvantages of both the methods are discussed and a final HYBRID method combining the advantages of both is proposed.

In Section II all the optimization algorithms are introduced mathematically, in the section III the problem set up is described and the results of the experimental tests are reported, in Section IV these results are discussed and a final conclusion is reported in Section V.

II. Optimization Methods

A. First Order Methods

First order methods are attractive methods in optimization for finding a local minimum of a differentiable function. This technique is based on the observation that if a multivariate-function $f(x)$ is defined and differentiable in a neighborhood of a point y, then it decreases the fastest in the direction of the negative gradient of function f at y, i.e. $-\nabla f(y)$.

There are multiple variations, but these methods can be represented using the following general update formula (1):

$$x_{n+1} = x_n - \eta_n \frac{m_n}{v_n} \tag{1}$$

where η_n is the learning rate, m_n is the first order momentum and v_n is the second order momentum.

The first and second momentum can be understood as the mean and non-centered variance of the gradients respectively, hence their name, and their purpose. Compared to the simple gradient descent rule, it is automatically adjusting the learning rate and a preconditionning the gradient using the previous updates.

A simple and popular update method is Stochastic Gradient Descent (SGD) with (first order) momentum, defined as:

$$m_n = \beta m_{n-1} + (1-\beta) g_n \qquad v_n = 1 \tag{2}$$

where g_n is the gradient of the mini-batch at the m-th iteration, and $0 \leq \beta < 1$ is the first momentum hyper-parameter defining the first momentum as a linear combination of the gradient and the previous update. This method built the foundations of optimization in deep learning but using it in practice can be challenging, as it is sensible to the starting point and learning rate.

To address this problem, one can use the geometry of the data by scaling the update with respect to the size of past gradients. This is the role of the second momentum v_n, that can be understood as a scaling or normalizing parameter. One notable method is ADAM, where the idea is to keep an estimation of the first and second moment in a small window of past iterations, where old moments will have exponentially less impact on the current moment estimation. The Adam optimization algorithm is defined by first moment estimator (3) and with the second moment estimator (4), where the update rule is defined in (1).

$$m_n = \frac{(1-\beta_1)\sum_{i=1}^{n} \beta_1^{n-i} \mathbf{g}_i}{1-\beta_1^n}, \tag{3}$$

$$v_n = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{n} \beta_2^{n-i} \mathbf{g}_i \mathbf{g}_i}{1-\beta_2^n}} \tag{4}$$

where $0 < \beta_1, \beta_2 < 1$.

B. Second Order Method: ADAHESSIAN

As the name suggest, Second Order Methods are powerful optimization methods that use information provided by the first and second derivatives of the function, i.e., it encapsulates the information given both by the slope and curvature of the loss landscape. *Newton's method* is a fundamental second order method, which the idea is a basis of many other optimization techniques. Using the second order Taylor expansion of f, and the necessary but not sufficient condition that $\nabla f(x_{n+1}) \simeq 0$:

$$f(x_n + p_n) \simeq f(x_n) + \nabla f(x_n)^T p_n + \frac{1}{2} {p_n}^T H_n p_n \tag{5}$$

$$0 = \nabla f(x_n) + H_n p_n \tag{6}$$

where H_n is the Hessian of f at x_n, and p_n is the direction of the new iteration such that $x_{n+1} := x_n + p_n$. The update iteration requires to find p_n such that

$$H_n p_n = -\nabla f(x_n) \tag{7}$$

This can be solved explicitly by inverting the Hessian. Computing the Hessian and its inverse is costly and may lead to stability issues. ADAHESSIAN is a state-of-the-art second order method that attempts to solve equation (7) using similar momentum techniques used in ADAM. Firstly, instead of using the full Hessian in equation (7), it can be shown that the Hessian diagonal $D = \text{diag}(H)$ has the same convergence rate on simple strongly convex and strictly

smooth function [1]. Furthermore, this diagonal can be efficiently estimated using the Hutchinson's method, denoted as:

$$D = \text{diag}(H) = \mathop{\mathbb{E}}_{z \sim Rademacher(0.5)}[z \odot (Hz)] \tag{8}$$

where $\odot$ is the Hadamard product, and random vector z follows a Rademacher(0.5) distribution.

Secondly, since the Hessian diagonal D may variate greatly in its components, we can compute the spatial average of D named $D^{(s)}$, because we are now looking at the vector D, and not on the Hessian matrix H [1]. Similarly, it can be shown that using the space averaged diagonal in equation (7) provides the same convergence rate for simple strongly convex and strictly smooth functions [1]. We define a new hyper-parameter b, which is the spatial average block size. The simple spatial average on the Hessian diagonal can be computed as shown in (9) equation.

$$\boldsymbol{D}^{(s)}[ib+j] = \frac{\sum_{k=1}^{b} D[ib+k]}{b}, \text{ for } 1 \leq j \leq b,\ 0 \leq i \leq \frac{d}{b} - 1 \tag{9}$$

Then AdaHessian combines the first order momentum used in ADAM defined in Equation (3), and the second momentum defined in the Equation (10).

$$v_n = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{n} \beta_2^{n-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^n}} \tag{10}$$

The result is an efficient computation of a sufficient estimation of the Hessian (the spatial average of the diagonal) which has $\mathcal{O}(d)$ memory complexity (as SGD), which provides states-of-the-art performance on deep learning training. The main overhead of this method is the estimation of the Hessian diagonal vector D by the Hutchinson's method. In total the computation time of a step of ADAHESSIAN is $2\times$ the one of SGD, which can be worsen by Pytorch, heavily optimized for first order methods.

We finally propose an HYBRID optimizer combining STOCHASTIC GRADIENT DESCENT using Momentum and ADAHESSIAN. Our hypothesis is that a simple first order optimizer is preferable in the first steps since it is faster and and the estimation of the Hessian in the first steps can be very ill-conditioned, while in the last steps, once the we are close to an optimum, the Local Convergence Theorem states that the convergence rates for a second order method is $\mathcal{O}(loglog(1/\epsilon))$ (super-exponential). Actually the theorem doesn't hold if the objective function is Non-Convex, and for this reason we want to experimentally study if there are benefits in combining the 2 methods. As a criteria to switch from a an optimizer to the other we considered a threshold tl in the loss evaluated on the validation set.

III. EXPERIMENTAL TESTS

A. Problem Setup

We compared all the methods discussed in Section II in a Non Convex optimization task: training a Deep Convolutional Neural Network (DCNN) for Image Classification. In particular we considered the deep residual network RESNET18 [2], having 11 175 370 parameters, to make digits classification on a subset (2%) of the MNIST dataset [3] composed by 1200 grey scale (1 channel) images 28x28 pixels for training, 200 for validation and 200 for test. For all the method we considered the same training setting, as well starting weights of the model, to facilitate the final comparisons. So, we fixed the number of epochs equal to 15, the batch size equal to 100 and we trained all the models minimizing the Cross Entropy Loss without any regularization. We used Google Colaboratory resources for training, which corresponds to a Nvidia K80 GPU with 12 GB of GPU memory.

B. Results

In order to make consistent comparisons we fixed the the training setting for all the methods and we individually optimized each method hyper-parameters using grid search. In particular for each hyper parameter we proposed 5 different values and we used the mean accuracy on the validation test among 5 experiments for each set of hyper-parameters to select to best ones. Here, we report the best hyper-parameters that we have found:

- SGD: $lr = 0.01$
- SGD with momentum: $lr = 0.005$, $\beta = 0.9$
- Adam: $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
- AdaHessian: $lr = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
- Hybrid: $tl = 1.0$, $lr^1 = 0.005$, $\beta^1 = 0.9$; $lr^2 = 0.001$, $\beta_1^2 = 0.9$, $\beta_2^2 = 0.999$

For all the methods considered, the average accuracy ($\pm$ standard deviation) on the training, validation, and test set among 10 experiments is reported in Table I. In Table II we also report for each optimizer the mean ($\pm$ standard deviation) training time among the same 10 experiments.

Optimizer	Train	Validation	Test
SGD (β = 0)	100± 0	87.55±1.17	87.45± 1.42
SGD (β = 0.9)	100± 0	94.5± 1.03	93.5± 1.35
Adam	99.95±0.11	93.85± 1.53	93.35± 3.51
AdaHessian	100± 0	93.625± 2.01	93± 2.28
Hybrid	100± 0	94.1± 1.33	94.2± 0.79

Table I: The mean and standard deviation of accuracy for each optimizer after 15 epochs among 10 executions

Optimizer	Training Time
SGD (β = 0)	14.57±0.60
SGD (β = 0.9)	13.79±1.78
Adam	11.30±1.63
AdaHessian	14.26±3.27
Hybrid	17.43±0.85

Table II: The mean and standard deviation of training time in seconds for each optimizer during 15 epochs and 10 executions

Out of all the 5 methods, the proposed HYBRID method reaches the highest accuracy on the test, with 94.2 in average. Followed by ADAHESSIAN and ADAM, this experimental results seem to confirm the viability of second order methods for deep learning. We observe than that second order method ADAHESSIAN takes in average a similar time for training than the other first order methods, but notice a much greater variance. The HYBRID method was the slowest with a training time of 17.43 seconds in average.

The evolution of the cross-entropy loss and the accuracy for the 5 optimization methods after 15 epochs is presented in the Figure 1 and 2. In particular one can see that SGD reaches a lower accuracy and higher loss on the validation than every other methods. Additionally ADAHESSIAN performs remarkably well and converges to a higher accuracy sooner than the first order methods.

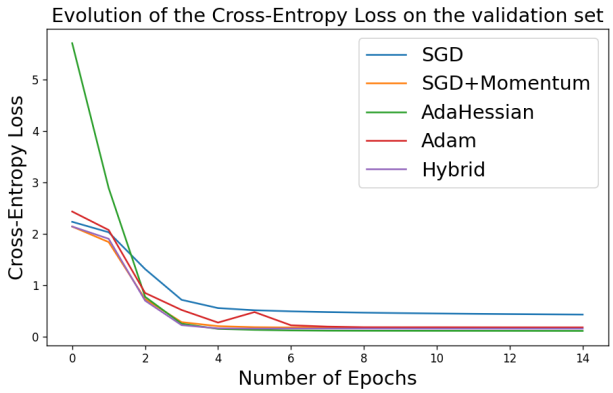

Figure 1: Evolution of the loss over the validation set, with respect to the number of epoch, for the 5 studied optimization methods.

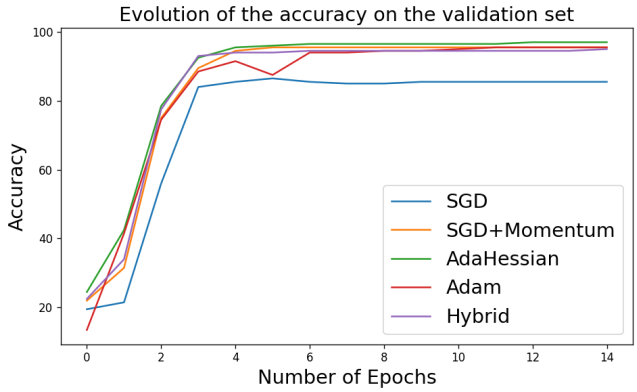

Figure 2: Evolution of the accuracy over the validation set, with respect to the number of epoch, for the 5 studied optimization methods.

Finally we decided to look at the spectral norm of the gradient of the loss function with respect to the weights of last linear layer and first convolutional layer (averaging among the 64 filters) of ResNet18 during the training. A decreasing norm of the gradient is associated to an inflection point, which may or may not be a local/global minimum, but provides a new insight in the optimization of the loss function. The plots of the evolution of the spectral norm of these gradients for all the methods are reported in Appendix A as supplementary material. The main result is that all the spectral norms seem to converge to 0, with the SGD Momentum and HYBRID optimizer decreasing first. We notice that the spectral norm of SGD does not decrease as quickly than the other methods, and ADAM is the only optimizer to show a loss which is not strictly decreasing.

IV. DISCUSSION

All the five methods considered are able to properly learn the multi-classification task (Accuracy > 0.8) after just few epochs (15) and a reduced version of the original dataset (2%). However different behaviours can be detected among them.

From the previous results we may experimentally observe that second order methods are a viable and practical solution for deep learning training. Both ADAHASSIAN and HYBRID reached in average comparable or better results than first order methods. Where second order methods fails are in the computation time, despite the approximation techniques from section II. The computation time of ADAHESSIAN was approximately 26% longer than ADAM, which at larger scale would be an issue. The performance and computation time observed for these two second order methods are as expected as seen in [1].

One additional insight, is that the first-order methods reach saturation (i.e. local training loss minima does not significantly improve) around 8-th epoch, whereas the second-order methods ADAHESSIAN and HYBRID optimization algorithm reach saturation around 4-th epoch. Therefore the second-order algorithms seem to require fewer number of training epochs, because they precondition the gradient before using it for weight update [1], i.e., they adapt every parameters update step, according to the parameters slope and curvature in the Loss landscape. We show that the computational training time cost of ADAHESSIAN is in the same comparable range as the first-order methods (Table II), but still tops out at the higher range of the computation time. The new techniques provided by the ADAHESSIAN optimization methods highlights the viability of the approximation made to minimize the loss function with second order methods. Despite all the approximations techniques the second order method still put some overhead on the computation time, but provide a faster convergence to a local optimum. Either or not this trade is worth it will depend on the problem and neural network architecture. From the observed results in this particular situation, we would say that both ADAHESSIAN and HYBRID are contenders to be state-of-the-art optimizers.

We have also shown that the spectral norms for the gradients of both first and the last layer go to zero, as the number of epochs increases (Figure 2). This is the result of the optimization algorithm, which reaches the local minima of the non-convex loss landscape, as optimizer tries to minimize it. The first layers gradient spectral norm is always higher than last layers gradient spectral norm, because the layers close to the output have more similarity of the output gradient (which is going to zero as we apply the optimization algorithm update step).

V. CONCLUSION

In this project we studied the convenience in using a modern Second Order Method (ADAHESSIAN) training a Deep Convolutional Neural Network. We considered SGD, SGD with momentum and Adam as a baseline. The main conclusion is that Second Order Methods are a promising alternative to First Order Methods when an efficient approximation of the Hessian is used. In fact both the computation time and the memory cost are in the same order of magnitude. In particular from the experiments it seems that the Local Convergence Theorem for Second Order Methods still holds even if the optimization problem is not convex. It follows our proposal to combine in the same training a first order optimizer for the first step, and a second order optimizer when we are close to the optimum. This hypothesis is confirmed by our tests (accuracy on the test of HYBRID outperforms all the other methods) but a wider comparison using different datasets and architecture should be conducted.

REFERENCES

[1] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. W. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," *arXiv preprint arXiv:2006.00719*, 2020.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX

In this section we report for each optimizer two additional plots describing the evolution of the cross-entropy loss during the training and the evolution of the log-scaled spectral norm of the gradient of the loss with respect to the weights in the last fully connected layer of ResNet18 and the First Convolutional Layer (average among all the 64 filters).

Stochastic Gradient Descent:

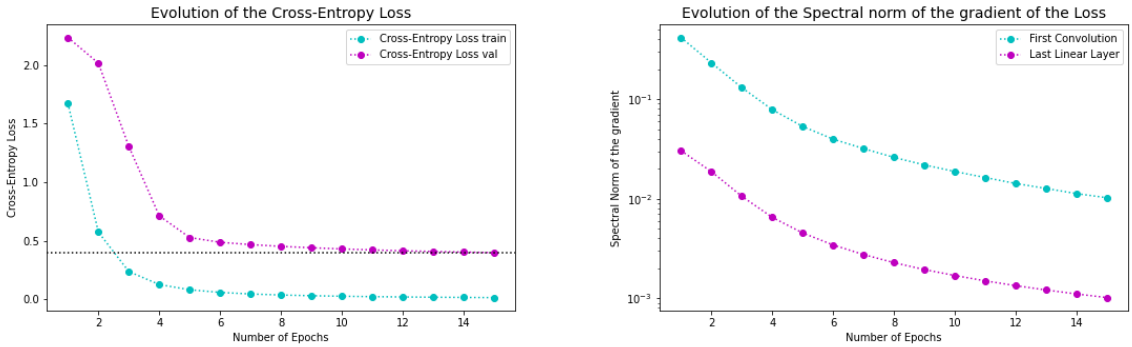

Figure 3: Evolution of the Training and Validation Cross-Entropy Loss (one the left) and of the log-scaled spectral norm of the first and last layer during the training using Stochastic Gradient Descent

Stochastic Gradient Descent with Momentum:

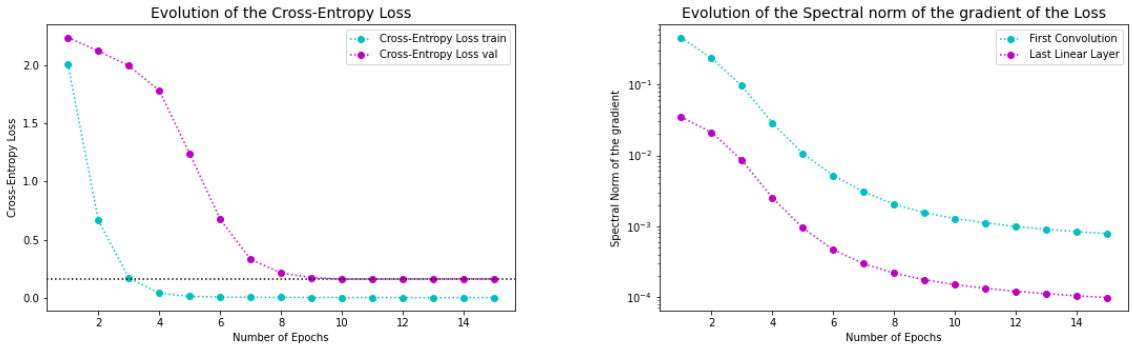

Figure 4: Evolution of the Training and Validation Cross-Entropy Loss (one the left) and of the log-scaled spectral norm of the first and last layer (one the right) during the training using Stochastic Gradient Descent with momentum

Adam:

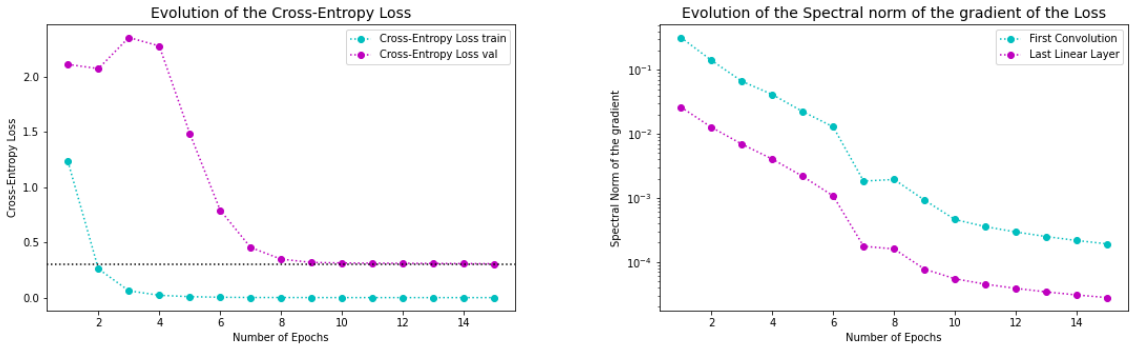

Figure 5: Evolution of the Training and Validation Cross-Entropy Loss (one the left) and of the log-scaled spectral norm of the first and last layer (one the right) during the training using Adam

AdaHessian:

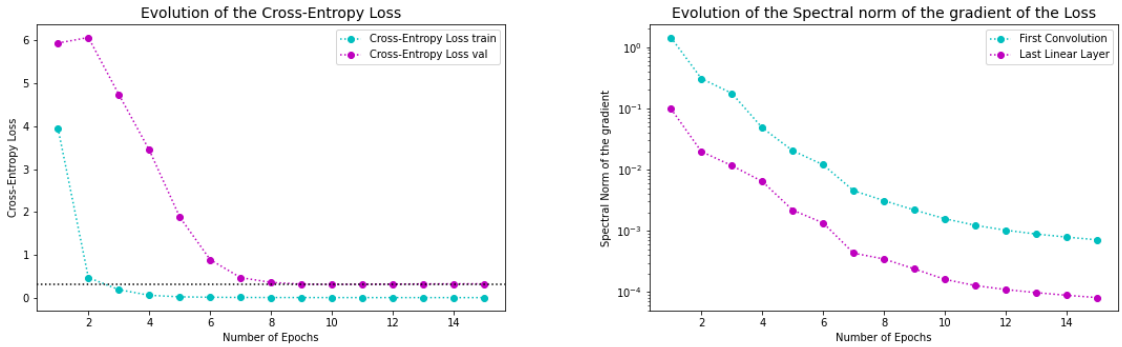

Figure 6: Evolution of the Training and Validation Cross-Entropy Loss (one the left) and of the log-scaled spectral norm of the first and last layer (one the right) during the training using AdaHessian

Hybrid:

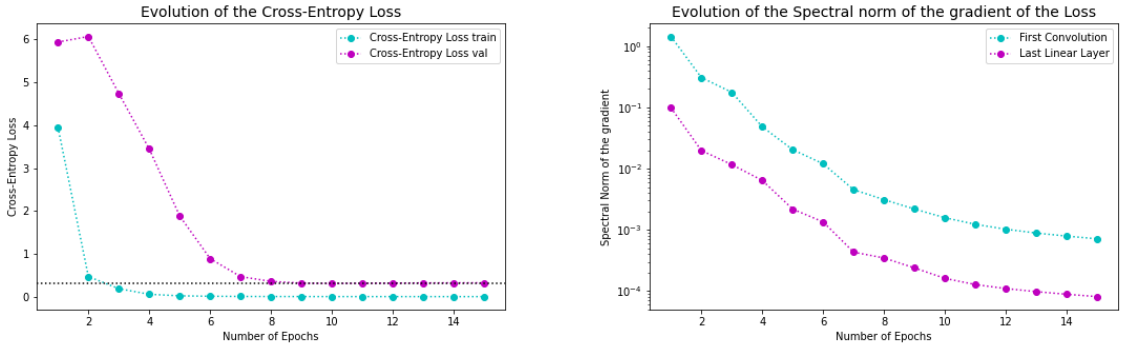

Figure 7: Evolution of the Training and Validation Cross-Entropy Loss (one the left) and of the log-scaled spectral norm of the first and last layer (one the right) during the training using HYBRID

We also report report in Figure 8 the evolution of the log-scaled spectral norm of the first and last layer during the training of all the methods together.

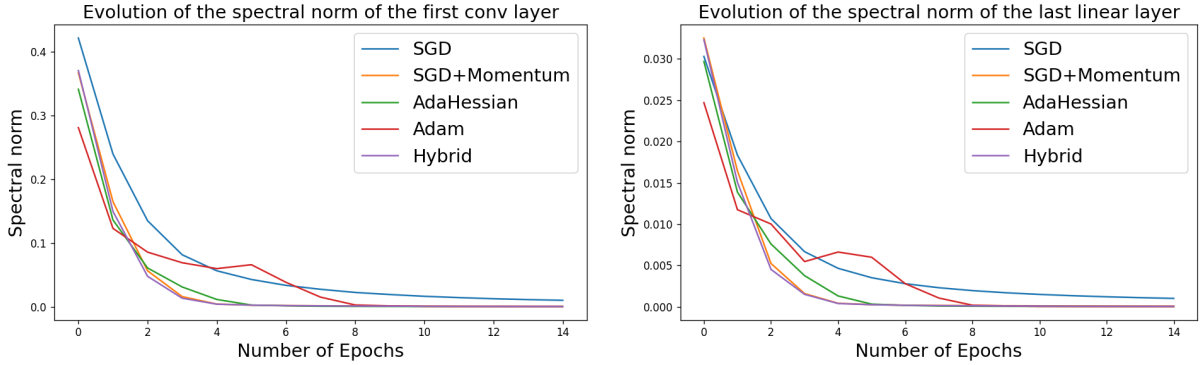

Figure 8: Evolution of the the spectral norm of gradient of the loss with respect to the weights of the first convolutional layer (on the left) and last liner layer during the training (one the right) of all the methods together.