Study of the curvature of the minimizers found by different optimizers and how it affects generalization

Mohammed Tarak Torgeman, Youssef Mehdi Attia, Hugo Manuel Serge Lacnfranchi
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—First order methods are the default optimization algorithms in Deep Learning; however, we were interested in studying the type of solution that other optimization methods may offer. For that we used a second order optimization method, and we compared how generalizable the different optimizers are, the shape of the local minima in some interesting directions , as well as some information on the Hessian (such as the eigenvalue density plot, the trace).**

I. Introduction

When training Neural Networks, engineers usually shy away from second-order methods; the deeper models could have in the millions of parameters, and calculating the needed Hessian matrix to use Newton's method is not feasible. We however were interested in seeing what these type of methods have to offer in terms of the type of solution we'd find, and used approximations of the Hessian to conduct our analysis using external libraries.
Furthermore, we would like to study the shape of the minima found by these different variants of the gradient descent. A flat solution is generally desirable because of its property that in a neighborhood of that kind of minima, the value of the loss function is almost constant and equal to the value found at that minima. They also are often indicative of a low complexity model, which should be less prone to overfitting. The function's value around a sharp solution, in contrast, varies quite a bit, and are generally considered undesirable. [1] specifically tries to efficiently find flat minimums for neural networks, while [2] argues that sharp solutions generalize less well, and presents ways to avoid them in the SGD algoritm by varying the batch sizes.
In contrast to both these works, [3] presents several results that point towards the fact that sharp minima can generalize well for deep networks, notably using Hessian information such as its trace and its spectral norm.
As such, we wanted to see the kind of solutions that different optimizers might converge to, and the problem we chose to illustrate this study is a computer visualization classification one, using the MNIST dataset of hand-drawn digits.

II. Model selection

In our project, we decided to use the LeNet architecture. This model answers multiple constraints we had concerning the predictive power and size of the network. We used the LeNet model described here, it is LeNet-5 whom consists of a mix of convolutional and pooling layers. This network is relatively low depth compared to most recent architectures, this allows for more computationally heavy operations on gradients on smaller hardware. Thanks to this model, we can visualize the different shapes on the loss function depending on the optimizer we chose, which will be an important topic in this project, another motivation for choosing a CNN such as LeNet is that we can simmulate an MLP using only convolutional layers [4], therefore our analysis may hold for MLP's and larger CNN's as well.

III. Different optimizers

In general, first order methods are used in Deep Neural Networks to train the models, the reason being that second order methods using the full Hessian (Newton's method) are both memory and computationally intensive. We were able to bypass that by using **AdaHessian**[5], which instead uses an approximation of the Hessian's diagonal.

A. First order

*1) **Stochastic gradient descent (SGD)**:* SGD is the basis optimization used when training neural networks; instead of using the full dataset to update its weight, it takes samples either one by one or in batches. This mitigates the memory footprint and the computation time it would take to go through all the data to update the gradient just once.
Calling γ the learning rate, x_n a randomly selected sample (the batch case follows trivially) and θ_t the weights at step t, SGD updates the weights as:

$$\theta_{t+1} = \theta_t - \gamma * \nabla f(\theta_t)$$

*2) **AdaGrad**:* AdaGrad extends SGD by adapting the learning rate to the parameters. It performs smaller updates for parameters associated with frequently occurring features. In its update rule, AdaGrad modifies the general learning rate γ at each time step for every parameter θ_i based on the past gradients. The update rule is as follows: Let $\beta_1, \beta_2, \gamma, \eta$ be the hyperparameters, the starting learning rate and the learning rate decay

$$\gamma_{t+1} = \frac{\gamma}{1 + (t-1)\eta}$$

$$v_{t+1} = v_t + \nabla f(\theta_{t+1})^2$$

$$\theta_{t+1} = \theta_t - \gamma_{t+1} \frac{\nabla f(\theta_{t+1})}{\sqrt{v_{t+1}} + \epsilon}$$

*3) **Adam**:* Adam works with momentums of first and second order. The intuition behind the algorithm is that we do not want to roll too fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. Adam also keeps an exponentially decaying average of past gradients The update rule for Adam is: Let $\beta_1, \beta_2, \epsilon, \gamma$ the hyperparameters and the learning rate

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \nabla f(\theta_i)}{1 - \beta_1^t}$$

$$v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_1^{t-i} \nabla f(\theta_t)_i \nabla f(\theta_t)_i}{(1-\beta_2^t)}}$$

$$\theta_{t+1} = \theta_t - \gamma \frac{m_t}{v_t + \epsilon}$$

B. *Second order: AdaHessian*

AdaHessian is a second order stochastic optimization algorithm, it is based on Newtons method but in order to be applicable to large models it estimates an approximation on the Hessian instead of computing it exactly. Therefore unlike the first order method were we locally approximate our function f by its first order Taylor's expansion we push our approximation to the second order. typically to find the step rule for the newtons method we perform the following.

$$f(x+h) \approx f(x) + \nabla f(x)^T h + h^T \nabla^2 f(x) h, \tag{1}$$

Now taking the derivative with respect to h and setting the expression to be equal to 0 we get:

$$\nabla^2 f(x) h = -\nabla f(x). \tag{2}$$

Now note this approach leads some issues: The hessian of an average model with 1 million of parameters will take more than 3 terabytes to store, using the conjugate gradient method to solve the system of equations might lead to issues if the matrix is ill conditioned which is to be expected with large matrices.

Therefore AdaHessian approximates the Hessian by its diagonal which it approximate using Hutchinson's method [5] to which they apply spatial averaging in order to smooth out spatial variations.

In order to avoid getting trapped in a local minima AdaHessian also uses the first and second moments. Therefore our update rule is : for $\mathbf{D}^{(\mathbf{s})}$ the spatial averaging of our estimation of the Hessian diagonal,Let$\beta_1, \beta_2, \epsilon, \gamma$ the hyperparameters and the learning rate, m_t is defined as in the section on Adam:

$$v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_1^{t-i} \mathbf{D}^{(\mathbf{s})}{}_i \mathbf{D}^{(\mathbf{s})}{}_i}{(1-\beta_2^t)}}$$

$$\theta_{t+1} = \theta_t - \gamma \frac{m_t}{v_t + \epsilon}$$

IV. Experiments and Results

A. *Comparing the best runs for each optimizers*

To begin this section, we start by tuning our LeNet-5 architecture, we seek the optimal parameters for this task described in 3. First, we investigate the loss depending on the learning rate of our model.

We observe that in 1, depending on the optimizer if we take a learning rate that is too small or too big our model will never go to a minimum, and even worse it will diverge leading to a huge loss.

Furthermore, we are going to observe the accuracy and loss of our model depending on the multiple optimizers we previously described.

We observe that in 2, depending on the optimizer we figure out that for each optimizer when the loss is minimized the accuracy is maximized.

B. *Eigenvalues/eigenvalues density plot*

In order to get an idea if the optimizers are close to a minima we decided to take a look at the eigenvalues of the Hessian as we know a minima has positive eigenvalues, as well as to get an idea on the flatness of the minima: if the Hessian matrix has small eigenvalues in terms of absolute value, it doesn't change much in the neighborhood of the solution. Therefore, to compute the eigenvalues of the Hessian we used PyHessian[6], which is a library for Hessian based analysis of neural network models. and it estimates the spectrum of the matrix using the Stochastic Lancoz algorithm.
In order to do so, PyHessian needs a part of the data to evaluate its

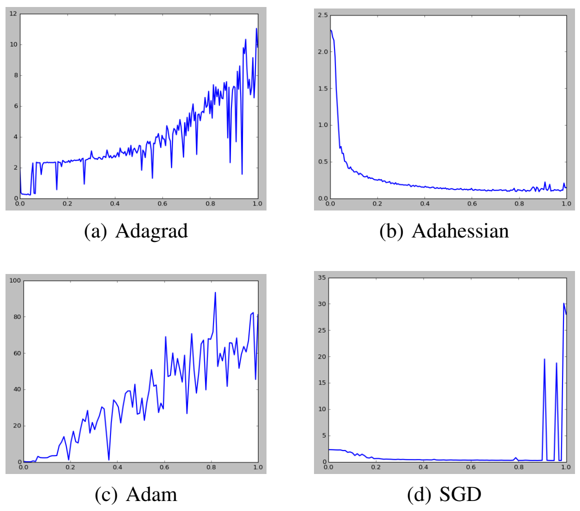
(a) Adagrad (b) Adahessian (c) Adam (d) SGD

Fig. 1: Plot of the loss of the model depending on the learning rate for the chosen multiple optimizers

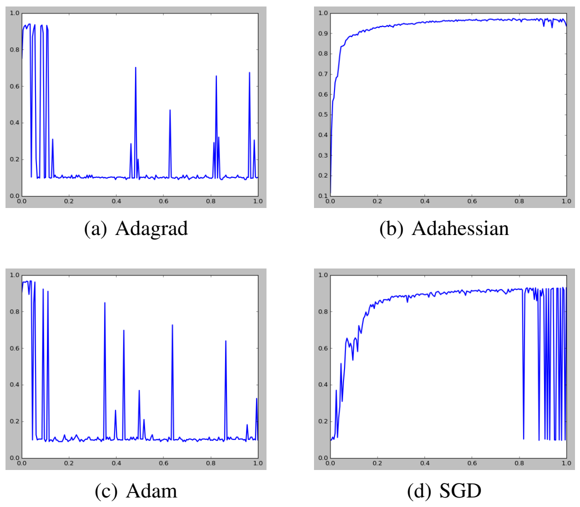
(a) Adagrad (b) Adahessian (c) Adam (d) SGD

Fig. 2: Plot of the accuracy of the model depending on the learning rate for the chosen multiple optimizers

Optimizer	Parameters	Accuracy
SGD	lr=0.1	99.03%
AdaGrad	lr=0.0012,lr decay=0	97.6%
Adam	lr=0.00472, betas=(0.675 ,0.999)	98.41%
AdaHessian	lr=0.1	98.89%

Fig. 3: Optimal parameters for the selected optimizers

approximations; we decided to pass it the whole MNIST dataset for every optimizer.

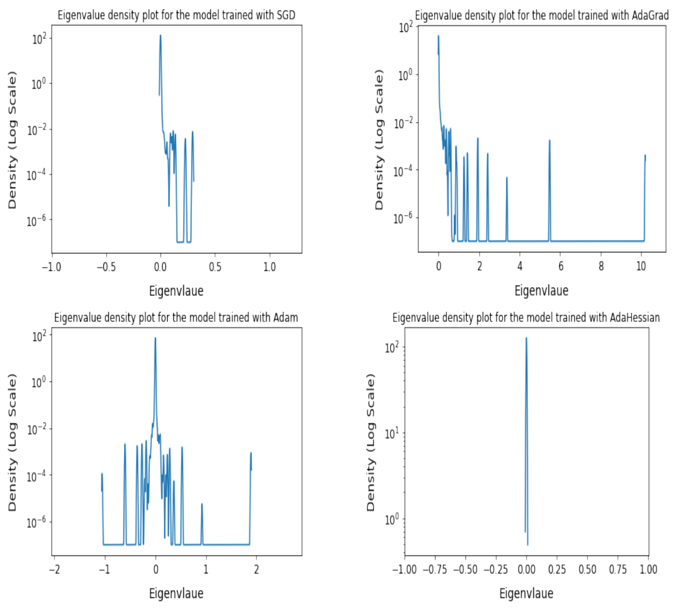

Fig. 4: The eigenvalue densities of the model's Hessian when trained with different optimizers

Firstly for SGD and AdaHessian the eigenvalues are fairly close to 0, especially for Adahessian who's eigevalues are almost all equal to 0 ; this seems to indicate that for both optimizers, the loss around the found solutions is (almost) flat in every direction. For Adagrad, the minima found has some large positive eigenvalues, which seems to indicate that the minima has some directions where the loss may change rapidly for small perturbation. Despite that, the eigenvalues are positive, so it seems to have reached a local minima. And finally Adam is quite hard to interpret, as despite having good performances, the norm of the gradient with respect to the parameters at the last iteration is not as negligible compared to the ones from the other optimizers. Combining that with the fact that the point found has positive and negative eigenvalues, we can see that Adam is not quite yet near a minima.

C. Checking the loss landscape f or each (ins the gradient's direction and in the direction of the top eigenvector)

In order to get a idea on the landscape of the loss we again used PyHessian. We were able to estimate the eigenvector associated with the largest eigenvalue in order to plot the loss in this direction. The reason is that when we take the second order Taylor approximation of our loss function in (1), if we are close to a local minima, then the gradient's norm is close to 0 therefore the approximation's behavior would be dominated by the Hessian term.

The results we got are visible on figure (5). It appears that among the different optimizers, AdaHessian and SGD gave us the flattest solutions, with the former being particularly flat. This correlates with the results we found with the eigenvalues: because their absolute values was close to 0, the loss doesn't change much in any direction, and we'd have a flatter landscape. This is really visible with AdaHessian, and considering how none of the first order methods gave us a flatness that is remotely close, this might be a property of second order methods.

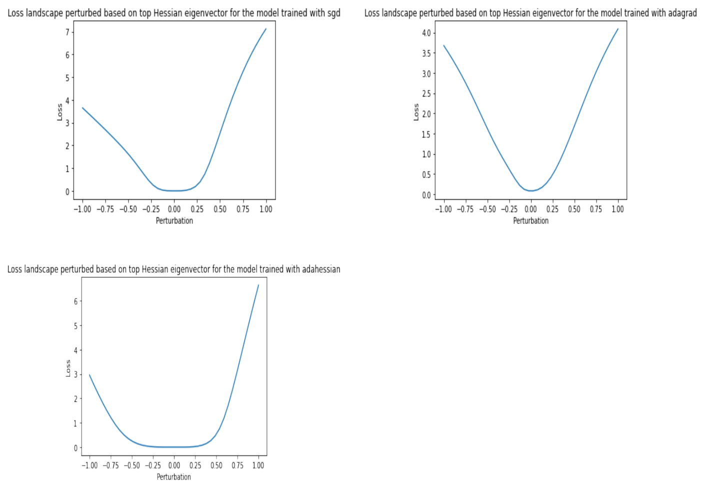

Fig. 5: 2D Loss landscape of most optimizers

Another interesting observation is the AdaGrad landscape; it appears to be the sharpest of all despite it generalizing well to unseen. This seems to go in line with the results found by [3], which is interesting because AdaGrad is not that different from SGD, and yet finds a sharper solution.

For Adam, we followed a different approach; seeing as the gradient norms are not close to 0, and seems to randomly oscillate (cf. Appendix below), we decided to plot it on two directions: the eigenvector corresponding to the highest eigenvalue, and the gradient direction (see figure (6)).

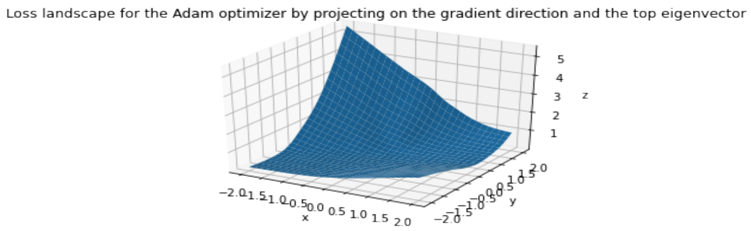

Fig. 6: 3D loss landscape for Adam

Looking at adam we can see that the landscape is curved and not exactly flat which shows that there's still some margin of improvement, despite that it outperforms AdaGrad, it is also slightly curved downward on the left.

V. Conclusion and discussion

Based on our analysis we can see that AdaHessian seems to achieve similar results as first order methods, however the minima that it found seems to be significantly flatter than the ones found by first order methods, as shown in the eigenvalue density plot as well as the landscape plot. However in this case we couldn't find a link between generalizability and the curvature around the minima found as both Adam and Adagrad are on steep curves, but in terms of performance they are not that far behind SGD and AdaHessian. Furthermore, AdaHessian is flatter than SGD but has a slightly lower accuracy.
It would be interesting to extend the work, and try the same experiment on deeper models such as AlexNet, or another type of model such as ResNets, and see if we get similar results.

References

[1] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural computation*, vol. 9, no. 1, pp. 1–42, 1997. [Online]. Available: https://direct.mit.edu/neco/article/9/1/1/6027/Flat-Minima

[2] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016. [Online]. Available: https://arxiv.org/abs/1609.04836

[3] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1019–1028. [Online]. Available: https://arxiv.org/abs/1703.04933

[4] F. Fleuret, "Lecture notes," 2022. [Online]. Available: https://fleuret.org/dlc/materials/dlc-slides-8-2-image-classification.pdf

[5] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," 2021. [Online]. Available: https://arxiv.org/abs/2006.00719

[6] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Pyhessian: Neural networks through the lens of the hessian," 2020. [Online]. Available: https://arxiv.org/abs/1912.07145

Appendix

Gradient norm during training

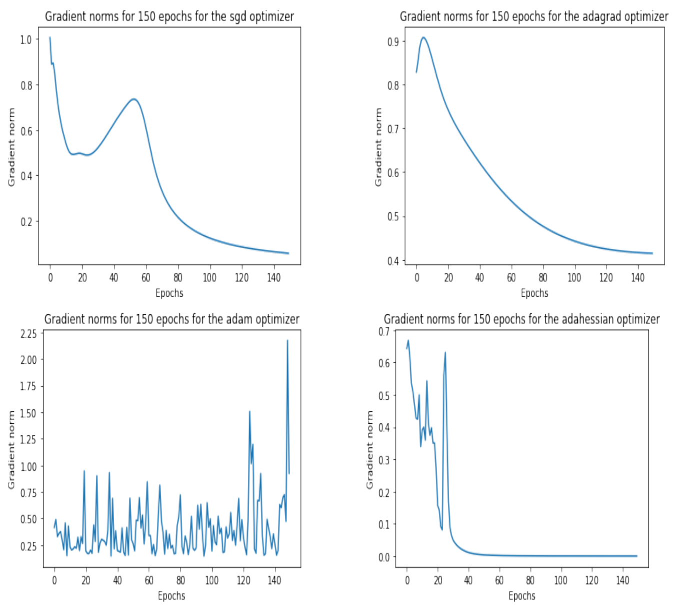

Fig. 7: The model's gradient norm during training for all optimizers

During training, we wished to check if when trained using different optimizers, the gradient with respect to the parameters' norm would always converge, preferably to 0, or at least decrease in a consistent fashion. Strangely enough, for Adam the gradient's norm seems to oscillate around a good solution (considering the good generalization it provides).

3D landscape for the other optimizers

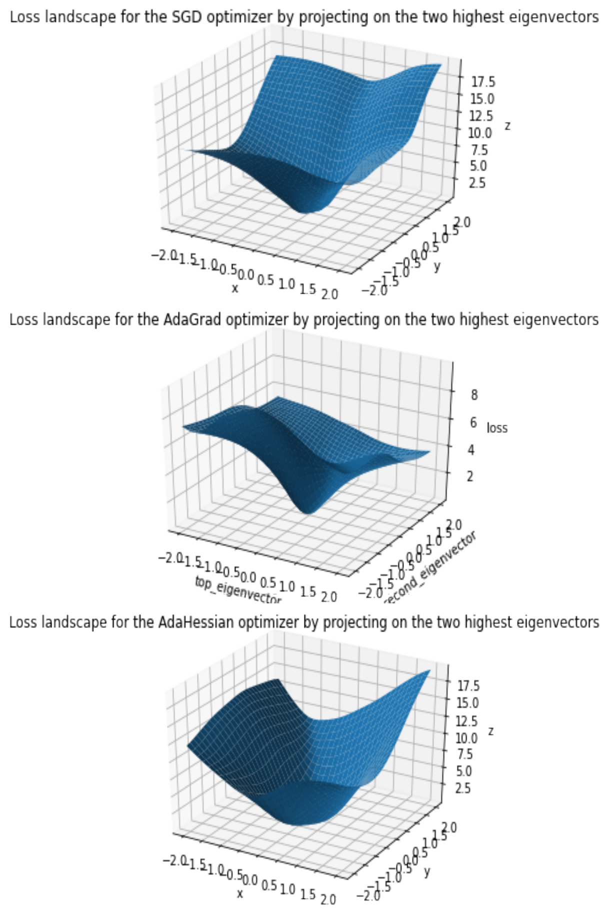

Fig. 8: 3D loss landscapes