Comparison between commonly used optimizers and analysis of possible exploitation of second order information at the end of training

Riccardo Brioschi, Giacomo Mossinelli, Havolli Albias

CS-439: Optimization for Machine Learning, Spring semester 2023

***Abstract*—In this paper, we empirically investigated how the usage of different optimizers affects the generalization properties of an architecture used for a classification task. First of all, we trained some models using four different optimizers, we evaluated their performances computing the accuracy on unseen data. After analysing how well these models generalize, we focused on the shape of the found local minima along specific directions and we tried to compute a stochastic estimate of the ratio between the two largest hessian eigenvalues in the last 100 iterations of each training process. We observed that, while it is not clear how and whether flatness of local minima relate to the generalization properties of a model, high ratio of the two largest Hessian matrix eigenvalues suggests the possibility of introducing second order information at the end of the training procedure even when using first-order methods.**

I. Introduction

Choosing the right optimizer to use when training a model is a crucial factor in determining the future performance of the architecture on unseen data. It is well known, both on a theoretical and empirical level, that different optimizers result in different learning trajectories, thus leading to distinct set of final parameters. The difference in the properties showed by these parameters is partially explained by the fact that each optimizer introduces a form of implicit regularization, exhibiting a preference for some solutions over the others. As an example, [1] perfectly illustrates how GD trajectories are repelled from regions where the gradient norm is large, while [1] and [2] explain how SGD seems to benefit from its inherent randomness. Moreover, the relationship between the generalization properties of a model and the role of implicit regularization, together with other factors such as specific choice of hyperparameters or usage of gradient clipping and early stopping techniques, have been recently investigated on a causal level in the paper [3] and it revealed to be a promising path for future works.

In light of the aforementioned results, we decided to investigate whether these theoretical differences in performance could be observed when training a small architecture for a classification task. For this purpose, we did not only compute the accuracy of the model on unseen data, but we also focused on the geometric shape of the local minima found after convergence. Flatness is an important aspect to consider when evaluating the goodness of a solution, since the loss is usually stable in the neighborhood of flat minima and therefore suggests that the final model has low complexity. Even though it is still not clear if flatness plays a key role in determining the generalization properties of a model [4]-[5], we decided to analyze it in order to better understand the differences among the chosen optimizers.

As a last step, we analyzed the values assumed by the ratio between the two largest hessian eigenvalues during the last few iterations of every training procedure. Such quantity is strongly related to the possibility of exploiting second order information while training, which might be beneficial to improve the quality of the final solution, since the exploration of the loss landscape would be driven by more precise information about the geometry of the current configuration. Even if this approach would increase the computational complexity of the method (the hessian matrix should be computed), large values assumed by this quantity would imply efficient algorithm (e.g. power method [6]) to identify both eigenvalues and eigenvectors of such matrix, thus making this interesting approach feasible.

Model and Data

To carry out our experiments, we used the MNIST dataset in order to perform a multiclass classification. Since most of the theoretical results are proven for small networks and because of our limited computational resources, we decided to use a simple LeNet architecture which combined convolutional and pooling layers. Given its small size and depth, this model well satisfies the restrictions regarding predictive power and complexity imposed by some of the results. Moreover, it allowed us to do more computationally expensive operations on gradients.

Optimizers and training models

In order to conduct our experiments, we selected four different optimizers. After that, we created a model and trained it with each of them. Table I shows the train and test accuracies achieved. These results were obtained using the optimal set of parameters as defined in the dedicated GitHub repository [7].

First order methods such as SGD, AdaGrad and ADAM are well known optimizers and very used to train deep neural networks: AdaGrad is an extension of SGD as it allows

	Train accuracy	Test accuracy
SGD	0.9989	0.9893
AdaGrad	0.9999	0.9837
ADAM	0.9996	0.9895
AdaHessian	0.9999	0.9910

TABLE I: Train and test accuracies for each optimizer

to choose an adapting, coordinate wise learning rate, while ADAM is a momentum variant of AdaGrad. On the other hand, second order methods that use the full Hessian are both memory and computationally intensive, thus being avoided in many scenarios, especially when training wide and deep networks. Despite this, given the usage of additional (Hessian) information, they usually provide better training trajectories and results. Therefore, to deal with all these aspects, we used AdaHessian optimizer, which instead of computing the full Hessian, computes only its diagonal [8].

Experiments and Results

A. Accuracy and in-class accuracy

As mentioned above, a model was trained for each of the optimizers. More specifically, a LeNet architecture [9] has been used on the classic MNIST dataset. Each of these models is capable of giving good results on this simple multiclass classification problem. However, it is worth focusing on the differences that can be inferred by analysing the situation more closely with the help of theory. Taking a look at table I, it is possible to notice the first differences in the train and test accuracy. Making an initial heuristic analysis, one can in fact observe the following. SGD performance is comparable to that of the other optimizers in our case, even if it is usually considered to be a simpler method. In fact, despite the wide usage of SGD, Adagrad [10] and ADAM [11] tend to be more effective than SGD in most cases [12], as they are able to more intelligently adapt the learning rate during training. More specifically, Adam tends to have a faster convergence rate by using adaptive estimates of gradient moments to update model weights. Moreover, it also reduces the need to manually tune the learning rate in general. A similar argument can be made for Adagrad, which automatically adjusts the learning rate for each parameter in the model, assigning a larger learning rate to parameters with smaller gradients and a smaller learning rate to parameters with larger gradients [13]. This allows Adagrad to handle gradient scaling problems more effectively than SGD.

In our specific case, SGD model has a slightly higher test accuracy than the one shown by AdaGrad, but the difference can be considered to be neglectable given the high accuracy of both methods. On the other hand, AdaHessian exhibits the best performance, which is probably due to the fact that capturing and exploiting second-order information on the curvature of the loss surface can offer potential advantages. Finally, if we focus on the in-class accuracy (see section A), all the models seem to perform well, without being biased towards some specific classes. This is another essential aspects that has to be considered when evaluating an optimization procedure, since skewed or non-balanced classes and data might strongly affect the generalization properties of the model.

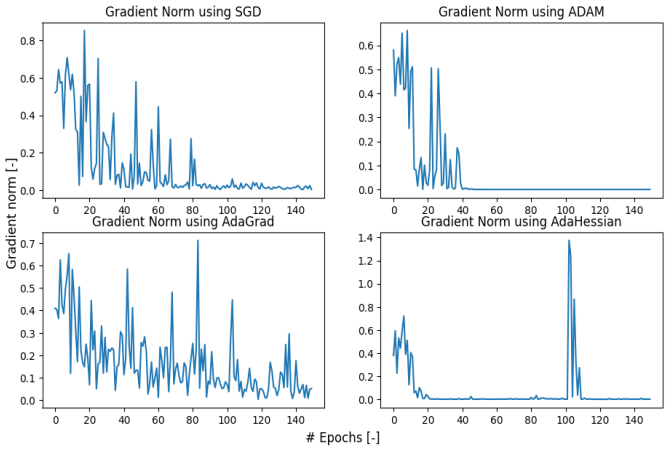

Fig. 1: Euclidean gradient norm for each optimizer computed at the end of each epoch.

B. Loss Landscape

As said before, we are also interested in comparing the geometry of the local minima returned by each optimizer. For this purpose, we reconstructed the loss surface in the neighborhood of the final set of parameters. Since we expect to move towards a local minimum (which implies the gradient norm is close to zero, as shown in figure [1]), we proceeded by plotting the second order approximation of the loss function along some specific directions (top hessian eigenvectors). As observed in figure [2], we notice that all optimizers converge to a final set of parameters for which the loss function seems to be robust to perturbations, i.e. the loss values do not explode even though we slightly perturb the model weights. If we then analyze the flatness of each solution, we observe that SGD and ADAM present a flatter landscape. Even though such behavior was expected when using SGD, AdaHessian shows better performance in terms of accuracy but a sharper landscape, thus making difficult to observe a strong relationship between flatness and good generalization properties.

This again enforces the idea for which, despite the intuition presented in the introduction (flatness implies less variance and better properties), this property alone is not sufficient to draw conclusions on the future performance of a model. Therefore, as showed in [3] and [5], a more careful analysis involving the network architecture and the used hyperparameters should be carried out to discover possible correlations and causal relations between flatness/sharpness and test error. A similar reasoning can be applied when observing plots at section B, where the loss has been reconstructed along the two highest hessian eigenvectors. Even when plotting the loss function along an additional dimension, the previous conclusions about sharpness and flatness of local minima hold, suggesting therefore the need of a more structured analysis.

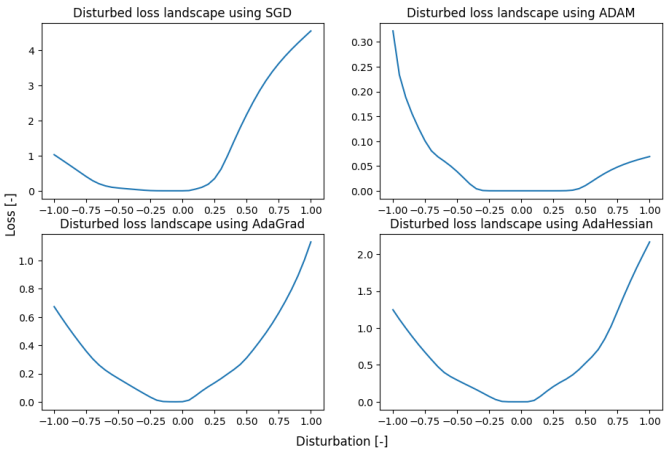

Fig. 2: Loss landscape in the direction of the top hessian eigenvector for each optimizer.

C. Spectral Gap and Eigenvalues Ratio

Given a matrix M, the spectral gap is defined as the difference between its two largest eigenvalues. This quantity is extremely important, since large values suggest that most of the information is contained along the dimension of the top eigenvector. Therefore, when performing dimensionality reduction or when projecting vectors onto subspaces, one could wisely choose to only consider the top hessian eigenspace, thus limiting the computational costs and managing to keep most of the information provided by the data. This intuition might be essential in order to improve the optimization procedure, even when using first order methods. As an example, calling V the projection matrix onto the top hessian eigenspace and $v = \nabla f$, one possible modification to the traditional gradient update rule might be obtained by using the following equation instead of v, thus exploiting second order information.

$$\tilde{v} = \lambda V v + (1 - \lambda)(I - V)v \tag{1}$$

Another quantity which is strongly related to the spectral gap is the ratio between the two largest eigenvalues: this value not only implicitly encodes how informative the top hessian eigenspace is, but it also describes how feasible retrieving the hessian eigenvectors is (it indeed appears in the bound of the power method for matrices [6]).

In this section, we therefore analyzed the behaviour of such ratio while training our architectures with different optimizers. More precisely, we saved and plot the values obtained during the last 100 iterations of each training procedure. The results are shown in figure [3]. In all these cases, we can observe that the ratio is not only positive (thus suggesting some weak form of local convexity towards the end of the procedure), but it also assumes large values on average. Therefore, the idea of adopting the update scheme in (1) might be feasible and could be investigated. Even though large values for the ratio ensure fast convergence rate when retrieving the eigenvectors (see Power Method [6]), the main disadvantage of adopting such technique relies on the need of computing the hessian matrix, which might involve computing millions of derivatives. The computation might be reduced by using a stochastic (un)biased estimation of the true hessian, as shown in [14], but the batch size plays a key role in the quality of the estimate. Because of limited computational resources, results in figure [3] are obtained using batch of size 100.

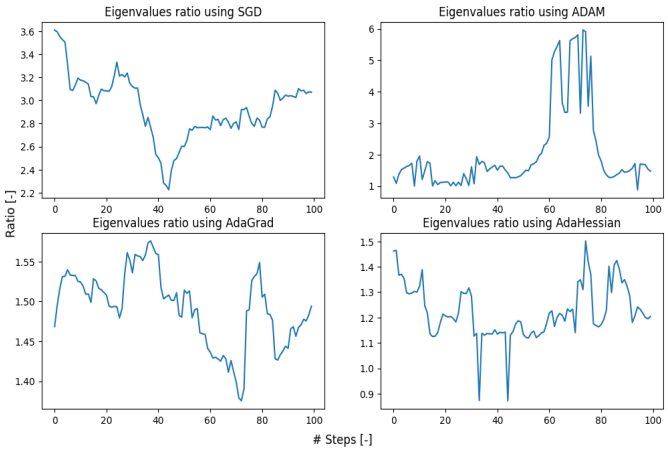

Fig. 3: Ratio of the two highest eigenvalues for the 100 last steps of the training procedure.

II. Conclusion and Further Work

The comparison between optimizers posed several challenges, thus making difficult to draw conclusion about how and whether the choice of different training procedures lead to significant differences in results on such an easy problem. In this project, four of the most popular optimizers were used and investigated (specifically, SGD, ADAM, AdaGrad and AdaHessian). From the analyses carried out, it became clear that the accuracy of an optimiser depends on a concurrence of several factors (e.g. chosen optimizer, hyperparameters), thus making difficult to independently identify the impact of each of them. While intuition would lead us to state that greater landscape flatness implies better properties, the results obtained in practice show that this fact is by no means certain and not yet generalizable. The analysis of the eigenvalues ratio show meaningful insights about the possibility of introducing hessian information when approaching local minima/locally convex regions. Given the high values in figure [3], computing the largest hessian eigenvalue seem to be feasible and therefore it would be interesting to further investigate the efficacy of the update schema proposed in [1], carefully evaluating the trade-off between increased computational costs and potential improvement in the solution.

As stated at the end of the previous section, the results obtained when computing the ratio might be inaccurate given the small batch complexity used to approximate both the true hessian matrix and its eigenvectors. As done in [14], the problem of stochastically estimate the hessian with a limited amount of samples can be explored in future works, in order to facilitate the adoption of second-order methods even when dealing with deep and wide neural networks.

REFERENCES

[1] S. J. Prince, *Understanding Deep Learning*. MIT Press, 2023. [Online]. Available: https://udlbook.github.io/udlbook/

[2] L. Wu, Z. Zhu, and W. E., "Towards understanding generalization of deep learning: Perspective of loss landscapes," 2017. [Online]. Available: https://arxiv.org/pdf/1706.10239.pdf

[3] Y. Jiang, H. M. Behnam Neyshabur, D. Krishnan, and S. Bengio, "Fantastic generalization measures and where to find them," 2019. [Online]. Available: https://arxiv.org/pdf/1912.02178.pdf

[4] L. Dinh, R. Pascanu, S. Bengio, and oshua Bengio, "Sharp minima can generalize for deep nets," 2017. [Online]. Available: https://arxiv.org/pdf/1703.04933.pdf

[5] S. Zhang, I. Reid, G. V. Pérez, and A. Louis, "why flatness does and does not correlate with generalization for deep neural networks," 2021. [Online]. Available: https://arxiv.org/pdf/2103.06219.pdf

[6] Wikipedia, "Power iteration method." [Online]. Available: https://en.wikipedia.org/wiki/Power_iteration

[7] B. Riccardo, M. Giacomo, and H. Albias, "Optimization for machine learning project." [Online]. Available: https://github.com/albias1996/OptML_project/blob/main/params.py

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

[9] L. Y., B. L., B. Y., and H. P., "Gradient-based learning applied to document recognition," pp. 2278–2324, 1998.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[12] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," 2020.

[13] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparametrization," 2018.

[14] H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma, "Sophia: A scalable stochastic second-order optimizer for language model pre-trainings," 2023. [Online]. Available: https://arxiv.org/pdf/2305.14342.pdf

APPENDIX

A. Heat maps

In this section, we present the heat map for each optimizer :

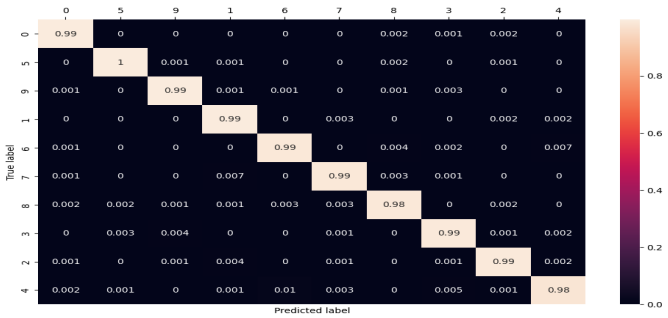

Fig. 4: Heat map using SGD optimizer

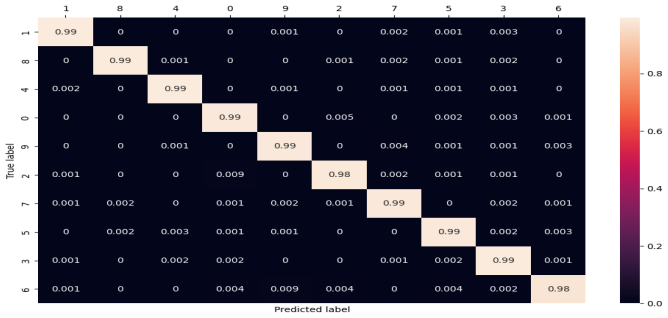

Fig. 5: Heat map using AdaGrad optimizer

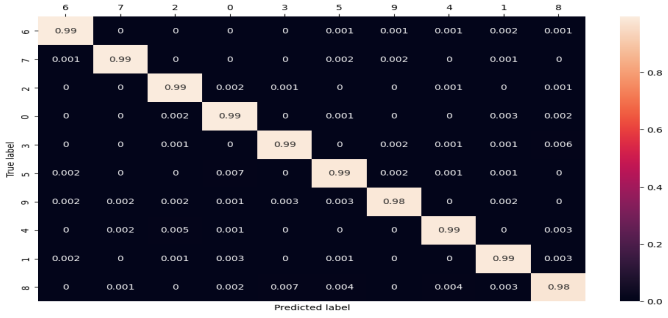

Fig. 6: Heat map using ADAM optimizer

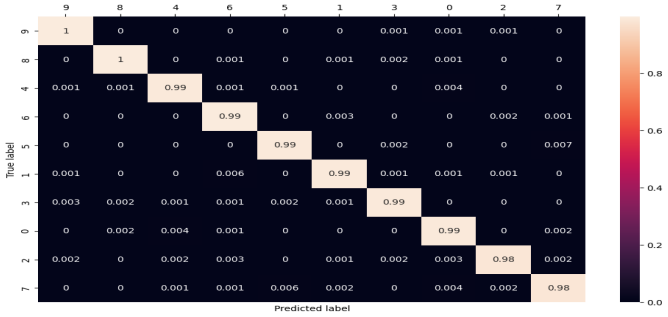

Fig. 7: Heat map using AdaHessian optimizer

B. 3D loss landscape

In this section, we show, for each optimizer, the 3D perturbed loss landscape along the two highest eigenvectors :

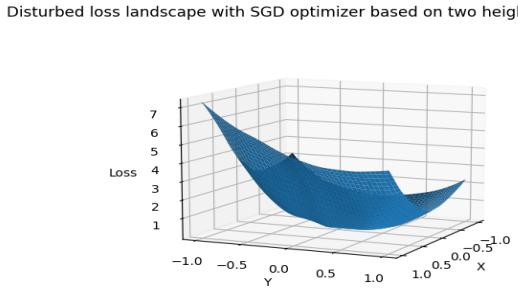

Fig. 8: 3D perturbed loss landscape with SGD optimizer

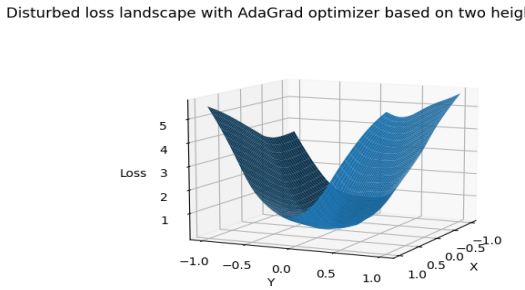

Fig. 9: 3D perturbed loss landscape with AdaGrad optimizer

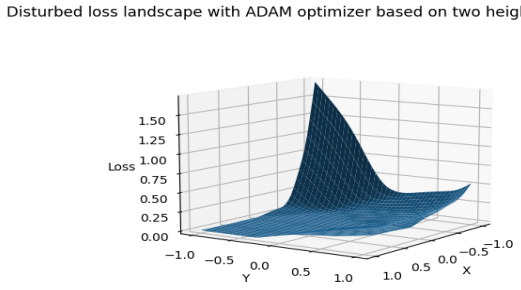

Fig. 10: 3D perturbed loss landscape with Adam optimizer

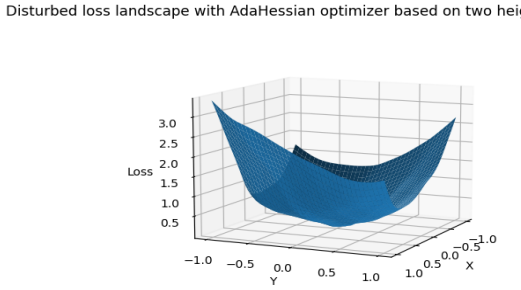

Fig. 11: 3D perturbed loss landscape with AdaHessian optimizer