Deep Learning Needs More Than Just the Gradient

Felix Dangel

Research statement

Abstract: Contemporary deep learning is powered by first-order methods that rely on the gradient. This is reflected in popular deep learning libraries which prioritize its computation. However, it narrows research to focus on variants of gradient descent.

To advance the field, we need to leverage information beyond the gradient (Figure 1).

My prior work demonstrates that **rich information beyond the gradient is affordable (Figure 2**). To reveal its under-unexplored potential, I want to use it to design novel methods and work on next-generation ML frameworks to push their efficient realization forward.

1 Problem statement

Deep learning relies on the gradient. Training neural networks is one of the most important and challenging, yet poorly understood deep learning tasks. We seek to minimize a non-convex empirical risk implied by data $\mathbb{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_n$ and a loss function ℓ over an extremely high-dimensional parameter space of a network $f_{\boldsymbol{\theta}}$ through noisy observations on a mini-batch $\mathbb{B} \sim \mathbb{D}$,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{B}|} \sum_{(\boldsymbol{x}_n, \boldsymbol{y}_n) \in \mathbb{B}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \boldsymbol{y}_n)\,. \qquad (1)$$

Current state-of-the-art deep learning optimizers use the mini-batch gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ which can be efficiently stored and automatically computed via reverse-mode automatic differentiation, aka backpropagation [37]. Popular deep learning libraries like PyTorch [32, 33] and TensorFlow [1] focus on efficiently computing the gradient via backpropagation. However, this has caused research to focus on gradient-based update rules and resulted in **more than 100 gradient descent variants** [39]. Large-scale comparisons find that

> *Despite efforts by the community, **there is currently no method that clearly dominates the competition**. [. . .] tuning helps about as much as trying other optimizers.* [39]

How can we enable novel research? I believe that **focusing on the gradient is not enough**: We need to incorporate more information to build better algorithms.

Higher-order information & challenges. Such quantities can be of statistical or geometrical nature (see Figure 1): Since a mini-batch gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ is the empirical mean of the distribution $\{\nabla_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta})\}_n$ of per-sample gradients, **higher-order statistical moments capture noise** of that distribution and the reliability of its mean. **Higher-order derivatives enable richer local approximations** of the loss landscape's geometry, for instance through curvature in form of the Hessian $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$. However, **efficiently computing with them is challenging**. Their explicit representations consume multiples of batch size or parameter dimension more memory

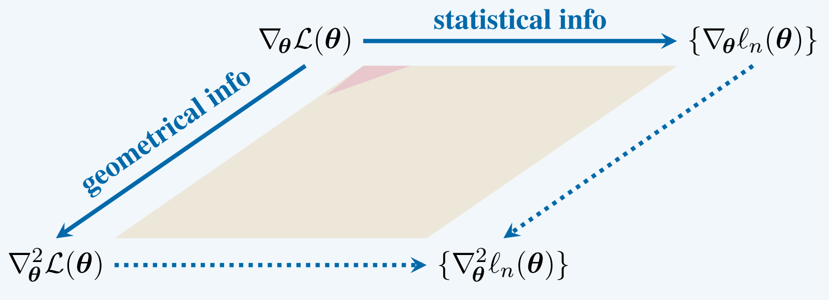
$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$
statistical info
$\{\nabla_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta})\}$
geometrical info
$\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$
$\{\nabla^2_{\boldsymbol{\theta}} \ell_n(\boldsymbol{\theta})\}$

Figure 1: Beyond the gradient. Higher-order statistical moments of per-sample quantities encode information about noise in the mini-batch. Second-order derivatives contain additional information about the loss landscape's geometry. Figure inspired by [7].

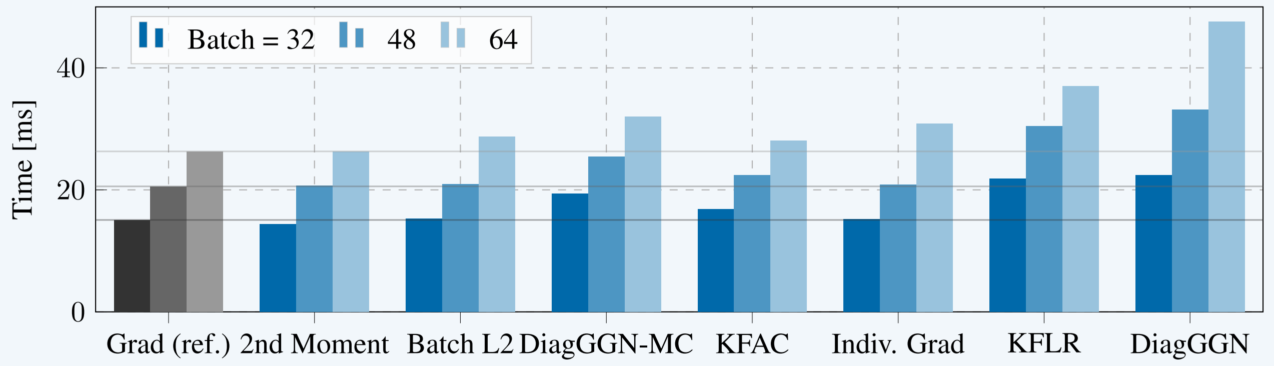

Figure 2: Rich information beyond the gradient is affordable. Run time comparison of computing (i) only the gradient (ref.) and (ii) the gradient with PyTorch *and* extensions with BackPACK. The architecture is a convolutional neural network (3C3D, CIFAR10) from DeepOBS [40]. **Most quantities add little overhead.**

compared to a gradient. Therefore, **practical methods rely on implicit schemes**, like matrix-free multiplication [35, 42], **and light-weight structured approximations**, like diagonal or Kronecker matrices [29]. But due to their more complicated nature, **efficient implementation is often burdensome and complex**.

Under-explored potential. This complicates experimentation and code sharing, e.g. for second-order methods in deep learning. Although these methods have been continuously investigated [4, 2, 28, 29, 48, 21, 17, 45], the **standard optimizers remain first-order methods, in stark contrast to other domains where second-order methods are the default** (convex optimization, generalized linear models). There may be scientific reasons why they do not work well in the deep learning setting. But one of the main hindrances to further explore their utility has been that **they are so complicated to implement that practitioners barely try them out.**[1]

2 Scientific achievements

During my PhD, I worked on easing experimentation with higher-order information and improving neural network training through monitoring. **I developed autodiff tools that efficiently extract rich information beyond the gradient by extending gradient backpropagation** [12, 13] in ML libraries, see Figure 2. This functionality helps to debug neural network training [41] and to compute with higher-order information in novel ways [14].

[1] For example, there are no fully-featured versions of the popular like Hessian-free [28] and K-FAC [29] optimizer for PyTorch [31].

2.1 Extending backpropagation to the Hessian

Gradient backpropagation is efficient, automated, and extensible. For sequential architectures, this carries over to second-order derivatives [12]. We can see this by considering the compute graph of such a layered model $f_{\boldsymbol{\theta}} = f^{(L)}_{\boldsymbol{\theta}^{(L)}} \circ \cdots \circ f^{(1)}_{\boldsymbol{\theta}^{(1)}}$ with parameters $\boldsymbol{\theta}^{(1,\dots,L)}$,

$$\begin{tikzcd}
 & \boldsymbol{\theta}^{(1)} \arrow[d, "f^{(1)}"'] & \boldsymbol{\theta}^{(2)} \arrow[d, "f^{(2)}"'] & \dots & \boldsymbol{\theta}^{(L)} \arrow[d, "f^{(L)}"'] & \boldsymbol{y}_n \arrow[d, "\ell"'] \\
\boldsymbol{z}_n^{(0)} \arrow[r] & \boldsymbol{z}_n^{(1)} \arrow[r] & \boldsymbol{z}_n^{(2)} \arrow[r] & \dots \arrow[r] & \boldsymbol{z}_n^{(L)} \arrow[r] & \ell_n
\end{tikzcd}$$

Inputs $\boldsymbol{x}_n = \boldsymbol{z}_n^{(0)}$ map to predictions $f_{\boldsymbol{\theta}}(\boldsymbol{x}_n) = \boldsymbol{z}_n^{(L)}$ via hidden features $\boldsymbol{z}_n^{(1,\dots,L-1)}$.
Backpropagation recovers the layer-wise gradients $\nabla_{\boldsymbol{\theta}^{(L,\dots,1)}}\ell_n$ by propagating $\nabla_{\boldsymbol{z}_n^{(L)}}\ell_n$ from the root to the leafs via vector-Jacobian products,

$$\nabla_{\bullet}\ell_n = \mathrm{J}_{\bullet}\boldsymbol{z}_n^{(l)\top}\nabla_{\boldsymbol{z}_n^{(l)}}\ell_n \qquad \begin{matrix} \bullet \in \{\boldsymbol{z}_n^{(l-1)}, \boldsymbol{\theta}^{(l)}\} \\ l = L,\dots,1 \end{matrix}$$

Hessian backpropagation recovers the per-layer Hessians $\nabla^2_{\boldsymbol{\theta}^{(L,\dots,1)}}\ell_n$, propagating the Hessian $\nabla^2_{\boldsymbol{z}_n^{(L)}}\ell_n$ via

$$\nabla^2_{\bullet}\ell_n = \mathrm{J}_{\bullet}\boldsymbol{z}_n^{(l)\top}\nabla^2_{\boldsymbol{z}_n^{(l)}}\ell_n\,\mathrm{J}_{\bullet}\boldsymbol{z}_n^{(l)} + \textstyle\sum_k \nabla^2_{\bullet} z^{(l)}_{n,k}\,\nabla_{\boldsymbol{z}_n^{(l)}}\ell_n\,.$$

It fully aligns the computation of local Hessians with gradients and **unifies the view on block-diagonal curvature approximations** like the block-diagonal generalized Gauss-Newton (GGN) [42], Fisher [2], and its Kronecker-factorized approximations [29, 19, 6, 11].

2.2 Packing more into backprop

The BackPACK [13] library provides efficient access to various deep learning quantities through implementing the insights on extended backpropagation for the Hessian on top of PyTorch. During a standard backward pass that computes the average gradient, it extracts

- per-sample gradients & gradient statistics, and
- approximate second-order derivatives in the form of diagonal and Kronecker-factorized curvature,

which often adds only little overhead (Figure 2). BackPACK easily integrates into existing code and simplifies experimentation with the above quantities: It **powers works on Bayesian neural networks [15, 23], out-of-distribution generalization [20, 36], and differential privacy [46], as well as my own research** (Sections 2.3 and 2.4). More than two years after its release, it is still actively used, with **roughly 400 downloads last week**.

2.3 Enabling a closer look into neural nets

Higher-order information as provided by BackPACK is valuable to guide the training of neural networks. **Common methods for real-time training** diagnostics, such as monitoring the loss, are limited because they **only indicate whether a model is training, but not why**. **Cockpit [41] enables a closer look into neural networks** during training. The live-monitoring tool visualizes established, recently proposed [27, 3, 9, 5, 44, 43, 24], and novel summary statistics that are efficiently computed by BackPACK. It **allows to identify common bugs in the machine learning pipeline**, such as improper data pre-processing or vanishing gradients, but also to guide learning rate selection, and to study implicit regularization [30, 18]. This **showcases the potential of higher-order information to assist practitioners** to better understand their model.

2.4 Novel ways to compute with curvature

BackPACK's **extended autdiff functionality also enables algorithmic advances to tackle limitations of existing curvature proxies [14]**: Diagonal or Kronecker-factorized curvatures are (i) not agnostic of noise in the mini-batch, (ii) strict approximations that do not become exact in any limit, and (iii) restricted to the block diagonal, ignoring curvature effects between layers.
This can be addressed **through the GGN's low-rank structure**, which allows for exact computation with the full—rather than block-diagonal—matrix, and principled approximations to reduce cost in exchange for accuracy. It also enables efficient computation of spectral properties, as well as **directional gradients and curvatures on a per-sample basis that quantify noise not only for the first, but also the second derivative**. Monitoring this noise through signal-to-noise-ratios helps understand its characteristics in deep learning [16] and to identify challenges for optimization and generalization from the interplay of noise and curvature [43].

3 Future research

Vision. Deep learning needs more than just the gradient. My goal is to **reveal the potential of higher-order information for building better training methods and push forward the development of ML autodiff** in next-generation frameworks like JAX [8] to efficiently realize them.

Noise-aware second-order methods. **Newton steps are powerful, but their stability is strongly affected by noise**—one corrupted step might undo all previous progress. Improving their stability is thus one key challenge to make them work in the mini-batch setting. To do that, however, we need to quantify noise in the mini-batch. But currently popular curvature proxies used in second-order methods are not noise-agnostic. Therefore, **we need curvature approximations that provide access to gradient and curvature noise**, for instance through per-sample information by leveraging the GGN's low-rank structure [14]. I want to use such information **to develop noise-aware stabilizing mechanisms** for Newton steps, like damping.

Optimizing run time & advancing autodiff for ML. In contrast to gradient-based methods, **run time performance of higher-order methods can still be significantly improved:** BackPACK outperforms naive implementations and achieves practical overheads, but relies on PyTorch's Python API and therefore regularly performs unnecessary operations. Recent advances in automatic differentiation, like **JAX [8] and functorch [22], allow for more efficient implementations.**
Even better performance can be achieved through linear algebra tricks, like properties of the Kronecker product [25] and matrix decompositions [e.g. 13, 14]. Recent work suggests that there is potential for improvement as a number of these optimizations are not yet re-

alized [38]. I want to **automate the optimization of operations in second-order methods by incorporating these common tricks for higher-order information** into JAX. This further reduces the overhead of second-order methods stemming from poor implementation, and makes these performance gains widely available to the ML community.

Closing remarks. Apart from my scientific goals, my connection to MLCommons benchmark experts [40, 39], who work on measuring algorithmic progress in neural network training, will be helpful to evaluate the developed methods, discuss interfaces to simplify their usage, and thereby make them more practical.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10, 2000.

[3] L. Balles, J. Romero, and P. Hennig. Coupling adaptive batch sizes with learning rates. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[4] S. Becker and Y. Lecun. Improving the convergence of back-propagation learning with second-order methods. 1989.

[5] R. Bollapragada, R. H. Byrd, and J. Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28, 2017.

[6] A. Botev, H. Ritter, and D. Barber. Practical Gauss-Newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, 2017.

[7] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review (SIREV)*, 60, 2016.

[8] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[9] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample size selection in optimization methods for machine learning. *Math. Program.*, 134, 2012.

[10] R. T. Q. Chen, D. Choi, L. Balles, D. Duvenaud, and P. Hennig. Self-tuning stochastic optimization with curvature-aware gradient filtering. *Advances in Neural Information Processing Systems (NeurIPS), Workshop I Can't Believe It's Not Better!*, 2020.

[11] S.-W. Chen, C.-N. Chou, and E. Chang. BDA-PCH: Block-diagonal approximation of positive-curvature Hessian for training neural networks. 2018.

[12] F. Dangel, S. Harmeling, and P. Hennig. Modular block-diagonal curvature approximations for feedforward architectures. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[13] F. Dangel, F. Kunstner, and P. Hennig. BackPACK: Packing more into backprop. In *International Conference on Learning Representations (ICLR)*, 2020.

[14] F. Dangel, L. Tatzel, and P. Hennig. ViViT: Curvature access through the generalized Gauss-Newton's low-rank structure. 2021.

[15] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux - effortless bayesian deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] F. Faghri, D. Duvenaud, D. J. Fleet, and J. Ba. A study of gradient variance in deep learning, 2020.

[17] M. Gargiani, A. Zanelli, M. Diehl, and F. Hutter. On the promise of the stochastic generalized Gauss-Newton method for training DNNs, 2020.

[18] B. Ginsburg. On regularization of gradient descent, layer imbalance and flat minima. 2020.

[19] R. Grosse and J. Martens. A kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning (ICML)*, 2016.

[20] I. Gulrajani and D. Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations (ICLR)*, 2021.

[21] J. F. Henriques, S. Ehrhardt, S. Albanie, and A. Vedaldi. Small steps and giant leaps: Minimal Newton solvers for deep learning, 2018.

[22] R. Z. Horace He. functorch: JAX-like composable function transforms for PyTorch, 2021.

[23] A. Immer, M. Korzepa, and M. Bauer. Improving predictions of Bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[24] J. Liu, Y. Bai, G. Jiang, T. Chen, and H. Wang. Understanding why neural networks generalize well through GSNR of parameters. In *International Conference on Learning Representations (ICLR)*, 2020.

[25] C. F. Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 2000.

[26] D. Maclaurin, D. Duvenaud, M. Johnson, and R. P. Adams. Autograd: Reverse-mode differentiation of native Python, 2015.

[27] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig. Early stopping without a validation set, 2017.

[28] J. Martens. Deep learning via Hessian-free optimization. In *International Conference on Machine Learning (ICML)*, 2010.

[29] J. Martens and R. Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, 2015.

[30] R. Mulayoff and T. Michaeli. Unique properties of flat minima in deep networks. In *International Conference on Machine Learning (ICML)*, 2020.

[31] M. Novik. torch-optimizer – collection of optimization algorithms for PyTorch, 2020.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Workshop on Autodiff*, 2017.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[34] A. Paszke, D. Johnson, D. Duvenaud, D. Vytiniotis, A. Radul, M. Johnson, J. Ragan-Kelley, and D. Maclaurin. Getting to the point. index sets and parallelism-preserving autodiff for pointful array programming. In *International Conference on Functional Programming*, 2021.

[35] B. A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6, 1994.

[36] A. Rame, C. Dancette, and M. Cord. Fishr: Invariant gradient variances for out-of-distribution generalization. In *International Conference on Machine Learning (ICML)*, 2022.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[38] A. Sankaran, N. A. Alashti, C. Psarras, and P. Bientinesi. Benchmarking the linear algebra awareness of TensorFlow and PyTorch, 2022.

[39] R. M. Schmidt, F. Schneider, and P. Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. In *International Conference on Machine Learning (ICML)*, 2021.

[40] F. Schneider, L. Balles, and P. Hennig. DeepOBS: A deep learning optimizer benchmark suite. In *International Conference on Learning Representations (ICLR)*, 2019.

[41] F. Schneider, F. Dangel, and P. Hennig. Cockpit: A practical debugging tool for the training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[42] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14, 2002.

[43] V. Thomas, F. Pedregosa, B. van Merriënboer, P.-A. Manzagol, Y. Bengio, and N. L. Roux. On the interplay between noise and curvature and its effect on optimization and generalization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[44] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney. PyHessian: Neural networks through the lens of the Hessian. In *IEEE International Conference on Big Data*, 2020.

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

[46] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in pytorch. In *Advances in Neural Information Processing Systems (NeurIPS), Workshop Privacy in Machine Learning*, 2021.

[47] G. Zhang, J. Martens, and R. B. Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[48] H. Zhang, C. Xiong, J. Bradbury, and R. Socher. Block-diagonal Hessian-free optimization for training neural networks, 2017.