ETH zürich

First Optimize, Then Discretize for SciML

Marius Zeinhofer

Workshop on Artificial Intelligence for Theoretical Sciences
at the Kavli Institute of Theoretical Sciences, Beijing, China

ETH Zürich, Seminar for Applied Mathematics,
Switzerland

November 14, 2024

Today's Talk

Part I – Function Space Optimization for SciML

Discretize function space algorithms in the tangent space of a neural network ansatz[1]

- Develop optimizers that provably mimic function space dynamics
- Design new algorithms and understand existing ones within this framework

Part II – Function Space Optimization at Scale

Scale the function space algorithms to millions of parameters[2]

- Leverage the Kronecker structure of linear layer Jacobians to approximate curvature matrices

[1] J. Müller and M. Zeinhofer (2024). "Position: Optimization in SciML Should Employ the Function Space Geometry". In: *Forty-first International Conference on Machine Learning*.

[2] F. Dangel, J. Müller, and M. Zeinhofer (2024). "Kronecker-Factored Approximate Curvature for Physics-Informed Neural Networks". In: *Advances in Neural Information Processing Systems*.

Table of Contents

Scientific Machine Learning

When referring to Scientific Machine Learning (SciML) we mean:

- Operator Learning[3]
- Variational Monte Carlo for Schrödinger's equation with NN ansatz[4]
- PINNs, Deep Ritz, etc.[5]
- Anything involving PDEs and neural networks

Observation

Introducing PDEs to objective functions complicates neural network training drastically.

[3] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar (2021). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations*.

[4] D. Pfau, J. S. Spencer, A. G. Matthews, and W. M. C. Foulkes (2020). "Ab Initio Solution of the Many-Electron Schrödinger Equation with Deep Neural Networks". In: *Physical Review Research* 2.3, p. 033429.

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis (2019). "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Optimization in SciML is a Challenge

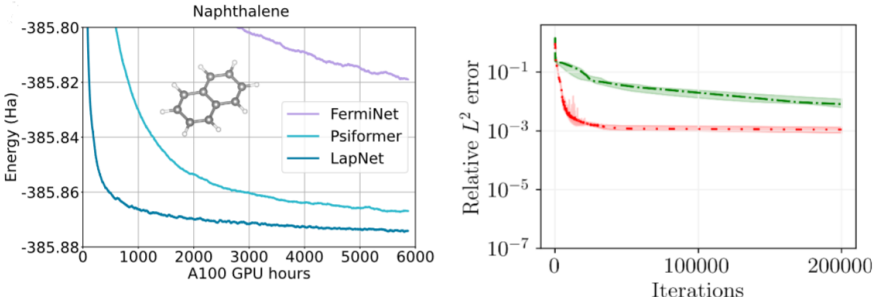


Figure: Illustration of neural network wavefunction optimization[6] and PINN optimization[7].

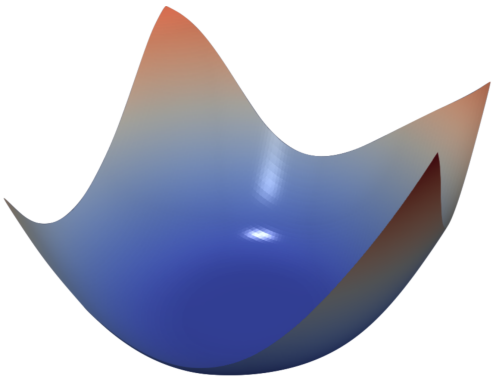
Proposition

Abandon first-order optimization. Exploit geometry in function space.

[6] R. Li et al. (2024). "A computational framework for neural network-based variational Monte Carlo with Forward Laplacian". In: *Nature Machine Intelligence*, pp. 1–11.

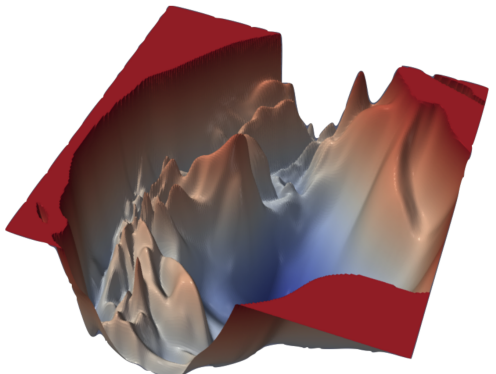
[7] J. Müller and M. Zeinhofer (2023). "Achieving High Accuracy with PINNs via Energy Natural Gradients". In: *International Conference on Machine Learning*.

Geometry in Function Space vs Paramter Space[8]



$u \mapsto E(u)$

"Function Space"



$\theta \mapsto L(\theta) = E(u_\theta)$

"Parameter Space"

[8] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein (2018). "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31.

Abstract Function Space Optimization[9]

Informal Roadmap: "First Optimize, Then Discretize"

Given: "a SciML problem" and a neural network ansatz. We propose:

(i) **Continuous Formulation:** Formulate minimization/saddle point/... problem in a Hilbert space $\mathcal{H}$

$$E : \mathcal{H} \to \mathbb{R}.$$

(ii) **Optimize:** Decide for an appropriate iterative algorithm *in function space* $\mathcal{H}$

$$u_{k+1} = u_k + d_k, \quad k = 0, 1, 2, \dots$$

(iii) **Discretize:** Project update directions on the tangent space of neural network ansatz.

[9] J. Müller and M. Zeinhofer (2024). "Position: Optimization in SciML Should Employ the Function Space Geometry". In: *Forty-first International Conference on Machine Learning*.

Recap

Newton's Method for Minimization in Function Spaces

For (general) objective $E : \mathcal{H} \to \mathbb{R}$ do second-order Taylor expansion (in function space)

$$E(u + d) \approx E(u) + DE(u)d + \frac{1}{2}D^2E(u)(d, d).$$

Explicitly minimize in the variable d:

$$d = u - D^2E(u)^{-1}[DE(u)]$$

- Think of the expression $D^2E(u)^{-1}[DE(u)]$ as solving a PDE[10].

[10] The Euler-Lagrange equations of the Taylor expansion of E.

Newton: Continuous Formulation

- Suppose we aim to solve Poisson's equation

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega \end{aligned}$$

- Reformulate as a minimization problem[11]

$$\min_{u \in H^2(\Omega)} E(u) = \frac{1}{2}\|\Delta u + f\|^2_{L^2(\Omega)} + \frac{1}{2}\|u - g\|^2_{L^2(\partial\Omega)}$$

- Ignore neural network ansatz classes.

[11] A formulation as a root finding problem or a variational formulation make sense, too.

Newton: Optimize

- For the quadratic minimization problem

$$u^* = \operatorname{argmin}_{u \in H^2(\Omega)} E(u) = \frac{1}{2}\|\Delta u + f\|^2_{L^2(\Omega)} + \frac{1}{2}\|u - g\|^2_{L^2(\partial\Omega)}$$

- Newton's method in function space has one step convergence

$$\begin{aligned} u^* &= u - D^2E(u)^{-1}[DE(u)] \\ &= u + \underbrace{(u^* - u)}_{=d}. \end{aligned}$$

Goal

Discretize such that we follow the optimal step $d = u^* - u$?

Newton: Discretize I – Bilinear Forms

Question

How to discretize the Newton step $D^2E(u)^{-1}[DE(u)]$?

- $D^2E(u) : H^2(\Omega) \times H^2(\Omega) \to \mathbb{R}$ is a *bilinear form*:

$$D^2E(u)(v, w) = \int_\Omega \Delta v \Delta w \, \mathrm{d}x + \int_{\partial\Omega} vw \mathrm{d}s$$

- $DE(u) : H^2(\Omega) \to \mathbb{R}$ is a *linear functional*:

$$DE(u)(v) = \int_\Omega (\Delta u + f)\Delta v \, \mathrm{d}x + \int_{\partial\Omega} (u - g)v \, \mathrm{d}s$$

- The expression $D^2E(u)^{-1}(DE(u))$ means: Find v such that

$$D^2E(u)(v, w) = DE(u)(w), \quad \text{for all} \quad w \in H^2(\Omega)$$

Newton: Discretize II – Galerkin Methods

Question

How to discretize an equation of the form: $D^2E(u)(v,w) = DE(u)(w), \quad$ for all $\quad w$?

- Choose basis functions $v_1, \ldots, v_n \in H^2(\Omega)$ and transfer to a matrix equation:

$$G_{ij} = D^2E(u)(v_j, v_i), \quad b_i = DE(u)(v_i)$$

- Solve the matrix equation

$$G\alpha = b$$

- Under suitable conditions, $v = \sum_{i=1}^{n} \alpha_i v_i$ is an approximate solution.

Naming convention

Discretizing a bilinear form with a finite dimensional vector space is a *Galerkin Method*.

Newton: Discretize III – Galerkin in Tangent Space

- Given a neural network ansatz $\{u_\theta \mid \theta \in \Theta\}$, as basis function choose

$$\partial_{\theta_1} u_\theta, \ldots, \partial_{\theta_p} u_\theta$$

- Discretize with a Galerkin ansatz using the basis functions $\partial_{\theta_1} u_\theta, \ldots, \partial_{\theta_p} u_\theta$:

$$G(\theta)_{ij} = D^2 E(u_\theta)(\partial_{\theta_j} u_\theta, \partial_{\theta_i} u_\theta), \qquad \nabla L(\theta)_i = DE(u_\theta)(\partial_{\theta_i} u_\theta)$$

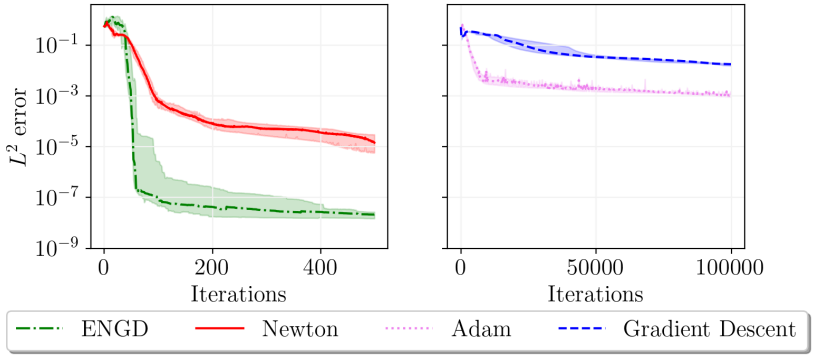
- This yields an optimization algorithm:

$$\theta_{k+1} = \theta_k - \eta_k G(\theta_k)^\dagger \nabla L(\theta_k).$$

- If $E(u) = \frac{1}{2}\|\Delta u + f\|^2_{L^2(\Omega)} + \frac{1}{2}\|u - g\|^2_{L^2(\partial\Omega)}$ we get:

$$G(\theta)_{ij} = \int_\Omega \Delta\partial_{\theta_i} u_\theta \Delta\partial_{\theta_j} u_\theta \,\mathrm{d}x + \int_{\partial\Omega} \partial_{\theta_i} u_\theta \partial_{\theta_j} u_\theta \mathrm{d}s$$

This Yields a Highly Efficient Method[12]



$$\theta_{k+1} = \theta_k - \eta_k G(\theta_k)^\dagger \nabla L(\theta_k), \quad k = 0, 1, 2 \dots$$

$$G(\theta)_{ij} = \int_\Omega \Delta \partial_{\theta_i} u_\theta \Delta \partial_{\theta_j} u_\theta \, \mathrm{d}x + \int_{\partial\Omega} \partial_{\theta_i} u_\theta \partial_{\theta_j} u_\theta \mathrm{d}s$$

[12] J. Müller and M. Zeinhofer (2023). "Achieving High Accuracy with PINNs via Energy Natural Gradients". In: *International Conference on Machine Learning*.

L^2 Gradient Flow for Rayleigh quotient: Cont. Form

- Suppose we want to minimize the Rayleigh quotient

$$R(\psi) = \frac{\langle H\psi, \psi \rangle}{\langle \psi, \psi \rangle},$$

  where H is the Hamiltonian (or any symmetric PDE operator).
- Rewrite the problem using $\tilde{\psi} = \psi / \|\psi\|_{L^2}$ as

$$\tilde{R}(\tilde{\psi}) = \langle H\tilde{\psi}, \tilde{\psi} \rangle$$

- The transformation $\psi / \|\psi\|_{L^2}$ is for mathematical convenience.

L^2 Gradient Flow for Rayleigh quotient: Optimize

- Do L^2 gradient flow with explicit Euler discretization on $\tilde{R}$

$$\tilde{\psi}_{k+1} = \tilde{\psi}_K - \eta_k T^{-1}(D\tilde{R}(\tilde{\psi}_k)),$$

- $T : L^2 \to (L^2)^*$ is the Riesz isometry[13] of L^2

$$T(\psi)(\phi) = \langle \psi, \phi \rangle$$

- $D\tilde{R}(\tilde{\psi})\phi = \langle H\tilde{\psi}, \phi \rangle$ is the Fréchet (functional) derivative of $\tilde{R}$.

[13] Can be viewed & discretized as bilinear form

L^2 Gradient Flow for Rayleigh quotient: Discretize

- The neural network ansatz and tangent space/Galerkin space are

$$\{\psi_\theta/\|\psi_\theta\|_{L^2} \mid \theta \in \Theta\}, \quad \operatorname{span}\{\partial_{\theta_1}\left(\frac{\psi_\theta}{\|\psi_\theta\|}\right), \ldots, \partial_{\theta_P}\left(\frac{\psi_\theta}{\|\psi_\theta\|}\right)\}$$

- Discretizing the Riesz map T yields

$$F_{ij} = T(\partial_{\theta_i}\left(\psi_\theta/\|\psi_\theta\|\right))(\partial_{\theta_j}\left(\psi_\theta/\|\psi_\theta\|\right)) = \langle \partial_{\theta_i}\left(\psi_\theta/\|\psi_\theta\|\right), \partial_{\theta_j}\left(\psi_\theta/\|\psi_\theta\|\right)\rangle$$

- Discretizing the functional derivative $D\tilde{R}$ yields the gradient of the loss $L(\theta) = R(\psi_\theta)$
- The final algorithm is $\theta_{k+1} = \theta_k - \eta_k F^\dagger \nabla L(\theta_k)$

Fact

One can compute that this is *stochastic reconfiguration*.

Translation: Function Space to Parameter Space

We can carry out the previous reasoning for different function space optimization methods[14]

Function Space	Parameter Space	Name in Literature
Newton	Generalized Gauss-Newton	ENGD[15]
Gradient Descent	Natural Gradient Descent[16]	—
Lagrange-Newton	Competitve Gradient Descent[17]	CPINNs[18]
Gauss-Newton	Gauss-Newton	GNNGD[19]

[14] J. Müller and M. Zeinhofer (2024). "Position: Optimization in SciML Should Employ the Function Space Geometry". In: *Forty-first International Conference on Machine Learning*.

[15] J. Müller and M. Zeinhofer (2023). "Achieving High Accuracy with PINNs via Energy Natural Gradients". In: *International Conference on Machine Learning*.

[16] S.-I. Amari (1998). "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2, pp. 251–276.

[17] F. Schäfer and A. Anandkumar (2019). "Competitive gradient descent". In: *Advances in Neural Information Processing Systems* 32.

[18] Q. Zeng, S. H. Bryngelson, and F. T. Schaefer (2023). "Competitive Physics Informed Networks". In: *International Conference on Learning Representations*.

[19] A. Jnini, F. Vella, and M. Zeinhofer (2024). "Gauss-Newton Natural Gradient Descent for Physics-Informed Computational Fluid Dynamics". In: *arXiv preprint arXiv:2402.10680*.

A General Class of Minimization Algorithms

- Many optimization methods are of the form:

$$u_{k+1} = u_k + T_{u_k}^{-1}[DE(u_k)]$$

  where T_k is a linear, continuous, bijective map $T_{u_k} : \mathcal{H} \to \mathcal{H}^*$.
- Gradient Descent, Newton, Gauss-Newton, Lagrange-Newton, ...
- Assumption: T_{u_k} is coercive, i.e.,

$$\langle T_{u_k} v, v \rangle \geq \alpha \|v\|_{\mathcal{H}}^2$$

  for some $\alpha > 0$.

Projection Theorem

Theorem

For the discretized algorithm $\theta_{k+1} = \theta_k - \eta_k G(\theta_k)^\dagger \nabla L(\theta_k)$ *it holds*

$$u_{\theta_{k+1}} = u_{\theta_k} - \eta_k \Pi_{u_{\theta_k}} \underbrace{[T^{-1}_{u_{\theta_k}}(DE(u_{\theta_k}))]}_{=d_k} + \epsilon_k$$

where Π_u *denotes the Galerkin projection onto the tangent space. The term* ϵ_k *corresponds to an error vanishing quadratically in the step and step size length*

$$\epsilon_k = O(\eta_k^2 \|G(\theta_k)^\dagger \nabla L(\theta_k)\|^2).$$

Proof.

Taylor expansion & Céa's Lemma. □

Visualization of the Projection Theorem

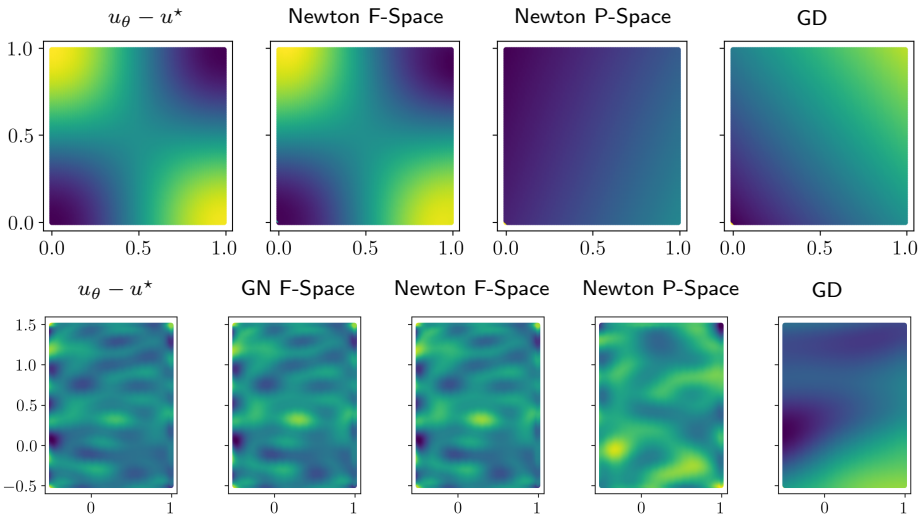


Table of Contents

The Price of Non-local Ansatz Functions

Recall the Structure of the ENGD Matrix

In every step of optimization the following matrix needs to be assembled and inverted

$$G(\theta)_{ij} = D^2 E(u_\theta)(\partial_{\theta_i} u_\theta, \partial_{\theta_j} u_\theta), \quad i, j = 1, \dots, p.$$

For neural network ansatz functions the matrix is dense, rank deficient and ill-conditioned.

- A direct solve works well for small networks.
- Naive application is doomed to fail, already for networks of modest size, say $p > 10^4$ trainable parameters.

What are our Options?

- Matrix-free solution methods for the system $G(\theta)d = \nabla L(\theta)$.
  - The matrix vector-product $G(\theta)v$ can be computed at comparable cost to the gradient $\nabla L(\theta)$[20, 21], hence use CG.
  - Drawback: CG iterates long due to ill-conditioning of $G(\theta)$.
- Derive cheap-to-invert approximations.
  - Derivatives of linear network layers yield Kronecker structure[22]. Leverage to build an approximation of $G(\theta)$!
  - The details fill the remaining talk.
- Combine both ideas, i.e., use preconditioned CG.

[20] N. N. Schraudolph (2002). "Fast curvature matrix-vector products for second-order gradient descent". In: *Neural computation* 14.7, pp. 1723–1738.

[21] A. Jnini, F. Vella, and M. Zeinhofer (2024). "Gauss-Newton Natural Gradient Descent for Physics-Informed Computational Fluid Dynamics". In: *arXiv preprint arXiv:2402.10680*.

[22] J. Martens (2020). "New Insights and Perspectives on the Natural Gradient Method". In: *The Journal of Machine Learning Research* 21.1, pp. 5776–5851.

Kronecker-Factor Approximation

- We discard cross-interactions between layers and use a Kronecker Approximation[23]:

$$\begin{aligned} G(\theta) &\approx \operatorname{diag}(G^{(1)}, \ldots, G^{(L)}) \\ &\approx \operatorname{diag}(A^{(1)} \otimes B^{(1)}, \ldots, A^{(L)} \otimes B^{(L)}). \end{aligned}$$

- Use the property of Kronecker product

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$$

- Ignoring bias terms, for layer l mapping from $\mathbb{R}^n \to \mathbb{R}^m$ the matrix $G^{(l)}(\theta)$ is of size (nm, nm) but the Kronecker factors are of size

$$A^{(l)} \in \mathbb{R}^{n \times n}, \quad B^{(l)} \in \mathbb{R}^{m \times m}$$

[23] We omit the θ dependency for brevity only.

Where does the Kronecker Structure Come From?

Structure of ENGD Matrix

Ignoring boundary terms, one can show that the ENGD matrix for Poisson is

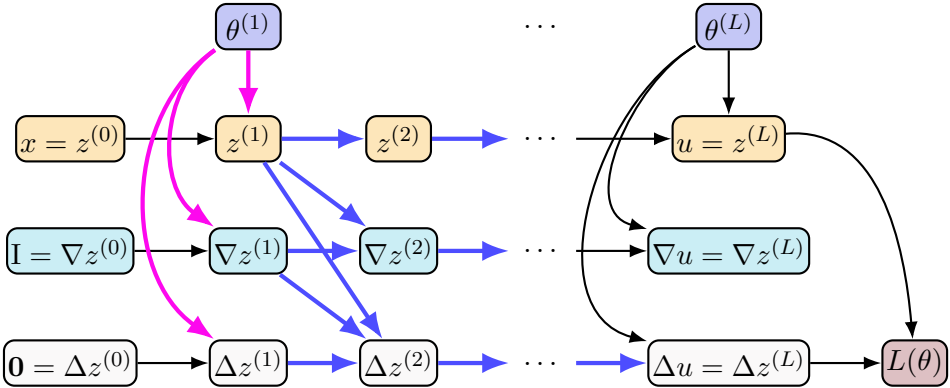
$$G^{(l)}(\theta) = \frac{1}{N_\Omega} \sum_{n=1}^{N_\Omega} \mathrm{J}_{\theta^{(l)}} \Delta u_\theta(x_n)^\top \, \mathrm{J}_{\theta^{(l)}} \Delta u_\theta(x_n)$$

- Key for Kronecker factors: Jacobian of $\theta \mapsto \theta x$, where θ is weight matrix

$$\mathrm{J}_\theta(\theta x) = x^\top \otimes \mathrm{Id}$$

- Need: children of $\theta^{(l)}$ in computational graph of $\theta^{(l)} \mapsto \Delta u_\theta(x_n)$.

Taylor Mode alias Forward Laplacian[24] Graph



$$\theta^{(l)} \mapsto \Delta u_\theta(x) = [(z^{(l)}, \nabla z^{(l)}, \Delta z^{(l)}) \mapsto \Delta u_\theta(x)] \circ [\theta^{(l)} \mapsto (z^{(l)}, \nabla z^{(l)}, \Delta z^{(l)})]$$

[24] R. Li et al. (2024). "A computational framework for neural network-based variational Monte Carlo with Forward Laplacian". In: *Nature Machine Intelligence*, pp. 1–11.

The Full K-FAC Approximation

Exact block of the ENGD matrix of a Laplace operator

$$G_\Omega^{(l)}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{s=1}^{S} \sum_{s'=1}^{S} Z_{n,s}^{(l-1)} Z_{n,s'}^{(l-1)\top} \otimes g_{n,s}^{(l)} g_{n,s'}^{(l)\top}.$$

K-FAC for the ENGD matrix of a Laplace operator

$$G_\Omega^{(l)}(\theta) \approx \left(\frac{1}{NS} \sum_{n,s=1}^{N,S} Z_{n,s}^{(l-1)} Z_{n,s}^{(l-1)\top} \right) \otimes \left(\frac{1}{N} \sum_{n,s=1}^{N,S} g_{n,s}^{(l)} g_{n,s}^{(l)\top} \right) =: A_\Omega^{(l)} \otimes B_\Omega^{(l)}$$

$$Z_{n,1}^{(l)} \coloneqq z_n^{(l)}, \quad Z_{n,2}^{(l)} \coloneqq \partial_{x_1} z_n^{(l)}, \dots, Z_{n,1+d}^{(l)} = \partial_{x_d} z_n^{(l)}, \quad Z_{n,2+d}^{(l)} \coloneqq \Delta z_n^{(l)}$$

$$g_{n,s}^{(l)} \coloneqq \mathrm{J}_{Z_{n,s}^{(l)}} \Delta u_n$$

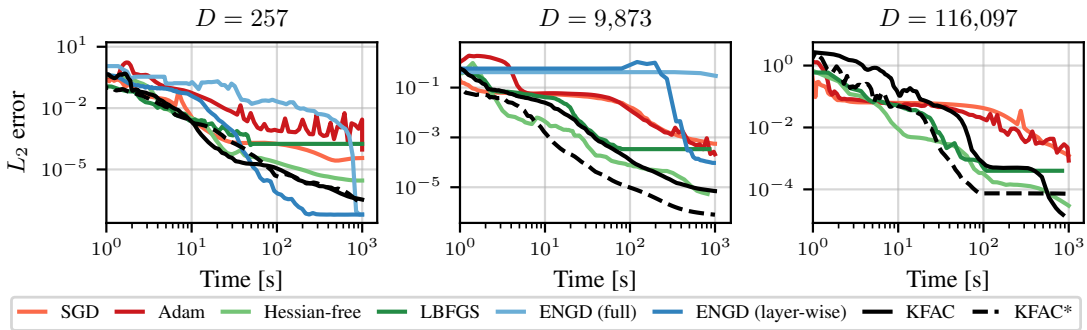
Further Details & Generalizations

- The K-FAC approximation works for more than just the Poisson equation:
  - general nonlinear PDEs using PINN formulation
  - Variational formulations (deep Ritz)
  - Neural Operators? Requires more work.
- Typically combined with
  - Trust-region methods for step-size choice
  - Exponential moving average for K-FAC-matrices
  - Infrequent updates of the K-FAC-matrices
  - Momentum on the natural gradient
- Find all details in the pre-print[25].

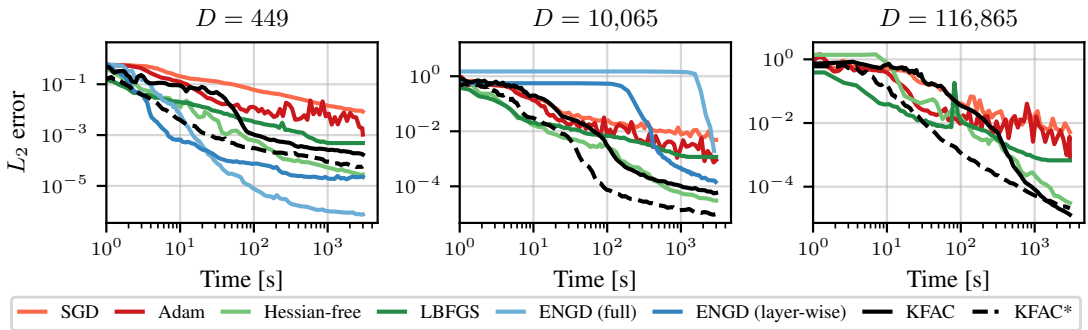
[25] F. Dangel, J. Müller, and M. Zeinhofer (2024). "Kronecker-Factored Approximate Curvature for Physics-Informed Neural Networks". In: *Advances in Neural Information Processing Systems*.

Table of Contents

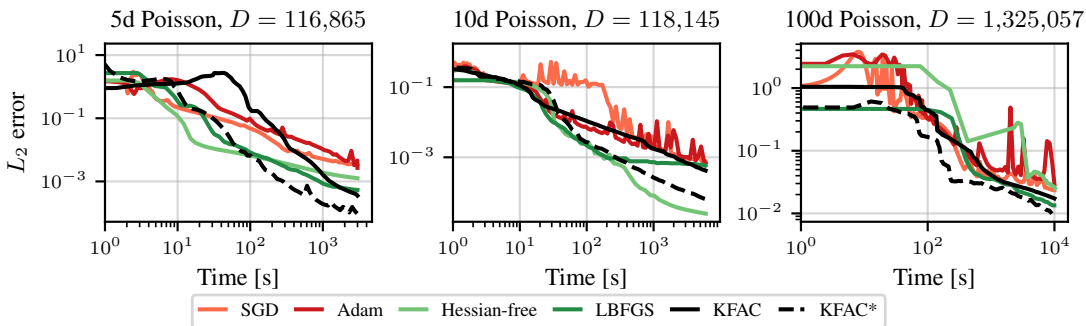
Poisson Equation Two Dimensions



Heat Equation Equation (4+1) Dimensions



High Dimensional Poisson Equation



Discussion

+ The K-FAC approximation allows to scale natural gradient methods to millions of parameters.
+ Even preliminary numerical experiments compare favorable to established optimizers.
+ The approximation generalizes to other PDEs (see pre-print[26] for details).
- The numerical experiments are preliminary, more equations need to follow.
- The K-FAC approximation is architecture dependent and complicated to handle.

[26] F. Dangel, J. Müller, and M. Zeinhofer (2024). "Kronecker-Factored Approximate Curvature for Physics-Informed Neural Networks". In: *Advances in Neural Information Processing Systems*.

Table of Contents

When is the Functional Viewpoint Useful?

- We discussed difficulty of NN optimization for loss functions including PDE operators.
- Infinite dimensional algorithms can exploit geometric structure.
- Obtain an algorithm by Galerkin discretization in tangent space.
- Scalability can be non-trivial.

When can this be useful?

The problem possesses a continuous formulation & contains PDE terms.

When does it not help?

First order methods are successful e.g., supervised deep learning applications.

References I

- Amari, S.-I. (1998). "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2, pp. 251–276.
- Dangel, F., J. Müller, and M. Zeinhofer (2024). "Kronecker-Factored Approximate Curvature for Physics-Informed Neural Networks". In: *Advances in Neural Information Processing Systems*.
- Eschenhagen, R. et al. (2024). "Kronecker-factored approximate curvature for modern neural network architectures". In: *Advances in Neural Information Processing Systems* 36.
- Hu, T., B. Jin, and Z. Zhou (2023). "Solving Poisson Problems in Polygonal Domains with Singularity Enriched Physics Informed Neural Networks". In: *arXiv preprint arXiv:2308.16429*.

References II

- Jnini, A., F. Vella, and M. Zeinhofer (2024). "Gauss-Newton Natural Gradient Descent for Physics-Informed Computational Fluid Dynamics". In: *arXiv preprint arXiv:2402.10680*.
- Li, H. et al. (2018). "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31.
- Li, R. et al. (2024). "A computational framework for neural network-based variational Monte Carlo with Forward Laplacian". In: *Nature Machine Intelligence*, pp. 1–11.
- Li, Z. et al. (2021). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations*.
- Martens, J. (2020). "New Insights and Perspectives on the Natural Gradient Method". In: *The Journal of Machine Learning Research* 21.1, pp. 5776–5851.
- Müller, J. and M. Zeinhofer (2023). "Achieving High Accuracy with PINNs via Energy Natural Gradients". In: *International Conference on Machine Learning*.

References III

- Müller, J. and M. Zeinhofer (2024). "Position: Optimization in SciML Should Employ the Function Space Geometry". In: *Forty-first International Conference on Machine Learning*.
- Pfau, D. et al. (2020). "Ab Initio Solution of the Many-Electron Schrödinger Equation with Deep Neural Networks". In: *Physical Review Research* 2.3, p. 033429.
- Raissi, M., P. Perdikaris, and G. E. Karniadakis (2019). "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* 378, pp. 686–707.
- Schäfer, F. and A. Anandkumar (2019). "Competitive gradient descent". In: *Advances in Neural Information Processing Systems* 32.
- Schraudolph, N. N. (2002). "Fast curvature matrix-vector products for second-order gradient descent". In: *Neural computation* 14.7, pp. 1723–1738.

References IV


Zeng, Q., S. H. Bryngelson, and F. T. Schaefer (2023). "Competitive Physics Informed Networks". In: *International Conference on Learning Representations*.