DE GRUYTER

Konstantin Fackeldey, Aswin Kannan, Sebastian Pokutta, Kartikey Sharma, Daniel Walter, Andrea Walther and Martin Weiser (Eds.)

MATHEMATICAL OPTIMIZATION FOR MACHINE LEARNING

PROCEEDINGS OF THE MATH+ THEMATIC EINSTEIN SEMESTER 2023

PROCEEDINGS IN MATHEMATICS

Konstantin Fackeldey, Aswin Kannan, Sebastian Pokutta, Kartikey Sharma, Daniel Walter, Andrea Walther, and Martin Weiser (Eds.)
Mathematical Optimization for Machine Learning

De Gruyter Proceedings in Mathematics

Mathematical Optimization for Machine Learning

Proceedings of the MATH+ Thematic Einstein Semester 2023

Edited by
Konstantin Fackeldey, Aswin Kannan, Sebastian Pokutta, Kartikey Sharma, Daniel Walter, Andrea Walther, and Martin Weiser

DE GRUYTER

Mathematics Subject Classification 2020
65K10, 68T05, 90Cxx

Editors
Aswin Kannan
Daniel Walter
Andrea Walther
Humboldt-University Berlin
Department of Mathematics
10099 Berlin
Germany
aswin.kannan@hu-berlin.de
daniel.walter@hu-berlin.de
andrea.walther@math.hu-berlin.de

Konstantin Fackeldey
Technische Universität Berlin
Institut für Mathematik
10623 Berlin
Germany
fackeldey@math.tu-berlin.de

Sebastian Pokutta
Kartikey Sharma
Martin Weiser
Zuse Institute Berlin
14195 Berlin
Germany
pokutta@zib.de
kartikey.sharma@zib.de
weiser@zib.de

ISBN 978-3-11-137585-4
e-ISBN (PDF) 978-3-11-137677-6
e-ISBN (EPUB) 978-3-11-137774-2
ISSN 2942-4801
DOI https://doi.org/10.1515/9783111376776

Library of Congress Control Number: 2024952514

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

Typesetting: VTeX UAB, Lithuania
Printing and binding: CPI books GmbH, Leck

www.degruyter.com
Questions about General Product Safety Regulation:
productsafety@degruyterbrill.com

Preface

Mathematical optimization often focuses on accuracy, computational efficiency, and robustness while machine learning (ML) aims to achieve effective results on real data sets, in particular concentrating on generalization, robustness, and resilience (to, e. g., perturbations of the inputs). Up to now, both research areas have been only loosely intertwined with optimization being a tool to execute a learning task from the ML perspective, and ML being just yet another application from the optimization perspective. This is illustrated by the fact that certain variants of stochastic gradient descent (e. g., out-of-the-box Adam) are still the method of choice in ML even though the application of this method is not justified by theory for a large number of methods from ML. On the other hand, mathematical optimization often studies approximation properties for machine learning tasks but so far has not developed optimization approaches that deliberately target the needs of ML (e. g., generalization or sparsity). Hence, there is a lack of advanced optimization techniques that can handle machine learning problems with the required efficiency or learning-based improvements to optimization. Connecting mathematical optimization perspectives with machine learning approaches can lead to new approaches and benefit existing ones. For instance, the problem of predicting cluster membership in unsupervised learning may be greatly aided by an optimization perspective; after all, as with many other data science approaches, the problem can be viewed as maximizing a (complicated) function subject to some constraints. Complementarily, ML can be used to tune optimization methods or develop surrogate models. It heavily influences downstream decision-making and more general decision support systems in many real-world applications across industries.

In the summer of 2023, the Thematic Einstein Semester on the topic of Mathematical Optimization for Machine Learning took place as part of the Excellence Cluster MATH+, with the express aim of fostering the connection between the machine learning and mathematical optimization communities and methodologies. In addition to a special lecture series, a summer school, and seminars, there were three workshops and a concluding conference that dealt with mathematical optimization and ML. The semester covered a wide range of mathematical methods, starting with topics in continuous optimization and extending to practical machine learning applications. The organizers invited around 200 scientists from around the world to talk about the topic. In addition to the exchange of knowledge, the aim was to connect young researchers and students.

This proceedings volume contains selected contributions from the special semester's participants and shall provide insight into various aspects of the topic. The contributed papers are grouped into three parts:

Machine Learning for Scientific and Engineering Applications

This part includes papers that apply machine learning to physics, control systems, and mathematical modeling.

https://doi.org/10.1515/9783111376776-201

- A Framework to Solve Inverse Problems for Parametric PDEs using Adaptive Finite Elements and Neural Networks.
- Generation of Value Function Data for Bilevel Optimal Control and Application to Hybrid Electric Vehicle.
- Graph Neural Networks to Predict Strokes From Blood Flow Simulations.
- Capturing the Macroscopic Behavior of Molecular Dynamics with Membership Functions.
- Adaptive Gradient Enhanced Gaussian Process Surrogates for Inverse Problems.

Optimization in Machine Learning and Control

This group focuses on optimization techniques in neural network training, control systems, and algorithmic advancements.

- Multifidelity Domain Decomposition-based Physics-informed Neural Networks and Operators for Time-dependent Problems.
- Constrained Piecewise Linear Optimization by an Active Signature Method.
- Parallel Trust-Region Approaches in Neural Network Training.
- Trustworthy Optimization Learning: A Brief Overview.
- Compression-aware Training of Neural Networks using Frank–Wolfe.

Theoretical Foundations of Optimization Algorithms

This part emphasizes theoretical exploration of optimization algorithms, regularity, and fundamental analysis.

- Approximation of Generalized Frequency Response Functions via Vector Fitting.
- On the Nonsmooth Regularity Condition LIKQ for Different Abs-normal Representations.
- Divergence of the ADAM Algorithm with Fixed-stepsize: a (very) Simple Example.

Berlin, February 2025

Konstantin Fackeldey
Aswin Kannan
Sebastian Pokutta
Kartikey Sharma
Daniel Walter
Andrea Walther
Martin Weiser

Acknowledgment

We would like to thank all contributing authors and participants of the thematic Einstein Semester, as well as Faisal Shah for TeXnical support.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

https://doi.org/10.1515/9783111376776-202

Contents

Preface —— V

Acknowledgment —— VII

Alexandre Caboussat, Maude Girardin, and Marco Picasso
A framework to solve inverse problems for parametric PDEs using adaptive finite elements and neural networks —— 1

Olivier Cots, Rémy Dutto, Sophie Jan, and Serge Laporte
Generation of value function data for bilevel optimal control and application to hybrid electric vehicle —— 17

Rohit Pochampalli and Nicolas R. Gauger
Graph neural networks to predict strokes from blood flow simulations —— 29

Alexander Sikorski, Robert Julian Rabben, Surahit Chewle, and Marcus Weber
Capturing the macroscopic behavior of molecular dynamics with membership functions —— 41

Phillip Semler and Martin Weiser
Adaptive gradient-enhanced Gaussian process surrogates for inverse problems —— 59

Alexander Heinlein, Amanda A. Howard, Damien Beecroft, and Panos Stinis
Multifidelity domain decomposition-based physics-informed neural networks and operators for time-dependent problems —— 79

Timo Kreimeier, Andrea Walther, and Andreas Griewank
Constrained piecewise linear optimization by an active signature method —— 93

Ken Trotti, Samuel A. Cruz Alegría, Alena Kopaničáková, and Rolf Krause
Parallel trust-region approaches in neural network training —— 107

Pascal Van Hentenryck
Trustworthy optimization learning: a brief overview —— 121

Max Zimmer, Christoph Spiegel, and Sebastian Pokutta
Compression-aware training of neural networks using Frank–Wolfe —— 137

Antonio Carlucci, Ion Victor Gosea, and Stefano Grivet-Talocia
Approximation of generalized frequency response functions via vector fitting —— 169

Ganna Shyshkanova, Timo Kreimeier, Lukas Baumgärtner, Franz Bethke, and Andrea Walther
On the nonsmooth regularity condition LIKQ for different abs-normal representations —— 181

Philippe L. Toint
Divergence of the ADAM algorithm with fixed-stepsize: a (very) simple example —— 195

Index —— 199

Alexandre Caboussat, Maude Girardin, and Marco Picasso

A framework to solve inverse problems for parametric PDEs using adaptive finite elements and neural networks

Abstract: Parameter identification is important in many engineering processes, and benefits from quick evaluations with reduced models. We present here a framework to solve inverse problems for parametric partial differential equations. The reduced model is built on a neural network method for the numerical approximation of a given parametric partial differential equation. Training data are generated thanks to adaptive finite element simulations. A supervised feedforward neural network is then used for the online approximation of the solution.

The inverse problem aims at identifying parameters using available measurement data. The corresponding optimization problem is solved with a particle swarm optimization method. Numerical results are presented for the parameter identification of several elliptic and hyperbolic model problems.

Keywords: Finite element method, parametric PDEs, neural networks, adaptive mesh refinement, inverse problem, particle swarm optimization

MSC 2020: 65N15, 65N30, 65N50, 68T07, 65M32

1 Introduction

We consider parametric partial differential equations that are either stationary

$$\mathcal{F}\big(u(x;\mu);\mu\big) = 0 \quad x \in \Omega,\ \mu \in \mathcal{P}, \tag{1.1}$$

or evolutive

$$\mathcal{F}\big(u(x,t;\mu);\mu\big) = 0 \quad x \in \Omega,\ t \in [0,T],\ \mu \in \mathcal{P}, \tag{1.2}$$

Alexandre Caboussat, Geneva School of Business Administration (HEG-Genève), University of Applied Sciences and Arts Western Switzerland (HES-SO), 17 Rue de la Tambourine, 1227 Carouge, Geneva, Switzerland, e-mail: alexandre.caboussat@hesge.ch
Maude Girardin, Geneva School of Business Administration (HEG-Genève), University of Applied Sciences and Arts Western Switzerland (HES-SO), 17 Rue de la Tambourine, 1227 Carouge, Geneva, Switzerland; and Institute of Mathematics, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, e-mail: maude.girardin@epfl.ch
Marco Picasso, Institute of Mathematics, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, e-mail: marco.picasso@epfl.ch

https://doi.org/10.1515/9783111376776-001

where $\Omega \subseteq \mathbb{R}^d$ ($d \geq 1$) is the physical space, $\mathcal{P} \subset \mathbb{R}^p$ is the parameter space (with $p \geq 1$ possibly large), and $\mathcal{F}$ is some differential operator. Given observed values of the solution u_i at given (space or space-time) locations Y_i, $i = 1 \ldots, N_{\text{obs}}$, the goal is to determine the optimal parameter $\mu^* \in \mathcal{P}$ that minimizes the objective function

$$F(\mu) := \frac{1}{N_{\text{obs}}} \sum_{i=1}^{N_{\text{obs}}} \left| u(Y_i; \mu) - u_i \right|^{\alpha}, \tag{1.3}$$

where $u(\cdot; \mu)$ is the solution of the parametric PDE for a given value of the parameter μ, and where $\alpha > 0$ is given (typically $\alpha = 1$ or $\alpha = 2$). We do not consider noise in the data u_i, and thus do not investigate potential smoothing terms. Any optimization method to minimize $F(\mu)$ requires to compute (an approximation of) the solution $u(\cdot; \mu)$ a large number of times. Such evaluation should thus be fast, hence the advantage to build and use a reduced model to approximate u. Depending on the type of PDE, several reduced-order modeling approaches may be considered, such as, e. g., reduced basis [10], proper orthogonal decomposition [24], polynomial chaos expansion [6], or neural networks [8, 11, 25].

Solving nonlinear optimization problems, including inverse problems, requires dedicated computational optimization methods. Among them, we can mention Newton-type approaches [7] or derivative-free approaches [1, 13, 18, 22]. We will in particular focus here on particle swarm optimization (PSO) [12, 16, 20]. On the other hand, among reduced-order models, we consider here data-driven feedforward neural networks [4, 8], in order to build a neural network approximation $u_{\mathcal{N}}$ of u that can be used for the direct evaluations of the solution. Adaptive mesh refinement techniques are used for the generation of training data [23], which allows to control the accuracy of the training data uniformly in the parameter space [3]. The finite element simulations, as well as the training of the neural network, are done during an *offline* phase, which can be time consuming but is done once and for all. Once trained, the network can be efficiently evaluated during the *online* phase to solve the parameter identification problem.

This work is organized as follows: the finite elements-neural network model is summarized in Section 2. Section 3 describes the solution method for the inverse problem, including the optimization framework relying on a Newton or a PSO approach. Numerical experiments, including both elliptic and hyperbolic PDEs, are presented in Section 4.

2 An adaptive finite elements—neural networks method for the direct problem

Let us recall briefly the numerical method presented in [2, 3] for the solution of the direct problem (1.1). Similarly as in [4], we denote by $\Upsilon^{W,L}(\sigma; d_{\text{in}}, d_{\text{out}})$ the set of fully-connected feedforward neural networks with input dimension d_{in}, output dimension

d_{out}, and L hidden layers, each constituted of W neurons having σ as activation function. We consider here a neural network $\mathcal{N} \in \Upsilon^{W,L}(\sigma; d_{\text{in}}, 1)$ to approximate the mapping $(Y;\mu) \mapsto u(Y;\mu)$, where $d_{\text{in}} = p + d$ if Y denotes the spatial coordinates $x \in \Omega$, and $d_{\text{in}} = p+d+1$ if Y denotes the space-time coordinates $(x,t) \in \Omega\times(0,T)$. Once its trainable parameters θ have been fixed, the network gives an approximation $u_{\mathcal{N}}(\cdot;\theta) : d_{\text{in}} \to d_{\text{out}}$ of u. The training procedure is as follows:

1. Randomly select training parameters $\{\tilde{\mu}_j\}_{j=1}^{N_{\text{train}}} \subseteq \mathcal{P}$, for $N_{\text{train}} \in \mathbb{N}$ given.
2. For each $\tilde{\mu}_j$, set an appropriate discretization of Ω or $\Omega\times[0,T]$, and compute a finite element approximation $u_h(\cdot;\tilde{\mu}_j)$ of $u(\cdot;\tilde{\mu}_j)$. In order to do so, use appropriate a posteriori error analysis to estimate the local error and iteratively obtain the optimal mesh that balances the error over all finite elements (and time steps). This ensures that all the numerical solutions have an accuracy close to a preset tolerance uniformly in $\mathcal{P}$, but implies that the discretizations used to compute the finite element solutions $u_h(\cdot;\tilde{\mu}_j)$ are different for each $\tilde{\mu}_j$.
3. Set the architecture of the neural network, that is, the number of layers L, the number of nodes per layer W, and the activation function σ.
4. Choose the trainable parameters θ of $\mathcal{N} \in \Upsilon^{W,L}(\sigma; d_{\text{in}}, 1)$ to minimize

$$\Phi(\theta) := \mathcal{L}_{N_{\text{train}}}(u_{\mathcal{N}}(\cdot;\theta); u_h).$$

   Here, the objective function $\Phi(\theta)$ is an averaged sum over the training parameters of $\|u_h(\cdot;\tilde{\mu}_j) - u_{\mathcal{N}}(\cdot;\tilde{\mu}_j;\theta)\|_{L^2(\Omega)}$ in the case of a stationary problem [3], and of $\|u_h(\cdot;\tilde{\mu}_j) - u_{\mathcal{N}}(\cdot;\tilde{\mu}_j;\theta)\|_{\ell^2((0,T);L^2(\Omega))}$ in the case of an evolutive problem [2]. We let θ^* be the set of parameters obtained by some random gradient descent type algorithm and denote $u_{\mathcal{N}}(Y;\mu;\theta^*)$ simply by $u_{\mathcal{N}}(Y;\mu)$ from now on. In practice, the Nadam optimizer [19] has been used to find θ^* for the numerical experiments presented in the sequel.

3 A framework for parameter identification

Once a network $\mathcal{N} \in \Upsilon^{W,L}(\sigma; d_{\text{in}}, 1)$ giving an approximation $u_{\mathcal{N}}$ of u has been set up, it can be used to solve inverse problems. Let us consider values u_i that the function u takes at given observation points Y_i, $i = 1,\dots,N_{\text{obs}}$. In Section 4, the points Y_i are either in a spatial domain $(x_i,y_i) \in \Omega$, or in a space-time domain $(x_i,y_i,t_i) \in \Omega\times(0,T)$. We are looking for the optimal parameter $\mu^* \in \mathcal{P}$ that minimizes

$$F(\mu) := \frac{1}{N_{\text{obs}}} \sum_{i=1}^{N_{\text{obs}}} \left|u_{\mathcal{N}}(Y_i;\mu) - u_i\right|^{\alpha}, \tag{1.4}$$

where $\alpha = 1$ or $\alpha = 2$. A minimizer of (1.4) can be searched using any optimization algorithm. The derivatives $\frac{\partial u_{\mathcal{N}}}{\partial \mu}(\cdot;\mu)$ can typically be computed using automatic differentiation [5], which allows the use of gradient-type methods. Nevertheless, these methods

usually require some a priori knowledge for the choice of the initial guess in order to converge. Numerical experiments show that such initial guess may be difficult to find uniformly in the parameter space. On the other hand, derivative-free approaches, such as, e. g., the Particle Swarm Optimization (PSO) algorithm [12, 16, 20], allow for global optimization and more robustness. We start by presenting here both approaches, before illustrating their behavior on several numerical experiments.

3.1 Particle-swarm optimization

Among derivative-free methods, particle-swarm optimization relies on iteratively updating a group of several particles, which individually and collectively evolve to find a global minimizer of a function. This approach has been proved to be very adapted and robust for global optimization problems where the function is nonconvex and may present several local minimizers. The underlying idea is to update the solution for each particle at each iteration taking into consideration its current position but also the positions in the whole group.

We initialize N_{part} particles in $\mathbb{R}^p$, and make their position evolve in order to find a minimizer of (1.4). At iteration $K+1$, the position of the particle i is

$$\mu_i(K+1) = \mu_i(K) + v_i(K+1) \in \mathbb{R}^p,$$

where the components of the velocity $v_i(K+1) = (v_{ij}(K+1))_{j=1}^p$ are

$$v_{ij}(K+1) = \omega(K)v_{ij}(K) + c_1(K)r_{1j}(K)\big(v_{ij}(K) - \mu_{ij}(K)\big) + c_2(K)r_{2j}(K)\big(\hat{v}_j(K) - \mu_{ij}(K)\big). \quad (1.5)$$

In (1.5), r_{kj} are random numbers with uniform distribution in $(0,1)$, $v_i(K) = (v_{ij}(K))_{j=1}^p$ denotes the best position that the particle i has reached during the previous iterations, and $\hat{v}(K) = (\hat{v}_j(K))_{j=1}^p$ denotes the best position that has been among all particles. As suggested in [14, 17], we use varying coefficients ω, c_1, c_2 given at iteration $0 \le K \le K_{\max}$ by

$$c_i(K) = (c_{i,F} - c_{i,I})\frac{K}{K_{\max}} + c_{i,I}, \quad i = 1, 2,$$
$$\omega(K) = (\omega_I - \omega_F - 0.2)\exp\left\{\left(1 + \frac{7K}{K_{\max}}\right)^{-1}\right\},$$

with $c_{1,I} = 2.5$, $c_{2,I} = 0.5$, $\omega_I = 0.95$, $c_{1,F} = 0.5$, $c_{2,F} = 2.5$, $\omega_F = 0.4$. In the sequel, unless explicitly stated otherwise, we choose $K_{\max} = 100$, and then allow the algorithm to perform another 200 iterations with constant values of c_1, c_2, and w. The optimization algorithm stops if the maximum number of iterations (300) is reached or if $F(\hat{v})$ does not decrease for 10 iterations in a row. The initial position of the particles is chosen uniformly at random in $\mathcal{P}$. During the optimization process, the particles are constrained in a slightly extended compact set $\tilde{\mathcal{P}} \supsetneq \mathcal{P}$ (but not strictly in $\mathcal{P}$), in order to be able

to recover parameters that lie on the boundary of $\mathcal{P}$. The implementation of the PSO algorithm has been made using `PySwarms` [15].

3.2 Gradient method

The alternative is to consider a gradient-based method in the parameter space. Given an initial guess $\mu(0)$, we make the position of the particle evolve as

$$\mu(K+1) = \mu(K) - \eta(K)\nabla_\mu F\big(\mu(K)\big),$$

where $\eta(K) = \eta\|\nabla_\mu F(\mu(K))\|^{-1}$. As for the PSO algorithm, we constrain the iterates $\mu(K)$ to remain in $\tilde{\mathcal{P}}$. The implementation of the gradient algorithm relies on automatic differentiation [5] and has been made using `Tensorflow` [21].

We choose as initial guess $\mu(0) = \mu^*(\mathbf{1} + \gamma\mathbf{n})$, where μ^* is the exact solution, $\mathbf{n}_i \in \mathcal{U}([-1,1])$, $i = 1,\dots,p$, and $\gamma > 0$ characterizes the size of the attraction basin of the method. This corresponds to choosing the initial guess in the neighborhood of the exact solution. If $\mu_b(i)$ denotes the best parameter that has been found during iterations $0,\dots,i$, the algorithm is stopped if

$$F\big(\mu(i)\big) > F\big(\mu_b(i-1)\big) \quad \text{or} \quad \frac{F(\mu(i)) - F(\mu_b(i-1))}{F(\mu_b(i-1))} < tol$$

for 20 iterations in a row.

4 Numerical experiments

4.1 A Poisson problem

The first model problem that we consider is a 2D parametric Poisson problem: find $u(\cdot;\mu) : \Omega \to \mathbb{R}$ such that

$$\begin{cases} -\Delta u(x;\mu) = f(x;\mu), & x \in \Omega,\ \mu \in \mathcal{P}, \\ u(x;\mu) = g(x;\mu), & x \in \partial\Omega,\ \mu \in \mathcal{P}, \end{cases} \tag{1.6}$$

with $d = 2$, $\Omega = [0,10] \times [0,2]$, $p = 4$, $\mathcal{P} = [1.5, 8.5] \times [1,10] \times [100,1000] \times [0.3,2]$. Here, f and g are such that the exact solution u of (1.6) is given by

$$u(x;\mu) = \frac{\mu_3}{\pi\mu_4^2} \exp\{-2\mu_4^{-2}\big((x_1-\mu_1)^2 + \mu_2(x_2-1)^2\big)\}.$$

In this case, μ_3 controls the amplitude of the solution, μ_4 the size of its support, μ_1 its position on the x_1-axis, and μ_2 its stretching along the latter. We consider the optimiza-

tion problem (1.4), with $\alpha = 2$. We assume that observed values u_i are available at given points $Y_i = (x_i, y_i)$, $i = 1, \ldots, N_{\text{obs}}$. In order to validate the algorithm, we start by setting an optimal value $\mu^* \in \mathcal{P}$ and $u_i = u(Y_i, \mu^*)$, $i = 1, \ldots, N_{\text{obs}}$.

To ensure that the values u_i are not all close to zero, the measure points are chosen randomly with uniform distribution in the set $\mathcal{S} := \{(x, y) \in \Omega : u(x, y; \mu^*) > u_{\min}\}$. For direct computations, we use a neural network $\mathcal{N} \in \Upsilon^{400,4}(\sigma; 6, 1)$ constructed and trained as described in Section 2. The training set is composed by data coming from 4000 finite element simulations, and the training of the network takes around 2.5 hours in this case. For more details on the building and training of the network, we refer to [3].

4.1.1 Results with the particle-swarm optimization method

Table 1.1 illustrates the mean value of the computed μ, as well as the standard deviation over a sample of 10 realizations of the PSO algorithm. The measure points are chosen randomly in $\mathcal{S}$ at the beginning of each realization of the algorithm; they thus differ from one realization to the other. The algorithm has always stopped before reaching the maximum number of iterations.

Table 1.1: Results of the PSO algorithm, for $N_{\text{part}} = 40$, $N_{\text{obs}} = 30$, $u_{\min} = 1$, $\alpha = 2$, and various values of μ^*.

μ^* (exact)	μ^* (computed)
(5.5, 5.5, 550, 1.15)	(5.50, 5.50, 551, 1.15) ± (0.000356, 0.00799, 0.463, 0.000668)
(8.5, 1, 1000, 0.3)	(8.50, 0.935, 988.5, 0.293) ± (0.000371, 0.00638, 5.10, 0.000712)
(1.5, 1, 100, 2)	(1.50, 1.22, 102, 2.01) ± (0.0130, 0.107, 5.02, 0.0629)
(5, 1, 1000, 2)	(5.00, 1.01, 1001, 2.00) ± (0.00145, 0.00412, 1.81, 0.00209)
(5, 10, 100, 0.3)	(5.00, 10.2, 104, 0.306) ± (0.000679, 0.336, 2.86, 0.00336)
(5, 10, 1000, 1)	(5.00, 9.99, 1002, 1.00) ± (0.000657, 0.0199, 1.15, 0.000480)

Note that the parameter in the first line of Table 1.1 is at the center of $\mathcal{P}$. All the other parameters lie on the boundary of $\mathcal{P}$, and are therefore more challenging to recover. We notice that, even in these extreme cases, μ^* is recovered with good accuracy and small variability among calculations. Figure 1.1 shows the evolution of the cost functional (1.4) during the iterations of the PSO algorithm, for 10 realizations of the algorithm and different μ^*. For each parameter, three distinct phases are observed during the PSO algorithm: first, the particles explore the parameter space and the cost functional decreases only slowly. Then particles all converge toward the (approximated) optimal parameter, and the cost functional drops drastically, before finally oscillating around the optimal parameter in a stationary manner. Note that for $\mu^* = (8.5, 1, 1000, 0.3)$ the cost remains relatively high (around 10). This is explained by the fact that this parameter corresponds to the solution with the highest amplitude and the strongest gradients, being thus one

of the most challenging to capture correctly with the neural network. For this model problem, the PSO algorithm typically takes around $8[s]$ to converge. Note that no finite element simulations are performed during the optimization algorithm.

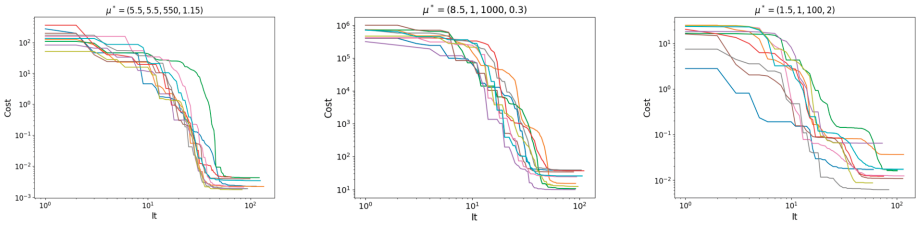


Figure 1.1: Evolution of the cost functional during the PSO iterations, for different parameters μ^* ($N_{\text{part}} = 40$, $N_{\text{obs}} = 30$, $u_{\min} = 1$).

4.1.2 Results with the gradient method

In order to compare the performance of the PSO and gradient descent algorithms, we test the gradient approach using the same parameters μ^* as in the previous section. The mean and the standard deviation of the recovered parameters over 10 realizations of the algorithm are reported in Table 1.2, for $N_{\text{obs}} = 30$, $\eta = 10^{-2}$, $\gamma = 0.3$, and $tol = 5 \cdot 10^{-3}$. Note that the parameters are more accurately recovered with the PSO algorithm.

Table 1.2: Results of the gradient algorithm, for $N_{\text{obs}} = 30$, $u_{\min} = 1$, $\alpha = 2$, $\gamma = 0.3$, and various values of μ^*.

μ^* (exact)	μ^* (computed)
(5.5, 5.5, 550, 1.15)	(5.51, 6.04, 584, 1.19) ± (0.0134, 0.668, 0.89.5, 0.0998)
(8.5, 1, 1000, 0.3)	(7.29, 0.887, 981, 0.355) ± (1.07, 0.177, 131, 0.0842)
(1.5, 1, 100, 2)	(1.50, 1.06, 93.5, 1.95) ± (0.0220, 0.220, 13.0, 0.110)
(5, 1, 1000, 2)	(5.00, 0.642, 849, 1.85) ± (0.0201, 0.267, 103, 0.0991)
(5, 10, 100, 0.3)	(5.16, 9.05, 105, 0.332) ± (0.821, 1.37, 12.2, 0.0647)
(5, 10, 1000, 1)	(5.00, 8.63, 971, 0.978) ± (0.0166, 0.879, 74.3, 0.0486)

Figures 1.2 and 1.3 show the evolution of the cost functional during the gradient iterations. We observe that the gradient method is efficient in terms of convergence speed. However, as illustrated in Figure 1.2, the method does not always converge toward the global minimum when the initial guess is too far from the solution ($\gamma = 0.3$). Actually the method requires an initial guess that is very close to the optimal solution for the method to converge and lacks robustness for particular values of the parameters μ^*. Figure 1.3 illustrates that effect for a given value of μ^* and several values of γ.

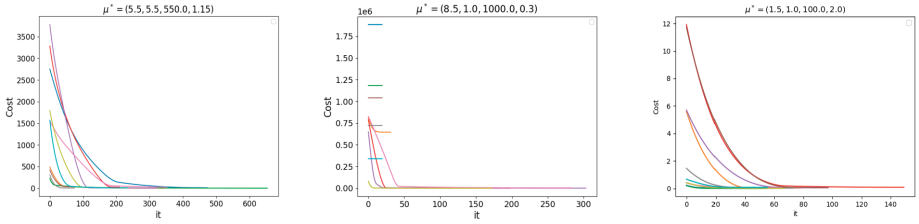


Figure 1.2: Evolution of the cost functional during the gradient iterations, for different parameters μ^* ($N_{\text{obs}} = 30$, $u_{\text{min}} = 1$, $\gamma = 0.3$).

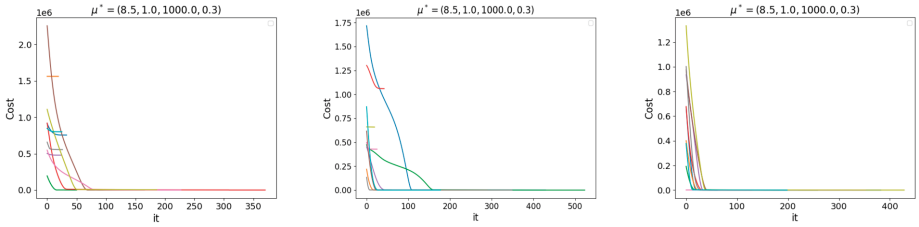


Figure 1.3: Evolution of the cost functional during the gradient iterations, for different values of γ and $\mu^* = (8.5, 1, 1000, 0.3)$ ($N_{\text{obs}} = 30$, $u_{\text{min}} = 1$). From left to right: $\gamma = 0.2, 0.1, 0.05$.

Therefore, for these robustness reasons, we favor the PSO approach in the next numerical experiments. Note that a PSO method can be afforded as long as the evaluation of F—and thus $u_{\mathcal{N}}$—is not too costly. This is the case for the kind of neural networks used here.

4.2 A hyperbolic problem

We consider a parametric transport equation in two dimensions of space and set $p = 3$. Given $\mu = (\mu_1, \mu_2, \mu_3) \in \mathcal{P}$, the problem consists in finding $u : \Omega \times [0, T] \mapsto \mathbb{R}$ satisfying

$$\frac{\partial u}{\partial t}(x, t; \mu) + \mathbf{a}(x, t; \mu)\nabla u(x, t; \mu) = 0, \quad x \in \Omega,\ t \in (0, T),\ \mu \in \mathcal{P}, \tag{1.7}$$

together with the initial condition $u(x, 0; \mu) = u_0(x; \mu)$ for all $x \in \Omega$ and $\mu \in \mathcal{P}$, with $\Omega = (0, 4) \times (0, 4)$ and $T = 2|\mu_1|^{-1}$. The vector field $\mathbf{a}$ is chosen such that there is no inflow boundary, and thus no boundary condition to enforce. We consider $\mu_1 \in [-2, -0.5] \cup [0.5, 2]$, $\mu_2 \in [0.15, 0.3]$, $\mu_3 \in [50, 150]$, and we set $\mathbf{a}(x, t; \mu) = \frac{\mu_1 \pi}{2}\begin{pmatrix} 2 - x_2 \\ x_1 - 2 \end{pmatrix}$ and $u_0(x; \mu) = \tanh(-\mu_3 \sqrt{(x_1 - 2)^2 + (x_2 - 2.5)^2} - \mu_2)$.

In this particular case, μ_1 is a parameter ruling the velocity, μ_2 characterizes the size of the support of the initial condition u_0, and μ_3 is a regularization parameter for the initial condition. As before, we assume that observed values u_i are available at given points $Y_i = (x_i, y_i, t_i)$, $i = 1, \ldots, N_{\text{obs}}$, and that there is no noise in the measurements. We start by choosing $\alpha = 2$ in the objective function (1.4). To ensure that the values u_i allow to recover the optimal parameter, the points Y_i are chosen randomly with uniform distribution in the set $\{(x, y, t) \in \Omega \times [0, T] : u(x, y, t; \mu^*) > u_{\min}\}$. For direct computations, we use a neural network $\mathcal{N} \in \Upsilon^{200,4}(\sigma; 6, 1)$ constructed and trained as in Section 2. See [2] for more details.

Numerical results are illustrated in Table 1.3, for two different numbers of measure points N_{obs} and different μ^*. As for the elliptic problem, we report the mean value and the standard deviation of the computed μ over a sample of 10 realizations of the PSO algorithm, with the measure points changing from one realization to the other. Note that all parameters μ^*, except the first one, lie on the boundary of $\mathcal{P}$, and should therefore be more challenging to capture accurately. In each case, the algorithm has stopped before reaching the maximum number of iterations.

Table 1.3: Results of the PSO algorithm, for $N_{\text{part}} = 200$, $u_{\min} = -0.9$, $\alpha = 2$, and various values of μ^*.

μ^* (exact)	μ^* (computed)
$N_{\text{obs}} = 30$	
(1.25, 0.225, 100)	(1.25, 0.225, 107) ± (0.00106, 0.000327, 13.2)
(2, 0.3, 50)	(2.00, 0.300, 52.1) ± (0.00187, 0.000958, 3.21)
(2, 0.15, 100)	(2.00, 0.150, 113) ± (0.00261, 0.000823, 8.56)
(0.5, 0.3, 50)	(0.498, 0.302, 55.2) ± (0.00584, 0.00459, 22.9)
(1, 0.2, 150)	(1.01, 0.209, 134) ± (0.0273, 0.0255, 25.3)
(0.5, 0.15, 150)	(0.500, 0.151, 136) ± (0.000745, 0.000828, 12.3)
$N_{\text{obs}} = 100$	
(1.25, 0.225, 100)	(1.25, 0.225, 100) ± (0.0000964, 0.0000404, 0.746)
(2, 0.3, 50)	(2.00, 0.300, 53.4) ± (0.00142, 0.000384, 5.35)
(2, 0.15, 100)	(2.00, 0.150, 109) ± (0.000451, 0.000168, 4.37)
(0.5, 0.3, 50)	(0.496, 0.302, 47.3) ± (0.00206, 0.00254, 5.58)
(1, 0.2, 150)	(1.00, 0.201, 138) ± (0.000731, 0.00310, 21.3)
(0.5, 0.15, 150)	(0.501, 0.152, 128) ± (0.00395, 0.00512, 9.66)

We note that μ_1 and μ_2 are recovered with good accuracy for all the parameters that we tested. Nevertheless, we observe in some cases less accurate recovery and/or large variability among calculations for the parameter μ_3 ruling the size of the boundary layer. This lack of accuracy may come from the error introduced by the neural network approximation and/or from the low number of observations in the boundary layer around the initial condition. Furthermore, we see that, as expected, the larger the number of observations (N_{obs}), the smaller the standard deviation of the results.

In order to increase the accuracy of the recovered μ_3, we investigate whether a ℓ^1-based objective function is more efficient to capture the sharp gradients in the boundary layer. Thus, we consider the cost functional (1.4) with $\alpha = 1$.

Numerical results for $N_{\text{obs}} = 100$ are shown in Table 1.4. We note that the choice of a ℓ^1 cost functional indeed allows in most cases to capture μ_3 more accurately and to reduce the variability across calculations. The uncertainty that remains in the recovery of $\mu^* = (0.5, 0.15, 150)$ is explained by the fact that this parameter corresponds to the function $u(\cdot; \mu)$ with the smallest support and the steepest boundary layer, making it hard to capture exactly by the neural network.

Table 1.4: Results of the PSO algorithm, for $N_{\text{part}} = 200$, $N_{\text{obs}} = 100$, $u_{\text{min}} = -0.9$, $\alpha = 1$, and various values of μ^*.

μ^* (exact)	μ^* (computed)
(1.25, 0.225, 100)	(1.25, 0.225, 103) ± (0.00290, 0.00150, 7.27)
(2, 0.3, 50)	(2.00, 0.299, 51.3) ± (0.00107, 0.000302, 1.12)
(2, 0.15, 100)	(2.00, 0.149, 104) ± (0.000424, 0.000124, 2.70)
(0.5, 0.3, 50)	(0.495, 0.301, 50.9) ± (0.00226, 0.00147, 1.96)
(1, 0.2, 150)	(1.00, 0.200, 148) ± (0.000148, 0.0000729, 4.38)
(0.5, 0.15, 150)	(0.500, 0.151, 134) ± (0.000216, 0.000605, 9.03)

Figure 1.4 shows the evolution of the cost functional during the iterations of the PSO algorithm, for 10 realizations of the algorithm and different μ^*. The cost functional evolves in a very similar way to the elliptic model problem, with three distinct phases. Note that for $\mu^* = (2.0.3, 50)$ and $\mu^* = (0.5, 0.3, 50)$, the parameters are accurately recovered, but the cost functional stagnates above 10^{-2}, mainly due to error introduced by the neural network approximation. For this model problem with $N_{\text{part}} = 200$, the PSO algorithm typically takes around $6[s]$ when $N_{\text{obs}} = 30$ and $20[s]$ when $N_{\text{obs}} = 100$.

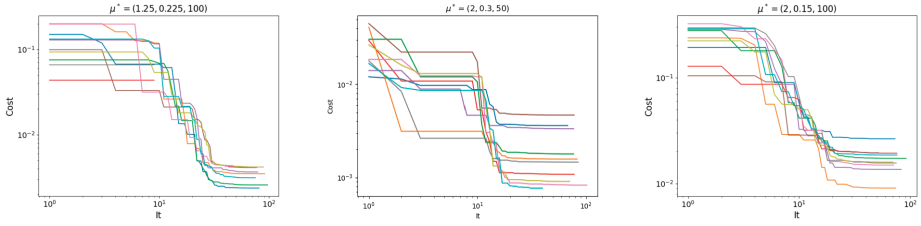


Figure 1.4: Evolution of the cost functional during the PSO iterations, for different parameters μ^*. $N_{\text{part}} = 200$, $N_{\text{obs}} = 100$, $u_{\text{min}} = -0.9$, $\alpha = 1$.

4.3 An elliptic problem with more parameters

Finally, we choose as a last numerical experiment the elliptic problem

$$\begin{cases} -\nabla \cdot (a(x;\mu)\nabla u(x;\mu)) = f(x), & (x,\mu) \in \Omega \times \mathcal{P}, \\ u(x;\mu) = 0, & (x;\mu) \in \partial\Omega \times \mathcal{P}. \end{cases} \tag{1.8}$$

As in [9], we take $\Omega = [0,1]^2$, $\mathcal{P} = [-\sqrt{3}, \sqrt{3}]^9$, and consider $f(x) = 10\sin(2\pi(x_1 + x_2))$ and $a(x;\mu) = 1 + 0.1\sum_{i=1}^{16} \psi_i(x)\mu_i$, with

$$\begin{gathered} \psi_1 = \lambda_1\phi_1(x_1)\phi_1(x_2), \quad \psi_2 = \sqrt{\lambda_1\lambda_2}\phi_1(x_1)\phi_2(x_2), \quad \psi_3 = \sqrt{\lambda_2\lambda_1}\phi_2(x_1)\phi_1(x_2), \\ \psi_4 = \lambda_2\phi_2(x_1)\phi_2(x_2), \quad \psi_5 = \sqrt{\lambda_1\lambda_3}\phi_1(x_1)\phi_3(x_2), \quad \psi_6 = \sqrt{\lambda_3\lambda_1}\phi_3(x_1)\phi_1(x_2), \\ \psi_7 = \sqrt{\lambda_2\lambda_3}\phi_2(x_1)\phi_3(x_2), \quad \psi_8 = \sqrt{\lambda_3\lambda_2}\phi_3(x_1)\phi_2(x_2), \quad \psi_9 = \lambda_3\phi_3(x_1)\phi_3(x_2), \\ \psi_{10} = \sqrt{\lambda_1\lambda_4}\phi_1(x_1)\phi_4(x_2), \quad \psi_{11} = \sqrt{\lambda_4\lambda_1}\phi_4(x_1)\phi_1(x_2), \\ \psi_{12} = \sqrt{\lambda_2\lambda_4}\phi_2(x_1)\phi_4(x_2), \quad \psi_{13} = \sqrt{\lambda_4\lambda_2}\phi_4(x_1)\phi_2(x_2), \\ \psi_{14} = \sqrt{\lambda_3\lambda_4}\phi_3(x_1)\phi_4(x_2), \quad \psi_{15} = \sqrt{\lambda_4\lambda_3}\phi_4(x_1)\phi_3(x_2), \\ \psi_{16} = \lambda_4\phi_4(x_1)\phi_4(x_2), \end{gathered}$$

where $\{\lambda_j, \phi_j\}, j = 1,2,3$, are the four first eigenpairs of the Karhunen–Loève expansion of a 1D Gaussian random field of covariance $C(x,x') = e^{|x-x'|}$. We refer to [9] for an illustration of the basis functions ψ_i. Figure 1.5 shows the solution $u(\cdot;\mu)$ corresponding to different parameters μ.

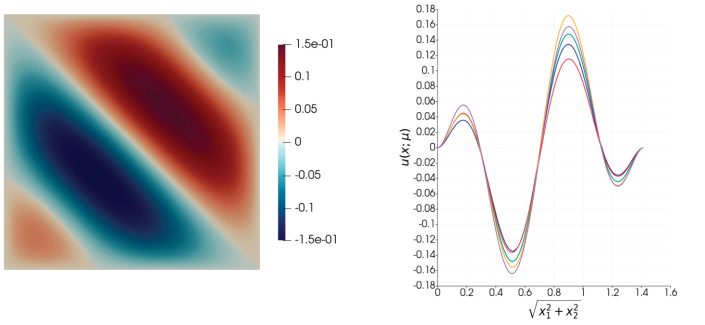


Figure 1.5: Left: Snapshot of an averaged solution $u(\cdot;\mu)$. Right: Cut of the solutions $u(\cdot;\mu)$ along the diagonal $x_1 = x_2$ for various parameters μ.

We truncate the expansion of a at $i = 4$, $i = 9$, and $i = 16$ (corresponding to the eigenpairs $\{\lambda_j, \phi_j\}$, for $j = 1,2,3$, resp.), obtaining thus three different networks $\mathcal{N}^4$, $\mathcal{N}^9$,

and $\mathcal{N}^{16}$. To train the latter, 3000 finite element simulations are performed, which takes around 7 hours offline. The training of the networks then take around 11 hours offline in the three cases.

Unlike the first two test cases, no analytical solution is known for the problem (1.8). For each of the three sizes of expansion, we therefore use the finite element solution computed on a very fine grid (with mesh size $h = 0.005$) as the exact solution to generate the data $\{u_i\}_{i=1}^{N_{\text{obs}}}$.

In the sequel, we compare the performances of the PSO algorithm for the different expansions and different number of measures points and number of particles.

Table 1.5 reports the mean value ± the standard deviation of the objective function F and the CPU time, computed over 50 realizations of the algorithm, performed with randomly chosen parameters. We observe that the performances of the algorithm are not significantly impacted as the dimension of the parameter space is increased. Similarly, the time taken by the algorithm to converge naturally depends on the number of observation points and of particles, but only weakly on the dimension of the parameter space.

Table 1.5: Average performances of the PSO algorithm, for different number of observations N_{obs}, number of particles N_{part} and different sizes of expansion N_{KL}.

N_{KL}	N_{obs}	N_{part}	F	*CPU* [s]
4	100	100	$1.89 \cdot 10^{-8} \pm 5.66 \cdot 10^{-8}$	9.07 ± 3.18
		1000	$8.62 \cdot 10^{-9} \pm 9.06 \cdot 10^{-9}$	32.9 ± 12.0
	400	100	$2.02 \cdot 10^{-8} \pm 3.04 \cdot 10^{-8}$	17.43 ± 7.55
		1000	$1.29 \cdot 10^{-8} \pm 8.36 \cdot 10^{-9}$	124 ± 48.8
9	100	100	$5.29 \cdot 10^{-8} \pm 1.59 \cdot 10^{-7}$	11.8 ± 3.80
		1000	$1.44 \cdot 10^{-8} \pm 2.88 \cdot 10^{-8}$	45.1 ± 13.1
	400	100	$2.83 \cdot 10^{-8} \pm 6.78 \cdot 10^{-8}$	24.9 ± 8.43
		1000	$1.64 \cdot 10^{-8} \pm 2.08 \cdot 10^{-8}$	171 ± 36.7
16	100	100	$2.98 \cdot 10^{-8} \pm 3.25 \cdot 10^{-8}$	12.6 ± 4.14
		1000	$2.39 \cdot 10^{-8} \pm 3.44 \cdot 10^{-8}$	53.4 ± 15.9
	400	100	$4.54 \cdot 10^{-8} \pm 8.17 \cdot 10^{-8}$	28.1 ± 8.06
		1000	$2.73 \cdot 10^{-8} \pm 2.19 \cdot 10^{-8}$	184 ± 56.5

Table 1.6 illustrates, for $N_{\text{obs}} = 400$ and $N_{\text{part}} = 1000$, the mean value ± the standard deviation of the error committed on each component of the parameters μ. We observe that the last components of the parameter become more and more difficult to recover accurately. This is easily explained as those components correspond to the basis functions that are more oscillatory and have smaller amplitude.

Finally, Table 1.7 illustrates, similar to the other experiments, for $N_{\text{part}} = 1000$, $N_{\text{obs}} = 400$, $n = 3$, and different targeted parameters μ^*, the parameters returned by the PSO algorithm.

Table 1.6: Average error for each component of the targeted parameter μ and different sizes of expansions N_{KL}.

N_{KL}	4	9	16
μ_1	$1.53 \cdot 10^{-3} \pm 2.58 \cdot 10^{-3}$	$2.58 \cdot 10^{-3} \pm 2.22 \cdot 10^{-3}$	$5.94 \cdot 10^{-3} \pm 3.84 \cdot 10^{-3}$
μ_2	$3.53 \cdot 10^{-3} \pm 4.11 \cdot 10^{-3}$	$1.03 \cdot 10^{-2} \pm 2.80 \cdot 10^{-2}$	$2.02 \cdot 10^{-2} \pm 1.42 \cdot 10^{-2}$
μ_3	$2.67 \cdot 10^{-3} \pm 3.36 \cdot 10^{-3}$	$5.98 \cdot 10^{-2} \pm 4.47 \cdot 10^{-3}$	$2.17 \cdot 10^{-2} \pm 1.67 \cdot 10^{-2}$
μ_4	$2.23 \cdot 10^{-2} \pm 2.56 \cdot 10^{-2}$	$1.80 \cdot 10^{-2} \pm 4.71 \cdot 10^{-2}$	$6.73 \cdot 10^{-2} \pm 4.06 \cdot 10^{-2}$
μ_5		$1.14 \cdot 10^{-2} \pm 2.19 \cdot 10^{-2}$	$3.94 \cdot 10^{-2} \pm 2.82 \cdot 10^{-2}$
μ_6		$9.72 \cdot 10^{-3} \pm 1.27 \cdot 10^{-2}$	$3.87 \cdot 10^{-2} \pm 2.77 \cdot 10^{-2}$
μ_7		$2.77 \cdot 10^{-2} \pm 2.98 \cdot 10^{-2}$	$9.66 \cdot 10^{-2} \pm 7.00 \cdot 10^{-2}$
μ_8		$5.43 \cdot 10^{-2} \pm 1.87 \cdot 10^{-1}$	$9.09 \cdot 10^{-2} \pm 7.52 \cdot 10^{-2}$
μ_9		$9.00 \cdot 10^{-2} \pm 6.26 \cdot 10^{-2}$	$1.24 \cdot 10^{-1} \pm 9.37 \cdot 10^{-2}$
μ_{10}			$6.28 \cdot 10^{-2} \pm 4.77 \cdot 10^{-2}$
μ_{11}			$5.86 \cdot 10^{-2} \pm 4.80 \cdot 10^{-2}$
μ_{12}			$1.80 \cdot 10^{-1} \pm 1.22 \cdot 10^{-1}$
μ_{13}			$1.91 \cdot 10^{-1} \pm 1.48 \cdot 10^{-1}$
μ_{14}			$3.21 \cdot 10^{-1} \pm 2.03 \cdot 10^{-1}$
μ_{15}			$3.01 \cdot 10^{-1} \pm 2.24 \cdot 10^{-1}$
μ_{16}			$5.97 \cdot 10^{-1} \pm 4.47 \cdot 10^{-1}$

Table 1.7: Results of the PSO algorithm, for $N_{\mathrm{part}} = 1000$, $N_{\mathrm{obs}} = 400$, $n = 3$, $\alpha = 2$, and various values of μ^*.

μ^* (exact)	μ^* (computed)
(0.866, 0.866, 0.866, 0.866, 0.866, 0.866, 0.866, 0.866, 0.866)	(0.869, 0.866, 0.861, 0.866, 0.861, 0.860, 0.865, 0.845, 0.848) ± (0.00154, 0.00371, 0.00505, 0.0166, 0.00921, 0.0104, 0.0282, 0.0251, 0.100)
(−0.957, −0.0973, 0.114, 0.144, −1.17, 1.30, −1.44, −0.904, 0.663)	(−0.955, −0.0999, 0.111, 0.150, −1.17, 1.30, −1.45, −0.923, 0.623) ± (0.00141, 0.00460, 0.00785, 0.0165, 0.00987, 0.00957, 0.0500, 0.0293, 0.0733)
(1.73, 1.73, 1.73, 1.73, 1.73, 1.73, 1.73, 1.73, 1.73)	(1.73, 1.71, 1.72, 1.67, 1.71, 1.73, 1.67, 1.79, 1.80) ± (0.00164, 0.00468, 0.00353, 0.0255, 0.0142, 0.0118, 0.0314, 0.219, 0.147)
(1.73, 1.73, −1.73, 1.73, −1.73, −1.73, −1.73, 1.73, 1.73)	(1.74, 1.72, −1.73, 1.71, −1.75, −1.75, −1.74, 1.72, 1.81) ± (0.00265, 0.00524, 0.00473, 0.0220, 0.0171, 0.0136, 0.0422, 0.0541, 0.155)
(0, 0, 0, 0, 0, 0, 0, 0, 1.73)	(0.00135, −0.0079, −0.00370, 0.00798, 0.00753, −0.00182, 0.00857, 0.0126, 1.65) ± (0.00155, 0.00565, 0.00409, 0.00812, 0.0107, 0.00863, 0.0288, 0.0356, 0.0369)

We note that the algorithm allows to recover a good approximation of the parameter μ^* overall. As already underlined above, the components of the parameter become harder to capture accurately as the corresponding basis functions become more oscillatory and of smaller amplitude. Finally, we note that the value of the cost functional is lower than for the first two test cases. This can be explained by the fact that the solution u has much higher amplitude in the first test case and exhibits boundary layers that are

hard to capture correctly in the second test case, whereas the solution is much smoother and has smaller amplitude for this last experiment. Figure 1.6 shows the evolution of the cost functional for 10 iterations of the PSO algorithm and different μ^*. We note that the evolution is similar to the one observed for the first two test cases, and thus does not depend significantly on the dimension of the parameter space.

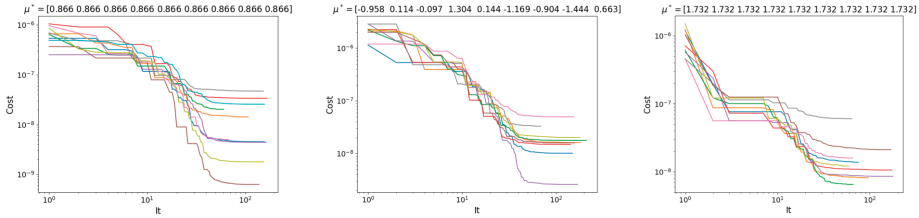


Figure 1.6: Evolution of the cost functional during the PSO iterations, for different parameters μ^*. $N_{\text{part}} = 1000$, $N_{\text{obs}} = 400$.

5 Conclusions and perspectives

A framework for parameter identification for parametric PDEs has been presented. The direct solution of the differential equations is approximated with an adaptive finite elements-neural network solver that has been previously validated in the literature.

Optimization algorithms for the solution of the inverse problem have been proposed. Gradient-type methods show a fast convergence, but with rather small attraction basin, especially when the dimension of the parameter set becomes large or when the exact solution are near the boundaries of that set. For robustness reasons, derivative-free approaches, such as particle-swarm optimization methods, have been favored.

Numerical experiments have shown that parameter identification can be achieved in a robust manner, for parameters whose dimension goes up to 16. Future work will include heuristics to better couple derivative-free approaches for the efficient localization of an initial guess for gradient-type methods. A requirement of this coupling is to determine the appropriate indicators to estimate the appropriate timing to switch between algorithms.

Bibliography

[1] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[2] A. Caboussat, M. Girardin, and M. Picasso. Error assessment for a finite elements-neural networks approach applied to parametric PDEs. The transport equation. In: *Proceedings of the 11th Edition of the International Conference on Adaptive Modeling and Simulation (ADMOS 2023)*, 2023.

[3] A. Caboussat, M. Girardin, and M. Picasso. Error assessment of an adaptive finite elements—neural networks method for an elliptic parametric PDE. *Computer Methods in Applied Mechanics and Engineering*, 421:116784, 2024.
[4] R. DeVore, B. Hanin, and G. Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.
[5] C. Elliott. The simple essence of automatic differentiation. In: *Proceedings of the ACM on Programming Languages*, volume 2(ICFP), pages 1–29, 2018.
[6] O. G. Ernst, A. Mugler, H.-J. Starkloff, and E. Ullmann. On the convergence of generalized polynomial chaos expansions. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46(2):317–339, 2012.
[7] A. Galántai. The theory of Newton's method. *Journal of Computational and Applied Mathematics*, 124(1–2):25–44, 2000.
[8] M. Geist, P. Petersen, M. Raslan, R. Schneider, and G. Kutyniok. Numerical solution of the parametric diffusion equation by deep neural networks. *Journal of Scientific Computing*, 88(1):1–37, 2021.
[9] D. S. Guignard. *A posteriori error estimation for partial differential equations with random input data*. PhD thesis, EPFL, 2016.
[10] J. S. Hesthaven, G. Rozza, B. Stamm, and et al.. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*, volume 590. Springer, 2016.
[11] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
[12] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In: *Proceedings of ICNN'95—International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.
[13] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.
[14] H.-R. Li and Y.-L. Gao. Particle swarm optimization algorithm with exponent decreasing inertia weight and stochastic mutation. In: *2009 Second International Conference on Information and Computing Science*, volume 1, pages 66–69. IEEE, 2009.
[15] L. J. Miranda. PySwarms: A research toolkit for Particle Swarm Optimization in Python. *The Journal of Open Source Software*, 3(21):433, 2018.
[16] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization: An overview. *Swarm Intelligence*, 1:33–57, 2007.
[17] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.
[18] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56:1247–1293, 2013.
[19] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
[20] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation—CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950. IEEE, 1999.
[21] Tensorflow Developers. Tensorflow. *Official website*, 2018.
[22] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.
[23] R. Verfürth. A posteriori error estimators for convection-diffusion equations. *Numerische Mathematik*, 80:641–663, 1998.
[24] S. Volkwein. Model reduction using proper orthogonal decomposition. *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz*. See https://www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/POD-Vorlesung.pdf, 1025, 2011.
[25] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, 7(40):eabi8605, 2021.

Olivier Cots, Rémy Dutto, Sophie Jan, and Serge Laporte

Generation of value function data for bilevel optimal control and application to hybrid electric vehicle

Abstract: In this article, we present two numerical methods to create a database for the approximation of the value function of a bilevel optimal control problem. The first method is based on the computation of the value function via indirect simple shooting, which implies to find the zeros of functions. The second one amounts to solve Cauchy problems. These two techniques are compared, in terms of prior information, computation cost, and data distribution, on an industrial application: the torque split and gear shift optimal control problem on hybrid electric vehicles.

Keywords: Database generation, bilevel optimal control, Pontryagin maximum principle, indirect method, value function, hybrid electric vehicle

MSC 2020: 49-06

1 Introduction

The value function is a central object in optimal control theory that describes how the optimal cost depends on the initial conditions. This function was studied in the 1950s by Richard Bellman and leads to the Hamilton–Jacobi–Bellman (HJB) partial differential equations. These equations provide necessary and sufficient conditions for an optimal control problem, as well as the optimal control in a feedback form. They are the baseline of dynamic programming [2] and reinforcement learning [14], which are two of the main optimal control methods used in industrial applications. However, these methods are subject to the curse of dimensionality, which is a key numerical issue for embedded solutions.

Acknowledgement: This research was partially supported by Vitesco Technologies. In particular, we want to thank Olivier Flebus, artificial intelligence and optimization group leader at Vitesco Technologies and Mariano Sans, optimal control senior expert at Vitesco Technologies, for their supervision, valuable insights, and suggestions.

Olivier Cots, Institut de Recherche en Informatique de Toulouse (IRIT), UMR CNRS 5505, Université de Toulouse, INP-ENSEEIHT, 6 allée Emile Monso, BP 34038, Toulouse, France, e-mail: olivier.cots@toulouse-inp.fr
Rémy Dutto, IRIT, IMT, Vitesco Technologies, Toulouse, France, e-mail: remy.dutto@orange.fr
Sophie Jan, **Serge Laporte,** Institut de Mathématiques de Toulouse (IMT), UMR CNRS 5219, Université de Toulouse, UPS, 118 route de Narbonne, 31062 TOULOUSE CEDEX 9, France, e-mails: sophie.jan@math.univ-toulouse.fr, serge.laporte@math.univ-toulouse.fr

https://doi.org/10.1515/9783111376776-002

The Pontryagin maximum principle [12] also introduced in the 1950s gives necessary optimality conditions for optimal control problems, and leads to the indirect numerical methods, which promise to be accurate and fast enough to be used for embedded solutions. For a description of classical numerical methods for optimal control problems, see [4].

A new method based on a bilevel decomposition of the optimal control problem, presented in [5], promises to reduce the number of computations and to be fast enough for embedded solution. This method uses some value functions between fixed times, which can rarely be calculated explicitly, and the strategy is to approximate them from a set of precomputed data. The main objective of this paper is to propose and compare two methods to create the associated database. The first method is based on the computation of the value function on a given grid thanks to the resolution of the shooting equations obtained from the Pontryagin maximum principle, while the second method amounts to evaluate the value function on a computed grid thanks to the integration of the underlying Hamiltonian flow.

The article is organized as follows. We first introduce the motivations in Section 2 based on the bilevel optimal control method. Then we present the two methods to create the value function database in Section 3. We compare in Section 4 the numerical results we have obtained when applying these two methods to an industrial application: the torque split and gear shift optimal control of a hybrid electric vehicle. Finally, Section 5 concludes the article.

2 Motivations

This paper is motivated by the torque split and gear shift optimal control problem described in [5] and used for the numerical results in Section 4. Another motivation is the resolution of this problem by the bilevel optimal control method, fully detailed in [5] and briefly presented in Section 2.2, which is robust, fast, and that can be embedded. One crucial feature of this method is the computation of approximations, denoted $C_i(a, b)$ hereafter, of value functions, denoted $V_i(a, b)$ and representing the optimal cost to transfer the state from the initial condition a to the target b. In the same reference [5], the cost approximations C_i are modeled by neural networks, trained on specific databases. The goal of this paper is to explain the method (referred as Method 2 in the following) used to create these databases in [5], but also to compare it to a more intuitive but naive approach (referred as Method 1 in the following).

2.1 Optimal control problem

We consider the following optimal control problem in Bolza form:

$$\text{(OCP)}\qquad \begin{cases} \min\limits_{x,u} \int_{t_0}^{t_f} f^0(t,x(t),u(t))\,\mathrm{d}t + g(x(t_f)), & (2.1)\\ \text{s. t.}\ \ \dot{x}(t) = f(t,x(t),u(t)), \quad t \in [t_0,t_f]\ \text{a. e.}, & (2.2)\\ \qquad u(t) \in \mathrm{U}, \qquad\qquad\quad t \in [t_0,t_f], & (2.3)\\ \qquad x(t_0) = x_0, & (2.4) \end{cases}$$

where the function $f^0\colon \mathbb{R}\times\mathbb{R}^n\times\mathbb{R}^m \to \mathbb{R}$, the terminal cost $g\colon \mathbb{R}^n \to \mathbb{R}$, and the state dynamic function $f\colon \mathbb{R}\times\mathbb{R}^n\times\mathbb{R}^m \to \mathbb{R}^n$ are of class $\mathscr{C}^1$. The initial and final times $t_0 < t_f$ are fixed, as well as the initial state $x_0 \in \mathbb{R}^n$. The control domain $\mathrm{U} \subset \mathbb{R}^m$ is a nonempty set. Solving (OCP) consists in finding an absolutely continuous[1] *state* $x \in \mathrm{AC}([t_0,t_f],\mathbb{R}^n)$ and an essentially bounded[2] *control* $u \in \mathrm{L}^\infty([t_0,t_f],\mathbb{R}^m)$, which minimize the cost (2.1) and satisfy the constraints (2.2), (2.3), and (2.4).

Remark. For conciseness and clarity, we consider an optimal control problem in this Bolza form but our approach can be easily extended to a more general problem with fixed final state or mixed limit conditions for instance.

2.2 Bilevel optimal control method

Considering intermediate times $t_0 < t_1 < \cdots < t_N < t_{N+1} = t_f$, the optimal control problem (OCP) can be decomposed into the following bilevel optimal control problem:

$$\text{(BOCP)}\qquad \begin{cases} \min_X \ \sum_{i=0}^{N} V_i(X_i,X_{i+1}) + g(X_{N+1}),\\ \text{s. t. } X = (X_0,\dots,X_{N+1}) \in \mathcal{X}, \quad X_0 = x_0, \end{cases}$$

where for all $i \in \mathbb{N}_N \coloneqq \{0,\dots,N\}$, the functions V_i are the *intermediate value functions*: for any admissible[3] pair (a,b), $V_i(a,b)$ is the optimal cost of

$$(\mathrm{OCP}_{i,a,b})\qquad \begin{cases} V_i(a,b) = \min_{x,u} \int_{t_i}^{t_{i+1}} f^0(t,x(t),u(t))\,\mathrm{d}t,\\ \qquad\qquad \text{s. t. } \dot{x}(t) = f(t,x(t),u(t)), \quad t \in [t_i,t_{i+1}]\ \text{a. e.},\\ \qquad\qquad\quad u(t) \in \mathrm{U}, \qquad\qquad\quad t \in [t_i,t_{i+1}],\\ \qquad\qquad\quad x(t_i) = a,\ x(t_{i+1}) = b. \end{cases}$$

The set $\mathcal{X}$ appearing in (BOCP) is defined as follows: $X \in \mathcal{X} \subset (\mathbb{R}^n)^{N+2}$ if, for all $i \in \mathbb{N}_N$, the pair (X_i,X_{i+1}) is admissible for $(\mathrm{OCP}_{i,X_i,X_{i+1}})$. As detailed in [5], problem (BOCP) can be seen as a bilevel optimal control problem [8].

---

1 $\mathrm{AC}([t_0,t_f],\mathbb{R}^n)$ is the set of absolutely continuous functions on $[t_0,t_f]$ valued in $\mathbb{R}^n$.

2 $\mathrm{L}^\infty([t_0,t_f],\mathbb{R}^m)$ is the set of essentially bounded functions on $[t_0,t_f]$ valued in $\mathbb{R}^m$.

3 The pair (a,b) is admissible for $(\mathrm{OCP}_{i,a,b})$ if b is reachable at time t_{i+1} from a at t_i.

In many applications, the intermediate value functions V_i cannot be computed explicitly. Thus, it has been proposed in [5] to replace them by approximations denoted C_i. This leads to a hierarchical method in two steps, called Macro–Micro. The first step consists in solving the finite-dimensional optimization problem

$$\text{(Macro)} \qquad \begin{cases} \min_X \sum_{i=0}^{N} C_i(X_i, X_{i+1}) + g(X_{N+1}), \\ \text{s. t. } X \in \mathcal{X}, \quad X_0 = x_0, \end{cases}$$

to get the intermediate state $\hat{X} = (\hat{X}_0, \dots, \hat{X}_{N+1})$. The second step is to solve $N+1$ independent optimal control problems of the following Lagrange form:

$$\text{(Micro)} \qquad \begin{cases} \min_{x,u} \int_{t_i}^{t_{i+1}} f^0(t, x(t), u(t))\, \mathrm{d}t, & \\ \text{s. t. } \dot{x}(t) = f(t, x(t), u(t)), & t \in [t_i, t_{i+1}] \text{ a. e.}, \\ \quad u(t) \in \mathrm{U}, & t \in [t_i, t_{i+1}], \\ \quad x(t_i) = \hat{X}_i, \quad x(t_{i+1}) = \hat{X}_{i+1}, & \end{cases}$$

to get the state and command trajectory on each subinterval $[t_i, t_{i+1}]$. This method is suboptimal, due to the approximation error between C_i and V_i, but it promises to be faster and to require less computations than classical methods, which is particularly interesting for embedded applications.

However, before applying the Macro–Micro method, we need to build the approximation functions C_i. We postpone to Section 3 the creation of the optimal values database, which will be used for that purpose, and which is the main contribution of this article. From now on, we consider one optimal control problem of the problem (Micro), for a given fixed $i \in \mathbb{N}_N$. The next section gives some classical details about the so-called simple indirect shooting method to solve it.

2.3 Indirect method

The considered optimal control problem of problem (Micro) may be solved by the classical indirect simple shooting method, which is based on the Pontryagin maximum principle [12]. The core function of this principle is the *pseudo-Hamiltonian*

$$h(t, x, p, u) = -f^0(t, x, u) + \big(p \,|\, f(t, x, u)\big),$$

where $(\cdot \,|\, \cdot)$ stands for the usual scalar product in $\mathbb{R}^n$. For a given initial state and costate couple $z_i = (x_i, p_i) \in \mathbb{R}^n \times \mathbb{R}^n$, let us define the problem of finding the state and costate trajectory $z(\cdot)$ solution of

$$\begin{cases} \dot{z}(t) = \vec{h}(t, z(t), u(t)), & t \in [t_i, t_{i+1}] \text{ a. e.}, \\ h(t, z(t), u(t)) = \max_{w \in \mathrm{U}} h(t, z(t), w), & t \in [t_i, t_{i+1}] \text{ a. e.}, \\ z(t_i) = z_i, & \end{cases} \tag{2.5}$$

where $\vec{h}$ is the *pseudo-Hamiltonian vector field* defined by

$$\vec{h}(t,(x,p),u) = \big(\nabla_p h(t,(x,p),u),\ -\nabla_x h(t,(x,p),u)\big).$$

If we assume that the *exponential map* $\exp_{\vec{h}}\colon (t_{i+1},t_i,z_i) \mapsto z(t_{i+1})$ is a well-defined application, where $z(\cdot)$ is a solution of problem (2.5), then the maximum principle leads to the resolution of the two-points boundary value problem

$$\text{(TPBVP)} \qquad \begin{cases} (x_{i+1},p_{i+1}) = \exp_{\vec{h}}(t_{i+1},t_i,(x_i,p_i)), \\ x_i = \hat{X}_i, \quad x_{i+1} = \hat{X}_{i+1}. \end{cases}$$

To use simple notation, let us define $\bar{x}_{i+1}(z_i)$ by

$$\bar{x}_{i+1}(z_i) = \pi_x\big(\exp_{\vec{h}}(t_{i+1},t_i,z_i)\big), \tag{2.6}$$

where $\pi_x(x,p) = x$ is the projection on the state space.

In the following, we assume that for all initial and final admissible state (a,b), if (x,u) is a solution of $(\mathrm{OCP}_{i,a,b})$, then there exists $p \in \mathrm{AC}([t_i,t_{i+1}],\mathbb{R}^n)$ such that the pair (x,p) is a solution of (TPBVP). Such a pair solution of problem (2.5) is called an *extremal*. A solution of (TPBVP) is an extremal that satisfies the boundary conditions, and it is called a *BC-extremal*. This assumption is linked to the normality of the BC-extremals associated to the solution (cf. [1] for more details). For the sake of simplicity, we denote by $\bar{c}_i(z_i)$ the cost of the state and command trajectory, given by the solution of problem (2.5), with the initial state and costate conditions $z_i = (x_i,p_i)$. We also consider the following hypothesis that will be numerically verified in our application.

Hypothesis 2.1. Let (x_i,x_{i+1}) be a given initial and final admissible state. Then all the zeros of

$$p_i \mapsto \bar{x}_{i+1}(x_i,p_i) - x_{i+1} \tag{2.7}$$

are associated to the same state and command trajectory (x,u), depending on the pair (x_i,x_{i+1}).

Remark. Under the previous assumptions and Hypothesis 2.1, for all initial and final admissible state (a,b), there exists at most one solution of $(\mathrm{OCP}_{i,a,b})$.

Remark. Hypothesis 2.1 comes from our application but is not necessary to apply the two proposed methods described in Section 3. However, it simplifies their descriptions and helps to understand their main characteristics. In fact, consider another application where it does not hold, namely where there are several zeros of (2.7) associated to different state-control pairs. If we still have a unique solution of $(\mathrm{OCP}_{i,a,b})$, we have to add another step to the two proposed methods, which consists simply of keeping the zero with the lowest cost. On the other hand, if there are several solutions, we can either pick one or keep all since it is not an issue to have redundant points in the database.

The indirect simple shooting method aims to solve (TPBVP) by finding a zero of the *shooting function*

$$S_{i,\hat{X}_i,\hat{X}_{i+1}}(p_i) = \bar{x}_{i+1}(\hat{X}_i, p_i) - \hat{X}_{i+1}. \tag{2.8}$$

Thanks to Hypothesis 2.1, all the zeros of the shooting function lead to the same optimal cost and, therefore, only one of them need to be found. On the opposite, without this hypothesis, it would be necessary to find all the zeros and to compare them in terms of cost to get the optimal one.

Equation (2.8) is solved using a classical Newton-like solver, and it is well known that finding a good initial guess for the costate is a critical issue. However, given a solution X^* of (BOCP), if V_i is differentiable at (X_i^*, X_{i+1}^*), it is established [3] that the vector $-\nabla_a V_i(X_i^*, X_{i+1}^*)$ is a zero of S_{i,X_i^*,X_{i+1}^*}. Unfortunately, V_i is not known, but since C_i is an approximation of V_i, the vector $-\nabla_a C_i(\hat{X}_i, \hat{X}_{i+1})$ could be a good initial guess for the associated optimal control problem of the problem (Micro) shooting function [7]. This initial guess is used with a geometric preconditioning method of the shooting function on the proposed application in [6].

Now that we have described how to solve the optimal control problems of the problem (Micro), we will focus on the creation of a database for the computation of C_i, which is the main contribution of this paper.

3 Data generation

In this section, we shall describe the creation of a database $\mathbb{D}_i$ containing optimal values of $(\mathrm{OCP}_{i,x_i,x_{i+1}})$ for various pairs (x_i, x_{i+1}):

$$\mathbb{D}_i \subset \{(x_i, x_{i+1}, c) \mid (x_i, x_{i+1}) \text{ admissible and } V_i(x_i, x_{i+1}) = c\}.$$

Two methods are presented to achieve this goal. They are compared in terms of prior information, computation cost, and data distribution.

3.1 Method 1

The first method consists in computing the value function V_i on a given grid by solving optimal control problems thanks to the simple shooting method. This is the intuitive approach compared to the second method presented in the next section. Let $\mathbb{X}_i$ be a discretization of admissible initial and final states. For all $(x_i, x_{i+1}) \in \mathbb{X}_i$, we solve the optimal control problem $(\mathrm{OCP}_{i,x_i,x_{i+1}})$ by an indirect simple shooting method and store the corresponding optimal value $V_i(x_i, x_{i+1})$ in $\mathbb{D}_i$. The main complexity comes from the fact we have to solve equations, which essentially amounts to invert the exponential map given in (2.6). Algorithm 2.1 provides an overview of these steps.

Algorithm 2.1 Method 1.

Require: $\mathbb{X}_i$
Ensure: $\mathbb{D}_i$
  for all $(x_i, x_{i+1}) \in \mathbb{X}_i$ **do**
    $p_i \leftarrow \text{solve}(S_{i,x_i,x_{i+1}}(p_i) = 0)$ ▷ Newton solver
    $c \leftarrow \bar{c}_i(x_i, p_i)$ ▷ Optimal cost
    $\mathbb{D}_i.\text{append}(x_i, x_{i+1}, c)$ ▷ Storage
  end for

Prior information

First of all, the set $\mathbb{X}_i$ needs to be created as follows: the initial state is first discretized, then for each of these values the set of associated reachable final states has to be determined and discretized. This ensures that the space of initial and final state is fully explored and that all pairs $(x_i, x_{i+1}) \in \mathbb{X}_i$ are admissible, which prevents the Newton solver to fail. The determination of the reachable final states is a difficulty in this method. See the remark on page 25 for details about how we compute this set of reachable states in our application.

Computation cost

For each iteration of the Newton solver used to find a zero of the shooting function, at least one integration of the Hamiltonian vector field is required.

Data distribution

By construction, the distribution of the pairs (x_i, x_{i+1}) is fully controlled inside the admissibility set. For example, a uniform distribution may be chosen for an easy interpolation.

3.2 Method 2

The second method, described in Algorithm 2.2, is based on a discretization $\mathbb{Z}_i$ of the initial state and costate space. For all $z_i = (x_i, p_i) \in \mathbb{Z}_i$, the exponential map $z_{i+1} = (x_{i+1}, p_{i+1}) = \exp_{\overrightarrow{h}}(t_{i+1}, t_i, z_i)$ is computed, and Hypothesis 2.1 ensures that $V_i(x_i, x_{i+1}) = \bar{c}_i(x_i, p_i)$. Compared to Method 1, here we only perform a direct computation of the exponential map $\exp_{\overrightarrow{h}}$ instead of inverting it.

Prior information

Since there are no admissibility constraints on (x_i, p_i), the set $\mathbb{Z}_i$ can be the Cartesian product between the discretization of initial state and costate. Nevertheless, the range of the costate discretization must be well chosen in order to ensure a full exploration of the reachable states x_{i+1} while limiting the number of useless calls of the exponential map.

Algorithm 2.2 Method 2.

Require: $\mathbb{Z}_i$
Ensure: $\mathbb{D}_i$
  for all $z_i = (x_i, p_i) \in \mathbb{Z}_i$ **do**
    $x_{i+1} \leftarrow \bar{x}_{i+1}(z_i)$ ▷ Final state
    $c \leftarrow \bar{c}_i(z_i)$ ▷ Optimal cost
    $\mathbb{D}_i$.append(x_i, x_{i+1}, c) ▷ Storage
  end for

Computation cost

In contrast to Method 1, no (Newton) solver is involved here. Hence, only one evaluation of $\exp_{\overrightarrow{h}}$ is needed to compute a point of $\mathbb{D}_i$.

Data distribution

In this method, the distribution of the reached final states x_{i+1} is no more controlled. The main consequence is the risk of empty zones. Fortunately, we shall see in Section 4 that, in our application, a sufficient density in the set $\mathbb{Z}_i$ ensures a satisfactory distribution of reached states x_{i+1}.

4 Results

Presentation of the numerical experiments

The considered application is the optimal control of the torque split and the gear shift of a Hybrid Electric Vehicle (HEV) on the *Worldwide harmonized Light vehicles Test Cycle* (WLTC). The goal is to minimize the fuel consumption of the Internal Combustion Engine (ICE) for fixed initial condition and fixed or free final state of charge of the battery by acting on the gear and on the torque split between the ICE and the Electric Motor (EM). We refer to [9, 10, 11, 13, 15, 16] for a general presentation of these kinds of applications.

Remark. The model of the considered application is given by an industrial code developed by the company, Vitesco Technologies, which uses tabulated data to model complex functions as transmission torque losses or fuel consumption of the ICE. We refer to [5] for a more detailed presentation of this specific nonlinear model.

This problem can be formulated as (OCP) and we consider that we stand in the frame of the assumptions stated in Section 2.3. The bilevel method can be applied in this context, and the time interval (1800 seconds of the WLTC) is arbitrary decomposed into 18 time subintervals of 100s.

The goal of this section is to compare the two methods presented in Sections 3.1 and 3.2 on our HEV application. For this purpose and to clarify the presentation of the

results, we only focus on the first subinterval $[0, 100]$ and we fix the initial state, denoted x_0 to 0.5. So next, we apply the two methods to create the database $\mathbb{D}_0$.

Validation of Hypothesis 2.1

The evolution of the function $p_0 \mapsto \bar{x}_1(x_0, p_0)$ is shown in Figure 2.1. It can be noticed that for all $p_0 \in [p_0^-, p_0^+]$, the function (2.6) is injective. Moreover, for all $p_0 \leq p_0^-$ (resp., $p_0 \geq p_0^+$), we numerically verified that the extremal resulting from the exponential map is associated to a unique (x, u). Hence, the Hypothesis 2.1 can be considered numerically valid.

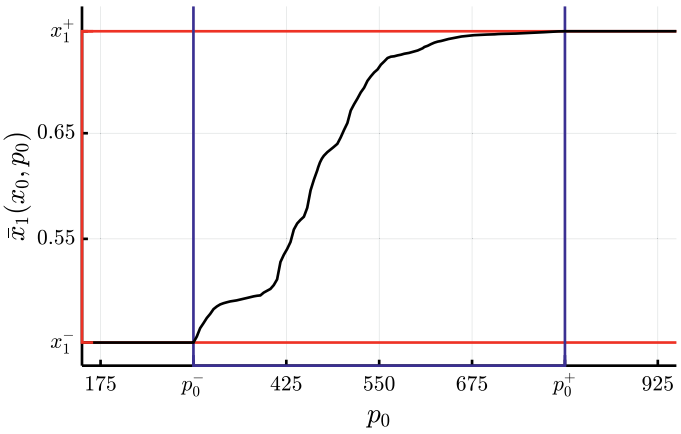


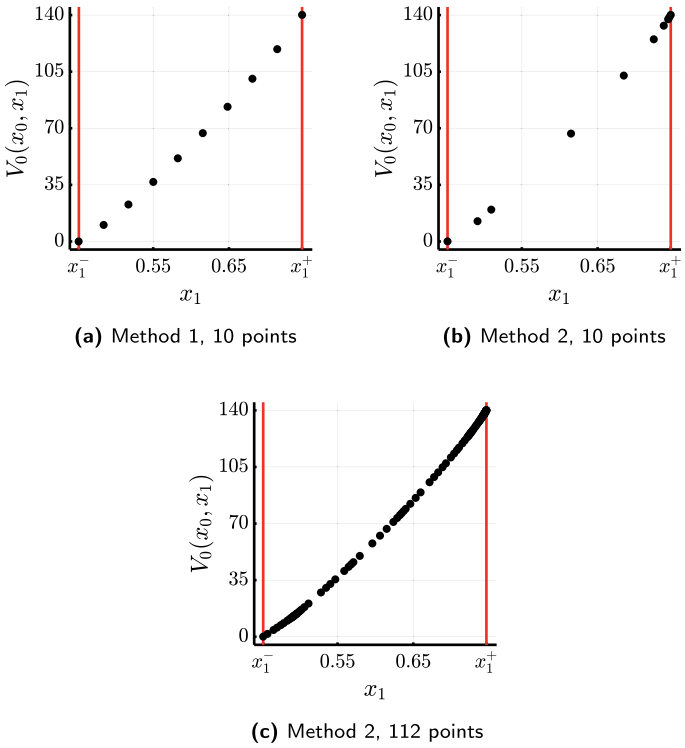
Figure 2.1: Evolution of the function (2.6), with a fixed initial state $x_0 = 0.5$. The red interval on the y-axis corresponds to the admissible final state set at time t_1, and the blue interval on the x-axis corresponds to the useful costate initialization set, mentioned in the Prior Information paragraph of Method 2.

Remark. As discussed in Section 3, the final states must be chosen in $[x_1^-, x_1^+]$ for Method 1. In contrast, for Method 2 they naturally belong to this interval, but although the initial costate can be taken in $\mathbb{R}^n$, its useful range is limited to the interval $[p_0^-, p_0^+]$; cf. Figure 2.1. In our application, the bounds x_1^- (resp., x_1^+) can be evaluated by simulation, fixing the command $u = 0$ (resp., the command that maximizes $\dot{x}(t)$ at each time step).

Results analysis

In order to compare both methods in terms of number of calls to the exponential map, 475 points (x_i, x_{i+1}, c) have been generated for various initial and final admissible states with Method 1. We observe that the Newton solver needs in average 11.2 iterations to find a zero, which means that this method requires at least 11.2 times more calls to $\exp_{\vec{h}}$ than Method 2. Hence, the computation time needed to obtain a point with Method 1 is at least 11.2 times longer than with Method 2.

The two methods are now compared in terms of distribution of the final states. First, 10 points of $\mathbb{D}_0$ are generated by each method. As expected, we can observe in Figures 2.2a and 2.2b that the final states are obviously uniformly distributed with Method 1, while there are not with Method 2. Second, for at least the same computational cost as Method 1, 112 points of $\mathbb{D}_0$ are generated by Method 2. The results are shown in Figure 2.2c. It can be observed that if the distribution remains nonuniform, the coverage of the reachable interval $[x_1^-, x_1^+]$ is much better than the one obtained by Method 1.



(a) Method 1, 10 points

(b) Method 2, 10 points

(c) Method 2, 112 points

Figure 2.2: Value function data created by the two proposed methods. The black points correspond to the computed points by the methods.

Remark. These results have been presented on the first subinterval of 100 seconds of the WLTC for a unique initial state $x_0 = 0.5$. Note that Method 2 has been implemented in [5] on all the 18 subintervals of the WLTC on a fine discretization of initial states and costates. Method 2 has been preferred essentially because it is fast and there is no equation to solve, just numerical integration to perform. In [5], this torque split and gear shift optimal control problem has been solved by the previously described bilevel method with neural networks C_i to approximate the value functions V_i, trained on databases created by Method 2.

5 Conclusion

The Macro–Micro method proposed in [5] requires the approximation of value functions, which can be performed using a database composed of points $(a, b, V_i(a, b))$. We present two numerical methods to create such a database. The first one, based on the computation of the value functions $V_i(a, b)$ for a set of given pairs (a, b), requires to find an initial costate p' solution of the shooting equation that appears in the indirect method. The second method consists in evaluating the exponential map $\exp_{\overrightarrow{h}}$ for a set of given pairs (a, p), where p is the initial costate to get a final state b' and the corresponding cost $\bar{c}_i(a, p)$, which under Hypothesis 2.1, is the optimal one: $V_i(a, b') = \bar{c}_i(a, p)$.

The pros and cons of both methods are gathered in Table 2.1. The uncontrolled distribution obtained by Method 2 can be balanced by its favorable computation cost. Indeed, in our application, for at least the same number of calls to the exponential map as in Method 1, it appears that the domain of reachable final states is well covered. This makes this Method 2 really competitive.

Table 2.1: Pros and cons of Method 1 and Method 2.

	Method 1	**Method 2**
Pros	Controlled distribution	Computation cost
Cons	Prior knowledge of admissible domain	Prior knowledge of initial costate range
	Computation cost	Non controlled distribution

Once the database is created, the associated value function approximation C_i has to be built. A wide range of methods can be used such as neural networks models. The latter are renowned for their generalization capacities and their ability to provide the gradient of the approximation C_i through the retropropagation technique. Indeed, the availability of this gradient should ease the resolution of the problem (Macro) and provide a good initialization for the resolution of the optimal control problems of the problem (Micro) by the indirect method.

Another perspective is to use the gradient of the value function to improve the fitting of the approximations C_i. Indeed, it can be shown under some assumptions that the gradient of the value function $V_i(x_i, x_{i+1})$ is the vector $(-p_i, p_{i+1})$. Therefore, the computation of the Hamiltonian flow $(x_{i+1}, p_{i+1}) = \exp_{\overrightarrow{h}}(t_{i+1}, t_i, (x_i, p_i))$ provides the optimal cost transition $V_i(x_i, x_{i+1})$ as well as its associated gradient. An extended database containing this additional information on the gradient could then be created without additional cost and used for instance to train Physics-Informed Neural Networks (PINNs).

Bibliography

[1] A. A. Agrachev and Y. L. Sachkov. *Control Theory from the Geometric Viewpoint*. Springer, Berlin Heidelberg, 2004.

[2] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[3] O. Bokanowski, A. Désilles, and H. Zidani. Relationship between maximum principle and dynamic programming in presence of intermediate and final state constraints. *ESAIM. Control, Optimisation and Calculus of Variations*, 27:91, 2021.

[4] J.-B. Caillau, R. Ferretti, E. Trélat, and H. Zidani. An algorithmic guide for finite-dimensional optimal control problems. In: *Handbook of Numerical Analysis: Numerical Control, Part B*. North-Holland, 2022.

[5] O. Cots, R. Dutto, S. Laporte, and S. Jan. A bilevel optimal control method and application to the hybrid electric vehicle, 2023.

[6] O. Cots, R. Dutto, S. Laporte, and S. Jan. Geometric preconditioner for indirect shooting and application to hybrid vehicle, 2024.

[7] E. Cristiani and P. Martinon. Initialization of the shooting method via the Hamilton–Jacobi–Bellman approach. *Journal of Optimization Theory and Applications*, 146(2):321–346, 2010.

[8] A. Didier and A. Svensson. A short state of the art on multi-leader-follower games. In: *Bilevel Optimization: Advances and Next Challenges*, pages 53–76. Springer, 2020. Chapter 3.

[9] L. Johannesson, M. Asbogard, and B. Egardt. Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming. In: *Proceedings. 2005 IEEE Intelligent Transportation Systems*, pages 366–371, 2005.

[10] N. Kim, S. Cha, and H. Peng. Optimal control of hybrid electric vehicles based on Pontryagin's minimum principle. *IEEE Transactions on Control Systems Technology*, 19(5):1279–1287, 2011.

[11] A. A. Malikopoulos. Supervisory power management control algorithms for hybrid electric vehicles: a survey. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1869–1885, 2014.

[12] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishechenko. *The Mathematical Theory of Optimal Processes*. New York, 1962.

[13] G. Rousseau. *Véhicule hybride et commande optimale*. PhD thesis, ENMP 2008.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[15] X. Wang, H. He, F. Sun, and J. Zhang. Application study on the dynamic programming algorithm for energy management of plug-in hybrid electric vehicles. *Energies*, 8(4):3225–3244, 2015.

[16] S. G. Wirasingha and A. Emadi. Classification and review of control strategies for plug-in hybrid electric vehicles. *IEEE Transactions on Vehicular Technology*, 60(1):111–122, 2011.

Rohit Pochampalli and Nicolas R. Gauger

Graph neural networks to predict strokes from blood flow simulations

Abstract: A novel approach for predicting stroke risk using graph neural networks (GNNs) from computational fluid dynamics (CFD) simulations of blood flow in patient-based arterial vessel geometries is proposed. Atherogenesis is the build-up of plaque along arterial vessel walls. It is influenced by the geometrical configuration of the vessel, typically in the vicinity of a bifurcation. Furthermore, some geometric features of the bifurcation such as larger curvature can cause intense perturbations in the flow. Using simulations of blood flow, it is possible to obtain flow parameters such as the distribution of wall shear stress, which would otherwise be impossible to measure. GNNs enable learning of the complex relationships linked to the geometry of the blood vessel and the development of atherosclerosis. Our approach provides new insights into the relationship between blood flow patterns and stroke risk, potentially enabling more personalized prevention and treatment strategies.

Keywords: Machine learning, graph neural networks, graph convolutions, cardiovascular disease, ischemic strokes, hemodynamic simulations

MSC 2020: 68T07, 92B20, 05C90

1 Introduction

In this work, we present an application of graph neural networks (GNNs) [45, 19, 3] to the problem of predicting the risk of ischemic strokes [13] using simulations of blood flow in arteries. Ischemic strokes are typically caused by atherosclerosis, which is a condition that presents as a localized constriction of arteries that supply blood to the heart and the brain. Cardiovascular diseases are a major cause of death and disability worldwide and understanding its pathology is of considerable importance. Specifically, the presence of plaque alone need not be a sign to assume that there is a risk of the development of

Acknowledgement: The authors gratefully acknowledge financial support from the Federal Ministry of Education and Research (Bundesinisterium für Bildung und Forschng, BMBF), under the project MLgSA (FKZ: 05MM20UKD) and the computational resources provided by the RHRZ high performance computing center via the "Elwetritsch" cluster at the RPTU Kaiserslautern-Landau. We also thank our colleagues at RPTU in Landau and at the University of Jena for providing the simulation data sets. We also acknowledge the feedback and comments from two anonymous reviewers, which greatly improved the present article.

Rohit Pochampalli, Nicolas R. Gauger, Scientific Computing Group, University of Kaiserslautern-Landau (RPTU), Kaiserslautern, Germany, e-mails: rohit.pochampalli@scicomp.uni-kl.de, nicolas.gauger@scicomp.uni-kl.de

https://doi.org/10.1515/9783111376776-003

a stroke. Instead, it is the "presence of thrombosis-prone plaque," which is of greater relevance [12].

Simulations of blood flow on patient-based arterial wall geometries (i. e., geometries generated from computed tomography (CT) scans) provide a novel approach to measure blood flow parameters, especially those that cannot be measured in vivo. The pipeline of such an application would consist of the following steps: isolation of the artery geometry from scans of the patient, generation of a suitable computational mesh from this geometry, and simulation of blood flow using an appropriate flow model (see [8], [34] and [35]).

The distribution of wall shear stresses on the arterial vessel walls is quite relevant to the application at hand. Investigations into the pathology of atherogenesis show that the distribution of wall shear stress (WSS) has a primary role. For instance, lower WSS is to be expected in the interior of branches with larger curvature [17]. Gnasso et al. [18] conclude that WSS is distributed asymmetrically among the left and right and that lower WSS indicates presence of atherosclerotic lesions, whereas Dolan, Kolega, and Meng [7] conclude that high WSS and a positive WSS gradient indicate the destabilization of plaque. The geometric structure of the arteries is of significant importance to atherogenesis as the configuration of vessels along a bifurcation influence the flow, and consequently, the WSS [39].

One is naturally led to the question of whether a mapping between the WSS distributions and the risk of an ischemic stroke can be learned by a machine learning algorithm. It is our hypothesis that presence of high WSS gradients and the corresponding local structure possess enough information to assess stroke risk. For this task, we develop a GNN model that maps a blood flow simulation to a risk category. The GNN isolates high WSS gradient locations using a novel clustering procedure based on the node-based total variation of the WSS (Section 3). In Section 1.1, we list some related research that apply machine learning methods to problems associated with strokes.

The task of approximating a mapping between wall shear stress and risk of stroke would require the consideration of perturbations in the wall sheer stress as well as the proximity of these perturbations to a bifurcation or other geometric characteristics such as the vessel curvature. Therefore, we rule out simple models in favor of neural networks that employ convolutional filters, which are the primary choice to learn from data with spatially dependent features. Each simulation naturally conforms to a graph structure if the computational grid is treated as a collection of nodes and edges. The WSS is then interpreted as a node feature and edges have weights corresponding to their lengths in space. In Section 3, we elaborate on how such a structure yields possibilities for learning from the spatial features alongside with the WSS.

An alternative to treating the simulations as graphs would be to work directly with images. While this is considerably faster due to the lack of geometry extraction and simulation steps, there are nontrivial issues with image based approaches. Several image classification data sets are comprised of tens of thousands if not millions of data points

[27, 6]. Second, induction from image data requires a preprocessing step called registration, which is the transformation of these images into a common coordinate system in order to treat each data point on an equal footing. This is especially important in medical applications where nonrigid structures are present [30, 36]. Finally, CNNs suffer from a lack of interpretability in their representations. It is not always clear why an image is assigned to a particular class from looking at the activations of the filters. This problem is somewhat mitigated by GNNs as we show in the results of our clustering layer, which constrains the receptive field of the model to nodes with high total variation.

1.1 Related work and contributions

A variety of research is associated with the application of machine learning techniques toward problems related to strokes. There is considerable separation in the research based on the problem addressed and the type of machine learning algorithm used. For instance, the following works have applied one or several machine learning models including k-nearest neighbors, random forests, support vector machines, gradient boosting, or neural networks to problems such as: predicting the functional outcome of some treatment [32, 23, 42]; classification into types of stroke disease [20, 1, 16], and segmentation of problem regions [41]. This is only a small selection of the published research in this field.

Graph classification has been studied before using kernels [25], boosting [44] and linear programming [37], to name a few. We are concerned with a class of supervised algorithms called message passing algorithms, which were developed as generalizations of convolutions [5]. This leads to the graph neural network (GNN) model, which is used to classify graphs based on recursive compression and aggregation of the node feature information [45, 3, 19, 38]. GNNs have been shown to possess universal function approximation properties [4]. To our knowledge, the novelty of our work stems from the use of CFD simulations of blood flow in the arteries as inputs to a GNN to predict stroke risk. The methodology used to extract subgraphs based on node total variation is also new.

2 Background

This section provides background on graphs, total variation on graphs, GNNs, and other details of the machine learning setup. An undirected graph $\mathcal{G}$ is a tuple of sets $(\mathcal{V}, \mathcal{E})$ where for each edge $e \in \mathcal{E}$, there are nodes $u, v \in \mathcal{V}$ such that $[u, v] \coloneqq e$ is the edge joining u and v. Graphs are represented by their adjacency matrices $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, the entries of which stipulate edge connectivity, and optionally, edge weight matrices $E \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where $e_{ij}\colon \mathcal{E} \to \mathbb{R}$ is the Euclidean length of the edge. The neighborhood $\mathcal{N}(u)$ of a node u is the set of nodes v such that $[u, v] \in \mathcal{E}$. The degree d_u of a node u is the

number of edges incident on u. For any subset of nodes, $V \subset \mathcal{V}$, the coboundary of V [31], written $\delta(V)$ is the set of edges that have one end in V and the other not in V. Functions on graph domains map each node in the graph to a real scalar or vector. Scalar functions on graphs will be called signals. The total variation (TV) of real or complex valued functions is a well-studied object in analysis [15]. A related concept of TV is used in signal processing with applications in image processing [29]. The notion of TV used in our work can be derived from the graph difference operator originally defined in Wang et al. [43] or similar notions found in Jung et al. [24] and Hansen and Bianchi [22], where it is a part of the objective function or regularization.

We say that data is graph-structured if each data point can be represented as a tuple consisting of a set of nodes, a corresponding adjacency matrix, a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times m}$, which appends an m-vector of features at each node of the graph. Given graph-structured data, $\mathcal{S} \coloneqq \{\mathcal{G}_1, \ldots, \mathcal{G}_s\}$ and a set of labels $\mathcal{C} \coloneqq \{c_1, \ldots, c_l\}$, the graph classification problem seeks to learn a classifier $h\colon \mathcal{G}_k \mapsto c^k$ that maps each graph to its correct label $c^k \in \mathcal{C}$. In practice, each graph $\mathcal{G}_k$ in $\mathcal{S}$ is a tuple consisting of the adjacency matrix $\mathcal{A}_k$, the node features $\mathcal{X}_k$ and possibly others such as edge features $\mathcal{W}_k$.

GNNs operate on graph-structured data using message passing operations, which preserve the graph structure of the input. Message passing is comprised of two steps called the message and update, which are denoted here by $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, respectively. The k^{th} iteration of the message passing operation updates hidden features $\boldsymbol{h}_u^k$, corresponding to a node $u \in \mathcal{V}$, using messages taken from the graph neighborhood $\mathcal{N}(u)$ as follows:

$$\boldsymbol{h}_u^{k+1} = \boldsymbol{\psi}^k(\boldsymbol{h}_u^k, \boldsymbol{\phi}^k(\{\boldsymbol{h}_v^k; e_{uv}\ \forall v \in \mathcal{N}(u)\})). \tag{3.1}$$

Moreover, the initial features are just the set of node features $\boldsymbol{h}_u^0 = \boldsymbol{x}_u$.

A specific type of message passing sequence called GraphSAGE (introduced in [21]) is used in our model. Here, after the aggregation of features from neighbors of a node u, a set of weight matrices W^t for $t \in \{1, \ldots, T\}$ are defined, which are used to propagate information between different layers of the network. In other words, if T GraphSAGE layers are present, then at each layer, the computations performed are:

$$\boldsymbol{h}_{\mathcal{N}(u)}^{k+1} = \boldsymbol{\phi}^k(\{\boldsymbol{h}_v^k; e_{uv}\ \forall v \in \mathcal{N}(u)\}), \tag{3.2}$$

$$\boldsymbol{h}_u^{k+1} = \sigma(W^t\ \text{CONCAT}(\boldsymbol{h}_u^k, \boldsymbol{h}_{\mathcal{N}(u)}^{k+1})). \tag{3.3}$$

In Eq. (3.2), at each node in the neighborhood of u, an embedding is generated from aggregating the features from all other nodes. In Eq. (3.3), these features are concatenated with the features of the corresponding node and combined using the weight matrices. Here, σ is an activation function and in our case, the ReLU function. The message ϕ in the GraphSAGE convolution is a simple linear operator. Finally, pooling layers are used to extract a graph level representation $\mathbf{h}_{\mathcal{G}}$ from node features $\boldsymbol{h}_v$ in order to perform graph classification. An example is the global mean pooling layer that returns the average of the feature vectors over all nodes $\mathbf{h}_{\mathcal{G}} = \frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}} \boldsymbol{h}_v$.

Subsequently, $\mathbf{h}_{\mathcal{G}}$ is passed to a multilayer perceptron (denoted by MLP) and then transformed into a categorical variable. The latter assigns a probability for the input data point corresponding to each label. The error with respect to the true label can be found with the Binary Cross Entropy (BCE) function when there are just two categories. The BCE loss $\mathcal{L}_{\text{BCE}}$ for a sample with true label y and corresponding predicted probability p is defined as follows:

$$\mathcal{L}_{\text{BCE}}(y,p) = -\big[y\log(p) + (1-y)\log(1-p)\big]. \tag{3.4}$$

This loss is then averaged over all the samples in the batch or mini-batch.

3 Methods

3.1 Variation on graphs

We fix a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in order to avoid cumbersome notation. The TV of a signal f (on $\mathcal{G}$) at a node v is defined as

$$\text{TV}^f(v) \coloneqq \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} \big|f(u) - f(v)\big|. \tag{3.5}$$

This is a signal on the same graph with interesting properties. First, TV^f is subadditive and absolutely homogeneous, that is, for any signals f, g, real numbers α, β, and every node v, we have

$$\text{TV}^{\alpha f+\beta g}(v) \le |\alpha|\text{TV}^f(v) + |\beta|\text{TV}^g(v). \tag{3.6}$$

It can be used to define a seminorm on the space of signals $\mathcal{F} \coloneqq \{f \mid f\colon \mathcal{G} \to \mathbb{R}\}$ as

$$\|f\|_{\text{TV}} \coloneqq \max_{v \in \mathcal{V}} \text{TV}^f(v). \tag{3.7}$$

$\|f\|_{\text{TV}}$ inherits the properties of subadditivity and absolute homogeneity from $\text{TV}^f(\cdot)$. Call two functions in $\mathcal{F}$ equivalent if their difference is a constant. This partitions the signal space into equivalence classes. Thus, a function has norm 0 if and only if it is a constant, which is in the equivalence class of 0. We have proved the following.

Lemma 3.1. *$\|f\|_{TV}$ is a norm on the set of equivalence classes of signals on a specific graph $\mathcal{G}$.*

Intuitively, $\text{TV}^f(v)$ captures the oscillation of the signal f in the neighborhood of v. It is easy to see that $\text{TV}^f(v) = 0$ implies that the signal f is constant on $\delta(\{v\})$, whereas nodes with high TV show jumps or oscillatory behavior on their respective coboundaries. Thus,

the TV offers a simple way to cluster nodes on the graph by demarcating those nodes v that have high TV on $\delta(\{v\})$. From Eq. (3.6), we conclude that as long as two signals are affinely dependent they produce the same clusters. Apropos our hypothesis, oscillatory WSS is expected to be localized near bifurcations or bottlenecks and high TV nodes are expected to reproduce this behavior.

3.2 GNN model and training

The data consists of 159 simulations of blood flow that are performed on patient based meshes created by RPTU Kaiserslautern-Landau in Landau and the University of Jena [8, 33, 35]. The database is available online [9]. The data set is labeled to form a binary categorization with 61 points labeled as at risk based on an estimated stenosis degree (cf. [10, 11, 34] for estimation details) of higher than 50 %. For cross validation, the data set is split into 75 % training and 25 % testing subsets. Adjacency and edge weight matrices for each simulation are computed and stored in sparse matrix format.

The model is comprised of a single GraphSAGE layer followed by a TV based clustering layer, several GraphSAGE layers, and then a feedforward layer. The implementation details of the clustering layer are now described. The top n nodes, ranked in terms of the total variation, and their immediate neighbors are extracted to form a subset V' of the vertices V by a function we call RankTV.

This creates a subgraph $\mathcal{G}'$ upon adjoining the set of edges $\{[u, v] \in E \mid u \in V'$ and $v \in V'\}$. If this subgraph is disconnected, we find the shortest path between a pair of clusters C_i and C_j as

$$d(C_i, C_j) \coloneqq \min_{u \in C_i, v \in C_j} d(u, v), \tag{3.8}$$

where $d(u, v)$ is the shortest path between nodes u and v. If two clusters are closer than a stipulated path length, they are connected, possibly adding nodes along the shortest path if necessary. Finally, the layer updates the nodes and edges to return the subgraph $\mathcal{G}'$.

The clustering layer produces a compressed localized representation of the entire graph. Compression of the graph is necessary for two reasons: from a computational perspective, this reduces the memory footprint of the algorithm, especially stemming from the computation of gradients on such large graphs. Second, this prioritizes the receptive field of the network on the nodes that are likely to be associated with the generation of stenosis. As a direct consequence of the above, clustering proves to be a remedy for the oversquashing problem that plagues GNNs [2]. Moreover, learning from these clusters would also support our hypothesis that the local structure around areas of high WSS gradient are especially important to the prediction of stroke risk. In Figure 3.1, we show the results of the clustering layer on some simulations.

We now describe the rest of the neural network. The subgraph $\mathcal{G}'$ forms the input to a set of T GraphSAGE convolutions. The output $H^{(T)}$ is pooled using the global mean pool

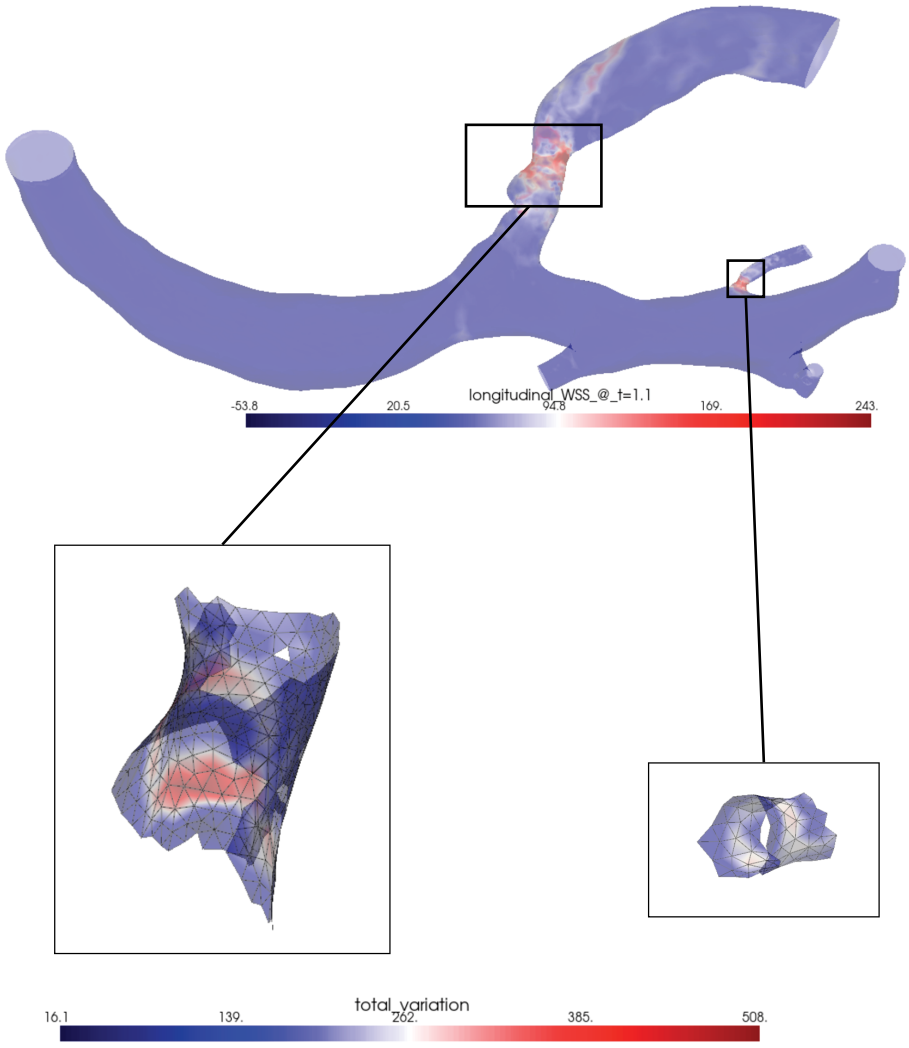


Figure 3.1: Results of the clustering layer. On top is the raw WSS data for a simulation. After message passing, points with highest total variation are clustered and extracted (bottom).

operator (which is equivalent to taking the average of the rows of the matrix $H^{(T)}$) and then transformed into a binary class using a simple feedforward layer. The loss function that is optimized is the BCE between the ground truth label and the label predicted by the GNN.

The pseudocode of the overall model is shown in Algorithm 3.1. Note some of the notation used therein: We denote the matrix of node features at layer k as $H^{(k)}$, which has size $n \times f^{(k)}$, where n is the number of nodes and $f^{(k)}$ is the number of features at layer k. Thus, $H_u^{(k)}$ denotes the row u in this matrix or the features of node u.

Algorithm 3.1 GNN model for stroke risk prediction.

Require: Graph $\mathcal{G} = (V, E, X)$, min path length d
Ensure: Output $\mathbf{y}$
Initialize $H^{(0)} \leftarrow X$
$H^{(1)} \leftarrow \text{GraphSAGE}(\mathcal{G})$
procedure TOTALVARIATIONCLUSTERING($\mathcal{G}$, d)
    $V \supset V' \leftarrow \text{RankTV}(H^{(1)})$ ▷ Total Variation
    $E \supset E' \leftarrow [u, v]\ \forall u \in V'$
    $\{C_1, \ldots, C_M\} \leftarrow \text{FindConnectedComponents}(\mathcal{G}' = (V', E', H^{(1)\prime}))$
    for all Clusters $C_i \in \{C_1, \ldots, C_M\}$ **do**
        $d_{C_i} \leftarrow \min\ \text{ShortestPath}(C_i, C_j)$
        if $d_{C_i} < d$ **then**
            $V' \leftarrow \text{AddNodesOnPath}(d_{C_i})$
        end if
    end for
    return updated graph $\mathcal{G}' = (V', E')$
end procedure
$H^{(T)} \leftarrow \text{GraphSAGE}(\mathcal{G}')$
$\mathbf{h}_G \leftarrow \frac{1}{|V|} \sum_{v \in V} H_v^{(T)}$ ▷ Global Mean Pooling Layer
$\mathbf{y} \leftarrow \texttt{MLP}(\mathbf{h}_G)$ ▷ Output Layer
return $\mathbf{y}$

4 Results

The graph neural network is implemented in Pytorch-Geometric [14]. Results with a graph neural network with optimizers Adam [26] and AdamW [28] are shown in Figure 3.2. The training process was repeated with several configurations of hyperparameters, as listed below, with very similar performance. The starting learning rate varied between 1×10^{-2} and 1×10^{-3} and weight decay set between 1×10^{-5} and 5×10^{-4}. The clustering layer extracts the top 50 nodes with the highest total variation. We set 5 as the maximum path length for which clusters are coalesced. The feature length (width) of the GraphSAGE layers (after clustering) was varied among 16 and 32 and the dropout [40] rate was varied among 30 %, 40 %, and 50 %. In these cases, the model accuracy plateaus between 82.5 % and 87.5 %. For the figure shown in Figure 3.2, the GraphSAGE model has depth 4 and width 16, with a dropout rate of 0.5 and we use a 3-layer MLP (of width 16). The precision recall curve, on the test set, for the model trained with AdamW is shown in Figure 3.3.

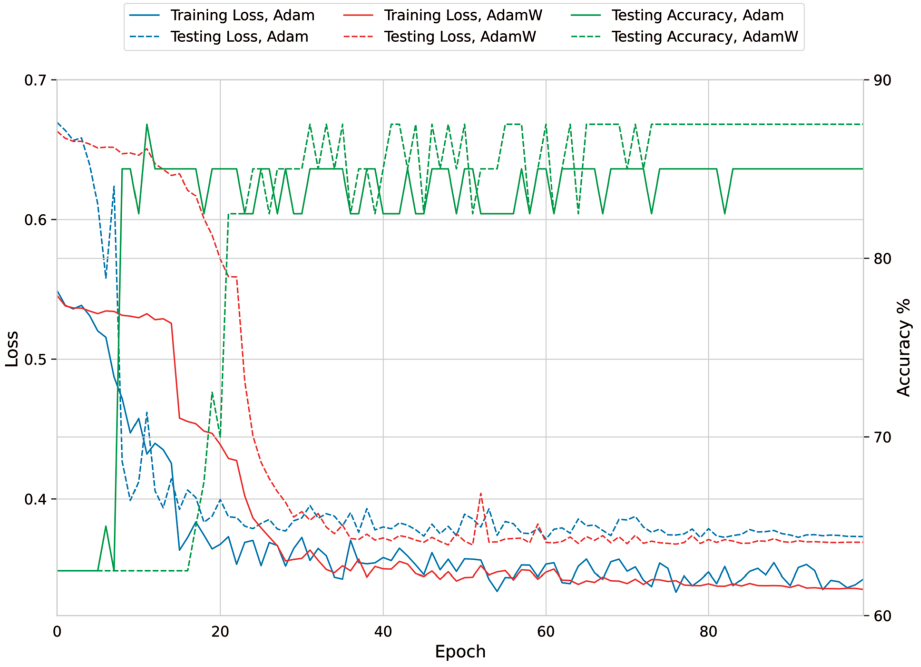


Figure 3.2: Plot of the optimization over 100 epochs (horizontal axis). The axis on the left is for the training and testing loss trends, measured in terms of the BCE loss function (cf. Eq. (3.4)), using optimizers Adam (solid and dashed blue) and AdamW (solid and dashed red). The axis on the right measures the testing accuracy trends for Adam (solid green) and AdamW (dashed green).

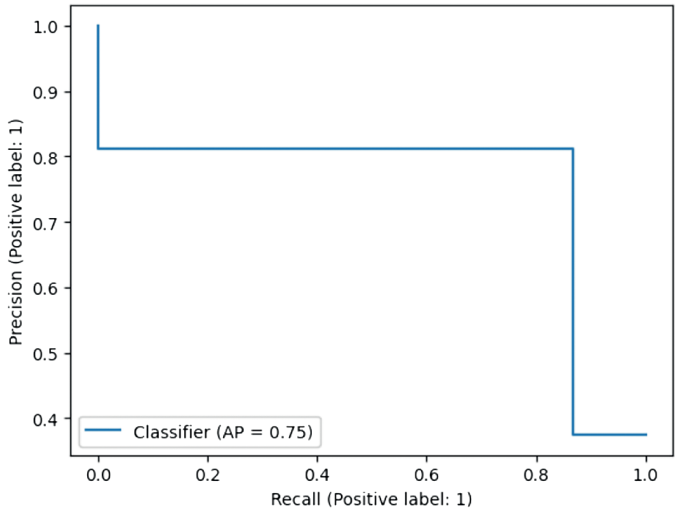


Figure 3.3: Precision-recall curve of the model trained with AdamW on the test set.

5 Conclusion

In this work, we have applied graph neural networks to simulations of blood flow in arterial vessels to predict the risk of stroke. This particular approach is novel with respect to the general methodology from preprocessing the fluid dynamics simulations to the features and the clustering methodology used in the model. The GNN's receptive field is restricted to a subgraph (extracted using node TV) in order to facilitate our hypothesis that the distribution of WSS near bifurcations may be a predictive factor for stroke risk. Incidentally, this has the effect of mitigating the oversquashing effect of GNNs. Formalizing the arguments about TV on graphs, as well as expanding this particular GNN methodology to larger data sets is planned for future work.

Bibliography

[1] S. Y. Adam, A. Yousif, and M. Bakri Bashir. Classification of ischemic stroke using machine learning algorithms. *International Journal of Computer Applications*, 149(10):26–31, 2016.

[2] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv:2006.05205* [cs.LG], June 2020.

[3] M. M. Bronstein, et al. Geometric deep learning: grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478* [cs.LG], Apr. 2021.

[4] R. B. Gabrielsson. Universal function approximation on graphs. *Advances in Neural Information Processing Systems*, 33:19762–19772, 2020.

[5] J. Bruna, et al. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[6] J. Deng, et al. Imagenet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[7] J. M. Dolan, J. Kolega, and H. Meng. High wall shear stress and spatial gradients in vascular pathology: a review. *Annals of Biomedical Engineering*, 41:1411–1427, 2013.

[8] P. Eulzer, et al. Visualizing carotid blood flow simulations for stroke prevention. *Computer Graphics Forum*, 40(3):435–446, June 2021. https://doi.org/10.1111/cgf.14319.

[9] P. Eulzer, et al. A dataset of reconstructed carotid bifurcation lumen and plaque models with centerline tree and simulated hemodynamics. Version 2.0.0. Zenodo, Feb. 2024. https://doi.org/10.5281/zenodo.10695923.

[10] P. Eulzer, et al. Instantaneous visual analysis of blood flow in stenoses using morphological similarity. *arXiv:2403.16653*, 2024.

[11] P. Eulzer, et al. Visualizing carotid stenoses for stroke treatment and prevention, 2023.

[12] E. Falk. Pathogenesis of atherosclerosis. *Journal of the American College of Cardiology*, 47(8S):C7–C12, 2006.

[13] S. K. Feske. Ischemic stroke. *The American Journal of Medicine*, 134(12):1457–1464, 2021.

[14] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. *arXiv:1903.02428*, 2019.

[15] G. B. Folland. *Real Analysis: Modern Techniques and Their Applications*, volume 40. John Wiley & Sons, 1999.

[16] R. Garg, et al. Automating ischemic stroke subtype classification using machine learning and natural language processing. *Journal of Stroke and Cerebrovascular Diseases*, 28(7):2045–2051, 2019.

[17] G. D. Giannoglou, et al. Flow and atherosclerosis in coronary bifurcations. *EuroIntervention*, 6(Suppl J):J16–J23, 2010.

[18] A. Gnasso, et al. In vivo association between low wall shear stress and plaque in subjects with asymmetrical carotid atherosclerosis. *Stroke*, 28(5):993–998, 1997.
[19] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*. IEEE, 2005. https://doi.org/10.1109/ijcnn.2005.1555942.
[20] P. Govindarajan, et al. Classification of stroke disease using machine learning algorithms. *Neural Computing & Applications*, 32(3):817–828, 2020.
[21] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.
[22] J. Berg Hansen and F. M. Bianchi. Total variation graph neural networks. In *International Conference on Machine Learning*, pages 12445–12468. PMLR, 2023.
[23] Mohamed Sobhi Jabal, et al. Interpretable machine learning modeling for ischemic stroke outcome prediction. *Frontiers in Neurology*, 13:884693, 2022.
[24] A. Jung, et al. Semi-supervised learning in network-structured data via total variation minimization. *IEEE Transactions on Signal Processing*, 67(24):6256–6269, Dec. 2019. https://doi.org/10.1109/TSP.2019.2953593.
[25] H. Kashima and A. Inokuchi. Kernels for graph classification. In *ICDM Workshop on Active Mining*, volume 2002, 2002.
[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
[27] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images, 2009.
[28] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2017.
[29] S. Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 1999.
[30] C. R. Maurer and J. M. Fitzpatrick. A review of medical image registration. *Interactive Image-Guided Neurosurgery*, 1:17–44, 1993.
[31] G. J. Minty. Monotone networks. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 257(1289):194–212, 1960.
[32] M. Monteiro, et al. Using machine learning to improve the prediction of functional outcome in ischemic stroke patients. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(6):1953–1959, 2018.
[33] T. Probst, K. Richter, and A. Hundertmark. Utilizing COMSOL® in a workflow to assess stroke risks in a large set of patients carotid arteries, Oct. 2023.
[34] K. Richter, et al. Longitudinal wall shear stress evaluation using centerline projection approach in the numerical simulations of the patient-based carotid artery. In *Computer Methods in Biomechanics and Biomedical Engineering*, pages 1–18, 2023.
[35] K. Richter, et al. Longitudinal wall shear stress evaluation using centerline projection approach in the numerical simulations of the patient-based carotid artery. *Computer Methods in Biomechanics and Biomedical Engineering*, 27(3):347–364, 2024. https://doi.org/10.1080/10255842.2023.2185478, PMID: 36880851.
[36] J. Ruiz-Alzola, et al. Nonrigid registration of 3D tensor medical data. *Medical Image Analysis*, 6(2):143–161, 2002.
[37] H. Saigo, et al. gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75:69–89, 2009.
[38] F. Scarselli, et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, Jan. 2009. https://doi.org/10.1109/tnn.2008.2005605.
[39] K. Spanos, et al. Carotid bifurcation geometry and atherosclerosis. *Angiology*, 68(9):757–764, 2017.
[40] N. Srivastava, et al. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
[41] L. Tomasetti, et al. Machine learning algorithms versus thresholding to segment ischemic regions in patients with acute ischemic stroke. *IEEE Journal of Biomedical and Health Informatics*, 26(2):660–672, 2021.

[42] H. J. Van Os, et al. Predicting outcome of endovascular treatment for acute ischemic stroke: potential value of machine learning algorithms. *Frontiers in Neurology*, 9:784, 2018.
[43] Y.-X. Wang, et al. Trend filtering on graphs. *Journal of Machine Learning Research*, 17(105):1–41, 2016. http://jmlr.org/papers/v17/15-147.html.
[44] J. Wu, et al. Boosting for multi-graph classification. *IEEE Transactions on Cybernetics*, 45(3):416–429, 2014.
[45] J. Zhou, et al. Graph neural networks: a review of methods and applications. *AI Open*, 1:57–81, 2020. https://doi.org/10.1016/j.aiopen.2021.01.001.

Alexander Sikorski, Robert Julian Rabben, Surahit Chewle, and Marcus Weber

Capturing the macroscopic behavior of molecular dynamics with membership functions

Abstract: Markov processes serve as foundational models in many scientific disciplines, such as molecular dynamics, and their simulation forms a common basis for analysis. While simulations produce useful trajectories, obtaining macroscopic information directly from micro state data presents significant challenges. This paper addresses this gap by introducing the concept of membership functions being the macro states themselves. We derive equations for the holding times of these macro states and demonstrate their consistency with the classical definition. Furthermore, we discuss the application of the ISOKANN method for learning these quantities from simulation data. In addition, we present a novel method for extracting reaction paths from simulations based on the ISOKANN results and demonstrate its efficacy by applying it to simulations of the μ-opioid receptor. With this approach, we provide a new perspective on the analysis of macroscopic behavior of Markov systems.

Keywords: Macro states, membership functions, reaction rates, holding times, Koopman, ISOKANN method, shortest path, reaction path

MSC 2020: 68T07, 60J35, 60G05, 62A01, 62F40, 47N30, 47N60, 47B33

1 Introduction

Many physical processes are best understood as autonomous Markov processes (e. g., [8]). A common mathematical model for them are stochastic differential equations, which allow to predict the probability for the future evolution of the system. Markovianity means that this probability is just depending on the initial condition of the system.

Acknowledgement: The research of A. Sikorski was funded by the DFG through the CRC 1114 "Scaling Cascades in Complex Systems" (project B03). The research of R. J. Rabben was funded by the NHR Graduate School of the NHR Alliance. The research of S. Chewle was funded by the BMBF through the project CCMAI (funding code 01GQ2109A).

Alexander Sikorski, Robert Julian Rabben, Surahit Chewle, Marcus Weber, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, e-mails: sikorski@zib.de, rabben@zib.de, chewle@zib.de, weber@zib.de, https://orcid.org/0000-0001-9051-650X, https://orcid.org/0000-0002-8048-7554, https://orcid.org/0000-0002-1428-349X, https://orcid.org/0000-0003-3939-410X

https://doi.org/10.1515/9783111376776-004

Solving the initial value problem is referred to as *simulation of the system*. The states of the generated trajectory are denoted as the *micro states* of the system.

As an illustration, consider the field of biomolecular system simulation [7]. Here, a common mechanism being studied is the transformation of an inactive micro state of a receptor protein into an active state (by accordingly changing the coordinates of its atoms). The process can be mathematically modeled by the overdamped Langevin dynamics, a stochastic differential equation for the 3D coordinates of each atom in the protein and the solvent [6]. Typically these are analyzed by simulation of the model, resulting in trajectories in a high-dimensional space.

However, often the main focus of interest lies on the *macro state behaviour* of the system. A possible question to answer would be: What is the mean first passage time for an inactive receptor protein to become active? The two macro states in this regard are denoted as "inactive" versus "active." At first glance, macro states S are subsets of the set of micro states Ω. In general, when given a starting set $S \subset \Omega$ of the system we want to know, how long on average the process stays in this set [20]. How long does a trajectory starting in the inactive macro state remains there before leaving it, that is, switching to the active macro state? The answer is given by the mean holding time $t_{\mathrm{mh}}(\boldsymbol{x})$ defined by the integral:

$$t_{\mathrm{mh}}(\boldsymbol{x}) = \int_0^\infty p_S(\boldsymbol{x}, t)\, dt,$$

where $\boldsymbol{x}$ denotes the micro state of the system (the starting point $x(0) = \boldsymbol{x}$ of the trajectory) and the function $p_S(\boldsymbol{x}, t)$ denotes the probability that a trajectory is still in set S and has never left it during the whole time interval $[0, t]$. Formally, $p_S(\boldsymbol{x}, t) = P_{\boldsymbol{x}}(T_S > t)$ is the holding probability of S conditioned on starting at the micro state $\boldsymbol{x}$ with T_S being the first exit time of S. t_{mh} (also called mean first exit time) is then given by $t_{\mathrm{mh}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{x}}[T_S]$.

If $t_{\mathrm{mh}}(\boldsymbol{x})$ denotes the expected time until the system reacts, then

$$\boldsymbol{r}(\boldsymbol{x}) = -\nabla t_{\mathrm{mh}}(\boldsymbol{x}) = -\nabla \int_0^\infty p_S(\boldsymbol{x}, t)\, dt$$

points into the direction where this time decreases the most. This can be seen as the micro-state-dependent reaction path direction $\boldsymbol{r}(\boldsymbol{x})$.

There is a conceptual problem now. We want to know the mean holding time for the macro state and not for every single micro state, but t_{mh} is a function of the micro states—the mean holding time in S depends on the starting point. Speaking of the "mean holding time of the macro state S" would only make sense if it was independent of the micro-states position inside that macro state, that is, if there was a set $S \subset \Omega$ and an exit rate $c_1 > 0$ allowing for a separation of the type

$$p_S(\boldsymbol{x}, t) = \mathbb{1}_S(\boldsymbol{x})\, \mathrm{e}^{-c_1 t}, \tag{4.1}$$

where $\mathbb{1}_S$ is the indicator function of the set S. In Markov state modeling, it is often asked for a decomposition of the state space Ω into subsets S which can be identified as "macro states" [11]. The separation approach (4.1) is a prerequisite for Markovianity of the macro state S and is approximately valid, if "leaving the set S" is a very rare event compared to the mixing within S. The constant $c_1 > 0$ corresponds to the exit rate of S, because in this case $t_{\mathrm{mh}}(\boldsymbol{x}) = \mathbb{1}_S(\boldsymbol{x})\,\frac{1}{c_1}$ is independent of the choice of the starting point in S. This decomposition, however, is not possible in general and corresponds to instantaneous transitions, which do not provide reaction paths inside S.

Classically, the computation of (micro-state) mean holding times in the case of an S-based theory is provided by solving a partial differential equation [9]:

$$\mathcal{L}t_{\mathrm{mh}}(\boldsymbol{x}) = -1,$$

for all $\boldsymbol{x} \in S$ with the boundary condition $t_{\mathrm{mh}}(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \notin S$. In this equation, the differential operator $\mathcal{L}$ is the infinitesimal generator of the Koopmann operator of an autonomous Markov process and the equation essentially prescribes the mean holding time to decrease by one time unit per time unit until hitting the boundary. If the decomposition $t_{\mathrm{mh}}(\boldsymbol{x}) = \mathbb{1}_S(\boldsymbol{x})\,\frac{1}{c_1}$ were valid, then this equation would read

$$\mathcal{L}\mathbb{1}_S(\boldsymbol{x}) = -c_1\mathbb{1}_S(\boldsymbol{x}). \tag{4.2}$$

Under the assumption of an ergodic system (e. g., for nondegenerate diffusion) the only set-based solutions are $S = \Omega$ with $c_1 = 0$ (the process never leaves Ω) or $S = \emptyset$. Clearly, these trivial solutions are not of any use. How can we solve this conceptual problem?

When deriving exit rates for biomolecular processes, then the macro states of this system cannot be rigorously described as subsets S in micro-state space. Rather, transitions are gradual from something that is more inactive to something that can be described as more active. We propose replacing the indicator function $\mathbb{1}_S$ in (4.1) by a membership function $\chi : \Omega \to [0,1]$, which quantifies how much a micro state $\boldsymbol{x} \in \Omega$ belongs to the starting macro state. The theoretical background of our article, therefore, starts with the more general separation of space and time via

$$p_\chi(\boldsymbol{x}, t) = \chi(\boldsymbol{x})\,\mathrm{e}^{-c_1 t}. \tag{4.3}$$

By definition, it satisfies the exponential decay of p_χ in t and at $t = 0$ the probability to be assigned to the starting state is given by χ. Therefore, the χ functions with the smallest exit rates c_1 are the most persistent observables or measurements on the system satisfying Markovianity. Considering that a macro state should be a function of the micro states and exhibit orderly, in our case Markovian, dynamics this gives rise to the fundamental idea of identifying the macro state with the membership function: χ *is* the macro state and the macro state is given in and through χ. Generalizing the classical set-based description to that by a χ-function allows us to obtain nontrivial macro states, which have

proper exit rates and, therefore, allow for a closed description of their dynamics. In this regard, our ansatz can be seen as a consequence of the desire for a coarse-grained description that is exactly reproducing the slowest timescales, which is not possible with set-based decompositions.

At this point, we have not provided a method to compute χ yet. χ cannot be chosen arbitrarily. In order to preserve the Markovian long term behavior of the system, a projection of the micro system to two different macro states has to be based on an invariant subspace of $\mathcal{L}$ [19, 21]. Further, requiring a decomposition into two complementary (i. e., such that they add up to one) macro states, the simplest nontrivial solution is given by

$$\mathcal{L}\chi(\boldsymbol{x}) = -c_1\chi(\boldsymbol{x}) + c_2\big(1-\chi(\boldsymbol{x})\big) \tag{4.4}$$

with $c_1, c_2 > 0$, and χ, $1-\chi$ describing the two macro states. These can be computed using PCCA+ [2] or ISOKANN [12, 16] and form an invariant subspace of $\mathcal{L}$, which also guarantees long-term consistency with the original dynamics [21]. Whereas any solution to (4.4) leads to a dynamically closed macroscopic description, we will in general aim for solutions with small rates c_1, c_2, which represent temporally stable macro states.

In order to derive (4.4), one has to understand how the Markov property of a projected Markov process can be completely preserved. It means that projection and propagation of the system have to commute. Thus, the projection has to be based on an invariant subspace of $\mathcal{L}$. The constant function is an eigenfunction of $\mathcal{L}$. Thus, a linear combination of a constant function and of a further eigenfunction leads to a feasible membership function χ. If we apply $\mathcal{L}$ to such a linear combination, we arrive at (4.4). For the whole derivation, we refer to [20].

In Section 2 of this paper, we demonstrate that our macro state formalism based on the χ-function is consistent with the traditional set-based method: We show that when the χ-functions approach indicator functions the χ based equations reduce to the classical ones. We further motivate its physical meaning by giving a path-based interpretation of the resulting holding probabilities in terms of the Feynman–Kac formula.

In Section 3, we will suggest an approach to apply these theoretic results to extract reaction paths from a set of given samples, before demonstrating its application to a molecular system given by the μ-opioid receptor in Section 4.

2 Theory of macroscopic exit rates

In the following, a consistent theory about macroscopic quantities based on a membership function $\chi : \Omega \to [0,1]$ is derived. More precisely, the following *quantities of interest* are discussed:

– the definition of a macro state via $\chi(\boldsymbol{x})$,
– the corresponding position-independent exit rate c_1,

– the mean holding time $t_{\mathrm{mh}}(\boldsymbol{x})$, and
– the reaction direction $\boldsymbol{r}$ proportional to the gradient $-\nabla\chi$.

Note that computing reaction directions $\boldsymbol{r}$ is possible for sets S, too, as described in the introduction. However, for sets we do not get this simple gradient form $-\nabla\chi$. The theory in this section is largely based on [20, 3]. We will extend the theory with statements about consistency and use it to support of the interpretation of χ as macro states, as explained in the introduction. The computation of χ will play an important role in our bio-molecular example; see Section 4.

2.1 Defining macro states via membership functions

The starting point in the introduction is that $p_\chi = \chi \mathrm{e}^{-c_1 t}$ would be a way to define holding times of macro states from a conceptual point of view to be able to interpret c_1 as an exit rate. The choice of χ is not arbitrary, but motivated by the necessity of an invariant subspace projection leading to (4.4). At this point, it is not yet shown that p_χ when using the solution χ of (4.4) is consistent with the stochastic meaning of a holding probability. We will now demonstrate that it converges to the established definition if χ becomes the indicator function of a set.

We start by recalling some basic definitions. Let the state space Ω of a molecular system comprising N atoms be given as $\Omega = \mathbb{R}^{3N}$ where the position of each individual atom is described by three Cartesian coordinates. Let $\rho(\boldsymbol{x}, t) : \Omega \times \mathbb{R} \to [0, 1]$ denote the probability density distribution of states of the nonlinear stochastic dynamics at time t. More precisely, the dynamics is given as a Markov process for which an infinitesimal generator $\mathcal{L}^* : L^1(\Omega) \to L^1(\Omega)$ (a differential operator, e. g., the Fokker–Planck operator) can be constructed that captures this time-dependent stochastic process. This operator is linear and describes the infinitesimal propagation of $\rho(\boldsymbol{x}, t)$:

$$(\mathcal{L}^*\rho)(\boldsymbol{x}, t) = \frac{d}{dt}\rho(\boldsymbol{x}, t). \tag{4.5}$$

It also gives rise to its adjoint generator $\mathcal{L} : L^\infty(\Omega) \to L^\infty(\Omega)$, which propagates observables instead of state densities. The partial differential equation (4.4) defining χ is formulated in terms of this adjoint. In this regard, χ can also be interpreted as an observable, that is, the measurement of the macro state. To allow for their interpretation as holding probability, we are interested in solutions $\chi(\boldsymbol{x}) : \Omega \to [0, 1]$ with corresponding constants $c_1 > 0, c_2 > 0$.

Assume such χ and c_1, c_2 are given. We then define the χ-holding probability as

$$p_\chi(\boldsymbol{x}, t) := \chi(\boldsymbol{x})\mathrm{e}^{-c_1 t} \tag{4.6}$$

such that (4.4) becomes

$$\mathcal{L}^* p_\chi = -c_1 p_\chi + c_2 p_\chi \frac{1-\chi}{\chi}. \tag{4.7}$$

Rearranging results in

$$\mathcal{L}^* p_\chi - c_2 p_\chi \frac{1-\chi}{\chi} = -c_1 p_\chi. \tag{4.8}$$

Due to the relationship

$$\frac{\partial}{\partial t} p_\chi = \frac{\partial}{\partial t} \chi(\boldsymbol{x}) \mathrm{e}^{-c_1 t} = -c_1 \chi(\boldsymbol{x}) \mathrm{e}^{-c_1 t} = -c_1 p_\chi. \tag{4.9}$$

Equation (4.8) can also be represented as follows:

$$\mathcal{L}^* p_\chi - c_2 p_\chi \frac{1-\chi}{\chi} = \frac{\partial}{\partial t} p_\chi. \tag{4.10}$$

The solution of the partial differential equation (4.10) together with the initial condition

$$p_\chi(\boldsymbol{x}, 0) = \chi(\boldsymbol{x}) \mathrm{e}^{-c_1 \cdot 0} = \chi(\boldsymbol{x}) \tag{4.11}$$

can be given in terms of the Feynman–Kac formula [20, 3, 5]:

$$p_\chi(\boldsymbol{x}, \tau) = \mathbb{E}\left[\chi(\boldsymbol{x}_\tau) \cdot \exp\left(-c_2 \int_0^\tau \frac{1-\chi(\boldsymbol{x}_t)}{\chi(\boldsymbol{x}_t)} dt \right) \,\middle|\, \boldsymbol{x}_0 = \boldsymbol{x} \right]. \tag{4.12}$$

This representation allows us to interpret the solution as an expectation over realizations of the stochastic process and, therefore, builds the bridge from an abstract definition to its interpretation as a probability.

To this end, let us consider the case where $\chi \approx \mathbb{1}_S$. The expectation is to be taken over all trajectories starting in $\boldsymbol{x}_0$. Once any such trajectory leaves the set S the integral becomes infinite and exponential function evaluates to 0. Otherwise, the exponential stays 1, as well as $\chi(\boldsymbol{x}_\tau) = 1$. We therefore recover the definition of classical holding probability $p_\chi(\boldsymbol{x}, \tau) = p_S(\boldsymbol{x}, \tau)$ in (4.1), that is, the probability to stay in S for time τ at least; see also (9) and (10) in [20], as well as (3.31) in [15].

We summarize this result in the following proposition.

Proposition 4.1. *Let $\chi \approx \mathbb{1}_S$ be a solution to* (4.4)*. Then the χ-holding probability approximates the classical holding probability of S:*

$$p_\chi(\boldsymbol{x}, t) \approx p_S(\boldsymbol{x}, t).$$

With regard to this interpretation, $p_\chi(\boldsymbol{x}, t)$ is seen as the holding probability of the macro state χ. Due to the separated term $\mathrm{e}^{-c_1 t}$ in (4.6), the holding probability decreases exponentially with the decay constant c_1. This means that c_1 is the exit rate from χ. Since

the function value of $p_\chi(\boldsymbol{x}, 0) = \chi(\boldsymbol{x})$ is interpreted as a holding probability, it is necessary that χ can only take values in the interval $[0, 1]$.

The time integral over the holding probability is the mean holding time $t_{\mathrm{mh}}(\boldsymbol{x})$, which is proportional to the inverse of the exit rate c_1:

$$t_{\mathrm{mh}}(\boldsymbol{x}) = \int_0^\infty p_\chi(\boldsymbol{x}, t)\, dt = \int_0^\infty \chi(\boldsymbol{x}) \mathrm{e}^{-c_1 t}\, dt = \lim_{t\to\infty} \left[-\frac{1}{c_1} \chi(\boldsymbol{x}) \mathrm{e}^{-c_1 t} \right]_0^t = \frac{1}{c_1} \chi(\boldsymbol{x}). \tag{4.13}$$

The mean holding time immediately leads to a definition of a reaction direction: Following the gradient of t_{mh} increases the time until "a reaction takes place." Therefore, by defining the reaction direction $\boldsymbol{r} : \Omega \to \mathbb{R}^{3N}$,

$$\boldsymbol{r}(\boldsymbol{x}) = -\nabla t_{\mathrm{mh}}(\boldsymbol{x}) = -\frac{1}{c_1} \nabla \chi(\boldsymbol{x}) \propto -\nabla \chi(\boldsymbol{x}), \tag{4.14}$$

we obtain a vector field along which the mean holding time decreases uniformly and which is proportional to $\nabla\chi$. This also means that χ itself can be understood as an order parameter, that is, a reaction coordinate for the system. Note that we obtain this time independent result only as a consequence of the initial time separation ansatz for p_χ. By integrating curves tangential to $\boldsymbol{r}$, one can obtain reaction paths in Ω. In Section 3, we will make use of the order parameter interpretation to subsample a representative reactive path from a given pool of simulation data.

The possibility of calculating these quantities of interest from (4.4) is a motivation to develop an efficient method for solving this equation. We will now show how to express these quantities in terms of the Koopman operator, before discussing ISOKANN, an algorithm for their computation.

2.2 Membership functions from Koopman operator

So far, the description of χ was based on the infinitesimal generator $\mathcal{L}$ but a suitable analytical solution of the corresponding partial differential equation (4.4) is not available. However, it is possible to transform (4.4) into an equation for which a constructive solution is possible. We will now show how we can similarly formulate it in terms of the Koopman operator $\mathcal{K}$ and how we can switch between the formalisms and work out the relation between χ function and eigenfunctions of $\mathcal{K}^\tau$. The problem of actually computing χ will then be addressed in the next subsection.

The Koopman operator $\mathcal{K}^\tau$ is the time-integral or solution operator of $\mathcal{L}^*$ for some lag-time $\tau > 0$ and can be formally defined as $\mathcal{K}^\tau = \mathrm{e}^{\tau\mathcal{L}^*}$. It can also be defined by its action on observable functions $f : \Omega \to \mathbb{R}$:

$$(\mathcal{K}^\tau f)(\boldsymbol{x}) := \mathbb{E}[f(x(\tau)) \mid x(0) = \boldsymbol{x}], \tag{4.15}$$

where the expectation is taken over independent realizations x of the process, starting in $x(0) = \boldsymbol{x}$. It can be understood as the expected measurement $\mathcal{K}^\tau f$ of an observable f after a lag time τ. Being an expectation value its action can be approximated using Monte-Carlo estimates over simulations, making it particularly suitable for applications.

To transform (4.4) into an equation for $\mathcal{K}^\tau$, we substitute

$$\alpha = c_1 + c_2 \tag{4.16}$$

to arrive at the following equation:

$$\mathcal{L}^*\chi = -\alpha\chi + c_2. \tag{4.17}$$

This shows that $\mathcal{L}$ acts as a shift-scale operator on χ if and only if χ solves (4.4). Making use of the formal exponential representation of $\mathcal{K}^\tau$ and its series expansion, one obtains [20, 3]

$$\mathcal{K}^\tau\chi = \mathrm{e}^{-\tau\alpha}\chi + \frac{c_2}{\alpha}(1 - \mathrm{e}^{-\tau\alpha}). \tag{4.18}$$

Setting

$$\gamma_1 = \mathrm{e}^{-\tau\alpha}, \quad \gamma_2 = \frac{c_2}{\alpha}(1 - \gamma_1), \tag{4.19}$$

this becomes

$$\mathcal{K}^\tau\chi = \gamma_1\chi + \gamma_2. \tag{4.20}$$

So, just as with equation (4.17), $\mathcal{K}^\tau$ acts as a shift-scale if and only if χ is a solution to (4.4).

Noting that $\mathcal{K}^\tau\mathbb{1}, = \mathbb{1}$, we further see that $f := \chi - \frac{c_2}{\alpha}$ is an eigenfunction of $\mathcal{K}^\tau$ with eigenvalue γ_1:

$$\mathcal{K}^\tau\left(\chi - \frac{c_2}{\alpha}\right) = \gamma_1\chi + \frac{c_2}{\alpha}(1 - \gamma_1) - \frac{c_2}{\alpha} = \gamma_1\left(\chi - \frac{c_2}{\alpha}\right). \tag{4.21}$$

These findings are summarized in the following proposition.

Proposition 4.2. *Let the parameters $c_1, c_2, \alpha, \gamma_1, \gamma_2$ satisfy their relations above. The following are equivalent:*

- *χ solves the ISOKANN problem* (4.4)*.*
- *$\mathcal{L}$ acts as a shift-scale on χ with scale $-\alpha$ and shift c_2.*
- *$\mathcal{K}^\tau$ acts as a shift-scale on χ with scale γ_1 and shift γ_2.*
- *$\chi - \frac{c_2}{\alpha}$ is an eigenfunction of $\mathcal{K}^\tau$ with eigenvalue γ_1.*

The above identities allow us to switch between the infinitesimal generator and Koopman framework. In particular, we can compute the exit rate c_1 from the Koopman parameters γ_1 and γ_2:

$$\alpha = -\frac{\ln \gamma_1}{\tau}, \quad c_2 = \frac{\alpha \gamma_2}{1-\gamma_1}, \quad c_1 = \alpha - c_2. \tag{4.22}$$

This allows to estimate the respective constants, for example, the exit rate c_1, by evaluating χ and $\mathcal{K}^\tau \chi$ at sample points $\boldsymbol{x} \in \Omega$ and solving the linear regression problem (4.20).

2.3 ISOKANN for computing membership functions

The ISOKANN (Invariant subspaces of Koopman operators learned by a neural network) method [12, 17, 16] is a fixed-point iteration, which combines the use of a neural network for representing the high-dimensional χ function with the Koopman formalism to enable its training on simulation data.

On convergence, it returns χ that solves (4.4), (4.17), and (4.20). Solving the partial differential equations involving $\mathcal{L}^*$ directly is not feasible due to the high dimensionality of molecular systems. Molecular simulations on the other hand enable us to estimate the action of $\mathcal{K}^\tau$ on an observable. For this reason, attempts to solve (4.20) by using the action of $\mathcal{K}^\tau$ for the calculation of $(\mathcal{K}^\tau \chi_i)(\boldsymbol{x})$, where χ_i is the ith iterate of χ and $\boldsymbol{x}$ is a training point. It approximates the expectation value

$$(\mathcal{K}^\tau \chi_i)(\boldsymbol{x}) := \mathbb{E}\big[\chi_i(x(\tau)) \mid x(0) = \boldsymbol{x}\big], \tag{4.23}$$

by a Monte Carlo estimate over trajectory simulations x starting in different starting points $\boldsymbol{x}$. The next iterate is then given by the shift-scaled $\mathcal{K}^\tau \chi_i$:

$$\chi_{i+1} := \frac{\mathcal{K}^\tau \chi_i - \min(\mathcal{K}^\tau \chi_i)}{\|\mathcal{K}^\tau \chi_i - \min(\mathcal{K}^\tau \chi_i)\|_\infty}, \tag{4.24}$$

which is motivated by inverting the shift and scale of (4.20), such that the solution χ to (4.20) is indeed a fixed point. The initial guess χ_0 is chosen randomly. ISOKANN is based on the power method, an iterative method used to obtain the dominant eigenfunction of a linear operator. In ISOKANN, additional scaling and shifting in each iteration ensures that it does not converge to the constant function, but against the membership function $\chi(\boldsymbol{x}) : \Omega \to [0,1]$ [17]—similar to targeting the second eigenvalue in the inverse power method. In order to represent the iterates χ_i, we approximate them by a neural network. The equality assignment in the iteration (4.24) thus becomes a supervised learning problem at data points $\boldsymbol{x}$ with labels given by the corresponding right hand side of (4.24) evaluated on the previous generation of the network. The iterations are then terminated by a stopping criterion, which can be either a high correlation coefficient $(\chi_i(\boldsymbol{x}_n), \chi_{i+1}(\boldsymbol{x}_n))_n$ or a small empirical loss $\|\chi_i - \chi_{i+1}\|_2$, indicating that we found an approximate solution to (4.20).

Assuming infinite data and perfect representation by the neural network, this iteration indeed converges to the solution χ of (4.20) [17] with the smallest rate c_1. It is spanned by the constant eigenfunction with eigenvalue 0 and the second eigenfunction

with eigenvalue closest to 0. This solution is unique, if the corresponding eigenvalue is simple. In practice however, for a cluster of low-lying eigenvalues, this routine can result close to one of multiple possible membership functions (each representing one of these slower processes). In that case, the procedure can be repeated and the resulting membership functions can be used to construct an invariant subspace of $\mathcal{K}$.

One of ISOKANN's main benefits is that it avoids discretizing the state space, which is crucial for the application to high-dimensional systems and avoiding the curse of dimensionality [12]. In the ISOKANN algorithm, it is possible to train artificial neural networks by collecting many short-term trajectories of simulation length τ from different starting points in high-dimensional spaces. Thus, ISOKANN can be applied to many independent short-time trajectories or it can even decide where in Ω to enrich simulation data [17]. In our illustrative example, we will apply it to a small number of medium-length trajectories. The resulting χ function will then be used to subsample a reactive path from the data, which leads us to the next section.

3 Extracting reaction paths from simulations

Once we obtained a χ function from ISOKANN, we might be tempted to directly compute a reaction path following the gradient of χ as in (4.14). This, however, is problematic as the neural network approximation is good only in the region of sufficient training data. The gradient of χ, however, could point away from this region, accumulating more and more approximation errors and quickly lead to unobserved unphysical states. One might solve this problem by projecting back to the physical regime, for example, via energy minimization. This, however, requires access to the original systems potential/forces and is not only computationally expensive but also involved from an implementation perspective.

We now propose an alternative method, which can be applied as a post-processing step to already sampled simulation data without requiring further information about the system. Applied as a post-processing step to molecular simulations, together with ISOKANN, the subsampled reaction path provides researchers with a direct step-by-step representation of the slowest process (which is contained in the simulation data) by identifying this process and filtering out the intermediate fluctuations.

To this end, we understand the learned χ values as an order parameter for a set of simulation data in Ω. As shown in (4.13), the membership values are interpreted to be proportional to mean holding times of a macro state. The level sets of the function $\chi(\boldsymbol{x})$ in this regard correspond to micro states $\boldsymbol{x}$ which "take place simultaneously" in this newly defined time axis. By interpreting this mean-holding-time as a "temporal" parameter, we can extract macroscopic reaction paths from simulation data. Replacing the time from the simulation by χ furthermore allows us to incorporate data of different simulations, treating them merely as χ ordered data points, and thus allowing for higher resolution of the results.

The core problem of choosing a path through the data samples consists of balancing the progress in the reaction direction, $\nabla\chi$, versus spatial movement on the level sets of χ. More formally we look for an ordered list $I \subset J = \{1, \dots, N\}$ of samples from simulation points $X_J = (\boldsymbol{x}_j)_{j\in J}$ in Ω, such that $\chi(\boldsymbol{x}_j)$ is an increasing sequence with smooth spatial transitions, that is, without large deviations $\boldsymbol{x}_{j+1} - \boldsymbol{x}_j$. To this end, we will model the spatial movement as a Brownian motion through the time-parameter χ.

For the classical Brownian motion $dX_t = \sigma\, dW_t$, the probability of obtaining a specific set of samples, conditioned on the sampling time points t_i, is given by the finite-dimensional distribution [9] (in our case we assume $\boldsymbol{b} \equiv 0$ and $\sigma(\boldsymbol{x}) \equiv \sigma$ as well as $t = \chi$):

$$p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n | t_1, \dots, t_n) = \left(\prod_{i=1}^{n-1} (2\pi\sigma^2 \Delta t_i)^{-d/2} \right) \exp\left(- \sum_{i=1}^{n-1} \frac{\|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|^2}{2\sigma^2 \Delta t_i} \right). \tag{4.25}$$

This formula is typically used to obtain the probability given a specific set of $\Delta t_i = t_{i+1} - t_i$. In our case, we will use it to also compare different sampling times t_i by setting $p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n | t_1, \dots, t_n) = p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n, t_1, \dots, t_n)$, which allows to balance temporal with spatial jumps. This can be justified from a Bayesian perspective as prescribing a uniform prior on the number and length of time intervals (as we have no preference for specific Δt values to appear in the solution) although further investigation of this view may be warranted. The parameter σ plays the role of a smoothing parameter, balancing the likelihood of jumps in space or time. A high σ allows for more erratic jumps over short time spans while a lower σ favors spatially closer jumps possibly necessitating longer time-spans, as illustrated in Figure 4.1.

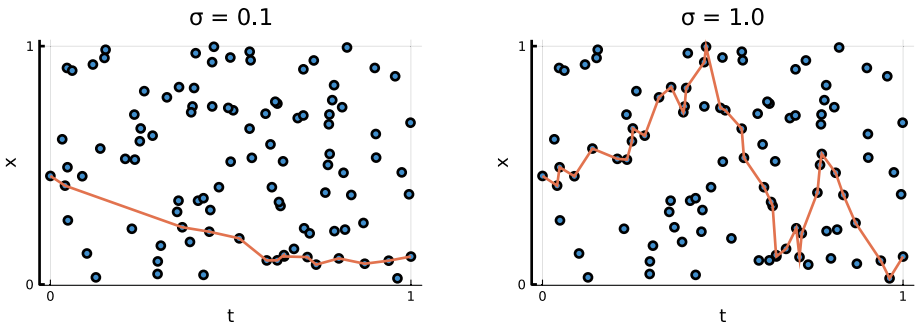


Figure 4.1: Illustration of the maximum likelihood path (4.25) on 100 points distributed uniformly in space and time. For lower σ values (left), the path prefers less jumps with small spatial displacement, while for higher σ (right) the path goes through more points at the cost of more erratic movement. Note that this is just an illustration of the maximum likelihood path with synthetic data in classical time t (which in our application will be replaced by χ).

We now continue to find the maximum likelihood path through our sampling data by replacing the classical time parameter with the samples χ value, such that $\Delta t_i = \chi(x_{i+1}) - \chi(x_i)$.

Taking the logarithm allows us to transform (4.25) into a sum of log probabilities, $\log p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i=1}^{n} \log p(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$ with

$$\log p(\boldsymbol{x}, \boldsymbol{y}) = \log\left((2\pi\sigma^2(\chi(\boldsymbol{y}) - \chi(\boldsymbol{x})))^{-d/2}\right) - \frac{\|\boldsymbol{y} - \boldsymbol{x}\|^2}{2\sigma^2(\chi(\boldsymbol{y}) - \chi(\boldsymbol{x}))}. \tag{4.26}$$

Finding the maximal likelihood path then corresponds to solving the shortest path problem from a (set of) point(s) $\chi(\boldsymbol{x}_i) \approx 0$ to a (set of) point(s) $\chi(\boldsymbol{x}_i) \approx 1$ with edge distance e_{ij} between two points (nodes)

$$e_{ij} = \begin{cases} -\log p(\boldsymbol{x}_i, \boldsymbol{x}_j) & \text{if } \chi(\boldsymbol{x}_i) < \chi(\boldsymbol{x}_j), \\ \infty & \text{otherwise,} \end{cases} \tag{4.27}$$

where transitions forward in time (i. e., increasing χ-value) are enforced by the corresponding ∞ weight. The shortest path problem can be solved with the Bellman–Ford algorithm [4].

Of course, the assumption of uniform Brownian motion for the underlying dynamics is far-fetched for an actual molecular system. However, with only finite simulation data, the locally possible jumps will be dictated mainly by the available data which already incorporates the physical drift. The Brownian assumption introduces only a small bias, which is largely negligible for the choice of paths in the temporal, that is, χ direction, as any larger deviations from the Brownian distribution will already be reflected in the available data.

We believe this can be improved in cases where the acceleration (force and masses) and the diffusion coefficient σ of the dynamics are known. This, however, is not straightforward, as we have replaced the ordinary time with χ, which requires some projection of the stochastic process.

In this regard, our proposed algorithm can be understood as a simple heuristic filtering method to obtain a smooth path through already provided samples along the learned reaction coordinate from $\chi = 0$ to $\chi = 1$ (or vice versa). The result is the most-likely path (under the Brownian assumption) between the extremal conformations identified by χ. Being composed of the actually simulated, hence physically relevant, data it provides a smooth transition through the identified slowest process indexed by the mean holding time of the corresponding macro state.

4 Illustrative example: opioid receptor

To illustrate the added value of χ computation, we will show an molecular dynamics (MD) example, which is part of a pharmaceutical project. We first learned the χ

function from molecular simulation trajectories and applied the described reaction path extraction along χ to a high-dimensional molecular system consisting of 4,734 atoms. The application background is given by understanding pain relief using opioids. Strong painkillers like morphine and fentanyl act upon a special type of receptor in the body known as the (MOR). This receptor is part of the family of opioid receptors and is a G-protein coupled receptor found in various parts of the body such as brain, spinal chord, and gastrointestinal tract [10]. The indiscriminate activation of the MOR across the whole body is one of the causes of severe side-effects of this family of strong painkillers [1].

However, it is proven that chemical changes at site of inflammation cause the creation of a micro environment [13]. The knowledge about the local micro environment can be used to design peripherally restricted strong pain killers with potentially less side-effects [18]. Different micro environments may lead to different dynamics of the MOR. One possible chemical change of the MOR in inflamed tissue postulates the formation of disulfide bonds as the concentration of reactive oxygens species goes up.

4.1 Algorithmic details

Our input data is taken from 10 different simulations of the MOR. After the primary proximity analysis on the active structure of the MOR (Protein Data Bank (PDB) ID 8EF5), a disulfide bond was introduced in the inactive structure of the MOR in between location CYS159 and CYS251 (Protein Data Bank ID 7UL4) [22, 14]. Ten simulations (with explicit water and with a lipid-bilayer for the MOR) were run. Each simulation spans an interval of 100 nanoseconds, totaling to 1 microsecond.

After simulation, pairwise distances over all α-carbons that can be observed to be closer then a threshold ($d_{\max} = 12$ Å) at least once over the simulation time (normalized to mean 0 and standard deviation 1) serve as input features of the corresponding neural network of the ISOKANN algorithm [16]. The action of the Koopman operator in (4.24) is estimated with one sample each forward and backward along the time axis, justified by the reversibility of the system. This is used to train the χ-function. With taking forward and backward steps, it is avoided that χ has all its mass in the terminal point. A unidirected trajectory, that is, if only forward or only backward steps were taken, would "transport" the χ values along the trajectory to the final point in the estimation of the Koopman operator: The Koopman estimation of each point in (4.24) would simply be the χ value at the trajectories predecessors point. Over time, all points would attain the value of the first point, whereas the shift-scale would enforce the last point to remain distinct.

The neural network is a multilayer perceptron with 3 fully connected hidden layers of size $(6161, 336, 18)$ with sigmoid activation functions and a single linear output neuron. For the optimization, we use ADAM with a learning rate of $\eta = 1\text{e}{-4}$ with a $L2$ regularization of $\lambda = 1\text{e}{-2}$ and a mini-batch size of 128. After training χ for 30,000 iterations, we concluded convergence based on the plateauing of the mean squared error

of the ISOKANN residuals at around 5e−4. The described shortest path is extracted with $\sigma = 0.7$ resulting in 1,231 selected frames.

Using a large regularization, for the neural network enforces a smooth structure to the χ function, which may also be understood as a smoothing of the data and thereby artificially connecting spatially adjacent samples. With regularization, ISOKANN thus isolates the spatially large transitions, which amount to macroscopic changes.

4.2 Application to MOR

ISOKANN serves as an efficient tool to analyze rare events in the simulation of the μ-opioid receptor. As mentioned earlier, finding a solution function χ of (4.20) with a low exit rate c_1 corresponds to the identification of a "reaction coordinate." In Figure 4.2, the resulting χ-values along 10 independent molecular simulations are shown (blue dots). One can see that the lowest and highest values of χ are not to be found within one trajectory. Using ISOKANN, it is possible to extract the "dynamically most distant" frames from the simulation; see Figure 4.3. Indeed the χ-values correspond to a reaction coordinate for the transition from an inactive to an active macro state of MOR. Although none of the 10 trajectories simulates this process completely, we can extract the time-determining steps along the reaction path by using the shortest path routine described in Section 3. The picked path is rather small in length (1,231 frames, orange in Figure 4.2) as compared to entire trajectory of 10,000 stored frames. It displays a physically plausible and compellingly smooth transition between the extremal states.

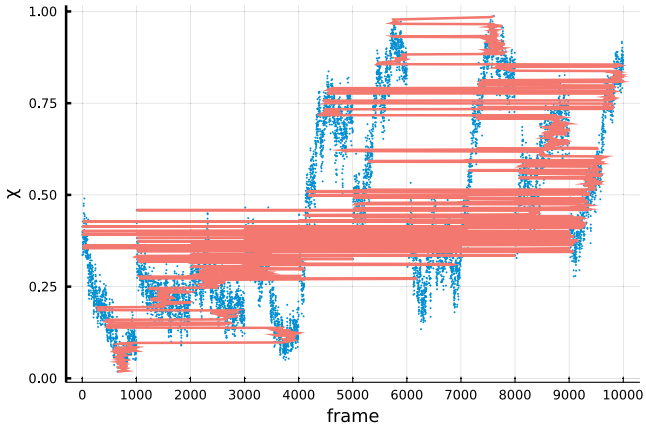


Figure 4.2: The membership value $\chi(x)$ for each state (frame) x obtained from 10 independent simulations, with each simulation comprising 1,000 frames. The y-axis represents the macroscopic transition, showing that different trajectories cover distinct segments of this transition while exhibiting partial overlap with other trajectories. The extracted reaction path (orange line) progresses monotonously from $\chi \approx 0$ to $\chi \approx 1$ while maintaining (not depicted) spatially smooth transitions. Note that it incorporates the data from among all 10 simulations and even jumps between them where facilitated by small spatial distances in Ω.

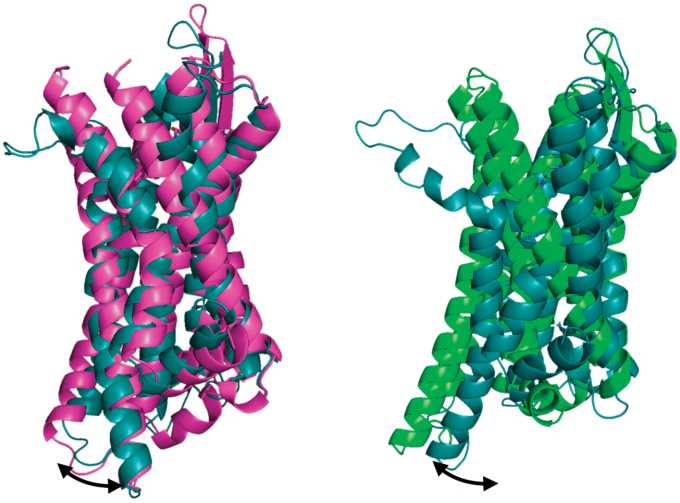


Figure 4.3: Visualization of the MOR states (teal) at the beginning (left) and end (right) of the reaction path from Figure 4.2 (orange). The superimposed purple and green structures are known to be representative of the inactive (PDB 7UL4), respectively, active (PDB 8EF5) states. We can see that ISOKANN isolates close similar to these known metastabilities from the simulation trajectories without their a priori knowledge. The reactive path clearly displays the tilting motion (black arrow) undergone by the transmembrane helix 6.

Whereas it was not at all clear whether the supplied simulation data was sufficient a priori, the resulting path identifies the known crystal structures and a path between them. Figure 4.3 displays the modified MOR (teal) with a disulfide bond at the starting point of the reaction path aligning with the inactive structure of the unmodified, natural state of the MOR (purple). Laboratory experiments suggest partial activation of the MOR, which is confirmed by the shortest path analysis in the tilting movement of the transmembrane helix 6 (bottom left) outwards like it is seen in the fully activated, natural MOR crystal structure (green). It is noteworthy that the start and end points of the reactive path were determined without prior knowledge of the crystal structures. Extracting these extremal states and the reaction path from the raw trajectory data frame-by-frame would pose a considerable challenge.

Using a linear regression to estimate γ_1, γ_2 in (4.20), we can compute the exit rate for a given lag time $\tau = 0.1\,\mathrm{ns}$ by (4.22) resulting in $c_1 \approx 0.06\,\mathrm{ns}^{-1}$.

5 Conclusion

In bridging the conceptual gap between micro state and macro state analyses, we introduced the notion of mean holding probabilities $p_\chi(\boldsymbol{x}, t) = \chi(\boldsymbol{x})\mathrm{e}^{-c_1 t}$ represented in terms of membership functions χ, which we interpret as the macro state itself. Taking this as a

theoretical starting point, the computation of mean holding times $t_{mh} \propto \chi$, which generalizes their classical set-based definitions, and of reaction paths along $\boldsymbol{r} \propto -\nabla\chi$ has been shown. Being proportional to a mean holding time, $\chi(\boldsymbol{x})$ represents a kind of temporal order of micro states $\boldsymbol{x} \in \Omega$ in a high-dimensional space Ω thereby serving as a reaction coordinate.

We briefly described ISOKANN, a machine-learning-based algorithm, which we use to approximately solve the high-dimensional partial differential equation (4.4) defining χ. In a further step we interpret the values of the obtained solution χ to define the weights of edges of a graph, in which the vertices represent biomolecular micro states of an opioid-receptor simulation. Solving a shortest path problem for this graph allows us to obtain a subsample of the simulation data, which captures the time-determining steps of macro molecular transitions.

Our method is able to pick micro states from different independent MD trajectories generated for the same biomolecular system in order to combine them into one "temporally and spatially" ordered path between macro states; see Figure 4.2. This path shows the time-determining steps of a rare transition event between the macro states.

In our example, this approach effectively condenses 10,000 frames into a concise set of 1,231 frames. Using ISOKANN together with shortest path computation extracts what "really is to be seen" in the trajectories. In this case, the extracted path accurately depicts the transition of the modified MOR from an inactive to an active state as also indicated in experimental conditions. It further enhances our understanding of the role of a disulfide bond resulting from oxidative stress. Supported by this findings detailed experiments highlighting the role of implicated cysteins pair (159 and 251) out of other 2 pairs are planned. This example highlights ISOKANN's potential to significantly streamline the analysis of long-term MD simulations and extract meaningful reaction paths.

Bibliography

[1] E. Darcq and B. L. Kieffer. Opioid receptors: drivers to addiction? *Nature Reviews. Neuroscience*, 19(8):499–514, 2018.

[2] P. Deuflhard and M. Weber. Robust Perron cluster analysis in conformation dynamics. *Linear Algebra and Its Applications*, 398c:161–184, 2005.

[3] N. Ernst, K. Fackeldey, A. Volkamer, O. Opatz, and M. Weber. Computation of temperature-dependent dissociation rates of metastable protein–ligand complexes. *Molecular Simulation*, 45(11):904–911, 2019.

[4] L. R. Ford. *Network Flow Theory*. Rand Corporation, Santa Monica, CA, 1956.

[5] H. Gzyl. The Feynman–Kac formula and the Hamilton–Jacobi equation. *Journal of Mathematical Analysis and Applications*, 142(1):74–82, 1989.

[6] B. Leimkuhler and C. Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*, 1st edition. Interdisciplinary Applied Mathematics. Springer International Publishing and Imprint, volume 39. Springer, Cham, 2015.

[7] C. Mura and C. E. McAnany. An introduction to biomolecular simulations and docking. *Molecular Simulation*, 40(10–11):732–764, Aug. 2014.

[8] J. R. Norris. *Markov Chains, volume 2*. Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 1998.

[9] G. A. Pavliotis. *Stochastic Processes and Applications—Diffusion Processes, the Fokker–Planck and Langevin Equations*. Springer, 2014.

[10] C. B. Pert and S. H. Snyder. Opiate receptor: demonstration in nervous tissue. *Science*, 179(4077):1011–1014, 1973.

[11] J. Prinz, H. Wu, M. Sarich, B. Keller, M. Senne, M. Held, J. Chodera, C. Schütte, and F. Noé. Markov models of molecular kinetics: generation and validation. *Journal of Chemical Physics*, 134(17):174105, 2011.

[12] R. J. Rabben, S. Ray, and M. Weber. ISOKANN: invariant subspaces of Koopman operators learned by a neural network. *Journal of Chemical Physics*, 153(11):114109, 2020.

[13] P. W. Reeh and K. H. Steen. Tissue acidosis in nociception and pain. *Progress in Brain Research*, 113:143–151, 1996.

[14] M. J. Robertson, M. M. Papasergi-Scott, F. He, A. B. Seven, J. G. Meyerowitz, O. Panova, M. C. Peroto, T. Che, and G. Skiniotis. Structure determination of inactive-state gpcrs with a universal nanobody. *Nature Structural & Molecular Biology*, 29(12):1188–1195, 2022.

[15] C. Schütte, S. Klus, and C. Hartmann. Overcoming the timescale barrier in molecular dynamics: transfer operators, variational principles and machine learning. *Acta Numerica*, 32:517–673, 2023.

[16] A. Sikorski. Julia Package: ISOKANN.jl. https://github.com/axsk/ISOKANN.jl, 2023.

[17] A. Sikorski, E. Ribera Borrell, and M. Weber. Learning Koopman eigenfunctions of stochastic diffusions with optimal importance sampling and ISOKANN. *Journal of Mathematical Physics*, 65(1):013502, 01 2024.

[18] V. Spahn, G. Del Vecchio, D. Labuz, A. Rodriguez-Gaztelumendi, N. Massaly, J. Temp, V. Durmaz, P. Sabri, M. Reidelbach, H. Machelska, M. Weber, and C. Stein. A nontoxic pain killer designed by modeling of pathological receptor conformations. *Science*, 355(6328):966–969, 2017.

[19] M. Weber. *A subspace approach to molecular Markov state models via a new infinitesimal generator*. Habilitation thesis, FU, Berlin, 2011.

[20] M. Weber and N. Ernst. A fuzzy-set theoretical framework for computing exit rates of rare events in potential-driven diffusion processes, 2017.

[21] M. Weber and S. Kube. Preserving the Markov property of reduced reversible Markov chains. In: T. E. Simos and C. Tsitouras, editors, *Numerical Analysis and Applied Mathematics: International Conference on Numerical Analysis and Applied Mathematics 2008*. American Institute of Physics Conference Series, volume 1048, pages 593–596. AIP, Sept. 2008.

[22] Y. Zhuang, Y. Wang, B. He, X. He, X. E. Zhou, S. Guo, Q. Rao, J. Yang, J. Liu, Q. Zhou, et al. Molecular recognition of morphine and fentanyl by the human μ-opioid receptor. *Cell*, 185(23):4361–4375, 2022.

Phillip Semler and Martin Weiser

Adaptive gradient-enhanced Gaussian process surrogates for inverse problems

Abstract: Generating simulated training data needed for constructing sufficiently accurate surrogate models to be used for efficient optimization or parameter identification can incur a huge computational effort in the offline phase. We consider a fully adaptive greedy approach to the computational design of experiments problem using gradient-enhanced Gaussian process regression as surrogates. Designs are incrementally defined by solving an optimization problem for accuracy given a certain computational budget. We address not only the choice of evaluation points but also of required simulation accuracy, both of values and gradients of the forward model. Numerical results show a significant reduction of the computational effort compared to just position-adaptive and static designs as well as a clear benefit of including gradient information into the surrogate training.

Keywords: Surrogate models, inverse problems, active learning, Gaussian process regression, design of computer experiments

MSC 2020: 62F35, 65N21, 90C59

1 Introduction

The response of many parameter-dependent physical models can be described by a functional relation y mapping parameters p to observable quantities $y(p)$. The inverse problem of inferring model parameters p from measurements y^m is frequently encountered in various fields including physics, engineering, finance, and biology. Point estimates p_*, for example, the maximum likelihood estimate $\arg\min_p \|y(p) - y^m\|$, are often computed by optimization methods or by sampling the posterior probability distribution of the parameters given the measurement data [7, 16].

Often, y is only available as a complex numerical procedure, such as the numerical solution of a partial differential equation. Solving an optimization problem for parameter estimation is then computationally expensive. The effort can be too high for online and real-time applications. Fast surrogate models approximating y are utilized as a replacement for the forward model when solving the inverse problems, in particular when both parameters p and measurements y^m are low-dimensional. Various types of surrogates are employed, including polynomials, sparse grids, tensor trains, artifi-

Phillip Semler, Martin Weiser, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, e-mails: semler@zib.de, weiser@zib.de, https://orcid.org/0000-0002-1071-0044

https://doi.org/10.1515/9783111376776-005

cial neural networks, and Gaussian process regression (GPR) [27, 38], on which we focus here.

Surrogate models rely on values $y(p_i)$ at specific evaluation points p_i as training data. The quality of the resulting surrogate heavily depends on the number and position of these sample points. Constructing an accurate surrogate model can become computationally expensive when a large number of evaluations is required. Consequently, strategies for selecting near-optimal evaluation points have been proposed, in particular for analytically well-understood GPR [25]. A priori point sets [10, 23] are effectively supplemented by adaptive designs [3, 15, 18, 31]. Active learning relies on pointwise estimates of the surrogate approximation error for guiding the selection process, usually including the parameter point that maximizes some acquisition function into the training set [14, 20, 36].

When numerical procedures like finite element (FE) solvers are employed to compute training data, the resulting evaluations of $y(p_i)$ are affected by discretization and truncation errors. While uniformly high accuracy can be used, this incurs a high computational effort. The trade-off between accuracy and cost has received limited systematic investigation. Besides two-level approaches using a low-fidelity and a high-fidelity model [22], an adaptive choice of evaluation tolerances has been proposed [26]. Significant efficiency gains by joint optimization of evaluation position and tolerance given a limited computational budget have been achieved [29].

In the current paper, we extend [29] by including gradient information. Gradients of FE simulations can often be obtained efficiently by solving tangent or adjoint equations [11]. We make use of gradient-enhanced GPR (GEGPR) promising higher accuracy than standard GPR [8, 35, 36]. Due to the high information content of gradient data, the required number of evaluation points is small compared to gradient-free GPR. We devise a greedy-type strategy that simultaneously optimizes the next positions and tolerances for a forward model and gradient evaluation. For that, we extend the accuracy and work models used in the optimizer to take gradient computation into account.

For measuring the accuracy of the surrogate model, we adopt a goal-oriented approach [32], assessing the approximation error by its impact on the accuracy of the identified parameter. We aim for a uniform absolute tolerance or, if that is not reachable, at least a uniform bounded deterioration relative to the exact model results. We focus on adaptive FE simulations, where standard a priori error estimates and coarse estimates of the computational work are available.

The remainder of the paper is structured as follows: In Section 2, we formalize the parameter identification and surrogate model construction problems, introducing notation and GEGPR. Their adaptive construction is given in Section 3 after introducing gradient-enhanced work and accuracy models. Numerical experiments are provided in Section 4.

2 Surrogate-based parameter identification

First, we briefly define the parameter identification as maximum posterior evaluation in a Bayesian context, before introducing GEGPR surrogate models. For a more detailed discussion, we refer to [29].

2.1 Inverse problem

Let the forward problem $y : \mathcal{X} \subset \mathbb{R}^d \to \mathbb{R}^m$ of mapping a parameter vector $p \in \mathcal{X}$ to a model output $y(p) \in \mathbb{R}^m$ be Lipschitz-continuously differentiable and the parameter space $\mathcal{X}$ be bounded and closed. We assume that numerical approximations $y_\tau(p)$ of $y(p)$ and $y'_{\tau'}(p)$ of the derivative $y'(p)$ can be computed, satisfying the error bounds $\|y_\tau(p) - y(p)\| \leq \tau$ and $\|y'_{\tau'}(p) - y'(p)\| \leq \tau'$, respectively, for any given tolerances $\tau, \tau' > 0$ in some norms $\|\cdot\|$. We consider the simplest Bayesian setting of normally distributed measurement errors with likelihood covariance $\Sigma_l \in \mathbb{R}^{m\times m}$ and a Gaussian prior with covariance $\Sigma_p \in \mathbb{R}^{d\times d}$ and mean p^0. Given measurements y^m, we are interested in the maximum posterior point estimate $p(y^m)$, a minimizer of the negative log-posterior

$$J(p; y^m) := \frac{1}{2}\left\|y(p) - y^m\right\|^2_{\Sigma_l^{-1}} + \frac{1}{2}\left\|p - p^0\right\|^2_{\Sigma_p^{-1}} \tag{5.1}$$

over $p \in \mathcal{X}$, where the norm induced by a positive symmetric definite matrix A is denoted by $\|v\|_A^2 := v^T A v$. By regularity of y and compactness of $\mathcal{X}$, J is Lipschitz-continuously differentiable, and a not necessarily unique, minimizer exists. For the numerical minimization of (5.1), we consider a Gauss–Newton (GN) method [4] and assume that the forward model is compatible and the measurement errors sufficiently small such that the residual $y(p(y^m)) - y^m$ is small, such that the GN method converges locally.

2.2 Gradient-enhanced Gaussian process regression

We aim at building a surrogate model y^* for $y : \mathcal{X} \to \mathbb{R}^m$ based on a set of evaluations of the model and its derivative with certain tolerances τ, τ'. We directly consider GEGPR; see [36] and the references therein, and refer to [6, 17, 24, 29] for purely value-based GPR surrogate modeling.

2.2.1 Designs and training data

We define $D := \{\mathcal{D} : \mathcal{X} \to (\mathbb{R}_+ \cup \{\infty\})^2 \mid \operatorname{card} X(\mathcal{D}) \in \mathbb{N}\}$ as the set of admissible designs, where $X(\mathcal{D}) := \{p \in \mathcal{X} \mid \min_{k\in\{1,2\}}\{\mathcal{D}(p)_k\} < \infty\}$ is the evaluation set. For $p_i \in X(\mathcal{D})$, the design $\mathcal{D}(p_i) = [\tau_i, \tau'_i]$ defines the evaluation tolerances for values and derivatives of the

forward model y forming the training data $y_i = y_{\tau_i}(p_i)$ and $y'_i = y'_{\tau'_i}(p_i)$. Note that $\mathcal{D}' \leq \mathcal{D}$ for two designs $\mathcal{D}'$ and $\mathcal{D}$ implies that $\mathcal{D}'$ is a refinement of $\mathcal{D}$, since $X(\mathcal{D}') \supset X(\mathcal{D})$ and $\tau'_i = \mathcal{D}'(p_i) \leq \mathcal{D}(p_i) = \tau_i$ for $\tau_i \in X(\mathcal{D})$ hold. Here, we use the componentwise partial ordering on $\mathbb{R}^2$.

We assume for simplicity that the componentwise evaluation errors $e_i = y_i - y(p_i)$ and $e'_i = y'_i - y'(p_i)$ are independent and normally distributed, that is, $e_i \sim \mathcal{N}(0, \tau_i^2 I_m)$ and $e'_i \sim \mathcal{N}(0, (\tau'_i)^2 I_{md})$, where we denote the identity matrix of dimension m by I_m and identify $e'_i \in \mathbb{R}^{m\times d}$ with its column-major vector representation in $\mathbb{R}^{md}$.

Remark. The assumption of normally i. i. d. evaluation errors is of course unrealistic. Discretization errors may be biased, as in P1 finite elements for the simple 1D Poisson problem with constant source term. They are for sure correlated locally in parameter space, as discrete decisions on mesh refinement, truncating an iterative solver, or rejecting time steps separate the parameter space into disjoint cells within which the errors are highly correlated, but between them there is less or no correlation. Ideally, one would have a stochastic model of the inexact evaluation errors, as considered in probabilistic numerics [2, 13]. Such models are, however, complex and extremely difficult to obtain a priori. Thus, we use the simplistic assumption of normally and independently distributed evaluation errors. Despite the obvious shortcomings of this assumption, it works surprisingly well in numerical examples.

The complete model evaluation at a parameter point p_i thus comprises a vector $y_i \in \mathbb{R}^m$ and a matrix $y'_i \in \mathbb{R}^{m\times d}$. Being interested in a joint representation of values y and derivatives y', we introduce the extended forward model $z : \mathcal{X} \to \mathbb{R}^{m+md}$, $p \mapsto (y(p), y'(p))$ as the vector representation of the complete evaluation. Analogously, we define the approximate evaluations $\hat{z}_i = (y_i, y'_i) \in \mathbb{R}^{m+md}$. The training data is then distributed as

$$\hat{z}_i \sim \mathcal{N}(z(p_i), E_i), \quad E_i = \begin{bmatrix} \tau_i I_m & \\ & \tau'_i I_{md} \end{bmatrix}. \tag{5.2}$$

Remark. In principle, the tolerances for computing values y and derivatives y' can be chosen independently. In practical computation, however, when evaluating derivatives, obtaining the value at the same position p_i with the same accuracy $\tau_i = \tau'_i$ usually comes for free. We therefore restrict the attention to designs satisfying $\tau_i \leq \tau'_i$ for all i.

2.2.2 Gaussian process prior

Assume that forward models are a Gaussian process, that is, model evaluations $z(p_i)$ are jointly normally distributed for any finite evaluation set $X = \{p_i \mid i = 1, \dots, n\}$, which forms the GPR prior distribution

$$\pi_{\text{prior}}(Z) = \mathcal{N}(\zeta, \tilde{K}) \tag{5.3}$$

for joint model responses $Z = (z(p_i))_{i=1,\dots,n} \in \mathbb{R}^{n(m+md)}$ with given mean ζ. The structure of Z induces a nested block form of the covariance matrix $\tilde{K} = (\hat{K}_{ij})_{ij}$, $i,j = 1,\dots,n$ (see [30, 36]) with

$$\hat{K}_{ij} = \begin{bmatrix} K_{ij} & K'_{ij} \\ K'_{ji} & K''_{ij} \end{bmatrix}. \tag{5.4}$$

Here, $K_{ij} = k(p_i,p_j)K \in \mathbb{R}^{m\times m}$ is the covariance of the model values $y(p_i)$ and $y(p_j)$, assumed to be separable, that is, defined as a product of a positive definite kernel k depending on the evaluation positions p_i and p_j only, and a matrix K describing the covariance of model response components independently of the evaluation position. Most often, k is assumed to depend on the distance of p_i and p_j only, such that $k(p_i,p_j) = k(\|p_i - p_j\|)$. A common choice is the Gaussian kernel $k(x) = \exp(-\sigma x^2)$.

The remaining blocks $K'_{ij} \in \mathbb{R}^{m\times dm}$ and $K''_{ij} \in \mathbb{R}^{dm\times dm}$ describe the covariance of model values and model derivatives y' and of the covariance of model derivatives, respectively. Since differentiation is a linear operation, the derivatives are again normally distributed, and the covariances are given in terms of the kernel's derivatives as $\operatorname{cov}(y(p),y'(q)) = \nabla_q k(p,q)$; see [25, Chapter 9.4]. We therefore can write the matrix blocks in terms of the Kronecker product $\otimes$ as

$$K'_{ij} = \partial_{p_j} k(p_i,p_j) \otimes K, \quad K'_{ji} = \partial_{p_i} k(p_i,p_j) \otimes K, \quad K''_{ij} = \partial_{p_i}\partial_{p_j} k(p_i,p_j) \otimes K. \tag{5.5}$$

2.2.3 Regression

Given approximate complete evaluations $\hat{Z} = (\hat{z}_i)_{i=1,\dots,n} \in \mathbb{R}^{n(m+md)}$ according to (5.2), the GPR likelihood is $\pi_{\text{like}}(\hat{Z} \mid Z) = \mathcal{N}(Z,\tilde{E})$, with $Z = z(X)$ the exact forward model evaluations and $\tilde{E} = \operatorname{diag}(E_1,\dots,E_n)$. With the GPR prior (5.3), the GPR posterior probability for the complete model response Z is $\pi_{\text{post}}(Z \mid \hat{Z}) \propto \pi_{\text{like}}(\hat{Z} \mid Z)\pi_{\text{prior}}(Z)$ by Bayes' rule. As a product of two Gaussian distributions, π_{post} is again Gaussian, with covariance $\Gamma = (\tilde{K}^{-1} + \tilde{E}^{-1})^{-1}$ and mean $\bar{Z} = \Gamma(\tilde{K}^{-1}\zeta + \tilde{E}^{-1}\hat{Z})$. In particular, for p_n, the marginal covariance $\Gamma_{\mathcal{D}}(p_n;\tau_n,\tau'_n) \in \mathbb{R}^{m(1+d)\times m(1+d)}$ according to the block structure of Z and Γ is a monotone function of τ_n and τ'_n, that is, $\Gamma_{\mathcal{D}}(p_n,\tau_n+\delta,\tau'_n+\delta') \geq \Gamma_{\mathcal{D}}(p_n,\tau_n,\tau'_n)$ for $\delta,\delta' \geq 0$.

Regression is performed for a point p_n with unknown model response, that is, formally $\tau_n = \tau'_n = \infty$, by extracting the marginal mean $z_{\mathcal{D}}(p_n) = \bar{Z}_n \in \mathbb{R}^{m+md}$ and the marginal covariance $\Gamma_{\mathcal{D}}(p_n)$. Note that $z_{\mathcal{D}}(p_n) = (y_{\mathcal{D}},y'_{\mathcal{D}})(p_n)$ provides an estimate for both, values and derivatives of the model at p_n, and $\Gamma_{\mathcal{D}}(p_n)$ quantifies the uncertainty of this estimate. Due to the choice (5.5) for the prior covariance of values and gradients, the GPR model is consistent, that is, the derivative estimate $y'_{\mathcal{D}}$ coincides with the parametric derivative of the value estimate $y_{\mathcal{D}}$.

Remark. If the components of the model response are assumed to be uncorrelated, the covariance matrix K is diagonal. With the independence of the evaluation errors postu-

lated in (5.2), both $\tilde{K}$ and $\tilde{E}$ and, therefore, Γ decompose into m independent blocks, one for each component. Then the GPR can be performed independently for each component using a simpler scalar GPR model. An appropriate covariance model for the components can, however, result in superior results [1].

3 Adaptive training data generation

Replacing exact model evaluations $y(p)$ in the objective (5.1) by a cheaper GPR model $y_{\mathcal{D}}(p)$ based on a design $\mathcal{D}$ yields maximum posterior point estimates

$$p_{\mathcal{D}}(y^m) = \operatorname*{arg\,min}_{p\in\mathcal{X}} J_{\mathcal{D}}(p;y^m) := \frac{1}{2}\left\|y_{\mathcal{D}}(p) - y^m\right\|^2_{\Sigma_l^{-1}} + \frac{1}{2}\left\|p - p^0\right\|^2_{\Sigma_p^{-1}}$$

and saves computational effort when computing Gauss–Newton steps $\Delta p_{\mathcal{D}}(p, y^m)$ by solving

$$(y'_{\mathcal{D}}(p)^T \Sigma_l^{-1} y'_{\mathcal{D}}(p) + \Sigma_p^{-1})\Delta p_{\mathcal{D}} = (y_{\mathcal{D}}(p)')^T \Sigma_l^{-1}(y_{\mathcal{D}}(p) - y^m) + \Sigma_p^{-1}(p^0 - p).$$

It also incurs both some error $p_{\mathcal{D}}(y^m) - p(y^m)$ of the resulting identified parameters and a considerable computational effort for evaluating the model according to the design $\mathcal{D}$ beforehand.

If computational resources are limited, the challenge is to determine which parameters p_i simulations should be run with, and which tolerances τ_i, τ_i' should be used to achieve the best accuracy. This design of experiments problem for $\mathcal{D}$ involves balancing the competing objectives of minimizing the expected approximation error $E(\mathcal{D})$ of the surrogate model and minimizing the computational effort $W(\mathcal{D})$ required to create the training data. Since little is known about the model derivative y', and consequently, about $E(\mathcal{D})$ before any simulations have been performed, we follow a sequential design of experiments approach [29] by incrementally spending computational budget of size ΔW. In each step, we thus have to solve an incremental design problem for $\tilde{\mathcal{D}} \in D$ refining a given preliminary design $\mathcal{D}$:

$$\min_{\tilde{\mathcal{D}}\leq\mathcal{D}} E(\tilde{\mathcal{D}}) \quad \text{subject to } W(\tilde{\mathcal{D}}|\mathcal{D}) \leq \Delta W. \tag{5.6}$$

We will establish quantitative error estimates $E(\mathcal{D})$ and work models $W(\mathcal{D})$, and then develop a heuristic for approximately solving problem (5.6).

3.1 Accuracy model

First, we need to quantify the parameter reconstruction error $p_{\mathcal{D}}(y^m) - p(y^m)$ in terms of the measurement error variance Σ_l and the surrogate model approximation quality

$y_{\mathcal{D}} - y$ depending on the design $\mathcal{D}$. We start by establishing an estimate of the parameter reconstruction error for deterministic functions $y(p)$ and $y_{\mathcal{D}}(p)$.

Theorem 5.1. *Assume that y is twice continuously differentiable with uniformly bounded first and second derivatives in a neighborhood B of a minimizer $p^* \in \mathcal{X}$ of $J(p; y^m)$ for some measurement data $y^m \approx y(p^*)$, such that the residual $\|y(p^*) - y^m\|_{\Sigma_l^{-1}}$ is sufficiently small and $y'(p^*)\Sigma_l^{-1}y'(p^*) + \Sigma_p^{-1}$ is positive definite.*

Then there are $\bar{\epsilon}, \bar{\epsilon}' > 0$ as well as constants a, a' depending on p^*, such that for all $\epsilon \le \bar{\epsilon}$ and $\epsilon' \le \bar{\epsilon}'$ and surrogate models $y_{\mathcal{D}} : \mathbb{R}^d \to \mathbb{R}^m$ with $\|y_{\mathcal{D}} - y\|_{L^\infty(B)} \le \epsilon$ and $\|y'_{\mathcal{D}} - y'\|_{L^\infty(B)} \le \epsilon'$ there is a locally unique minimizer $p_{\mathcal{D}}(y^m)$ of $J_{\mathcal{D}}$ satisfying the error bound

$$\left\| p_{\mathcal{D}}(y^m) - p^* \right\| \le a\epsilon + a'\epsilon'. \tag{5.7}$$

A more detailed claim and the proof is given in [29, Corollary 3.1.1].

Though the constants $a(p^*), a(p^*)'$ are quantitatively unavailable for concrete problems, Theorem 5.1 establishes a linear relation between the surrogate model accuracy in terms of ϵ and ϵ', and the incurred error $p_{\mathcal{D}}(y^m) - p^*$ in the parameter estimate. The factors a and a' can be estimated numerically by bounding linearized error transport through a Gauss–Newton iteration.

We aim at a minimal absolute reconstruction error, but due to measurement errors, we cannot expect the error to be much less than the true posterior standard deviation level

$$e_0(p) = \left\| y'(p)^T \Sigma_l^{-1} y'(p) + \Sigma_p^{-1} \right\|^{1/2},$$

even if the exact forward model is used. For a closely related definition of the unavoidable error level in terms of Theorem 5.1, we refer to [29]. With an expected error of magnitude e_0, we consider a small relative error with respect to e_0 incurred by the surrogate model error to be acceptable. We therefore define the local error quantity:

$$e_{\mathcal{D}}(p) := \frac{a(p)\epsilon + a(p)'\epsilon'}{1 + \alpha e_0(p)}. \tag{5.8}$$

Since during the construction of the surrogate model $y_{\mathcal{D}}$ by minimizing (5.6) the measurement values y^m, and hence the parameter position $p = p(y^m)$ of interest are unknown, the error quantity (5.8) needs to be considered over the whole parameter region $\mathcal{X}$. We therefore define the accuracy model

$$E(\mathcal{D}) := \|e_{\mathcal{D}}\|_{L^q(\mathcal{X})} \quad \text{for some } 1 \le q < \infty, \tag{5.9}$$

that is to be minimized by selecting an appropriate design $\mathcal{D}$. Choosing $q \approx 1$ would focus on minimizing the average parameter reconstruction error, while choosing q very large would focus on the worst case and impose a roughly uniform accuracy. For the numerical experiments in Section 4, we have chosen $q = 2$.

Still missing are the surrogate model error bounds ϵ, ϵ' in terms of the design $\mathcal{D}$. Unfortunately, for virtually all cases of practical interest, there is little hope for obtaining simultaneously rigorous and quantitatively useful bounds. For GPR surrogate models, the global support of the posterior probability density precludes the existence of a strict bound, though its fast decay provides thresholds that are with high probability not exceeded. The assumed normal distribution of errors $z(p) - z_{\mathcal{D}}(p) \sim \mathcal{N}(0, \Gamma_{\mathcal{D}}(p))$ implies that both the Euclidean norm $\|y(p) - y_{\mathcal{D}}(p)\|_2$ and the Frobenius norm $\|y'(p) - y'_{\mathcal{D}}(p)\|_F$ are generalized-χ^2-distributed, and hence formally unbounded. Instead of a strict bound, we use a representative statistical quantity for ϵ and ϵ' based on the marginal covariance $\Gamma_{y_{\mathcal{D}}} = \Gamma_{\mathcal{D}}(p)_{1:m,1:m}$, such as the mean [19]

$$\epsilon := \operatorname{tr}(\Gamma_{y_{\mathcal{D}}}). \tag{5.10}$$

Analogously, ϵ' can be defined in terms of $\Gamma_{y'_{\mathcal{D}}} = \Gamma_{\mathcal{D}}(p)_{m+1:m(d+1),m+1:m(d+1)}$. Inserting the chosen values of ϵ and ϵ' into (5.8) completes the accuracy model.

One advantage of integrating derivative information into the GPR surrogate is that there is an explicit variance estimate available for defining ϵ'. In contrast, GPR surrogates defined only in terms of the values y as considered in [29] must rely on an empirical relation of ϵ and ϵ'.

3.2 Work model

The evaluation of the forward model $y(p)$ usually involves some kind of numerical approximation, resulting in an approximation $y_\tau(p)$. While in principle any accuracy $\|y_\tau(p) - y(p)\| \le \tau$ for arbitrary tolerance $\tau > 0$ can be achieved, this requires a computational effort $W(\tau)$ to be spent on the evaluation. For adaptive finite element computations in $\mathbb{R}^l$ with ansatz order r and N degrees of freedom, we expect a discretization error $\epsilon = \mathcal{O}(N^{-r/l})$ [5]. Assuming an optimal solver with computational work $\mathcal{O}(N)$, we obtain a work model $W(\tau) = \tau^{-2s}$ with $s = l/(2r) > 0$; see [34]. W is monotone and satisfies the barrier property $W(\tau) \to \infty$ for $\tau \to 0$ and the minimum effort property $W(\tau) \to 0$ for $\tau \to \infty$.

Including gradient information is of particular interest if derivatives can be computed efficiently, for example, with adjoint methods, such that the cost of derivative computation is a small multiple c of the value computation cost, and independent of the number d of parameters [11]. This leads to the work model $W(\tau') = c(\tau')^{-2s}$.

The computational effort incurred by a design $\mathcal{D}$ is then

$$W(\mathcal{D}) := \sum_{p_i \in X(\mathcal{D})} \tau_i^{-2s} + c\left(\tau'_i\right)^{-2s}. \tag{5.11}$$

Being interested in *incremental designs*, we assume $\mathcal{D}$ to be a design already realized. Evaluating the model on a finer design $\tilde{\mathcal{D}} \le \mathcal{D}$ can consist of simulating the model for

parameters $p \notin X(\mathcal{D})$, or improving the accuracy of already performed simulations for $p \in X(\mathcal{D})$ with $\tilde{\mathcal{D}}(p)_k < \mathcal{D}(p)_k$ for some $k \in \{1, 2\}$, or both. If already conducted simulations can be continued instead of started again, the computational effort of obtaining the training data set $\tilde{\mathcal{D}}$ from $\mathcal{D}$ is

$$W(\tilde{\mathcal{D}} \mid \mathcal{D}) = W(\tilde{\mathcal{D}}) - W(\mathcal{D}). \tag{5.12}$$

3.3 The design of computer experiments problem

The design problem (5.6) is combinatorial in nature due to the unknown number n of evaluation points and the decision between introducing new points or reusing existing ones. It is therefore highly nonlinear and difficult to treat rigorously due to the parameter locations p_i to be optimized. In particular, relaxing the design to the space of nonnegative regular Borel measures, as used in [21], is not feasible due to the nonlinearity of the work model (5.12).

We therefore take a heuristic two-stage approach. We simplify the problem by decoupling the choice of evaluation positions p_i from the choice of evaluation accuracies τ_i, τ_i'. In the first stage, we select a few promising additional points for inclusion into the evaluation set X. In the second stage, the evaluation tolerances τ_i and τ_i' are then optimized for minimal error $E(\tilde{\mathcal{D}})$ given the computational budget constraint $W(\tilde{\mathcal{D}}|\mathcal{D}) \leq \Delta W$. Then the necessary evaluations of the forward model are performed and the GP surrogate model is updated, yielding improved error estimates for the next incremental design.

A parameter point p is particularly promising for inclusion if its predicted impact on the overall error E is large. The impact can be estimated by the local derivative of $e_{\mathcal{D}}(p)^q$ with respect to work spent for approximating $y(p)$. We assume $\tau' = \beta\tau$ for some $\beta \geq 1$ and consider

$$\begin{aligned}\frac{\partial e_{\mathcal{D}}(p)}{\partial W(p)}(\beta) &= \frac{\partial e_{\mathcal{D}}(p)}{\partial(\epsilon, \epsilon')} \frac{(\partial\epsilon, \epsilon')}{\partial\tau} \left(\frac{dW}{d\tau}\right)^{-1} \\ &= \frac{[a(p), a(p)']}{1 + \alpha e_0(p)} \left[\operatorname{tr} \frac{\partial \Gamma_{y_{\mathcal{D}}}}{\partial(\tau, \tau')}, \operatorname{tr} \frac{\partial \Gamma_{y_{\mathcal{D}}'}}{\partial(\tau, \tau')}\right] \begin{bmatrix} 1 \\ \beta \end{bmatrix} \frac{\tau^{2s+1}}{(-2s)(1 + c\beta^{-2s})}.\end{aligned}$$

Neglecting constant factors, which are irrelevant for the *relative* merit of candidate points, and evaluating the derivatives at the current tolerance level $\operatorname{tr}(\Gamma_{y_{\mathcal{D}}})$, we define the acquisition function

$$\phi(p) = \max_{\beta \geq 1} e_{\mathcal{D}}(p)^{q-1} \frac{\partial e_{\mathcal{D}}(p)}{\partial W(p)}(\beta). \tag{5.13}$$

As it is often only practical to compute derivatives along with the values or not at all, we may restrict the choice of β to the two extreme cases $\beta \in \{1, \infty\}$. Selecting candidate

points for inclusion into the evaluation set X can be based on finding local maximizers of the acquisition function ϕ, or finding the best points from a random sampling or from low discrepancy sequences.

With the evaluation set X fixed, the optimization problem (5.6) reduces to the nonlinear programming problem $\min_{\tilde\tau,\tilde\tau'} E(\tilde\tau,\tilde\tau')$ for the new evaluation tolerances $\tilde\tau,\tilde\tau'$, subject to the pure improvement requirement $\tilde\tau \le \tau$, $\tilde\tau' \le \tau'$, and the computational budget constraint $W(\tilde\tau,\tilde\tau') \le W(\tau,\tau') + \Delta W$. For convexity reasons, we consider an equivalent reformulation in terms of $v = \tau^{-2}$ and $v' = (\tau')^{-2}$:

$$\min_{\tilde v,\tilde v'\in\mathbb{R}^n_+} \tilde E(\tilde v,\tilde v')^q \quad \text{s.t.} \quad \tilde v \ge v,\ \tilde v' \ge v',\ \tilde v' \le \tilde v,\ W(\tilde v,\tilde v') \le W(v,v') + \Delta W. \tag{5.14}$$

Theorem 5.2. *The objective $\tilde E(\tilde v,\tilde v')^q$ is convex.*

Proof. As in [29, Theorem 3.2], $\epsilon = \operatorname{tr}(\Gamma_{y_{\mathcal{D}}})$ and $\epsilon' = \operatorname{tr}(\Gamma_{y'_{\mathcal{D}}})$ are convex in v and v' at all $p \in \mathcal{X}$, and consequently, $e_{\mathcal{D}}(p)$ is convex as well. Due to convexity and monotonicity, $e_{\mathcal{D}}(p)^q$, and due to linearity, also its integral E are convex. □

Remark. In contrast to purely value-based GPR surrogates, here the objective E^q is in general not strictly convex since the covariance kernel containing entries of the form $\partial_{p_i} k(p_i,p_j)$ is not pointwise positive.

The convexity of the admissible set $\{\tilde v \in \mathbb{R}^{n+j}_+ \mid W(\tilde v) \le \Delta W + W(v)\}$ depends on the exponent s in the work model (5.11). Clearly, for $s \ge 1$, W is convex, whereas for $s < 1$ it is in general nonconvex, not even quasiconvex. In combination with Theorem 5.2, we obtain the following result.

Corollary 5.3. *For exponents $s \ge 1$, the tolerance design problem* (5.14) *is convex.*

Finite elements of order r in l dimensions with an optimal solver yield $s = l/(2r)$. Consequently, for linear finite elements in two or three space dimensions, any minimizer is a global one, most often unique and nonsparse. In contrast, higher-order finite elements with $r \ge 2$ lead to $s < 1$ and non-convex admissible sets. Their pronounced corners on the coordinate axes make the sparsity of a minimizer likely; see Figure 5.1 right. This agrees with intuition: if increasing the accuracy at a specific sample point is computationally expensive, it is advantageous to distribute the work on a lower accuracy level to several points. If increasing the accuracy is cheap, then it is often better to increase the accuracy of a single point, that to some extent, shares its increased accuracy in a certain neighborhood. Level lines of the gradient-enhanced objective $E(v)$ are also shown (green). A smaller error can be achieved with the same amount of computational work. Note that gradient data does not affect the convexity of the problem.

While in the convex case $s \ge 1$ the optimization is straightforward with any nonlinear programming solver, the nonconvex case is more difficult. Fortunately, guaranteed global optimality is in practice not necessary. The expected sparsity structure suggests a particular heuristic approach: For $i = 1,\dots,n+j$, consider $\tilde v_i = v + a e_i$ with $a > 0$, such that

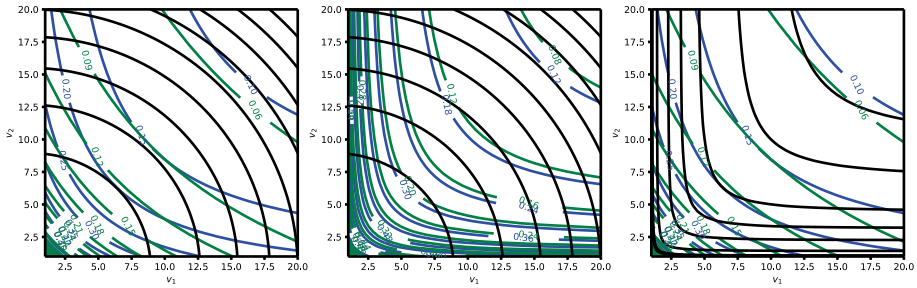


Figure 5.1: Sketch of the design problem (5.14) for $n = 2$ points. Level lines of the objective $E(v)$ without gradient data are drawn by blue lines, whereas those of the budget constraint are indicated by dashed lines. The gradient-based version of $E(v)$ is drawn by solid green lines. *Left:* For $s > 1$, there is a unique nonsparse solution. *Middle:* A smaller correlation length makes sparsity even less likely. *Right:* For $s < 1$, the admissible sets are nonconvex, and we may expect multiple local sparse minimizers.

$W(v) = \Delta W + W(\mathcal{D})$, that is, the accuracy of only a single point is improved, and select the design $\tilde{v}_i$ with smallest objective. If this satisfies the necessary first-order conditions, accept it as solution. Otherwise, perform a local minimization starting from this point.

4 Numerical examples

We present two numerical examples, a low-dimensional one with simple model function y, and a PDE model. We compare the results of the adaptive phase with and without gradient data.

4.1 Analytical example

As a model y, we consider the rotated parabolic cylinder, that is,

$$y_\phi(p) = \big(\cos(\phi)(p_1 + p_2) + \sin(\phi)(p_2 - p_1)\big)^2 \quad \text{for } p \in \mathcal{X} = [0,2]^2,\ \phi \in \mathbb{R}_{>0}.$$

We acquire $m = 3$ measurements for $\phi \in \{0, 2, 4\}$, assume different accuracies of these independent measurements, and a likelihood $\Sigma_L = 10^{-2}\,\mathrm{diag}(1, 0.1, 1)$ with a prior mean of $p^0 = (1.0, 1.5)$ and variance $\Sigma_p = 10^{-2} I$.

We start with an initial design of seven evaluation points (see Figure 5.2) and an evaluation variance of $\sigma^2 = 0.1$. We use the work model with $s = 1/2$. Evaluation of the error quantity E from (5.9) is performed by Monte Carlo integration. To determine candidate points for inclusion in the evaluation set X, we sample the acquisition function (5.13) for $\beta \in \{0, \infty\}$. Only the point with the largest value is included, and gradient information included by setting $\tau' = \tau$ if $\beta = 1$ obtained a higher value. The adaptive phase is terminated if the error E drops below the desired tolerance $\mathrm{TOL} = 10^{-2}$.

Resulting designs

Figure 5.2 illustrates the resulting design with (left) and without (right) gradient data. Black dots are evaluation points without gradient data, while red triangles mark points with gradient information. The size of each marker corresponds to the evaluation accuracy, with larger markers indicating higher accuracy. Red dots represent the initial design, while the color mapping represents the estimated error. The gradient-based algorithm required 24 points, 11 of them with gradient information, each with different evaluation accuracy. The purely value-based design required 30 points. A comparison of the computational costs of the two systems shows that the regular system without gradient data requires costs of $W(\mathcal{D}) = 1.09{\cdot}10^6$, while the gradient-enhanced system requires costs of $W(\mathcal{D}) = 0.11 \cdot 10^5$, and thus costs 10 times less to achieve the desired accuracy.

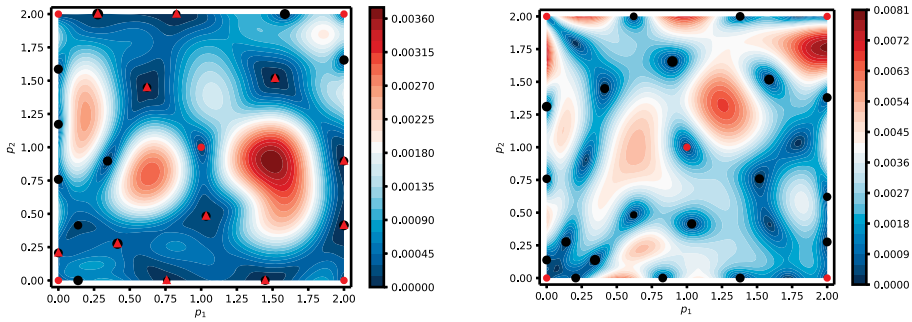


Figure 5.2: Contour plot of the local error density, left with gradient information, right without. Red points show the initial design. Adaptively added data points are indicated by black dots. Small markers indicate low accuracy. Gradient information is indicated by red triangles. The color mapping shows the estimated local reconstruction error evaluated on a dense grid of 10^3 points. The designs were obtained using an incremental budget of $\Delta W = 10^4$ with a desired tolerance of TOL $= 0.01$.

Convergence

Figure 5.3 presents the error against the computational work with different incremental budgets. On the left, we compare designs with gradient data (solid lines) and without (dashed lines). On the right, we compare adaptive designs with gradient data with a position-adaptive design using uniform tolerances for values and gradients. Including derivative information is clearly more efficient, by roughly one order of magnitude, but also leads to pronounced nonmonotone convergence. This is likely an effect of the larger condition number of the covariance matrix when including derivative information, which can lead to suboptimal solutions and high variance in the hyperparameter optimization performed in every step. In contrast, value-based designs exhibit a mostly monotonic behavior. Also apparent is that smaller incremental budgets are more efficient, in particular, for low desired accuracy. On the right, the behavior of uniform tolerance designs is shown. For large tolerances ($\tau = 10^{-1}$), the computational work is

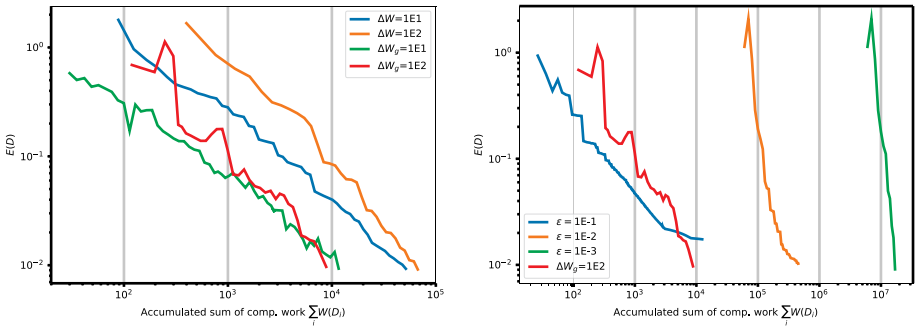


Figure 5.3: Estimated global error $E(\mathcal{D})$ versus accumulated computational work in GEGPR surrogates. *Left*: $E(\mathcal{D})$ for different incremental work ΔW. Solid lines with gradient data. *Right*: Different uniform tolerances in position-adaptive designs compared with different curves for $\Delta W_g = 100$.

small, but the desired accuracy is not reached. Smaller tolerances ($\tau \leq 10^{-2}$) achieve the desired accuracy, but at a much higher cost, roughly two orders of magnitude and more.

Reliability of local error estimates

The error model (5.9) is coarse and may not capture the actual error in identified parameters correctly. It is affected both by linearization error in estimated error transport and the GPR error estimate, which in turn depends also on the hyperparameter optimization. We compare the estimated global error $E(\mathcal{D})$ to the actually obtained errors. For 1600 points p_i, sampled randomly from $\mathcal{X}$, we compute the local error estimate $e_i = e_{\mathcal{D}}(p_i)$ from (5.8), and compare this to the expected actual error, approximated by the sample mean $\tilde{e}_i := n_k^{-1} \sum_{k=1}^{n_k} \|p(y(p_i) + \delta_{i,k}) - p_{\mathcal{D}}(y(p_i) + \delta_{i,k})\|$ with realizations $\delta_{i,k}$ of the measurement error distributed as $\mathcal{N}(0, \Sigma_l)$. The identified parameters $p(y(p_i) + \delta_{i,k})$ and $p_{\mathcal{D}}(y(p_i) + \delta_{i,k})$ have been computed by a Gauss–Newton method starting from p_i and using the true model and the surrogate, respectively. The Gauss–Newton iteration is only locally convergent, but in this setting, the problem's nonlinearity and the sampled errors $\delta_{i,k}$ are sufficiently small for the method to converge to the nearest local minimizer without further globalization. We construct histograms of the ratio $e_i/\tilde{e}_i$. Ratios less than 1 indicate an underestimation of the error, while values greater than 1 indicate an overestimation. The results in Figure 5.4 suggest that value-based and gradient-enhanced GPR surrogates behave similar. The coarse a priori error estimate is not strictly reliable, but appears to be reasonably accurate for steering the adaptive design process.

Parameter reconstruction

For an exemplary parameter reconstruction, we assume a true parameter $p^* = (1, 1.5)$ and exact measurements. Table 5.1 shows the reconstructed parameter p lying well within the prescribed global tolerance with comparable errors.

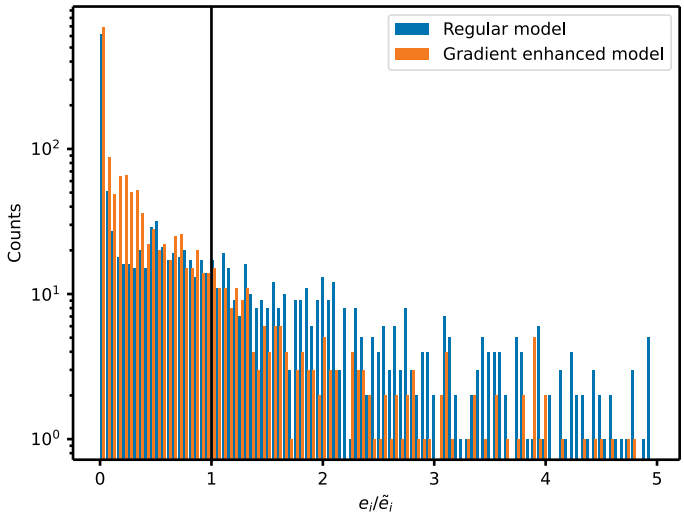


Figure 5.4: Log-histogram of $e_i/\tilde{e}_i$ for value-based GPR (blue) and GEGPR (orange) surrogates.

Table 5.1: Reconstruction result p using the surrogate model for $\Delta W = 10^4$ to be compared with the true parameter p^*.

	p_1	$\lvert p_1 - p_1^* \rvert$	p_2	$\lvert p_2 - p_2^* \rvert$
value	0.9968	0.0032	1.496	0.0047
value+gradient	0.9978	0.0022	1.502	0.0023

4.2 FEM example

Scatterometry is a more complex problem from optical metrology [12, 28, 9]. The aim is to identify geometric parameters in a nano-structured diffraction pattern from the intensity of reflected monochromatic light of different polarizations and incidence angles ϕ, θ; see Figure 5.5 and [37]. A forward model $y(p)$ was developed using JCMsuite[1] as a solver for the governing Maxwell's equations. The model is parametrized by the geometry of the line grid sample; see Figure 5.5 and Table 5.2.

The numerical model maps model parameters $p = [\mathrm{cd}, t, r_{\mathrm{top}}, r_{\mathrm{bot}}] \in \mathcal{X} \subset \mathbb{R}^4$ to $m = 4$ zero'th-order scattering intensities $y(p) \in \mathbb{R}^4$, given by incrementing θ from 5° to 11° in steps of 2°, utilizing P-polarized light with an angle of incidence $\phi = 0°$.

At the beginning, $\mathcal{X}$ is covered with $2^7 = 128$ points from a Sobol sequence. According to the known measurement uncertainty, we set $\Sigma_l = 10^{-2}\,\mathrm{diag}(8.57, 8.56, 8.60, 8.63)$ and ask for $E(\mathcal{D}) \leq \mathrm{TOL} = 10^{-3}$.

1 https://www.jcmwave.com/jcmsuite

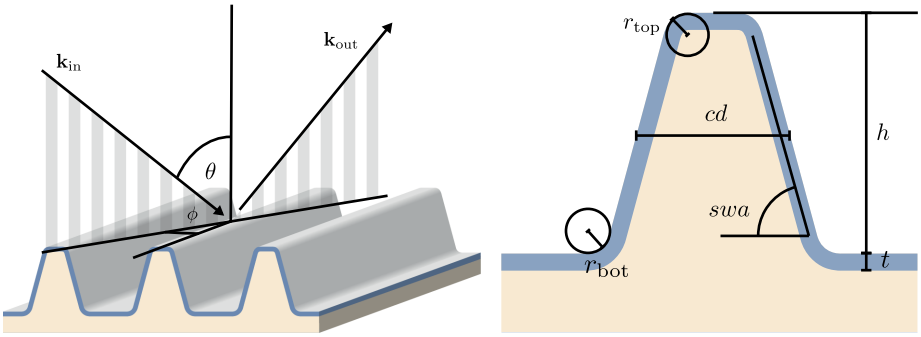


Figure 5.5: *Left*: Scatterometry setup used for the characterization of periodic nanostructures on surfaces. The incident light is varied at angles θ and ϕ and the diffraction patterns are recorded. *Right*: Geometry parametrized in terms of radii r_{top} and r_{bot}, height of the grating, side wall angle (swa), critical dimension (cd), and oxide layer thickness (t).

Table 5.2: The parameter domain $\mathcal{X}$ for the scatterometry problem.

Parameter	**cd**	***t***	r_{top}	r_{bot}	***h***	**swa**
range	[24, 28] nm	[4, 6] nm	[8, 10] nm	[3, 7] nm	48.3 nm	87.98°

Convergence

In Figure 5.6, we plot the estimated global error against the computational work. On the left, we a GPR design (dashed) with a GEGPR design (solid) for the same incremental budget ΔW. As in the analytical example, including gradient information, improves the efficiency at the cost of a nonmonotone convergence behavior. The improved efficiency is consistent, though quantitatively highly varying, over a wide range of incremental

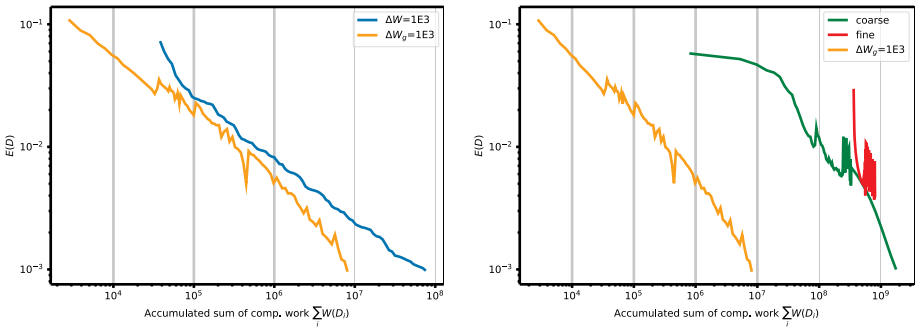


Figure 5.6: Estimated global error $E(\mathcal{D})$ versus computational work. *Left*: $E(\mathcal{D})$ for the same incremental work $\Delta W = \Delta W_g$. The subscript g indicates a gradient enhanced surrogate model. *Right*: Two different fixed evaluation accuracies within a position-adaptive algorithm compared to a fully adaptive gradient-enhanced design for $\Delta W_g = 10^3$.

Table 5.3: Observed performance improvement factor W/W_g due to including gradient information in the adaptive design selection for different incremental budgets.

ΔW	10^3	10^4	10^5	10^6
W/W_g	10	500	10	50

work budgets, as shown in Table 5.3. Note that the performance gain factor 500 for $\Delta W = 10^4$ is a lucky outlier and suggests that the actual performance gain is subject to high variance.

In Figure 5.6 right, we compare the presented approach with position-adaptive designs using fixed-finite element grids. The coarse grid has a maximum edge length of $h_c = 16\,\text{nm}$, the fine grid has a maximum edge length of $h_f = 1\,\text{nm}$, corresponding to FE discretization errors $\epsilon_c = 10^{-2}$ and $\epsilon_f = 10^{-4}$, respectively. When commencing with the coarse grid, the error reduction in the curve is initially gradual. However, it significantly accelerates beyond a certain threshold, identified as $W \approx 3 \cdot 10^7$. Interestingly, beyond this threshold, the presence of pronounced oscillations becomes evident, causing the error to temporarily plateau between $W = 2 \cdot 10^8$ and $W = 4 \cdot 10^8$. Subsequently, the curve resumes its descent and eventually achieves the desired tolerance level around $W \approx 2 \cdot 10^9$. Notably, during the analysis, it was observed that within the aforementioned oscillation range, the hyperparameter optimization for a specific parameter component failed to yield satisfactory results. As a result, it was necessary to set this parameter to a default value of $L = 1$, providing a plausible explanation for the observed behavior.

Surprisingly, we encountered difficulties in achieving convergence for the fine grid. Despite an initial rapid decrease in error, the curve exhibited pronounced oscillations across various settings, necessitating the termination of the procedure. Additional tests involving different constant hyperparameters and varied hyperparameter bounds within the optimization failed to yield any improvements. Consequently, further research is required to comprehensively elucidate and resolve this issue.

Thus, in the direct comparison between fully adaptive and semiadaptive algorithm, we can see that we can save computational work by a factor of ≈ 100.

Reliability of local error estimator

As before, we employ $7^4 = 2401$ parameter points to generate the estimated local errors e_i and the expected true errors $\tilde{e}_i$ with and without gradient information. A FE simulation on a fine grid with maximum edge length of 1 nm is used as the exact forward model. In contrast to the analytical example, the GPR and GEGPR surrogates differ slightly, with GPR consistently overestimating the true errors; see Figure 5.7. This suggests that the GPR model should provide smaller actual errors than aimed at. Again, the GEGPR error estimator is not strictly reliable, but apparently sufficiently robust for steering the adaptive design selection process.

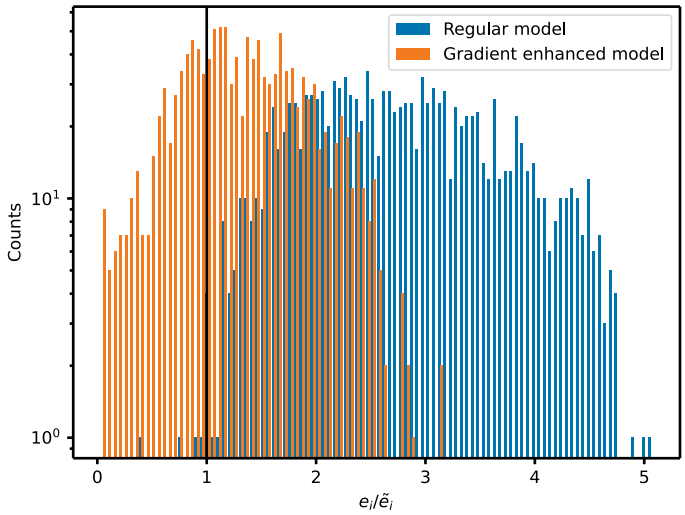


Figure 5.7: Log-histogram of $e_i \cdot \tilde{e}_i^{-1}$. The blue histogram was generated for the regular model, while the orange histogram was generated for the gradient-enhanced model.

Parameter reconstruction

We perform an exemplary parameter reconstruction with true parameters $p_{\text{true}} = (26.0, 5.0, 10.5, 5.0)$ nm and simulated measurements with artificial measurement noise of size 10^{-3}. The parameters are recovered within the imposed tolerance, as shown in Table 5.4.

Table 5.4: Reconstruction results for the scatterometry problem.

Parameter	p_{CD}	$\lvert\Delta p_{\text{CD}}\rvert$	p_t	$\lvert\Delta p_t\rvert$	p_{top}	$\lvert\Delta p_{\text{top}}\rvert$	p_{bot}	$\lvert\Delta p_{\text{bot}}\rvert$
value	26.001	1.000E-3	4.992	0.878E-3	10.502	2.00E-3	4.991	9.20E-3
value+gradient	25.999	0.983E-4	4.999	0.812E-4	10.499	0.87E-4	4.999	1.54E-4

5 Conclusion

The joint adaptive selection of evaluation positions and evaluation tolerances with a greedy heuristic improves the efficiency of building a GPR surrogate from finite element simulation significantly. Including gradient information enhances this further if derivatives can be computed cheaply. In numerical experiments, improvement factors between 100 and 1000 have been observed compared to methods relying on only selecting evaluation positions. While the error estimator based on the GPR variance is not strictly reliable, it appears to be sufficiently well suited for steering the adaptive design selection.

Several open questions remain, and need to be addressed in the future, such as the higher sensitivity of gradient-enhanced surrogate models to hyperparameter optimization, leading to less desirable nonmonotone convergence. One limitation of the approach is the assumption of normally and independently distributed evaluation errors with vanishing mean, which is quite obviously not realistic. The use of non-Gaussian stochastic process regression as well as biased evaluation errors correlated in parameter space are therefore of high interest. Moreover, besides real-time parameter identification, surrogate models are also used in sampling posterior distributions with Markov chain Monte Carlo methods. Here, a different notion of error and a different organization of forward model evaluation are necessary [33].

Funding
This work has been supported by Bundesministerium für Bildung und Forschung – BMBF, project number 05M20ZAA (siMLopt).

Bibliography

[1] T. Bai, A. L. Teckentrup, and K. C. Zygalakis. Gaussian processes for Bayesian inverse problems associated with linear partial differential equations. arXiv:2307.08343, 2023.
[2] P. R. Conrad, M. A. Girolami, S. Särkkä, A. M. Stuart, and K. C. Zygalakis. Statistical analysis of differential equations: introducing probability measures on numerical solutions. *Statistics and Computing*, 27:1065–1082, 2017.
[3] K. Crombecq, E. Laermans, and T. Dhaene. Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research*, 214:683–696, 2011.
[4] P. Deuflhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Computational Mathematics, volume 35. Springer, 2004.
[5] P. Deuflhard and M. Weiser. *Adaptive Numerical Solution of PDEs*. de Gruyter, 2012.
[6] D. Duvenau. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.
[7] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer, 1996.
[8] D. Eriksson, K. Dong, E. Lee, D. Bindel, and A. Wilson. Scaling gaussian process regression with derivatives. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
[9] N. Farchmin, M. Hammerschmidtm, P. I. Schneider, M. Wurm, B. Bodermann, M. Bär, and S. Heidenreich. Efficient global sensitivity analysis for silicon line gratings using polynomial chaos. In: B. Bodermann, K. Frenner and R. M. Silver, editors, *Modeling Aspects in Optical Metrology VII*, volume 11057, page 110570J. International Society for Optics and Photonics, SPIE, 2019.
[10] A. Giunta, S. Wojtkiewicz, and M. Eldred. Overview of modern design of experiments methods for computational simulations (invited). In: *41st Aerospace Sciences Meeting and Exhibit, AIAA 2003-649*, pages 1–17, 2003.
[11] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.
[12] M. Hammerschmidt, M. Weiser, X. Garcia Santiago, L. Zschiedrich, B. Bodermann, and S. Burger. Quantifying parameter uncertainties in optical scatterometry using Bayesian inversion. In: B. Bodermann, K. Frenner and R. M. Silver, editors, *Modeling Aspects in Optical Metrology VI*, volume 10330 page 1033004. International Society for Optics and Photonics, SPIE, 2017.

[13] P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A*, 471:20150142, 2015.
[14] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.
[15] V. Joseph and Y. Hung. Orthogonal-maximin latin hypercube designs. *Statistica Sinica*, 18:171–186, 2008.
[16] J. Kaipio and E. Somersalo. *Statistical and Computational Inverse Problems*. Springer, 2005.
[17] M. Kuß. *Gaussian process models for robust regression, classification, and reinforcement learning*. PhD Thesis, Technische Universität Darmstadt, 2006.
[18] R. Lehmensiek, P. Meyer, and M. Müller. Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits. *International Journal of RF and Microwave Computer-Aided Engineering*, 12(4):332–340, 2002.
[19] A. M. Mathai and S. B. Provost. *Quadratic Forms in Random Variables*. Marcel Dekker, 1992.
[20] J. Močkus. On Bayesian methods for seeking the extremum. In: *Optimization Techniques IFIP Technical Conference Novosibirsk*, pages 400–404. Springer, 1975.
[21] I. Neitzel, K. Pieper, B. Vexler, and D. Walter. A sparse control approach to optimal sensor placement in PDE-constrained parameter estimation problems. *Numerische Mathematik*, 143:943–984, 2019.
[22] J. Nitzler, J. Biehler, N. Fehn, P.-S. Koutsourelakis, and A. Wall. A generalized probabilistic learning approach for multi-fidelity uncertainty quantification in complex physical simulations. *Computer Methods in Applied Mechanics and Engineering*, 400:115600, 2022.
[23] N. Queipo, R. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, 2005.
[24] J. Quiñonero-Candela, C. E. Rasmussen, and C. K. I. Williams. Approximation methods for gaussian process regression. In: *Large-Scale Kernel Machines, Neural Information Processing*, pages 203–223. MIT Press, 2007.
[25] C. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
[26] G. Sagnol, H.-C. Hege, and M. Weiser. Using sparse kernels to design computer experiments with tunable precision. In: *Proceedings of COMPSTAT 2016*, pages 397–408, 2016.
[27] P. Schneider, M. Hammerschmidt, L. Zschiedrich, and S. Burger. Using Gaussian process regression for efficient parameter reconstruction. In: *Metrology, Inspection, and Process Control for Microlithography XXXIII*, volume 10959. SPIE, 2019.
[28] P. Schneider, M. Hammerschmidt, L. Zschiedrich, and S. Burger. Using Gaussian process regression for efficient parameter reconstruction. In: V. A. Ukraintsev and O. Adan, editors, *Metrology, Inspection, and Process Control for Microlithography XXXIII*, volume 10959, page 1095911. International Society for Optics and Photonics, SPIE, 2019.
[29] P. Semler and M. Weiser. Adaptive Gaussian process regression for efficient building of surrogate models in inverse problems. *Inverse Problems*, 39:125003, 2023.
[30] E. Solak, M. Roderick, W. E. Leithead, D. Leith, and C. Rasmussen. Derivative observations in gaussian process models of dynamic systems, pages 1033–1040, Jan. 2002.
[31] M. Sugiyama. Active learning in approximately linear regression based on conditional expectation of generalization error. *Journal of Machine Learning Research*, 7:141, 2006.
[32] B. Vexler. Adaptive finite element methods for parameter identification problems. In: H. G. Bock, T. Carraro, W. Jäger, S. Körkel, R. Rannacher and J. P. Schlöder, editors, *Model Based Parameter Estimation: Theory and Applications*, pages 31–54. Springer, 2013.
[33] P. Villani, J. F. Unger, and M. Weiser. Adaptive gaussian process regression for bayesian inverse problems. In: *Proc. Algoritmy 2024*, pages 214–224, 2024.
[34] M. Weiser and S. Ghosh. Theoretically optimal inexact spectral deferred correction methods. *Communications in Applied Mathematics and Computational Science*, 13(1):53–86, 2018.
[35] A. Wu, M. C. Aoi, and J. W. Pillow. Exploiting gradients and Hessians in Bayesian optimization and quadrature. arXiv:1704.00060, 2017.

[36] J. Wu, M. Poloczek, A. G. Wilson, and P. I. Frazier. Bayesian optimization with gradients. In: *Advances in Neural Information Processing Systems*, volume 30, pages 3–6. Curran Associates, 2017.

[37] M. Wurm, S. Bonifer, B. Bodermann, and J. Richter. Deep ultraviolet scatterometer for dimensional characterization of nanostructures: system improvements and test measurements. *Measurement Science & Technology*, 22(9):094024, 2011.

[38] A. Zaytsev. Reliable surrogate modeling of engineering data with more than two levels of fidelity. In: *2016 7th International Conference on Mechanical and Aerospace Engineering (ICMAE)*, pages 341–345, 2016.

Alexander Heinlein, Amanda A. Howard, Damien Beecroft, and Panos Stinis

Multifidelity domain decomposition-based physics-informed neural networks and operators for time-dependent problems

Abstract: Multiscale problems are challenging for neural network-based discretizations of differential equations, such as physics-informed neural networks (PINNs) and operator networks. This can be (partly) attributed to the so-called spectral bias of neural networks. To improve the performance of PINNs for time-dependent problems, a combination of multifidelity stacking PINNs and domain decomposition-based finite basis PINNs is employed. In particular, to learn the high-fidelity part of the multifidelity model, a domain decomposition in time is employed. The performance is investigated for a pendulum and a two-frequency problem as well as the Allen–Cahn equation. It can be observed that the domain decomposition approach clearly improves the PINN and stacking PINN approaches. Finally, it is demonstrated that the FBPINN approach can be extended to multifidelity physics-informed deep operator networks.

Keywords: Multifidelity, domain decomposition, physics-informed neural network, deep operator networks

MSC 2020: 65M22, 65M55, 68T07

1 Introduction

Many problems arising in science and engineering exhibit a multiscale nature, with different processes taking place on various temporal and spatial scales. The solution of

Acknowledgement: This project was completed with support from the U. S. Department of Energy, Advanced Scientific Computing Research program, under the Scalable, Efficient, and Accelerated Causal Reasoning Operators, Graphs and Spikes for Earth and Embedded Systems (SEA-CROGS) project (Project No. 80278). Pacific Northwest National Laboratory (PNNL) is a multiprogram national laboratory operated for the U. S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830. The computational work was performed using PNNL Institutional Computing.

Alexander Heinlein, Delft University of Technology, Delft Institute of Applied Mathematics, 4 Mekelweg, 2628 CD Delft, South Holland, Netherlands, e-mail: a.heinlein@tudelft.nl
Amanda A. Howard, **Panos Stinis,** Battelle Memorial Institute, P. O. Box 999, Richland, WA, 99352, USA, e-mails: amanda.howard@pnnl.gov, panos.stinis@pnnl.gov
Damien Beecroft, University of Washington, Applied Mathematics, 7462 Woodlawn Ave NE, Seattle, WA, 98195, USA, e-mail: dob1998@uw.edu

https://doi.org/10.1515/9783111376776-006

these problems is generally difficult for numerical methods. Multiscale methods have been developed to make numerical simulations of such problems feasible; examples are the multiscale finite element [12], homogeneous multiscale [9], generalized finite element [4], or variational multiscale [16] methods.

In recent years, inspired by the early work by Lagaris et al. [17], machine learning-based techniques for the solution of partial differential equations (PDEs) have been developed. In this paper, we focus on physics-informed neural networks (PINNs) [25]; other methods, such as the Deep Ritz method [10], have been developed around the same time. Those methods have many potential advantages: they are easy to implement, allow for direct integration of data, and can be employed to solve inverse and high-dimensional problems. However, their convergence properties are not yet well understood, and hence the resulting accuracy is often limited. This can be partly accounted to the spectral bias of neural networks (NNs) [24], meaning that NNs tend to learn low frequency components of functions much faster than high frequency components. In multiscale problems, the high frequency components typically correspond to the fine scales, whereas the low frequency components correspond to the coarse scales. Therefore, multiscale problems are also particularly challenging to solve using PINNs.

In this paper, we aim to combine two techniques that have recently been developed to improve the training of PINNs in this context. On the one hand, we consider the multifidelity training approach introduced for PINNs [21] and extended to Deep Operator Networks (DeepONets, [18]) in [14, 19, 6]. In particular, we consider the approach of stacked PINNs [13] in which multiple networks are stacked on top of each other, such that models on top of the stack may learn those features that are not captured by the previous models. On the other hand, we employ multilevel Schwarz domain decomposition neural network architectures [8], which are based on the finite-basis PINNs (FBPINNs) [22] approach. In this approach, the learning of multiscale features is improved by localization. In particular, the network architecture is decomposed, such that the individual parts of the network learn features on the corresponding spatial or temporal scale. For an overview on the combination of domain decomposition approaches and machine learning see, for instance, [11].

In related recent works, methods for iteratively training PINNs to progressively reduce the errors have been developed; see [1, 2, 3, 31]. These approaches, which vary in their implementation details, train each new network to reduce the residual from the previous network. In contrast, the work presented here trains for the entire solution at each iteration.

This paper is structured as follows: First, in Section 2, we describe the methodological framework. In particular, we first discuss PINNs in Section 2.1, then we describe multifidelity stacking PINNs in Section 2.2, as well as the domain decomposition approach in Section 2.3. Next, we introduce the specific domain decomposition in time employed in the model problems in Section 3. In Section 4, we present numerical results for several model problems, a pendulum, and a two-frequency problem as well as the Allen–Cahn equation. We extend the results to DeepONets in Section 5. We conclude with a brief

discussion of the current and future work in Section 6. All training parameters used to generate the results are given in Table 6.1.

2 Methodology

2.1 Physics-informed neural networks

We consider a generic differential equation-based problem in residual form: Find u such that

$$\begin{aligned} \mathcal{A}u &= 0 \quad \text{in } \Omega, \\ \mathcal{B}u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{6.1}$$

where $\mathcal{A}$ is a differential operator and $\mathcal{B}$ an operator for specifying the initial or boundary conditions. The solution u is defined on the domain Ω and should have sufficient regularity to apply $\mathcal{A}$ and $\mathcal{B}$. In order to solve Eq. (6.1), we follow [17] and employ a collocation approach. In particular, we exploit that solving Eq. (6.1) is equivalent to solving $\arg\min_{\mathcal{B}u=0 \text{ on } \partial\Omega} \int_\Omega (\mathcal{A}u(\mathbf{x}))^2 \, d\mathbf{x}$. We discretize the solution using a neural network $\hat{u}(\mathbf{x}, \theta)$, with parameters θ, and the integral is approximated by the sum

$$\underset{\mathcal{B}\hat{u}(\mathbf{x},\theta)=0 \text{ on } \partial\Omega}{\arg\min} \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2,$$

where the collocation points $\mathbf{x}_i$ are sampled from Ω. Different types of neural network architectures may be employed, and we will employ a combination of the approaches explained in Sections 2.2 and 2.3.

The initial or boundary conditions in the second equation of Eq. (6.1) can be enforced via hard or soft constraints. In the approach of hard constraints, they are explicitly implemented in the neural network function; cf. [17]. In this paper, we employ the approach of soft constraints instead, in which we incorporate the constraints into the loss function:

$$\underset{\theta}{\arg\min}\ \lambda_r \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_{bc} \sum_{\mathbf{x}_i \in \partial\Omega} \left(\mathcal{B}\hat{u}(\mathbf{x}_i, \theta)\right)^2 \tag{6.2}$$

Here, λ_r and λ_{bc} weight the residual of the differential equation and the initial and boundary conditions in the loss function, respectively. As discussed, for instance in [28], an appropriate weighting is crucial for the convergence in optimizing Eq. (6.2) using a gradient-based optimization method. θ denotes all the trainable parameters in the network.

This approach has also been denoted as physics-informed neural networks (PINNs) in [25].

2.2 Multifidelity stacking PINNs

Multifidelity PINNs use two NNs to learn the correlation between low- and high-fidelity physics [21]. The goal is to train a linear network (with no activation function) to learn the linear correlation between the low- and high-fidelity models, and a nonlinear network to learn the nonlinear correlation. By training a linear network, the resulting model is more expressive than just assuming that the correlation between the models is the identity. Moreover, under the assumption that the main part of the correlation is linear, separating the network into the linear and nonlinear parts allows for a smaller nonlinear network.

To train a multifidelity PINN, we first train a standard single fidelity PINN $\hat{u}^{\mathrm{SF}}(\mathbf{x}, \theta^{\mathrm{SF}})$. In a second step, we then train a multifidelity network $\hat{u}^{\mathrm{MF}}$, which consists of linear and nonlinear subnetworks that learn the correlation between the single fidelity PINN $\hat{u}^{\mathrm{SF}}(\mathbf{x}, \theta^{\mathrm{SF}})$ and the solution:

$$\hat{u}^{\mathrm{MF}}(\mathbf{x}, \theta^{\mathrm{MF}}) = (1 - |\alpha|)\hat{u}^{\mathrm{MF}}_{\mathrm{linear}}(\mathbf{x}, \hat{u}^{\mathrm{SF}}, \theta^{\mathrm{MF}}) + |\alpha|\hat{u}^{\mathrm{MF}}_{\mathrm{nonlinear}}(\mathbf{x}, \hat{u}^{\mathrm{SF}}, \theta^{\mathrm{MF}}). \tag{6.3}$$

The linear network does not have activation functions to force learning a linear correlation, and can be very small. α is a trainable parameter to enforce maximizing the linear correlation.

The loss function in Eq. (6.2) is modified to include the penalty α^4:

$$\arg\min_{\theta} \lambda_r \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_{bc} \sum_{\mathbf{x}_i \in \partial\Omega} \left(\mathcal{B}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_a \alpha^4 \tag{6.4}$$

In multifidelity stacking PINNs as presented in [13], multifidelity PINNs are trained recursively, each taking the output of the previously trained stacking layer as input. In this way, the previous layer serves as the low-fidelity model for the new stacking layer. The difference between [13] and the current work is that [13] does not consider domain decomposition, so each stacking layer has a single multifidelity PINN covering the entire domain. The approach considered here is more flexible, and as we will show in Section 4, results in smaller relative errors when trained on the same equations.

2.3 Domain decomposition-based neural network architectures

It has been observed in [22] that the high frequency components in the solution can be learned better if a domain decomposition is introduced into the PINN approach. To scale to larger numbers of subdomains, this approach has first been extended to two-levels in [7] and then to an arbitrary number of levels in [8]. The general idea of the domain decomposition-based finite basis PINNs (FBPINNs) is to decompose the computational domain Ω into J overlapping subdomains Ω_j, $\Omega = \bigcup_{j=1}^{J} \Omega_j$. As before, Ω may be a space-time domain, and in this work we will focus on domain decomposition in time. On each subdomain, we define a space of network functions $\mathcal{V}_j = \{\hat{u}_j(\mathbf{x}, \theta_j) \mid \mathbf{x} \in \Omega_j, \theta_j \in \Theta_j\}$,

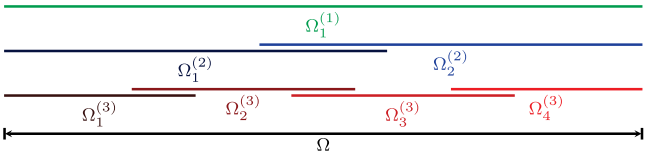


Figure 6.1: Multilevel overlapping domain decomposition of Ω with $L = 3$ levels.

where $\hat{u}_j(\mathbf{x}, \theta_j)$ denotes a PINN model, $\Theta_j = \mathbb{R}^{k_j}$ is the space of all trainable neural network parameters, and k_j is the number of network parameters.

In order to represent the global solution of a given problem, we define window functions ω_j with $\text{supp}(\omega_j) \subset \Omega_j$ such that $\{\omega_j\}_{j=1}^J$ form a partition of unity, that is, $\sum_{j=1}^J \omega_j = 1$ on Ω. Then we can define a global neural network space $\mathcal{V} = \sum_{j=1}^J \omega_j \mathcal{V}_j$, and the global FBPINN function reads $\hat{u}(\mathbf{x}, \theta) = \sum_{j=1}^J \omega_j \hat{u}_j(\mathbf{x}, \theta_j)$. It has been observed that this approach may significantly improve the performance of PINNs; cf. [22]. However, similar to classical domain decomposition methods [27], the one-level approach is not scalable to large numbers of subdomains; see [7, 8].

To improve the scalability and the performance for multiscale problems, a hierarchy of domain decompositions may be employed. Define L levels of domain decompositions, with the overlapping domain decomposition at level l denoted by $D^{(l)} = \{\Omega_j^{(l)}\}_{j=1}^{J^{(l)}}$, where $\Omega = \bigcup_{j=1}^{J^{(l)}} \Omega_j^{(l)}$ and $J^{(l)}$ is the number of subdomains at level l; cf. Figure 6.1. Even though there is generally no restriction on the overlapping domain decompositions, we choose $J^{(1)} = 1$, so the first level corresponds to a single global subdomain, and $J^{(l)} < J^{(l+1)}$ for all $l = 1, \ldots, L$.

Now, on each level l we define window functions $\omega_j^{(l)}$ to be a partition of unity, so $\sum_{j=1}^{J^{(l)}} \omega_j^{(l)} = 1$, and $\text{supp}(\omega_j^{(l)}) \subset \Omega_j^{(l)}$. Similar to the one-level case, this yields the global neural network space $\mathcal{V} = \sum_{l=1}^L \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \mathcal{V}_j^{(l)}$ and the global network function defined in terms of $\theta = \bigcup_{l=1}^L \theta^{(l)}$ and $\theta^{(l)} = \bigcup_{j=1}^{J^{(l)}} \theta_j^{(l)}$:

$$\hat{u}(\mathbf{x}, \theta) = \frac{1}{L} \sum_{l=1}^{L} \hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) \quad \text{with } \hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) = \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \hat{u}_j^{(l)}(\mathbf{x}, \theta_j^{(l)}). \tag{6.5}$$

It has been observed in [7, 8] that due to increased communication between the subdomain models, the multilevel FBPINN approach may significantly improve the performance over the one-level approach.

2.4 Stacking FBPINNs

We combine the multifidelity stacking PINNs and the FBPINNs as follows: In the first level, we train a standard single fidelity PINN across the full domain $\Omega^{(0)} = \Omega$. Then, for each level $l > 0$, we use a FBPINN network architecture modified to consist of multifi-

delity networks that takes as input the network from the previous level $l-1$:

$$\hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) = \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \hat{u}_{j,MF}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}). \tag{6.6}$$

We note that Eq. (6.6) differs from Eq. (6.5) by a factor of $1/L$, because the output of FBPINNs is the sum of the networks trained at all levels, while the output of stacking FBPINNs is the sum of the networks for the final level. The networks at level l learn the correlation between the output of the $l-1$ level and the solution, and take as input the previously learned solution $\hat{u}^{(l-1)}(\mathbf{x}, \theta^{(l-1)})$:

$$\hat{u}_{j,MF}^{(l)}(\mathbf{x}, \theta_j^{(l)}) = (1-|\alpha|)\hat{u}_{j,\text{lin}}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}) + |\alpha|\hat{u}_{j,\text{nonlin}}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}).$$

3 Domain decomposition in time

In this work, we are particularly interested in cases where classical PINNs fail to learn the temporal evolution, such as a damped pendulum and the Allen–Cahn equation. We consider a domain $\Omega = \mathbf{X} \times [0, T]$ where $\mathbf{X}$ denotes the spatial domain and $T \in \mathbb{R}$. Therefore, for the stacking FBPINN approach, we consider the domain decomposition in time:

$$\Omega_j^{(l)} = \left[\frac{(j-1)T - \delta T/2}{J^{(l)} - 1}, \frac{(j-1)T + \delta T/2}{J^{(l)} - 1} \right],$$

where $\delta > 1$ is the overlap ratio. For $l = 0$, we take $\Omega_1^{(0)} = [0.5T - \delta T/2, 0.5T + \delta T/2]$. The partition of unity functions are given by $\omega_j^{(l)} = \frac{\hat{\omega}_j^{(l)}}{\sum_{j=1}^{J^{(l)}} \hat{\omega}_j^{(l)}}$, where

$$\hat{\omega}_j^{(l)}(t) = \begin{cases} 1 & l = 0, \\ [1 + \cos(\pi(t - \mu_j^{(l)})/\sigma_j^{(l)})]^2 & l > 0, \end{cases} \tag{6.7}$$

$\mu_j^{(l)} = T(j-1)/(J^{(l)} - 1)$, and $\sigma_j^{(l)} = (\delta T/2)/(J^{(l)} - 1)$. For simplicity, we take $J^{(l)} = 2^l$ in each case. An illustration of the window functions for $T = 1$ and $l = 2$ ($J^{(2)} = 4$) is given in Figure 6.2.

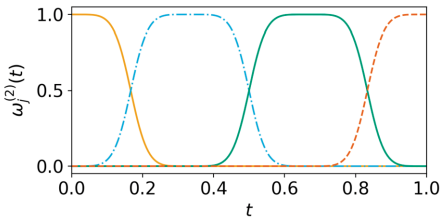


Figure 6.2: Window functions ω_j for $l = 2$ and $T = 1$.

As set up, each network only covers a small part of the time domain. To ease training, we scale the input in each domain to be in the range $[-1, 1]$ by using a scaled time $\hat{t} = t(l-1)/T - j$ as the input to network j. This scaling improves the robustness of the training.

In our applications, we calculate the relative ℓ_2 error $\frac{\|u(\mathbf{x}) - \hat{u}(\mathbf{x}, \theta)\|_2}{\|u(\mathbf{x})\|_2}$ where u denotes the exact solution and $\hat{u}$ denotes the output from the multifidelity FBPINN.

4 Results

4.1 Pendulum

While a relatively simple system, accurately training a PINN to predict the movement of a pendulum for long times presents challenges [29]. The pendulum movement is governed by a system of two first-order ODEs for $t \in [0, T]$,

$$\frac{ds_1}{dt} = s_2, \tag{6.8}$$

$$\frac{ds_2}{dt} = -\frac{b}{m} s_2 - \frac{g}{L} \sin(s_1), \tag{6.9}$$

where s_1 and s_2 are the position and velocity of the pendulum, respectively. We employ the same parameters used in [29], that is, $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$. We take $s_1(0) = s_2(0) = 1$. We compare the results with those for the stacking PINN from [13], which uses the same multifidelity architecture but only a single PINN on each level. As shown in Figure 6.3, the stacking FBPINN is able to reach a significantly lower relative ℓ_2 error. In addition, each network in the stacking FBPINN is significantly smaller than the networks used in the stacking PINN with the result that, at three stacking layers, the stacking FBPINN reaches a relative ℓ_2 error of $7.4 \cdot 10^{-3}$ with only 34 570 trainable parameters. In comparison, the best case stacking PINN from [13] requires four stacking levels to reach a relative ℓ_2 error of $1.3 \cdot 10^{-2}$ with 63 018 trainable parameters.

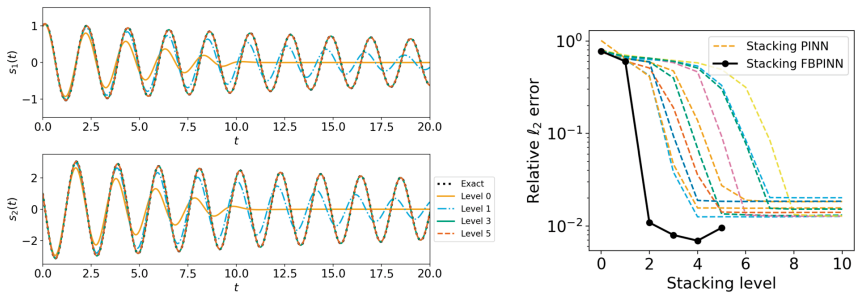


Figure 6.3: Stacking FBPINN results for the pendulum problem: **Left:** Stacking FBPINN results for an illustrative example of s_1 (top) and s_2 (bottom) as a function of time for the pendulum problem up to five stacking FBPINN levels. **Right:** Pendulum relative ℓ_2 training errors comparing the work in the current paper (solid line) with the approach from [13] (dashed lines).

4.2 Multiscale problem

We now consider a toy model problem with a low and high frequency component, inspired by [22]:

$$\begin{aligned} \frac{ds}{dx} &= \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x), \\ s(0) &= 0, \end{aligned}$$

on domain $\Omega = [0, 20]$ with $\omega_1 = 1$ and $\omega_2 = 15$. The exact solution for this problem is $s(x) = \sin(\omega_1 x) + \sin(\omega_2 x)$.

The results are shown in Figure 6.4. After two stacking levels, the stacking FBPINN reaches a relative ℓ_2 error of $4.2 \cdot 10^{-3}$, with 7 822 trainable parameters. A comparable relative ℓ_2 error of $6.1 \cdot 10^{-3}$ is reached after 10 stacking levels with a stacking PINN with 11 179 trainable parameters. Also shown in Figure 6.4 (right) is the best case SF network from [13], which has a relative ℓ_2 error of $9.5 \cdot 10^{-2}$ with 16 833 trainable parameters. The stacking FBPINN outperforms the SF PINN with an error more than an order of magnitude lower, with less than half the trainable parameters. Additionally, the final stacking FBPINN reaches a relative ℓ_2 error of $8.3 \cdot 10^{-4}$, an order of magnitude lower than the final stacking PINN.

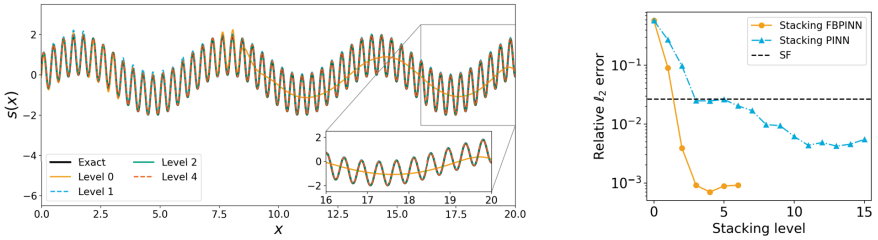


Figure 6.4: Stacking FBPINN results for the multiscale problem: **Left:** Stacking FBPINN results for the single fidelity level 0 and the first four stacking FBPINN levels. **Right:** Multiscale relative ℓ_2 training errors comparing the work in the current paper with [13].

4.3 Allen–Cahn equation

Our third example is based on the Allen–Cahn equation and is given by

$$\begin{aligned} s_t - 0.0001 s_{xx} + 5s^3 - 5s = 0, \qquad & t \in (0, 1], x \in [-1, 1], \\ s(x, 0) = x^2 \cos(\pi x), \qquad & x \in [-1, 1], \\ s(x, t) = s(-x, t), \qquad & t \in [0, 1], x = -1, x = 1, \\ s_x(x, t) = s_x(-x, t), \qquad & t \in [0, 1], x = -1, x = 1. \end{aligned}$$

The Allen–Cahn equation presents difficulties for PINNs when attempting to learn the full solution from $t = 0$ to 1 with a single PINN; see, for example, [32, 20, 26].

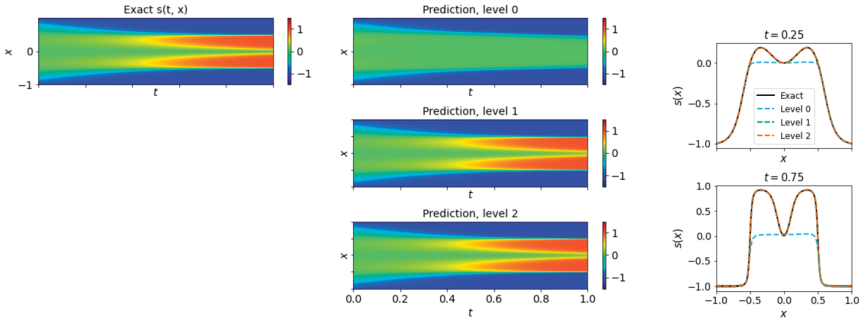


Figure 6.5: Stacking FBPINN results for the Allen–Cahn equation. **Left:** Stacking FBPINN results for the single fidelity level 0 and the first two stacking FBPINN levels. **Right:** Line plots of the results from the stacking FBPINN at $t = 0.25$ (top) and $t = 0.75$ (bottom).

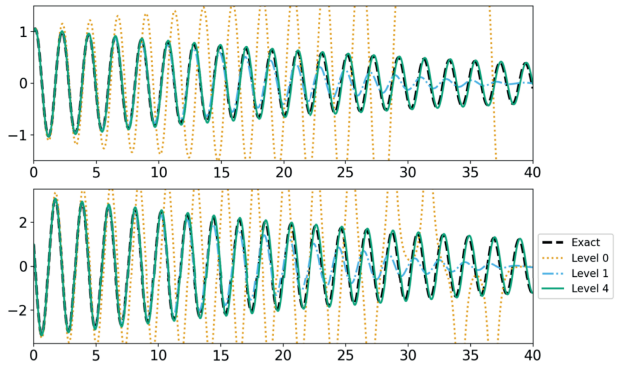
We solve the Allen–Cahn equation by dividing the time domain into subdomains, as presented in Section 3. The corresponding results for the stacking FBPINN are shown in Figure 6.5. The relative ℓ_2 error for applying two levels of the stacking FBPINN is $5.9 \cdot 10^{-3}$. Previously reported values for the relative error in literature include $1.68 \cdot 10^{-2}$ for the backward compatible PINN [20] and $2.33 \cdot 10^{-2}$ for PINNs with adaptive resampling [32].

5 Extension to DeepONets

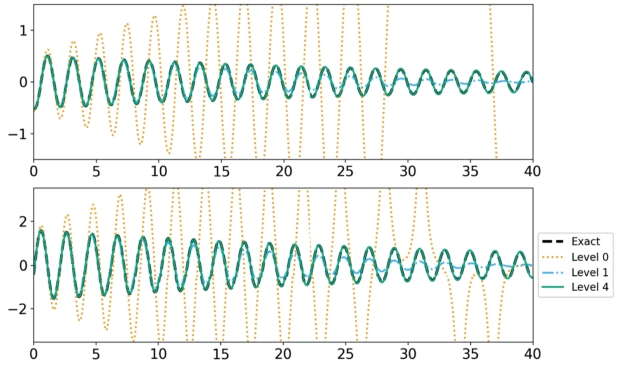
The method presented in Section 2 can be extended seamlessly to multifidelity stacking DeepONets from [14, 13]; we denote the resulting method as finite-basis DeepONets (FB-DONs). For the sake of brevity, we refer to [18, 14, 13] for details on the DeepONet approach. As an example, we present results for the pendulum problem in Section 4.1 and train a model mapping given initial conditions $(s_1(0), s_2(0))$ to the corresponding solution $(s_1(t), s_2(t))$ on the whole time interval $[0, 20]$. This is referred to as operator learning since we learn a mapping between the initial conditions and the solution space instead of a single solution. One each level l, $l > 0$, we train 2^l DeepONets with partition of unity functions as defined in Eq. (6.7). As training data, we employ 50 000 randomly chosen pairs $(s_1(0), s_2(0)) \in [-2, 2] \times [-1.2, 1.2]$, and the loss is given by Eq. (6.2) and the differential equations in Eqs. (6.8) and (6.9). After training, the resulting FB-DON model is then able to predict the solution for any initial condition in the training range, as shown in Figure 6.6. Training parameters are given in Table 6.2.

6 Discussion

In this paper, we have introduced the stacking FBPINN and FB-DON approaches. For the considered time-dependent problems, stacking FBPINNs yielded more accurate results than stacking PINNs alone, and in some cases, they additionally required fewer



(a) $s_1(0) = 1, s_2(0) = 1$



(b) $s_1(0) = -0.5, s_2(0) = -0.4$

Figure 6.6: Stacking FB-DON results for the pendulum system at two different sets of initial conditions.

total trainable parameters. This indicates that a domain decomposition in time can greatly improve the performance of stacking PINNs. In contrast to prior work on stacking PINNs and DeepONets, stacking FBPINNs and FB-DONs use a sum of subdomain networks weighted by the partition of unity functions on the corresponding level. In contrast to multilevel FBPINNS in [8], in which the subdomain networks are summed across all levels and trained simultaneously, the architecture and training of stacking FBPINNs and FB-DONs is sequential with respect to the levels; the idea is similar to multiplicative coupling as discussed in [7] but implemented differently using the stacking approach. This difference allows for stacking FBPINNs and FB-DONs to consider different equations on different levels, akin to simulated annealing, as considered in [13], or to consider different physical models at different length scales. We leave this for future work. The extension to stacking FB-DONs allows for use of physics-informed FB-DONs as surrogate models in place of traditional numerical solvers. The computation of a solution using a trained stacking FB-DONs is very efficient: it requires only one forward pass of the networks and, therefore, the computational time compared with classical

numerical solvers can be greatly reduced. One advantage of the FB-DON approach is that it can be used in conjunction with existing methods for increasing accuracy of physics-informed DeepONets, including long-time integration [29] and adaptive weighting schemes [30, 15, 23].

7 Training parameters

Table 6.1: Training parameters for the FBPINN results in the paper. The learning rate is set using the `exponential_decay` function in Jax [5] with the given learning rate and decay rate and 2 000 decay steps. The training parameters used for the stacking PINN results are given in [13].

	Section 4.1	**Section 4.2**	**Section 4.3**
Level 0 learning rate & decay rate	$5 \cdot 10^{-3}, 0.99$	$10^{-3}, 0.99$	$10^{-4}, 0.99$
Level 0 network width	100	32	100
Level 0 network layers	3	3	6
Level 0 iterations	200 000	200 000	200 000
Nonlinear network width	32	16	200
Nonlinear network layers	3	4	4
Linear network size	[2, 4, 2]	[1, 5, 1]	[1, 5, 1]
MF learning rate & decay rate	$5 \cdot 10^{-3}, 0.99$	$5 \cdot 10^{-3}, 0.95$	$5 \cdot 10^{-3}, 0.95$
BC batch size	1	1	128
Residual batch size	400	400	1024
Iterations	200 000	300 000	300 000
$\lambda_r, \lambda_{bc}, \lambda_a$	1.0, 1.0, 1.0	10.0, 1.0, 1.0	$10.0, 1.0, 10^{-5}$
Level 0 activation function	swish	swish	tanh
MF activation function	swish	swish	swish

Table 6.2: Training parameters for the FBDeepONet results in the paper. The learning rate is set using the `exponential_decay` function in Jax [5] with the given learning rate and decay rate and 2 000 decay steps.

	Section 5
Level 0 learning rate & decay rate	$5 \cdot 10^{-3}, 0.9$
Level 0 branch and trunk width	100
Level 0 branch and trunk layers	5
Level 0 iterations	100 000
Nonlinear branch and trunk width	100
Nonlinear branch and trunk layers	3
Linear branch and trunk width	10
Linear branch and trunk layers	1
MF learning rate & decay rate	$5 \cdot 10^{-3}, 0.9$
BC batch size	1 000
Residual batch size	10 000
Iterations	200 000
$\lambda_r, \lambda_{bc}, \lambda_a$	1.0, 1.0, 1.0
Level 0 activation function	sin
MF activation function	sin

Bibliography

[1] M. Ainsworth and J. Dong. Galerkin neural networks: a framework for approximating variational equations with error control. *SIAM Journal on Scientific Computing*, 43(4):A2474–A2501, 2021.

[2] M. Ainsworth and J. Dong. Galerkin neural network approximation of singularly-perturbed elliptic systems. *Computer Methods in Applied Mechanics and Engineering*, 402:115169, 2022.

[3] Z. Aldirany, R. Cottereau, M. Laforest, and S. Prudhomme. Multi-level neural networks for accurate solutions of boundary-value problems. arXiv:2308.11503, 2023.

[4] I. Babuška and J. E. Osborn. Generalized finite element methods: their performance and their relation to mixed methods. *SIAM Journal on Numerical Analysis*, 20(3):510–536, June 1983.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[6] S. De, M. Reynolds, M. Hassanaly, R. N. King, and A. Doostan. Bi-fidelity modeling of uncertain and partially unknown systems using deeponets. arXiv:2204.00997, 2022.

[7] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. Finite basis physics-informed neural networks as a Schwarz domain decomposition method. arXiv:2211.05560, November 2022.

[8] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. Multilevel domain decomposition-based architectures for physics-informed neural networks. arXiv:2306.05486 [cs, math], June 2023.

[9] W. E, B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden. Heterogeneous multiscale methods: a review. *Communications in Computational Physics*, 2(3):367–450, June 2007.

[10] W. E and B. Yu. The Deep Ritz Method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, March 2018.

[11] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021, 28 pp.

[12] T. Y. Hou and Y. Efendiev. *Multiscale Finite Element Methods: Theory and Applications*. Springer, New York, NY, 2009.

[13] A. A. Howard, S. H. Murphy, S. E. Ahmed, and P. Stinis. Stacked networks improve physics-informed training: applications to neural networks and deep operator networks. *Foundations of Data Science*, 7(1):134–162, March 2025.

[14] A. A. Howard, M. Perego, G. E. Karniadakis, and P. Stinis. Multifidelity deep operator networks for data-driven and physics-informed problems. *Journal of Computational Physics*, 493:112462, November 2023.

[15] A. A. Howard, S. Qadeer, A. W. Engel, A. Tsou, M. Vargas, T. Chiang, and P. Stinis. The conjugate kernel for efficient training of physics-informed deep operator networks. In *ICLR 2024 Workshop on AI4DifferentialEquations in Science*, 2024.

[16] T. J. R. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Computer Methods in Applied Mechanics and Engineering*, 127(1):387–401, November 1995.

[17] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, September 1998. Conference Name: IEEE Transactions on Neural Networks.

[18] L. Lu, P. Jin, and G. E. Karniadakis. DeepONet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. arXiv:1910.03193.

[19] L. Lu, R. Pestourie, S. G. Johnson, and G. Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

[20] R. Mattey and S. Ghosh. A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

[21] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: application to function approximation and inverse PDE problems. *Journal of Computational Physics*, 401:109020, 2020.

[22] B. Moseley, A. Markham, and T. Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, 49(4):62, July 2023.

[23] S. Qadeer, A. Engel, A. Tsou, M. Vargas, P. Stinis, and T. Chiang. Efficient kernel surrogates for neural network-based regression. arXiv:2310.18612, 2023.

[24] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. arXiv:1806.08734, May 2019.

[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[26] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger. On the role of fixed points of dynamical systems in training physics-informed neural networks. arXiv:2203.13648, 2022.

[27] A. Toselli and O. Widlund. *Domain Decomposition Methods—Algorithms and Theory*. Springer Series in Computational Mathematics, volume 34. Springer, Berlin, 2005.

[28] S. Wang, X. Yu, and P. Perdikaris. When and why PINNs fail to train: a neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, January 2022.

[29] S. Wang and P. Perdikaris. Long-time integration of parametric evolution equations with physics-informed DeepONets. *Journal of Computational Physics*, 475:111855, 2023.

[30] S. Wang, H. Wang, and P. Perdikaris. Improved architectures and training algorithms for deep operator networks. *Journal of Scientific Computing*, 92(2):35, 2022.

[31] Y. Wang and C.-Y. Lai. Multi-stage neural networks: function approximator of machine precision. arXiv:2307.08934, 2023.

[32] C. L. Wight and J. Zhao. Solving Allen–Cahn and Cahn–Hilliard equations using the adaptive physics informed neural networks. arXiv:2007.04542, 2020.

Timo Kreimeier, Andrea Walther, and Andreas Griewank

Constrained piecewise linear optimization by an active signature method

Abstract: In this paper, we consider the solution of optimization tasks with a piecewise linear objective function and piecewise linear constraints. We propose the so-called Constrained Active Signature Method (CASM) that explicitly exploits the given piecewise linear structure. Finite convergence of the algorithm is proven. Numerical results for three test cases including linear complementarity constraints illustrate the performance of CASM.

Keywords: Piecewise linear, constrained optimization, abs-linear form, linear independence kink qualification (LIKQ), active signature method (ASM)

MSC 2020: 49M05, 49M27, 65K05, 65K10, 90C26, 90C30, 90C57

1 Introduction and definitions

Motivated by numerous applications, there has been a growing interest in optimization problems that lack differentiability. One important class of such problems is given by piecewise linear functions, where corresponding optimization tasks arise, for example, as local models [16] or in the training of deep neural networks with the Rectified Linear Unit (ReLU) as the activation function [20].

So far, there is only a limited number of algorithms to solve constrained nonsmooth optimization problems available. Possible approaches comprise, for example, quasi-Newton methods [4] and bundle methods [17]. They all have in common that they do not exploit the structure that is available in the nonsmooth setting. For unconstrained optimization problems with piecewise linear (PL) objective functions, the so-called Active Signature Method (ASM) for determining local minima has been proposed in [8].

Acknowledgement: The authors thank the DFG for support within project B10 in the TRR 154 *Mathematical Modelling, Simulation and Optimization using the Example of Gas Networks* (project ID: 239904186). The research was funded partly by the DFG under Germany's Excellence Strategy—The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID:390685689).
The authors appreciate the constructive feedback of the reviewers. In addition, the authors would like to express their sincere thanks to Marc Steinbach for his detailed feedback on this optimization approach.
The data that support the findings of this study are available from the corresponding author upon request.
Conflicts of Interest: The authors declare no conflict of interest.

Timo Kreimeier, Andrea Walther, Andreas Griewank, Humboldt-Universität zu Berlin, 6 Unter den Linden, 10099 Berlin, Germany, e-mail: andrea.walther@math.hu-berlin.de

https://doi.org/10.1515/9783111376776-007

This approach explicitly builds on the nonsmooth structure yielding optimality conditions that can be verified in polynomial time. In [11], the general setting of nonlinear, so-called abs-smooth constrained optimization problems was considered and optimality conditions that can be also verified in polynomial time were given. However, no solution algorithm was proposed.

In this paper, an extension of ASM will be presented, which, in addition to the PL objective function, also takes PL (in)equality constraints into account. It exploits explicitly the nonsmooth structure of the optimization problem such that it can guarantee that a local minimizer is reached by the algorithm. Using the reformulations

$$\max(x_1, x_2) = \frac{1}{2}(x_1 + x_2 + |x_1 - x_2|) \quad \text{and}$$
$$\min(x_1, x_2) = \frac{1}{2}(x_1 + x_2 - |x_1 - x_2|)$$

as well as [19, Prop. 2.2.2], it follows that every continuous PL function can be represented in an abs-linear form as introduced for the first time in [6].

Definition 7.1. A continuous PL function $f\colon \mathbb{R}^n \to \mathbb{R}$ is in *abs-linear form* if $y = f(x)$ is given by

$$y = d + a^\top x + b^\top z, \tag{7.1a}$$
$$z = c + Zx + Mz + L|z|, \tag{7.1b}$$

with the *switching vector* $z \in \mathbb{R}^s$ containing the switching variables, and constants $d \in \mathbb{R}$, $a \in \mathbb{R}^n$, $b, c \in \mathbb{R}^s$, $Z \in \mathbb{R}^{s\times n}$, $L, M \in \mathbb{R}^{s\times s}$, where L and M are strictly lower triangular. Equation (7.1b) is called *a switching system*. The *signature vector* is defined as $\sigma(x) = (\mathbf{sgn}(z_1(x)), \ldots, \mathbf{sgn}(z_s(x))) \in \{-1, 0, 1\}^s$ and the *signature matrix* as $\Sigma(x) = \operatorname{diag}(\sigma(x))$.

Here and throughout, $|z|$ denotes the componentwise modulus of a vector z. Without loss of generality, we can set $d = 0$. In contrast to previous publications, for example, [8], a slightly different abs-linear form is defined to cover more general formulations of the target function and to unify the formulation of the constraints considered in this paper. Frequently, fixed signature vectors $\sigma \in \{-1, 0, 1\}$ and corresponding fixed signature matrices $\Sigma = \operatorname{diag}\sigma$ are considered that do not depend of x. Therefore, the dependence on x is explicitly stated if there is one.

It is possible to decompose the $\mathbb{R}^n$ into polyhedra as follows (cf. [9]): For a fixed $\sigma \in \{-1, 0, 1\}^s$ and the resulting signature matrix $\Sigma = \operatorname{diag}(\sigma)$, we define the *signature domain*

$$\mathcal{P}_\sigma := \{x \in \mathbb{R}^n \mid \mathbf{sgn}(z(x)) = \sigma\} \subset \overline{\mathcal{P}}_\sigma := \{x \in \mathbb{R}^n \mid \Sigma z(x) = |z(x)|\},$$

where $\overline{\mathcal{P}}_\sigma$ is called *extended signature domain*. The sets $\mathcal{P}_\sigma$ are given as inverse images of σ representing a disjoint decomposition of $\mathbb{R}^n$ into relatively open polyhedra. The

boundaries of the polyhedra $\mathcal{P}_\sigma$ are usually the sets where the PL function f is nonsmooth. Motivated by the graphical representation in low dimensions, these sets of points are called kinks.

The optimization problem for which we will present and analyze a solution algorithm, that is, the *constrained abs-linear optimization problem* (CALOP), has the structure

$$\begin{aligned} \min_{x\in\mathbb{R}^n, z\in\mathbb{R}^s} \quad & a^\top x + b^\top z \quad \text{such that} \\ 0 &= g + Ax + Bz + C|z|, \\ 0 &\geq h + Dx + Ez + F|z|, \\ z &= c + Zx + Mz + L|z|, \end{aligned} \tag{CALOP}$$

where $g \in \mathbb{R}^m$, $h \in \mathbb{R}^p$, $A \in \mathbb{R}^{m\times n}$, $B, C \in \mathbb{R}^{m\times s}$, $D \in \mathbb{R}^{p\times n}$, and $E, F \in \mathbb{R}^{p\times s}$. As can be seen, we assume that the objective function combined with the switching system in the last constraint is in abs-linear form; cf. Equation (7.1). For later use, we define $H\colon \mathbb{R}^n \to \mathbb{R}^p$, $(x, z, |z|) \mapsto h + Dx + Ez + F|z|$.

It is important to note that for the application of the algorithm proposed in this paper the user does not have to state the function evaluation in the form (CALOP), since correspondingly adapted AD tools like ADOL-C [7] can generate this representation from a given code segment evaluating the piecewise target function and the piecewise linear (in)equality constraints.

The remainder of the paper is organized as follows: In Section 2, the ASM is extended to cover also PL constrained optimization problems of the form (CALOP) as one main contribution of the paper. This includes also a statement on finite convergence for the new algorithm. Numerical results for different test problems are presented in Section 3. Finally, the paper concludes with a summary and an outlook in Section 4.

2 The constrained active signature method

In this section, we extend the ASM proposed in [8] to problems of the form (CALOP). Due to the additional equality and inequality constraints, the set of feasible points could be empty. Throughout, we assume that this is not the case such that the iteration can start with a feasible point. Subsequently, feasibility is maintained, that is, the derived algorithm is a feasible point method. If no feasible starting point is given from the application context, one can calculate such a starting point with a Phase-I-like method known from linear optimization (cf. [18, Chapter 16]).

2.1 The algorithm

Within an iteration, first a search direction is calculated as described in the following first paragraph. Then a step size is calculated as explained in the second paragraph. The third paragraph discusses the optimality condition. Based on these three main components, the entire algorithm is stated in the last paragraph of this subsection.

Computing a descent direction

To find a local minimum, the basic idea is to optimize a penalized version of the objective function on the polyhedra as defined by the signature vectors switching from one polyhedron to the next in an appropriate way. For this purpose, we add a quadratic penalty term with a positive definite matrix $Q = Q^\top \in \mathbb{R}^{n\times n}$ to the target function ensuring that there exists a minimizer on each polyhedron. Fixing one signature vector $\sigma \in \{-1, 0, 1\}^s$, we obtain from (CALOP) the restricted optimization problem

$$\min_{x\in\mathbb{R}^n, z\in\mathbb{R}^s} a^\top x + b^\top|\Sigma|z + \frac{1}{2}x^\top Qx \quad \text{such that} \tag{7.2a}$$

$$0 = g + Ax + +B|\Sigma|z + C\Sigma z, \tag{7.2b}$$

$$0 \geq h + Dx + E|\Sigma|z + F\Sigma z, \tag{7.2c}$$

$$0 = |\Sigma|z - \tilde{c} - \tilde{Z}x, \tag{7.2d}$$

$$0 \leq \Sigma z, \tag{7.2e}$$

with $\tilde{Z} = (I_s - M - L\Sigma)^{-1}Z$ and $\tilde{c} = (I_s - M - L\Sigma)^{-1}c$. Here, a fixed Q allows to apply CASM also for a broader range of problems. For example, when considering LASSO problems, the quadratic part of the LASSO problem can be coded using the matrix Q. For the unconstrained case, this approach was considered in [8]. Due to the fixed-signature vector, this optimization problem is smooth with a quadratic objective function and linear constraints. Applying standard KKT theory with Lagrange multipliers $\delta \in \mathbb{R}^m$, $\nu \in \mathbb{R}^p$, $\lambda \in \mathbb{R}^s$, and $\mu \in \mathbb{R}^s$, we obtain the following necessary optimality conditions:

$$0 = a^\top + x^\top Q + \delta^\top A + \nu^\top D - \lambda^\top \tilde{Z}, \tag{7.3a}$$

$$0 = b^\top|\Sigma| + \delta^\top\big(B|\Sigma| + C\Sigma\big) + \nu^\top\big(E|\Sigma| + F\Sigma\big) + \lambda^\top|\Sigma| - \mu^\top\Sigma, \tag{7.3b}$$

$$0 = g + Ax + B|\Sigma|z + C\Sigma z, \tag{7.3c}$$

$$0 \geq h + Dx + E|\Sigma|z + F\Sigma z, \tag{7.3d}$$

$$0 = |\Sigma|z - \tilde{c} - \tilde{Z}x, \tag{7.3e}$$

$$0 \leq \Sigma z, \quad 0 \leq \mu, \quad 0 = \mu^\top\Sigma z, \tag{7.3f}$$

$$0 \leq \nu, \quad 0 = \nu^\top\big(h + Dx + E|\Sigma|z + F\Sigma z\big). \tag{7.3g}$$

The optimization problem (7.2) could be solved using a standard QP method. However, we want to exploit the structure provided by the signature vector as an additional feature. Thus, multiplying Equation (7.3b) by Σ from the right and using Equation (7.3f) yield

$$0 \le \mu^\top |\Sigma| = b^\top \Sigma + \delta^\top \big(B\Sigma + C|\Sigma|\big) + \nu^\top \big(E\Sigma + F|\Sigma|\big) + \lambda^\top \Sigma. \tag{7.4}$$

Due to the complementarity condition $\mu^\top \Sigma z = 0$, this inequality must hold as an equality. Hence, it follows that

$$-b^\top \Sigma = \delta^\top \big(B\Sigma + C|\Sigma|\big) + \nu^\top \big(E\Sigma + F|\Sigma|\big) + \lambda^\top \Sigma.$$

Thus, with $\omega = \mathbf{sgn}(H(x, |z|))$ and $\Omega = \mathrm{diag}(\omega)$ denoting the projection onto the inactive inequality constraints, we get the linear system

$$\begin{bmatrix} Q & 0 & -\tilde{Z}^\top & A^\top & D^\top \\ 0 & 0 & \Sigma & \Sigma B^\top + |\Sigma| C^\top & \Sigma E^\top + |\Sigma| F^\top \\ \tilde{Z} & -|\Sigma| & 0 & 0 & 0 \\ A & B|\Sigma| + C\Sigma & 0 & 0 & 0 \\ \bar{\Omega} D & \bar{\Omega}(E|\Sigma| + F\Sigma) & 0 & 0 & \Omega \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{z} \\ \lambda \\ \delta \\ \nu \end{bmatrix} = - \begin{bmatrix} a \\ \Sigma b \\ \tilde{c} \\ g \\ \bar{\Omega} \end{bmatrix}, \tag{7.5}$$

where $\bar{\Omega} = I_p - |\Omega|$ forces the active inequalities to vanish. The matrix Ω in the right lower corner ensures that ν is zero for the inactive inequality constraints. Due to the assumption that a feasible starting point exists, Equation (7.5) always has a solution, which does not necessarily have to be unique. Denoting such a solution by $(\hat{x}, \hat{z}, \lambda, \delta, \nu)$, we define for the current iterate x and z the directions toward the next iterate as

$$\Delta x := \hat{x} - x \quad \text{and} \quad \Delta z := \hat{z} - z. \tag{7.6}$$

If $\Delta x = 0$, one has $\Delta z = 0$ due to the assumed abs-linear form. For $\Delta x = 0$ and $\Delta z = 0$, it follows from Equations (7.5) and (7.2e) that x and z satisfy the KKT conditions for the optimization task (7.2), that is, x is optimal on the current polyhedron defined by σ a situation denoted by *feasible signature optimal*.

Computing a step size β

For the solution $(\hat{x}, \hat{z})$ of Equation (7.5), we check if $\sigma(\hat{x}) = \sigma$ is still valid and that the inequality constraints of Equation (7.3) still hold to ensure feasibility. For this purpose, we calculate two step sizes. First, we consider the step length β^z from the current x in the direction Δx to a possible kink, that is, a sign change in one component of z, given by

$$\beta^z = \inf_{1 \le l \le s} \left\{ \beta_l^z = \frac{-z_l}{\hat{z}_l - z_l} \;\middle|\; (\hat{z}_l - z_l)\sigma_l < 0 \right\} \in (0, \infty]. \tag{7.7}$$

If $\beta^z < \infty$, the first index for which the minimum is attained is denoted by j^z. For $\beta^z < 1$, there exists a blocking kink. For $x \in \mathcal{P}_\sigma$ it follows for $\sigma_i \neq 0$ also that $z_i(x) \neq 0$. Thus, for σ with $\sigma_i \neq 0$, one has $z_i(x) \neq 0$. Hence, $\beta^z > 0$ must hold. Second, we consider the step length from the current x in direction Δx to a possible blocking inequality constraint $H_l(x, z, \Sigma z), 1 \le l \le p$. This step size β^H is given by

$$\beta^H = \inf_{1\le l\le p}\left\{\beta_l^H = \frac{H_l}{H_l - \hat{H}_l}\;\middle|\; (\hat{H}_l - H_l)\omega_l < 0\right\} \in (0,\infty], \tag{7.8}$$

where $H = H(x,z,\Sigma z)$, $\hat{H} = H(\hat{x},\hat{z},\Sigma\hat{z})$, and l denotes the lth component of H and $\hat{H}$, respectively. If $\beta^H < \infty, j^H$ is set to the smallest index for which the minimum is attained. For $\beta^H < 1$, there exists a blocking inequality constraint, that is, the solution $\hat{x}$ is not feasible. Therefore, the new iterate x^+ should be chosen such that the j^Hth components of $H(x^+,z^+,\Sigma z^+)$ and $\omega(x^+)$ drop to zero in comparison to $H(x,z,\Sigma z)$ and ω, respectively. Setting $\omega^+_{j^H} = 0$ changes the optimality system (7.3) and a new solution of Equation (7.5) has to be computed. If $\beta^H \le \beta^z$, then we have $z_{j^H}\hat{z}_{j^H} \ge 0$ and the iterate $\hat{x}$ is still contained in $\mathcal{P}_\sigma$, that is, $\sigma(\hat{x}) = \sigma$ is still valid. Since all inactive constraints are encoded in ω, one must have $\beta^H > 0$. The actual step size is determined as

$$\beta = \min\{\beta^z, \beta^H, 1\} \in (0,1], \tag{7.9}$$

where the upper bound 1 on β ensures with the update $x^+ = (1-\beta)x + \beta\hat{x} = x + \beta\Delta x$ that the next iterate is still feasible for the optimization task (7.2) and the current σ. As can be seen, the case $\beta < 1$ corresponds to the activation of a kink or inequality constraint, respectively. We will refer to this situation as a restriction of σ or ω, respectively. If $\beta = 1$, one has for the new iterate $x^+ = \hat{x}$ that $\sigma(x^+) = \sigma$ and $\omega(x^+) = \omega$. In this case, x^+ is called *signature stationary* since the two signature vectors are kept.

Checking the optimality

If x^+ is signature stationary on the current polyhedron $\mathcal{P}_\sigma$, one has to check whether x^+ is a minimizer of (CALOP). If this is the case, the iteration stops. Otherwise, the optimization continues in one of the neighboring polyhedra $\mathcal{P}_{\tilde{\sigma}}$ with $\tilde{\sigma} \succ \sigma$, where

$$\tilde{\sigma} \succeq \sigma \quad\Longleftrightarrow\quad \tilde{\sigma}_j\sigma_j \ge \sigma_j^2 \quad \text{for} \quad j = 1,\dots,s.$$

Such a $\tilde{\sigma}$ can be decomposed into $\sigma + \gamma$ where $|\sigma|^\top|\gamma| = 0$. Replacing Σ in the optimality conditions (7.3) by the corresponding $\Sigma+\Gamma$ and using $z_{\tilde{\sigma}}(x) = z_{\sigma+\gamma}(x) = (I_s - \tilde{L}\Gamma)^{-1}(\tilde{c}+\tilde{Z}x)$, most of the relations are still fulfilled by $\hat{x}$, $\hat{z}$ and λ. Just Equation (7.4) changes in that it has as many new nontrivial component as γ, which can be written as

$$0 \le \mu^\top|\Gamma| = \left(b^\top + \delta^\top B + \nu^\top E + \lambda^\top\right)\Gamma + \left(\delta^\top C + \nu^\top F - \lambda^\top\tilde{L}\right)|\Gamma|,$$

where $\Gamma = \operatorname{diag}(\gamma)$. This condition is violated if and only if there exists at least one index k such that $\gamma = -\mathbf{sgn}(b_k + \delta^\top Be_k + \nu^\top Ee_k + \lambda_k)e_k$ satisfies

$$0 > \left(\delta^\top C + \nu^\top F - \lambda^\top\tilde{L}\right)e_k - \left|b^\top + \delta^\top B + \nu^\top E + \lambda^\top\right|e_k \quad \text{and} \quad \sigma_k = 0, \tag{7.10}$$

which represented the strongest condition. In addition, we must check whether any of the components $\nu_l, 1 \le l \le p$ of the Lagrange multiplier ν associated with the inequality constraints is negative. If $\nu \ge 0$ does not hold, we choose the component for which $\nu \ge 0$

is most violated and drop the corresponding constraint. Hence, the associated entry of ω is set to -1 relaxing ω. If $\nu \geq 0$ holds, the current iterate is feasible signature optimal. Then we check the optimality condition given by Equation (7.10). If Equation (7.10) is not fulfilled for at least one index k, the current point is not a minimizer of (CALOP) and we leave a kink by relaxing σ setting the corresponding entry σ_k to a nonzero value via $\sigma^+ = \sigma + \gamma$.

The overall algorithm

Putting everything together, one obtains Algorithm 7.1 consisting of three parts: The computation of the search direction (cf. line 2), the determination of the step size and in case of blocking kinks and/or inequality constraints the restriction of σ and/or ω, respectively (cf. lines 3–6) and the verification of optimality and the relaxation of kinks or constraints if required (cf. lines 7–15). As can be seen from line 2 of Algorithm 7.1, in each iteration of CASM one system of linear equations, (see Equation (7.5)) has to be solved. The dimension of this system depends on the dimensions n, s, m, and p, that is, the number of variables, the number of kinks, as well as the number of equality and inequality constraints, respectively. The numerical examples in Section 3 serve as an illustration for this complexity. Depending on the software used, properties such as the block structure (see also [13, Lemma 4.9]) or sparsity can be exploited to reduce to overall computational effort. In the applications considered so far, the matrices occurring in (CALOP) are usually very sparse. The complexity of the remaining steps is negligible.

Algorithm 7.1 Constrained active signature method (CASM).

Require: Optimization problem (CALOP), feasible start point $x \in \mathbb{R}^n$, $\beta = 0$
Set: $z := z(x)$ via Equation (7.1), $\sigma := \sigma(x)$ and $\omega := \omega(x)$

1: **loop**
2: Compute $(\hat{x}, \hat{z}, \lambda, \delta, \nu)$ by solving Equation (7.5)
3: Compute β^z via Equation (7.7), β^H via Equation (7.8) and β via Equation (7.9)
4: Set $(x^+, z^+) = (1 - \beta)(x, z) + \beta(\hat{x}, \hat{z})$
5: **if** $\beta^H = \beta$ **then** Restrict ω
6: **if** $\beta^z = \beta$ **then** Restrict σ
7: **if** $\beta = 1$ **then**
8: **if** $\nu \not\geq 0$ **then**
9: Relax ω, set $\beta = 0$
10: **else**
11: **if** Equation (7.10) holds true **then**
12: Relax σ, set $\beta = 0$
13: **else**
14: **return** (x^+, z^+)
15: Set $(x, z) = (x^+, z^+)$

2.2 Convergence analysis of CASM

First, we examine the question, whether CASM yields a monotone decreasing sequence of function values. Using standard assumptions, one can show that the solution of the saddle point system (7.5) yields a descent direction $(\Delta x, \Delta z)$ as defined above in Equation (7.6). Since the proof is rather technical without offering any additional benefit, we refer here to the corresponding Lemma 4.10 in the PhD thesis of T. Kreimeier [13]. Next, we reformulate the optimization problem (7.2) in that we minimize for a given point $(\bar{x}, \bar{z})$ just along the direction $(\Delta x, \Delta z)$. Furthermore, we consider only equality constraints and the active inequality constraints described by the projection $P_{\mathcal{I}}$, where $\mathcal{I} = \mathcal{I}(x)$ collects the indices of the active inequality constraints at x. Then we obtain for

$$f_\Delta(\Delta x, \Delta z) := (a + Q\bar{x})^\top \Delta x + b^\top |\Sigma| \Delta z + \frac{1}{2} \Delta x^\top Q \Delta x$$

the optimization problem:

$$\begin{aligned} \min_{(\Delta x, \Delta z) \in \mathbb{R}^{n+s}} \quad & f_\Delta(\Delta x, \Delta z) \quad \text{such that} \\ & 0 = A\Delta x + B|\Sigma|\Delta z + C\Sigma\Delta z, \\ & 0 = P_{\mathcal{I}}^\top \big(D\Delta x + E|\Sigma|\Delta z + F\Sigma\Delta z\big), \\ & 0 = |\Sigma|\Delta z - \tilde{Z}\Delta x. \end{aligned} \tag{7.11}$$

This optimization task comprises exactly the same constraints as the saddle point system (7.5). Hence, it will be used to show descent. Under additional regularity assumptions, the descent direction is even unique. For this purpose, we employ the following constraint qualification (cf. [11, 13]).

Definition 7.2 (LIKQ). Let an optimization problem of the form (CALOP) and a signature vector $\sigma \in \{-1, 0, 1\}^s$ be given. The *Linear Independence Kink Qualification (LIKQ)* holds at a feasible point x_σ if the active Jacobian

$$\mathcal{J}_\sigma = \begin{bmatrix} A + B|\Sigma|\tilde{Z} + C\Sigma\tilde{Z} \\ P_{\mathcal{I}}(D + E|\Sigma|\tilde{Z} + F\Sigma\tilde{Z}) \\ P_\alpha \tilde{Z} \end{bmatrix} \in \mathbb{R}^{(m+|\mathcal{I}|+|\alpha|)\times n},$$

at x_σ has full row rank $m + |\mathcal{I}| + |\alpha|$. Here, α denotes the *active switching set* $\alpha(x) = \{i \in \{1, \ldots, \tilde{s}\} \mid z_i(x) = 0\}$.

Hence, LIKQ is very similar to the standard regularity condition LICQ for the smooth case in that it requires that the gradients of the active constraints must be linearly independent. For LIKQ, this additionally applies to the constraints related to the active kinks together with the active constraints.

Lemma 7.3. *Let* $(\Delta x^*, \Delta z^*)$ *be a solution of Equation* (7.11) *with* $\Delta x^* \neq 0$. *Suppose that the zero vector is no solution of Equation (*7.11*). Then* $f_\Delta(\cdot, \cdot)$ *is strictly decreasing along* $(\Delta x^*, \Delta z^*)$. *If LIKQ holds, this direction is unique.*

Proof. The proof is rather technical; see [13, Lemma 4.11]. □

Theorem 7.4. *Suppose that an optimization problem of the form* (CALOP) *is given, LIKQ holds at every feasible point and let* $Q = Q^\top \in \mathbb{R}^{n\times n}$ *be a positive definite matrix. Then, for any feasible starting point* $x \in \mathbb{R}^n$*, Algorithm* 7.1 *terminates after finitely many iterations at a minimizer of the quadratically penalized version of the optimization problem* (CALOP)*.*

Proof. Algorithm 7.1 prioritizes the multiplier ω of the inequality constraints; see line 8 versus line 11. Therefore, as long as the signature vector σ does not change, the proposed approach resembles an active set method to solve QPs. Furthermore, we always ensure a decrease in the target (see Lemma 7.3) and a step size $\beta > 0$. Such an approach determines a minimizer of problems with the structure (7.2) in finitely many steps; see, for example, [18, Chap. 16].

If the current iterate is a feasible signature stationary point of (7.2) for the currently active set of constraints, Algorithm 7.1 may change also the signature vector σ (see line 12) resulting in a change to a different polyhedron. Since there are only finitely many polyhedra and the value of the function value is consistently reduced, Algorithm 7.1 can modify the signature vector only finitely many times leading to a finite convergence of the overall algorithm. □

The last theorem considers the penalized version of the original optimization task (CALOP). Hence, when Algorithm 7.1 stops at a local minimizer of the penalized version in line 14, one has to verify that the current iterate is a minimizer of the original problem (CALOP). If this is not the case, one has to reduce the quadratic penalty term and start Algorithm 7.1 again. If (CALOP) has a minimizer and the influence of the penalty term is driven to zero in finitely many steps, this yields convergence to a minimizer of (CALOP) as proven next. In our numerical tests preformed so far, such a reduction was not necessary.

Theorem 7.5. *Suppose that for an optimization problem of the form* (CALOP) *the target function* f *is bounded from below on the bounded, feasible set. Let LIKQ hold at every feasible point. Then, for* $Q \to 0$*, the solutions generated by Algorithm* 7.1 *converge to a solution of the original optimization problem.*

Proof. See the proof of [13, Theorem 4.13]. □

3 Numerical results

We implemented Algorithm 7.1 in Matlab and illustrate its performance by means of three constrained PL test problems.

Example 7.6 (Constrained HUL). In [12], Hiriart-Urruty and Lemaréchal considered the piecewise linear and convex function $\varphi : \mathbb{R}^n \to \mathbb{R}$,

$$\varphi(x) = \max\{\max\{-100, 2x_1 + 5|x_2|\}, 3x_1 + 2|x_2|\}.$$

To test our algorithm, we add the two constraints

$$H_1(x) = -0.25x_1 - x_2 - 10 \leq 0,$$
$$H_2(x) = 2 - 0.2|x_1 + 9| - |x_2 + 1| \leq 0,$$

and choose the feasible starting point $x^0 = (9, -2.5)$. After reformulation of the max functions by means of the absolute value, this optimization problem requires the six switching variables and the target function

$$\begin{aligned} z_1 &= x_2, & z_2 &= -100 - 2x_1 - 5|z_1|, \\ z_3 &= -50 - 2x_1 + 0.5|z_1| + 0.5|z_2|, & z_4 &= x_1 + 9, \\ z_5 &= x_2 + 1, & z_6 &= 2.25|z_1| + 0.25|z_2| + 0.5|z_3|, \\ y &= -25 + 2x_1 + z_6, \end{aligned}$$

for example, one has $n = 2$, $s = 6$, $m = 0$, and $p = 2$.

Using Algorithm 7.1, 15 iterations are needed. Figure 7.1 shows a plot of the resulting kinks originating from the objective function (blue lines) and from the constraints (cyan blue lines). The inequality constraints are marked by the red lines and, therefore, the red area represents the feasible set. Finally, the iterates generated by Algorithm 7.1 are denoted by the black dots. In the plot, only 8 of the 15 iterations are marked. This is due to the fact that some of the iterations duplicate the point x when σ and ω are restricted or relaxed, that is, kinks or constraints are activated or deactivated.

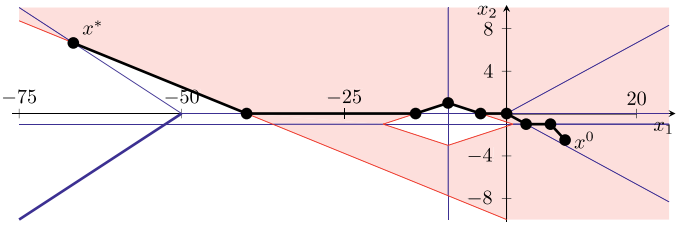


Figure 7.1: Iterates generated by Algorithm 7.1 for Example 7.6.

Example 7.7 (Constrained Rosenbrock–Nesterov II). According to [10], Nesterov suggested the Rosenbrock-like test function:

$$\varphi : \mathbb{R}^n \to \mathbb{R}, \quad \varphi(x) = \frac{1}{4}|x_1 - 1| + \sum_{i=1}^{n-1} |x_{i+1} - 2|x_i| + 1|$$

that is PL and nonconvex. It has the global minimizer $x^* = (1, 1, \ldots, 1) \in \mathbb{R}^n$ and $2^{n-1} - 1$ other Clarke stationary points none of which is a local minimizer [10]. From the literature [10], it is known that the starting point $x_1^0 = -1$, $x_i^0 = 1$, $2 \leq i \leq n$ is particularly challenging, since the nonoptimal, stationary points tend to lie on the way to the only minimizer. In this paper, we consider constrained problems, therefore we add the PL constraint

$$\sum_{i=1}^{n} |x_i - 1| \geq \frac{1}{2n}.$$

Hence, there is an n-dimensional rhombus around the global minimizer, which is cut out of the $\mathbb{R}^n$. The remaining $2^{n-1} - 1$ stationary points are still feasible and the minimizer is given by

$$x_i^* = 1 - \frac{2^{i-1}}{2^n - 1} \cdot \frac{1}{2n} \in (0, 1) \quad \text{for } 1 \leq i \leq n.$$

For $1 \leq n \leq 20$, the number of iterations required by CASM is shown in Table 7.1. The number of switches is given by $s = 3n - 1$. Hence, the number of visited polyhedra is much less than the total number of polyhedra given by 2^s. We also applied the MPBNGC solver [17] that implements a multiobjective proximal bundle method for nonconvex, nonsmooth, and generally constrained minimization. For $n = 1$, MPBNGC needed seven iterations, already for $n = 2$, the solver got stuck after seven iterations at a nonoptimal, stationary point. The same behavior was observed for larger values of n.

Table 7.1: Number of iterations of CASM needed for Example 7.7 and different values of n.

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
#	2	5	14	27	64	117	238	439	856	1685
n	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
#	3382	6807	13592	26285	42994	82995	131096	262173	605342	1119907

Example 7.8. Next, we consider a linear complementarity problem given by

$$Mx + q \geq 0 \quad \text{and} \quad x^\top (Mx + q) = 0 \tag{7.12}$$

for $0 \leq x \in \mathbb{R}^n$, $M \in \mathbb{R}^{n \times n}$, and $q \in \mathbb{R}^n$. In [2], the linear complementarity problem (LCP) is formulated as a system of PL equations

$$\min(x, Mx + q) = 0, \tag{7.13}$$

where the minimum operator acts componentwise. In the same paper, the authors present an algorithm that can be viewed as a semismooth Newton method and show nonconvergence, for example, for M set to

$$M_3 = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \quad \text{and} \quad M_4 = \begin{bmatrix} 1 & 0 & \frac{1}{2} & \frac{4}{3} \\ \frac{4}{3} & 1 & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{4}{3} & 1 & 0 \\ 0 & \frac{1}{2} & \frac{4}{3} & 1 \end{bmatrix}.$$

Furthermore, they pointed out that the LCP has a unique solution for any $q \in \mathbb{R}^n$ if and only if M is a **P**-matrix, that is, M has positive principal minors $\det M_{II}$ for all nonempty $I \subset \{1, \ldots, n\}$. To solve (7.13) with CASM, we reformulate the LCP as

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} |\min(x_i, (Mx + q)_i)|,$$

where $q = \mathbf{1}$ as the vector with 1 in every component of appropriate dimension. For the starting point set to the first unit vector in $\mathbb{R}^n$ as also proposed in [2], CASM needs five iterations in both cases, that is, for $M = M_3$ and $M = M_4$, respectively, to reach the solution, that is, the zero vector of the appropriate dimension. In [2, Proposition 3.7], it is shown that the algorithm proposed in that paper does not converge but generates a circle of three and four reoccurring iterates, respectively, for these settings.

4 Summary and outlook

We considered optimization problems with a piecewise linear target function and piecewise linear constraints as they arise, for example, in linear complementarity problems or certain bilevel optimization problems. For this purpose, we developed an extension of the already known active signature method for the constrained case by handling the inequality constraints in an active set sense. Numerical results for two test cases illustrate the performance of the resulting Constrained Active Signature Method (CASM).

The optimization problems presented in this paper have been of academic nature. In the future, we want to apply the algorithm to larger problems stemming from realistic applications. For example, solution approaches for the optimization of gas networks yield constrained piecewise linear subproblems; cf. [1, 15]. The first promising results were already obtained; see [14]. The optimization problems considered there are of much larger dimension having more than 500 optimization variables, 1,000 constraints, and almost 2,000 switches. A more involved implementation could solve the system of linear equations in parallel to speed up the step computation. However, it is important

to note that the proposed algorithm does not target the solution large-scale problems like the training of huge deep neural networks. This is mainly due to the matrix-based representation of the considered function class. An optimization approach that is more suitable for nonsmooth and large scale problems is considered in [3].

The constrained active signature method proposed in this paper could be used as a solver for the inner loop problem of a solver for constrained piecewise smooth problems starting, for example, from the SALMIN approach [5]. However, similar to the smooth situation, this can lead to infeasible iterates in the outer loop dealing with the nonlinear problem. Therefore, appropriate strategies to handle this infeasibility need to be developed.

Bibliography

[1] D. Aßmann, et al. Decomposable robust two-stage optimization: an application to gas network operations under uncertainty. *Networks*, 74(1):40–61, 2019.

[2] I. Ben Gharbia and J. C. Gilbert. Nonconvergence of the plain Newton-min algorithm for linear complementarity problems with a P-matrix. *Mathematical Programming*, 134:349–364, 2012.

[3] F. Bethke, A. Griewank, and A. Walther. A semismooth conjugate gradients method—theoretical analysis. *Optimization Methods & Software*, 39(4):911–935, 2024.

[4] F. E. Curtis, et al. A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. *Optimization Methods & Software*, 32(1):148–181, 2017.

[5] S. Fiege, et al. An algorithm for nonsmooth optimization by successive piecewise linearization. *Mathematical Programming Series A*, 177(1–2):343–370, 2019.

[6] A. Griewank. On stable piecewise linearization and generalized algorithmic differentiation. *Optimization Methods & Software*, 28(6):1139–1178, 2013.

[7] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd edition. SIAM, Philadelphia, 2008.

[8] A. Griewank and A. Walther. Finite convergence of an active signature method to local minima of piecewise linear functions. *Optimization Methods & Software*, 34(5):1035–1055, 2019.

[9] A. Griewank, et al. On Lipschitz optimization based on gray-box piecewise linearization. *Mathematical Programming Series A*, 158(1–2):383–415, 2016.

[10] M. Gürbüzbalaban and M. L. Overton. On Nesterov's nonsmooth Chebyshev–Rosenbrock functions. *Nonlinear Analysis*, 75(3):1282–1289, 2012.

[11] L. C. Hegerhorst-Schultchen and M. C. Steinbach. On first and second order optimality conditions for abs-Normal NLP. *Optimization*, 69(12):2629–2656, 2020.

[12] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms. I*. Springer, Berlin, 1993.

[13] T. Kreimeier. Solving constrained piecewise linear optimization problems by exploiting the abs-linear approach. PhD thesis, Humboldt-Universität zu Berlin, 2023.

[14] T. Kreimeier, et al. Towards the solution of robust gas network optimization problems using the constrained active signature method. In: C. Büsing and A. M. C. A. Koster, editors, *Proc. 10th Int. Network Opt. Conf.*, OpenProceedings.org, 2022.

[15] M. Kuchlbauer, et al. Adaptive bundle methods for nonlinear robust optimization. *INFORMS Journal on Computing*, 34(4):2106–2124, 2022.

[16] F. Liers and M. Merkert. Structural investigation of piecewise linearized network flow problems. *SIAM Journal on Optimization*, 26(4):2863–2886, 2016.

[17] M. M. Mäkelä, et al. Proximal bundle method for nonsmooth and nonconvex multiobjective optimization. In: *Mathematical Modeling and Optimization of Complex Structures*. Comput. Methods Appl. Sci., volume 40. Springer, Cham, 2016.
[18] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, 2006.
[19] S. Scholtes. *Introduction to Piecewise Differentiable Equations*. SpringerBriefs in Optimization. Springer New York, New York, NY, 2012.
[20] Q. Tao, et al. Piecewise linear neural networks and deep learning. *Nature Reviews Methods Primers*, 2(1):42, 2022.

Ken Trotti, Samuel A. Cruz Alegría, Alena Kopaničáková, and Rolf Krause

Parallel trust-region approaches in neural network training

Abstract: We propose to train neural networks (NNs) using a novel variant of the "Additively Preconditioned Trust-region Strategy" (APTS). The proposed method is based on a parallelizable additive domain decomposition approach applied to the neural network's parameters. Built upon the TR framework, the APTS method ensures global convergence toward a minimizer. Moreover, it eliminates the need for computationally expensive hyper-parameter tuning, as the TR algorithm automatically determines the step size in each iteration. We demonstrate the capabilities, strengths, and limitations of the proposed APTS training method by performing a series of numerical experiments. The presented numerical study includes a comparison with widely used training methods such as SGD, Adam, LBFGS, and the standard TR method.

Keywords: Machine learning, optimization, domain decomposition methods, preconditioning, trust-region

MSC 2020: 65K10, 68W10, 68T01, 65M55

1 Introduction

We consider the following *supervised learning problem*:

$$\min_{\theta\in\mathbb{R}^n} \ell(\theta;\mathcal{D}) := \min_{\theta\in\mathbb{R}^n} \frac{1}{p}\sum_{i=1}^{p} l(\mathcal{N}(x_i;\theta,\mathcal{D}),c_i), \tag{8.1}$$

where $\ell\colon \mathbb{R}^n \longrightarrow \mathbb{R}$ is the *training loss*, $\mathcal{N}$ denotes the neural network (NN), and $\theta \in \mathbb{R}^n$ are its parameters. The tuple $(x_i, c_i) \in \mathcal{D}$ is an input-target pair in the labeled data set $\mathcal{D}$

Acknowledgement: This work was supported by the Swiss Platform for Advanced Scientific Computing (PASC) project ExaTrain (funding periods 2017–2021 and 2021–2024) and by the Swiss National Science Foundation through the projects "ML^2 – Multilevel and Domain Decomposition Methods for Machine Learning" (197041) and "Multilevel training of DeepONets – multiscale and multiphysics applications" (206745).

Ken Trotti, Università della Svizzera italiana, 13 Via Giuseppe Buffi, 6900 Lugano, Switzerland, e-mail: ken.trotti@usi.ch

Samuel A. Cruz Alegría, Rolf Krause, Università della Svizzera italiana, 13 Via Giuseppe Buffi, 6900 Lugano, Switzerland; and UniDistance Suisse, 18 Schinerstrasse, 3900 Brig, Switzerland, e-mails: samuel.adolfo.cruz.alegria@usi.ch, samuel.cruz@fernuni.ch, rolf.krause@usi.ch, rolf.krause@fernuni.ch

Alena Kopaničáková, Università della Svizzera italiana, 13 Via Giuseppe Buffi, 6900 Lugano, Switzerland; and University of Toulouse, 41 All. Jules Guesde, 31000 Toulouse, France, e-mails: alena.kopanicakova@usi.ch, alena.kopanicakova@toulouse-inp.fr

https://doi.org/10.1515/9783111376776-008

with cardinality p. The *loss function* $l\colon \mathbb{R}^d_{\geq 0} \times \mathbb{R}^d_{\geq 0} \longrightarrow \mathbb{R}$ measures the difference between the predicted target $\hat{c}_i := \mathcal{N}(x_i; \theta, \mathcal{D})$ and the exact target $c_i \in \mathbb{R}^d$ [21]. Minimizing (8.1) is called *training* and successful training implies the model's capability to accurately predict unseen data [42], that is, the data not contained in the data set $\mathcal{D}$.

Several optimization techniques are applicable for training neural networks. The most prominent approaches include for example stochastic gradient descent (SGD) [52, 6, 14], ADAptive Moment estimation (Adam) [32, 46, 47], and the limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm [44], each with unique features and challenges. The SGD method is efficient, and generalizes well, but requires careful hyperparameter tuning and diminishing step sizes for global convergence [52, 6, 14]. Adam is efficient and adjusts its learning rate adaptively but lacks theoretical global convergence guarantees for nonconvex problems and may even face issues with convex problems [32, 46]. The LFBGS method approximates the Hessian using the secant pairs [44] and it is therefore suited for solving ill-conditioned problems. However, the LBFGS method is computationally more expensive than the first-order methods and it requires a use of novel techniques for evaluating the secant pairs in stochastic settings [4, 3]. Trust-Region (TR) methods [11], though less common, are gaining recognition in machine learning. Their popularity arises mostly from the fact that they adjust the step size dynamically at each iteration and that they offer global convergence for convex as well as nonconvex minimization problems [44, 11]. However, its extension to multi-level settings [25] and to stochastic settings requires nontrivial theoretical and algorithmic adjustments; see, for example, [45, 13, 20, 22, 23, 34].

With growing data and network complexity, parallelization approaches become essential [9, 5, 49, 27]. Current parallel approaches include data-parallel and model-parallel methods. Data parallelism divides data among processing units, each training a model copy, which is especially useful for large data sets [48]. This paradigm has been applied in SGD variants [52, 43, 51], ensemble learning [31, 15], and federated learning [40, 41, 38]. In contrast, model parallelism aims at training different parts of the NN across different processing units [2].

Recently, domain decomposition (DD) methods show promise in parallelizing NN's training [48, 2, 43]. DD methods were originally proposed for solving discretized partial differential equations. The idea behind these methods is to subdivide the main problem into smaller subproblems, a solution of which is used to enhance the solution process of the original problem. The subproblems are typically solved independently of each other, that is, in an additive manner, which in turn enables parallel computing [10, 50, 19, 17, 39]. Notable examples of additive nonlinear DD methods include ASPIN [8], GASPIN [26], RASPEN [16], or APTS [24]. While most of these methods were developed for solving nonlinear systems of equations, APTS and GASPIN were specifically designed for addressing nonconvex minimization problems, such as one given in Equation (8.1). Furthermore, both APTS and GASPIN employ a TR globalization strategy and, therefore, provide global convergence guarantees.

In the field of machine learning, DD approaches are increasingly utilized to enhance NN training. The decomposition and composition of deep Convolutional Neural Networks (CNNs), as discussed in [28, 29], offer insights into accelerating training through subnetwork transfer learning. In [30], the authors explore layer-parallel training of deep residual neural networks (ResNets), demonstrating their effectiveness in tasks like image classification. In [35], the authors introduce preconditioning strategies for physics-informed neural networks (PINNs), employing a nonlinear layerwise preconditioner for the LFBGS optimizer. Finally, in [33], the authors present a novel CNN-Deep Neural Network (DNN) architecture that is inspired by DD methods, specifically made for model-parallel training. This architecture is designed to handle various image recognition tasks effectively. It incorporates local CNN models or subnetworks that process either overlapping or nonoverlapping segments of the input data, such as subimages, demonstrating its adaptability and efficiency in different image recognition scenarios. These studies collectively underscore the significance of DD methods in training NNs across different domains.

In this work, we introduce a variant of the APTS method [24] to the field of machine learning. Unlike conventional NN-training methods, the unique aspect of our proposed algorithm is its ability to:

- Guarantee global convergence in deterministic settings,
- Eliminate the need for extensive hyperparameter tuning, and
- Enable parallelization.

Thus, APTS allows for a first step to perform training on large supercomputers with efficient and parameter-free approaches. We demonstrate the convergence properties of the proposed method numerically by utilizing benchmark problems from the field of classification. Our numerical study includes a comparative analysis with the SGD, Adam, and LFBGS methods.

The paper is organized as follows. In Section 2, we briefly recall the TR method and we provide a detailed explanation of the APTS optimizer. Then, in Section 3, we show numerical results comparing APTS with classic optimization strategies using two famous data sets, that is, MNIST and CIFAR-10. Finally, in Section 4, we draw conclusions.

2 Training methods

In this section, we provide a comprehensive explanation of the fundamental principles and techniques associated with the TR and APTS methods, focusing on their application to the training of NNs.

2.1 Trust-region method

TR methods [44, 11] are advanced globally convergent iterative algorithms utilized in nonlinear, convex, and nonconvex optimization to find local minima of objective func-

tions. These methods iteratively construct a simplified model of the objective function within a localized vicinity of the iterate estimate, referred to as the *trust region*. The size of the trust region is adaptively adjusted at each iteration to ensure an accurate approximation of the true function behavior.

More precisely, in the TR framework at iteration k, the optimization problem (8.1) is approximated by a first- or second-order model m_k. In order to obtain the search direction s^k, the model m_k is minimized as follows:

$$\min_{\|s\| \leq \Delta^k} m^k(s) = \min_{\|s\| \leq \Delta^k} \nabla \ell(\theta^k; \mathcal{D})^T s + \frac{1}{2} s^T H^k s, \tag{8.2}$$

where $H^k \in \mathbb{R}^{n \times n}$ is a symmetric matrix, typically an approximation to the Hessian of ℓ at θ^k, and $\Delta^k > 0$ is the TR radius. In case of a first-order model m_k, the quadratic term in (8.2) vanishes. The quality of the step s^k obtained by solving (8.2) is assessed using the ratio

$$\rho^k = \frac{\ell(\theta^k) - \ell(\theta^k + s^k)}{m^k(0) - m^k(s^k)}, \tag{8.3}$$

where $\ell(\theta^k) - \ell(\theta^k + s^k)$ is the actual reduction and $m^k(0) - m^k(s^k)$ is the predicted reduction. The TR radius and parameters are then updated based on the value of ρ^k. If $\rho_k \approx 1$, the model m_k provides a good approximation of the objective function, so the TR radius can be increased and the parameters can be updated. Otherwise, if ρ_k is smaller than a given threshold $\eta > 0$, the TR radius is shrunk and the parameters remain unchanged. We refer the reader to [11] for a more detailed explanation and the algorithm.

In this work, we adopt the Limited-memory Symmetric Rank 1 (LSR1) method [18, 44] for approximating the Hessian of ℓ. The LSR1 approach systematically generates an approximation of the Hessian matrix by incorporating Rank 1 updates derived from gradient differences at each iteration. As the number of updates accumulates and reaches the predetermined limit, l, the most recent update replaces the oldest one, ensuring that the total number of updates remains constant at l. Unlike the LBFGS method, which ensures a positive definite approximated Hessian, the LSR1 method allows for an indefinite Hessian approximation, potentially providing more accurate curvature information.

In order to solve efficiently the subproblem (8.2), we employ the Orthonormal Basis SR1 (OBS) technique proposed in [7]. The OBS method leverages an orthonormal basis spanned by the LSR1 updates to transform the dimensions of the TR subproblem from $n \times n$ to a more manageable $m \times m$, where typically $m \ll n$.

2.2 APTS with the decomposition of network's parameters

APTS [24] is a right-preconditioned TR strategy designed for solving non-convex optimization problems. In this work, we construct the nonlinear preconditioner by utilizing

the layerwise decomposition of NN's parameters [35]. Thus, in order to define the N subdomains, we introduce the projectors

$$R_i : \mathbb{R}^n \longrightarrow \mathbb{R}^{n_i},\ i = 1, \ldots, N, \quad \text{with } R_i R_j^T = 0,\ \forall i \neq j, \tag{8.4}$$

where the second property implies that the subsets of parameters generated by R_i are pairwise disjoint. Here, n denotes the dimensionality of the global parameter vector, and n_i represents the dimensionality of the local parameter vector, associated with the ith subdomain. We can now define the ith network copy as $\mathcal{N}_i(\cdot; \theta_i, \phi_i, \mathcal{D})$, where the subdomain parameters $\theta_i = R_i\theta$ are defined as the set of trainable parameters. Moreover, with a slight abuse of notation, we define $\phi_i = \theta \setminus \theta_i$ as the remaining nontrainable parameters. Clearly, the overall structure of $\mathcal{N}$ does not change—what changes is the trainable/nontrainable setting of its parameters.

The APTS algorithm can be executed using the entire data set or on minibatches, with the latter scenario being referred to as Stochastic APTS (SAPTS). Although the convergence proof of the deterministic version does not transfer directly to the stochastic version, the numerical results provided in Section 3 are promising. Algorithm 8.1 provides a summary of the proposed APTS training algorithm. In steps 2 and 4, we establish the convergence criteria and select the training approach, which can be either using the full data set (APTS) or employing a mini-batch strategy (SAPTS). In steps 5–6, we compute and store the loss and gradient with the current training data set. The vector g is normalized in step 6 so that it has a similar length to s (see step 11), such that their contribution is similar in step 16. In step 8, we synchronize all the local TR radii $\Delta_{L,i}$ with the global one and all the network copies, $\mathcal{N}_i$, with the current global parameters from $\mathcal{N}$. This synchronization preserves the state of trainable and nontrainable parameters that were previously established. Following this, in step 9, we initiate a series of TR iterations for each local network copy, which we call *local TR iterations*. At each iteration, the local TR algorithm will perform steps with length in the interval $[\Delta^L_{\min}, \Delta^L_{\max}]$ with increase/decrease factors α, β, and reduction/acceptance ratios η_1, η_2. This process iterates until the cumulative step size falls below the predefined global TR radius, Δ^G, signified by the condition $s^k_{i,j} = \|\theta^k_{i,0} - \theta^k_{i,j}\|_\infty \leq \Delta^G$, where i denotes the submodel and 0 and j are the local TR iterations. The iteration halts when either $s^k_{i,j}$ equals Δ^G or when the iteration count reaches ν.

Following the local TR iterations, step 10 involves aggregating all the updated weights from the local networks and summing them up. Here, we emphasize that the update s^k made in step 11 is bounded by the global TR radius Δ^G, that is,

$$\left\|\theta^k - \tilde{\theta}\right\|_\infty = \left\|\sum_{i=1}^N R_i^T \theta^k_{i,0} - \sum_{i=1}^N R_i^T \theta^k_{i,\text{new}}\right\|_\infty = \left\|\sum_{i=1}^N R_i^T (\theta^k_{i,0} - \theta^k_{i,\text{new}})\right\|_\infty \leq \Delta^G.$$

In addition, the operator R_i^T serves the specific purpose of padding the vector $\theta^k_{i,\text{new}}$ with zeros, transforming it to the same dimensionality as θ^k. This ensures a nonoverlapping aggregation during the summation process of $R_i^T \theta^k_{i,\text{new}}$ across $i = 1, \ldots, N$, a condition which follows from the property of R_i delineated in equation (8.4).

Algorithm 8.1: Scheme of APTS/SAPTS in weight.

Input: Objective function ℓ, the data set $\mathcal{D}$ or mini-batches $D_1, D_2, \ldots$, main NN $\mathcal{N}(\cdot; \theta^0, \mathcal{D})$, NN copies $\mathcal{N}_i(\cdot; \theta^0_{i,0}, \phi^0, \mathcal{D})$, maximum local TR iterations ν, the value of the Boolean `fdl` variable, global TR radius Δ^G and the min/max global TR radii $\Delta^G_{\min}/\Delta^G_{\max}$, the decrease/increase factors $0 < \alpha < 1 < \beta$ (local and global TR), the min/max radii $\Delta^L_{\min}/\Delta^L_{\max}$ of the local TR, the reduction/acceptance ratios $0 < \eta_1 < \eta_2 < 1$ (local and global TR).

1 epoch $\leftarrow 0$
2 **while** *not converged* **do**
3 $k \leftarrow 0$
4 **for** $D \in [\mathcal{D}]$ *(APTS) or* $D \in [D_1, D_2, \ldots]$ *(SAPTS)* **do**
5 $\ell^k \leftarrow \ell(\theta^k; D)$
6 $g^k \leftarrow -\frac{\nabla \ell(\theta^k; D)}{\|\nabla \ell(\theta^k; D)\|_\infty} \Delta^G$
7 **for** $i \in \{1, \ldots, N\}$ *in parallel* **do**
8 Set $(\theta^k_{i,0}, \phi^k) \leftarrow (R_i\theta^k, \theta^k \setminus R_i\theta^k)$ and $\Delta^L_i \leftarrow \Delta^G$
9 $\theta_{i,\text{new}} \leftarrow \min_{\theta_i} \ell(\theta_i; D)$ using ν TR steps on $\mathcal{N}_i(\cdot; \theta^k_{i,0}, \phi^k, D)$
10 Gather and sum up all local updates: $\tilde{\theta} \leftarrow \sum_{i=1}^N R_i^T \theta^k_{i,\text{new}}$
11 Evaluate the preconditioning step: $s^k \leftarrow \theta^k - \tilde{\theta}$
12 Update the weights of the main model $\mathcal{N}$ as $\theta^{k+\frac{1}{2}} \leftarrow \theta^k + s^k$
13 $w \leftarrow 0, \tilde{\Delta} \leftarrow \Delta^G$
14 **while** $\ell(\theta^{k+\frac{1}{2}}; D) > \ell^k$ *and* $w \neq 1$ *and* `fdl == True` **do**
15 $w \leftarrow \min\{w + 0.2, 1\}$ and $\tilde{\Delta} \leftarrow \max\{\Delta^G_{\min}, \alpha\tilde{\Delta}\}$
16 $\theta^{k+\frac{1}{2}} \leftarrow \theta^k + \frac{wg^k + (1-w)s^k}{\|wg^k + (1-w)s^k\|_\infty} \tilde{\Delta}$
17 $\theta^{k+1} \leftarrow \arg\min_\theta \ell(\theta; D)$ using one TR step on $\mathcal{N}(\cdot, \theta^{k+\frac{1}{2}}, D)$
18 $k \leftarrow k + 1$
19 epoch $\leftarrow$ epoch $+ 1$

In step 12, we evaluate the preconditioning step, that is, the search direction obtained during the preconditioning iteration. This direction is then used to update the weights of the global model in step 13. In steps 14–16, in case the *forced decreasing loss* (fdl) input variable is set to True, we make sure that the current loss $\ell(\theta^{k+\frac{1}{2}}; D)$ on the global model is smaller than the old loss ℓ^k. If this is not a case, then we modify s^k by averaging it with the gradient g^k. Moreover, we reduce the radius Δ to shrink s^k. This process is different from the one in [24] and it is inspired by the Dogleg method [44], which is repeated until a step is accepted or until the step is the gradient, that is, when $w = 1$.

In the final phase, a single TR iteration is executed on the global neural network $\mathcal{N}$ in step 17. The resulting step will have length in $[\Delta^G_{\min}, \Delta^G_{\max}]$ and TR will be set with increase/decrease factors α, β and reduction/acceptance ratios η_1, η_2.

3 Numerical results

In this section, we compare the performance of APTS against well-established training algorithms such as SGD, LBFGS, and Adam. All tests, accessible through the repository in [12], were conducted using Python 3.9, with the implementation leveraging PyTorch version 2.0.1+cu117. For SGD, LBFGS, and Adam, we utilized the standard optimizers provided by PyTorch.

We tuned the hyperparameters of SGD, LBFGS, and Adam to ensure a fair comparison, recording the average best results in 10 trials across various parameter configurations. Specifically:

- For SGD, we evaluated 63 combinations, varying the learning rate in $[10^{-4}, 1]$ and the momentum in $[0.1, 0.9]$.
- For LBFGS, we considered 45 combinations, adjusting both the learning rate within the interval $[10^{-4}, 1]$ and the history size (the amount of stored updates of the Hessian) within $[3, 50]$. Note that we set the `line_search_fn` parameter to `strong_wolfe`.
- For Adam, we examined 13 different learning rates within the interval $[10^{-5}, 10^{-1}]$.

For these three algorithms, all other input parameters were kept as their standard PyTorch default values. Moreover, the selected optimal parameters are shown in the legend of the figures. In contrast, APTS and TR were consistently run with the same set of input parameters across all tests. Detailed descriptions of these settings can be found in the next section.

Afterwards, we provide a detailed analysis of the obtained results, highlighting the characteristics of APTS in comparison to SGD, LBFGS, Adam, and TR. Note that in the absence of the mini-batch strategy, the SGD optimizer simplifies to the traditional Gradient Descent (GD) algorithm. We remark that one epoch of APTS consists in the preconditioning step and the global step in lines 4–17 of Algorithm 8.1.

Finally, some of the following plots will depict the average test accuracy and training loss (represented by a solid line) alongside the variance observed in the worst and best runs (indicated by dashed lines) across 10 independent trials, each initialized with different random NN parameters.

3.1 APTS implementation details and parameter configuration

The PyTorch programming environment imposes certain constraints on the level of control one has over parameter training. Parameters are configured in vectors, with entire vectors designated as either trainable or nontrainable. This means that, for instance, a fully connected layer, which consists of weights and biases, is stored as a structure composed by two distinct vectors of learnable parameters, and they can be configured to be trained independently. Therefore, the projectors R_i in Equation (8.4) are built in a way to maximize the efficiency in training, that is, there will not be two weights belonging to

the same PyTorch vectors with different trainable/non-trainable settings. Furthermore, the projectors are randomly generated, that is, if we have a total of M vectors of parameters and we want to split them across N models, each model will have approximately round($\frac{M}{N}$) randomly selected trainable vectors.

Key configurations in our implementation of the APTS algorithm are as follows:

- In step 8 of APTS, that is, Algorithm 8.1, the amount of subdomains (submodels $\mathcal{N}_i$) is specified in the legend of the figures in Section 3 and the trainable weight vectors are randomly selected.
- In step 10, concerning the *local* TR, we configured:
  - A maximum amount of iterations to $\nu = 5$.
  - A minimum radius of $\Delta^L_{\min} = 0$, an initial radius equal to the current global TR radius $\Delta^L = \Delta^G$, and a dynamically adjusted maximum radius $\Delta^L_{\max} = \Delta^G - \|\theta^k_{i,0} - \theta^k_{i,j}\|$ at the jth iteration to ensure that the cumulative step remains bounded by the global TR radius.
  - Whenever a quadratic model (8.2) is considered, history size for the SR1 Hessian is set to 5. Moreover, at the start of the preconditioning iteration, the local Hessian history is synchronized with the global one.
  - The reduction and acceptance ratios $0 < \eta_1 < \eta_2 < 1$ are the standard values for TR, that is, $\eta_1 = 0.25$ and $\eta_2 = 0.75$.
  - The decrease and increase factors α and β are set to 0.5 and 2, respectively.
- In step 17, which involves the *global* TR:
  - Minimum, maximum, and initial radii are set to $\Delta^G_{\min} = 10^{-4}$, $\Delta^G_{\max} = 10^{-1}$, and $\Delta^G = 10^{-2}$, respectively.
  - The values of α, β, η_1, η_2 match the ones of the local TR.
  - The reduction and acceptance ratios match those of the local TR.
  - Whenever a quadratic model (8.2) is considered, history size for the SR1 approximated Hessian is set to 5.

Note that both the global and local TR use the infinity norm to compute the length of the steps. This does not conflict with 2-norm used by the OBS method, as the 2-norm is bounded by the infinity-norm.

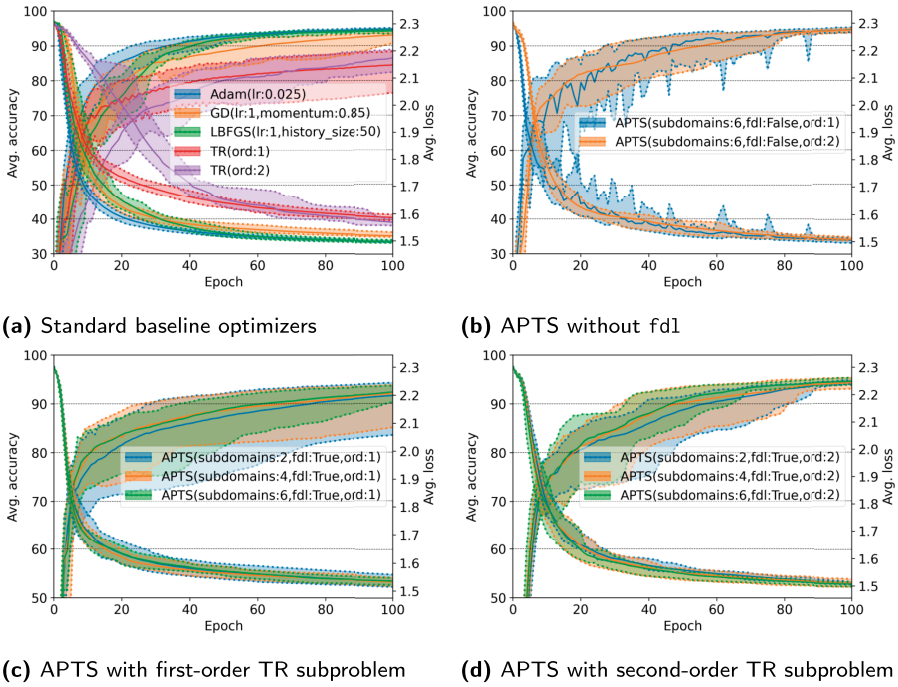
In the numerical results Section 3, the TR method as stand-alone solver, denoted by TR in the figures, has the same settings as the global TR method in APTS.

3.2 MNIST

We conducted some preliminary experiments using the MNIST data set [37]. This data set comprises 60,000 training images and 10,000 test images of handwritten digits, each of which is a 28×28 pixel grayscale image. For this experiment, we considered a Fully Connected Neural Network (FCNN) trained on the entire data set, that is, without the use of a mini-batch strategy.

The FCNN used in our experiments consists of three fully connected layers (6 PyTorch vectors), resulting in a total of 26,506 parameters. To investigate the performance of APTS, we configured it with 2, 4, and 6 subdomains and with an enabled/disabled forced-decreasing-loss parameter (`fdl`), while other parameters are reported in Section 3.1.

Observing Figure 8.1a, we note that Adam, LBFGS, and GD achieve comparable levels of accuracy. However, GD exhibits a marginally higher number of epochs to converge and a greater variance in its results. The TR variants, utilizing both first- and second-order information in Equation (8.2) (denoted by `ord = 1` and `ord = 2`, respectively), demonstrate a slower convergence compared to the benchmark algorithms. TR(`ord = 2`) overtakes its first-order counterpart TR(`ord = 1`) after 70 epochs. This can be attributed to the enhanced reliability of the approximated Hessian close to a minimizer of the training loss, which has a flatter landscape in comparison to the initial training phase. Moreover, the introduction of second-order information tends to stabilize the variation among TR iterations compared to the first-order approach.



(a) Standard baseline optimizers

(b) APTS without `fdl`

(c) APTS with first-order TR subproblem

(d) APTS with second-order TR subproblem

Figure 8.1: Comparison of APTS variants and baseline optimizers on FCNN trained on MNIST.

In Figure 8.1b, the APTS variants employing first- and second-order derivatives with `fdl` turned off are presented. The use of second-order derivatives seems to facilitate a smoother progression of the loss function over the training epochs. Conversely, reliance

on first-order derivatives results in more fluctuating loss values. Nonetheless, both variants converge to comparable levels of loss and accuracy, with a limited variance suggesting a consistent convergence pattern across different runs.

Figures 8.1c and 8.1d showcase the APTS variants with `fdl` enabled. Here, the application of second-order information leads to reduced variance and marginally superior accuracy relative to the first-order variant. Additionally, an increase in the number of subdomains correlates with improved performance. APTS with 6 subdomains not only slightly outperforms the 2 and 4 subdomain configurations in terms of accuracy, but also shows a reduced variance. This enhanced performance likely results from the added subdomains enabling more thorough exploration of the loss landscape through independent parameter adjustments, in contrast to the constrained exploration provided by a lower count of subdomains. This concept aligns with prior studies advocating for distinct weight and bias training strategies to improve training dynamics [1].

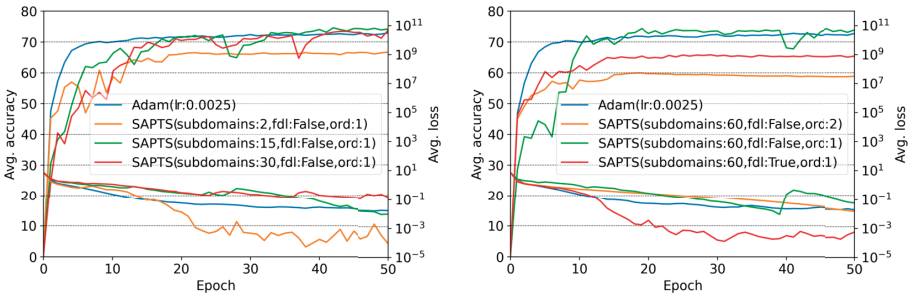
A comparison of Figures 8.1a, 8.1b, and 8.1c reveals that APTS with six subdomains, with `fdl` disabled and first-order information, attains similar performances to Adam, with Adam reaching the maximum accuracy slightly faster, and a marginally superior accuracy and lower loss than its counterpart with `fdl` enabled. This highlights the importance of unrestricted parameter exploration in improving training efficiency, similar to the exploration afforded by a higher subdomain count.

3.3 CIFAR-10

Following our MNIST investigation, we evaluated the solution strategies using the CIFAR-10 data set [36], containing 50,000 training and 10,000 test 32×32 color images across 10 classes. For these experiments, we employed the ResNet18 architecture from the PyTorch library with 11,689,512 parameters stored in 62 PyTorch vectors. Training was conducted using a mini-batch strategy, with 20 overlapping mini-batches, each containing 2,975 samples, which were created with a 1 % overlap between the mini-batches in order to reduce variance.

In Figure 8.2, we present a comparative analysis between stochastic Adam and various configurations of SAPTS. Specifically, Figure 8.2a features SAPTS with 2, 15, and 30 subdomains, a first-order TR approach for the subproblems, and with the feature `fdl` deactivated. In Figure 8.2b, the number of subdomains is set to 60 while we vary parameters such as `fdl` and `order`. We do not provide results for SGD here as Adam achieved a higher test accuracy, and our aim is to provide a comparison with APTS.

Our primary focus was on configurations with `fdl = False` and `ord = 1`. The rationale behind this is that activating `fdl` (as shown in Figure 8.2b) leads to a deterioration in the generalization abilities of SAPTS, albeit with a higher reduction in loss compared to other SAPTS variants. Regarding `ord = 2`, SAPTS exhibited poor generalization and did not significantly reduce the loss. However, it maintained a smoother loss curve compared to other tested SAPTS variants as in the previous nonstochastic cases.



(a) Adam and APTS with $2, 15, 30$ subdomains

(b) Adam and APTS variants with 60 subdomains

Figure 8.2: Comparison of APTS variants and baseline optimizers on CNN trained on CIFAR-10.

When analyzing SAPTS with `fdl = False` and `ord = 1` across different numbers of subdomains, we observe a correlation similar to that in non-stochastic tests: SAPTS' performance improves with an increased number of subdomains. Additionally, when comparing SAPTS with 60 subdomains against Adam, we find that SAPTS demonstrates marginally better generalization over a longer period and requires only a few more epochs to match the performance level of Adam.

4 Conclusion

In this work, we employed APTS method, which utilized the decomposition of the NN's parameters, to efficiently train neural networks. Our numerical study reveals that the proposed APTS method attains comparable or superior generalization capabilities to traditional optimizers like SGD, LBFGS, and Adam. The strength of the developed APTS method lies in absent hyper-parameter tuning and in its parallelization capabilities. Moreover, the APTS achieves faster convergence with a growing number of subdomains, showing encouraging algorithmic scalability.

Bibliography

[1] M. Ainsworth and Y. Shin. Plateau phenomenon in gradient descent training of ReLU networks: explanation, quantification and avoidance. *SIAM Journal on Scientific Computing*, 43(5):A3438–A3468, 2021.

[2] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. *ACM Computing Surveys*, 52(4):1–43, 2019.

[3] A. S. Berahas, M. Jahani, P. Richtárik, and M. Takáč. Quasi-Newton methods for machine learning: forget the past, just sample. *Optimization Methods & Software*, 37(5):1668–1704, 2022.

[4] A. S. Berahas, J. Nocedal, and M. Takáč. A multi-batch L-BFGS method for machine learning. In: *Advances in Neural Information Processing Systems*, volume 29, 2016.
[5] A. Blum and R. Rivest. Training a 3-node neural network is NP-complete. In: *Advances in Neural Information Processing Systems*, volume 1, 1988.
[6] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.
[7] J. Brust, J. B. Erway, and R. F. Marcia. On solving L-SR1 trust-region subproblems. *Computational Optimization and Applications*, 66:245–266, 2017.
[8] X. C. Cai and D. E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing*, 24(1):183–200, 2002.
[9] J. Catlett. Mega induction: a test flight. In: *Machine Learning Proceedings 1991*, pages 596–599. Morgan Kaufmann, 1991.
[10] T. F. Chan and J. Zou. Additive Schwarz domain decomposition methods for elliptic problems on unstructured meshes. *Numerical Algorithms*, 8(2):329–346, 1994.
[11] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000.
[12] S. A. Cruz Alegría and K. Trotti. ML_APTS, 2023. https://github.com/cruzas/ML_APTS.
[13] F. E. Curtis, K. Scheinberg, and R. Shi. A stochastic trust region algorithm based on careful step normalization. *INFORMS Journal on Optimization*, 1(3):200–220, 2019.
[14] X. Dai and Y. Zhu. Towards theoretical understanding of large batch training in stochastic gradient descent. arXiv:1812.00542, 2018.
[15] T. G. Dietterich. Ensemble methods in machine learning. In: *International Workshop on Multiple Classifier Systems*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.
[16] V. Dolean, M. J. Gander, W. Kheriji, F. Kwok, and R. Masson. Nonlinear preconditioning: how to use a nonlinear Schwarz method to precondition Newton's method. *SIAM Journal on Scientific Computing*, 38(6):A3357–A3380, 2016.
[17] J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund. *Domain Decomposition Methods in Science and Engineering XXI*. Springer, 2014.
[18] J. B. Erway and R. F. Marcia. On efficiently computing the eigenvalues of limited-memory quasi-Newton matrices. *SIAM Journal on Matrix Analysis and Applications*, 36(3):1338–1359, 2015.
[19] M. J. Gander. Optimized Schwarz methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, 2006.
[20] Y. Gao and M. K. Ng. A momentum accelerated adaptive cubic regularization method for nonconvex optimization. arXiv:2210.05987, 2022.
[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[22] S. Gratton, S. Jerad, and P. L. Toint. Complexity of a class of first-order objective-function-free optimization algorithms, 2023.
[23] S. Gratton, A. Kopaničáková, and P. L. Toint. Multilevel objective-function-free optimization with an application to neural networks training. *SIAM Journal on Optimization*, 33(4):2772–2800, 2023.
[24] C. Groß. *A unifying theory for nonlinear additively and multiplicatively preconditioned globalization strategies convergence results and examples from the field of nonlinear elastostatics and elastodynamics*. PhD thesis, Bonn International Graduate School, University of Bonn, 2009. URN: https://nbn-resolving.org/urn:nbn:de:hbz:5N-18682. Available online at: [Online-Publikationen an deutschen Hochschulen, Bonn, Univ., Diss., 2009].
[25] C. Gross and R. Krause. On the convergence of recursive trust-region methods for multiscale nonlinear optimization and applications to nonlinear mechanics. *SIAM Journal on Numerical Analysis*, 47(4):3044–3069, 2009.
[26] C. Groß and R. Krause. On the globalization of ASPIN employing trust-region control strategies—convergence analysis and numerical examples. arXiv:2104.05672, 2021.
[27] R. Grosse. CSC321 Lecture 6: Backpropagation. Lecture at the University of Toronto, 2018.

[28] L. Gu, W. Zhang, J. Liu, and X. C. Cai. Decomposition and composition of deep convolutional neural networks and training acceleration via sub-network transfer learning. *Electronic Transactions on Numerical Analysis*, 56:157–186, 2022.
[29] L. Gu, W. Zhang, J. Liu, and X. C. Cai. Decomposition and preconditioning of deep convolutional neural networks for training acceleration. In: *Domain Decomposition Methods in Science and Engineering XXVI*, pages 153–160. Springer, Cham, 2023.
[30] S. Gunther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020.
[31] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
[32] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. arXiv:1412.6980, 2014.
[33] A. Klawonn, M. Lanser, and J. Weber. A domain decomposition-based CNN-DNN architecture for model parallel training applied to image recognition problems. arXiv:2302.06564, 2023.
[34] A. Kopaničáková and R. Krause. Globally convergent multilevel training of deep residual networks. *SIAM Journal on Scientific Computing*, 0:S254–S280, 2022.
[35] A. Kopaničáková, H. Kothari, G. E. Karniadakis, and R. Krause. Enhancing training of physics-informed neural networks using domain decomposition–based preconditioning strategies. *SIAM Journal on Scientific Computing*, 46(5):S46–S67, 2024.
[36] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[38] G. Malinovsky, K. Yi, and P. Richtárik. Variance reduced Proxskip: algorithm, theory and application to federated learning. In: *Advances in Neural Information Processing Systems*, volume 35, pages 15176–15189, 2022.
[39] T. P. Mathew. *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*. Springer, 2008.
[40] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In: *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, April 2017.
[41] K. Mishchenko, G. Malinovsky, S. Stich, and P. Richtárik. Proxskip: yes! Local gradient steps provably lead to communication acceleration! finally! In: *International Conference on Machine Learning*, pages 15750–15769. PMLR, June 2022.
[42] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
[43] D. Nichols, S. Singh, S. H. Lin, and A. Bhatele. A survey and empirical evaluation of parallel deep learning frameworks. arXiv:2111.04949, 2021.
[44] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer New York, New York, NY, 1999.
[45] J. Rafati, O. DeGuchy, and R. F. Marcia. Trust-region minimization algorithm for training responses (TRMinATR): the rise of machine learning techniques. In: *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 2015–2019. IEEE, 2018.
[46] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. arXiv:1904.09237, 2019.
[47] S. Ruder. An overview of gradient descent optimization algorithms. arXiv:1609.04747, 2016.
[48] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. arXiv:1811.03600, 2018.
[49] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
[50] A. Toselli and O. Widlund. *Domain Decomposition Methods—Algorithms and Theory*, volume 34. Springer, 2004.
[51] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In: *Advances in Neural Information Processing Systems*, volume 28, 2015.
[52] M. Zinkevich, M. Weimer, L. Li, and A. Smola. Parallelized stochastic gradient descent. In: *Advances in Neural Information Processing Systems*, volume 23, 2010.

Pascal Van Hentenryck

Trustworthy optimization learning: a brief overview

Abstract: This article introduces the concept of trustworthy optimization learning, a methodology to design optimization proxies that learn the input/output mapping of parametric optimization problems. These optimization proxies are trustworthy by design: they compute feasible solutions to the underlying optimization problems, provide quality guarantees on the returned solutions, and scale to large instances. Optimization proxies are differentiable programs that combine traditional deep learning technology with repair or completion layers to produce feasible solutions. The article shows that optimization proxies can be trained end-to-end in a self-supervised way. It presents methodologies to provide performance guarantees and to scale optimization proxies to large-scale optimization problems. The potential of optimization proxies is highlighted through applications in power systems and, in particular, real-time risk assessment and security-constrained optimal power flow.

Keywords: Optimization, nonlinear optimization, deep learning, self-supervised learning, primal-dual learning

MSC 2020: 68T05, 90C05, 90C25, 90C30, 90C90

1 Introduction

Optimization technologies have been widely successful in industry: they dispatch power grids, route transportation and logistic systems, plan and operate supply chains, schedule manufacturing systems, orchestrate evacuations and relief operations, clear markets for organ exchanges to name only a few. Yet there remain applications where optimization technologies still face computational challenges, including when solutions must be produced in real time or when there are planners and operators in the loop who expect fast interactions with an underlying decision-support system.

Many engineering applications however operate on physical infrastructures that change relatively slowly. As a result, optimization technologies are often used to solve the same core optimization problem repeatedly on instances that are somewhat similar

Acknowledgement: This research is partly supported by NSF under Awards 2007095 and 2112533. Special thanks to the power systems team of the NSF Institute for Advances in Optimization (AI4OPT) and, in particular, Mathieu Tanneau, Seonho Park, Wenbo Chen, Michael Klamkin, Guancheng Qiu, and Andrew Rosemberg.

Pascal Van Hentenryck, NSF AI Institute for Advances in Optimization, Georgia Institute of Technology, Coda Building, 12th Floor, 756 West Peachtree Street, GA-30332 Atlanta, GA, USA, e-mail: pvh@gatech.edu

https://doi.org/10.1515/9783111376776-009

in nature. In other words, these applications require solving *parametric optimization problems* of the form

$$\Phi(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{p}} f_{\mathbf{x}}(\mathbf{p}) \quad \text{subject to} \quad \mathbf{h}_{\mathbf{x}}(\mathbf{p}) = 0 \;\&\; \mathbf{g}_{\mathbf{x}}(\mathbf{p}) \geq 0$$

where $\mathbf{x}$ represents instance parameters that determine the objective function $f_{\mathbf{x}}$ and the constraints $\mathbf{h}_{\mathbf{x}}$ and $\mathbf{g}_{\mathbf{x}}$. A solution to such a parametric optimization is a mapping from inputs $\mathbf{x}$ to outputs $\mathbf{p} = \Phi(\mathbf{x})$, which are optimal solutions to an optimization problem. In addition, it is reasonable to assume that the instance parameters $\mathbf{x}$ are characterized by a distribution learned from historical data.

These considerations led to the belief that machine learning could learn these parametric optimization problems and replace optimization altogether. A machine learning model, say a deep neural network, could be trained to approximate the mapping $\mathbf{x} \mapsto \Phi(\mathbf{x})$ using, as input, a data set $\mathcal{D} = \{\mathbf{x}_i\}_{i\in[n]}$ of instance parameters or a data set $\mathcal{D}^* = \{(\mathbf{x}_i, \Phi(\mathbf{x}_i))\}_{i\in[n]}$ that contains pairs of instance parameters and their associated optimal solutions.

The *supervised learning* indexsupervided learning version of this learning task consists in fitting the parameters θ of a parametric function $\tilde{\Phi}_\theta$ to minimize a loss function of the form

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{x}_i, \tilde{\Phi}_\theta(\mathbf{x}_i))$$

where

$$\mathcal{L}(\mathbf{x}, \mathbf{p}) = \left\|\mathbf{p} - \Phi(\mathbf{x})\right\|$$

and is known as *an empirical risk minimization under constraints*. Interestingly, since the parametric optimization problem is available in explicit form, it is possible to define a *self-supervised* version of the learning task that only uses the data set $\mathcal{D}$. Assume that $\tilde{\Phi}_\theta(\mathbf{x})$ returns a feasible solution $\tilde{\mathbf{p}}$, that is, $\mathbf{h}_{\mathbf{x}}(\tilde{\mathbf{p}}) = 0 \;\&\; \mathbf{g}_{\mathbf{x}}(\tilde{\mathbf{p}}) \geq 0$.[1] The self-supervised learning task amounts to minimizing

$$\frac{1}{n}\sum_{i=1}^{n} f_{\mathbf{x}}(\tilde{\Phi}_\theta(\mathbf{x}_i))$$

which is expressed in terms of the objective function $f_{\mathbf{x}}$ of the original parametric optimization problem (e. g., [7, 20]). This self-supervised approach has a significant benefit: it does not rely on the availability of optimal solutions which may be costly to obtain in practice. In addition, the objective function in supervised learning approaches may not be perfectly aligned with the objective of the original optimization problem, reducing the accuracy of the learning step.

1 The paper discusses repair layers to achieve this later.

Unfortunately, in engineering tasks, such an approximation is typically not satisfactory: by virtue of being a regression task, the machine learning predictions are unlikely to satisfy the problem constraints, which may have some significant consequences when optimization models are used to plan, operate, or control physical infrastructures. Early research on this topic combines machine learning and Lagrangian duality [8], using a loss function of the form

$$\mathcal{L}(\mathbf{x}, \mathbf{p}) = \|\mathbf{p} - \Phi(\mathbf{x})\| + \boldsymbol{\lambda}|\mathbf{h}_{\mathbf{x}}(\mathbf{p})| + \boldsymbol{\nu} \max(0, -\mathbf{g}_{\mathbf{x}}(\mathbf{p})).$$

The multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ can be trained by mimicking subgradient methods for computing Lagrangian duals, alternating between training the machine learning model and adjusting the multipliers by subgradient steps. However, although they reduce constraint violations, the resulting machine learning models are not guaranteed to find feasible solutions.

This paper reviews progress in designing and implementing *trustworthy optimization proxies*, that is, learning architectures that fuse machine learning and optimization in order to produce *feasible* solutions with *quality guarantees* in a *scalable* way. In addition, the paper illustrates that trustworthy optimization proxies can deliver outcomes that cannot be achieved by the two technologies independently, opening a new avenue to overcome the limitations of optimization technologies mentioned earlier. It focuses on continuous optimization and the methodological contributions are illustrated with applications in power systems. Applications of optimization proxies in supply chains, manufacturing, and transportation, where the optimization problems contain discrete variables, can be found in [12, 17, 30, 3].

2 Optimization proxies

A high-level outline of the architectures of optimization proxies is shown in Figure 9.1. Optimization proxies combine a predictive component (typically a deep neural network) that produces an approximation $\tilde{\mathbf{p}}$ with a *repair layer* that transforms $\tilde{\mathbf{p}}$ into a feasible solution $\bar{\mathbf{p}}$, that is, $\bar{\mathbf{p}}$ satisfies $\mathbf{h}_{\mathbf{x}}(\bar{\mathbf{p}}) = 0$ & $\mathbf{g}_{\mathbf{x}}(\bar{\mathbf{p}}) \geq 0$. In a first approximation, the repair layer can be thought of as a projection of $\tilde{\mathbf{p}}$ into the feasible space of the optimization problem. It could be computed for instance using the following optimization model [26]:

$$\bar{\mathbf{p}} = \underset{\mathbf{p}}{\operatorname{argmin}} \|\mathbf{p} - \tilde{\mathbf{p}}\| \quad \text{subject to} \quad \mathbf{h}_{\mathbf{x}}(\mathbf{p}) = 0 \;\&\; \mathbf{g}_{\mathbf{x}}(\mathbf{p}) \geq 0$$

In practice, however, there are limitations with this approach:

– The projection may be computationally expensive and the optimization proxy may not bring significant benefits compared to the original optimization.

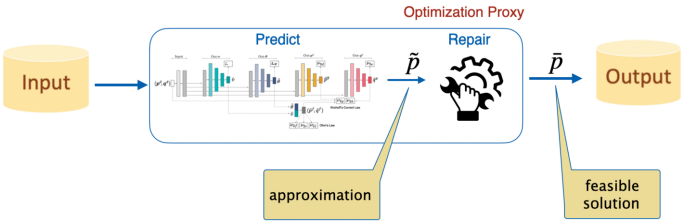


Figure 9.1: The architecture of optimization proxies.

– When training the model end-to-end in the style of [9, 7], the projection may again introduce significant computational costs. Other approaches (e. g., [13, 25, 11]) assume that a strictly feasible point is available, which is not always the case in practice.

As a result, whenever possible, it is preferable to design dedicated repair layers and learning architectures to ensure fast training and inference times.

Differentiable programming

The repair layers of optimization proxies may rely on *differentiable programming*, a generalization of deep neural networks that exploits Dynamic Computational Graphs (DCGs). These DCGs can be automatically and transparently differentiated, making it possible to train optimization proxies end-to-end efficiently. For illustration purposes, consider a simplified version of the Economic Dispatch (ED) run by the Independent Systems Operators (ISOs) in the United States:

$$\min_{\mathbf{p},\xi_{\text{th}}} \quad c(\mathbf{p}) + M_{\text{th}}\|\xi_{\text{th}}\|_1 \tag{9.1a}$$

$$\text{s. t.} \quad \mathbf{e}^\top \mathbf{p} = \mathbf{e}^\top \mathbf{d}, \tag{9.1b}$$

$$\mathbf{0} \le \mathbf{p} \le \bar{\mathbf{p}}, \tag{9.1c}$$

$$\underline{\mathbf{f}} - \xi_{\text{th}} \le \Phi(\mathbf{p} - \mathbf{d}) \le \bar{\mathbf{f}} + \xi_{\text{th}}, \tag{9.1d}$$

$$\xi_{\text{th}} \ge \mathbf{0}. \tag{9.1e}$$

The goal of the ED optimization is to find a generator dispatch that minimizes generation costs and satisfies the power balance, reserve, and generator constraints. (The reserve constraints are omitted here for simplicity). Objective (9.1a) minimizes the generation costs and the violations of the line thermal limits. Constant M_{th} is a large coefficient to ensure that these constraints are almost always satisfied. Constraints (9.1b) capture the global power balance and Constraints (9.1c) enforce minimum and maximum limits on each generator energy. Constraints (9.1d) express the thermal constraints on each branch using a power transfer distribution factor representation. As traditional in electricity markets in the US [14, 15], the thermal constraints are soft constraints, that is, they

can be violated but doing so incurs a (high) cost. By virtue of being a regression, the approximation $\tilde{\mathbf{p}}$ in the proxy does not satisfy the power balance constraint. The repair layer however uses ideas from control systems, and scales all generators proportionally, up or down, to obtain a feasible solution $\bar{\mathbf{p}}$ as follows:

$$\bar{\mathbf{p}} = \begin{cases} (1-\eta^{\uparrow})\tilde{\mathbf{p}} + \eta^{\uparrow}\overline{\mathbf{g}} & \text{if } \mathbf{1}^{\top}\tilde{\mathbf{p}} < \mathbf{1}^{\top}\mathbf{d} \\ (1-\eta^{\downarrow})\tilde{\mathbf{p}} + \eta^{\downarrow}\underline{\mathbf{g}} & \text{otherwise} \end{cases}$$

where $\eta^{\uparrow}$ and $\eta^{\downarrow}$ are defined as

$$\eta^{\uparrow} = \frac{\mathbf{1}^{\top}\mathbf{d} - \mathbf{1}^{\top}\tilde{\mathbf{p}}}{\mathbf{1}^{\top}\overline{\mathbf{g}} - \mathbf{1}^{\top}\tilde{\mathbf{p}}}, \quad \eta^{\downarrow} = \frac{\mathbf{1}^{\top}\tilde{\mathbf{p}} - \mathbf{1}^{\top}\mathbf{d}}{\mathbf{1}^{\top}\tilde{\mathbf{p}} - \mathbf{1}^{\top}\underline{\mathbf{g}}}$$

and $\underline{\mathbf{g}}$ and $\overline{\mathbf{g}}$ are the lower and upper bounds on the generators. This repair layer is differentiable almost everywhere and can thus be naturally integrated in the training of the machine learning model. The optimization proxy, which is depicted in Figure 9.2 for the ED problem with reserve constraints, guarantees feasibility during training and inference.

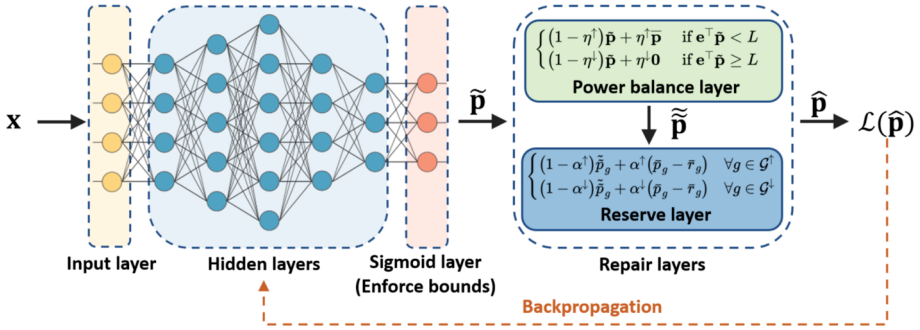


Figure 9.2: The optimization proxy for the economic dispatch problem.

More precisely, the optimization proxy is a differentiable program $\tilde{\Phi}_{\theta}$ that can be trained end-to-end using self-supervised learning. $\tilde{\Phi}_{\theta}$ can be trained using stochastic gradient descent: Its loss function

$$c(\hat{\mathbf{p}}) + M_{\text{th}} \left\| \hat{\xi}_{\text{th}} \right\|_1$$

is backpropagated through the repair layers to adjust the parameters θ of the neural network. Once trained, the resulting differentiable program $\tilde{\Phi}_{\theta^*}$ with tuned parameters θ^* can be used at inference time to produce a feasible solution $\tilde{\Phi}_{\theta^*}(\mathbf{x})$ in milliseconds for any instance $\mathbf{x}$ [5].

Primal-dual learning

Another approach to find feasible solutions is to develop optimization proxies that adapt traditional optimization algorithms to the learning context. Consider the constrained optimization problem:

$$\operatorname*{argmin}_{\mathbf{p}} f_{\mathbf{x}}(\mathbf{p}) \quad \text{subject to} \quad \mathbf{h}_{\mathbf{x}}(\mathbf{p}) = 0,$$

where $\mathbf{x}$ represents instance parameters that determine the objective function $f_{\mathbf{x}}$ and the equality constraints $\mathbf{h}_{\mathbf{x}}$. The *Augmented Lagrangian Method* (ALM) solves unconstrained optimization problems of the form

$$f_{\mathbf{x}}(\mathbf{p}) + \boldsymbol{\lambda}^T \mathbf{h}_{\mathbf{x}}(\mathbf{p}) + \frac{\rho}{2} \mathbf{1}^\top \mathbf{h}_{\mathbf{x}}(\mathbf{p})^2, \tag{9.2}$$

where ρ is a penalty coefficient and $\boldsymbol{\lambda}$ are the Lagrangian multiplier approximations. These multipliers are updated using the rule

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \rho \mathbf{h}_{\mathbf{x}}(\mathbf{p}). \tag{9.3}$$

Primal-Dual Learning (PDL) [20] jointly learns two neural networks: a primal neural network P_θ that learns the input/output mapping of the ALM unconstrained optimizations and a dual network D_ϕ that learns the dual updates. The overall architecture of PDL is shown in Figure 9.3. At each iteration, the *primal learning* step updates the parameters θ of the primal network while keeping the dual network D_ϕ fixed. After completion of the primal learning, PDL applies a *dual learning* step that updates the parameters ϕ of the dual network D_ϕ. The training of the primal network uses the loss function

$$\mathcal{L}_{\mathrm{p}}(\mathbf{p}|\boldsymbol{\lambda}, \rho) = f_{\mathbf{x}}(\mathbf{p}) + \boldsymbol{\lambda}^T \mathbf{h}_{\mathbf{x}}(\mathbf{p}) + \frac{\rho}{2} \mathbf{1}^\top \mathbf{h}_{\mathbf{x}}(\mathbf{p})^2,$$

which is the direct counterpart of the unconstrained optimization (9.2). The dual learning training uses the loss function

$$\mathcal{L}_{\mathrm{d}}(\boldsymbol{\lambda}|\mathbf{p}, \boldsymbol{\lambda}_k, \rho) = \left\| \boldsymbol{\lambda} - \boldsymbol{\lambda}_k + \rho \mathbf{h}_{\mathbf{x}}(\mathbf{p}) \right\|,$$

which is the direct counterpart of the update rule for the Lagrangian multipliers of the ALM (9.3). These two steps, that is, the training of the primal and dual networks, are iterated in sequence until convergence. Observe that PDL is self-supervised and does not require labeled data. PDL has been applied to preventive Security-Constrained Optimal Power Flow (SCOPF) problems with automatic primary response [19], an application which cannot be solved by state-of-the-art optimization and is discussed later in the paper.

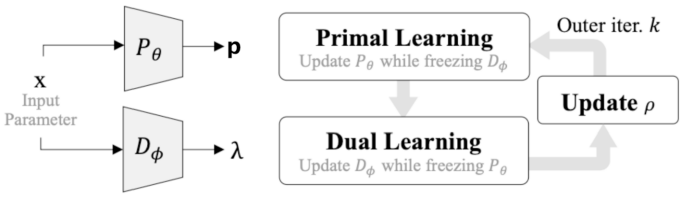


Figure 9.3: The architecture of primal-dual learning (from [20]).

3 Quality guarantees

An ideal outcome in optimization practice is to produce a pair of primal and dual solutions with a small duality gap. Optimization proxies also have the ability to find *dual feasible solutions* for many convex optimization problems arising in engineering, thus providing a quality guarantee to any primal solution delivered by another optimization proxy. In other words, it is possible to design two optimization proxies, that is, a pair of primal and dual differentiable programs, that deliver primal and dual feasible solutions respectively. To illustrate the *dual optimization proxies*, consider the following linear program and its dual:

$$\begin{array}{lll} \min & c^T\mathbf{x} & \\ s.t. & A\mathbf{x} \geq b & \\ & l \leq \mathbf{x} \leq u & \end{array} \qquad \begin{array}{lll} \max & b^T\mathbf{y} + l^T\lambda - u^T\gamma & \\ s.t. & \mathbf{y}A - \lambda + \gamma = c & \\ & \mathbf{y}, \boldsymbol{\lambda}, \boldsymbol{\gamma} \geq 0 & \end{array}$$

The formulation of the primal optimization reflects the fact that, in almost all engineering problems, *the decision variables have lower and upper bounds*. This is an important property because it makes it possible to systematically restore feasibility in the dual space. Indeed, the dual optimization proxy can predict $\mathbf{y}$ and then use $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$, that is, the dual variables associated with the bound constraints, to obtain a dual feasible solution. In other words, the dual optimization proxy predicts a subset of the variables and its completion layer obtains a dual feasible solution by carefully assigning the duals attached to the primal bound constraints. In fact, the optimal values for λ and γ admit a closed-form solution [10]:

$$\gamma = |c - \mathbf{y}A|^+ \quad \text{and} \quad \lambda = |c - \mathbf{y}A|^-,$$

where $|x|^+ = \max(0, x)$ and $|x|^- = \max(0, -x)$ denote the positive and negative parts of $x \in \mathbb{R}$. The methodology underlying dual optimization proxies is illustrated in Figure 9.4.

Dual optimization proxies have first been proposed for the Second-Order Cone Programming (SOCP) relaxation of the AC power flow equations in [21]. The completion layer is particularly interesting in that paper: it uses properties of the dual optimal solutions to determine how to complete the set of dual variables not predicted by the

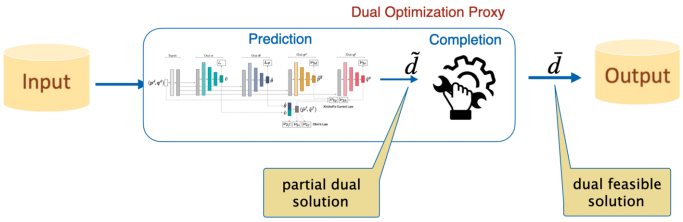


Figure 9.4: The structure of dual optimization proxies.

neural network. Again, the dual optimization proxy is a differentiable program that can be trained end-to-end to produce feasible solutions at training and inference times. Experimental results have shown that the resulting proxies can find near-optimal dual feasible solutions. See also [24] for a generalization of this result to dual conic proxies.

More generally, a methodology to find quality guarantees for an optimization problem Φ can be summarized as follows:

1. Find a convex approximation Ψ to problem Φ.
2. Obtain the dual Ψ^d of Ψ.
3. Derive a dual optimization proxy $\tilde{\Psi}^d$ for Ψ^d.

Now, given an instance $\mathbf{x}$, the pair $\langle \tilde{\Phi}(\mathbf{x}), \tilde{\Psi}^d(\mathbf{x}) \rangle$ provides a pair of primal and dual feasible solutions.

4 Scalability

Optimization problems in engineering often have outputs of high dimensions, which makes them quite different from traditional applications of machine learning. Interestingly, given that they operate on physical or highly engineered infrastructures, there is often significant structure in the output space. Figure 9.5 depicts the results of a Principal Component Analysis (PCA) on the *output space* of an Optimal Power Flow (OPF), where the x-axis represents the principal component ratio (i. e., the ratio of the number of the principal components in use to the dimension of the optimal solution) and the y-axis represents the explained variance ratio (i. e., the ratio of the cumulative sum of eigenvalues of the principal components in descending order to the sum of the all eigenvalues). The analysis shows that an almost lossless compression can be achieved with a few principal components. For instance, for the `13659_pegase` test case, 1 % of principal components preserves 99.92 % of information of the AC-OPF (OPF using an Alternative Current (AC) formulation) optimal solutions.

These observations can be used to design optimization proxies that learn in a small subspace of the principal components and then maps the solution into the original output space, as shown in Figure 9.6. This approach, called *Compact Optimization Learning*

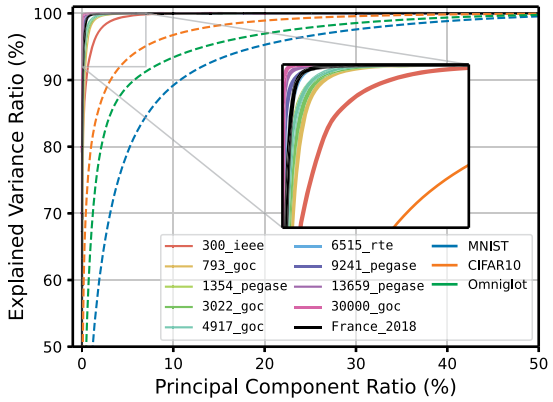


Figure 9.5: Principal component analysis of optimal power flows (from [18]).

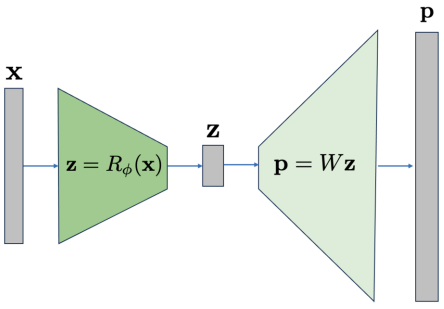


Figure 9.6: The structure of compact optimization learning. The optimization proxy R_ϕ predicts a small subset $\mathbf{z}$ of the principle components from input $\mathbf{x}$, which are then mapped by the linear transformation W into the actual output space $\mathbf{p}$.

[18], dramatically reduces the number of parameters to fit during training. It allows for the design of optimization proxies that learn optimal power flows for the largest electrical power grids.

5 Compositionality

Optimization models often appear as components in other optimization models. This is the case in decomposition techniques, bilevel optimization, and stochastic optimization for instance. One interesting question is whether optimization proxies can be as compositional as traditional optimization models. A neural network with ReLU activation functions can be encoded as a MIP, but this is not necessarily desirable given the resulting nonconvexities and computational challenges. An intriguing possibility is the use of *convex neural networks*, or more precisely, *Input-Convex Neural Networks* (*ICNNs*). A neural network with ReLU activation functions computes a convex function if all its weights are constrained to be nonnegative. An input-convex neural network generalizes the idea: it adds a first layer whose weights are unconstrained, and skip connections to

all other "convex" layers. More precisely, the kth layer ($k > 0$) of an input-convex neural network is of the form

$$\mathbf{x}^k = h^k(\mathbf{x}^{k-1}) = \text{ReLU}(W^k\mathbf{x}^{k-1} + H^k\mathbf{x}^0 + d^k), \tag{9.4}$$

where $\mathbf{x}^k$ and $\mathbf{x}^{k-1}$ denote the outputs of layer k and k–1, $\mathbf{x}^0$ denotes the input of the ICNN, d^k is the bias vector, and W^k, H^k are weight matrices. Skip-connections feed the ICNN input $\mathbf{x}^0$ to each layer. The coefficients of W^k are nonnegative, whereas H^k may take positive or negative values without affecting convexity [1]. Once trained, the input-convex neural network defines a convex function and its gradients with respect to its inputs, which meets the requirements of a number of applications. It has been shown that input-convex neural networks can approximate the objective value of AC-OPF, its second-order cone relaxation, and its DC approximation with high accuracy, opening many promising avenues for practical applications [23]. For instance, Figure 9.7 compares the accuracy of an ICNN and a regular ReLU-based neural networks for estimating the objective value of the SOCP relaxation of the optimal power flow on the RTE network.

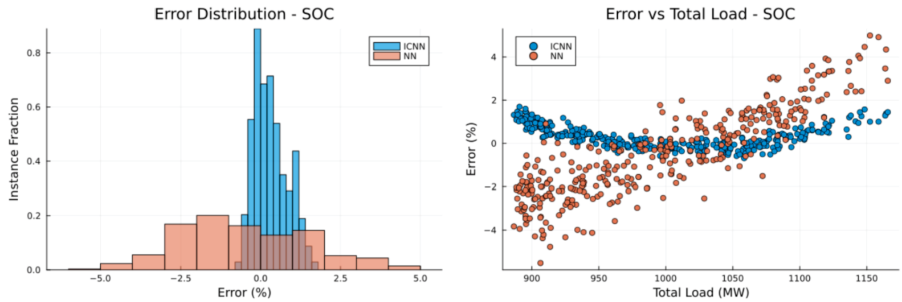


Figure 9.7: The performance of ICNNs on the SOCP relaxation of optimal power flow (from [23]).

When successful, a trained ICCN provides a high-fidelity approximation of the value function of an optimization problem and its gradients with respect to the inputs. This is exactly the functionalities needed by Benders decomposition (and its generalizations) when applied to deterministic and stochastic optimization problems. The ICCN can then replace a complex optimization subproblem to speed up these applications when only the optimal sub-objective value and its gradients are needed. ICNNs should also be useful for bilevel optimization problems, although case studies are necessary to demonstrate their benefits in these settings.

6 The impact of optimization proxies

Optimization proxies have the potential to transform various classes of applications for which they may bring orders of magnitude improvements in efficiency. One such appli-

cation is real-time risk-assessment of power systems. System operators are increasingly worried about the operational and financial risks in energy grids, given the significant growth in volatility coming from renewable sources of energy and the electrification of the transportation infrastructure. They are interested in dashboards able to quantify such risks in real time. Consider for instance the real-time risk-assessment framework described in Figure 9.8, which runs a collection of Monte Carlo scenarios. For each scenario, it is necessary to run 288 optimizations, one for every 5 minutes of the next 24 hours. These 288 optimizations for a specific scenario take about 15 minutes, and the risk assessment platform needs to evaluate thousands of them. By replacing these optimizations by a proxy, every scenario can now be evaluated in about 5 seconds [5]. It then becomes possible to quantify asset-level, system-level, and financial risks in an electrical power grid in real time. For instance, Figure 9.9 reports the probability of an adverse event for the next 24 hours using the risk assessment framework E2ELR [4] that uses the optimization proxies described earlier in this paper. It compares E2ELR with various learning architectures that have been proposed in the past (e. g., DeepOPF, DC3) and the ground truth (i. e., GRB) that denotes the risk assessment using a state-of-the-art optimization solver. The left figure depicts the probability of an adverse event on the balance constraint: both the ground truth and E2ELR show no such event. In contrast, the other methods systematically report potential adverse events, since they cannot guarantee feasibility of that constraint. The right figure shows the violations of the thermal limit of a congested line. Again, E2ELR isolates the potential violations with great accuracy. Other methods report many false positives across the day.

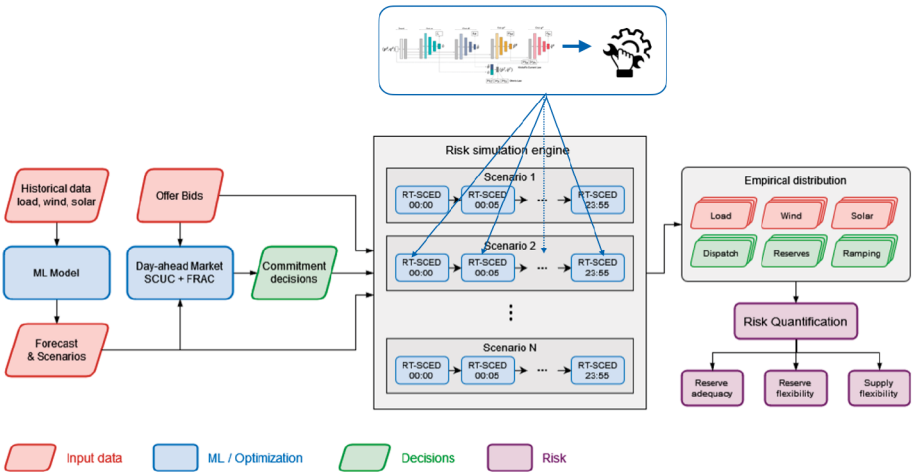


Figure 9.8: A risk assessment framework with optimization proxies.

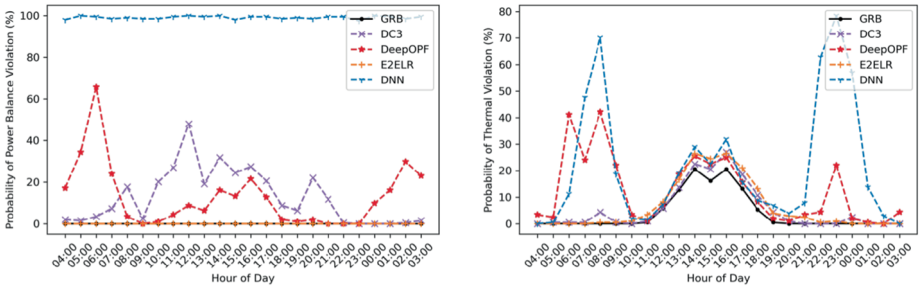


Figure 9.9: Probability of an adverse event with a proxy-based risk assessment (from [4]).

Optimization proxies also have the ability to deploy optimization models that would otherwise be too complex to meet real-time requirements. By shifting most of the computational burden offline during training, optimization proxies provide a way to produce high-quality solutions to optimization models that cannot be solved fast enough to meet real-time constraints. An interesting example is the security-constrained optimal power flow problem that captures the automatic primary response of generators in case of transmission line or generator contingencies. The solving of the SCOPF on a real network takes multiple hours, even using advanced optimization methods [27]. Interestingly, an optimization proxy, called PDL-SCOPF, that mimics the column and constraint generation algorithm of [27] can be trained in reasonable time and produce high-quality solutions in milliseconds [19]. The training is self-supervised and uses the primal-dual learning methodology presented earlier. The overall architecture is depicted in Figure 9.10: its repair layer, not only restores feasibility of the base (precontingency) dispatch, but also uses a binary search that computes the responses of the contingency dispatches. The resulting differentiable program, which uses a repair layer inspired by the original column and constraint generation algorithm [27], computes sophisticated DCGs at each iteration that are backpropagated to tune the primal fully connected network. On the

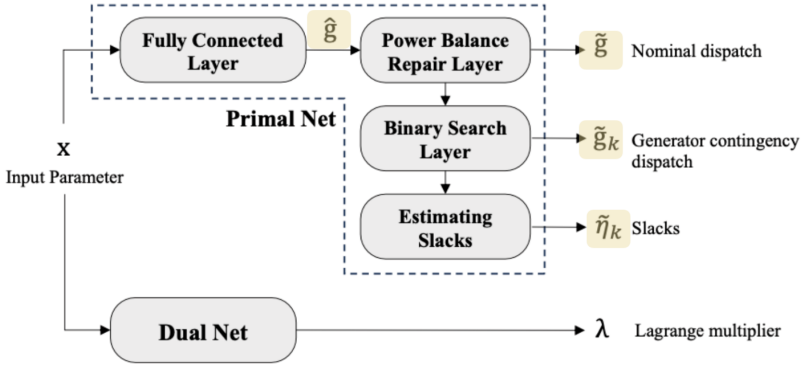


Figure 9.10: Primal-dual learning for security-constrained optimal power flow (from [19]).

RTE network (6,515 buses), PDL-SCOPF can be trained in less than 3 hours, and produces a feasible solution in 10 milliseconds with an optimality gap below 1 %.

7 Conclusion

This article introduced the concept of *trustworthy optimization learning*, a methodology to design optimization proxies that learn the input/output mapping of parametric optimization problems. These optimization proxies are trustworthy by design: they compute feasible solutions to the underlying optimization problems, provide quality guarantees on the returned solutions, and scale to large instances. Optimization proxies are differentiable programs that combine traditional deep learning technology with repair or completion layers.

The article highlighted that such optimization proxies can be trained end-to-end in a self-supervised manner and do not need a large collection of pre-solved instances. It presented a methodology to produce quality guarantees by exploiting the bounds on primal variables and convex optimization. The article also showcased the idea of primal-dual learning that mimics, in a learning framework, existing augmented Lagrangian methods to find solutions to complex nonlinear problems. The potential of input-convex neural networks to provide learning building blocks was also discussed.

This fusion of machine learning and optimization has the potential to open new applications for optimization technology. The article illustrated this potential on two case studies. It showed that optimization proxies can replace optimization models in risk assessment frameworks for energy systems in order to meet real-time constraints. It also indicated that optimization proxies deliver near-optimal solutions to security-constrained optimal power flow applications that are too challenging for practical uses with existing technology.

There are many avenues for further research, as this area is still in its infancy. Dual optimization proxies provide guarantees at inference time for a given instance, but it would be beneficial to obtain formal worst-case guarantees after the learning phase [28, 16]. This has been studied by various authors and additional work is needed for scaling existing approaches [2, 29, 22, 31, 6]. Despite some successes (e. g., [12, 17, 30, 3]), combinatorial optimization problems and mixed continuous-discrete optimization problems also raise significant challenges, as gradients are not available. Applications in stochastic optimization and bilevel optimization also seem fertile areas for trustworthy optimization learning.

Bibliography

[1] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In: *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[2] S. Chen, E. Wong, J. Z. Kolter, and M. Fazlyab. Deepsplit: scalable verification of deep neural networks via operator splitting. *IEEE Open Journal of Control Systems*, 1:126–140, 2022.

[3] W. Chen, R. Khir, and P. Van Hentenryck. Two-stage learning for the flexible job shop scheduling problem. *CoRR*, arXiv:2301.09703, 2023.

[4] W. Chen, M. Tanneau, and P. Van Hentenryck. Real-time risk analysis with optimization proxies. In: *Proceedings of the 23rd International Power Systems Computation Conference (PSCC'2024)*, Paris-Saclay, France, June 2024.

[5] W. Chen, M. Tanneau, and P. Van Hentenryck. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems*, 39(2):4723–4734, 2023.

[6] W. Chen, H. Zhao, M. Tanneau, and P. Van Hentenryck. Compact optimality verification for optimization proxies. In: *Proceedings of The 41st International Conference on Machine Learning (ICML 2024)*, Vienna, Austria, July 2024.

[7] P. L. Donti, D. Rolnick, and J. Z. Kolter. DC3: a learning method for optimization with hard constraints. In: *International Conference on Learning Representations*, 2021.

[8] F. Fioretto, T. W. K. Mak, and P. Van Hentenryck. Predicting AC optimal power flows: Combining deep learning and lagrangian dual methods. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 630–637. AAAI Press, February 2020.

[9] S. Gould, R. Hartley, and D. Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(08):3988–4004, Aug. 2022.

[10] M. Klamkin, M. Tanneau, and P. Van Hentenryck. Dual interior-point optimization learning. arXiv:2402.02596, 2024.

[11] A. V. Konstantinov and L. V. Utkin. A new computationally simple approach for implementing neural networks with output hard constraints. In: *Doklady Mathematics*, pages 1–9. Springer, 2024.

[12] J. Kotary, F. Fioretto, and P. Van Hentenryck. Fast approximations for job shop scheduling: a lagrangian dual deep learning method. In: *the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*, February 2022.

[13] M. Li, S. Kolouri, and J. Mohammadi. Learning to solve optimization problems with hard linear constraints. *IEEE Access*, 11:59995–60004, 2023.

[14] X. Ma, H. Song, M. Hong, J. Wan, Y. Chen, and E. Zak. The security-constrained commitment and dispatch for midwest ISO day-ahead co-optimized energy and ancillary service market. In: *2009 IEEE Power & Energy Society General Meeting*, pages 1–8, 2009.

[15] MISO. Real-time energy and operating reserve market software formulations and business logic, 2022. Business Practices Manual Energy and Operating Reserve Markets Attachment D.

[16] R. Nellikkath and S. Chatzivasileiadis. Physicsf-informed neural networks for minimising worst-case violations in DC optimal power flow. In: *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 419–424. IEEE, 2021.

[17] R. Ojha, W. Chen, H. Zhang, R. Khir, A. Erera, and P. Van Hentenryck. Optimization-based learning for dynamic load planning in trucking service networks. arXiv:2307.04050, 2024.

[18] S. Park, W. Chen, T. W. K. Mak, and P. Van Hentenryck. Compact optimization learning for ac optimal power flow. *IEEE Transactions on Power Systems*, 1–10, 2023.

[19] S. Park and P. Van Hentenryck. Self-supervised learning for large-scale preventive security constrained dc optimal power flow. arXiv:2311.18072 [cs.LG], November 2023.

[20] S. Park and P. Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.

[21] G. Qiu, M. Tanneau, and P. Van Hentenryck. Dual conic proxies for AC optimal power flow. In: *Proceedings of the 23rd International Power Systems Computation Conference (PSCC'2024)*, Paris-Saclay, France, June 2024.
[22] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. arXiv:1811.01057, 2018.
[23] A. Rosemberg, M. Tanneau, B. Fanzeres, J. Garcia, and P. Van Hentenryck. Learning optimal power flow value functions with input-convex neural networks. In: *Proceedings of the 23rd International Power Systems Computation Conference (PSCC'2024)*, Paris-Saclay, France, June 2024.
[24] M. Tanneau and P. Van Hentenryck. Dual lagrangian learning for conic optimization. arXiv:2402.03086, 2024.
[25] J. Tordesillas, J. P. How, and M. Hutter. Rayen: imposition of hard convex constraints on neural networks, 2023.
[26] A. Velloso and P. Van Hentenryck. Combining deep learning and optimization for preventive security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(4):3618–3628, 2021.
[27] A. Velloso, P. Van Hentenryck, and E. S. Johnson. An exact and scalable problem decomposition for security-constrained optimal power flow. *Electric Power Systems Research*, 195:106677, 2021.
[28] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis. Learning optimal power flow: worst-case guarantees for neural networks. arXiv:2006.11029, 2020.
[29] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. Fast and complete: enabling complete neural network verification with rapid and massively parallel incomplete verifiers. arXiv:2011.13824, 2021.
[30] E. Yuan, W. Chen, and P. Van Hentenryck. Reinforcement learning from optimization proxy for ride-hailing vehicle relocation. *Journal of Artificial Intelligence Research*, 75:985–1002, 2022.
[31] H. Zhao, H. Hijazi, H. Jones, J. Moore, M. Tanneau, and P. Van Hentenryck. Bound tightening using rolling-horizon decomposition for neural network verification. In: *The 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2024)*, Uppsala, Sweden, May 2024.

Max Zimmer, Christoph Spiegel, and Sebastian Pokutta

Compression-aware training of neural networks using Frank–Wolfe

Abstract: Many existing Neural Network (NN) pruning approaches rely on either retraining or inducing a strong bias in order to converge to a sparse solution throughout training. A third paradigm, "compression-aware" training, aims to obtain state-of-the-art dense models that are robust to a wide range of compression ratios using a single dense training run while also avoiding retraining. We propose a framework centered around a versatile family of norm constraints and the *Stochastic Frank–Wolfe* (SFW) algorithm that encourage convergence to well-performing solutions while inducing robustness towards filter pruning and low-rank matrix decomposition. Our method outperforms existing compression-aware approaches and, in the case of low-rank matrix decomposition, it also requires significantly less computational resources than approaches based on nuclear-norm regularization. Our findings indicate that dynamically adjusting the learning rate of SFW, as suggested by Pokutta et al. [57], is crucial for convergence and robustness of SFW-trained models and we establish a theoretical foundation for that practice.

Keywords: Compression aware training, neural networks, stochastic Frank–Wolfe, pruning, constrained optimization

MSC 2020: 68T07

1 Introduction

The astonishing success of Neural Networks (NNs) relies heavily on over-parameterized architectures [79] containing up to billions of parameters. Consequently, modern networks require large amounts of storage and increasingly long, computationally intensive training and inference times, entailing tremendous financial and environmental costs [64]. To address this, a large body of work focuses on compressing networks, resulting in *sparse models* that require only a fraction of memory or floating-point operations while being as performant as their *dense* counterparts. Recent techniques include

Acknowledgement: This research was partially supported by the DFG Cluster of Excellence MATH+ (EXC-2046/1, project id 390685689) funded by the Deutsche Forschungsgemeinschaft (DFG).

Max Zimmer, Christoph Spiegel, Sebastian Pokutta, Department for AI in Society, Science, and Technology, Zuse Institute Berlin & Institute of Mathematics, TU Berlin, 7 Takustrasse, 14195 Berlin, Germany, e-mail: zimmer@zib.de

https://doi.org/10.1515/9783111376776-010

the *pruning* of individual parameters [26, 24, 49, 8] or group entities such as convolutional filters and entire neurons [46, 2, 51, 75], as well as classical matrix- or tensor-decompositions [80, 65, 72, 48] in order to reduce the number of parameters.

A higher degree of sparsification typically leads to a degradation in predictive power of the network. Addressing this, two main paradigms emerged: *pruning after training*, like Iterative Magnitude Pruning (IMP) [26], is characterized by a three-stage pipeline of regular (sparsity-agnostic) training followed by prune-retrain cycles that are either performed once (*One-Shot*) or iteratively. The need for retraining to recover pruning-induced losses is often considered to be an inherent disadvantage and computationally impractical [50, 20, 70, 49, 85]. Alternatively, *pruning during training* avoids retraining by inducing strong inductive biases to converge to a sparse model at the end of training [82, 13, 37, 50]. A drawback of such approaches is the necessity to train one model per sparsity level, while IMP just needs one pretrained model to generate the entire accuracy-vs.-sparsity frontier.

A third paradigm, being the focus of this work, emerges when no retraining is allowed and training multiple times to generate the accuracy-vs.-sparsity tradeoff frontier is prohibitive. Ideally, the optimization procedure is "compression-aware" [3, 56] or "pruning-aware" [53], allowing to train once and then being able to compress to various degrees while keeping most of the performance without retraining (termed *pruning stability*). Compression-aware training procedures are expected to yield state-of-the-art dense models, which are robust to pruning without its hyperparameters being selected for a particular level of compression. While many such methods employ SGD-variants to discriminate between seemingly "important" and "unimportant" parameters [20, 13, 19, 83], actively encouraging the former to grow and penalizing the latter, an interesting line of research considers using optimizers other than SGD. An optimization approach that is particularly suited is the first-order and projection-free *Stochastic Frank–Wolfe* (SFW) algorithm [23, 7, 57, 66, 53]. While being used throughout various domains of machine learning for its highly structured, sparsity-enhancing update directions [39, 78, 12, 9], the algorithm has only recently been considered for promoting sparsity in NN architectures.

Addressing the issue of compression-aware training, we propose leveraging the SFW algorithm in conjunction with a family of norm constraints that actively encourage robustness to convolutional filter pruning and low-rank matrix decomposition. Our approach, using the group-k-support norm and variants thereof [4, 58, 52], is able to train state-of-the-art image-classification and semantic-segmentation architectures on large data sets to high accuracy while biasing the network toward compression-robustness. Similarly, motivated by the work of Pokutta et al. [57] and concurrent to our work, Miao et al. [53] showed the effectiveness of k-sparse constraints, focusing solely on unstructured weight pruning. Our approach generalizes the unstructured pruning case and further mitigates existing convergence and hyperparameter stability issues. To the best of our knowledge, our work is the first to apply SFW for structured pruning tasks. In analyzing the techniques introduced by Pokutta et al. [57], we find that the *gradient*

rescaling of the learning rate is important for obtaining high performing and pruning stable models. We lay the theoretical foundation for this practice by proving the convergence of SFW with gradient rescaling in the nonconvex stochastic case, extending results of Reddi et al. [60].

Contributions

The contributions of our work can be summarized as follows:

1. We propose a constrained optimization framework centered around a versatile family of norm constraints, which together with the SFW algorithm, produces well-performing models that are robust toward filter pruning as well as low-rank matrix decomposition. We empirically show that the proposed method is able to perform on par with or better than existing approaches. In the decomposition case, our approach can require significantly less compute resources than nuclear-norm regularization based approaches.
2. As a special case, our derivation includes the unstructured pruning setting. We show that our approach enjoys favorable properties when compared to the existing k-sparse approach [57, 53].
3. We empirically show that the robustness of SFW-trained NNs can largely be attributed to the usage of the *gradient rescaling* of the learning rate. To justify the usage of gradient rescaling theoretically, we prove the convergence of SFW with batch gradient dependent step size in the non-convex setting.

Related work

The *Frank–Wolfe* (FW) or *conditional gradient* algorithm [23, 44] is widely used in machine learning for handling complex structural requirements efficiently [39, 78, 22, 34, 55]. Lacoste-Julien [38] extended the convergence theory of FW to the nonconvex setting, while Hazan and Luo [27] and Reddi et al. [60] provide convergence rates for the stochastic variant of the algorithm (SFW). Accelerated variants include variance reduction approaches [27, 76, 63], adaptive gradients [16], and momentum [54, 14]. For a comprehensive review, see Braun et al. [9].

SFW has also received increased interest for training NNs with studies exploring parameter constraints [59], SFW applied for shallow networks [71], NN-specific variants [7], adversarial training [66], and achieving benchmark performances in image classification [57]. While classical FW has been applied to sparsity problems in machine learning, few have explored its structure-enhancing benefits in deep learning. Grigas et al. [25] remove neurons from three-layer convolutional networks. Pokutta et al. [57] constrain the parameters to lie within a k-sparse polytope, resulting in a large fraction having small magnitude. Miao et al. [53] leverage this idea for unstructured pruning in the "once-for-all" [10] pruning-aware training setting. We refer to Hoefler et al. [30] for a detailed account of sparsification approaches.

2 Methodology: compression-aware training

For $x \in \mathbb{R}^n$, we denote the ith coordinate of x by $[x]_i$. The diagonal matrix with x on its diagonal is denoted by $\operatorname{diag}(x) \in \mathbb{R}^{n,n}$. For $p \in [1, \infty]$, the l_p-ball of radius τ is denoted by $B_p(\tau)$. $\|x\|_0$ denotes the number of nonzero components of $x \in \mathbb{R}^n$. For any compact convex set $\mathcal{C} \subseteq \mathbb{R}^n$, let us further denote the l_2-diameter of $\mathcal{C}$ by $\mathcal{D}(\mathcal{C}) = \max_{x,y \in \mathcal{C}} \|x - y\|_2$. As usual, we denote the gradient of a function $\mathcal{L}$ at θ by $\nabla\mathcal{L}(\theta)$ and the batch gradient estimator by $\widetilde{\nabla}\mathcal{L}(\theta)$. We abuse notation and apply univariate functions to vectors in an elementwise fashion, for example, $|x|$ denotes the vector $|x| \coloneqq (|x_1| \dots, |x_n|)$. If not indicated otherwise, we treat a tensor x of a network as a vector $x \in \mathbb{R}^n$.

Constrained optimization using stochastic Frank–Wolfe

We aim at optimizing the parameters θ of an NN while enforcing structure-inducing constraints by considering the constrained finite-sum optimization problem

$$\min_{\theta \in \mathcal{C}} \mathcal{L}(\theta) = \min_{\theta \in \mathcal{C}} \frac{1}{m} \sum_{i=1}^{m} \ell_i(\theta), \tag{10.1}$$

where the per-sample loss functions ℓ_i are differentiable in θ and $\mathcal{C}$ is a compact, convex set. When using SGD, imposing hard constraints requires a potentially costly projection back to $\mathcal{C}$ to ensure feasibility of the iterates [15]. An alternative is the *Stochastic Frank–Wolfe* (SFW) algorithm [23, 7, 57], being projection-free and perfectly suited for yielding solutions with structural properties. To ensure feasibility, SFW does not use the gradient direction for its updates but rather chooses a boundary point or vertex of $\mathcal{C}$ that is best aligned with the descent direction. In each iteration t, SFW calls a *linear minimization oracle* (LMO) on the stochastic batch gradient $\nabla_t = \widetilde{\nabla}\mathcal{L}(\theta_t)$ to solve

$$v_t = \arg\min_{v \in \mathcal{C}} \langle v, \nabla_t \rangle, \tag{10.2}$$

which is then used as to update the parameters using the convex combination

$$\theta_{t+1} \leftarrow (1 - \eta_t)\theta_t + \eta_t v_t, \tag{10.3}$$

where $\eta_t \in [0, 1]$ is a suitable learning rate. If the initial parameters θ_0 are ensured to lie in the convex set $\mathcal{C}$, then the convex update rule ensures feasibility of the parameters in each iteration. Solving Equation (10.2) is often much cheaper than performing a projection step [34, 15], in many cases even admitting a closed-form solution. If $\mathcal{C}$ is given by the convex hull of (possibly infinitely many) vertices, a so-called *atomic domain*, then the solution to Equation (10.2) is attained at one of these vertices.

Inducing structure through the feasible region

Apart from constraining the parameters to satisfy a certain norm constraint, for example, a bounded Euclidean norm as in the case of weight decay, the unique update rule

Equation (10.3) of the SFW algorithm can be used to induce structure through the feasible region. A structured feasible region with update directions v_t satisfying highly structural constraints, cannot only enhance generalization [59, 57] but also induce desirable network properties like sparsity. A recent example is the *k-sparse polytope* introduced by Pokutta et al. [57] as a generalization of the l_1-ball B_1. The k-sparse polytope $\mathcal{C} = P_k(\tau)$ is defined as the convex hull of vectors $v \in \{0, \pm\tau\}^n$ with at most k nonzero entries, which form the solution set to Equation (10.2). For small k, v_t exhibits a high degree of sparsity and by Equation (10.3) only k parameters are activated while all remaining parameters are discounted strongly, encouraging convergence toward sparse solutions [57, 53].

In the following, we propose leveraging a suitable family of norm constraints, which arise naturally from l_2-regularization with sparsification requirements. In general, our goal is to choose the constraints to result in sparse update directions when applying the update rule of Equation (10.3), discriminating between predefined groups of parameters resulting in a decay on seemingly "unimportant" groups of parameters while allowing others to grow. Similarly to Pokutta et al. [57] and Miao et al. [53], we control the degree of sparsification with a tunable hyperparameter k such that the update vectors v_t are k-sparse, that is, nonzero at most k entries. However, existing approach are limited to the sparsification of NNs on an individual-weight basis (i. e., unstructured pruning) and may lead to hyperparameter and convergence instability. We construct constraints for structured sparsification, while also including the unstructured case. In addition, we ensure that (similar to classical SGD), the individual parameters receive updates corresponding to the magnitude of the gradient, enabling better convergence independent of the choice of k.

2.1 Inducing group sparsity to NNs

Given a disjoint partition of the network's parameters into groups $G \in \mathcal{G}$, we define the *group-k-support norm* [58] ball of radius τ as

$$\mathcal{C}_k^{\mathcal{G}}(\tau) = \operatorname{conv}\{v \mid \|v\|_{0,\mathcal{G}} \leq k,\ \|v\|_2 \leq \tau\}, \tag{10.4}$$

where $\|v\|_{0,\mathcal{G}}$ is the smallest number of groups that are needed to cover the support of v, that is, its nonzero entries. In other words, the vertex set of $\mathcal{C}_k^{\mathcal{G}}(\tau)$ is given by all (vectorized) parameters for which the Euclidean norm is bounded by τ and where at most k groups contain nonzero entries. Here, the definition of a set of groups $\mathcal{G}$ is left abstract, as it could be any disjoint partition of the parameters, that is each parameter w must lie in exactly one group $w \in G \in \mathcal{G}$.

Choosing $\mathcal{G}$ as the set of all convolutional filters of the lth convolutional layer, we can now state the solution to Equation (10.2) as follows. For a filter $G \in \mathcal{G}$, let $\|x\|_G$ be the l_2-norm of $x \in \mathbb{R}^n$ when only considering elements of G. Given the batch gradient of the lth convolutional layer ∇_t^l, let $G_1, \ldots, G_k$ be the k groups or filters with the largest

gradient l_2-norm $\|\nabla_t^l\|_{G_i}$ and let $\mathcal{C} \coloneqq H = \bigcup_{i=1}^{k} G_i$. The solution to Equation (10.2) is then given by

$$[v_t]_i = \begin{cases} -\tau[\nabla_t^l]_i / \|\nabla_t^l\|_H & \text{if } i \in H, \\ 0 & \text{otherwise.} \end{cases}$$

A proof can be found in Rao et al. [58]. Originally, the norm was motivated by the group lasso [74], which is common in the statistics and classical machine learning literature. To the best of our knowledge, these constraints have not been previously applied to deep NNs. SFW applied to $\mathcal{C} \coloneqq \mathcal{C}_k^{\mathcal{G}}(\tau)$ updates the k filters whose gradient entries correspond to those of fastest loss decrease while accounting for the distribution of magnitude among them instead of using the same magnitude for all parameters. Due to the convex update rule, the weights of the remaining filters are decayed, eventually resulting in few of them not being close to zero and thus making the trained network robust toward filter pruning. Figure 10.4 in the Appendix A shows how different values of k as a fraction of the overall number of filters influence the relative distance to the pruned model, indicating that k allows controlling the degree of robustness toward sparsification.

Unstructured sparsity as a special case

Our approach is easily extendable to the pruning of other groups, such as neurons. A special case arises when each weight is a group on its own group, naturally extending the above rationale to unstructured pruning. One then recovers the *k-support norm* [4], being a suitable candidate for comparison to the k-sparse polytope approach leveraged by Miao et al. [53]. The k-sparse approach suffers from two drawbacks, both being mitigated by our proposed method.

First of all, all activated parameters will receive an update of same magnitude, namely τ. This hinders convergence, especially when k is larger. Consider, for example, the worst-case scenario in which k equals the number of parameters n, then every single parameter of the network will receive an update of magnitude τ, essentially losing the entire information of the gradient apart from the entrywise sign. On the contrary, the k-support norm with $k = n$ will lead to optimization over the l_2-ball, yielding the default and best converging case of NN training. Figure 10.5 in the Appendix A shows the facilitated convergence of our approach, which is nonetheless highly robust towards unstructured pruning. The k-sparse approach performs well in the medium to high sparsity regime, but quickly collapses for higher compression rates. A clear advantage of k-support norm ball constraints is that SFW is able to obtain this performance in the high compression regime while not suffering from underperformance before pruning.

Second, Pokutta et al. [57] and Miao et al. [53] specify the desired l_2-diameter $\mathcal{D}$ of $\mathcal{C} = P_k(\tau)$ to control the regularization strength and then in turn choose the radius τ such that $\mathcal{D}(P_k(\tau)) = \mathcal{D}$. Defined this way, τ depends on k as $\tau = \mathcal{D}/(2\sqrt{k})$. This is counterintuitive, since k controls both the amount of activated parameters as well as the

magnitude of the parameter updates, resulting in unnecessarily coupled parameters. As opposed to the k-sparse polytope, the diameter of the k-support norm ball does not depend on k, and hence decouples the parameters k and τ as desired. Figure 10.6 in the Appendix A shows the successful decoupling of the radius and k. The k-support norm ball is less sensitive to hyperparameter changes and obtains better results throughout a wide range of hyperparameter configurations than its k-sparse counterpart.

2.2 A different sparsity notion: pruning singular values

So far, we considered one particular notion of sparsity, namely that of the existence of zeros in a matrix or tensor. Instead of removing individual parameters or groups thereof, networks can also be compressed after training by decomposing parameter matrices into a product of smaller matrices, allowing one to replace a layer by two consecutive ones that require a drastically smaller amount of FLOPs at inference [18]. The key ingredient is the truncated singular value decomposition (SVD), where setting the smallest singular values to zero leads to an optimal low-rank approximation by virtue of the Eckart–Young–Mirsky theorem. More precisely, given a rank r parameter matrix $\mathcal{W} \in \mathbb{R}^{n,m}$ with singular values $\sigma = (\sigma_1, \dots, \sigma_r)$ and SVD $U \in \mathbb{R}^{n,r}$, $\Sigma \in \mathbb{R}^{r,r}$, $V \in \mathbb{R}^{m,r}$, the k-SVD of $\mathcal{W}$ approximates $\mathcal{W}$ as

$$\mathcal{W} = U\Sigma V^T = \sum_{i=1}^{r} u_i \sigma_i v_i^T \approx \sum_{i=1}^{k} u_i \sigma_i v_i^T = U_k \Sigma_k V_k^T,$$

where the magnitude of "pruned" singular values quantifies the error in approximation. A detailed account of this approach can be found in the Appendix A. A natural approach to ensure robustness to matrix decomposition is hence based on penalizing the nuclear norm $\|\mathcal{W}\|_* = \|\sigma(\mathcal{W})\|_1$ [65, 3], which requires the costly computation of the full SVD in each iteration.

When constraining the parameters to have bounded nuclear norm instead of penalizing it, the LMO solution to Equation (10.2) utilized by SFW can be computed efficiently by requiring only the first singular value-vector-pair [34]. Extending this notion to consider the k largest singular pairs, we propose utilizing the *spectral-k-support norm* [52], for which the ball of radius τ is defined as

$$\mathcal{C}_k^{\sigma}(\tau) = \operatorname{conv}\{\mathcal{W} \mid \operatorname{rank}(\mathcal{W}) \leq k, \|\sigma(\mathcal{W})\|_2 \leq \tau\}, \tag{10.5}$$

where $\|\sigma(\mathcal{W})\|_2$ is the 2-Schattennorm of $\mathcal{W}$, being the l_2-norm of the vector of singular values $\sigma(\mathcal{W})$ of matrix $\mathcal{W} \in \mathbb{R}^{n \times m}$ [34]. The following lemma allows to efficiently compute the LMO solution to batch gradient $\nabla_t \in \mathbb{R}^{n \times m}$.

Lemma 10.1. *Let $\mathcal{W}_t = -\tau \|\sigma(\Sigma_k)\|_2^{-1} U_k \Sigma_k V_k^T \in \mathcal{C}_k^{\sigma}(\tau)$, where $U_k \Sigma_k V_k^T$ is the truncated k-SVD of ∇_t such that only the k largest singular values are kept. Then $\mathcal{W}_t$ is a solution to Equation* (10.2)*.*

A proof can be found in the Appendix A. Note that similar to the group-k-support norm ball taking the magnitude of k largest gradient groups into account, the scaling by $\Sigma_k \|\sigma(\Sigma_k)\|_2^{-1}$ takes the magnitude of k largest singular values into account. In Section 3.2, we study the capabilities of SFW when constraining the spectral-k-support norm of convolutional tensors, which account for the majority of FLOPs at inference [26]. While there exist higher-order generalizations of the SVD to decompose tensors directly [cf. 42, 35], we follow the approach of interpreting the tensor $\mathcal{W} \in \mathbb{R}^{n \times c \times d \times d}$ with c in-channels, n convolutional filters, and spatial size d as an $(n \times cd^2)$-matrix [3, 32].

3 Experimental results

We trained convolutional architectures like *Residual Networks* [28] and *Wide Residual Networks* [77] on data sets including *ImageNet-1K* [62], *TinyImageNet* [41], *CIFAR-100*, *CIFAR-10* [36], and *PSPNet* [81] on *CityScapes* [17]. Details on training setups and grid searches are in the Appendix A, following the pruning guidelines of Blalock et al. [8]. We averaged results over two seeds, reporting min-max bands for plots and standard deviations for tables. We used 10 % of the training data as a validation set for hyperparameter selection. Our code is available at github.com/ZIB-IOL/compression-aware-SFW.

In the compression-aware setting, we are interested in finding single hyperparameter configurations that perform well under a wide variety of compression rates, that is, without tuning hyperparameters for each sparsity. When comparing the performance for multiple compression rates at once, we have to decide how to select the "best" hyperparameter configuration. To that end, we select the configuration for each method that results in the highest on-average validation accuracy among all sparsities at stake.

3.1 Compression awareness: structured filter pruning

We apply the PFEC method [46] for convolutional filter pruning, sorting filters by their l_1-norm and removing the smallest to achieve desired compression, maintaining uniform sparsity across layers. Unlike Li et al. [46], we adopt a One-Shot approach without retraining. Our approach, *SparseFW*, is compared to the natural baseline of momentum SGD training and several recent filter pruning methods: *SSL* [68] uses a group penalty, *GLT* [2] applies a group-lasso followed by soft-thresholding, *ABFP* [19] balances filter penalization and growth, and *SFP* [29] allows filter recovery by soft pruning after each epoch. SparseFW introduces group-k-support norm constraints, with hyperparameters for the feasible region's radius and k, specifying the fraction of filters to activate in each convolutional layer per iteration. Nonconvolutional layers are optimized via SGD using default hyperparameters.

Figure 10.1 shows the accuracy-vs.-sparsity tradeoff for ResNet-50 trained on ImageNet-1K. SparseFW is able to converge to solutions that are robust to a wide range

of filter pruning ratios, maintaining performance even at high sparsities, unlike most methods except ABFP. Detailed results for other datasets and architectures are in the Appendix A.

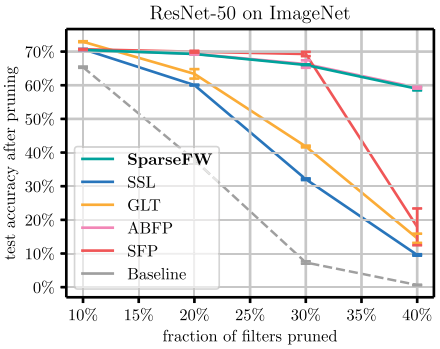


Figure 10.1: Accuracy-vs.-sparsity tradeoff curves for structured convolutional filter pruning on ImageNet. The plots show the parameter configuration with highest test accuracy after pruning when averaging over all sparsities at stake.

3.2 Compression awareness: low-rank decomposition

We compare SparseFW with spectral-k-support norm constraints to other approaches aiming for robustness to tensor decomposition. At the end of training, we prune the smallest singular values of each layer, replacing it with two layers matrix as described in the Appendix A. This corresponds to a uniform pruning approach, noting that there exist other ways to determine per-layer pruning ratios [48].

Apart from the standard momentum SGD with weight decay baseline, we examine low-rank methods like *NUC* [18] and *SVDEnergy* [3], which use nuclear norm regularization and *singular value thresholding* [11], respectively. Note that while the authors proposed to apply the thresholding after each epoch, we found it to be more effective when applying it after every iteration, however, at the cost of decreased efficiency. *TRP* [72] sets parameters to a low-rank representation periodically, the singular value pruning threshold being a hyperparameter. *Force Regularization (FR)* [69] forces convolutional filters to lie in a lower-dimensional subspace without having to compute the nuclear norm.

Figure 10.2 shows post-pruning test performance (accuracy or IoU) for WideResNet on CIFAR-100, PSPNet on CityScapes, and ResNet-50 on TinyImageNet across sparsity levels, indicating the fraction of pruned singular values. SparseFW consistently outperforms others, with minimal accuracy loss at high sparsity and negligible degradation at medium sparsity. For semantic segmentation, SparseFW reduces the IoU impact, however, also resulting in larger decreases, which we attribute to the usage of a pretrained backbone. Again, the performances correspond to the on-average best hyperparameter configuration.

Low-rank methods typically opt for either high-accuracy but costly singular value decomposition or more efficient, albeit less precise, approximations [73]. SparseFW

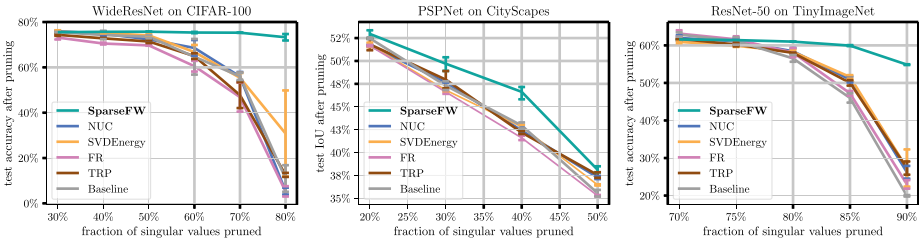


Figure 10.2: Test performance-vs.-sparsity tradeoff curves for low-rank tensor decomposition on CIFAR100 (left), CityScapes (middle), and TinyImageNet (right).

addresses this tradeoff by focusing on the most significant singular vector-value pairs through k-SVD, optimizing for both precision and efficiency. The full SVD of an $n \times m$ matrix demands $\mathcal{O}(nm \min(n,m))$ operations, but computing k-SVD is more efficient for small $k \leq \min(n,m)$ [1]. Table 10.1 demonstrates SparseFW's superior images-per-second throughput compared to methods like NUC, SVDEnergy, and TRP, which all require full SVD in each iteration, using consistent hardware (24-core Xeon Gold with Nvidia Tesla V100 GPU). FR, avoiding SVD, nearly matches regular training's efficiency but generally underperforms. SparseFW's throughput advantage is dependent on k, with efficiency reported for the k given by the on-average best performing hyperparameter configuration. We emphasize that we used a naive implementation of the k-SVD power method [6], noting that there are more sophisticated and faster algorithms to compute the k-SVD [1].

Table 10.1: Training images-per-second throughput of the low-rank methods in comparison to the baseline of regular training.

	# training images per second		
Method	**CIFAR-10**	**CIFAR-100**	**TinyImageNet**
Baseline	5664	1197	756
SparseFW (ours)	1156	372	338
NUC	566	204	160
SVDEnergy	493	174	98
FR	4397	1140	742
TRP	565	199	159

3.3 The impact of the learning rate schedule

The learning rate $\eta_t \in [0,1]$ determines the length of the parameter update relative to the size of the feasible region. This coupling between regularization strength and step size makes the tuning of the learning rate cumbersome. To decouple the tuning of the learning rate from the size of the feasible region, Pokutta et al. [57] propose two different

learning rate rescaling mechanisms: *diameter rescaling* and *gradient rescaling*, the latter being used throughout our experiments in the preceding sections. While the former divides the learning rate by the l_2-diameter $\mathcal{D}(\mathcal{C})$ of $\mathcal{C}$, gradient rescaling rescales the update direction length to that of the batch gradient, that is, $\hat{\eta}_t \coloneqq \eta_t \|\nabla_t\|_2 / \|v_t - \theta_t\|_2$.

The impact of normalization schemes on convergence and pruning robustness remains largely unexplored, especially since SFW's learning rate directly influences the decay of nonactivated parameters. Figure 10.3 demonstrates that gradient rescaling surpasses diameter rescaling in maintaining test accuracy both before and after magnitude-pruning when applied with k-support norm constraints. We interpret this result in detail in the Appendix A. The following theorem lays the theoretical foundation by showing that incorporating the batch gradient norm into the learning rate leads to convergence of SFW at the specified rate, that is, the expected product of exact gradient norm and the *Frank–Wolfe Gap*

$$\mathcal{G}(\theta_t) = \max_{v \in \mathcal{C}} \langle v - \theta_t, -\nabla L(\theta_t) \rangle,$$

being the measure of convergence [60], decays at a rate of $\mathcal{O}(T^{-1/2})$. The precise statement as well as a proof can be found in Appendix A.5.

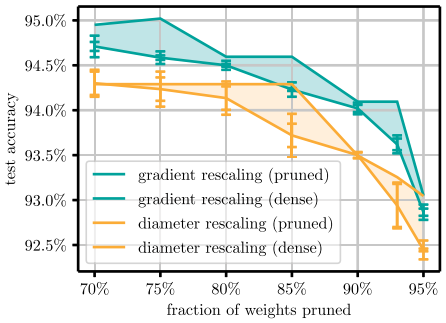


Figure 10.3: ResNet-18 on CIFAR-10: For each pruning amount, the best hyperparameter configuration w. r. t. the accuracy after pruning (*pruned*) is depicted. The corresponding value before pruning (*dense*) is depicted as a dashed line.

Theorem 10.2 (Convergence of gradient rescaling, informal). *Let L be M-smooth, ℓ be G-Lipschitz and $\eta_t = \|\nabla_t\|\eta$ for appropriately chosen η and all $0 \leq t < T$. If θ_a is chosen uniformly at random from the SFW iterates $\{\theta_i\}_{i<T}$, then we have $\mathbb{E}\left[\mathcal{G}(\theta_a) \cdot \|\nabla L(\theta_a)\|\right] = \mathcal{O}(T^{-1/2})$, where $\mathbb{E}$ denotes the expectation w. r. t. all the randomness present.*

4 Discussion and outlook

We proposed to utilize a versatile family of norm constraints to, together with the SFW algorithm, train deep NNs to state-of-the-art dense performance as well as robustness to compression for a wide range of compression ratios. Our experimental results show that

SFW can leverage highly structured feasible regions to avoid performance degradation when performing convolutional filter pruning or low-rank tensor decomposition. For the latter, SFW can significantly outperform similar methods both in terms of accuracy and efficiency. As a special case, our proposed approach includes the unstructured pruning case and we showed how utilizing the proposed norm can mitigate the drawbacks of and improve upon the results of Miao et al. [53]. We hope that our findings regarding the importance of the learning rate rescaling as well as Theorem 10.2 stimulate further research in the direction of compression-aware training with SFW.

We emphasize that our results hold primarily in the setting that we described, namely that of compression-aware training, where the training is sparsity-agnostic and retraining is prohibitive. Our goal was to show the versatility of SFW, which provides a suitable algorithmic framework for enforcing structure throughout training. If the sparsity can be incorporated into training, significantly more complex approaches can be applied.

Appendix A

In the following, we detail the technical details, hyperparameter grids, additional experiments as well as proofs of the theoretical results. Since the initial version of this paper, there has been follow-up work to improve the performance of pruned models in the case where retraining is feasible. We show that weight-averaging multiple models retrained from the same pruned base model, albeit with varying hyperparameter configuration, yields sparse models with enhanced generalization capabilities [86]. Similarly, we address the issue of retraining in a resource-constrained setting by retraining only a small, but critical subset of the parameters without performance degradation compared to full retraining [84].

A.1 Technical details

We employ momentum SFW [57] and *local* constraints, that is, we use a separate feasible region per layer. Note that the expressivity of the network can be severely limited if τ is chosen too small. As suggested by Pokutta et al. [57], we do not tune the radius τ of the feasible region $\mathcal{C}(\tau)$ directly but rather specify a scalar factor $w > 0$ and set the l_2-diameter of the feasible region of each layer as

$$\mathcal{D} = 2w\mathbb{E}(\|\theta\|_2),$$

where the expected l_2-norm of the layer parameters θ are simply estimated by computing the mean among multiple default initializations. $\mathcal{D}$ is then in turn used to compute τ such that $\mathcal{D}(\mathcal{C}(\tau)) = \mathcal{D}$. This allows us to control the l_2-regularization strength of each layer. We do not prune biases and batch-normalization parameters, as they only account

for a small fraction of the total number of parameters yet are crucial for obtaining well-performing models [21]. Apart from the direct comparison between diameter and gradient rescaling, we use gradient rescaling throughout all experiments.

While there exist multiple successful strategies to retrain after pruning [61, 40, 85], the compression-aware training setting requires the methods to be compared directly after pruning without retraining. However, similar to Li et al. [45] and Peste et al. [56], we notice that the validation accuracy after pruning can be significantly increased by recomputing the batch-normalization [33] statistics, which have to be recalibrated since the preactivations of the hidden layers are distorted by pruning. To that end, we recompute the statistics on the train data set after pruning and note that in practice only a fraction of the training data is necessary to recalibrate the batch-normalization layers.

In the following, we state technical details of our approaches to structured and unstructured pruning, as well as low-rank matrix decomposition.

A.1.1 Structured filter and unstructured pruning

As outlined in the main section, we follow the approach of Li et al. [46] and remove the convolutional filters with smallest l_1-norm. Since each filter might correspond to a different number of parameters, depending on the convolutional layer it is located in, the l_1-norm of filters is incomparable among different layers. We hence follow the local approach and prune the same amount in each convolutional layer until the desired sparsity is met. Since this will lead to the same theoretical speedup, independent of the algorithm at stake, we omit these values.

For unstructured pruning, we employ the usual magnitude pruning, that is, we remove the parameters with smallest absolute value until we meet the desired level of compression. As we found it to work best among all algorithms, we prune *globally*, that is, we select the smallest weights among all network parameters eligible for pruning. We note however that there exists several different magnitude-based selection approaches [cf., e. g., 26, 24, 43].

A.1.2 Low-rank matrix decomposition

We describe the rationale behind the decomposition of matrices with (preferably) low rank. Low-rank matrix decomposition is centered around the idea of truncating the singular value decomposition (SVD). Let $\theta = U\Sigma V^T = \sum_{i=1}^{r} u_i \sigma_i v_i^T$ be the SVD of rank-r matrix $\theta \in \mathbb{R}^{n\times m}$, where u_i, v_i are the singular vectors to singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0$. The goal is now to find the best low-rank approximation $\hat{\theta} \in \mathbb{R}^{n\times m}$ to θ, namely to solve

$$\min_{\operatorname{rank}(\hat{\theta})\leq t} \left\|\theta - \hat{\theta}\right\|_F. \tag{10.6}$$

By the Eckart–Young–Mirsky theorem, a minimizer is given by $U_t\Sigma_t V_t^T \coloneqq \sum_{i=1}^{t} u_i \sigma_i v_i^T$, where the smallest $t - r$ singular values are truncated. If the singular values now decay

rapidly, that is, only the first t singular values contain most of the *energy* of θ, then this approximation can be applied as a post-processing step without much change in the output of a linear layer [18]. Consequently, a natural approach to ensure stability w. r. t. matrix decomposition is based on encouraging the parameter matrices to have low-rank throughout training, most prominently done by regularizing the nuclear norm of the weight matrix [65, 3]. If t is small, then the number of FLOPs can be drastically reduced by decomposing a layer into multiple layers. For linear layers this is straightforward: let (θ, β) be weights and biases of a linear layer with SVD of θ as above. For input x, the layer computes

$$\theta x + \beta = U\Sigma V^T x + \beta \approx U_t(\Sigma_t V_t^T x) + \beta, \tag{10.7}$$

being interpretable as the consecutive application of two linear layers: $(\Sigma_t V_t^T, 0)$ followed by (U_t, β), possibly reducing the number of parameters from nm to $t(n + m)$. For a four-dimensional convolutional tensor $\theta \in \mathbb{R}^{n\times c\times d\times d}$, where c is the number of in-channels, n the number of convolutional filters, and d is the spatial size, we cannot directly construct the SVD. However, we follow an approach similar to those of Alvarez and Salzmann [3] and Idelbayev and Carreira-Perpinán [32], interpreting θ as a $(n \times cd^2)$ matrix, whose truncated SVD decomposition allows us to replace the layer by two consecutive convolutional layers, the first one having t filters, c channels, and spatial size d, followed by n filters, t channels, and spatial size of one.

A.2 Experimental setup and extended results

Table 10.2 shows the exact training configurations we used throughout all experiments, where we always relied on a linearly decaying learning rate, as suggested by Li et al. [47]. In the following, we state the hyperparameter grids used as well as full tables and missing plots.

Table 10.2: Exact training configurations used throughout the experiments. For all data sets and architectures we use a linear decay of the learning rate starting from 0.1. The dense test accuracy refers to the optimal accuracy we achieve using momentum SGD with weight decay. For the semantic segmentation task we report the Intersection-over-Union (IoU) on the test set.

Data set	Network (number of weights)	Epochs	Batch size	Momentum	Dense test accuracy/IoU
CIFAR-10	ResNet-18 (11 Mio)	100	128	0.9	95.0 % ±0.04 %
CIFAR-100	WideResNet-28x10 (37 Mio)	100	128	0.9	76.7 % ±0.2 %
TinyImagenet	ResNet-50 (26 Mio)	100	128	0.9	64.9 % ±0.1 %
ImageNet	ResNet-50 (26 Mio)	90	1024	0.9	75.35 % ±0.1 %
CityScapes	PSPNet (68 Mio)	200	12	0.9	58.5 IoU ±0.5

A.2.1 Structured filter pruning

If not specified otherwise, we use weight decay values of 1e-4, 5e-4. Tables 10.3, 10.4, 10.5 and 10.6 compare the filter pruning approaches for CIFAR-10, CIFAR-100, TinyImagenet and ImageNet, respectively.

CIFAR-10 hyperparameter grids

- SparseFW: $k \in \{0.1, 0.2, 0.3\}$, l_2-multiplier $w \in \{10, 20, 30\}$.
- SSL: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- GLT: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3), lasso tradeoff (0, 0.5).
- ABFP: $k \in \{0.1, 0.2, 0.3, 0.4\}$, filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- SFP: $k \in \{0.5, 0.6, 0.7, 0.8\}$, sparsification start epoch $\{0, 10, 25\}$.

CIFAR-100 hyperparameter grids

- SparseFW: $k \in \{0.15, 0.2, 0.25, 0.3\}$, l_2-multiplier $w \in \{20, 30, 40\}$.
- SSL: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- GLT: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3), lasso tradeoff (0, 0.5).
- ABFP: $k \in \{0.1, 0.2, 0.3, 0.4\}$, filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- SFP: $k \in \{0.6, 0.7, 0.8, 0.9\}$, sparsification start epoch $\{0, 10, 25\}$.

TinyImagenet hyperparameter grids

- SparseFW: $k \in \{0.15, 0.2, 0.25, 0.3\}$, l_2-multiplier $w \in \{20, 30, 40\}$.
- SSL: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- GLT: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3), lasso tradeoff (0, 0.5).
- ABFP: $k \in \{0.1, 0.2, 0.3\}$, filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- SFP: $k \in \{0.6, 0.7, 0.75, 0.8, 0.85, 0.9\}$, sparsification start epoch $\{0, 20, 50\}$.

Imagenet hyperparameter grids

If not specified otherwise, we use weight decay values of 1e-4 for all algorithms.

- SparseFW: $k \in \{0.15, 0.2, 0.25, 0.3, 0.35\}$, l_2-multiplier $w \in \{20, 25, 30, 35\}$.
- SSL: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- GLT: Filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3), lasso tradeoff (0, 0.5).
- ABFP: $k \in \{0.1, 0.2, 0.3\}$, filter group penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
- SFP: $k \in \{0.6, 0.7, 0.75, 0.8, 0.85, 0.9\}$, sparsification start epoch $\{0, 20, 50\}$.

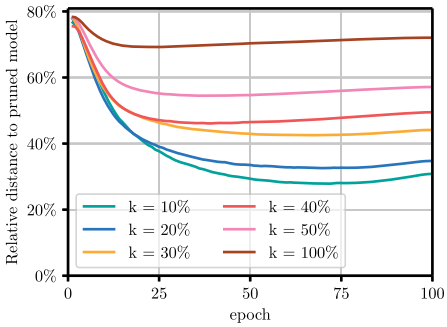


Figure 10.4: ResNet-18 on CIFAR-10: Relative distance to filter pruned model corresponding to 70% sparsity when training with the proposed approach and varying the fraction of activated filters.

Table 10.3: ResNet-18 on CIFAR-10: Comparison of filter pruning approaches. For each sparsity, we indicate the achieved test accuracy after pruning averaged over all random seeds including standard deviation.

	Sparsity			
Method	**60 %**	**70 %**	**80 %**	**90 %**
Baseline	60.32 ±0.18	32.83 ±0.52	14.49 ±4.68	10.92 ±0.20
SparseFW	90.72 ±0.09	90.39 ±0.16	87.51 ±0.44	35.15 ±1.30
SSL	91.26 ±0.13	87.13 ±1.72	63.66 ±0.36	16.25 ±1.24
GLT	68.54 ±6.12	45.42 ±6.99	21.03 ±3.69	9.36 ±0.22
ABFP	91.45 ±0.49	91.43 ±0.52	91.17 ±0.42	28.91 ±0.71
SFP	88.90 ±3.78	67.66 ±17.25	28.44 ±12.21	10.55 ±1.33

Table 10.4: WideResNet on CIFAR-100: Comparison of filter pruning approaches. For each sparsity, we indicate the achieved test accuracy after pruning averaged over all random seeds including standard deviation.

	Sparsity			
Method	**10 %**	**20 %**	**30 %**	**40 %**
Baseline	71.78 ±2.29	65.30 ±2.82	51.33 ±0.34	35.00 ±2.59
SparseFW	72.21 ±0.30	72.24 ±0.29	72.20 ±0.23	71.23 ±0.18
SSL	72.10 ±1.20	67.51 ±0.25	59.26 ±0.77	42.18 ±2.67
GLT	74.08 ±0.87	69.37 ±1.54	54.43 ±4.77	28.19 ±6.30
ABFP	71.49 ±0.12	71.50 ±0.17	71.53 ±0.11	71.51 ±0.13
SFP	73.45 ±0.35	73.47 ±0.34	73.05 ±0.04	63.82 ±2.14

A.2.2 Unstructured weight pruning

CIFAR-10 hyperparameter grids

For both the k-sparse polytope as well as k-support norm ball, we tune the fractional $k \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the multiplier $w \in \{10, 20, 30, 40, 50\}$.

Figure 10.6 compares the two feasible regions in an even larger hyperparameter search. The rows correspond to the k-sparse polytope (above) and k-support norm ball (below), respectively. The left column shows a heatmap of the test accuracy before prun-

Table 10.5: ResNet-50 on TinyImagenet: Comparison of filter pruning approaches. For each sparsity, we indicate the achieved test accuracy after pruning averaged over all random seeds including standard deviation.

	Sparsity					
Method	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**
Baseline	62.20 ±0.26	57.44 ±0.04	49.69 ±0.47	39.63 ±0.69	26.75 ±1.32	13.04 ±0.46
SparseFW	62.49 ±0.15	62.51 ±0.03	62.34 ±0.13	61.84 ±0.22	60.98 ±0.03	58.01 ±0.41
SSL	60.44 ±0.37	58.86 ±0.59	56.58 ±1.46	53.06 ±2.28	46.82 ±2.69	36.79 ±2.11
GLT	60.71 ±0.46	57.47 ±1.09	51.75 ±0.52	42.96 ±1.36	30.09 ±1.17	14.86 ±1.21
SFP	62.06 ±0.38	62.07 ±0.37	62.06 ±0.37	62.06 ±0.37	49.31 ±1.33	26.76 ±2.22
ABFP	60.28 ±0.63	60.31 ±0.69	60.28 ±0.58	60.20 ±0.59	60.20 ±0.49	59.99 ±0.55

Table 10.6: ResNet-50 on Imagenet: Comparison of filter pruning approaches. For each sparsity, we indicate the achieved test accuracy after pruning averaged over all random seeds including standard deviation.

	Sparsity			
Method	**10 %**	**20 %**	**30 %**	**40 %**
Baseline	65.37 ±0.25	37.78 ±1.46	7.33 ±0.48	0.65 ±0.04
SparseFW	70.50 ±0.19	69.28 ±0.00	66.03 ±0.26	58.95 ±0.67
SSL	70.73 ±0.08	60.05 ±0.19	32.08 ±0.35	9.58 ±0.30
GLT	72.93 ±0.20	63.39 ±1.98	41.80 ±0.36	14.53 ±1.94
SFP	70.69 ±0.07	69.97 ±0.38	69.28 ±0.89	17.92 ±7.73
ABFP	70.54 ±0.23	69.33 ±0.33	66.29 ±1.56	59.27 ±0.39

ing. While both approaches lead to well performing models for a wide range of hyperparameter configurations (indicated as the radius multiplier w on the x-axis and k on the y-axis), the k-support norm ball reaches higher results and converges properly for all configurations at stake. The k-sparse polytope approach fails to yield adequately trained dense models when the radius is relatively small but k becomes larger, which is counterintuitive, since larger k allows a larger fraction of the parameters to be activated. The right column shows the corresponding heatmap of the test accuracy right after pruning. Clearly, the proposed approach is robust to pruning for a wider hyperparameter range.

Miao et al. [53] showed that SFW (with k-sparse polytope constraints) outperforms SGD with weight decay, which in turn clearly, and unsurprisingly, outperforms the SFW-based approach when it is allowed to retrain. Our experiments indicate that while being less robust to pruning, SGD is able to reach on-par or better results after retraining, even when SFW is allowed to be retrained for the same amount of time. Leaving the domain of compression-aware training, this raises a more general question: in the case that retraining is not prohibited, is it beneficial to aim for robustness at pruning when trying to maximize the post-retraining accuracy?

Figure 10.7 illustrates an experiment where we investigate this exact question by performing One-Shot IMP [26] to a sparsity of 95 % and retraining for 10 epochs using

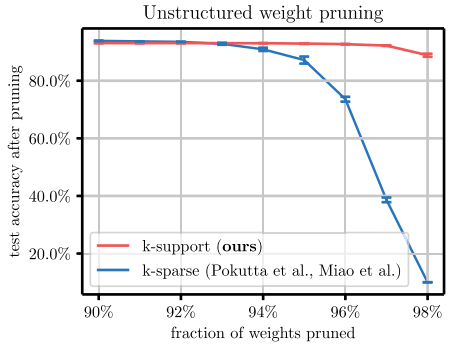


Figure 10.5: ResNet-18 on CIFAR-10: Accuracy-vs.-sparsity tradeoff curves for unstructured weight pruning comparing our approach to the existing k-sparse approach.

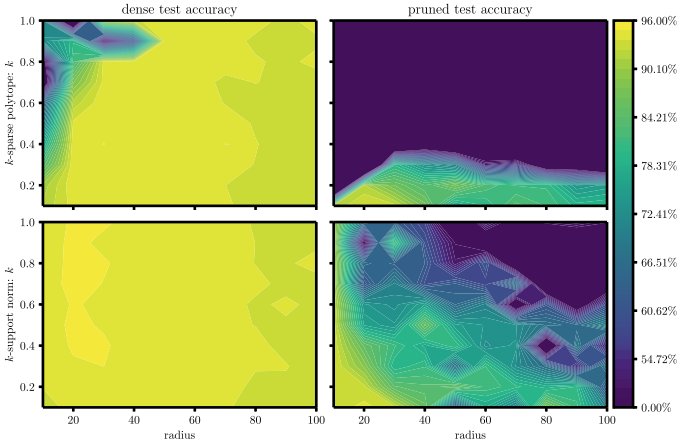


Figure 10.6: ResNet-18 on CIFAR-10: Contour plot when performing a large hyperparameter search over the radius and k of the feasible regions, where the first row corresponds to the k-sparse polytope and the second one corresponds to the k-support norm ball. The left column shows the test accuracy before pruning, while the right column shows the test accuracy after pruning. The k-support norm approach leads to better performing dense models given the hyperparameter search at stake, which in turn are more stable to pruning.

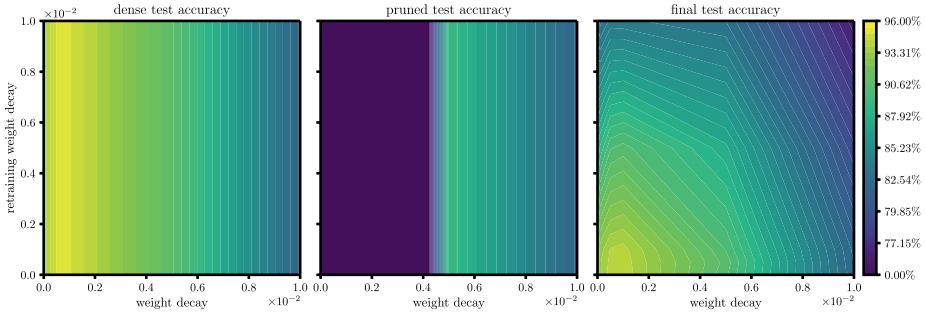


Figure 10.7: ResNet-18 on CIFAR-10: Test accuracy heatmap before pruning (left), after pruning (middle) and after retraining (right) when training SGD and applying One-Shot pruning, tuning both the weight decay during training (x-axis) as well as during retraining (y-axis).

LLR [85]. We tuned both the weight decay for regular retraining as well as the weight decay for the retraining phase. Not surprisingly, there is a weight decay sweet spot when it comes to maximizing the prepruning accuracy (left). The middle plot shows that higher weight decay typically leads to more robustness to pruning, however a too large weight decay hinders convergence of the dense model and might lower the performance after pruning. Surprisingly, however, as depicted in the right plot showing the test accuracy after retraining, the optimal parameter configuration is the one that leads to the highest accuracy before pruning, which is also the least robust to pruning. This aligns with previous findings of Bartoldson et al. [5], who question the strive for pruning stability when retraining is not prohibitive.

A.2.3 Low-rank matrix decomposition

Tables 10.7, 10.8, 10.9, 10.10 and 10.11 compare the filter pruning approaches for CIFAR-10, CIFAR-100, TinyImagenet, ImageNet and CityScapes, respectively.

CIFAR-10 hyperparameter grids

- SparseFW: $k \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$, l_2-multiplier $w \in \{20, 30, 50\}$.
- NUC: Nuclear norm penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
- SVDEnergy: Nuclear norm thresholding λ (1e-2, 5e-2, 1e-1, 3e-1, 5e-1, 7e-1, 9e-1, 1e-0, 5e-0).
- FR: Force regularization penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
- TRP: Nuclear norm penalty factor λ (0, 1e-4, 5e-4, 1e-3), Singular value threshold (2e-2, 5e-2). The reparameterization using the truncated SVD is applied after each epoch except the last.

CIFAR-100 hyperparameter grids

- SparseFW: $k \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$, l_2-multiplier $w \in \{20, 30, 50\}$.
- NUC: Nuclear norm penalty factor λ (1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2).
- SVDEnergy: Nuclear norm thresholding λ (1e-2, 5e-2, 1e-1, 3e-1, 5e-1, 7e-1, 9e-1, 1e-0, 5e-0), Weight decay (1e-4, 2e-4, 5e-4).
- FR: Force regularization penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
- TRP: Nuclear norm penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3), Singular value threshold (1e-1, 2e-1). The reparameterization using the truncated SVD is applied after each epoch except the last.

TinyImageNet hyperparameter grids

- SparseFW: $k \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$, l_2-multiplier $w \in \{20, 30, 50\}$.
- NUC: Nuclear norm penalty factor λ (1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).

– SVDEnergy: Nuclear norm thresholding λ (1e-2, 5e-2, 1e-1, 3e-1, 5e-1, 7e-1, 9e-1, 1e-0, 5e-0).
– FR: Force regularization penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
– TRP: Nuclear norm penalty factor λ (0, 1e-5, 5e-5, 1e-4, 5e-4), Singular value threshold (1e-1, 2e-1). The reparameterization using the truncated SVD is applied after each epoch except the last.

ImageNet hyperparameter grids

If not specified otherwise, we use weight decay values of {1e-4} for all algorithms.
– SparseFW: $k \in \{0.1, 0.2, 0.3\}$, l_2-multiplier $w \in \{20, 30, 50\}$.
– NUC: Nuclear norm penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
– SVDEnergy: Nuclear norm thresholding λ (1e-2, 5e-2, 1e-1, 3e-1, 5e-1, 7e-1, 9e-1, 1e-0, 5e-0).
– FR: Force regularization penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
– TRP: Nuclear norm penalty factor λ (0, 1e-4, 5e-4, 1e-3, 5e-3), Singular value threshold (1e-1, 2e-1). The reparameterization using the truncated SVD is applied after each epoch except the last.

CityScapes hyperparameter grids

– SparseFW: $k \in \{0.1, 0.2, 0.3\}$, l_2-multiplier $w \in \{20, 30, 50\}$.
– NUC: Nuclear norm penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3).
– SVDEnergy: Nuclear norm thresholding λ (1e-6, 5e-6, 1e-5, 5e-5, 1e-2, 5e-2, 1e-1, 5e-1, 1e-0, 5e-0).
– FR: Force regularization penalty factor λ (1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 8e-3, 5e-3, 1e-2, 5e-2).
– TRP: Nuclear norm penalty factor λ (0, 1e-5, 5e-5, 1e-4, 5e-4), Singular value threshold (1e-1, 2e-1). The reparameterization using the truncated SVD is applied after each epoch except the last.

Table 10.7: ResNet-18 on CIFAR-10: Comparison of approaches for encouraging low-rank matrices throughout training. The second column indicates the images-per-second throughput throughout training, where higher throughput corresponds to higher efficiency.

		Sparsity					
Method	**# img/s**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**	**90 %**
Baseline	5664	93.19 ±0.23	93.02 ±0.13	91.66 ±0.17	89.95 ±0.14	82.07 ±1.41	53.07 ±1.98
SparseFW	1156	92.19 ±0.11	92.12 ±0.21	92.14 ±0.23	91.96 ±0.22	90.72 ±0.05	74.94 ±2.40
NUC	566	92.56 ±0.18	92.48 ±0.27	92.59 ±0.17	92.45 ±0.33	89.82 ±0.07	64.83 ±1.16
SVDEnergy	493	92.75 ±0.81	92.62 ±0.64	92.48 ±0.30	91.68 ±0.42	87.99 ±2.18	65.23 ±8.99
FR	4397	94.75 ±0.03	94.46 ±0.01	94.02 ±0.05	92.54 ±0.32	85.39 ±0.43	56.94 ±4.11
TRP	565	92.59 ±0.47	92.68 ±0.37	92.59 ±0.38	92.01 ±0.52	89.05 ±1.97	61.85 ±1.77

Table 10.8: WideResNet on CIFAR-100: Comparison of approaches for encouraging low-rank matrices throughout training. The second column indicates the images-per-second throughput throughout training, where higher throughput corresponds to higher efficiency.

		Sparsity					
Method	**# img/s**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Baseline	1197	75.57 ±0.10	74.28 ±2.59	73.52 ±0.95	64.67 ±11.12	56.04 ±2.24	10.97 ±8.29
SparseFW	372	75.53 ±0.18	75.69 ±0.04	75.75 ±0.08	75.37 ±0.40	75.30 ±0.10	73.28 ±2.02
NUC	204	75.96 ±0.29	74.86 ±1.53	72.35 ±0.48	68.57 ±4.95	55.97 ±2.80	5.40 ±2.12
SVDEnergy	174	75.69 ±0.95	75.14 ±0.18	74.11 ±0.81	66.76 ±4.41	55.40 ±3.04	30.82 ±26.87
FR	1140	73.14 ±1.17	70.52 ±0.58	69.69 ±0.18	60.52 ±3.34	46.86 ±9.03	5.30 ±3.22
TRP	199	74.34 ±0.62	72.80 ±0.29	71.40 ±0.95	64.96 ±1.77	47.74 ±8.07	12.56 ±1.25

Table 10.9: ResNet-50 on TinyImagenet: Comparison of approaches for encouraging low-rank matrices throughout training. The second column indicates the images-per-second throughput throughout training, where higher throughput corresponds to higher efficiency.

		Sparsity				
Method	**# img/s**	**70 %**	**75 %**	**80 %**	**85 %**	**90 %**
Baseline	756	62.90 ±0.66	61.17 ±1.41	56.53 ±1.02	46.01 ±1.76	19.94 ±0.21
SparseFW	338	61.58 ±0.02	61.41 ±0.07	61.00 ±0.01	59.94 ±0.18	54.87 ±0.14
NUC	160	61.98 ±1.05	61.11 ±1.18	58.44 ±1.19	50.89 ±1.11	26.20 ±2.40
SVDEnergy	98	60.94 ±0.46	59.91 ±0.45	58.25 ±0.83	51.59 ±0.67	27.34 ±7.00
FR	742	63.14 ±1.24	61.61 ±1.15	58.07 ±1.73	47.24 ±0.88	22.98 ±1.60
TRP	159	61.72 ±0.43	60.24 ±0.71	58.01 ±0.01	50.02 ±1.03	27.32 ±2.49

Table 10.10: ResNet-50 on Imagenet: Comparison of approaches for encouraging low-rank matrices throughout training. The second column indicates the images-per-second throughput throughout training, where higher throughput corresponds to higher efficiency.

		Sparsity				
Method	**# img/s**	**70 %**	**75 %**	**80 %**	**85 %**	**90 %**
Baseline	1386	73.88 ±0.13	70.22 ±0.09	60.72 ±0.99	37.44 ±1.82	1.88 ±0.19
SparseFW	796	73.27 ±0.18	72.72 ±0.27	71.39 ±0.01	65.18 ±0.28	26.17 ±1.11
NUC	741	74.91 ±0.01	74.29 ±0.02	71.16 ±1.12	57.21 ±1.53	4.60 ±2.66
SVDEnergy	621	74.74 ±0.01	74.12 ±0.06	69.50 ±0.61	53.55 ±0.98	3.19 ±1.09
FR	1418	71.87 ±0.45	67.93 ±0.19	55.98 ±0.05	30.80 ±0.66	2.10 ±0.20
TRP	740	74.91 ±0.08	74.33 ±0.21	70.25 ±1.95	55.21 ±2.53	4.62 ±2.52

A.3 The dynamics of gradient rescaling

We found the denominator of gradient rescaling not to be subject to much variation, whereas the batch gradient norm dynamically changes the learning rate over time. Figure 10.8 compares the evolution of $\|\nabla_t\|$ for two different radii of the k-support norm ball

Table 10.11: PSPNet on CityScapes: Comparison of approaches for encouraging low-rank matrices throughout training. The second column indicates the images-per-second throughput throughout training, where higher throughput corresponds to higher efficiency.

		Sparsity			
Method	**# img/s**	**20 %**	**30 %**	**40 %**	**50 %**
Baseline	38	52.47 ±0.14	47.31 ±0.67	42.95 ±0.48	35.56 ±0.51
SparseFW	11	52.95 ±0.45	49.70 ±0.78	46.63 ±0.61	38.11 ±0.28
NUC	7	51.68 ±0.12	47.67 ±0.03	42.40 ±0.42	37.28 ±0.25
SVDEnergy	5	51.76 ±0.35	46.82 ±0.28	42.81 ±0.33	36.47 ±0.09
FR	37	51.67 ±0.10	46.61 ±0.29	41.60 ±0.34	35.29 ±0.11
TRP	7	51.82 ±0.91	47.96 ±1.32	42.18 ±0.22	37.54 ±0.43

(with fixed k), where we compare to usual SGD training with weight decay. For both SGD and SFW, $\|\nabla_t\|$ is subject to noise and increases until 75 % of the training process, despite the continuous decrease of the train loss. In fact, the batch gradient norm is not significantly smaller than at the start of training even though the loss converges. This behavior might best be explainable by the presence of *batch-normalization* layers, whose interplay with weight decay has been analyzed by van Laarhoven [67] and Hoffer et al. [31]: layers preceding a batch-normalization are rescaling invariant, that is, their output remains unchanged when multiplying all parameters by a scalar; however, rescaling them results in inverse rescaling of the gradient norm in subsequent layers and iterations. Weight decay continuously decreases the scale of the parameters, and hence increases the scale of the batch gradient, where stronger decay of the former leads to stronger increase of the latter. Since in gradient rescaling the norm of the batch gradient also influences the strength of the decay of the parameters, this process has a self-accelerating dynamic. This dynamic results in larger steps toward the (sparse) vertices of the k-support norm ball, leading to a stronger decay on the previous parameter configuration, which in turn increases the robustness to pruning, making gradient rescaling the method of choice in that setting.

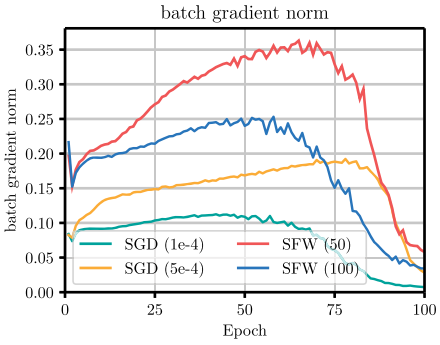


Figure 10.8: ResNet-18 on CIFAR-10: The evolution of the batch gradient norm when training SFW for different values of k and SGD for two weight decay strengths. The metric is averaged with respect to two random seeds and over all iterations within one epoch.

A.4 Proofs of LMO constructions

In the following, we state the missing proof of the k-support norm LMO (being a special case of the group-k-support norm) and Lemma 10.1.

Lemma 10.3. *Given ∇_t, let $v_t \in \mathcal{C}_k(\tau) = \operatorname{conv}\{v \mid \|v\|_0 \leq k, \|v\|_2 \leq \tau\}$ such that*

$$[v_t]_i = \begin{cases} -\tau[\nabla_t]_i / \|\nabla_t^{\operatorname{top}_k}\|_2 & \text{if } i \in \operatorname{top}_k(|\nabla_t|), \\ 0 & \text{otherwise}, \end{cases}$$

where $\nabla_t^{\operatorname{top}_k}$ is the vector obtained by setting to zero all $n-k$ entries $[\nabla_t]_j$ of ∇_t with $j \notin \operatorname{top}_k(|\nabla_t|)$. Then $v_t \in \arg\min_{v \in \mathcal{C}_k(\tau)} \langle v, \nabla_t \rangle$ is a solution to Equation (10.2)*.*

Proof. By construction, all vertices v of $\mathcal{C}_k(\tau)$ satisfy $\|v\|_2 = \tau$ and are k-Sparse, that is, $\|v\|_0 \leq k$. Note that being k-Sparse includes cases where more than $n-k$ entries are zero. The minimum of Equation (10.2) is attained at one such v. Further recall the following reformulation of the euclidean inner product:

$$\langle v, \nabla_t \rangle = \|v\|_2 \|\nabla_t\|_2 \cos\big(\angle(v, \nabla_t)\big) = \tau \|\nabla_t\|_2 \cos\big(\angle(v, \nabla_t)\big), \tag{10.8}$$

where $\angle(v, \nabla_t)$ denotes the angle between v and ∇_t. This term is minimized as soon as the angle between v and ∇_t is maximal. If v was not required to be k-Sparse, that is, v would be allowed to lie anywhere on the border of $B_2(\tau)$, the solution would clearly be given by $-\tau\nabla_t / \|\nabla_t\|_2$. However, since v is k-Sparse, the vector maximizing the angle to ∇_t is the one that is closest to $-\tau\nabla_t / \|\nabla_t\|_2$ but is k-Sparse at the same time. This is exactly the one claimed. □

Lemma 10.4. *Given $\nabla_t \in \mathbb{R}^{n \times m}$, let $\mathcal{W}_t \in \mathcal{C}_k^\sigma(\tau)$ such that*

$$\mathcal{W}_t = \frac{-\tau}{\|\sigma(\Sigma_k)\|_2} U_k \Sigma_k V_k^T,$$

where $U_k \Sigma_k V_k^T$ is the truncated SVD of ∇_t such that only the k largest singular values are kept. Then $\mathcal{W}_t \in \arg\min_{v \in \mathcal{C}_k^\sigma(\tau)} \langle v, \nabla_t \rangle$ is a solution to Equation (10.2)*.*

Proof. Recall that

$$\mathcal{C}_k^\sigma(\tau) = \operatorname{conv}\{\mathcal{W} \in \mathbb{R}^{n \times m} \mid \operatorname{rank}(\mathcal{W}) \leq k, \|\sigma(\mathcal{W})\|_2 \leq \tau\}.$$

Let $\mathcal{W}$ be some minimizer. Note that rescaling a matrix by a scalar has no effect on its rank. Let us hence assume that $\|\sigma(\mathcal{W})\|_2 = \alpha$ for some $\alpha > 0$ and characterize $\mathcal{W} \in \arg\min_{\operatorname{rank}(v) \leq k} \langle v, \nabla_t \rangle$. Again, we have

$$\langle \mathcal{W}, \nabla_t \rangle_F = \langle \overleftarrow{\mathcal{W}}, \overleftarrow{\nabla_t} \rangle_2 = \alpha \|\overleftarrow{\nabla_t}\|_2 \cos\big(\angle(\overleftarrow{\mathcal{W}}, \overleftarrow{\nabla_t})\big), \tag{10.9}$$

where $\overleftarrow{x}$ is there vectorized form of matrix x. Since we can choose $\alpha \leq \tau$, this term is minimal as soon as the angle $\angle(\overleftarrow{\mathcal{W}}, \overleftarrow{\nabla_t})$ is maximal, that is, $\cos(\angle(\overleftarrow{\mathcal{W}}, \overleftarrow{\nabla_t})) < 0$ and $\alpha = \tau$, where

we use the same Euclidean-geometric interpretation as in the proof for Lemma 10.3 above. To obtain a maximal angle, we hence minimize the l_2-distance between $-\overleftarrow{\mathcal{W}}$ and $\overleftarrow{\nabla_t}$ in compliance with the rank constraint. Since again $\|-\overleftarrow{\mathcal{W}} - \overleftarrow{\nabla_t}\|_2 = \|-\mathcal{W} - \nabla_t\|_F$, the Eckart–Young–Mirsky theorem yields the claim, where we rescale appropriately to meet the Schattennorm constraint. □

A.5 Convergence of SFW with gradient rescaling

Before proving the convergence of SFW with gradient rescaling as stated informally in Theorem 10.2, we first recall some central definitions and assumptions.

A.5.1 Setting

Let Ω be the set of training datapoints from which we sample uniformly at random. In Equation (10.1), we defined a unique loss function ℓ_i for each datapoint. In the following, let $\ell(\theta, \omega_i) = \ell_i(\theta)$ for $\omega_i \in \Omega$. Similar to Reddi et al. [60] and Pokutta et al. [57], we define the SFW algorithm as in Algorithm 10.1, where the output θ_a is chosen uniformly at random from all iterates $\theta_0, \dots, \theta_{T-1}$.

Algorithm 10.1 Stochastic Frank–Wolfe (SFW).

Input: Initial parameters $\theta_0 \in \mathcal{C}$, learning rate $\eta_t \in [0,1]$, batch size b_t, number of steps T.

Output: Iterate θ_a chosen uniformly at random from $\theta_0, \dots, \theta_{T-1}$

1: **for** $t = 0$ **to** $T - 1$ **do**
2: sample i. i. d. $\omega_1^{(t)}, \dots, \omega_{b_t}^{(t)} \in \Omega$
3: $\tilde{\nabla} L(\theta_t) \leftarrow \frac{1}{b_t} \sum_{j=1}^{b_t} \nabla \ell(\theta_t, \omega_j^{(t)})$
4: $v_t \leftarrow \arg\min_{v \in \mathcal{C}} \langle \tilde{\nabla} L(\theta_t), v \rangle$
5: $\theta_{t+1} \leftarrow \theta_t + \eta_t(v_t - \theta_t)$
6: **end for**

Let us recall some definitions. We denote the globally optimal solution by $\theta^\star$ and the *Frank–Wolfe Gap* at θ as

$$\mathcal{G}(\theta) = \max_{v \in \mathcal{C}} \langle v - \theta, -\nabla L(\theta) \rangle. \tag{10.10}$$

We will use the same assumptions as Reddi et al. [60]. First of all, let us assume that L is M-smooth, that is,

$$\|\nabla L(x) - \nabla L(y)\| \leq M \|x - y\| \tag{10.11}$$

for all $x, y \in \mathcal{C}$, which implies the well-known inequality

$$L(x) \leq L(y) + \langle \nabla L(y), x - y \rangle + \frac{M}{2}\|x - y\|^2. \tag{10.12}$$

Further, we assume the function ℓ to be G-Lipschitz, that is, for all $x \in \mathcal{C}$ and $\omega \in \Omega$ we have

$$\|\nabla \ell(x, \omega)\| \leq G. \tag{10.13}$$

A direct consequence is that the norm of the gradient estimator can be bounded as $\|\tilde{\nabla} L(\theta_t)\| \leq G$.

A.5.2 Convergence proof

The following well-established lemma quantifies how closely $\tilde{\nabla} L(\theta)$ approximates $\nabla L(\theta)$. A proof can be found in Reddi et al. [60].

Lemma 10.5. *Let $\omega_1, \ldots, \omega_b$ be i. i. d. samples in Ω, $\theta \in \mathcal{C}$, and $\tilde{\nabla} L(\theta) = \frac{1}{b}\sum_{j=1}^{b} \nabla \ell(\theta_t, \omega_j)$. If ℓ is G-Lipschitz, then*

$$\mathbb{E}\|\tilde{\nabla} L(\theta) - \nabla L(\theta)\| \leq \frac{G}{b^{1/2}}. \tag{10.14}$$

In the following, we denote the gradient estimator at iteration t as $\nabla_t \coloneqq \tilde{\nabla} L(\theta_t)$ and the l_2-diameter $\mathcal{D}(\mathcal{C})$ as $\mathcal{D}$. Let $\beta \in \mathbb{R}$ satisfy

$$\beta \geq \frac{2h(\theta_0)}{MD^2}, \tag{10.15}$$

for some given initialization $\theta_0 \in \mathcal{C}$ of the parameters, where $h(\theta_0) = L(\theta_0) - L(\theta^\star)$ denotes the optimality gap of θ_0.

Theorem 10.6. *For all $0 \leq t < T$, let $b_t = b = T$ and $\eta_t = \|\nabla_t\|\eta$ where $\eta = (\frac{h(\theta_0)}{TMD^2G^2\beta})^{1/2}$. If θ_a is chosen uniformly at random from the SFW iterates $\{\theta_i : 0 \leq i < T\}$, then we have*

$$\mathbb{E}\left[\mathcal{G}(\theta_a) \cdot \|\nabla L(\theta_a)\|\right] \leq \frac{D}{\sqrt{T}}\left(\sqrt{h(\theta_0)MG^2\beta} + G^2 + \frac{MGD}{2\sqrt{2}}\right),$$

where $\mathbb{E}$ denotes the expectation w. r. t. all the randomness present.

Proof. First of all, notice that η_t is well-defined: Using β as defined above, we have

$$\eta \leq \left(\frac{1}{2TG^2}\right)^{1/2} = \frac{1}{G}\frac{1}{\sqrt{2T}}, \tag{10.16}$$

and consequently, we obtain $\eta_t = \|\nabla_t\|\eta \leq \frac{1}{\sqrt{2T}} \leq 1$ by using that $\|\nabla_t\| \leq G$. By M-smoothness of L, we have

$$L(\theta_{t+1}) \leq L(\theta_t) + \langle \nabla L(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{M}{2} \|\theta_{t+1} - \theta_t\|^2.$$

Using the fact that $\theta_{t+1} = \theta_t + \eta_t(v_t - \theta_t)$ and that $\|v_t - \theta_t\| \leq D$, it follows that

$$L(\theta_{t+1}) \leq L(\theta_t) + \eta_t \langle \nabla L(\theta_t), v_t - \theta_t \rangle + \frac{MD^2\eta_t^2}{2}. \tag{10.17}$$

Now let

$$\hat{v}_t = \arg\min_{v \in \mathcal{C}} \langle \nabla L(\theta_t), v \rangle = \arg\max_{v \in \mathcal{C}} \langle -\nabla L(\theta_t), v \rangle \tag{10.18}$$

be the LMO solution if we knew the exact gradient at iterate θ_t, where $t = 0, \ldots, T-1$. This minimizer is not part of the algorithm but is crucial in the subsequent analysis. Note that we have

$$\mathcal{G}(\theta_t) = \max_{v \in \mathcal{C}} \langle v - \theta_t, -\nabla L(\theta_t) \rangle = \langle \hat{v}_t - \theta_t, -\nabla L(\theta_t) \rangle. \tag{10.19}$$

Continuing from Equation (10.17), we therefore have

$$\begin{aligned}
L(\theta_{t+1}) &\leq L(\theta_t) + \eta_t \langle \tilde{\nabla} L(\theta_t), v_t - \theta_t \rangle + \eta_t \langle \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t), v_t - \theta_t \rangle + \frac{MD^2\eta_t^2}{2} \\
&\leq L(\theta_t) + \eta_t \langle \tilde{\nabla} L(\theta_t), \hat{v}_t - \theta_t \rangle + \eta_t \langle \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t), v_t - \theta_t \rangle + \frac{MD^2\eta_t^2}{2} \\
&= L(\theta_t) + \eta_t \langle \nabla L(\theta_t), \hat{v}_t - \theta_t \rangle + \eta_t \langle \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t), v_t - \hat{v}_t \rangle + \frac{MD^2\eta_t^2}{2} \\
&= L(\theta_t) - \eta_t \mathcal{G}(\theta_t) + \eta_t \langle \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t), v_t - \hat{v}_t \rangle + \frac{MD^2\eta_t^2}{2},
\end{aligned}$$

where the first inequality is just a reformulation of Equation (10.17) and the second one is due to the minimality of v_t. Applying Cauchy–Schwarz and using the fact that the diameter of $\mathcal{C}$ is D, we therefore have

$$L(\theta_{t+1}) \leq L(\theta_t) - \eta_t \mathcal{G}(\theta_t) + \eta_t D \left\| \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t) \right\| + \frac{MD^2\eta_t^2}{2}. \tag{10.20}$$

Now note that $\eta_t = \|\nabla_t\| \eta \leq G\eta$, yielding

$$L(\theta_{t+1}) \leq L(\theta_t) - \eta_t \mathcal{G}(\theta_t) + \eta GD \left\| \nabla L(\theta_t) - \tilde{\nabla} L(\theta_t) \right\| + \frac{MD^2G^2\eta^2}{2}. \tag{10.21}$$

Let $\theta_{0:t}$ denote the sequence $\theta_0, \ldots, \theta_t$. Taking expectations and applying Lemma 10.5, we get

$$\mathbb{E}_{\theta_{0:t+1}} L(\theta_{t+1}) \leq \mathbb{E}_{\theta_{0:t+1}} L(\theta_t) - \mathbb{E}_{\theta_{0:t+1}} [\eta_t \mathcal{G}(\theta_t)] + \frac{DG^2\eta}{b^{1/2}} + \frac{MD^2G^2\eta^2}{2}. \tag{10.22}$$

By rearranging and summing over $t = 0, \dots, T-1$, we get the upper bound

$$\sum_{t=0}^{T-1} \mathbb{E}_{\theta_{0:t+1}}[\eta_t \mathcal{G}(\theta_t)] \leq L(\theta_0) - \mathbb{E}_{\theta_{0:T}} L(\theta_T) + \frac{TDG^2\eta}{b^{1/2}} + \frac{TMD^2G^2\eta^2}{2}$$
$$\leq L(\theta_0) - L(\theta^*) + \frac{TDG^2\eta}{b^{1/2}} + \frac{TMD^2G^2\eta^2}{2}. \tag{10.23}$$

Now fix t and apply the law of total expectation to reformulate

$$\mathbb{E}_{\theta_{0:t+1}}[\eta_t \mathcal{G}(\theta_t)] = \mathbb{E}_{\theta_{0:t}} \mathbb{E}_{\theta_{0:t+1}}[\eta_t \mathcal{G}(\theta_t) \mid \theta_{0:t}] = \mathbb{E}_{\theta_{0:t}}[\mathcal{G}(\theta_t)\eta \cdot \mathbb{E}_{\theta_{0:t+1}}[\|\nabla_t\| \mid \theta_{0:t}]], \tag{10.24}$$

where we exploited that once $\theta_{0:t}$ is available, $\mathcal{G}(\theta_t)$ is not subject to randomness anymore. The expected norm of the gradient estimator given θ_t depends only on the uniform selection of samples, allowing us to exploit the unbiasedness of the estimator as well as the convexity of the norm $\|\cdot\|$ using Jensen's inequality as follows:

$$\mathbb{E}_{\theta_{0:t+1}}[\|\nabla_t\| \mid \theta_{0:t}] = \mathbb{E}_{\omega}[\|\nabla_t\| \mid \theta_{0:t}] \tag{10.25}$$
$$\geq \left\|\mathbb{E}_{\omega}[\nabla_t \mid \theta_{0:t}]\right\| \tag{10.26}$$
$$= \left\|\frac{1}{b}\sum_{j=1}^{b} \mathbb{E}_{\omega_j} \nabla \ell(\theta_t, \omega_j)\right\| \tag{10.27}$$
$$= \left\|\nabla L(\theta_t)\right\|. \tag{10.28}$$

Combining this with Equation (10.23), we obtain

$$\eta \sum_{t=0}^{T-1} \mathbb{E}_{\theta_{0:t}}[\mathcal{G}(\theta_t) \cdot \left\|\nabla L(\theta_t)\right\|] \leq h(\theta_0) + \frac{TDG^2\eta}{b^{1/2}} + \frac{TMD^2G^2\eta^2}{2}. \tag{10.29}$$

Using the definition of θ_a, being a uniformly at random chosen iterate from $\theta_0, \dots, \theta_{T-1}$, we conclude the proof with the following inequality:

$$\mathbb{E}\left[\mathcal{G}(\theta_a) \cdot \left\|\nabla L(\theta_a)\right\|\right] \leq \frac{h(\theta_0)}{T\eta} + \frac{DG^2}{b^{1/2}} + \frac{MD^2G^2\eta}{2} \tag{10.30}$$
$$\leq \frac{D}{\sqrt{T}}\left(\sqrt{h(\theta_0)MG^2\beta} + G^2 + \frac{MGD}{2\sqrt{2}}\right) \tag{10.31}$$

□

Bibliography

[1] Z. Allen-Zhu and Y. Li. Lazysvd: even faster SVD decomposition yet without agonizing pain. *Advances in Neural Information Processing Systems*, 29, 2016.

[2] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[3] J. M. Alvarez and M. Salzmann. Compression-aware training of deep networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[4] A. Argyriou, R. Foygel, and N. Srebro. Sparse prediction with the k-support norm. *Advances in Neural Information Processing Systems*, 25, 2012.

[5] B. Bartoldson, A. Morcos, A. Barbu, and G. Erlebacher. The generalization-stability tradeoff in neural network pruning. In: H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20852–20864. Curran Associates, Inc., 2020.

[6] A. H. Bentbib and A. Kanber. Block power method for svd decomposition. *Analele Universităţii "Ovidius" Constanţa. Seria Matematică*, 23(2):45–58, 2015.

[7] L. Berrada, A. Zisserman, and M. Pawan Kumar. Deep Frank–Wolfe for neural network optimization. In: *International Conference on Learning Representations 2019*, November 2018.

[8] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In: I. Dhillon, D. Papailiopoulos and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020.

[9] G. Braun, A. Carderera, C. W. Combettes, H. Hassani, A. Karbasi, A. Mokhtari, and S. Pokutta. Conditional gradient methods, November 2022.

[10] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-all: train one network and specialize it for efficient deployment. In: *International Conference on Learning Representations*, 2020.

[11] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[12] A. Carderera, S. Pokutta, C. Schütte, and M. Weiser. Cindy: conditional gradient-based identification of non-linear dynamics—noise-robust recovery, January 2021.

[13] M. A. Carreira-Perpinán and Y. Idelbayev. Learning-compression algorithms for neural net pruning. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] L. Chen, C. Harshaw, H. Hassani, and A. Karbasi. Projection-free online optimization with stochastic gradient: from convexity to submodularity. In: *International Conference on Machine Learning*, pages 814–823. PMLR, 2018.

[15] C. W. Combettes and S. Pokutta. Complexity of linear minimization and projection on some sets, January 2021.

[16] C. W. Combettes, C. Spiegel, and S. Pokutta. Projection-free adaptive gradients for large-scale optimization, September 2020.

[17] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016.

[18] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in Neural Information Processing Systems*, 27, 2014.

[19] X. Ding, G. Ding, J. Han, and S. Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[20] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu. Global sparse momentum SGD for pruning very deep neural networks. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[21] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: making all tickets winners. In: Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, 13–18 Jul, 2020. Proceedings of Machine Learning Research, volume 119, pages 2943–2952. PMLR, 2020.

[22] E. Frandi, R. Nanculef, S. Lodi, C. Sartori, and J. A. K. Suykens. Fast and scalable lasso via stochastic Frank–Wolfe methods with a convergence guarantee, October 2015.
[23] M. Frank, P. Wolfe, et al. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1–2):95–110, 1956.
[24] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. arXiv:1902.09574, 2019.
[25] P. Grigas, A. Lobos, and N. Vermeersch. Stochastic in-face Frank–Wolfe methods for non-convex optimization and sparse neural network training, June 2019.
[26] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural networks. In: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
[27] E. Hazan and H. Luo. Variance-reduced and projection-free stochastic optimization. In: *International Conference on Machine Learning*, pages 1263–1271. PMLR, 2016.
[28] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.
[29] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks, August 2018.
[30] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. arXiv:2102.00554, January 2021.
[31] E. Hoffer, R. Banner, I. Golan, and D. Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
[32] Y. Idelbayev and M. A. Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.
[33] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, Lille, France, 6–11 July, 2015. JMLR Workshop and Conference Proceedings, volume 37, pages 448–456. JMLR.org, 2015.
[34] M. Jaggi. Revisiting Frank–Wolfe: projection-free sparse convex optimization. In: *Proceedings of the 30th International Conference on Machine Learning*, pages 427–435, 2013.
[35] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications, November 2015.
[36] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009.
[37] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In: Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, 13–18 Jul, 2020. Proceedings of Machine Learning Research, volume 119, pages 5544–5555. PMLR, 2020.
[38] S. Lacoste-Julien. Convergence rate of Frank–Wolfe for non-convex objectives, July 2016.
[39] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank–Wolfe optimization for structural SVMs. In: *International Conference on Machine Learning*, pages 53–61. PMLR, 2013.
[40] D. H. Le and B.-S. Hua. Network pruning that matters: a case study on retraining variants. In: *International Conference on Learning Representations*, 2021.
[41] Y. Le and X. Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.
[42] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition, December 2014.
[43] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin. Layer-adaptive sparsity for the magnitude-based pruning. In: *International Conference on Learning Representations*, October 2020.

[44] E. S. Levitin and B. T. Polyak. Constrained minimization methods. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 6(5):1–50, 1966.
[45] B. Li, B. Wu, J. Su, and G. Wang. Eagleeye: fast sub-net evaluation for efficient neural network pruning. In: *Computer Vision—ECCV 2020: 16th European Conference*, Glasgow, UK, August 23–28, 2020. Proceedings, Part II 16, pages 639–654. Springer, 2020.
[46] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets, August 2016.
[47] M. Li, E. Yumer, and D. Ramanan. Budgeted training: rethinking deep neural network training under resource constraints. In: *International Conference on Learning Representations*, 2020.
[48] L. Liebenwein, A. Maalouf, O. Gal, D. Feldman, and D. Rus. Compressing neural networks: towards determining the optimal layer-wise decomposition, July 2021.
[49] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. In: *International Conference on Learning Representations*, 2020.
[50] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. H. So. Dynamic sparse training: find efficient sparse network from scratch with trainable masked layers. In: *International Conference on Learning Representations*, 2020.
[51] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning, October 2018.
[52] A. McDonald, M. Pontil, and D. Stamos. Fitting spectral decay with the k-support norm. In: *Artificial Intelligence and Statistics*, pages 1061–1069. PMLR, 2016.
[53] L. Miao, X. Luo, T. Chen, W. Chen, D. Liu, and Z. Wang. Learning pruning-friendly networks via Frank–Wolfe: one-shot, any-sparsity, and no retraining. In: *International Conference on Learning Representations*, 2022.
[54] A. Mokhtari, H. Hassani, and A. Karbasi. Conditional gradient method for stochastic submodular maximization: closing the gap. In: *International Conference on Artificial Intelligence and Statistics*, pages 1886–1895. PMLR, 2018.
[55] G. Négiar, G. Dresdner, A. Tsai, L. El Ghaoui, F. Locatello, R. Freund, and F. Pedregosa. Stochastic Frank–Wolfe for constrained finite-sum minimization. In: *International Conference on Machine Learning*, pages 7253–7262. PMLR, 2020.
[56] A. Peste, A. Vladu, D. Alistarh, and C. H. Lampert. Cram: a compression-aware minimizer, July 2022.
[57] S. Pokutta, C. Spiegel, and M. Zimmer. Deep neural network training with Frank–Wolfe. arXiv:2010.07243, 2020.
[58] N. Rao, M. Dudík, and Z. Harchaoui. The group k-support norm for learning with structured sparsity. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2402–2406, 2017.
[59] S. N. Ravi, T. Dinh, V. Lokhande, and V. Singh. Constrained deep learning using conditional gradient and applications in computer vision, March 2018.
[60] S. J. Reddi, S. Sra, B. Póczos, and A. Smola. Stochastic Frank–Wolfe methods for nonconvex optimization. In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1244–1251. IEEE, 2016.
[61] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. In: *International Conference on Learning Representations*, 2020.
[62] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
[63] Z. Shen, C. Fang, P. Zhao, J. Huang, and H. Qian. Complexities in projection-free stochastic non-convex minimization. In: *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2868–2876. PMLR, 2019.
[64] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP, June 2019.

[65] C. Tai, T. Xiao, Y. Zhang, X. Wang, and E. Weinan. Convolutional neural networks with low-rank regularization, November 2015.
[66] T. Tsiligkaridis and J. Roberts. On Frank–Wolfe optimization for adversarial robustness and interpretability, December 2020.
[67] T. van Laarhoven. L2 regularization versus batch and weight normalization, June 2017.
[68] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks, August 2016.
[69] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks, March 2017.
[70] M. Wortsman, A. Farhadi, and M. Rastegari. Discovering neural wirings. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
[71] J. Xie, Z. Shen, C. Zhang, B. Wang, and H. Qian. Efficient projection-free online methods with stochastic recursive gradient, October 2019.
[72] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. TRP: trained rank pruning for efficient deep neural networks, April 2020.
[73] H. Yang, M. Tang, W. Wen, F. Yan, D. Hu, A. Li, H. Li, and Y. Chen. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 678–679, 2020.
[74] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B, Statistical Methodology*, 68(1):49–67, 2006.
[75] X. Yuan, P. H. Pamplona Savarese, and M. Maire. Growing efficient deep networks by structured continuous sparsification. In: *International Conference on Learning Representations*, 2021.
[76] A. Yurtsever, S. Sra, and V. Cevher. Conditional gradient methods via stochastic path-integrated differential estimator. In: *International Conference on Machine Learning*, pages 7282–7291. PMLR, 2019.
[77] S. Zagoruyko and N. Komodakis. Wide residual networks. arXiv:1605.07146, May 2016.
[78] X. Zeng and M. A. T. Figueiredo. The ordered weighted ℓ_1 norm: atomic formulation, projections, and algorithms, September 2014.
[79] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. arXiv:1611.03530, November 2016.
[80] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1943–1955, Oct. 2016.
[81] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2881–2890, 2017.
[82] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv:1710.01878, October 2017.
[83] T. Zhuang, Z. Zhang, Y. Huang, X. Zeng, K. Shuang, and X. Li. Neuron-level structured pruning using polarization regularizer. *Advances in Neural Information Processing Systems*, 33:9865–9877, 2020.
[84] M. Zimmer, M. Andoni, C. Spiegel, and S. Pokutta. PERP: rethinking the prune-retrain paradigm in the era of LLMs, December 2023.
[85] M. Zimmer, C. Spiegel, and S. Pokutta. How I learned to stop worrying and love retraining. In: *International Conference on Learning Representations*, 2023.
[86] M. Zimmer, C. Spiegel, and S. Pokutta. Sparse model soups: a recipe for improved pruning via model averaging. In: *The Twelfth International Conference on Learning Representations*, 2024.

Antonio Carlucci, Ion Victor Gosea, and Stefano Grivet-Talocia

Approximation of generalized frequency response functions via vector fitting

Abstract: This paper describes a method for data-driven approximation of nonlinear systems, using the notion of generalized frequency response functions as provided by the Volterra series theory. Using rational approximation, frequency-domain input–output data are fitted to a bilinear model structure. The proposed algorithm performs optimization of model coefficients through a greedy approach that relies on linear least-squares solves only, thus ensuring speed and scalability.

Keywords: Volterra series, bilinear systems, vector fitting algorithm, generalized frequency response functions, least-squares fitting, Burgers' equation

MSC 2020: 41A20, 93C10, 93C80

1 Introduction and background

The Volterra series theory for nonlinear systems [21, 20] is a modeling approach based on expanding a given time-invariant input–output map $y(t) = G[u(t)]$ using a series of generalized time-domain convolutions as

$$y(t) = \sum_{m=1}^{\infty} \int_0^t h_m(\tau_1, \ldots, \tau_m) u(t-\tau_1) \cdots u(t-\tau_m) \mathrm{d}\tau_1 \ldots \mathrm{d}\tau_m, \tag{11.1}$$

where $u(t)$ is the system's input (vanishing for $t < 0$), while $y(t)$ is the output. This paper focuses on single-input, single-output (SISO) systems, that is, $u(t), y(t) \in \mathbb{R}$ are scalars. In (11.1), the output is decomposed as a sum of contributions $y_m(t)$ from *homogeneous* subsystems, whose response is given as a multidimensional convolution with the one-sided kernel $h_m(\tau_1, \ldots, \tau_m)$. This series expansion is typically used to model mildly nonlinear systems since it has local validity around an operating point (in this case, $u(t) = 0$) and it is ensured to be convergent if the amplitude of $u(t)$ is sufficiently small (see [20, 5] for details). Equation (11.1) is not enough to uniquely define the kernels h_m, since different choices can describe the same input–output relation. Requiring that the h_m's are symmetric, that is, invariant under arbitrary permutations of the arguments, leads to

Antonio Carlucci, **Stefano Grivet-Talocia,** Department of Electronics and Telecommunications, Politecnico di Torino, 24 Corso Duca degli Abruzzi, 10129 Turin, Italy, e-mails: antonio.carlucci@polito.it, stefano.grivet@polito.it

Ion Victor Gosea, Max Planck Institute for Dynamics of Complex Technical Systems, CSC group, Sandtorstraße 1, 39106 Magdeburg, Germany, e-mail: gosea@mpi-magdeburg.mpg.de

https://doi.org/10.1515/9783111376776-011

a unique definition that is particularly convenient because they can be easily obtained analytically and measured from simulations. This formalism is valuable from a modeling standpoint because it generalizes the well-known convolution formula of linear systems, and allows defining generalized frequency response functions (GFRFs) through the multivariate Laplace transform [10]

$$H_m(s_1,\dots,s_m) = \int_0^\infty \cdots \int_0^\infty h_m(\tau_1,\dots,\tau_m) e^{-\sum_{i=1}^m s_i\tau_i} \mathrm{d}\tau_1 \dots \mathrm{d}\tau_m \tag{11.2}$$

where $h_m(\tau_1,\dots,\tau_m)$ is the symmetric kernel and $H_m(s_1,\dots,s_m)$ is also called the degree-m symmetric transfer function.

The objective of this paper is to derive a dynamical model starting from sampled data of symmetric transfer functions $H_m(s_1,\dots,s_m)$. To this aim, a particularly convenient choice of the model structure is that of bilinear systems. These can be represented with the following state-space form:

$$\dot{x}(t) = Ax(t) + Nx(t)u(t) + Bu(t), \quad y(t) = Cx(t). \tag{11.3}$$

Such systems can be also viewed as time-varying linear systems, by rewriting the state equation as $\dot{x}(t) = \overline{A}(t)x(t) + Bu(t)$ with $\overline{A}(t) = A + Nu(t)$. Therefore, stability and other relevant properties can be deduced from the already existing theory for linear systems, and hence, the bilinear structure has proven very useful and tractable for modeling. It is also sufficiently general since a broad class of systems with analytic nonlinearities can be embedded into a bilinear representation, for example, through the Carleman linearization approach [7] as in [6]. Additionally, by imposing specific boundary conditions for PDEs as in [3], bilinear models can be derived after semidiscretization in the spatial domain. For bilinear systems, several model reduction (MOR) techniques have been described as an immediate extension of their linear counterpart. Intrusive methods for MOR of bilinear systems, mostly based on moment matching (rational Krylov approaches) or balancing, were proposed in [6, 3] (see also the references therein), together with optimality-enforced methods and matching of infinite series [23, 2, 13]. Then data-driven approaches were also proposed, starting with a direct extension of the Loewner framework to bilinear systems in [1], and with more recent work in [16] that uses time-domain data to infer values of (symmetric) transfer functions; the latter was extended to quadratic systems in [17]. Subspace identification techniques for bilinear systems were proposed in [12], while the problem of structure-preserving MOR was treated in [4]. This paper proposes a novel approach toward data-driven approximation of GFRFs based on rational fitting. We provide an extension of the vector fitting algorithm to the nonlinear setting, featuring a greedy strategy whereby a sequence of linear least-squares problems is solved to optimize the coefficients of a conveniently chosen bilinear model structure.

This paper is organized as follows. The notation and the model structure are provided in Section 2. The core ideas for fitting GFRFs are introduced in Section 2.1, followed

by a discussion on the relation between the fitting error and the input–output response in Section 2.2. Then algorithmic tools for rational approximation based on Vector Fitting (VF) are reviewed and adapted in Section 2.3. Two numerical experiments are studied and the results are reported in Section 3, before concluding with Section 4.

2 Formulation

In order to introduce the problem data, let us use the shorthand $\mathbf{s}_m \triangleq (s_1, \dots, s_m)$ to denote a point in the m-dimensional frequency space. The starting point for model inference is thus a data set $\mathcal{H} = \{\mathcal{H}_m\}_{m=1}^{M}$ with evaluations of degree-m transfer functions at an arbitrary set of points

$$\mathcal{H}_m = \{(\mathbf{s}_m^{(k)}, \check{H}_m^{(k)}),\ k = 1, \dots, K_m\}, \quad m = 1, \dots, M. \tag{11.4}$$

This data is fitted with a predefined model structure corresponding to a particular bilinear dynamical system G with the following form:

$$G: \begin{cases} \dot{x}_1(t) = A_1 x_1(t) + B_1 u(t), \\ \dot{x}_m(t) = A_m x_m(t) + N_m x_{m-1}(t) u(t), \quad 2 \le m \le M, \\ y(t) = \sum_{m=1}^{M} C_m x_m(t). \end{cases} \tag{11.5}$$

This was introduced in [20] as a way to construct bilinear realizations of Volterra transfer functions. Its structure is such that all kernels beyond the first M are zero, implying that G is a *polynomial* system of degree M (i. e., the sum in (11.1) contains M terms). In addition, state equations corresponding to every individual m in (11.5) correspond to the mth degree *homogeneous subsystem* of G [20], that is, the mth term in (11.1). The system (11.5) can be interpreted as a cascade of linear systems with a multiplicative nonlinearity in between. Denoting with $H_m^{(k)}$ the symmetric transfer functions of G evaluated at $\mathbf{s}_m^{(k)}$, the model coefficients should be optimized so that H_m is close to $\check{H}_m$, that is, $H_m^{(k)} \approx \check{H}_m^{(k)}$.

It is to be mentioned that G in (11.5) could be recast into a standard bilinear system format as in (11.3). The way to do this is as suggested by the realization procedure in [20], that is, by putting together the bilinear system matrices (A, B, C, N), as follows:

$$A = \operatorname{blkdiag}\{A_1, \dots, A_M\}, \quad B = \operatorname{col}\{B_1, 0, \dots, 0\},$$
$$N = \begin{pmatrix} 0 & & & \\ N_2 & 0 & & \\ & N_3 & 0 & \\ & & \ddots & \ddots \end{pmatrix}, \quad C = \begin{pmatrix} C_1 & \cdots & C_M \end{pmatrix}. \tag{11.6}$$

Different from [16], where the matrix N of the fitted bilinear system is computed to match the realization (A, B, C) computed with the Loewner framework in the first step, here we construct individual submodels (A_m, N_m, C_m) at each step $m \geq 2$. If necessary, these submodels may be assembled into a standard bilinear system, by following (11.6).

2.1 Fitting of symmetric transfer functions

The symmetric transfer functions of (11.5) can be expressed recursively. Starting from $m = 1$, let us define the auxiliary functions

$$X_1(s_1) = (s_1 I - A_1)^{-1} B_1, \tag{11.7}$$

$$X_m(\mathbf{s}_m) = \left[(s_1 + \cdots + s_m)I - A_m\right]^{-1} N_m Q_m(\mathbf{s}_m), \quad m \geq 2, \tag{11.8}$$

$$Q_m(\mathbf{s}_m) = \sum_{i=1}^{M} X_{m-1}(\mathbf{s}_m \backslash s_i), \tag{11.9}$$

with $\mathbf{s}_m \backslash s_i \triangleq (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_m)$. The symmetric transfer functions become $H_m(\mathbf{s}_m) = C_m X_m(\mathbf{s}_m)$, with the explicit formulation below:

$$H_m(\mathbf{s}_m) = \begin{cases} C_1(s_1 I - A_1)^{-1} B_1, & m = 1, \\ \frac{1}{m!} C_m[(s_1 + \cdots + s_m)I - A_m]^{-1} N_m Q_m(\mathbf{s}_m), & m \geq 2. \end{cases} \tag{11.10}$$

Expressions (11.9)–(11.10) can be easily derived for the assumed system structure (11.5) by applying the so-called growing exponential approach; see [20].

In principle, the approximation problem we need to solve is to find model parameters so that the following cost function is minimized:

$$J = \sum_{m=1}^{M} \sum_{k=1}^{K_m} \left| H_m(s_m^{(k)}) - \check{H}_m(s_m^{(k)}) \right|^2. \tag{11.11}$$

Our approach is to greedily optimize individual terms of the summation corresponding to each m. Starting from $m = 1$, we consider the approximation

$$C_1(s_1^{(k)} I - A_1)^{-1} B_1 \approx \check{H}_1(s_1^{(k)}), \quad k = 1, \ldots, K_1, \tag{11.12}$$

that is equivalent to fitting a univariate rational function to given data, a problem known as *rational fitting* and efficiently solvable using existing algorithms, such as AAA [19] or vector fitting [15].

As for $m > 1$, the recursive expression in (11.10) clearly shows the main idea of our formulation, based on observing that the mth degree transfer function is the product of a univariate transfer function

$$F_m(s) = \frac{1}{m!} C_m (sI - A_m)^{-1} N_m, \tag{11.13}$$

evaluated at $s = s_1 + \cdots + s_m$ with the function $Q_m(\mathbf{s}_m)$, which only depends on lower-degree subsystems. This implies that, if the system coefficients $A_1, \ldots, A_m$ and $B_1, N_2, \ldots, N_m$ are known up to index m, the quantity $Q_m(\mathbf{s}_m)$ is entirely determined. Therefore, the approximation problem is reduced to optimizing the coefficients of $F_m(s)$ for each higher-degree index $m = 2, \ldots, M$ so that

$$F_m(s_1^{(k)} + \cdots + s_m^{(k)})Q(\mathbf{s}_m^{(k)}) \approx \check{H}_m(\mathbf{s}_m^{(k)}), \quad k = 1, \ldots, K_m, \tag{11.14}$$

which can be tackled again with rational fitting algorithms. The greedy aspect of this method lies in considering $Q(\mathbf{s}_m^{(k)})$ a given constant as resulting from optimization of previous terms of the summation (11.11), rather than jointly optimizing all system parameters in (11.5) simultaneously.

2.2 Cost function and model error

The motivation for using J in (11.11) as a measure of model mismatch is that it works as a proxy for the H_2 norm of the error system. In particular, using the notation $\check{h}_m(\tau_1, \ldots, \tau_n)$ for the *true* Volterra kernels of the original system and $h_m(\tau_1, \ldots, \tau_n)$ for the fitted degree-M model, we can define the error $e_m(\tau_1, \ldots, \tau_n) = h_m(\tau_1, \ldots, \tau_n) - \check{h}_m(\tau_1, \ldots, \tau_n)$. Following the definition in [23, 13], the H_2 norm of the error system E would be

$$\|E\|_{H_2} = \sqrt{\sum_{m=1}^{\infty} \|e_m\|_2^2}, \quad \|e_m\|_2^2 = \int_0^{\infty} \cdots \int_0^{\infty} e_m(\tau_1, \ldots, \tau_m)^2 d\tau_1 \cdots d\tau_m. \tag{11.15}$$

Considering the magnitude of the output error $e(t) = \check{y}(t) - y(t)$,

$$|e(t)| \leq \sqrt{\sum_{m=1}^{M} \|e_m\|_2^2} \sqrt{\sum_{m=1}^{M} \|u\|_2^{2m}} + \left| \sum_{m=M+1}^{\infty} \check{y}_m(t) \right|. \tag{11.16}$$

This estimate shows that the error contains two components, the first one depends on the modeling error of the first M kernels, while the second is the unmodeled nonlinearity of degree higher than M. The connection with the cost function J is made through Plancherel's theorem to express $\|e_m\|_2^2$ in terms of the H_2-norm of $E_m(\mathbf{s}_m) = H_m(\mathbf{s}_m) - \check{H}_m(\mathbf{s}_m)$,

$$\sum_{m=1}^{M} \|e_m\|_2^2 = \sum_{m=1}^{M} \frac{1}{(2\pi)^m} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \left| E_m(j\omega_1, \ldots, j\omega_m) \right|^2 d\omega_1 \ldots d\omega_m, \tag{11.17}$$

and J can be viewed as a discrete analogue of the frequency-domain integrals.

As for the choice of M, Equation (11.16) suggests that it should be chosen as the smallest degree for which the error due to the higher-degree responses ($m \geq M+1$) are cumu-

latively negligible compared to the modeling error $\sum_{m=1}^{M} \|e_m\|^2$ arising from the fitting procedure.

2.3 Weighted vector fitting for GFRFs

This section completes the formulation by describing the main enabling tool to solve the approximation problems (11.14) and (11.12), that is, the weighted vector fitting algorithm. In particular, (11.12) can be tackled with the standard formulation of VF [15, 11, 14]. On the other hand, (11.14) requires incorporating the arbitrary but known weighting vectors $Q(\mathbf{s}_m^{(k)})$ using a modified formulation such as [8], here reviewed for the specific problem at hand. First of all, consider problem (11.14) for a particular m, restated as

$$\text{minimize} \sum_{k=1}^{K} \left| F_m(s_1^{(k)} + \cdots + s_m^{(k)}) Q(\mathbf{s}_m^{(k)}) - \check{H}_m^{(k)} \right|^2, \tag{11.18}$$

where $F_m(s)$ is unknown and represented in pole-residue form as

$$F_m(s) = \sum_{i=1}^{\nu} \frac{R_i}{s - p_i}, \tag{11.19}$$

with the index m being omitted in both the R_i's and also the p_i's. Note that this pole-residue expansion is another way of parameterizing $F_m(s)$ through coefficients R_i and p_i, instead of the coefficients C_m, A_m, N_m used in (11.13). However, it is always possible to write $F_m(s)$ in the form (11.13) starting from the pole-residue (11.19), through a *realization* procedure (see [14]).

Going back to the optimization problem in (11.18), a key observation is that if p_i are given constants, (11.18) is an easily solvable linear least-squares (LS) problem in the variables R_i. However, if p_i are to be optimized, this is no longer true and the problem becomes much harder. For this reason, VF goes through a preliminary pole identification phase to find poles p_i through an iterated solution of a simplified problem. In this stage, a double barycentric form is used to represent $F_m(s)$ as

$$F_m(s) = \frac{N(s)}{d(s)}, \quad N(s) = \sum_{i=1}^{\nu} \frac{N_i}{s - q_i}, \quad d(s) = 1 + \sum_{i=1}^{\nu} \frac{d_i}{s - q_i}, \tag{11.20}$$

using a set of auxiliary poles $\{q_i\}_{i=1}^{\nu}$. Starting from an initial guess of q_i, this is updated through several iterations where the following modified cost function is minimized:

$$\min \sum_{k=1}^{K} \left| N(s_1^{(k)} + \cdots + s_m^{(k)}) Q(\mathbf{s}_m^{(k)}) - \check{H}_m^{(k)} d(s_1^{(k)} + \cdots + s_m^{(k)}) \right|^2. \tag{11.21}$$

Solving (11.21) at a given iteration yields coefficients d_i of $d(s)$, that are used to update the auxiliary poles for the next iteration through the rule $q_i \leftarrow \text{zeros}\{d(s)\}$. In this way,

(11.21) is optimized repeatedly with an updated choice of q_i each time. Upon convergence or after a maximum number of iterations, the q_i's are chosen to be the model poles p_i in (11.19). Finally, (11.18) and (11.19) with fixed p_i are solved via linear LS to find R_i.

Convergence to a given set of poles q_i corresponds to the condition $d(s) \to 1$, detected either by checking whether the norm of coefficients $\sum_{i=1}^{\nu} d_i^2$ falls below a predefined tolerance, or by looking at the maximum deviation $|d(s) - 1|_\infty$. Convergence of VF is discussed in [18], where it is shown that the algorithm might not converge in a strong sense. Nonetheless, VF still produces good solutions to the rational approximation problem, for which a globally optimal solution is hard to obtain due to the strongly nonconvex nature of the related optimization problem. In fact, thanks to its robustness and reliability, VF has become the method of choice in design automation tools and flows available on the market, in particular for electronic system design applications.

The proposed algorithm for symmetric transfer function fitting is summarized as the pseudocode in Algorithm 11.1.

Algorithm 11.1 Approximation of GFRFs.

Require: M, K_m, ν and the data set $\mathcal{H}$ as defined in (11.4)
  Fit rational function $H_1(s)$ in pole-residue form to data $\check{H}_1(s_1^{(k)})$ using VF
  Turn pole-residue form of $H_1(s)$ into a realization (A_1, B_1, C_1)
  Compute $Q_2(\mathbf{s}_2^{(k)})$ for $k = 1, \ldots, K_2$, as defined in (11.9)
  for $m = 2, \ldots, M$ **do**
    Fit $F_m(s)$ in pole-residue form by solving (11.18) using weighted VF
    Turn pole-residue form of $F_m(s)$ into a realization (A_m, N_m, C_m)
    Compute $Q_{m+1}(\mathbf{s}_{m+1}^{(k)})$ for $k = 1, \ldots, K_{m+1}$ as defined in (11.9)
  end for

3 Numerical examples and discussion

3.1 Viscous Burgers' equation

This section reports on the example of the viscous Burgers' equation

$$\frac{\partial}{\partial t} v(x,t) - \nu \frac{\partial^2}{\partial x^2} v(x,t) + v(x,t) \frac{\partial}{\partial x} v(x,t) = 0, \tag{11.22}$$

describing the velocity $v(x,t)$ of a fluid in one spatial coordinate. The viscosity parameter was set to $\nu = 0.02$. By following [6], the PDE was semi-discretized in space through a finite difference scheme with 200 points. The resulting nonlinear dynamical system describing the temporal evolution was formulated as a quadratic-bilinear system of size $q = 200$.

A data-driven bilinear model with degree $M = 2$ could be constructed by sampling the first- and second-degree transfer functions $H_1(s_1)$ and $H_2(s_1, s_2)$. In particular, $H_1(s_1)$ was sampled on the imaginary axis at $K_1 = 200$ points, log-spaced and corresponding to the frequency band $[10^{-5}, 800]$ Hz. The function $H_2(s_1, s_2)$ was sampled at $K_2 = 4964$ points, with both s_1 and s_2 purely imaginary and in the frequency interval $[10^{-5}, 400]$ Hz. Running the proposed algorithm with $\nu = 8$ poles, 40 iterations, and $M = 2$ to approximate H_1 and H_2 leads to a fitted bilinear model with overall order 13. Frequency-domain accuracy is reported in Figure 11.1, where the left panel shows the relative error with respect to both s_1 and s_2, and the right panel directly compares the responses.

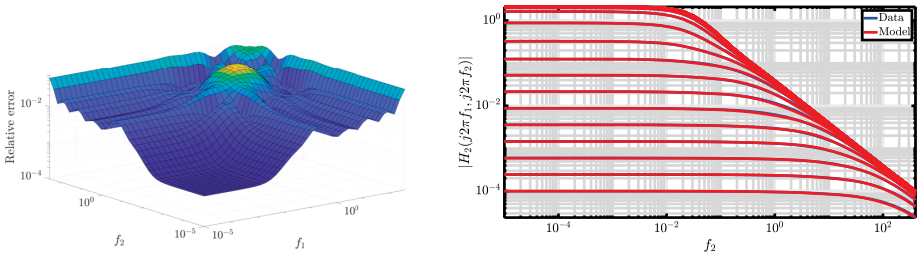


Figure 11.1: Error analysis for the example in Section 3.1. *Left panel*: Relative error on H_2 for both s_1 and s_2 on the imaginary axis. *Right panel*: Model-data comparison for $H_2(j2\pi f_1, j2\pi f_2)$ with respect to f_2 and for several fixed values of f_1.

In the time domain, the responses of the full-order model are compared with the fitted one in Figure 11.2 (left panel). In this experiment, $t \in [0, 10]$ s and the input is an amplitude-modulated chirp $u(t) = 0.1\cos[2\pi f_0(t)t][1 - 1/2\cos(2\pi 2t)]$, with $f_0(t)$ sweeping from 1 mHz to 10 Hz, in order to test accuracy over a broad frequency range. In the right panel of Figure 11.2, we also show that the system nonlinearity is being modeled correctly because adding the degree-2 subsystem leads to a substantial error reduction compared to a simple degree-1 approximation (where only H_1 is fitted).

3.2 Nonlinear RC ladder

This example is an electrical network with nonlinear elements, originally introduced in [9]. In particular, it is an RC ladder circuit with diodes in parallel to resistors (see Figure 11.3). The input $u(t)$ is a current source, while the output $y(t)$ is the voltage at the first node.

Diodes have an exponential characteristic so that the original state equation contains nonpolynomial nonlinearities. Therefore, starting from a model with $q = 50$ states, Carleman bilinearization was used to obtain a lifted bilinear model with $q_{\text{bil}} = 50^2 + 50$.

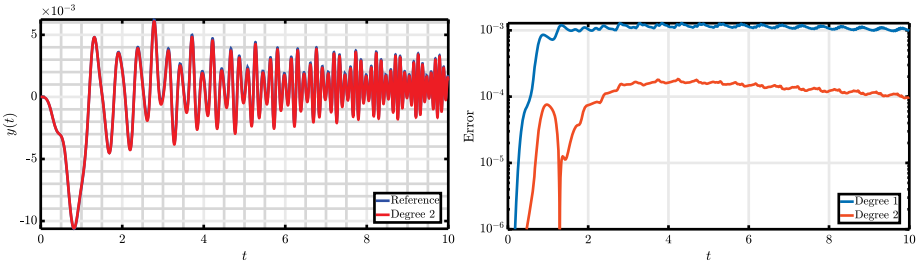


Figure 11.2: Time-domain solution of the example in Section 3.1. The *left panel* compares the full-order model response with that of the fitted model (of degree 2). The *right panel* reports the instantaneous error for two models with $M = 1, 2$, showing that the nonlinear ($M = 2$) model is more accurate than the linear approximation ($M = 1$).

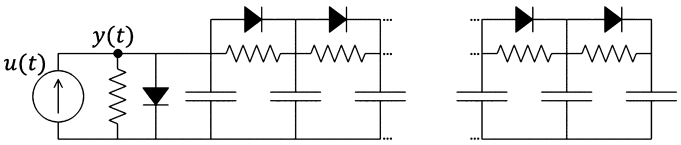


Figure 11.3: Nonlinear electrical network of the example in Section 3.2, picture adapted from [22].

The resulting system is considered the starting point for applying the proposed data-driven modeling scheme.

We sampled $H_1(s_1)$ at $K_1 = 101$ points on the imaginary axis $s_1 = j2\pi f_1$, with $f_1 \in [10^{-1}, 10^2]$ in addition to $s_1^{(1)} = 0$. Regarding higher-order transfer functions, we randomly selected $K_2 = 343$ and $K_3 = 101$ points to sample $H_2(s_1, s_2)$ and $H_3(s_1, s_2, s_3)$.

Using the proposed algorithm with $\nu = 6$ poles and 20 iterations leads to the accurate fitting of the first $M = 3$ frequency responses, as shown in Figure 11.4 for $m = 2$. In time-domain, the model was tested in a 2-s long simulation with amplitude-modulated chirp input $u(t) = 2 \cdot \sin[2\pi f_0(t)t + \pi/2][1 - \frac{1}{2}\cos(2\pi t)]$, with $f_0(t)$ varying linearly from 0.5 Hz to 50 Hz.

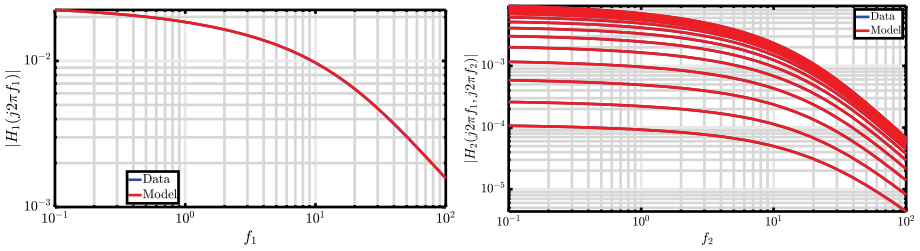


Figure 11.4: Error analysis for the example in Section 3.2. *Left panel*: Model-data comparison for the degree-1 transfer function. *Right panel*: Degree-2 transfer function with respect to f_2 for several fixed values of f_1.

Figure 11.5 compares the original and reduced model responses, showing that adding higher-order transfer functions decreases the error and that the degree-3 model is sufficient to produce a response that is indistinguishable from the full-order model for this case. Note that the error curves depicted in Figure 11.5 are related to the time-domain simulation used for testing that involves an oscillatory signal. The input has here been chosen to push the system into a regime of nonlinear operation where the contribution from the higher-order transfer functions can be appreciated and, in particular, models with $M = 1, 2$ are insufficient, as shown in Figure 11.6.

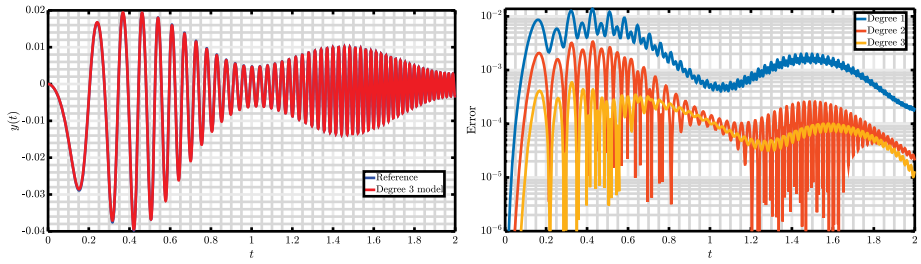


Figure 11.5: Time-domain solution for the example in Section 3.2. *Left panel*: Comparison of full-order and fitted model of degree $M = 3$. *Right panel*: Evolution of instantaneous error with respect to model degree.

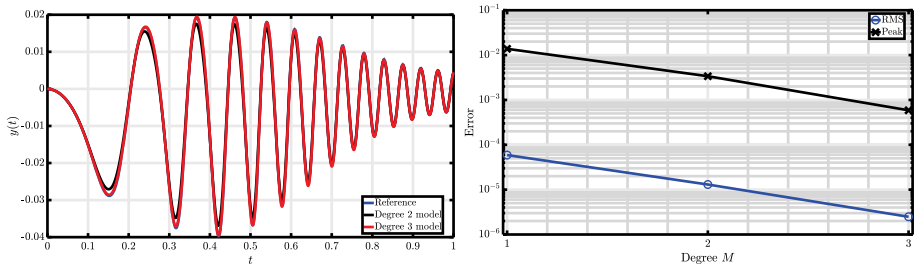


Figure 11.6: *Left panel*: Time-domain response of example in Section 3.2, highlighting the qualitative difference between models with $M = 2$ and $M = 3$. *Right panel*: Error in the time-domain solution of Figure 11.5, measured as root mean square (RMS) and peak error (max. in $t \in [0, 2]$).

4 Conclusions

To summarize, this work described a new algorithm for data-driven approximation of nonlinear input–output maps in the frequency domain, based on Volterra series. The method uses samples of the symmetric transfer functions, directly measurable from response evaluation, without requiring access to the first-principles description of the system to be modeled. Compared to previous work [16], this framework is flexible as it can

approximate GFRFs beyond the second degree. The entire procedure is fast and scalable as it consists of a sequence of linear least-squares optimization problems, thanks to a greedy approach that provides simplicity in exchange for exact optimality. Future work could address this issue, by investigating extensions of this algorithm in which transfer functions are fitted simultaneously rather than sequentially.

Bibliography

[1] A. C. Antoulas, I. V. Gosea, and A. C. Ionita. Model reduction of bilinear systems in the Loewner framework. *SIAM Journal on Scientific Computing*, 38(5):B889–B916, 2016.

[2] P. Benner and T. Breiten. Interpolation-based $\mathcal{H}_2$-model reduction of bilinear control systems. *SIAM Journal on Matrix Analysis and Applications*, 33(3):859–885, 2012.

[3] P. Benner and T. Damm. Lyapunov equations, energy functionals, and model order reduction of bilinear and stochastic systems. *SIAM Journal on Control and Optimization*, 49(2):686–711, 2011.

[4] P. Benner, S. Gugercin, and S. W. R. Werner. Structure-preserving interpolation of bilinear control systems. *Advances in Computational Mathematics*, 47(3):43, 2021.

[5] S. Boyd, L. O. Chua, and C. A. Desoer. Analytical foundations of Volterra series. *IMA Journal of Mathematical Control and Information*, 1(3):243–282, 1984.

[6] T. Breiten and T. Damm. Krylov subspace methods for model order reduction of bilinear control systems. *Systems & Control Letters*, 59(8):443–450, 2010.

[7] T. Carleman. Application de la théorie des équations intégrales linéaires aux systèmes d'équations différentielles non linéaires. *Acta Mathematica*, 59:63–87, 1932.

[8] A. Carlucci, T. Bradde, and S. Grivet-Talocia. Addressing load sensitivity of rational macromodels. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 13(10):1591–1602, 2023.

[9] Y. Chen. Model reduction for nonlinear systems. Master's thesis, Massachusetts Institute of Technology, 1999.

[10] J. Debnath and R. S. Dahiya. Theorems on multidimensional Laplace transform for solution of boundary value problems. *Computers & Mathematics with Applications*, 18(12):1033–1056, 1989.

[11] D. Deschrijver, M. Mrozowski, T. Dhaene, and D. De Zutter. Macromodeling of multiport systems using a fast implementation of the vector fitting method. *IEEE Microwave and Wireless Components Letters*, 18(6):383–385, 2008.

[12] W. Favoreel, B. De Moor, and P. Van Overschee. Subspace identification of bilinear systems subject to white inputs. *IEEE Transactions on Automatic Control*, 44(6):1157–1165, 1999.

[13] G. Flagg and S. Gugercin. Multipoint Volterra series interpolation and $\mathcal{H}_2$ optimal model reduction of bilinear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(2):549–579, 2015.

[14] S. Grivet-Talocia and B. Gustavsen. *Passive Macromodeling: Theory and Applications*. Wiley Series in Microwave and Optical Engineering. Wiley, 2015.

[15] B. Gustavsen and A. Semlyen. Rational approximation of frequency domain responses by vector fitting. *IEEE Transactions on Power Delivery*, 14(3):1052–1061, 1999.

[16] D. S. Karachalios, I. V. Gosea, and A. C. Antoulas. On bilinear time-domain identification and reduction in the Loewner framework. In: *Model Reduction of Complex Dynamical Systems*. International Series of Numerical Mathematics, volume 171, pages 3–30. Birkhäuser, Cham, 2021.

[17] D. S. Karachalios, I. V. Gosea, L. Gkimisis, and A. C. Antoulas. Data-driven quadratic modeling in the loewner framework from input-output time-domain measurements. *SIAM Journal on Applied Dynamical Systems*, 24(1):457–500, 2025.

[18] S. Lefteriu and A. C. Antoulas. On the convergence of the vector-fitting algorithm. *IEEE Transactions on Microwave Theory and Techniques*, 61(4):1435–1443, 2013.

[19] Y. Nakatsukasa, O. Sète, and L. N. Trefethen. The AAA algorithm for rational approximation. *SIAM Journal on Scientific Computing*, 40(3):A1494–A1522, 2018.
[20] W. J. Rugh. *Nonlinear System Theory: The Volterra/Wiener Approach*. Johns Hopkins Series in Information Sciences and Systems. Johns Hopkins University Press, 1981.
[21] M. Schetzen. *The Volterra and Wiener Theories of Nonlinear Systems*. John Wiley & Sons Inc, 1980. ISBN-10: 0471044555.
[22] The MORwiki Community. Nonlinear RC Ladder. MORwiki—Model Order Reduction Wiki, 2018.
[23] L. Zhang and J. Lam. On H_2 model reduction of bilinear systems. *Automatica*, 38(2):205–216, 2002.

Ganna Shyshkanova, Timo Kreimeier, Lukas Baumgärtner, Franz Bethke, and Andrea Walther

On the nonsmooth regularity condition LIKQ for different abs-normal representations

Abstract: Nonsmoothness is widely present in practical optimization problems for physical, technological, methodological, or numerical reasons, for example, due to modeling of valves or regularization with the ℓ^1-norm. Many of these optimization problems can be described by a composition of smooth functions and the evaluations of the absolute value. Over the years, the precise definition of the resulting class of functions evolved also due to an improved understanding of such functions. This work analyzes the different representations of an element of this function class and the reformulation from one version to another. It is proved that the linear independent kink qualification (LIKQ) as one regularity condition needed for optimality conditions is preserved. Illustrative examples demonstrate comprehension and application of the results, elucidating the theoretical material.

Keywords: Nonsmooth optimization, abs-smooth function, linear independence kink qualification, abs-normal formulation, active switching variables

MSC 2020: 49J52, 90C30, 90C56

1 Introduction

Nonsmooth phenomena in physics, technology, and optimization are ubiquitous, manifesting naturally and frequently across a multitude of problem domains. These phenomena characterized, for example, by sharp edges in functions or objective surfaces, challenge conventional smooth optimization techniques and demand specialized methodologies for effective solution. Consequently, there arises a pressing need

Acknowledgement: The authors thank the DFG for support within project B10 in the TRR 154 *Mathematical Modelling, Simulation, and Optimization using the Example of Gas Networks* (project ID: 239904186). The research was funded also partly by the DFG under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID:390685689).
Conflicts of Interest: The authors declare no conflict of interest.

Ganna Shyshkanova, Timo Kreimeier, Lukas Baumgärtner, Franz Bethke, Andrea Walther,
Humboldt-Universität zu Berlin, Institut für Mathematik, 6 Unter den Linden, 10099 Berlin, Germany,
e-mails: ganna.shyshkanova@hu-berlin.de, lukas.baumgaertner@hu-berlin.de, franz.bethke@hu-berlin.de, andrea.walther@math.hu-berlin.de

https://doi.org/10.1515/9783111376776-012

for proficient tools and algorithms tailored to navigate the complexities inherent in nonsmooth optimization landscapes, ensuring the development of robust and efficient solutions to pertinent problems. Rockafellar's book [13] is one seminal work in this area, introducing subdifferentials and optimality conditions. More subdifferentials and quasidifferentials along with corresponding optimality conditions were introduced, for example, by Clarke [1] or Mordukhovich [12] in the years following. However, in a numerical algorithm, these generalized derivatives and optimality conditions are quite often intractable, since the set of generalized derivatives cannot be computed or inclusions cannot be verified with an acceptable computational effort.

A few years ago, Griewank and Walther examined unconstrained finite-dimensional nonlinear optimization problems, where the nonsmoothness is caused by possibly nested occurrences of the absolute value function [6]. For example, any piecewise smooth function that is specified by an evaluation procedure involving only smooth elemental functions and piecewise linear functions like min and max fits into this setting. Then the derivative information required by an optimization algorithms can be practically computed for this class of functions using appropriately extended Algorithmic Differentiation (AD) tools like ADOL-C [5]. In [6], necessary and sufficient optimality conditions were derived that can be verified in polynomial time in contrast to both the Clarke and the Mordukhovich necessary condition, provided the regularity condition LIKQ, which is similar to the linear independence constraint qualification (LICQ) in the smooth case, holds. For this purpose, only classical smooth derivatives have to be computed.

The theory of Griewank and Walther is extended to a certain class of nonsmooth nonlinear programs in [7]. One of the key findings, also interesting for the present work, was that LIKQ was preserved under the reformulation proposed in [7] such that the results for the unconstrained case could be transferred. Hegerhorst-Schultchen and Steinbach have shown that the considered class of nonsmooth nonlinear programs is equivalent to the class of Mathematical Programs with Equilibrium Constraints (MPECs) [8, 7]. Hence, the regularity condition LIKQ turns out to be equivalent to MPEC-LICQ.

The compositions of smooth elemental functions and the absolute value function $\mathrm{abs}(x) = |x|$ are widely used in practice. For example, $\max(0, x)$ is commonly known as rectified linear unit (ReLU) function in the machine learning community [2]. Owing to its ease of training and propensity for yielding superior performance metrics, ReLU is used within many neural network architectures [15].

The paper is structured as follows: In Section 2, the development of the abs-smooth form is illustrated highlighting the differences of three different formulations. Section 3 analyzes the transformation of these different function representations into each other. The preservation of LIKQ for the different versions is studied in Section 4. Finally, Section 5 provides a short summary and outlook.

2 Development of different abs-smooth forms

Throughout, we consider the minimization problems of the form

$$\min_{x\in\mathbb{R}^n} \varphi(x) \tag{12.1}$$

for a function $\varphi\colon \mathbb{R}^n \to \mathbb{R}, y = \varphi(x)$ that exhibit so-called Level-1 nonsmoothness. Namely, the objective function φ is defined as a composition of smooth elemental functions and the absolute value function $\mathrm{abs}(x) = |x|$ as introduced in [6]. From an AD perspective, any such function can be stated as shown in Table 12.1. Here, all consecutive smooth elemental functions are combined into larger smooth elemental functions $\psi_i(.)$ such that all evaluations of the absolute value function can be clearly identified and exploited. The number of absolute values occurring during the function evaluation will be denoted by $s \in \mathbb{N}$. The arguments $z_i = z_i(x)$ of the absolute value functions cause the switches in the corresponding derivative values and therefore, $z = (z_i) \in \mathbb{R}^s$ is called switching vector. The values of z always depend on the value of x. However, for notational simplicity, we skip this dependence in the notation. The switching vector defines the signature vector $\sigma = (\sigma_i) = (\mathrm{sign}(z_i(x))) \in \mathbb{R}^s$.

Table 12.1: Evaluation procedure including absolute value evaluations.

$v_{i-n} = x_i$		$i = 1 \ldots n$
$z_i = \psi_i(v_j)_{j<i}$	}	$i = 1 \ldots s$
$\sigma_i = sign(z_i)$		
$v_i = \sigma_i z_i = abs(z_i)$		
$y = \psi_{s+1}(v_j)_{j<s+1}$		

An evaluation procedure as shown in Table 12.1, where z_j can only influence z_i if $j < i$, was the starting point to derive abs-smooth forms. The calculation of derivative information for such an extended evaluation procedure in the sense of AD was analyzed in [3]. However, to build mathematical theory on top of it, an evaluation procedure is not a beneficial representation of a function. Therefore, in [4, 6], the so-called *abs-normal form* was introduced covering all functions that can be evaluated by a pseudocode as given in Table 12.1. For that purpose, the calculation of all switching variables is formulated as an equality constraint by means of a function F. Furthermore, the vector of the absolute values of the switching variables is introduced as an extra argument of the target function f and the constraining function F. Thus, we obtain the following representation:

$$y = \varphi(x) = f\big(x, |z|\big), \tag{12.2a}$$

$$z = F\big(x, |z|\big), \tag{12.2b}$$

where $f\colon \mathbb{R}^{n+s} \to \mathbb{R}$ and $F\colon \mathbb{R}^{n+s} \to \mathbb{R}^s$ are at least once differentiable in the region of interest. Here and throughout, $|z|$ denotes the componentwise modulus of a vector z.

Furthermore, sometimes $|z|$ and similar vectors of absolute values are considered as variable vectors in their own right. With this reformulation, the unconstrained problem in (12.1) can be stated as a constrained optimization problem with the equality constraints given by (12.2b).

As mentioned already above, Equation (12.2) was called abs-normal form in earlier papers like [4, 6]. To unify the notation, we will call such a representation of φ now *abs-smooth form of type* (12.2).

Since Equation (12.2b) defines the arguments of the absolute value function and hence also causes nonsmoothness, this equation is called *switching equation*. Motivated by the evaluation procedure, it is assumed that for the calculation of z_i only values z_j, $1 \leq j < i$, are needed. Hence, z is uniquely determined by Equation (12.2b). Furthermore, we assume that all components of z are used as arguments of the absolute value function, that is, they are all used for the calculation of the objective function, such that they indeed cause nonsmoothness. If this assumption does not hold, the corresponding component of z can be eliminated yielding a reduced function representation without further impact.

Depending on the specific application at hand it might be a bit restrictive to allow only absolute values of the vector z as argument of F and f, respectively. An explicit dependence on z might also lead to larger but much sparser derivative matrices. For a discussion of this effect see, for example, [9]. Therefore, an alternative to the abs-smooth form of type (12.2) is given by

$$y = \varphi(x) = \hat{f}(x, \hat{z}, |\hat{z}|), \tag{12.3a}$$

$$\hat{z} = \hat{F}(x, \hat{z}, |\hat{z}|), \tag{12.3b}$$

where $\hat{f}\colon \mathbb{R}^{n+s+s} \to \mathbb{R}$ and $\hat{F}\colon \mathbb{R}^{n+s+s} \to \mathbb{R}^s$ are again at least once differentiable in the region of interest. As above, we assume that $\hat{z}_i$ only depends on values $\hat{z}_j$, $1 \leq j < i$, such that $\hat{z}$ is uniquely determined by Equation (12.3b) and that all $\hat{z}_i$ serve as argument of an absolute value evaluation that is used to calculate the value of the target function. We will call this representation of an element of the function class considered in this paper *abs-smooth form of type* (12.3).

As it turned out in the theoretical analysis, it might be advantageous to allow the absolute value function only in the switching equation. This setting was considered, for example, in [8, 11] yielding the representation

$$y = \varphi(x) = \tilde{f}(x, \tilde{z}), \tag{12.4a}$$

$$\tilde{z} = \tilde{F}(x, \tilde{z}, |\tilde{z}|), \tag{12.4b}$$

where $\tilde{f}\colon \mathbb{R}^{n+(s+1)} \to \mathbb{R}$ and $\tilde{F}\colon \mathbb{R}^{n+(s+1)+(s+1)} \to \mathbb{R}^{s+1}$ are again at least once differentiable in the region of interest. Once more, we assume that $\tilde{z}$ is uniquely determined by Equation (12.4b) in that for computing $\tilde{z}_i$ only values $\tilde{z}_j$, $1 \leq j < i$ are used. We will call this representation of an element of the function class considered in this paper *abs-smooth form of type* (12.4). It is important to note that for the abs-smooth form of type (12.4), the last

element of $\tilde{z}$, that is, $\tilde{z}_{s+1}$, is not used as an argument of the absolute value. Hence, it does not cause any nonsmoothness. This observation also motivates our notation $\tilde{z} = (\hat{z}, \tilde{z}_{s+1})$, where all components of $\hat{z} \in \mathbb{R}^s$ are assumed to be used as arguments of the absolute value similar to the abs-smooth forms of type (12.2) and (12.3).

To illustrate our later results, we consider the following target function that can be seen as prototype of a very simple neural network.

Example 12.1. Let $n \coloneqq 4$, $s \coloneqq 2$. For $x \in \mathbb{R}^4$ and $\tilde{z} \in \mathbb{R}^{2+1}$, define

$$\tilde{F}(x, \tilde{z}, |\tilde{z}|) \coloneqq \begin{bmatrix} 2x_1 + x_2 \\ \frac{x_3}{2}(|\tilde{z}_1| + \tilde{z}_1) + x_4 - 2 \\ |\tilde{z}_2| \end{bmatrix},$$
$$\tilde{f}(x, \tilde{z}) \coloneqq \tilde{z}_3.$$

For $\tilde{z}$ as a solution to the switching equation (12.4b), that is, $\tilde{z} = \tilde{F}(x, \tilde{z}, |\tilde{z}|)$, using

$$\mathrm{ReLU}(w) = \max(w, 0) = \big(|w| + w\big)/2$$

yields

$$\begin{aligned} \tilde{f}(x, \tilde{z}) &= \left| \frac{x_3}{2}\big(|\tilde{z}_1| + \tilde{z}_1\big) + x_4 - 2 \right| \\ &= \big|x_3\,\mathrm{ReLU}(\tilde{z}_1) + x_4 - 2\big| \\ &= \big|x_3\,\mathrm{ReLU}(2x_1 + x_2) + x_4 - 2\big| \end{aligned}$$

Thus, the functions $\tilde{f}$ and $\tilde{F}$ represent $\varphi(x) = |x_3\,\mathrm{ReLU}(2x_1 + x_2) + x_4 - 2|$.

3 Transitioning between abs-smooth forms

As described above, so far there are three possible different representations for a function that is given by a composition of smooth elemental functions and absolute value evaluations: the abs-smooth form of type (12.2), the abs-smooth form of type (12.3), and the abs-smooth form of type (12.4); which form is beneficial may depend on the specific application at hand. Therefore, an obvious question is whether it is always possible to transfer one representation into the other ideally preserving the regularity condition LIKQ. Obviously, neither of the representations is unique, a fact that we state here for completeness.

Proposition 12.2. *For any two functions f and F in the abs-smooth form of type* (12.2)*, there are functions $\hat{f}$ and $\hat{F}$ in the abs-smooth form of type* (12.3) *which are equivalent in the following sense:*

a) *The solutions z and $\hat{z}$ to their respective switching equations coincide.*
b) *Both forms represent the same nonsmooth function φ.*

The same holds true for the converse direction of reformulation.

Proof. Let f and F be functions in the abs-smooth form of type (12.2) representing some φ. For $x \in \mathbb{R}^n$ and $\hat{z} \in \mathbb{R}^s$, define

$$\begin{aligned} \hat{f}(x, \hat{z}, |\hat{z}|) &\coloneqq f(x, |\hat{z}|), \\ \hat{F}(x, \hat{z}, |\hat{z}|) &\coloneqq F(x, |\hat{z}|), \end{aligned} \tag{12.5}$$

that is, by ignoring the second input. Now, for a given $x \in \mathbb{R}^n$, let $z \in \mathbb{R}^s$ be determined by the switching equation (12.2b). By the definition of $\hat{F}$, we have

$$z = F(x, |z|) = \hat{F}(x, z, |z|),$$

that is, z is equal to the unique solution $\hat{z} \in \mathbb{R}^s$ of the switching equation (12.3b). Thus, $\varphi(x) = f(x, |z|) = \hat{f}(x, z, |z|)$ showing that $\hat{f}$ and $\hat{F}$ also represent φ.

Let $\hat{f}$ and $\hat{F}$ be functions in the abs-smooth form of type (12.3) representing some φ. To obtain a representation in the abs-smooth form of type (12.2), we observe that $\hat{z}_i = \hat{F}_i(x, \hat{z}, |\hat{z}|)$ does only depend on the values of $\hat{z}_1, \ldots, \hat{z}_{i-1}$. Thus, omitting these spurious dependencies in the notation, for $z \in \mathbb{R}^s$ we define

$$\begin{aligned} F_1(x, |z|) &\coloneqq \hat{F}_1(x), \\ F_i(x, |z|) &\coloneqq \hat{F}_i(x, F_1(x, |z|), \ldots, F_{i-1}(x, |z|), |z_1|, \ldots, |z_{i-1}|), \end{aligned} \tag{12.6}$$

for $i = 2, \ldots, s$ and

$$f(x, |z|) \coloneqq \hat{f}(x, F(x, |z|), |z|). \tag{12.7}$$

Let again $x \in \mathbb{R}^n$ be given and this time let $\hat{z} \in \mathbb{R}^s$ denote the solution to the switching equation (12.3b). Then, with $\hat{z}_1 = \hat{F}_1(x) = F_1(x, |\hat{z}|)$, induction yields

$$\begin{aligned} \hat{z}_i &= \hat{F}_i(x, \hat{z}_1, \ldots, \hat{z}_{i-1}, |\hat{z}_1|, \ldots, |\hat{z}_{i-1}|) \\ &= \hat{F}_i(x, F_1(x, |\hat{z}|), \ldots, F_{i-1}(x, |\hat{z}|), |\hat{z}_1|, \ldots, |\hat{z}_{i-1}|) \\ &= F_i(x, |\hat{z}|). \end{aligned}$$

Thus, $\hat{z}$ is equal to the unique solution $z \in \mathbb{R}^s$ to the switching equation (12.2b), and hence, $\varphi(x) = \hat{f}(x, \hat{z}, |\hat{z}|) = \hat{f}(x, F(x, |\hat{z}|), |\hat{z}|) = f(x, |\hat{z}|)$. □

Proposition 12.3. *For any two functions $\hat{f}$ and $\hat{F}$ in the abs-smooth form of type* (12.3)*, there are functions $\tilde{f}$ and $\tilde{F}$ in the abs-smooth form of type* (12.4) *which are equivalent in the following sense:*

a) *The solution $\hat{z}$ of the switching equation* (12.3b) *coincides with the first s components of the solution $\tilde{z}$ of the switching equation* (12.4b).
b) *Both forms represent the same nonsmooth function φ.*

The same holds true for the converse direction of reformulation.

Proof. Let $\hat{f}$ and $\hat{F}$ be functions in the abs-smooth form of type (12.3) representing some φ. For $x \in \mathbb{R}^n$ and $\tilde{z} \in \mathbb{R}^{s+1}$, we define

$$\begin{aligned} \tilde{F}(x, \tilde{z}, |\tilde{z}|) &:= \begin{bmatrix} \hat{F}(x, \tilde{z}_1, \ldots, \tilde{z}_s, |\tilde{z}_1|, \ldots, |\tilde{z}_s|) \\ \hat{f}(x, \tilde{z}_1, \ldots, \tilde{z}_s, |\tilde{z}_1|, \ldots, |\tilde{z}_s|) \end{bmatrix}, \\ \tilde{f}(x, \tilde{z}) &:= \tilde{z}_{s+1}. \end{aligned} \tag{12.8}$$

Now, for a given $x \in \mathbb{R}^n$ let $\tilde{z} := (\hat{z}, y) \in \mathbb{R}^{s+1}$ be the solution to (12.3), then

$$\tilde{z} = \begin{bmatrix} \hat{z} \\ y \end{bmatrix} = \begin{bmatrix} \hat{F}(x, \hat{z}, |\hat{z}|) \\ \hat{f}(x, \hat{z}, |\hat{z}|) \end{bmatrix} = \tilde{F}(x, \tilde{z}, |\tilde{z}|).$$

Thereby, it holds $\varphi(x) = y = \tilde{z}_{s+1} = \tilde{f}(x, \tilde{z})$.

Let now $\tilde{f}$ and $\tilde{F}$ be functions in the abs-smooth form of type (12.4) representing some φ. Recall that due to the switching structure, the value $\tilde{F}(x, \tilde{z}, |\tilde{z}|)$ does not depend on the argument $\tilde{z}_{s+1}$. Thus, for any $x \in \mathbb{R}^n$, $\hat{z} \in \mathbb{R}^s$ define

$$\begin{aligned} \hat{F}_i(x, \hat{z}, |\hat{z}|) &:= \tilde{F}_i(x, (\hat{z}, 0), |(\hat{z}, 0)|) \quad i = 1, \ldots, s, \\ \hat{f}(x, \hat{z}, |\hat{z}|) &:= \tilde{f}(x, \tilde{F}(x, (\hat{z}, 0), |(\hat{z}, 0)|)) \\ &= \tilde{f}(x, \tilde{F}(x, (\hat{z}, \tilde{z}_{s+1}), |(\hat{z}, \tilde{z}_{s+1})|)) \quad \text{for all } \tilde{z}_{s+1} \in \mathbb{R}. \end{aligned} \tag{12.9}$$

For a given $x \in \mathbb{R}^n$, let $\tilde{z} := (\hat{z}, \tilde{z}_{s+1})$ be a solution to the switching equation (12.4b). Then

$$\begin{bmatrix} \hat{z} \\ \tilde{z}_{s+1} \end{bmatrix} = \tilde{z} = \tilde{F}(x, \tilde{z}, |\tilde{z}|) = \begin{bmatrix} \hat{F}(x, \hat{z}, |\hat{z}|) \\ \tilde{F}_{s+1}(x, \hat{z}, |\hat{z}|) \end{bmatrix}.$$

Therefore, $\hat{z}$ is a solution to the switching equation in (12.3b) for $\hat{F}$. Furthermore, $\varphi(x) = \tilde{f}(x, \tilde{z}) = \tilde{f}(x, \tilde{F}(x, \tilde{z}, |\tilde{z}|)) = \hat{f}(x, \hat{z}, |\hat{z}|)$, where the last equality is due to the definition in (12.9). □

Example 12.4. Recall the abs-smooth form of type (12.4) given in Example 12.1:

$$\begin{aligned} \tilde{F}(x, \tilde{z}, |\tilde{z}|) &= \begin{bmatrix} 2x_1 + x_2 \\ \frac{x_3}{2}(|\tilde{z}_1| + \tilde{z}_1) + x_4 - 2 \\ |\tilde{z}_2| \end{bmatrix}, \\ \tilde{f}(x, \tilde{z}) &= \tilde{z}_3. \end{aligned}$$

We find the abs-smooth form of type (12.3) by using Equation (12.9) in Proposition 12.3. This yields

$$\hat{F}(x,\hat{z},|\hat{z}|) = \begin{bmatrix} 2x_1 + x_2 \\ \frac{x_3}{2}(|\hat{z}_1| + \hat{z}_1) + x_4 - 2 \end{bmatrix},$$
$$\hat{f}(x,\hat{z},|\hat{z}|) = |\hat{z}_2|.$$

From this, we obtain the abs-smooth form of type (12.2) by using Equation (12.6) and Equation (12.7) in Proposition 12.2. This yields

$$F(x,|z|) = \begin{bmatrix} \hat{F}_1(x) \\ \hat{F}_2(x, F_1(x,|z|),|z_1|) \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 \\ \frac{x_3}{2}(|z_1| + 2x_1 + x_2) + x_4 - 2 \end{bmatrix},$$
$$f(x,|z|) = \hat{f}(x, F(x,|z|),|z|) = |z_2|.$$

4 Equivalence of the LIKQ conditions

For this section, fix $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^s$, $\hat{z} \in \mathbb{R}^s$ and $\tilde{z} \in \mathbb{R}^{s+1}$ as the solutions to the respective switching equations (12.2b), (12.3b), and (12.4b). In the smooth case, the regularity requirement LICQ poses a condition on the active constraints resulting in a uniqueness of the Lagrange multipliers. Exactly this uniqueness is needed to derive optimality conditions that can be verified in polynomial time for the nonsmooth case. For this purpose, we now specify the active constraints in the nonsmooth setting considered here.

Definition 12.5 (Signature vector and active switching variables). For a given switching vector $z \in \mathbb{R}^s$, the vector $\sigma \coloneqq \operatorname{sign}(z) \in \{-1,0,1\}^s$ is called *a signature vector*. A *switching variable* z_i is called *active* if $\sigma_i = 0$, for $i \in \{1,\dots,s\}$. The *active switching set* α collects all indices of active switching variables, that is,

$$\alpha \coloneqq \{i \in \{1,\dots,s\} \mid \sigma_i = 0\}. \tag{12.10}$$

The size of the active switching set is defined as $|\alpha|$.

It is important to note that by Proposition 12.2 and Proposition 12.3 we have $z_i = \hat{z}_i = \tilde{z}_i$ for $i \in \{1,\dots,s\}$. Hence, the definitions of σ and α are independent of the specific abs-smooth form that is used. Note that the chosen $x \in \mathbb{R}^n$ also determines a fixed signature vector σ throughout this section.

To state and analyze the LIKQ condition for the three different function representations, it is beneficial to introduce some notation. We abbreviate the partial derivatives of the switching functions F, $\hat{F}$, and $\tilde{F}$ by

$$\begin{aligned} Z &\coloneqq \frac{\partial F}{\partial x}(x,|z|), & M &\coloneqq 0, & L &\coloneqq \frac{\partial F}{\partial |z|}(x,|z|), \\ \hat{Z} &\coloneqq \frac{\partial \hat{F}}{\partial x}(x,\hat{z},|\hat{z}|), & \hat{M} &\coloneqq \frac{\partial \hat{F}}{\partial \hat{z}}(x,\hat{z},|\hat{z}|), & \hat{L} &\coloneqq \frac{\partial \hat{F}}{\partial |\hat{z}|}(x,\hat{z},|\hat{z}|), \\ \tilde{Z} &\coloneqq \frac{\partial \tilde{F}}{\partial x}(x,\tilde{z},|\tilde{z}|), & \tilde{M} &\coloneqq \frac{\partial \tilde{F}}{\partial \tilde{z}}(x,\tilde{z},|\tilde{z}|), & \tilde{L} &\coloneqq \frac{\partial \tilde{F}}{\partial |\tilde{z}|}(x,\tilde{z},|\tilde{z}|). \end{aligned} \tag{12.11}$$

Note that the derivative matrices $\hat{M}, \tilde{M}, L, \hat{L}$, and $\tilde{L}$ are strictly lower triangular due to the assumption made for all three abs-smooth forms, respectively. This is motivated by the evaluation procedure shown in Table 12.1 as starting point of the nonsmooth approach considered in this paper. Furthermore, this assumption ensures the invertibility of some of the matrices considered below. Alternatively, one could assume a corresponding invertibility directly. However, then instead of the simple solution of a linear system with a triangular system matrix a more involved solution approach has to be applied.

Let $\Sigma \coloneqq \operatorname{diag}(\sigma) \in \mathbb{R}^{s,s}$. We refer to the matrix

$$\nabla z^{\sigma} \coloneqq (I - L\Sigma)^{-1} Z \in \mathbb{R}^{s \times n} \tag{12.12}$$

as the *Jacobian* of type (12.2) for the fixed switching system

$$z^{\sigma} = F(x, \Sigma z^{\sigma}). \tag{12.13}$$

Notice that the inverse in (12.12) is well-defined due to the strictly lower triangular structure of L. The same argument applies for all the inverse matrices that are used throughout this section. The term *Jacobian* as well as the symbol ∇z^{σ} acknowledge the fact that z^{σ} as the solution to (12.13) can be seen as a function of x and its derivative can be obtained by the implicit function theorem. Analogously, one obtains for the abs-smooth form of type (12.3) the Jacobian

$$\nabla \hat{z}^{\sigma} \coloneqq \left(I - \hat{M} - \hat{L}\Sigma\right)^{-1} \hat{Z} \in \mathbb{R}^{s \times n}, \tag{12.14}$$

for the fixed switching system

$$\hat{z}^{\sigma} = \hat{F}(x, \hat{z}^{\sigma}, \Sigma \hat{z}^{\sigma}). \tag{12.15}$$

With $\sigma_{s+1} \coloneqq \operatorname{sign}(\tilde{z}_{s+1}) \in \{-1, 0, 1\}$ and $\tilde{\Sigma} \coloneqq \operatorname{diag}(\sigma, \sigma_{s+1})$, we define the Jacobian for the abs-smooth form of type (12.4) by

$$\left(\nabla \hat{z}^{\sigma}, \nabla \tilde{z}^{\sigma}_{s+1}\right) \coloneqq \nabla \tilde{z}^{\sigma} \coloneqq \left(I - \tilde{M} - \tilde{L}\tilde{\Sigma}\right)^{-1} \tilde{Z} \in \mathbb{R}^{(s+1) \times n}, \tag{12.16}$$

for the fixed switching system

$$\tilde{z}^{\sigma} = \tilde{F}(x, \tilde{z}^{\sigma}, \tilde{\Sigma} \tilde{z}^{\sigma}). \tag{12.17}$$

Then the required regularity condition as introduced in [6, Definition 2] can be formulated as follows.

Definition 12.6 (Linear independence kink qualification (LIKQ)). A switching function F, $\hat{F}$, or $\tilde{F}$ of type (12.2b), (12.3b), or (12.4b), respectively, is said to satisfy the LIKQ condition at x, if the corresponding active Jacobian from (12.12), (12.14), or (12.16), respectively, has full row rank $|\alpha|$, which requires in particular that $|\sigma| \geq s - n$.

Proposition 12.7. *The abs-smooth form of type* (12.2) *and the abs-smooth form of type* (12.3) *have concurrent LIKQ, that is, a switching function F satisfies LIKQ at some $x \in \mathbb{R}^n$, if and only if the corresponding function $\hat{F}$ from Proposition* 12.2 *has LIKQ at x.*

Proof. It is sufficient to show that the Jacobians ∇z^σ and $\nabla \hat{z}^\sigma$ coincide under the reformulations from Proposition 12.2. For $i \in \{1, \dots, s\}$ and $k \in \{1, \dots, n\}$, the definition of F from $\hat{F}$ in Proposition 12.2 implies

$$\begin{aligned}\frac{\partial F_i}{\partial x_k} &= \frac{d}{dx_k}\hat{F}_i\big(x, F_1(x), \dots, F_{i-1}\big(x, |\hat{z}_1|, \dots |\hat{z}_{i-2}|\big), |\hat{z}_1|, \dots, |\hat{z}_{i-1}|\big)\\ &= \frac{\partial \hat{F}_i}{\partial x_k} + \sum_{j=1}^{i-1} \frac{\partial \hat{F}_i}{\partial \hat{z}_j}\frac{\partial F_j}{\partial x_k}.\end{aligned}$$

In matrix representation, this identity reads $Z = \hat{Z} + \hat{M}Z$, such that one has $Z = (I - \hat{M})^{-1}\hat{Z}$. Analogously, one can also show that $L = (I - \hat{M})^{-1}\hat{L}$. From (12.14), it readily follows that

$$\begin{aligned}\nabla \hat{z}^\sigma &= \big(I - \hat{M} - \hat{L}\Sigma\big)^{-1}\hat{Z}\\ &= \big((I - \hat{M})^{-1}(I - \hat{M} - \hat{L}\Sigma)\big)^{-1}(I - \hat{M})^{-1}\hat{Z}\\ &= \big(\big(I - (I - \hat{M})^{-1}\hat{L}\Sigma\big)\big)^{-1}(I - \hat{M})^{-1}\hat{Z}\\ &= (I - L\Sigma)^{-1}Z\\ &= \nabla z^\sigma.\end{aligned}$$

For the converse reformulation, that is, if $\hat{F}$ is defined by F, it holds that

$$\begin{aligned}\hat{Z} &= \frac{\partial \hat{F}}{\partial x} = \frac{\partial F}{\partial x} = Z,\\ \hat{L} &= \frac{\partial \hat{F}}{\partial |z|} = \frac{\partial F}{\partial |z|} = L,\\ \hat{M} &= \frac{\partial \hat{F}}{\partial z} = 0.\end{aligned}$$

Thereby,

$$\nabla \hat{z}^\sigma = \big(I - \hat{M} - \hat{L}\Sigma\big)^{-1}\hat{Z} = (I - L\Sigma)^{-1}Z = \nabla z^\sigma. \qquad \square$$

Proposition 12.8. *The abs-smooth form of type* (12.3) *and the abs-smooth form of type* (12.4) *have concurrent LIKQ, that is, a switching function $\hat{F}$ satisfies LIKQ at some $x \in \mathbb{R}^n$, if and only if the corresponding function $\tilde{F}$ from Proposition* 12.3 *has LIKQ at x.*

Proof. By the reformulations of Proposition 12.3, the partial derivative of $\hat{F}$ and $\tilde{F}$ (see (12.11)) are related by

$$\tilde{Z} = \frac{\partial}{\partial x}\begin{bmatrix}\hat{F}(x, \hat{z}, |\hat{z}|)\\ \hat{f}(x, \hat{z}, |\hat{z}|)\end{bmatrix} = \begin{bmatrix}\hat{Z}\\ \partial_x \hat{f}(x, \hat{z}, |\hat{z}|)\end{bmatrix},$$

$$\tilde{M} = \begin{bmatrix} \hat{M} & 0_{s\times 1} \\ \partial_{\hat{z}} \hat{f}(x, \hat{z}, |\hat{z}|) & 0_{1\times 1} \end{bmatrix},$$
$$\tilde{L} = \begin{bmatrix} \hat{L} & 0_{s\times 1} \\ \partial_{|\hat{z}|} \hat{f}(x, \hat{z}, |\hat{z}|) & 0_{1\times 1} \end{bmatrix}.$$

Thereby, using (12.14) and (12.16), the Jacobians are related by

$$\begin{aligned} \nabla \tilde{z}^\sigma &= \left(I - \tilde{M} - \tilde{L}\tilde{\Sigma}\right)^{-1} \tilde{Z} \\ &= \begin{bmatrix} I - \hat{M} - \hat{L}\Sigma & 0_{s\times 1} \\ -\partial_{\hat{z}} \hat{f} - \partial_{|\hat{z}|} \hat{f}\Sigma & 1 \end{bmatrix}^{-1} \tilde{Z} \\ &= \begin{bmatrix} (I - \hat{M} - \hat{L}\Sigma)^{-1} & 0_{s\times 1} \\ * & 1 \end{bmatrix} \tilde{Z} \\ &= \begin{bmatrix} (I - \hat{M} - \hat{L}\Sigma)^{-1}\hat{Z} \\ * \end{bmatrix} = \begin{bmatrix} \nabla \hat{z}^\sigma \\ * \end{bmatrix}, \end{aligned}$$

where * is some row matrix obtained as a result of the calculations. This last row is not taken into account for this active Jacobian, hence the LIKQ condition is equivalent under this reformulation.

The converse reformulation from the abs-smooth form of type (12.4) to the abs-smooth form of type (12.3) yields

$$\tilde{Z} = \frac{\partial}{\partial x} \begin{bmatrix} \hat{F}(x, \hat{z}, |\hat{z}|) \\ \tilde{F}_{s+1}(x, \hat{z}, |\hat{z}|) \end{bmatrix} = \begin{bmatrix} \hat{Z} \\ \partial_x \tilde{F}_{s+1}(x, \hat{z}, |\hat{z}|) \end{bmatrix},$$
$$\tilde{M} = \begin{bmatrix} \hat{M} & 0_{s\times 1} \\ \partial_{\hat{z}} \tilde{F}_{s+1}(x, \hat{z}, |\hat{z}|) & 0_{1\times 1} \end{bmatrix},$$
$$\tilde{L} = \begin{bmatrix} \hat{L} & 0_{s\times 1} \\ \partial_{|\hat{z}|} \tilde{F}_{s+1}(x, \hat{z}, |\hat{z}|) & 0_{1\times 1} \end{bmatrix}.$$

Similarly, as before, one obtains

$$\nabla \tilde{z}^\sigma = \left(I - \tilde{M} - \tilde{L}\tilde{\Sigma}\right)^{-1} \tilde{Z} = \begin{bmatrix} (I - \hat{M} - \hat{L}\Sigma)^{-1}\hat{Z} \\ * \end{bmatrix} = \begin{bmatrix} \nabla \hat{z}^\sigma \\ * \end{bmatrix},$$

with a different row matrix for *, which is again irrelevant in the active Jacobian. □

Remark 12.9. The results of Proposition 12.7 and Proposition 12.8 could also be obtained by extending Proposition 12.2 and Proposition 12.3 for the fixed switching equations (12.13), (12.15), and (12.17). Afterwards, the implicit function theorem can be used to show that (12.12), (12.14), and (12.16) are three different representations of the Jacobian of $z^\sigma(x)$.

Example 12.10. Recall Example 12.4. We set $x = (1, -2, 1, 2)$ such that $z = \sigma = (0, 0)$, $\alpha = \{1, 2\}$, and thus $|\alpha| = 2$. Additionally, for the abs-smooth form of type (12.4), we get

$\tilde{z}_{s+1} = 0$ and $\sigma_{s+1} = 0$. We compute the matrices from (12.11) for the different abs-smooth forms as

$$Z = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & \frac{1}{2} & 0 & 1 \end{bmatrix}, \quad M = 0, \quad L = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{bmatrix},$$

$$\hat{Z} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \hat{M} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{bmatrix}, \quad \hat{L} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{bmatrix},$$

$$\tilde{Z} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \tilde{M} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \tilde{L} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

As $\Sigma = 0$ and $\tilde{\Sigma} = 0$, we get

$$\nabla z^{\sigma} = (I - 0 - 0)^{-1} Z = Z = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix} \hat{Z} = \left(I - \hat{M}\right)^{-1} \hat{Z} = \nabla \hat{z}^{\sigma}.$$

Similarly, we also see that

$$\nabla \tilde{z}^{\sigma} = \left(I - \tilde{M}\right)^{-1} \tilde{Z} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{Z} = \begin{bmatrix} Z \\ 0 \end{bmatrix}.$$

In particular, as Z has full rank 2 all forms satisfy the LIKQ condition at x.

5 Summary

In this paper, three abs-normal formulations of the nonsmooth optimization problem were considered. It was shown that the three variants, (12.2), (12.3), and (12.4) can be reformulated into each other. The definition of LIKQ, for (12.2) given in [6] and for (12.4) given in [10], was extended for the case of formulation (12.3), which exhibits an explicit dependence on the switching variable z in both f and F. Furthermore, it was proven that the LIKQ condition is invariant under the proposed reformulations. The results were accompanied by an example, which was motivated by neural networks. Future work could consider the weaker kink qualification MFKQ introduced in [14] as well as the LIKQ condition under additional constraints as in [10].

Bibliography

[1] F. H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley and Sons, 1986.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, London, 2017.

[3] A. Griewank. On stable piecewise linearization and generalized algorithmic differentiation. *Optimization Methods & Software*, 28(6):1139–1178, 2013.
[4] A. Griewank, J.-U. Bernt, M. Randons, and T. Streubel. Solving piecewise linear equations in abs-normal form. *Linear Algebra and Its Applications*, 471:500–530, 2015.
[5] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.
[6] A. Griewank and A. Walther. First- and second-order optimality conditions for piecewise smooth objective functions. *Optimization Methods & Software*, 31(5):904–930, 2016.
[7] L. C. Hegerhorst-Schultchen, C. Kirches, and M. C. Steinbach. Relations between abs-normal NLPs and MPCCs. Part 1: strong constraint qualifications. *Journal of Nonsmooth Analysis and Optimization*, 2, Feb. 2021.
[8] L. C. Hegerhorst-Schultchen and M. C. Steinbach. On first and second order optimality conditions for abs-Normal NLP. *Optimization*, 69(12):2629–2656, 2020. https://doi.org/10.1080/02331934.2019.1626386.
[9] A. Kannan, T. Kreimeier, and A. Walther. On solving nonsmooth retail portfolio maximization problems using active signature methods. Tech. Rep. TRR 154 preprint 520. Humboldt-Universität zu Berlin, 2023.
[10] T. Kreimeier. *Solving constrained piecewise linear optimization problems by exploiting the abs-linear approach*. PhD thesis, Humboldt-Universität zu Berlin, 2023.
[11] T. Kreimeier, M. Kuchlbauer, F. Liers, M. Stingl, and A. Walther. Towards the solution of robust gas network optimization problems using the constrained active signature method. In: *International Network Optimization Conference*, 2022.
[12] B. Mordukhovich. *Variational Analysis and Generalized Differentiation I*. Springer-Verlag, Berlin Heidelberg, 2006.
[13] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
[14] A. Walther and A. Griewank. Characterizing and testing subdifferential regularity in piecewise smooth optimization. *SIAM Journal on Optimization*, 29(2):1473–1501, 2019.
[15] D. Zou, Y. Cao, D. Zhou, and Q. Gu. Gradient descent optimizes over-parameterized deep ReLU networks. *Machine Learning*, 109:467–492, 2020.

Philippe L. Toint

Divergence of the ADAM algorithm with fixed-stepsize: a (very) simple example

Abstract: A very simple unidimensional function with Lipschitz continuous gradient is constructed such that the ADAM algorithm with constant stepsize, started from the origin, diverges when applied to minimize this function in the absence of noise on the gradient. Divergence occurs irrespective of the choice of the method parameters.

Keywords: ADAM algorithm, machine learning, deterministic nonconvex optimization

1 Introduction

This short note provides a new explicit example of failure of the ADAM algorithm [3], one of the most popular training methods in machine learning. Given the problem

$$\min_{x\in\mathbb{R}^n} f(x) \tag{13.1}$$

where f is continuously differentiable function from $\mathbb{R}^n$ into $\mathbb{R}$ with the Lipschitz continuous gradient, and a starting iterate x_0, the ADAM sequence of iterates is defined (see [4]), for $i \in \{1,\ldots,n\}$ and $k \geq 0$, by the recurrences

$$[m_k]_i = \beta_1[m_{k-1}]_i + (1-\beta_1)[g_k]_i, \tag{13.2}$$

$$[v_k]_i = \beta_2[v_{k-1}]_i + (1-\beta_2)[g_k]_i^2, \tag{13.3}$$

$$[x_{k+1}]_i = [x_k]_i - \alpha\frac{[m_k]_i}{\sqrt{[v_k]_i}}, \tag{13.4}$$

where $[v]_i$ is the ith component of the vector $v \in \mathbb{R}^n$, m_k is the kth "momentum," x_k is the kth iterate, $g_k = \nabla_x^1 f(x_k)$, $\beta_1 \in [0,1)$ is the momentum parameter and $\beta_2 \in [0,1)$ is the "forgetting" parameter, and $\alpha > 0$ is a (fixed) steplength/learning-rate parameter. The recurrences (13.2) and (13.3) are initialized by setting, for $i \in \{1,\ldots,n\}$, $[m_{-1}]_i = [g_0]_i$ and $[v_{-1}]_i = [g_0]_i^2$, respectively. ADAM is intended to find first-order points for problem (13.1), in the sense that, for each $i \in \{1,\ldots,n\}$, $|[g_k]_i|$ should converge to zero when k tends to infinity. In practice, this algorithm is most often used in a stochastic context

Acknowledgement: Thanks to Satyen Kale, Omri Weinstein, Serge Gratton, and Alena Kopaničáková for interesting exchanges, and to two anonymous referees for their suggestions.

Philippe L. Toint, Namur Center for Complex Systems (naXys), University of Namur, 50 Boulevard de la Meuse, B-5100 Namur, Belgium, e-mail: philippe.toint@unamur.be

https://doi.org/10.1515/9783111376776-013

where the gradient g_k is contaminated by noise (typically resulting from sampling) and has generated a considerable interest in the machine learning community.

Despite its widespread use, difficulties with this algorithm are not new. In the noiseless (deterministic/full batch) case, obstacles for proving convergence were in particular mentioned in [2], essentially pointing out the possibility that second-order terms in the Taylor's expansion of the objective function could not vanish quickly enough. In [4, Theorem 1], an example of nonconvergence on a convex function was produced in the online-learning stochastic context, but this example crucially depends on the nonzero variance of the noise. In a recent discussion at the June 2023 Thematic Einstein Semester on Optimization and Machine Learning in Berlin, it was suggested that, although likely, no explicit example of failure of Adam with fixed stepsize was available for the deterministic case (where the variance is zero). This prompted the author to produce the (very simple) one which is, for the record, detailed in the next section. We note that an again convex but deterministic example had already been provided in the comprehensive analysis of ADAM's behaviour (with decreasing stepsize) detailed in [5] (see Propositions 3.3 and E1). This analysis describes conditions, which delineate a region strictly included in $[0,1)^2$ such that ADAM with parameters β_1 and β_2 chosen in this region generates a diverging sequence on this example. In contrast, the simple example we are about to discuss is nonconvex and applies to the entire $[0,1)^2$, but requires constant stepsize. It can therefore be seen as complementing the analysis of [5].

2 The example

To show that the ADAM algorithm may fail to converge on nonconvex functions with Lipschitz gradient, we will exhibit an example in dimension one, which we construct in two stages. We first define sequences of iterates, together with associated function and gradient values, which remain constant throughout the iterations. We next verify that these sequences may be considered as generated by applying the ADAM algorithm to a nonconvex objective function with Lipschitz gradient. (Since the example is unidimensional, we omit the component indices (i) if what follows.) For $k \geq 0$, let the sequence of function values and gradients be defined by

$$f_k = 0 \quad \text{and} \quad g_k = -1, \tag{13.5}$$

and the sequence of (potential) iterates be defined (from (13.2)–(13.4)) by $x_0 = 0$ and

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k = -1, \tag{13.6}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2 = 1, \tag{13.7}$$

$$x_{k+1} = x_k - \alpha \frac{m_k}{\sqrt{v_k}} = x_k + \alpha, \tag{13.8}$$

for $k \geq 1$, where we used (13.5) to derive the last equality in (13.6) and (13.7). Thus,

$$s_k = x_{k+1} - x_k = \alpha, \tag{13.9}$$

for $k \geq 0$ and $x_k = \alpha k$ tends to infinity. We now show that there exists a (nonconvex) univariate function f_1 defined on $\mathbb{R}^+$ with Lipschitz continuous gradient such that $f_k = f_1(x_k) = 0$ and $g_k = \nabla_x^1 f_1(x_k) = -1$ for all $k \geq 0$. Indeed, a simple Hermite interpolation calculation based of these conditions yields that, for all $t \in [\alpha k, \alpha(k+1)]$,

$$f_1(t) = -(t - \alpha k) + \frac{3}{\alpha}(t - \alpha k)^2 - \frac{2}{\alpha^2}(t - \alpha k)^3. \tag{13.10}$$

We may then define

$$f(t) = \begin{cases} f_1(t) & \text{if } t \geq 0, \\ -t & \text{if } t < 0, \end{cases}$$

so that $f(t)$ is well-defined on the whole of $\mathbb{R}$ has Lipschitz continuous gradient and is such that the ADAM algorithm (13.6)–(13.8) applied on f starting from $x_0 = 0$ generates iterates with $|g_k| = 1$ for all $k \geq 0$. We thus conclude that the ADAM algorithm fails to converge on this particular instance of problem (13.1). A graph of $f(t)$ for $t \in [-1, 10]$, $\alpha = 1$ is shown in Figure 13.1. One also verifies that the Lipschitz constant on the interval $[x_k, x_{k+1}]$ is given by

$$L = \max_{k \geq 0} \sup_{t \in [\alpha k, \alpha(k+1)]} \left| \nabla_t^2 f(t) \right| = \frac{6}{\alpha}.$$

Moreover, defining $T_k(s) = f_k + g_k s$, it results from (13.5), [1, Theorem A.9.2], (13.9) and the inequalities

$$\left| f_{k+1} - T_k(s_k) \right| = s_k \leq \frac{1}{\alpha} s_k^2 \quad \text{and} \quad \left| g_{k+1} - \nabla_s^1 T_k(s) \right| = |-1 + 1| \leq \frac{1}{\alpha} s_k$$

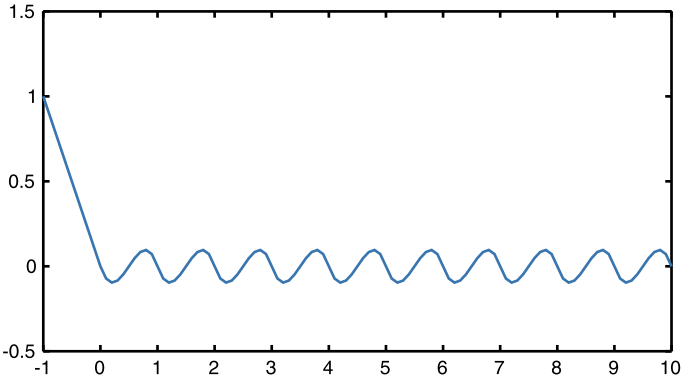


Figure 13.1: The shape of $f(t)$ for small values of $|t|$.

that $f(t)$ is bounded below by a constant only depending on α. As a consequence, we see that, *for any fixed* $(\beta_1, \beta_2) \in [0,1)^2$ *and* $\alpha > 0$*, there exist unidimensional functions with Lipschitz continuous gradient whose gradient's Lipschitz constant is as small as* $6/\alpha$*, which is bounded below by a constant only depending on* α *and for which the ADAM algorithm (*13.6*)–(*13.8*) starting from* $x_0 = 0$ *generates iterates tending to infinity with constant nonzero gradients (therefore failing to converge).*

Since our example is unidimensional and since ADAM is defined componentwise, the same conclusion obviously applies irrespective of n, the problem dimension. Indeed divergence in a single component implies divergence on the whole space.

Our result thus extends that of [5] in that it includes methods for arbitrary $(\beta_1, \beta_2) \in (0,1)^2$ but fixed stepsize. Note that $|\nabla_t^1 f(t)|$ is bounded by L for all $t \in \mathbb{R}$, again at variance with the example of this reference.

Observe that our conclusions would also hold if we had fixed g_k to another negative constant (we can multiply f by this constant) or if, instead of (13.4), we had considered

$$[x_{k+1}]_i = [x_k]_i - \frac{\alpha_k\,[m_k]_i}{\sqrt{\epsilon + [v_k]_i^2}}, \quad \text{or} \quad [x_{k+1}]_i = [x_k]_i - \frac{\alpha_k\,[m_k]_i}{\epsilon + \sqrt{[v_k]_i^2}},$$

where ϵ is a small positive constant and the (now iteration-dependent) stepsizes α_k are bounded away from zero, but they do not apply in the more realistic situation where stepsizes $\alpha_k \to 0$ are used (as is for instance the case in [5, Proposition 1.1], where α_k is a multiple of $1/\sqrt{k}$). We also remark that, because of the periodicity of our example, it would be possible to derive a trigonometric variant. We finally note that we have chosen a constant zero value for f_k in order to simplify our bounds, but that it is also possible to choose $f_{k+1} > f_k$ (leading to an monotonically increasing sequence of function values) without qualitatively affecting our conclusion, although this leads to a larger value of L.

Bibliography

[1] C. Cartis, N. I. M. Gould, and Ph. L. Toint. *Evaluation Complexity of Algorithms for Nonconvex Optimization*. MOS-SIAM Series on Optimization, volume 30. SIAM, Philadelphia, USA, June 2022.

[2] A. Défossez, L. Bottou, F. Bach, and N. Usunier. A simple convergence proof for Adam and Adagrad. *Transactions on Machine Learning Research*, October 2022.

[3] D. Kingma and J. Ba. Adam. A method for stochastic optimization. In: *Proceedings in the International Conference on Learning Representations (ICLR)*, 2015.

[4] S. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. In: *Proceedings in the International Conference on Learning Representations (ICLR)*, 2018.

[5] Y. Zhang, C. Chen, N. Shi, R. Sun, and Z.-Q. Luo. Adam can converge without any modification on update rules. arXiv:2208.09632v5, 2022.

Index

μ-opioid receptor 53

a posteriori error analysis 3
abs-linear form 94
abs-normal form 183
acquisition function 67
Active Signature Method (ASM) 93
ADAM 195
ADAM divergence, example 196
Allen–Cahn equation 86
APTS 107
atherosclerosis 29

bilevel optimal control 18
bilinear system 171, 172
Burgers' equation 169, 175

compact optimization learning 128
Constrained Active Signature Method (CASM) 99
convergence failure 196

data-driven 169, 170, 176, 178
Deep operator network 87
DeepONet 87
derivative-free method 4
design 61
design of experiments 64, 67
diameter rescaling 147
differentiable programming 124
domain decomposition 82, 107
dual optimization proxy 127

embedded 20
empirical risk minimization under constraints 122
extremal 21

failure, of convergence 196
FBPINN 82
finite basis physics-informed neural network 82
forward problem 61
Frank–Wolfe 140
frequency response 169, 170
fully-connected feedforward neural networks 2

Gauss–Newton method 64
Gaussian process 62
gradient rescaling 147
gradient-type method 3
graph 31
graph clustering 31, 34, 38
graph neural networks 29, 32
graph total variation 38
group sparsity 141

Hamilton–Jacobi–Bellman equations 17
Hamiltonian 20
holding probability
– macroscopic 45
– set based 42
hybrid electric vehicle 18

image registration 31
indirect simple shooting method 20
inverse problem 3, 61
ISOKANN 49

k-sparse polytope 141
Karhunen–Loève expansion 11

least-squares 169, 170, 174, 179
Linear Independence Kink Qualification (LIKQ) 100, 189
linear minimization oracle, LMO 140
loss function 81

macro state
– as membership function 45
– conceptual 43
maximum posterior estimate 61
message passing 32
multifidelity model 82
multiscale problem 86

neural networks 3, 18, 81, 107

optical metrology 72
optimal control 17
optimization 108
optimization proxies 123

parametric optimization 122
parametric partial differential equation 1
particle swarm optimization 4
partition of unity 84
pendulum 85

https://doi.org/10.1515/9783111376776-014

physics-informed neural network 81
PINN 81
Pontryagin maximum principle 18
preconditioned 107
primal-dual learning 126

rational approximation 169, 171, 175
reaction path
– conceptual 47
– subsampled 50
reduced model 2

self-supervised learning 122
switching equation 184
symmetric transfer functions 170–172, 178

total variation on graphs 34
training 107
trust-region 107
two-points bundary value problem 21

undirected graph 31

value function 17
vector fitting 169–172, 174
Volterra series 169, 178

wall shear stress 29, 30, 38

De Gruyter Proceedings in Mathematics

Sergey V. Meleshko, Eckart Schulz, Sibusiso Moyo (Eds.)
Analytical Methods in Differential Equations. Conference Proceedings in Honor of Lev V. Ovsiannikov's 105th Birthday Anniversary, 2025
ISBN 978-3-11-156913-0, e-ISBN 978-3-11-157051-8

Manoj Kumar Patel, Triloki Nath, Ram Kishor Pandey, Diwakar Shukla (Eds.)
Advances in Mathematical and Computational Sciences. Proceedings of The ICRTMPCS International Conference 2023, 2024
ISBN 978-3-11-130437-3, e-ISBN 978-3-11-131363-4

Bruce M. Landman, Florian Luca, Melvyn B. Nathanson, Jaroslav Nešetřil, Aaron Robertson (Eds.)
Combinatorial Number Theory. Proceedings of the Integers Conference 2023, 2024
ISBN 978-3-11-139540-1, e-ISBN 978-3-11-139559-3

Oleg V. Kaptsov, Evgeniy I. Kaptsov (Eds.)
Mathematical Models and Integration Methods. Seminar Proceedings on Applications to Mechanics and Physics, 2024
ISBN 978-3-11-154625-4, e-ISBN 978-3-11-154666-7

Toke Knudsen, Jessica Carter (Eds.)
Mastering the History of Pure and Applied Mathematics. Essays in Honor of Jesper Lützen, 2024
ISBN 978-3-11-076990-6, e-ISBN 978-3-11-076996-8

Manoj Kumar Patel, Ratnesh Kumar Mishra, Shiv Datt Kumar (Eds.)
Advances in Pure and Applied Algebra. Proceedings of the CONIAPS XXVII International Conference 2021, 2023
ISBN 978-3-11-078572-2, e-ISBN 978-3-11-078580-7

Richard J. Nowakowski, Bruce M. Landman, Florian Luca, Melvyn B. Nathanson, Jaroslav Nešetřil, Aaron Robertson (Eds.)
Combinatorial Game Theory. A Special Collection in Honor of Elwyn Berlekamp, John H. Conway and Richard K. Guy, 2022
ISBN 978-3-11-075534-3, e-ISBN 978-3-11-075541-1

Bruce M. Landman, Florian Luca, Melvyn B. Nathanson, Jaroslav Nešetřil, Aaron Robertson (Eds.)
Number Theory and Combinatorics. A Collection in Honor of the Mathematics of Ronald Graham, 2022
ISBN 978-3-11-075343-1, e-ISBN 978-3-11-075421-6

www.degruyter.com