Learned surrogates and stochastic gradients for accelerating numerical modeling, simulation, and design

Alex Beatson

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Ryan P. Adams

September 2021

Abstract

Numerical methods, such as discretization-based methods for solving ODEs and PDEs, allow us to model and design complex devices, structures, and systems. However, this is often very costly in terms of both computation and the time of the expert who must specify the physical governing equations, the discretization, the solver, and all other aspects of the numerical model. This thesis presents work using deep learning and stochastic gradient estimation to speed up numerical modeling procedures.

In the first chapter we provide a broad introduction to numerical modeling, discuss the motivation for using machine learning (and other approximate methods) to speed it up, and discuss a few of the many methods which have been developed to do so.

In chapter 2 we present composable energy surrogates, in which neural surrogates are trained to model a potential energy in sub-components or sub-domains of a PDE, and then composed together to solve a larger system by minimizing the sum of potentials across components. This allows surrogate modeling without requiring the full system to be solved with an expensive ground-truth finite element solver to generate training data. Instead, training data are generated cheaply by performing finite element analysis with individual components. We show that these surrogates can accelerate simulation of parametric meta-materials and produce accurate macroscopic behavior when composed.

In chapter 3 we discuss randomized telescoping gradient estimators, which provide unbiased gradient estimators for objectives which are the limit of a sequence of increasingly accurate, increasingly costly approximations – as we often encounter in numerical modeling. These estimators represent the limit as a telescoping sum and sample linear combinations of terms to provide cheap unbiased estimates. We discuss conditions which permit finite variance and computation, optimality of certain estimators within this class, and application to problems in numerical modeling and machine learning.

In chapter 4 we discuss meta-learned implicit PDE solvers, which allow a new API for surrogate modeling. These models condition on a functional representation of a PDE and its domain by directly taking as input the PDE constraint and a method which returns samples in the domain and on the boundary. This avoids having to fix a parametric representation for PDEs within the class for which we wish to fit a surrogate, and allows fitting surrogate models for PDEs with arbitrarily varying geometry and governing equations.

In aggregate, the work in this thesis aims to take machine learning in numerical modeling beyond

simple regression-based surrogate modeling, and instead tailor machine learning methods to exploit and dovetail with the computational and physical structure of numerical models. This allows methods which are more computationally and data-efficient, and which have less-restrictive APIs, which might better empower scientists and engineers.

Acknowledgements

This PhD has been a wonderful journey, entirely due to the host of people whom I have had the privilege of sharing it with.

For most of my PhD I have had the great fortune to be advised by Ryan Adams. Ryan has taught me how to do research and perhaps more importantly how to be a researcher. As well as being a technical encyclopedia and teaching me how to reason, work, write, and communicate, I have learned much from him about collaboration and how to navigate the professional landscapes of machine learning, tech, and academia. I fall short of Ryan's example and work ethic, but it has been an absolute privilege to work with him. Ryan also has a great ability to guide you and fully engage with your work while also making you feel a full-fledged collaborator. The best gift a mentor can give, to paraphrase David Whyte, is not any rewards that may have been earned, but the invitation to be a full participant in the conversation, as we walk this road with all its difficulties and minor triumphs, and the invitation to discover a sense of artistry and joy in the journey itself.

I'm incredibly grateful to the other collaborators I've worked with during this PhD, who have made the work in this thesis possible and from whom I've learned so much, and to my colleagues in LIPS, who have provided insightful discussions, banter, and camaraderie: Sachin Ravi, Tianju Xue, Jordan Ash, Deniz Oktay, Ricky Chen, Ari Seff, Geoffrey Roeder, Yucen Luo, Nick McGreivy, Sunny Qin, Josh Aduol, Daniel Suo, Zhaoran Wang, Diana Cai, Gregory Gundersen, Sulin Liu, Jad Rahme, Yaniv Ovadia, Sam Barnett, and more. The times we spent at the whiteboard, pair programming, discussing papers or ideas, or even on slack or zoom, have been the most enjoyable and educational times of my PhD.

I'd also like to extend my gratitude to my initial mentor at Princeton, Barbara Engelhardt, and her lab (the BEEhive). Even though genomics (her application area) was not my area of interest, Barbara and her group provided an incredibly welcoming environment for someone who was new to the USA and relatively new to machine learning. I owe a debt to her and to BEEhive members Bianca, Allison, Greg, Li-Fang, Derek and Niranjani for helping me feel at home and for helping me navigate my initial steps both in the field of machine learning and in Princeton and the Computer Science department.

For the first two years of my PhD I worked with Han Liu. Han is a great teacher and font of statistical knowledge. I will forever benefit from learning statistical ML and linear algebra from him. I'm also indebted to his former student Zhaoran Wang, who took me under his wing in my first year and helped me write my first NeurIPS paper. I had no idea what I was doing, but Zhaoran

handed me an idea and provided clear guidance and every resource each step of the way. The reality distortion field of his enthusiasm, confidence and guidance tricked me into working step by step until we had written a paper seemingly without encountering an obstacle. It wasn't until later that I understood providing this sort of "effortless" guidance to another is much more difficult than doing something yourself.

I'm very grateful to my mentors from wonderful summers at Google: Pedro Moreno and Mohamed Elfeky from Google NYC, who hosted me in 2016, and Olivier Teytaud, Sylvain Gelly, and Karol Kurach from Google Brain Zurich, who hosted me in 2017. Both experiences taught me much about how to code, do machine learning at scale, and navigate the world of industry.

My time at Princeton and in the USA has been a joy due to the many and hopefully lifelong friends I've been lucky enough to spend it with. These have been some of the best years of my life, and I hope to share many more adventures with you all. I am thankful for the CS lunch and happy hour crews, officemates at Prospect Ave and on each floor of Olden St, the movie and book clubs, my flatmates over the years, those who have joined in road trips, outdoor adventures, dinners, and tennis, and especially to those who have been there for years of friendship, adventures, and countless hours of great conversation - you know who you are. I'm also grateful to friends made in Philadelphia when I escaped from the Princeton bubble for a year in the big city, those who made me feel at home during summers in New York and Zurich, those who welcomed me into their lives and gave me an instant social life when I fled Covid-19 to Auckland, and those who have hosted me in Princeton and Philly during the nomadic period in which I have written up this thesis.

A few people from home should be mentioned. I am probably doing research today due to coffees in college with Shaun and his excitement and clarity about his own work. Our bet, where the person who showed up early to the lab the fewest times in a week had to pay for coffee, brought me the closest I've ever been to a consistent work schedule. Appreciation also goes to Nick for showing me by example that there is a whole world beyond NZ to explore, Andrew for showing how to be ambitious while remaining grounded and focussed on what matters, and Timmy for providing a welcoming home and tour guide every time I visited Auckland. Geoff Chase, my undergrad senior year advisor, made this path possible with his mentorship and set me on it by insisting to write me a recommendation letter for PhD programs rather than masters programs.

Finally, all of this has been made possible by my family. Thank you to Erin and Tane for your empathy, insight into your own crafts and processes, and camaraderie as we go through life. Thank you to my parents Meg and Rick for your unconditional and endless love and support, both during these years and the decades prior, without which nothing I do would be possible.

Contents

Chapter 1

Introduction

Numerical methods, such as discretization-based methods for solving ODEs and PDEs, have for centuries helped humanity achieve numerous feats of analysis and design. Just a few examples include: using numerically-evaluated sequences and series to build early understanding of geometry; predicting the motion of celestial bodies using ordinary differential equations (ODEs); modeling massive structures under gravitational, wind and seismic loads using partial differential equations (PDEs); optimizing the design of aircraft wings to maximize lift and structure and materials of fuselage to achieve safety with low weight; and modeling electromagnetic and thermodynamic processes in varied regimes such as the human cardiac system, the atmosphere, and the plasma in a fusion reactor. Such numerical methods are costly in terms of both computation and the time of the expert who must specify the physical model, the discretization, and the solution procedure. In this thesis I present a number of methods which use tools from machine learning to accelerate numerical modeling, simulation and design. Before the research contributions which form the bulk of the thesis, in this introduction I give an overview of numerical modeling, motivate both its importance and the need for machine learning techniques which can reduce its costs, and give a brief overview and history of existing techniques which do so.

1.1 Numerical methods: a vital engineering tool

In this section I aim to give a broad perspective on why numerical methods and models are important, but why they have some significant costs that we should wish to reduce with machine learning. It is impossible to give a comprehensive overview of numerical methods in this brief introduction, as they touch on many rich areas of applied mathematics. To the interested reader I strongly recommend

the Princeton Companion to Applied Mathematics (Higham et al., 2015), which has a great overview of the foundational concepts which connect numerical methods to each other and to other areas of applied mathematics, and a broad overview of important methods and application domains.

1.1.1 Numerical reasoning

Why are numerical methods important? They allow us to model systems governed by some mathematically defined laws under simulation. Often, but not always, these are physical systems and the laws are known laws of physics. Being able to model such systems allows us to reason about the world around us without resorting to brute-force experimental trial and error. I will use *numerical modeling* to refer to the actions the practitioner takes in modeling such systems, which include (i) specifying a mathematical model and (ii) choosing and using appropriate numerical methods to obtain an approximate solution.

Numerical modeling is used to solve several related problems. The three most important are simulation, optimization, and system or parameter identification. Simulation is the most basic and fundamental of these. Given some set of laws and parameters, to simulate the system is to predict or understand how it behaves or evolves by modeling the system's solution (whether steady-state or as a function of a variable such as time).

The second is system optimization. Given some laws and an objective function which measures the desirability of a solution – of the output of a simulation – what system parameters give rise to the optimal solution? There are many approaches to and algorithms for optimization depending on the specifics of the problem. However, in most cases simulation is a critical subroutine which may be performed many times. We must be able to evaluate the solution resulting from some given parameters in order to evaluate the objective function corresponding to those parameters.

The third is system identification. Given some observations about the world, coupled with some known laws, can we identify the system parameters which gave rise to these observations? As with optimization, this task has simulation as a critical subroutine – we need to simulate the system to know what observations some given parameters could generate, and whether these observations are close to the observations we made. Often, system ID reduces to an optimization problem. This reduction holds when we seek parameters which minimize an error metric between the true observations and the simulated observations, or equivalently, seek the maximum-likelihood parameters under some observation noise model. SysID does not naively reduce to optimization in some cases, such as when using Bayesian inference to sample from a posterior distribution over parameters. Such cases are

beyond the scope of this thesis, but almost always still have simulation as a critical subroutine.

The key thing to note is that numerical analysis or simulation allows us to do counterfactual reasoning. Given a hypothetical set of laws and parameters, what happens? This counterfactual reasoning allows both system optimization and system identification.

Such reasoning and such tasks do not *necessarily* require numerical analysis, which is one of three options we have to evaluate outcomes. The other options are: to evaluate the outcome of a system by building or testing the system in the real world, or to use an analytic mathematical model if one is available. However, both options have severe limitations.

Alternative: analytic reasoning

Analytic solutions are often desirable when they are available. These permit system analysis/simulation, optimization, and identification in the same way as numerical solutions. One begins with the laws and parameters which specify a system, mathematically derives a solution, and potentially iterates between updating parameters and deriving a solution as part of an optimization or inference procedure.

The difference is that, as opposed to numerical solution, an analytic solution involves writing an expression for the *exact* solution, which may be evaluated in closed form. This avoids the computation/error trade-off which is central to numerical methods.

For example, analytic formulae exist for the roots of general polynomials up to degree 4. It is usually better to use these formulae than to use a numerical method such as Newton's method to find the roots. For polynomials of degree greater than 4, the Abel-Ruffini theorem states that there is no general analytic formula for their roots. Some special higher-order polynomials do permit analytic solution if the mathematician reasons about the structure of that particular polynomial, but almost all do not. If either the polynomial is one of the majority that do not – or if one wants to avoid the mathematical labor of determining the analytic solution for one of the few polynomials that permit it – then the only option is to resort to numerical methods.

Partial differential equations (PDEs), which we will discuss in depth later, exhibit a similar pattern. Many simple linear PDEs, such as Poisson's equation, permit analytic solutions when the domain is simple (e.g. a disc or unit square) and the boundary conditions (BCs) and parameters belong to a simple class of functions (e.g., constant or quadratic). However, it might be tedious to derive analytic solutions for each given realization of the PDE consisting of a unique combination of domain, BCs, and parameters. And most nonlinear PDEs, and linear PDEs with less simple domains, BCs, or parameters, do not permit analytic solution.

Alternative: experimental reasoning

One might also perform system analysis, optimization or identification experimentally. In this case, analysis involves observing the outcome of a real-world system: e.g., measuring the heat at some point on an object or measuring the drag over an object in a wind tunnel instead of deriving analytic or numerical solutions to the corresponding thermodynamic PDEs.

Counterfactual reasoning (for system optimization or identification) requires the ability to construct and observe the system in the real world. This is fine when trials are cheap – e.g. when learning how to throw a ball to maximize the distance, or learning how to imitate someone's tennis serve, the cost of evaluating the system (throwing or serving a ball) is negligible. It is also a necessary step when faced by incredibly complex systems we are unable to model with sufficient accuracy, such as in medicine, where even the best numerical models for predicting the effect of a drug on the human body – or even the interaction of a drug molecule with one given protein – are far too simple and inaccurate to completely remove the need for experimental trials. And in many cases, experimental trial is an essential step in validating results obtained from a numerical or analytic model – e.g., testing a numerically-optimized aeroplane wing in a wind tunnel – as inaccuracies or approximations in the mathematical system specification or in the solution procedure might cause error in the results.

However, for systems which can be modeled and which are expensive to build or have a high cost of failure, we cannot fully rely on experimental trial. If we wish to optimize the design of a fusion reactor, or make sure that a design for a skyscraper will not collapse, it is not feasible to iterate blindly building a design and seeing what happens. We need to use an analytic or numerical model to reason about the system *before* building the design in the real world.

1.1.2 Elements of numerical modeling

What is a numerical model? It is a mathematical model for a system or quantity of interest, coupled with numerical methods used to solve the system and reveal the result of this model.

In the work in this thesis the mathematical model is often a differential equation such as an ordinary differential equation (ODE) or partial differential equation (PDE), although naturally there are many other possibilities. The scientist will often fix some structure of the model, while allowing some structure or parameters to vary over the course of optimizing or fitting the model, or of considering different scenarios. For example, we may know that a system should obey the PDE given by Poisson's equation, or the ODE for celestial motion given by coupling Newton's laws of motion with his law of gravity, but over the course of multiple analyses the domain, BCs or source

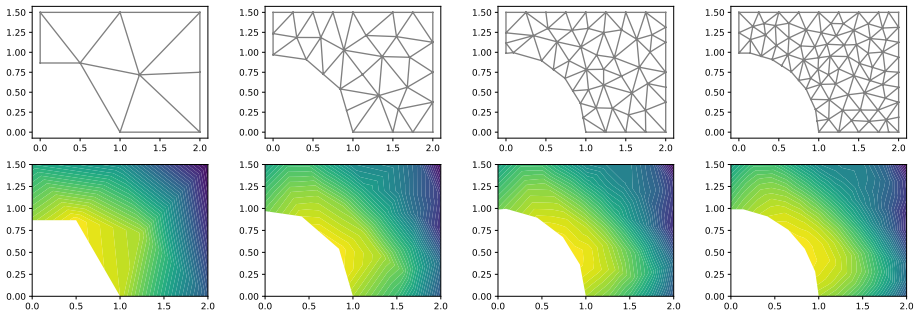


Figure 1.1: A finite element mesh modeling a rectangular body with an inverse circular fillet in one corner. Top: the mesh. Bottom: a scalar field u approximated by a piecewise linear function on this mesh. Left to right: the mesh is refined, leading to a better approximation of the geometry and the function u.

terms of Poisson's equation may change, or the bodies considered in the n-body problem may change in number, mass, or initial position and velocity.

Given this mathematical model, if an analytic solution is not available (and for all but the simplest systems it usually is not), it must be solved numerically. There are a multitude of options, even when considering a single mathematical system. However, most have some common elements, outlined below.

– *Representation.* First, we need to choose a basis for the solution. The true solution will often be a continuous *field*, but computers cannot work with fields directly. We need to represent the solution with some finite data structure. Usually this means *discretizing* the solution: approximating the solution with a representation in terms of a finite set of points or coefficients. For example, in celestial mechanics, the solution might be a function $y(t) : \mathbb{R}^1 \to \mathbb{R}^d$, but we might be happy to consider only the positions and velocities of the bodies at some finite set of time points: y_t for $t = t_0, t_1, ..., T$. For a Poisson problem the solution might be a field $u(x, y) : \mathbb{R}^2 \to \mathbb{R}^1$, but in order to solve it numerically with finite element analysis (FEA), we represent the continuous solution field with a piecewise low-degree polynomial over the domain, which has some finite set of coefficients or interpolation points. Usually there is some parameter which controls the fidelity of this approximation: e.g., the number of steps used in solving an ODE or the fineness of a finite element mesh. Figure 1.1 shows mesh refinement for a finite element model.

– *Translated laws.* Next, we must translate our mathematical model into a form which can be applied to the chosen approximate representation. If the solution is discretized, the mathematical model must be discretized, i.e. turned into an equation which the coefficients of the discretized approximate solution should satisfy. The approximate solution given representation and translated

laws should converge as fast as possible to the true solution as the fidelity parameter is increased, so that we may obtain an accurate solution without too much computation. For example, in finite element analysis we first translate the PDE constraint (such as Poisson's equation, $\delta u(x) = f(x) \forall x \in Omega$) to a variational form defining what constraints a candidate solution $u \in \mathcal{U}$ should satisfy with respect to test functions $v \in \mathcal{V}$. When $\mathcal{U}$ and $\mathcal{V}$ are sufficiently high-order Sobolev spaces consisting of functions $\Omega \to \mathbb{R}^d$, we recover the true solution of the PDE u^*, while if we let them be e.g. piecewise polynomials on a finite element mesh, we retain convergence of the $u \in \mathcal{U}$ which satisfies the variational form to the true solution as we increase the fineness of the mesh and the order of the polynomials. In solving an ODE, we must choose how to go from the law $\frac{\partial y(t)}{\partial t} = f(t, y(t))$ to a rule in terms of the discrete solution, $y_{t+1} = y_t + g(t, y_t)$. Common choices include the forward Euler method, $y_{t+1} = y_t + \delta_t f(t, y_{t+1})$, backward Euler, $y_{t+1} = y_t + \delta_t f(t+1, y_{t+1})$ or Runge Kutta methods, which take multiple trial Euler-like steps and combine the results to achieve faster convergence with the step size δ_t (Süli and Mayers, 2003).

– *Numerical solution.* Finally, we must find the solution in the approximating basis which satisfies the translated laws. Sometimes, as in the forward Euler method, this step is direct (just evaluate the right hand side of $y_{t+1} = y_t + g(t, y_t)$ given the current value of y_t). Sometimes this step requires a numerical subroutine such as an iterative method. Implicit ODE solvers require a root finding method; e.g., to find the y_{t+1} which satisfies $y_{t+1} = y_t + g(t, y_t)$ in the backward Euler method. In solving PDEs with FEA, we assemble the variational form into a linear or nonlinear system in terms of the coefficients of the approximate solution in the finite element basis. If the PDE is linear, we solve the system with a direct or iterative method; if it is nonlinear, we use an iterative algorithm such as Newton's method or Picard iteration.

For some problems, some of the above steps are trivial, such as the numerical solution step for the forward Euler method. In other cases, one or more of the steps may involve numerical subroutines about which whole textbooks have been written, such as in FEA, where we must use specialized algorithms to generate a good mesh to represent the solution, assemble the variational form into a sparse (non)linear system, solve the nonlinear system with a Newton-like method, and solve the linearized system at each step of root finding.

Figure 1.2: Areas of n sided polygons converge to the area of a circle in the limit $n \to \infty$.

1.1.3 Examples of numerical modeling

Modeling limits of sequences and series

One of the oldest examples of numerical modeling is the *method of exhaustion*, used in ancient Greece and China to reason about areas and volumes. Archimedes used this method to estimate the value of π, by estimating the area of a unit circle as the limit of a sequence of polygons.

The underlying mathematical law is the relationship $A = \pi r^2$. However, the area of the circle is not known, so π cannot be determined. We choose to represent A as the limit of areas A_n of a sequence of regular polygons each with n sides. Whether the polygon is inscribed inside the circle (A_n^-, where $A_n^- < A$) or outside the circle (A_n^+, where $A_n^+ > A$), the difference in areas $|A_n - A|$ can be reduced by increasing n. The sequences A_n^+ and A_n^- provide upper and lower bounds on the value of A. The discretized problem takes the form of a series, with the translated law $A_n = \hat{\pi}_n r^2$, where r is the radius of the circle around which or within which the polygon is inscribed. To solve the problem, one takes the A_n corresponding to the most many-sided polygon for which one can compute the area (Archimedes used 96), and uses the corresponding $\hat{\pi}_n^+$ and $\hat{\pi}_n^-$ as the upper and lower bounds on π.

Modeling temporal processes with ODEs

Temporal processes often include a variable of interest y and a rate of change with respect to time, $\frac{dy}{dt}$, which is a function of time and the current state: $\frac{dy}{dt} = f(y, t)$. An example is Newtonian mechanics, such as those governing celestial motion. Here, $y = [x, v]$, $x, v \in \mathbb{R}^{n\times 3}$ is the positions and velocities of the n bodies in space ($\mathbb{R}^3$). We have $\frac{dy}{dt} = [\frac{dx}{dt}, \frac{dv}{dt}] = [v, a]$, where $a \in \mathbb{R}^{n\times 3}$ is the acceleration of the bodies due to gravitational force from the other n bodies (which depends on x). While these equations are easily understandable, there is no analytic solution when $n \geq 3$: even for the three body problem, it is very hard to reason by hand about all but very special scenarios.

For most n-body problems, as for many temporally evolving systems in general, numerical methods are required. The relation $\frac{dy}{dt} = f(y, t)$ specifies an *ordinary differential equation* or ODE. To solve

them, we usually search for a solution $y(t)$ such that $\frac{dy}{dt} = f(y,t)$ for all t in the considered time range $[t_0, t_f]$, given some initial conditions y_0.

Usually, we represent a solution in terms of some set of y_t, $t = [t_0, t_1, ..., t_N]$: a series of state values at particular points t. It remains to translate the law $\frac{dy}{dt} = f(y,t)$ to the discretized system. The easiest way to do this is to take the forwad Euler method: $y_{t+1} = y_t + f(y_t, t)$, in which case the "numerical solution" step simply involves evaluating each iterate in turn. However, much better convergence of the approximate solution to the true solution can often be obtained. Replacing the *explicit* forwad Euler with an *implicit* method such as the backward Euler, $y_{t+1} = y_t + f(y_{t+1}, t+1)$, which requires a root-finding algorithm such as Newton's method to numerically solve for each subsequent iterate y_{t+1}, can greatly improve stability and convergence for ODEs which are "stiff" (i.e., tend to be unstable with explicit solvers, unless extremely small step sizes are used, even though the solution is smooth). Replacing the first-order Euler method (forward, backward, or other) with a higher order method (such as Runge Kutta or linear multistep methods) can improve the rate of convergence of the error from $\mathcal{O}(1/N)$, where N is the number of steps used, to $\mathcal{O}(1/N^p)$, where p is the order of the method. Runge Kutta methods can be interpreted as (possibly repeatedly) applying *Richardson extrapolation*, a sequence acceleration method discussed later, to Euler's method, to develop faster-converging approximations.

Modeling spatial and spatiotemporal systems with PDEs

Spatial systems and spatiotemporal systems, or other processes which involve partial derivatives with respect to multiple variables, can often be modeled by *partial differential equations*, or PDEs. The solution to a PDE is a field u which maps from a coordinate $x \in \Omega$, where usually $\Omega \subset \mathbb{R}^{d_1}$, to a value $u(x) \in \mathbb{R}^{d_2}$. The solution is described by a law $F(u)(x) = 0$, $x \in \Omega$, where F is a linear or nonlinear operator involving u and its partial derivatives. For example, in the Poisson equation (a steady-state equation arising often in electrostatics and fluid mechanics) we have $d_2 = 1$ (i.e. u is a scalar potential on Ω) and $F(u) = \delta u - f$, where f is a (possibly spatially varying) source term and δ is the Laplace operator; the trace of the Hessian, trace($\frac{\partial^2 u}{\partial x^2}$). (In physics the Laplace operator is often written as $\nabla^2 u$, however we avoid this notation as it might confuse a machine learning audience used to ∇^2 describing a Hessian, not the trace of the Hessian.) Figure 1.3 shows an example mesh, source term, and solution for the Poisson problem on a disc. In the heat equation we have $F(u) = \frac{\partial u}{\partial t} - \delta u$. Both of these are *linear* PDEs, i.e. the operator F is linear in u and/or its partial derivatives, however many systems of interest are *nonlinear* PDEs. A simple example is a nonlinear Poisson problem, varieties of which arise in many scenarios when simplifications used to obtain linearity do not hold.

For example, $F(u) = \text{div}((1+u^2)\nabla u) - f$, which is equivalent to the standard Poisson problem when $u \approx 0$. We usually also have some boundary conditions which constrain the value of u (a "Dirichlet boundary condition") or the derivative of u (a "Neumann boundary condition") on some or all of the boundary of the domain $\partial\Omega$.

As with ODEs, the equations are simple, but analytic solutions are not available except for special cases. This is very often true when F is linear but almost always so when it is nonlinear. In order to solve PDEs numerically, as we cannot easily reason about arbitrary fields, we must introduce some approximate family of functions with which to represent u. A common choice is finite elements, employed in *finite element analysis* (FEA). The domain Ω is discretized using a mesh and the solution is represented as a piecewise polynomial with the mesh defining the pieces.

To translate the law $F(u)(x) = 0 \ \in \Omega$ to this discretized space, FEA first introduces a variational or weak form of the PDE, $\int_\Omega < F(u)(x), v(x) > dx = 0 \ \ \forall v \in \mathcal{V}$. Ignoring mathematical subtleties, this weak form is equivalent to the original form if the family $\mathcal{V}$ is chosen to be the set of all functions mapping from Ω to $\mathbb{R}^{d_2}$. This variational form is amenable to discretization. Let the family of piecewise polynomial functions representable in the finite element function space be denoted $\mathcal{V}_{p,n}$, where n is a measure of the number of mesh elements and p a measure of the polynomial order of the space used, and consider an approximate solution $u_{p,n}$ in this space. The translated law is $\int_\Omega < F(u_{p,n})(x), v(x) > dx = 0 \ \ \forall v \in \mathcal{V}_{p,n}$. To test this constraint it suffices to test the integral is zero for a finite set of functions which form a basis for $\mathcal{V}_{p,n}$. We assemble the integral $\int_\Omega < F(u_{p,n})(x), v_i(x) > dx$ for each v_i in this basis, and rewrite it as a function of $\vec{u}$, the coefficients of the function $u_{p,n}$ in the piecewise polynomial basis. The result is a system of equations $\vec{F}(\vec{u}) = 0$, which $\vec{u}$ must satisfy in order to satisfy the discretized weak form of the PDE for all v_i forming a basis for $\mathcal{V}_{p,n}$. We have translated the continuous law into a numerical system of equations. If the PDE is linear, the system will also be linear and we can solve it with an appropriate method such as QR decomposition or conjugate gradients. If the PDE is nonlinear, the system of equations will be nonlinear and we will need to apply an iterative root finder such as Newton's method (which will usually call a linear solver at each root finding step). Importantly, as p and n increase and under appropriate conditions, the solution $u_{p,n}$ which satisfies this system of equations converges to the true solution u to the PDE.

Statistical and optimization-based modeling

Modern machine learning is, naturally, also a form of numerical modeling, albeit one with a very different flavor to more traditional approaches such as those for modeling systems with ODEs and

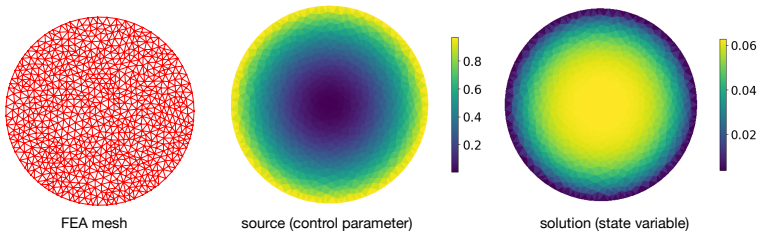


Figure 1.3: The Poisson equation on a unit disc. Left: the finite element mesh. Center: the source term. Right: the approximate finite element solution u, found with FEA. Figure: Xue et al. (2020a).

PDEs. We are often interested in a "ground truth" data generating or labeling process or function, or interested in an optimal control policy or decision rule. The different flavor of machine learning arises because the description of this function does not usually come in the form of physical constraints it should obey, but in the form of some data (e.g., labeled training examples) or interaction (e.g., state-action sequences of a reinforcement learning agent and the associated rewards) coupled with a loss function (which might include per-datum losses and regularization or a prior) we want our function of interest to minimize. (Bayesian inference instead comes with an observation model, and the goal is to characterize the posterior distribution over functions or parameters of interest).

Another difference is the function approximation typically employed – neural networks, kernel methods, and linear models, as opposed to the highly structured piecewise polynomials and piecewise linear functions often found in physical modeling. This is in part because the input and sometimes output spaces in machine learning are usually much higher dimensional than in physical problems where $\mathbb{R}^2$, $\mathbb{R}^3$, or in general $d < 10$ are the most common.

Nonetheless, there is much similarity. It is hard to reason analytically about the optimal learned function. We thus restrict to some family of functions which can be represented numerically – e.g. by the weights and biases in a neural network, or by the data itself (used for interpolation) for Gaussian process regression or a kernel method. In the former case, we search for a good function within the function class via (possibly stochastic) optimization, and query this on new inputs of interest by passing the data through the neural network. In the latter case, finding the optimal representation requires no work for an exact GP (just storage of the data points) but querying the "learned" function on new data requires solving a large linear system. As with physical models, we also often wish to place ML-style numerical modeling (whether with NNs, GPs, or many other methods) within an optimization loop, to find the hyperparameters which give the best learning performance for a particular task under the metric of interest (usually, generalization or test error).

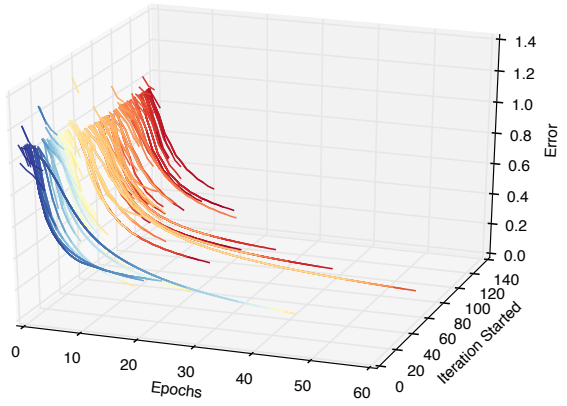


Figure 1.4: Learning curves (performance vs epoch) at each iteration of optimizing the hyperparameters of a neural network. The relative performance with a given hyperparameter setting at earlier epochs of training is a useful but imperfect approximation of the performance at later epochs. Figure: Swersky et al. (2014).

1.1.4 The costs of numerical modeling and design

We have discussed why numerical modeling is important. Why should we seek to accelerate it with machine learning? Numerical modeling procedures have significant nested computational and human costs. These costs are compounded in numerical design and SysID. Consider using FEA to perform design when the system will be governed by a nonlinear timestepping PDE:

Algorithm 1: Numerical design with FEA

for *each qualitatively different candidate design or component* **do**
  Define a model (PDE, BCs, domain) and FEA approximation (mesh, element type, solver);
  Define an objective for PDE-constrained optimization (a set of operating conditions, fitness functions to measure the quality of the solution under each condition);
  for *each of G gradient steps optimizing the free system parameters* **do**
    for *each of C operating conditions* **do**
      for *each of T time steps in the simulation* **do**
        for *each of N Newton steps solving the nonlinear PDE* **do**
          Solve a sparse linear system of size $\mathcal{O}(n^d \times n^d)$, where we have n elements along each of d spatial dimensions;
        end
      end
      Use the adjoint method to compute the gradient of the objective;
    end
  end
end

The outer *for* loop contains the significant human costs of designing an appropriate model, discretization, and objective. Often this process will need to be iterated until results of simulation and optimization are satisfactory. Sometimes significant computation will also need to be spent here (e.g. generating a mesh).

The inner *for* loops include significant computational expense from the many linear system solves required. Unlike in many machine learning workloads, the cost of computing the gradient of the objective is not significant next to the cost of running the simulation and computing the objective. Given a solution and an objective function placed on that solution, the gradient with respect to the PDE parameters can be taken with the *adjoint method* (Lions, 1971; Mitusch et al., 2019). Excluding the cost of the adjoint method, we see $G \times C \times T \times N$ calls to the linear system solver. Computing the gradient with the adjoint method has approximately the same cost as solving a linearized form of the PDE, i.e. a cost of $\mathcal{O}(G \times C \times T)$ solver calls. The cost of PDE-constrained optimization is dominated by the cost of solving the PDE at each optimization step. The cost of each call to the linear solver will depend on the chosen solver and on the structure of the finite element discretization. For a regular grid on the 2d plane and using any direct linear solver, there is a lower bound of $\mathcal{O}(n^3)$ computation for n elements along each spatial dimension (i.e. n^2 total elements). In some cases indirect solvers may do better, and irregular grids may incur extra cost. However, we can naturally never hope to do better than $\mathcal{O}(m)$ for m total elements or degrees of freedom (e.g., $m = n^2$ for a regular $n \times n$ grid in $\mathbb{R}^2$) (Hoffman et al., 1973).

How large can we expect this system to be? For small problems, dozens or hundreds of elements might suffice. However, modeling complex systems like a rocket engine, a fusion reactor, or a large mechanical structure might use as many as millions of elements. The engineer must spend significant effort specifying both the design and numerical method, and may have to wait hours, days, or weeks to see the outcome of a specific simulation, the results of which might then suggest the need to adjust the design or method.

Of course, engineers make use of approximations to lessen this load. Such approximations might take the form of splitting a design or system into components which are simulated separately; introducing approximations into the mathematical model by e.g. simulating something in 2d instead of 3d, considering steady-state instead of temporal simulations, or making assumptions about a particular term in an equation; or simply using a coarser discretization than would be ideal. These approximations both increase the human effort required and introduce a degree of error.

An important concept is the Pareto frontier of effort (computational or human) vs accuracy. There will always be a need for new methods or approximations which improve this frontier. Such

new methods not only decrease the human and computer effort required to run existing workloads, but can increase the accuracy of our numerical modeling (because we no longer have to cut the corners we once did), and allow us to analyze, identify, and design systems too large or complex for the previous state of the art. However, if we introduce a new method, it is important to measure it against the Pareto frontier of an existing method – not a single point on said frontier.

Consider a situation where we introduce a method for using neural networks to predict the solution to a PDE from some parameters, and show that this produces solutions which have only a small amount of error when compared to a given finite element model, or only 10% higher error when both methods are compared to some other known ground truth, while being many times as fast. This information is not sufficient to say that the new method is useful. By changing the fidelity of the finite element method (controlled by the density of the finite element mesh and the polynomial order of the elements) we can also trade off accuracy for speed, and it might be the case that simply decreasing the density of the mesh can create the same speed up with the same or less increase in error. We must measure new methods against a Pareto frontier of the existing methods we compare them to. Unfortunately, much work in the burgeoning area of machine learning for engineering does not do this.

We now understand something of the cost of numerical modeling and design, and how methods to reduce this cost should be evaluated. In the next section, we discuss how machine learning can be and has been used to accelerate numerical modeling.

1.2 Accelerating numerical modeling with machine learning

Leveraging approximations to accelerate numerical modeling and design is not a new idea, and has been an important topic in applied mathematics since at least the early 1900s. We will first outline a general concept of what we might hope to achieve – how we might hope to leverage different sources of information within a problem to better estimate some quantity without fully resolving it with our numerical model. Then we will take a brief tour of prior work along these lines, both work focussing on the general approximation of functions with resolution-speed trade offs, and work which leverages specific physical or algorithmic structure of simulations. It is not possible for this section to be exhaustive, and there will be much important work that we miss for the sake of brevity.

1.2.1 What can we hope to achieve?

Generally speaking, we can consider some baseline numerical model f which acts on some representation of the scenario or design ϕ and returns a quantity of interest $y = f(\phi)$. We wish to approximate y without running f (to avoid the expense involved), instead taking $y \approx \hat{f}(\phi)$, where $\hat{f}$ is our approximate model.

Sources of information

If we are not to call f, then $\hat{f}$ must use some information about f available from one or more other sources in order to provide better-than-arbitrary estimates of y. What such sources of information are available? Many methods will use more than one of the following, and we discuss examples of each in both the overview of existing approaches and in the research contributions of the thesis.

- *Results of cheaper or lower-fidelity simulations* $y_i = f_i(\phi)$. In numerical modeling we often have access to some sequence of approximations f_i, where as $i \to \infty$, $y_i = f_i(\phi)$ converges to the true quantity we are interested in, but so does the difficulty or cost of evaluating f_i. If we know something about the convergence properties of the sequence of models f_i, this knowledge might allow us to use evaluations of some of the models $f_{1:i-1}$ to approximate the result of f_i as $\hat{y}_i = \hat{f}_i(y_1, y_2, ...y_{i-1})$ without evaluating it. A quintessential example of this is *sequence acceleration* (Osada, 1991), discussed in the next section.

- *Results of simulation of similar problems,* $y' = f(\phi')$. Given a training set consisting of parameterizations of particular problems ϕ_i and the solutions to those problems y_i, we may try to learn an approximation $\hat{f}$ to the function/model f which is cheaper to evaluate than f itself. A quintessential example of this is surrogate modeling via simply regressing from the training ϕ_i to the target solutions y_i.

- *Known computational or physical structure associated with the model* f. This is a broad category. In many cases, leveraging known computational structure in the numerical model f or known physical or mathematical properties of the system f approximates will improve methods which rely on generalization from lower-fidelity iterates $f_{<i}$ or from similar problems $y' = f(\phi')$. However, leveraging this structure can also be used to design new numerical models $y \approx \hat{f}(\phi)$ which can be used as simulators without requiring information from f for training or evaluation.

1.2.2 Extrapolation and sequence acceleration

One of the earliest examples of work with a similar flavor to that in this thesis is the subfield of applied mathematics concerned with *sequence acceleration* or *series acceleration*. See Osada (1991) for an excellent overview. Much of the key work in sequence acceleration was done in the early 20th century, however some early transformations were known to Stirling, Euler, Maclaurin, and Seki Kowa.

Given a sequence s_n which converges to a limit, $\lim_{n\to\infty} s_n = s^*$, sequence acceleration is concerned with constructing a transformed sequence s'_n which converges to the same limit but with a faster rate of convergence. This can be restated as designing a transformation operator $\mathcal{T}$, where $s'_n = \mathcal{T}(s_n, s_{n-1}, ..., s_1)$. Formally, $\mathcal{T}$ accelerates the sequence if $\lim_{n\to\infty} \frac{s'_n - s^*}{s_n - s^*} = 0$. However, we are often interested not in just whether $\mathcal{T}$ accelerates the sequence but in how fast the new convergence of s'_n to s^* is: e.g. if the sequence is polynomially converging, $|s'_n - s^*| = \mathcal{O}(1/n^p)$, we would like $p > 0$ to be as high as possible, whereas it would be even better if it were geometrically converging, $|s'_n - s^*| = \mathcal{O}(1-p)^n$, where we would like $0 < p < 1$ to be as large as possible. These are often called logarithmic and linear convergence in cases where the studied quantity is not the big-O behavior of $|s_n - s^*|$ but the limiting ratio of subsequent errors, $\lim_{n\to\infty} \frac{|s_{n+1} - s^*|}{|s_n - s^*|}$.

These methods can be used to accelerate numerical modeling when we have some parameter which trades off computation or effort for fidelity: for example, the resolution of a finite element mesh, or the number of steps or inverse step-size used to solve an ODE, or the number of iterations in a Newton method or in an optimization procedure used to train a neural network. In such cases, we are often interested in the limit of some quantity as we allow an unbounded amount of computation. However, we only have a finite computational budget. Given this finite budget, we might hope that by evaluating the quantity of interest at several different fidelities ($s_1 ... s_n$), we might be able to extrapolate (s'_n) to get a better estimate of the limit s^* than using s_n alone.

It is not possible to find an operator $\mathcal{T}$ which accelerates all sequences. Some assumptions must be made about the original sequence, e.g. its original rate of convergence, and it is also useful to know if the sequence is asymptotically alternating or monotonic.

Each sequence acceleration method has a class of sequences which it will accelerate. Two of the most famous methods are *Richardson extrapolation*, for polynomially convergent sequences, and *Aitken's delta-squared method*, for geometrically convergent sequences.

Richardson extrapolation is designed for sequences which have an asymptotic expansion $s_n = s^* + C1/n^p + \mathcal{O}(1/n^{p+1})$. (Often, n is the inverse of the step-size of an ODE integrator). We choose

some factor $\tau > 1$ and construct the extrapolation

$$R(n, \tau) = \frac{\tau^p s_n - s_{n/\tau}}{\tau^p - 1}$$

We then have:

$$\begin{aligned} R(n,\tau)(\tau^n - 1) &= \tau^p(s^* + C{}^1\!/_{n^p} + \mathcal{O}({}^1\!/_{n^{p+1}})) - (s^* + C{}^1\!/_{(n/\tau)^p} + \mathcal{O}({}^1\!/_{(n/\tau)^{p+1}})) \\ \implies R(n,\tau) &= s^* + \mathcal{O}({}^1\!/_{n^{p+1}}) \end{aligned}$$

Thus, Richardson extrapolation eliminates the $C{}^1\!/_{n^p}$ term in the asymptotic expansion of s_n, giving a higher order / faster converging error $\mathcal{O}({}^1\!/_{n^{p+1}}$.

Aitken's delta-squared process is the transformation:

$$s'_n = \frac{s_n s_{n-2} - s_{n+1}}{s_n + s_{n-2} - 2 * s_{n-1}},$$

or equivalently,

$$s'_n = s_n - \frac{(s_n - s_{n-1})^2}{s_n - 2s_{n-1} + s_{n-2}}$$

Aitken's delta-squared process accelerates any geometrically (or linearly) convergent sequence. As opposed to Richardson extrapolation, which eliminates the slowest polynomial error term, Aitken's delta-squared process can be thought of as eliminating the slowest geometric error term. If the series has an asymptotic expansion $s_n = s^* + a^n$ with $0 < a < 1$, then the transformation is exact ($s'_n = s^*$) for all n. If the series has an asymptotic expansion $s_n = s^* + a^n + \mathcal{O}(b^n)$ where $0 < b < a < 1$, then the transformed series has the asymptotic expansion $s'_n = s^* + \mathcal{O}(b^n)$, i.e. eliminating the slower a^n term.

Sequence acceleration methods are not today very commonly used by practitioners. However, as well as being of historical and intellectual interest, they underpin a number of fast-converging numerical methods for various applications, which can be derived by application of one of these sequence acceleration methods to a "base" numerical method. For example, higher-order Runge-Kutta methods for ODE integration can be derived from first-order Euler methods by repeatedly applying Richardson extrapolation $p - 1$ times to achieve a method with convergence of order p. (Runge-Kutta methods slightly predate Richardson's introduction of his general extrapolation scheme, but their motivation, method and analysis is an exact application of the extrapolation). Richardson extrapolation is also used to derive Romberg's method for integration by repeated application to the

trapezium rule. Aitken's delta-squared process can be applied to a fixed point iteration to derive Steffenson's method, which has quadratic convergence (a la Newton's method) while not requiring derivatives. Historically, sequence acceleration methods have been applied to one-dimensional sequences, and they can sometimes be lacking when applied naively to higher-dimensional problems (e.g., Steffenson's method runs into a "curse of dimensionality" with high-dimensional root-finding problems). Much of the work on sequence acceleration in recent decades has been on extending or developing methods for vector sequences (Osada, 1991).

1.2.3 Surrogate modeling

A natural approach to accelerating numerical modeling is to regress from a parameter of the design or factor of variation of the numerical model on to the solution (or a quantity of interest associated with the solution). This regression model provides a *surrogate model* for the base simulator. If the surrogate model is accurate, it can be queried in place of the simulator to do analysis, optimization or system ID – i.e., to answer questions about the system's behavior in particular scenarios, the optimality of designs, or the likelihood of system parameters given observations.

Any and every regression method can and has been used for such tasks – linear regression, polynomial regression, kernel methods, neural networks, and more. For overviews of the broad volume of work in this area, see Simpson et al. (2001); Martin and Simpson (2005); Queipo et al. (2005); Koziel and Leifsson (2013); Willard et al. (2020). A distinction beyond the regressor is *what is predicted*. One route is to directly predict the solution y to a numerical model f with parameters ϕ, $y = f(\phi)$, using the learned regressor: $y \approx \hat{f}(\phi)$. This lets one use the surrogate to answer whatever questions one wishes about the solution y. However, y might be indexed by a high-dimensional vector of coefficients, and the learning task of predicting y from ϕ might be very hard. Worse, the way in which y is represented (e.g., the size and meaning of the vector of coefficients) might change depending on ϕ: e.g. if changing ϕ means a problem domain is larger or smaller, or has different geometry. This will break many regression models. As such, it is sometimes more useful to choose an alternate quantity to predict. If one is interested in the performance of a design, measured by an objective function, then one can predict that scalar objective value directly from ϕ without modeling the solution. This could be an easier learning problem if, for example, there are ranges of ϕ which cause large, unstable variations of the solution but which all have a poor objective value.

Another choice is possible when simulation can be framed as a bi-level problem. In this scenario, the solution to one or more "inner" problems $z = g(\gamma)$ is used when solving the "outer" problem

$y = f(\phi)$, and where g is the solution procedure for the inner problem; γ are the parameters, and z is the output of the inner problem. The inner problem(s) might have more regularity than the outer problem – i.e., the mapping $z = g(\gamma)$ may be easier to learn, or the outer problem solution representation y may change significantly with ϕ while the inner solution representation z does not – often this occurs when g may be called multiple times, varying with ϕ, to compute $z_{1..l}$ which are used in computing y. If a surrogate model for g would significantly reduce the cost of one full solution, training a surrogate $z \approx \hat{g}(\gamma)$ and using this as a component in the numerical model for y may be an effective strategy. For example, Tompson et al. (2017) accelerate solving linear systems which are the computational bottleneck for Eulerian simulation of the Navier-Stokes equations, while in Beatson et al. (2020) we accelerate computing strain energies and strain energy gradients which must be computed at each Newton iteration of solving a continuum mechanics PDE.

One other factor of interest for surrogate models is the training method. Naively applying regression requires one to solve many "full" simulations to generate training data, and is thus only useful if one would otherwise need to query the simulator many more times than the number of training points required (which might be thousands). Instead training a model to predict the solution of some sub-problem can allow training data to be generated much more cheaply (Tompson et al., 2017; Beatson et al., 2020). Another avenue is to exploit the particular structure of the numerical model to avoid the need to call the original simulator entirely, e.g. by learning a model $y \approx \hat{f}(\phi)$ by minimizing some physically-motivated loss $\mathcal{L}(y, \phi)$ which is also minimized by the solution to the original numerical model f (Xue et al., 2020a).

1.2.4 Model order reduction

Rather than predicting the outcome of a simulation or of one of the subroutines in the simulation, *model order reduction* methods learn a solution basis which allows for more efficient simulation. Most of them rely on a projection operator to project the solution to a lower-dimensional latent space (or return it from this to the full solution space). The laws governing the numerical model must then be approximated by some laws in the latent space which give rise to a similar result.

The most well known method is the *proper orthogonal decomposition* (POD) (Chatterjee, 2000), used to reduce the complexity of finite element simulations. POD involves performing principal component analysis on some set of observations, called snapshots, of y, where the solution y to the numerical model is a field of interest such as the velocity, density and pressure fields in a Navier-Stokes simulation. We represent $y_i(x) = \sum_{k=1}^{K} \alpha_{i,k}\eta_k(x)$, where η_k are the principal components, y_i are

the solutions at different times or for different systems or designs, $\alpha_{i,k}$ are the weights, and x is a coordinate in the domain Ω. As with PCA in machine learning, the POD is often used for understanding and visualizing data, which in the POD's case arises from real-world experiments or from simulation. However, the POD is also used as a tool to accelerate simulation, by projection of the linear system governing the finite element model, $A\vec{y} = b$, into a lower dimensional space, $A'\vec{\alpha} = b'$.

Another dimension reduction method related to the work in this thesis is static condensation, or Guyan reduction (Guyan, 1965). In finite element analysis it is often the case that only some degrees of freedom (often those on the boundary) may be "loaded" (have values constrained via Dirichlet boundary conditions or external forces imposed via Neumann boundary conditions) while the others are "unloaded" and will always be governed by the same physical laws, regardless of the loading scenario. Static condensation provides a way to develop an *exact* reduced-order model when the only factor of variation is the load or boundary conditions imposed and when the underlying PDE is linear. Using finite element analysis, we write the coefficients of the solution in the finite element basis in terms of a linear system $A\vec{y} = b$. This can be partitioned in terms of the free and loaded degrees of freedom as:

$$\begin{bmatrix} A_{l,l} & A_{l,f} \\ A_{f,l} & A_{f,f} \end{bmatrix} \begin{Bmatrix} \vec{y}_l \\ \vec{y}_f \end{Bmatrix} = \begin{Bmatrix} b \\ 0 \end{Bmatrix}$$

This linear system is rearranged to eliminate $\vec{y}_f$, and be written in terms of $\vec{y}_l$ alone. This might be useful when, for example, we are modeling a system formed of multiple components and for many of the components we are not interested in the internal degrees of freedom – only in how those components interact with others via the boundary.

The above methods are linear, and typically are restricted to linear or approximately linear PDEs. Developing model order reduction methods for nonlinear PDEs is an exciting area for machine learning (which has seen a lot of recent work on learning nonlinear schemes for compressing data to latent representations with neural networks). We present some work along this line in the thesis. Other interesting recent work in the same theme includes Bar-Sinai et al. (2019), Maulik et al. (2021), and many more.

1.2.5 Randomized methods

A concept and approach somewhat orthogonal to the aforementinoed methods is to reduce computation via randomization. Often, this involves using randomization as an important tool within e.g.

projection-based model order reduction methods. Random snapshot selection can, given a suitable distribution over snapshots, quickly converge to a basis which can represent the solution with smaller error. In other cases, this involves using randomization to speed up a core subroutine of the numerical solution procedure. For example, randomized algorithms can be used to provide fast approixmations to the matrix SVD or to the solution of linear systems (Drineas et al., 2006; Drineas and Mahoney, 2016), by projecting the system with a random matrix (often of i.i.d. Gaussian vectors) to a lower-dimensional subspace.

Possibly the most well-known use of randomness to accelerate computation is the use of stochastic gradient descent in machine learning (Robbins and Monro, 1951; Bottou, 2010). Given a large dataset, it is computationally prohibitive to estimate the gradient of a model's loss averaged over all examples in the dataset. However, if we randomly choose a batch of examples on which to compute and average the loss, the gradient of this loss is a random but unbiased estimator of the full gradient, with variance depending on the size of the batch. Using this gradient estimator for first-order optimization can lead to much faster optimization (in terms of reduction of the loss as a function of computational effort) than the full-batch gradient, especially as the size of a dataset increases, and particularly as most datasets have some covariance between per-example gradients (i.e. many examples in the dataset may have similar loss and gradient functions). In (Beatson and Adams, 2019), included in this thesis, we show how to build a cheap stochastic gradient estimator for objectives involving discretization-based numerical models.

1.3 Research contributions in this thesis

In this thesis we present some recently developed methods for accelerating numerical modeling, simulation, and design. This work has been been the fruit of collaboration with a number of wonderful colleagues, without whom it the work would not have been possible, and without whom I would have learned far less and had infinitely less fun. Code is available at https://github.com/PrincetonLIPS.

In **Chapter 2** we present *composable energy surrogates* (Beatson et al., 2020); a method which leverages modular structure to learn component-level surrogates for modular PDEs; specifically those governing mechanical meta-material deformation. These mix attributes of surrogate modeling and model order reduction, and allow learning of fast approximate solvers while only using supervision from data collected by solving cheap PDEs governing component subregions, rather than requiring data collected by solving the PDEs governing the full system we wish to solve in deployment. This work was presented at NeurIPS 2020 and was carried out in collaboration with Jordan T. Ash,

Geoffrey Roeder, Tianju Xue, and Ryan P. Adams.

In **Chapter 3** we present *randomized telescoping gradient estimators* (Beatson and Adams, 2019), which are randomized gradient estimators for objectives which are the limit of a sequence of approximations (as when performing optimization or design with numerical models). By randomizing the fidelity of the approximation, we obtain unbiased gradient estimators which trade computation for variance. We provide recipes for choosing within the family of such estimators and demonstrate their effectiveness in system identification and machine learning hyperparameter optimization. This work was presented at ICML 2019 and was carried out in collaboration with Ryan P. Adams.

In **Chapter 3**, we present *meta-learned implicit PDE solvers* (Meta-PDE), which are neural networks representing PDE solution fields, which have initializations meta-learned across a family or distribution of PDEs such that they converge quickly to represent the solution of a given PDE in a few gradient steps of minimizing that PDE's variational energy. This provides a surrogate modeling method which is agnostic to geometry and is mesh-free: while many other surrogate modeling techniques require a geometry and mesh to be fixed across the family of problems they amortize, this method only requires the user to supply a sampler for the variational energy of (or a sampler for the geometric domain of) each problem within the family to be amortized – usually a much easier task. This work is in preparation for publication, and has been carried out in collaboration with Sunny T Qin, Nick McGreivy, and Ryan P. Adams.

Beyond the work presented in the thesis, these themes and ideas have been both cultivated by and continued in papers led by other phenomenal researchers with whom I've had the pleasure of collaborating. "SUMO: Unbiased Estimation of Log Marginal Probability for Latent Variable Models" (Luo et al., 2019a), led by Yucen Luo and Ricky Chen, demonstrates an important application of randomized telescope-like estimators to variational inference. "Randomized Automatic Differentiation" (Oktay et al., 2021), led by Deniz Oktay, uses similar philosophy to randomized telescopes to develop a stochastic gradient method for a regime beyond the usual one of a separable dataset, but in this case for the general setting of linearizable computational graphs. In (Ravi and Beatson, 2018), led by Sachin Ravi, Sachin and I realized the potential of meta-learning as a tool for amortizing optimization and computation (rather than "just" being a tool for few-shot generalization), which is a crucial principle for Meta-PDE; our time thinking hard about bi-level optimization also helped ferment the other two chapters. Finally, "Amortized Finite Element Analysis for Fast PDE-Constrained Optimization" (Xue et al., 2020a), led by Tianju Xue, laid the foundation for the Meta-PDE project by establishing the principle of using a neural network to minimize a variational energy rather than a supervised loss, avoiding expensive training data generation for surrogate modeling.

Chapter 2

Learning composable energy surrogates for PDE order reduction

2.1 Abstract

Meta-materials are an important emerging class of engineered materials in which complex macroscopic behaviour–whether electromagnetic, thermal, or mechanical–arises from modular substructure. Simulation and optimization of these materials are computationally challenging, as rich substructures necessitate high-fidelity finite element meshes to solve the governing PDEs. To address this, we leverage *parametric* modular structure to learn component-level surrogates, enabling cheaper high-fidelity simulation. We use a neural network to model the stored potential energy in a component given boundary conditions. This yields a structured prediction task: macroscopic behavior is determined by the minimizer of the system's total potential energy, which can be approximated by composing these surrogate models. Composable energy surrogates thus permit simulation in the reduced basis of component boundaries. Costly ground-truth simulation of the full structure is avoided, as training data are generated by performing finite element analysis of individual components. Using dataset aggregation to choose training data allows us to learn energy surrogates which produce accurate macroscopic behavior when composed, accelerating simulation of parametric meta-materials.

2.2 Introduction

Many physical, biological, and mathematical systems can be modeled by partial differential equations (PDEs). Analytic solutions are rarely available for PDEs of practical importance; thus, computational

methods to approximate PDE solutions are critical for many problems in science and engineering. Finite element analysis (FEA) is one of the most widely used techniques for solving PDEs on spatial domains; the continuous problem is discretized and replaced by basis functions on a mesh.

The accuracy of FEA and related methods requires a sufficiently fine discrete approximation, i.e., finite element mesh. Complicated domains can require fine meshes that make it prohibitively expensive to solve the PDE. This problem is compounded for parameter identification or design optimization, where the PDE must be repeatedly solved in the inner loop of a bi-level optimization problem.

An important domain where this challenge is particularly relevant is in modeling mechanical meta-materials. Meta-materials are solids in which microstructure leads to rich spaces of macroscopic behavior, which can achieve electromagnetic and/or mechanical properties that are impossible with homogenous materials and traditional design approaches (Poddubny et al., 2013; Cai and Shalaev, 2010; Bertoldi et al., 2017). We focus on the *cellular* mechanical meta-materials proposed by Overvelde and Bertoldi (2014), which promise new high-performance materials for soft robotics and other domains (see Sec 3). Simulation of these meta-materials is challenging due to the need to accurately capture microstructure and small-scale nonlinear elastic behavior. Finite element methods have limited ability to scale to these problems, and automated meta-material design demands accurate, efficient approximate solutions to the associated PDE.

We develop a framework which exploits spatially local structure in large-scale optimization problems—here the minimization of energy as a function of meta-material displacements. Only a small subset of material displacements are of interest, so we "collapse out" the remainder using a learned surrogate. Given a component with substructure defined by local parameters, the surrogate produces an accurate proxy energy in terms of the displacement of the component boundary. A single surrogate can be trained then used to predict energy in a larger solid by summing energies of sub-components. This allows solving the PDE in a reduced basis of component boundaries by minimizing this sum.

Other methods exist for reducing the solution cost of large PDEs. One such is the boundary element method (Aliabadi, 2002), which as with our method "collapses out" the internal degrees of freedom in a PDE leaving a problem in terms of the solution on the boundary. Unlike our method, this is performed analytically and is typically only valid for linear PDEs. Our method might be seen as a *learned* boundary element method for a particular parametric class of nonlinear PDEs. Another related line of work is homogenization. Whether micro-scale effects are modeled with fine-resolution FEM (Schröder, 2014) or a neural network (Xue et al., 2020b), homogenized models require a PDE

formed of homogenous representative volume elements (RVEs), and are accurate only as the ratio between the size of the RVE and the size of the macro-scale problem tends to zero.

Some approaches amortize PDE solving more directly, using neural networks to map from PDE parameters to solutions (Zhu et al., 2019; Nie et al., 2020) or constructing reduced bases via solving eigenvalue problems or interpolating between snapshots (Berkooz et al., 1993; Chatterjee, 2000). These approaches typically require solving full systems to produce training data. Our framework uses the modular decomposition of energy to train surrogate models on data generated by querying the finite element "expert" on the energy in small components, avoiding performing FEA on large systems which are expensive to solve.

2.3 Learning to optimize in collapsed bases

Solving PDEs like those that govern meta-material behavior involves finding a solution u which minimizes an energy $E(u)$ subject to constraints. For mechanical meta-materials, $E(u)$ is the stored elastic potential energy in the material. We propose a framework for amortizing high-dimensional optimization problems where the objective has special conditional independence structure, such as that found in solving these PDEs. Consider the general problem of solving

$$u^* = \arg\min E(u)\,. \tag{2.1}$$

u may be a vector in $\mathbb{R}^d$ or may belong to a richer space of functions. Often we are interested in a subset of the vector u^*, or the values the function u^* takes on a small subdomain. To reflect this, view the solution space as the Cartesian product of a space of primary interest and a "nuisance" space. Denote the solutions as concatenations $u = [x, y]$ where y is the object of interest, and x is the object whose value is not of interest to an application. x is roughly similar to auxiliary variables that appear in probabilistic models, but are marginalized away or discarded from the simulation. We use this decomposition to frame Eq. 2.1 as a bi-level optimization problem:

$$y^* = \arg\min_y \min_x E(x, y)\,. \tag{2.2}$$

Consider the *collapsed objective*, $\tilde{E}(y) = \min_x E(x, y)$. If $\tilde{E}(y)$ can be queried without representing x, we may perform *collapsed optimization* in the reduced basis of y, avoiding optimization in the larger basis of u (Eq. 2.1), or performing bi-level optimization (Eq. 2.2). However, $\tilde{E}$ is not usually available in closed form. We consider approximating $\tilde{E}(y)$ via supervised learning. In general, this would require solving $\tilde{E} = \min_x E(x, y)$ for each example y we wish to include in our training set.

This is the procedure used by many surrogate-based optimization techniques (Queipo et al., 2005; Forrester and Keane, 2009; Shahriari et al., 2015). The high cost of gathering each training example makes this prohibitive when x is high dimensional (and minimization is difficult) or when y is high dimensional (and supervised learning requires many examples). Compositional structure in E may assist us with approximating $\tilde{E}$. Many objectives may be represented as a sum:

$$E(x, y) = \sum_i E_i(x_i, y)\,. \tag{2.3}$$

Given this decomposition, x_i and x_j are conditionally independent given y; i.e., if we constrain x_i and y to take some values and perform minimization, the resulting x_j or $E_j(x_j, y)$ do not vary with the value chosen for x_i. This follows from the partial derivative structure $\frac{\partial E_i}{\partial x_j} = 0$ for $i \neq j$.

We propose to *learn* a collapsed objective $\tilde{E}$, which exploits conditional independence structure by representing $\tilde{E}(y) = \sum_i \tilde{E}_i(y)$. This representation as a sum allows us to use $\min_{x_i} E_i(x_i, y)$ as targets for supervision, which may be found more cheaply than performing a full minimization. The learned approximations to $\tilde{E}_i$ may be composed to form an energy function with larger domain.

The language we use to describe this decomposition is chosen to reflect the conceptual similarity of our framework to *collapsed variational inference* (Teh et al., 2007) and *collapsed Gibbs sampling* (Geman and Geman, 1984; Liu, 1994), in which conditional independence allows optimization or sampling to proceed in a collapsed space where nuisance random variables are marginalized out of the relevant densities. We exploit similar structure to these techniques, albeit in a deterministic setting. Other approaches to accelerating Eq. 2.2 which do not exploit (2.3) or directly model $\tilde{E}(y)$ include amortizing the inner optimization by predicting $x^*(y) = \arg\min_x E(x, y)$ (Kingma and Welling, 2014; Brock et al., 2017), or truncation of the inner loop, either deterministic (Wu et al., 2018; Shaban et al., 2018) or randomized to reduce bias (Tallec and Ollivier, 2017; Beatson and Adams, 2019).

The optimization procedure we accelerate is the simulation of mechanical materials, where the objective corresponds to a physically meaningful energy, and the conditional independence structure arises from spatial decomposition of the domain and spatial locality of the energy density. We believe this spatial decomposition of domain and energy could be generalized to learn collapsed energies for solving many other PDEs in reduced bases. This collapsed-basis approach may also be applicable to other bi-level optimization problems where the objective decomposes as a sum of local terms.

2.4 Mechanical meta-materials

Meta-materials are engineered materials with microstructure which results in macroscopic behavior not found in nature. The most popularly known are electromagnetic meta-materials such

as negative refraction index solids and "invisibility cloaks" which conceal an object through engineered distortion of electromagnetic fields (Poddubny et al., 2013; Cai and Shalaev, 2010). However, they also hold great promise in other domains: *mechanical* meta-materials use substructure to achieve unusual macroscopic behavior such as negative Poisson's ratio and nonlinear elastic responses; pores and lattices undergo reversible collapse under large deformation, enabling the engineering of complex physical affordances in soft robotics (Bertoldi et al., 2017). Meta-materials hold promise for modern engineering design but are challenging to simulate as the microstructure necessitates a very fine finite element mesh, and as the nonlinear response makes them difficult to approximate with a macroscopic material model. Most work on meta-materials has relied on engineers and scientists to hand-design materials, rather than numerically optimizing substructure to maximize some objective (Ion et al., 2016).

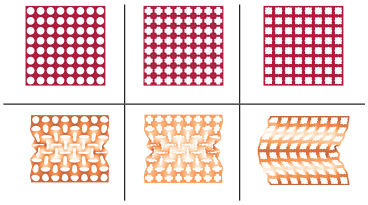


Figure 2.1: Cellular meta-materials. Top: at rest. Bottom: under compression, exhibiting periodic instability varying with pore shape. The left two structures exhibit negative Poisson's ratio, which does not occur in nature.

We focus on building surrogate models for the two-dimensional cellular solids investigated in Overvelde and Bertoldi (2014). These meta-materials consist of square "cells" of elastomer, each of which has a pore in its center. The pore shapes are defined by parameters α and β which characterize the pore shape in polar coordinates: $r(\theta) = r_0[1 + \alpha\cos(4\theta) + \beta\cos(8\theta)]$. The parameter r_0 is chosen such that the pore covers half the cell's volume: $r_0 = {}^{L_0}/\sqrt{\pi(2+\alpha^2+\beta^2}$. Constraints are placed on α and β to enforce a minimum material thicknesses and ensure that $\min_\theta r(\theta) > 0$ as in Overvelde and Bertoldi (2014).

These pore shapes give rise to complicated nonlinear elastic behavior, including negative Poisson's ratio and double energy wells (i.e., stored elastic energy which does not increase monotonically with strain). Realizations of this class of materials are shown under axial strain in Figure 2.1. The continuum mechanics behavior of these elastomer meta-materials can be captured by a neo-Hookean energy model (Ogden, 1997). Let $X \in \mathbb{R}^d$, where $d \leq 3$ in physical problems, be a point in the resting undeformed material reference configuration, and $u(X)$ be the displacement of this point from reference configuration. The stored energy in a neo-Hookean solid is $E = \int_\Omega W(u)dX$, where $W(u)$ is a scalar energy density over Ω, defined for bulk and shear moduli μ and κ as:

$$W = \frac{\mu}{2}\Big((\det F)^{-2/d}\mathrm{tr}(FF^T) - d\Big) + \frac{\kappa}{2}(\det F - 1)^2 \tag{2.4}$$

where F is the deformation gradient, $F(X) = \frac{\partial u(X)}{\partial X} + I$. Pores influence the structure of these

equations by changing the material domain Ω. These solids can be simulated by solving:

$$\text{Div } S = 0 \quad X \in \Omega \tag{2.5}$$

$$G(u) = 0 \quad X \in \partial\Omega \tag{2.6}$$

where $S = \frac{\partial W}{\partial F}$ is known as the first Piola-Kirchoff stress, and where Eq. 2.6 defines a boundary condition. E.g. $G(u) = u - u_b$ is a Dirichlet boundary condition; in our case, an externally imposed displacement. $G(u) = \frac{\partial W}{\partial u} - f_b$ corresponds to an external force exerting a pressure on the boundary.

To simulate these meta-materials, Eq. 2.5 is typically solved via finite element analysis. Solving with large meta-material structures is computationally challenging due to fine mesh needed to capture pore geometry and due to the nonlinear response induced by buckling under large displacements.

Solving the PDE in Eq. 2.5 corresponds to finding the u which minimizes the stored energy in the material subject to boundary conditions. That is, Eqs. 2.5 and 2.6 may be equivalently be expressed in an energy minimization form:

$$u = \arg\min \int_{X \in \Omega} W(u) dX \qquad \text{subject to } G(u) = 0 \in \partial\Omega \tag{2.7}$$

We use this form to learn surrogates which solve the PDE in a reduced basis of cell boundaries.

2.5 Composable energy surrogates

We apply the idea of learning collapsed objectives to the problem of simulating two-dimensional cellular mechanical meta-material behavior. The material response is determined by the displacement field u which minimizes the energy $\int_\Omega W dX$, subject to boundary conditions. We divide Ω into regular square subregions Ω_i, which we choose to be cells with 2×2 arrays of pores, and denote the intersection of the subregion boundaries with $\mathcal{B} = \partial\Omega_1 \cup \partial\Omega_2 \cup \ldots$ We let u_i be the restriction of u to Ω_i. We take the quantity of interest to be $u_\mathcal{B}$, the restriction of u to $\mathcal{B}$, and the nuisance variables to be the restriction of u to $\Omega \setminus \mathcal{B}$. The partitioning of Ω is shown in Figure 2.2.

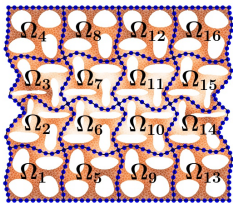


Figure 2.2: Meta-material domain Ω, partitioned into Ω_1 to Ω_{16}. Black lines show $\mathcal{B}$. Blue points are control points of splines used to represent $\tilde{u}$.

The total energy decomposes as a sum over regions:

$$E(u) = \int_{X \in \Omega} W(u) dX \qquad = \sum_i \int_{X \in \Omega_i} W(u_i) dX := \sum_i E(u_i)$$

Let $\tilde{u}_i$ be the restriction of u to $\partial\Omega_i$. Note $\partial\Omega_i = \mathcal{B} \cap \Omega_i$. Let the collapsed component energy be:

$$\tilde{E}_i(\tilde{u}_i) := \min_{u_i} E(u_i) \quad \text{subject to } u_i(X) = \tilde{u}_i(X) \quad X \in \partial\Omega_i \,.$$

This quantity is the lowest energy achievable by displacements of the *interior* of the cell Ω_i, given the boundary conditions specified by $\tilde{u}_i$ on $\partial\Omega_i$. $\tilde{E}_i(\tilde{u}_i)$ depends on the shape of the region Ω_i, i.e., on the geometry of the pores. Rather than each possible pore shape having a unique collapsed energy function, we introduce the pore shape parameter $\xi = (\alpha, \beta)$ as an argument, replacing $\tilde{E}_i(\tilde{u}_i)$ with $\tilde{E}(\tilde{u}_i, \xi_i)$. The macroscopic behavior of the material is fully determined by this *single* collapsed energy function $\tilde{E}(\tilde{u}_i, \xi_i)$. Given the true collapsed energy functions, we could accurately simulate material behavior in the reduced basis of the boundaries between each component Ω_i.[1]

We learn to approximate this collapsed energy function from data. This function may be duplicated and composed to simulate the material in the reduced basis $\mathcal{B}$, an approach we term *composable energy surrogates* (CESs). A single CES is trained to approximate the function $\tilde{E}$ by fitting to supervised data $(\tilde{u}_i, \xi_i, \tilde{E}(\tilde{u}_i, \xi_i))$, where ξ_i and $\tilde{u}_i$ may be drawn from any distribution corresponding to anticipated pore shapes and displacements, and the targets $\tilde{E}(\tilde{u}_i, \xi_i)$ are generated by solving the PDE in a small region Ω_i with geometry defined by ξ_i and with $\tilde{u}_i$ imposed as a boundary condition. This CES may be used to approximate the energy in multiple spatial locations: it may be "composed" to approximate the total energy of larger cellular meta-materials.

To efficiently solve for a reduced-basis displacement $u_\mathcal{B}$ on $\mathcal{B}$, we minimize the composed surrogate energy, $\hat{E}(u_\mathcal{B}) = \sum_i \hat{E}(\tilde{u}_i, \xi_i)$, where $\hat{E}(\tilde{u}_i, \xi_i)$ is the model's prediction of $\tilde{E}(\tilde{u}_i, \xi_i)$, the collapsed energy of one component. Training CES which produce accurate reduced-basis solutions may be thought of as a highly-structured imitation learning problem. A sufficient condition for finding the correct minimum is for the "action" taken by the surrogate—the derivative of the energy approximation $\nabla_{u_\mathcal{B}} \hat{E}$—to match the "action" taken by an expert—the *total* derivative, $\nabla_{u_\mathcal{B}} \min_{u \notin \mathcal{B}} E(u)$—along the optimization trajectory. If so, the surrogate will follow the trajectory of a valid, if non-standard, bilevel gradient-based procedure for minimizing the energy, corresponding to (2.2). Given an imperfect surrogate, the error in the final solution will depend on the error in approximating $\nabla_{u_\mathcal{B}} \min_{u \notin \mathcal{B}} E(u)$ with $\nabla_{u_\mathcal{B}} \hat{E}$ along the trajectory. This observation informs our model, training, and data collection procedures, described in the following sections.

[1] So long as forces and constraints are only applied on $\mathcal{B}$.

2.6 Model architecture

Our CESs take the form of a neural architecture, designed to respect known properties of the true potential energy and to maximize usefulness as surrogate energy to be minimized via a gradient-based procedure. The effects of these design choices are quantified via an ablation study in the appendix.

Reduced-basis parameterization. We use one cubic spline for each horizontal and vertical displacement function along each face of the square, with evenly spaced control points and "not-a-knot" boundary conditions. Our vector representation of $\tilde{u}$ is $\mathbf{u} \in \mathbb{R}^{2n}$, formed from the horizontal and the vertical displacement values at each of the n control points. Splines on adjacent faces share a control point at the corner. Using N control points to parameterize the function along each face requires $n = 4 * (N-1)$ control points to parameterize a $1d$ function around a single cell. For all experiments we use $N = 10$ control points along each edge, resulting in $\mathbf{u} \in \mathbb{R}^{72}$.

Model structure and loss. Our model structure and losses are shown below. In the energy model $\hat{E}$, f_ϕ is a neural network with parameters ϕ and $\mathcal{R}$ removes rigid-body rotation and translation. Our loss function is $\mathcal{L} = \mathcal{L}^0 + \mathcal{L}^1 + \mathcal{L}^2$, which is a weighted sum of losses on the 0th, 1st and 2nd energy derivatives. $\nabla_{\mathbf{u}}$ and $\nabla^2_{\mathbf{u}}$ are the gradient and Hessian of the surrogate energy $\hat{E}$ or the ground-truth energy $\tilde{E}$ with respect to $\mathbf{u}$, and v is sampled independently for each training example in a batch.

$$\hat{E}(\mathbf{u}, \xi) = \underbrace{||\mathcal{R}(\mathbf{u})||_2^2}_{\text{Linear elastic component}} \underbrace{\exp\{f_\phi(\mathcal{R}(\mathbf{u}), \xi)\}}_{\text{Stiffness}}, \qquad \mathcal{L}^0 = \underbrace{\left\| f_\phi(\mathcal{R}(\mathbf{u}), \xi) - \log \frac{\tilde{E}(\tilde{u})}{||\mathcal{R}(\mathbf{u})||_2^2} \right\|_2^2}_{\text{Log-stiffness loss}},$$

$$\mathcal{L}^1 = \underbrace{1 - \frac{\langle \nabla_{\mathbf{u}} \hat{E}, \nabla_{\mathbf{u}} \tilde{E} \rangle}{||\nabla_{\mathbf{u}} \hat{E}|| ||\nabla_{\mathbf{u}} \tilde{E}||}}_{\text{Cosine distance between gradients}}, \qquad \mathcal{L}^2 = \underbrace{1 - \frac{\langle \nabla^2_{\mathbf{u}} \hat{E} v, \nabla^2_{\mathbf{u}} \tilde{E} v \rangle}{||\nabla^2_{\mathbf{u}} \hat{E} v|| ||\nabla^2_{\mathbf{u}} \tilde{E} v||}}_{\substack{\text{Cosine distance between} \\ \text{Hessian-vector products}}} \qquad \underbrace{v \sim \mathcal{N}(0, I^{2n})}_{\text{Projection vector for Hessian}}.$$

Invariance to rigid body transforms. The true elastic energy is invariant to rigid body transforms of a solid. This invariance may be hard to learn exactly from data. We use a module $\mathcal{R}$ which applies *Procrustes analysis*, i.e. finds and applies the rigid body transform which minimizes the Euclidean distance to a reference (we use the rest configuration). This is differentiable and closed-form.

Encoding a linear elastic bias. The energy is approximated well by a linear elastic model when at rest: $\tilde{E}^i(\tilde{u}_i) \approx \mathcal{R}(\mathbf{u}_i)^T A^i \mathcal{R}(\mathbf{u}_i)$ for a stiffness matrix A^i depending on ξ_i. We scale our net's outputs by $||\mathcal{R}(\mathbf{u}_i)||_2^2$ so that it needs only capture a "scalar stiffness" $E/||\mathcal{R}(\mathbf{u}_i)||_2^2$ accounting for the geometry of A^i given ξ_i and for deviation from the linear elastic model.

Parameterizing the log-stiffness. The energy of a component $\tilde{E}^i(u_{0,i})$ is nonnegative, and the ratio of energy to a linear elastic approximation varies over many orders of magnitude. We

parameterize the log of the scalar stiffness with our neural network f_ϕ rather than the stiffness.

Log-stiffness loss. We wish to find neural network parameters ϕ which lead to accurate energy predictions for many different orders of magnitude of energy and displacement. Minimizing the ℓ^2 loss between predicted and true energies penalizes errors in predicting large energies more than proportional errors predicting small energies. Instead, we take the ℓ^2 loss between the predicted log-stiffness $f_\phi(\mathcal{R}(\mathbf{u}), \xi)$ and the effective ground-truth log-stiffness, $\log {}^{\tilde{E}(\tilde{u})}/_{||\mathcal{R}(\mathbf{u})||_2^2}$.

Sobolev training with gradients and Hessian-vector products. "Sobolev training" on derivatives of a target function can aid generalization (Czarnecki et al., 2017). Accuracy of CES' derivatives is crucial, so we Sobolev train on energy gradients and Hessians. We obtain ground-truth gradients cheaply via the adjoint method (Lions, 1971). Given a solution u_i to the PDE in Ω_i with boundary conditions $\tilde{u}_i$, the gradient $\nabla_{\tilde{u}_i}\tilde{E}_i(\tilde{u}_i)$ requires solving a linear system with the same cost as one Newton step of solving the PDE (Mitusch et al., 2019). The spline is a linear map $\mathcal{M}$ from $\mathbf{u}_i$ to $\tilde{u}_i$ in the finite element basis, so $\nabla_{\mathbf{u}_i}\tilde{E}_i(\tilde{u}_i) = \mathcal{M}^T\nabla_{\tilde{u}_i}\tilde{E}_i(\tilde{u}_i)$. The surrogate gradient, $\nabla_{\mathbf{u}_i}\hat{E}_\phi(\mathbf{u}_i, \xi_i)$, is computed with one backward pass. Given solution and gradient, we compute $\nabla^2_{\mathbf{u}}\tilde{E}$ with one linear solve per entry of $\mathbf{u}$. As $\mathbf{u} \in \mathbb{R}^{72}$ and many more than 72 Newton steps are usually needed to solve the PDE, this does not dominate the cost of data collection. Computing the full Hessian of the surrogate energy, $\nabla^2_{\mathbf{u}_i}\hat{E}_\phi(\mathbf{u}_i, \xi_i)$, would require $2n$ backward passes. Instead we train on Hessian-vector products, which require only one additional backward pass.

Cosine distance loss for Sobolev training. Energy gradient and Hessian values vary over many orders of magnitude, with higher energies leading to larger derivatives. We wish our model to be accurate across a range of operating conditions. Rather than placing an ℓ^2 loss on the gradient and Hessian-vector products as in Czarnecki et al. (2017), we minimize the cosine distance between ground truth and approximate gradients and Hessians, which is naturally bounded in $[0, 1]$.

2.7 Data and training

Data collection has two phases. First, we collect training and validation datasets using Hamiltonian Monte Carlo (Duane et al., 1987) to preferentially sample displacements which correspond to lower energy modes. Next, we perform dataset aggregation (Ross et al., 2011) to augment the dataset so that the surrogate will be accurate on states encountered when deployed. We provide details of the hardware and the software packages used in the appendix.

Solving the PDE. To collect training data, we use the reduced-basis displacement $\tilde{u}$ corresponding to a vector of spline coefficients $\mathbf{u}$ as the boundary condition around a domain Ω representing a

2×2-pore subdomain, and solve the PDE using a load-stepped relaxed Newton's method (Sheng et al., 2002). The relaxed Newton's method takes the iteration $\vec{u} \leftarrow \vec{u} - \lambda(\frac{\partial^2 E}{\partial \vec{u}^2})^{-1} \frac{\partial E}{\partial \vec{u}}$. Here, $0 < \lambda < 1$ is the relaxation parameter (analogous to a step size), and $\vec{u}$ is the vector of coefficients defining u in the FEA basis. Newton's method requires an initial guess which is sufficiently close to the true solution (Kythe et al., 2004). Smaller relaxation parameters yield a greater radius of convergence but necessitate more steps to solve the PDE.

The radius of convergence can also be aided by load-stepping: solving the PDE for a sequence of boundary conditions, annealing from an initial boundary condition for which we have a good initial guess (e.g., the rest configuration) to a final boundary condition $\tilde{u}$, using the solution to the previous problem as an initial guess for Newton's method for the next problem. We find that combining load stepping with a relaxed Newton's method is more efficient than using either alone. Except where specified, we linearly anneal from rest to $\tilde{u}$ over 10 load steps and use a relaxation parameter $\lambda = 0.1$.

Initial dataset collection. We wish to train on varied displacement boundary conditions. As solution procedures minimize energy, lower energy modes will be encountered in the solve. We choose a distribution with density the product of a Boltzmann density $\exp\{\tilde{E}\}/Z$ and a Gaussian density $\mathcal{N}(\bar{x}(\mathbf{u}); \bar{\mu}, \Sigma)$, where $\bar{x}(\mathbf{u}) \in \mathbb{R}^{2\times 2}$ is a macroscopic strain tensor[2] corresponding to $\mathbf{u}$, $\bar{\mu}$ is a target strain drawn from an i.i.d. Gaussian with standard deviation 0.15, and Σ is set to $(\bar{\mu} \circ \bar{\mu})^{-1}$.

Given a solution to the PDE, the log-density and its displacement may be cheaply computed (the latter via the adjoint method). Making use of these gradients, we sample data points with Hamiltonian Monte Carlo (HMC). After sampling a data point, we compute the corresponding Hessian and save the tuple $(\mathbf{u}, \xi, \tilde{E}, \nabla_{\mathbf{u}}\tilde{E}, \nabla^2_{\mathbf{u}}\tilde{E})$ as a data point.

We initialize each HMC data collector by sampling a macroscopic displacement target and a random pore shape. We do not use load-stepping, instead using the solution for the $\mathbf{u}$ used in the previous iteration of HMC's leapfrog integration as an initial guess for solving the PDE. We randomize HMC hyperparameters for each collector to attempt to minimize the impact of specific settings: see the appendix for exact ranges. We sample 55000 training examples and 5000 validation examples altogether. We visualize displacements drawn from this distribution in the appendix.

Data aggregation. Surrogate deployment defies standard i.i.d. assumptions in supervised learning. The deployed surrogate encounters states determined by the energy it defines and by boundary conditions on the composed body. Given a dataset such as that we sampled with HMC, the distribution over states encountered by the surrogate in deployment may be very different to the distribution of states in this dataset.

[2]See the appendix for approximating $\bar{x}$ from $\mathbf{u}$.

This problem—that training an agent to predict expert actions can lead to trajectories dissimilar to those on which it was trained—is a central concern in the imitation learning literature. A number of solutions exist (Schroecker and Isbell, 2017). One is dataset aggregation, or DAgger (Ross et al., 2011), which reduces imitation learning or structured prediction to online learning.

In DAgger, a policy is deployed and trajectories are collected. The expert is queried on the states in these trajectories. The state-action pairs are appended to the dataset, and the policy is retrained on this dataset. This process of deployment, querying, appending data, and retraining, is iterated. The distribution of states encountered in deployment and the distribution of states in the dataset converge. Under appropriate assumptions, the instantaneous regret of the learned policy vanishes with the number of iterations, i.e., the learned policy matches the expert policy on its own trajectories.

Ross et al. (2011) present DAgger as a method for discrete action spaces. We have a continuous action space: the gradient of the energy in a cell. We do not investigate generalizing DAgger's regret guarantees to continuous action spaces, but the intuition holds that we wish our model to "imitate" the finite element "expert" on the optimization trajectories the model produces.

We initialize our training data with HMC as described earlier. We then apply DAgger by iterating: (i) training the surrogate; (ii) composing surrogates and finding displacements which minimize the composed energy; (iii) sampling displacements along the surrogate's solution path, querying the ground-truth energy and energy derivatives using FEA, and adding these new data points to the dataset. We visualize displacements generated by DAgger in the appendix.

2.8 Software and hardware

We implement the finite element models in `dolfin` (Logg and Wells, 2010; Logg et al., 2012c), a Python front end to FEniCS (Alnæs et al., 2015a; Logg et al., 2012b). To differentiate through finite element solutions, we use the package `dolfin-adjoint` (Mitusch et al., 2019). We implement surrogate models in PyTorch (Paszke et al., 2019).

We use Ray (Moritz et al., 2018) to run distributed workloads on Amazon EC2. The initial dataset is collected using 80 M4.xlarge CPU spot workers. While training the surrogate, we use a GPU P3.large driver node to train the model, and 80 M4.xlarge CPU spot worker nodes performing DAgger in parallel. These workers receive updated surrogate model parameters, compose and deploy the surrogate, sample displacements along the solution path, query the finite element model for energy and derivatives, and return data to the driver node. Initial dataset collection and model

training with DAGGER each take about one day in wall-clock time.

2.9 Empirical evaluation

We demonstrate the ability of Composable Energy Surrogates (CES) to efficiently produce accurate solutions. We consider the systems constructed in Overvelde and Bertoldi (2014): structures with an 8×8 array of pores, corresponding to a 4×4 assembly of our surrogates, each representing a 2×2-pore component. We sample pore shapes from a uniform distribution over valid shapes defined in Overvelde and Bertoldi (2014). For DAGGER, we sample vertical axial strain magnitudes from $\mathcal{U}(0., 0.3)$, and apply compression with probability 0.8 (as compressive displacements involve more interesting pore collapse) or tension with probability 0.2.

We compare our composed surrogates to finite element analysis with different-fidelity meshes under axial compression and tension with a macroscopic displacement of $0.125L_0$, where L_0 is the original length of the solid. See the appendix for details of the finite element meshes. We use seven pore shapes: $\xi = (0, 0)$, corresponding to circular pores, and six ξ sampled from a uniform distribution over pore parameters defined as valid in Overvelde and Bertoldi (2014).

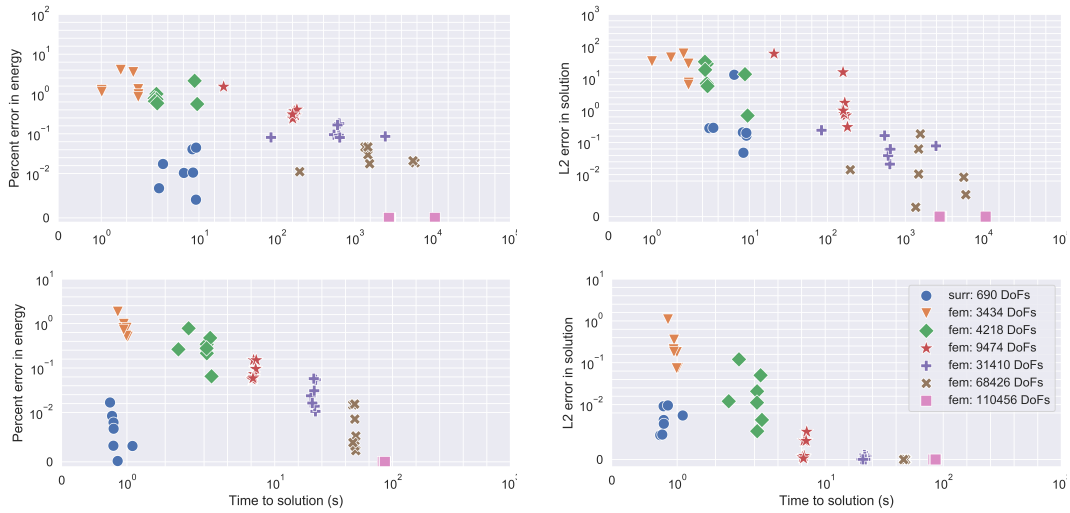


Figure 2.3: Error in solution and in estimated energy vs solution wall clock time for the composed energy surrogate and for finite element models with varying mesh sizes. Top: axial compression. Bottom: axial tension.

We use PyTorch's L-BFGS routine to minimize the composed surrogate energy, with step size 0.25 and default criteria for checking convergence. We attempt to solve each finite element model with FEniCS' Newton method with $[1, 2, 5, 10, 20]$ load steps and relaxation parameters $[0.9, 0.7, 0.4, 0.1, 0.05]$, and record time taken for the *fastest* convergent solve. Under compression these solids exhibit nonlinear behavior, and only more conservative solves converge. Under tension

they behave closer to a linear elastic model, and Newton's method converges quickly. Measurements are taken on an AWS M4.xlarge EC2 CPU instance. Using a GPU could provide further acceleration.

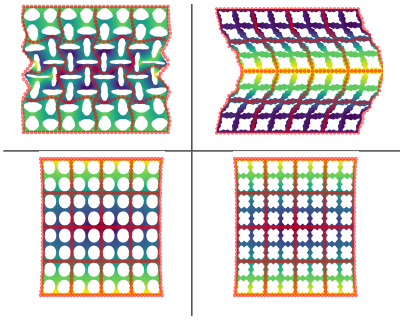


Figure 2.4: Meta-materials under compression (top) and tension (bottom), with solution found via CES shown in red at spline control points.

We measure error in the solution and in the macroscopic energy. The former is $||\hat{u} - u^*||_2^2$, where $\hat{u}$ and u^* are the approximation and ground-truth evaluated at spline control points. The latter is the relative error $|\hat{E}(\hat{u}) - E^*(u^*)|/E^*(u^*)$, where $\hat{E}(\hat{u})$ is the approximated energy of the approximate solution, and $E^*(u^*)$ is the ground-truth energy of the ground-truth solution. As the energy function determines behavior, accuracy of energy is a potential indicator of ability to generalize to larger structures. The highest-fidelity finite element model is taken as ground truth, and thus has an error of zero on both metrics. Multiple minimizers exist as energy is preserved under rigid body transforms, so before comparing a solution $\hat{u}$ to the ground-truth u^* we check each vertical and horizontal flip and use the flip which minimizes the solution error.

Figure 2.3 shows our evaluation. Composed energy surrogates are more efficient than high-fidelity FEA simulations yet more accurate than low-fidelity simulations. CES produces solutions with equivalent ℓ^2 error to FEA solutions which use an order of magnitude more variables or computation time, and with an order of magnitude less ℓ^2 error than FEM solutions requiring the same computation. This gap increases to several orders of magnitude when we consider percentage error in the predicted strain energy. We visualize the ground-truth and the CES approximation in Figure 2.4. See the appendix for visualization of FEM and CES solutions for the remaining structures.

2.10 Limitations and opportunities

Use of DAgger. We use DAgger to help CES match the ground-truth on the states encountered during the solution trajectory. This requires one to specify in advance the conditions under which the surrogate will be deployed. Investigating CES' ability to generalize to novel deployment conditions–and designing surrogates which can do so effectively–is an important direction for future work.

Error estimation, refinement, and guarantees. Finite element methods permit a straightforward way to estimate the error (compare to the solution in a more-refined basis) and control it (via refinement). CES currently lacks these properties.

Finite element baseline. There is an immense body of work on finite element methods and

iterative solvers. We provide a representative baseline, but our work should not be taken as a comparison with the "state-of-the-art". We show that composable machine-learned energy surrogates enjoy advantages over a reasonable baseline, and hold promise for scalable amortization of solving modular PDEs.

Hyperparameters. Both our method and the finite element baseline rely on a multitude of hyperparameters: the size of the spline reduced basis; the size and learning rate of the neural network; the size and degree of the finite element approximation; and the specific variant of Newton's method to solve the finite element model. We do not attempt a formal, exhaustive search over these parameters.

Known structure. We leave much fruit on the vine in terms of engineering structure known from the into our surrogate. For example, one could also use a more expressive normalizer than $||\mathbf{u}||_2^2$, e.g. the energy predicted by a coarse-grained linear elastic model, or exploit spatially local correlation, e.g. by using a 1-d convolutional network around the boundary of the cell.

2.11 Conclusion

We present a framework for collapsing optimization problems with local bilevel structure by learning composable energy surrogates. This framework is applied to amortizing the solution of PDEs corresponding to mechanical meta-material behavior. Learned composable energy surrogates are more efficient than high-fidelity FEA yet more accurate than low-fidelity FEA, occupying a new point on the Pareto frontier. We believe that these surrogates could accelerate meta-material design, as well as design and identification of other systems described by PDEs with parametric modular structure.

Chapter 3

Efficient optimization of loops and limits with randomized telescoping sums

We consider optimization problems in which the objective requires an inner loop with many steps or is the limit of a sequence of increasingly costly approximations. Meta-learning, training recurrent neural networks, and optimization of the solutions to differential equations are all examples of optimization problems with this character. In such problems, it can be expensive to compute the objective function value and its gradient, but truncating the loop or using less accurate approximations can induce biases that damage the overall solution. We propose *randomized telescope* (RT) gradient estimators, which represent the objective as the sum of a telescoping series and sample linear combinations of terms to provide cheap unbiased gradient estimates. We identify conditions under which RT estimators achieve optimization convergence rates independent of the length of the loop or the required accuracy of the approximation. We also derive a method for tuning RT estimators online to maximize a lower bound on the expected decrease in loss per unit of computation. We evaluate our adaptive RT estimators on a range of applications including meta-optimization of learning rates, variational inference of ODE parameters, and training an LSTM to model long sequences.

3.1 Introduction

Many important optimization problems consist of objective functions that can only be computed iteratively or as the limit of an approximation. Machine learning and scientific computing provide many important examples. In meta-learning, evaluation of the objective typically requires the training of a model, a case of bi-level optimization. When training a model on sequential data or to make decisions over time, each learning step requires looping over time steps. More broadly, in many scientific and engineering applications one wishes to optimize an objective that is defined as the limit of a sequence of approximations with both fidelity and computational cost increasing according to a natural number $n \geq 1$. Inner-loop examples include: integration by Monte Carlo or quadrature with n evaluation points; solving ordinary differential equations (ODEs) with an Euler or Runge Kutta method with n steps and $\mathcal{O}(\frac{1}{n})$ step size; and solving partial differential equations (PDEs) with a finite element basis with size or order increasing with n.

Whether the task is fitting parameters to data, identifying the parameters of a natural system, or optimizing the design of a mechanical part, in this work we seek to more rapidly solve problems in which the objective function demands a tradeoff between computational cost and accuracy. We formalize this by considering parameters $\theta \in \mathbb{R}^D$ and a loss function $\mathcal{L}(\theta)$ that is the uniform limit of a sequence $\mathcal{L}_n(\theta)$:

$$\min_\theta \mathcal{L}(\theta) = \min_\theta \lim_{n\to\infty} \mathcal{L}_n(\theta)\,. \tag{3.1}$$

In some problems there may be a finite $n = H$ (horizon) that achieves the limit. We also introduce a cost function $C : \mathbb{N}_+ \to \mathbb{R}$ that is nondecreasing in n to represent the cost of computing $\mathcal{L}_n$ and its gradient.

A principal challenge of optimization problems with the form in Eq.3.1 is selecting a finite N such that the minimum of the surrogate $\mathcal{L}_N$ is close to that of $\mathcal{L}$, but without $\mathcal{L}_N$ (or its gradients) being too expensive. Choosing a large N can be computationally prohibitive, while choosing a small N may bias optimization. Meta-optimizing learning rates with truncated horizons can choose wrong hyperparameters by orders of magnitude Wu et al. (2018). Truncating backpropogation through time for recurrent neural networks (RNNs) favors short term dependencies Tallec and Ollivier (2017). Using too coarse a discretization to solve an ODE or PDE can cause error in the solution and bias outer-loop optimization. These optimization problems thus experience a sharp trade off between efficient computation and bias.

We propose *randomized telescope* (RT) gradient estimators, which provide cheap unbiased gradient estimates to allow efficient optimization of these objectives. RT estimators represent the objective or its gradients as a telescoping series of differences between intermediate values, and draw weighted samples from this series to maintain unbiasedness while balancing variance and expected computation.

The paper proceeds as follows. Section 2 introduces RT estimators and their history. Section 3 formalizes using RT estimators for optimization, and discusses related work in optimization. Section 4 proves RT estimators can achieve optimization guarantees for loops and limits. Section 5 discusses designing RT estimators by maximizing a bound on expected improvement per unit of computation. Section 6 describes practical considerations adapting RT estimators online. Section 7 presents experimental results. Section 8 discusses limitations and future work. Appendix A presents algorithm pseudocode. Appendix B presents proofs.

3.2 Unbiased randomized truncation

In this section, we discuss the general problem of estimating limits through randomized truncation. The first subsection presents the randomized telescope family of unbiased estimators, while the second subsection describes their history (dating back to von Neumann and Ulam). In the following sections, we will describe how this technique can be used to provide cheap unbiased gradient estimates and accelerate optimization for many problems.

3.2.1 Randomized telescope estimators

Consider estimating any quantity $Y_H := \lim_{n \to H} Y_n$ for $n \in \mathbb{N}_+$ where $H \in \mathbb{N}_+ \cup \{\infty\}$. Assume that we can compute Y_n for any finite $n \in \mathbb{N}_+$, but since the cost is nondecreasing in n there is a point at which this becomes impractical. Rather than truncating at some fixed value short of the limit, then, we may find it useful to construct an unbiased estimator of Y_H and take on some randomness in return for reduced computational cost.

Define the backward difference Δ_n and represent the quantity of interest Y_H with a telescoping series:

$$Y_H = \sum_{n=1}^{H} \Delta_n \qquad \text{where} \quad \Delta_n = \begin{cases} Y_n - Y_{n-1} & n > 1 \\ Y_1 & n = 1 \end{cases}.$$

We may sample from this telescoping series to provide unbiased estimates of Y_H, introducing variance

to our estimator in exchange for reducing expected computation. We use the name *randomized telescope* (RT) to refer to the family of estimators indexed by a distribution q over the integers $1, \ldots, H$ and a weight function $W(n, N)$:

$$\hat{Y}_H = \sum_{n=1}^{N} \Delta_n W(n, N) \qquad N \in \{1, \ldots, H\} \sim q\,. \tag{3.2}$$

Proposition 3.2.1. ***Unbiasedness of RT estimators.*** *The RT estimators in (3.2) are unbiased estimators of Y_H as long as*

$$\mathbb{E}_{N\sim q}[W(n, N)] = \sum_{N=n}^{H} W(n, N) q(N) = 1 \quad \forall n\,. \tag{3.3}$$

See Appendix B for a short proof. Although we are coining the term "randomized telescope" to refer to the family of estimators with the form of Eq. 3.2, the underlying trick has a long history, discussed in the next section. The literature we are aware of focusses on one or both of two special cases of Eq. 3.2, defined by choice of weight function $W(n, N)$. We will also focus on these two variants of RT estimators, but we observe that there is a larger family.

Most related work uses the "Russian roulette" estimator originally discovered and named by von Neumann and Ulam Kahn (1955), which we term *RT-RR* and has the form

$$W(n, N) = \frac{1}{1 - \sum_{n'=1}^{n-1} q(n')} \mathbb{1}\{N \geq n\}\,. \tag{3.4}$$

It can be seen as unrolling and summing the iterates Δ_n while flipping a biased coin at each new iterate Δ_n. With probability $q(n)$, the series is truncated at term n and the unrolling stops. With probability $1 - q(n)$, the process continues, and all future terms are upweighted by $\frac{1}{1-q(n)}$ to maintain unbiasedness.

The other important special case of Eq. 3.2 is the "single sample" estimator *RT-SS*, referred to as "single term weighted truncation" in Lyne et al. (2015). RT-SS takes

$$W(n, N) = \frac{1}{q(N)} \mathbb{1}\{n = N\}\,. \tag{3.5}$$

This is directly importance sampling the differences Δ_n.

We will later prove conditions under which RT-SS and RT-RR should be preferred. Of all estimators in the form of (3.2) which obey proposition 3.2.1 and for all q, RT-SS minimizes the worst-case variance across an adversarial choice of diagonal covariances $\mathrm{Cov}(\Delta_i, \Delta_j)$. Within the

same family, RT-RR achieves minimum variance when Δ_i and Δ_j are independent for all i, j.

3.2.2 A brief history of unbiased randomized truncation

The essential trick—unbiased estimation of a quantity via randomized truncation of a series—dates back to unpublished work from John von Neumann and Stanislaw Ulam. They are credited for using it to develop a Monte Carlo method for matrix inversion in Forsythe and Leibler (1950), and for a method for particle diffusion in Kahn (1955).

It has been applied and rediscovered in a number of fields and applications. The early work from von Neumann and Ulam led to its use in computational physics, in neutron transport problems (Spanier and Gelbard, 1969), for studying lattice fermions (Kuti, 1982), and to estimate functional integrals (Wagner, 1987). In computer graphics Arvo and Kirk (1990) introduced its use for ray tracing; it is now widely used in rendering software. In statistical estimation, it has been used for estimation of derivatives (Rychlik, 1990), unbiased kernel density estimation (Rychlik, 1995), doubly-intractable Bayesian posterior distributions (Girolami et al., 2013; Lyne et al., 2015; Wei and Murray, 2016), and unbiased MCMC (Jacob et al., 2017).

The underlying trick has been rediscovered by Fearnhead et al. (2008) for unbiased estimation in particle filtering, by McLeish (2010) for debiasing Monte Carlo estimates, by Rhee and Glynn (2012, 2015) for unbiased estimation in stochastic differential equations, and by Tallec and Ollivier (2017) to debias truncated backpropagation. The latter also uses RT estimators for optimization; however, it only considers fixed "Russian roulette"-style randomized telescope estimators and does not consider convergence rates or how to adapt the estimator online (our main contributions).

3.3 Optimizing loops and limits

In this paper, we consider optimizing functions defined as limits, where loops are an important special case. Consider a problem where, given parameters θ we can obtain a series of approximate losses $\mathcal{L}_n(\theta)$, which converges uniformly to some limit $\lim_{n\to H} \mathcal{L}_n := \mathcal{L}$, for $n \in \mathbb{N}_+$ and $H \in \mathbb{N}_+ \cup \{\infty\}$. We assume the sequence of gradients with respect to θ, denoted $G_n(\theta) := \nabla_\theta \mathcal{L}_n(\theta)$ converge uniformly to a limit $G(\theta)$. Under this uniform convergence and assuming convergence of $\mathcal{L}_n$, we have $\lim_{n\to H} \nabla_\theta \mathcal{L}_n(\theta) = \nabla_\theta \lim_{n\to H} \mathcal{L}_n(\theta)$ (see Theorem 7.17 in Rudin et al. (1976)), and so $G(\theta)$ is indeed the gradient of our objective $\mathcal{L}(\theta)$. We assume there is some computation cost $C(n)$ associated with evaluating $\mathcal{L}_n$ or G_n, which is nondecreasing with n, and we wish to efficiently minimize $\mathcal{L}$ with respect to θ. Loops are a special case of this framework, where $\mathcal{L}_n$ is the loss resulting from running

the loop for some number of steps increasing in n.

3.3.1 Randomized telescopes for optimization

We propose using randomized telescopes to provide a stochastic gradient estimator for such optimization problems. Our aim is to accelerate optimization in much the same manner as minibatch stochastic gradient descent accelerates optimization for large datasets: using sampling to decrease the expected computation cost of each optimization step, at the price of increasing variance in the gradient estimates, without introducing bias.

Consider the gradient $G(\theta) = \lim_{n\to H} G_n(\theta)$, and the backward difference $\Delta_n(\theta) = G_n(\theta) - G_{n-1}(\theta)$, where $G_0(\theta) = 0$, so that $G(\theta) = \sum_{n=1}^{H} \Delta_n(\theta)$. We use the randomized telescope estimator

$$\hat{G}(\theta) = G_N(\theta) = \sum_{n=1}^{N} \Delta_n(\theta) W(n, N) \tag{3.6}$$

where $N \in \{1, 2, \ldots, H\}$ is drawn according to a proposal distribution q, and together W and q satisfy proposition 3.2.1.

Note that due to linearity of differentiation, and letting $\mathcal{L}_0(\theta) := 0$, we have

$$\sum_{n=1}^{N} \Delta_n(\theta) W(n, N) = \nabla_\theta \sum_{n=1}^{N} (\mathcal{L}_n(\theta) - \mathcal{L}_{n-1}(\theta)) W(n, N)\,.$$

Thus, when computing $\mathcal{L}_n(\theta)$ can reuse most of the computation performed for computing $\mathcal{L}_{n-1}(\theta)$, we can evaluate $\hat{G}_N(\theta)$ via forward or backward automatic differentiation with cost approximately equal to computing $G_N(\theta)$, i.e., $\hat{G}_N(\theta)$ has computation cost $\approx C(N)$. This most often occurs when evaluating $\mathcal{L}_n(\theta)$ involves an inner loop and the step size used for the inner loop does not change with n, such as in meta-learning and training RNNs. When computing $\mathcal{L}_n(\theta)$ does not reuse computation, e.g., when solving an ODE or PDE where n describes how fine a discretization to use, evaluating $\hat{G}_N(\theta)$ requires separately computing $\mathcal{L}_n(\theta)$ for all $n \leq N$ for which $W(n, N) \neq 0$, i.e., $\hat{G}_N(\theta)$ has computation cost $\sum_{n=1}^{N} C(n) \mathbb{1}\{W(n, N) \neq 0\}$.

3.3.2 Related work in optimization

Gradient-based bilevel optimization has seen extensive work in literature. See Jameson (1988) for an early example of optimizing implicit functions, Christianson (1998) for a mathematical treatment, and Maclaurin et al. (2015); Franceschi et al. (2017) for recent treatments in machine learning. Shaban et al. (2018) propose truncating only the backward pass by only backpropagating through

the final few optimization steps to reduce memory requirements. Metz et al. (2018) propose linearly increasing the number of inner steps over the course of the outer optimization.

An important case of bi-level optimization is optimization of architectures and hyperparameters. Truncation causes bias, as shown by Wu et al. (2018) for learning rates and by Metz et al. (2018) for neural optimizers.

Bi-level optimization is also used for meta-learning across related tasks (Schmidhuber, 1987; Bengio et al., 1992). Ravi and Larochelle (2016) train an initialization and optimizer, and Finn et al. (2017) only an initialization, to minimize validation loss. The latter paper shows increasing performance with the number of steps used in the inner optimization. However, in practice the number of inner loop steps must be kept small to allow training over many tasks.

Bi-level optimization can be accelerated by amortization. Variational inference can be seen as bi-level optimization; variational autoencoders (Kingma and Welling, 2014) amortize the inner optimization with a predictive model of the solution to the inner objective. Recent work such as Brock et al. (2018); Lorraine and Duvenaud (2018) amortizes hyperparameter optimization in a similar fashion.

However, amortizing the inner loop induces bias. Cremer et al. (2018) demonstrate this in VAEs, while Kim et al. (2018) show that in VAEs, combining amortization with truncation by taking several gradient steps on the output of the encoder can reduce this bias. This shows these techniques are orthogonal to our contributions: while fully amortizing the inner optimization causes bias, predictive models of the limit can accelerate convergence of $\mathcal{L}_n$ to $\mathcal{L}$.

Our work is also related to work on training sequence models. Tallec and Ollivier (2017) use the Russian roulette estimator to debias truncated backpropagation through time. They use a fixed geometrically decaying $q(N)$, and show that this improves validation loss for Penn Treebank. They do not consider efficiency of optimization, or methods to automatically set or adapt the hyperparameters of the randomized telescope. Trinh et al. (2018) learn long term dependencies with auxiliary losses. Other work accelerates optimization of sequence models by replacing recurrent models which require backpropagation through time with models which use convolution or attention (Vaswani et al., 2017), which can be trained more efficiently.

3.4 Convergence rates with fixed RT estimators

Before considering more complex large-scale problems, we examine the simple RT estimator for stochastic gradient descent on convex problems. We assume that the sequence $\mathcal{L}_n(\theta)$ and units

for C are chosen such that $C(n) = n$. We study RT-SS, with $q(n)$ fixed *a priori*. We consider optimizing parameters $\theta \in \mathcal{K}$, where $\mathcal{K} \subset \mathcal{R}^d$ is a bounded, convex and compact set with diameter bounded by D. We assume $\mathcal{L}(\theta)$ is convex in θ, and $G_n(\theta)$ converge according to $||\Delta_n||_2 \leq \psi_n$, where ψ_n converges polynomially or geometrically. The quantity of interest is the instantaneous regret, $R_t = \mathcal{L}(\theta_t) - \min_\theta \mathcal{L}(\theta)$, where θ_t is the parameter after t steps of SGD.

In this setting, any fixed truncation scheme using $\mathcal{L}_N$ as a surrogate for $\mathcal{L}$, with fixed $N < H$, cannot achieve $\lim_{t\to\infty} R_t = 0$. Meanwhile, the fully unrolled estimator has computation cost which scales with H. In the many situations where $H \to \infty$, it is impossible to take even a single gradient step with this estimator.

The randomized telescope estimator overcomes these drawbacks by exploiting the fact that G_n converges according to $||\Delta_n||_2 \leq \psi_n$. As long as q is chosen to have tails no lighter than ψ_n, for sufficiently fast convergence, the resulting RT-SS gradient estimator achieves asymptotic regret bounds invariant to H in terms of convergence rate.

All proofs are deferred to Appendix B. We begin by proving bounds on the variance and expected computation for polynomially decaying $q(n)$ and ψ_n.

Theorem 3.4.1. ***Bounded variance and compute with polynomial convergence of*** ψ. *Assume ψ converges according to $\psi_n \leq \frac{c_\psi}{(n)^p}$ or faster, for constants $p > 0$ and $c_\psi > 0$. Choose the RT-SS estimator with $q(n) \propto 1/((n)^{p+1/2})$. The resulting estimator $\hat{G}$ achieves expected compute $C \leq (\mathcal{H}_H^{p-\frac{1}{2}})^2$, where $\mathcal{H}_H^i$ is the Hth generalized harmonic number of order i, and expected squared norm $\mathbb{E}[||\hat{G}||_2^2] \leq c_\psi^2(\mathcal{H}_H^{p-\frac{1}{2}})^2 := \tilde{G}^2$. The limit $\lim_{H\to\infty} \mathcal{H}_H^{p-\frac{1}{2}}$ is finite iff $p > \frac{3}{2}$, in which case it is given by the Riemannian zeta function, $\lim_{H\to\infty} \mathcal{H}_H^{p-\frac{1}{2}} = \zeta(p - \frac{1}{2})$. Accordingly, the estimator achieves horizon-agnostic variance and expected compute bounds iff $p > \frac{3}{2}$.*

The corresponding bounds for geometrically decaying $q(n)$ and ψ_n follow.

Theorem 3.4.2. ***Bounded variance and compute with geometric convergence of*** ψ. *Assume ψ_n converges according to $\psi_n \leq c_\psi p^n$, or faster, for $0 < p < 1$. Choose RT-SS and with $q(n) \propto p^n$. The resulting estimator $\hat{G}$ achieves expected compute $C \leq (1-p)^{-2}$ and expected squared norm $||\hat{G}||_2^2 \leq \frac{c_\psi}{(1-p)^2} := \tilde{G}^2$. Thus, the estimator achieves horizon-agnostic variance and expected compute bounds for all $0 < p < 1$.*

Given a setting and estimator $\hat{G}$ from either 3.4.1 or 3.4.2, with corresponding expected compute cost C and upper bound on expected squared norm $\tilde{G}^2$, the following theorem considers regret guarantees when using this estimator to perform stochastic gradient descent.

Theorem 3.4.3. ***Asymptotic regret bounds for optimizing infinite-horizon programs.*** *Assume the setting from 3.4.1 or 3.4.2, and the corresponding C and $\tilde{G}$ from those theorems.*

Let R_t be the instantaneous regret at the tth step of optimization, $R_t = \mathcal{L}(\theta_t) - \min_\theta \mathcal{L}(\theta)$. Let $t(B)$ be the greatest t such that a computational budget B is not exceeded. Use online gradient descent with step size $\eta_t = \frac{D}{\sqrt{t\mathbb{E}[||\hat{G}||_2^2]}}$. As $B \to \infty$, the asymptotic instantaneous regret is bounded by $R_{t(B)} \leq \mathcal{O}(\tilde{G}D\sqrt{\frac{C}{B}})$, independent of H.

Theorem 3.4.3 indicates that if G_n converges sufficiently fast and $\mathcal{L}_n$ is convex, the RT estimator provably optimizes the limit.

3.5 Adaptive RT estimators

In practice, the estimator considered in the previous section may have high variance. This section develops an objective for designing such estimators, and derives closed-form $W(n, N)$ and q which maximize this objective given estimates of $\mathbb{E}[||\Delta_i||_2^2]$ and assumptions on $\mathrm{Cov}(\Delta_i, \Delta_j)$.

3.5.1 Choosing between unbiased estimators

We propose choosing an estimator which achieves the best lower bound on on the expected improvement per compute unit spent, given smoothness assumptions on the loss. Our analysis builds on that of Balles et al. (2016): they adaptively choose a batch size using batch covariance information, while we choose between between arbitrary unbiased gradient estimators using knowledge of those estimators' expected squared norms and computation cost.

Here we assume that the true gradient of the objective $\nabla_\theta \mathbb{E}[\mathcal{L}(\theta)] := \nabla_\theta$ (for compactness of notation) is smooth in θ. We do not assume convexity. Note that ∇_θ is not necessarily equal to $G(\theta)$, as the loss $\mathcal{L}(\theta)$ and its gradient $G(\theta)$ may be random variables due to sampling of data and/or latent variables.

We assume that $\mathcal{L}$ is L-smooth (the gradients of $\mathcal{L}(\theta)$ are L-Lipschitz), i.e., there exists a constant $L > 0$ such that $\nabla_{\theta_b} - \nabla_{\theta_a} \leq L||\theta_b - \theta_a||_2 \quad \forall \theta_a, \theta_b \in \mathbb{R}^d$. It follows (Balles et al., 2016; Bottou et al., 2018) that, when performing SGD with an unbiased stochastic gradient estimator $\hat{G}_t$,

$$\mathbb{E}[\mathcal{L}_H(\theta_t)] - \mathbb{E}[\mathcal{L}_H(\theta_{t+1})] \geq \mathbb{E}[\eta_t \nabla_{\theta_t}^T \hat{G}_t(\theta_t)] - \mathbb{E}[\frac{L\eta_t^2}{2}||\hat{G}_t(\theta_t)||_2^2]\,.$$

Unbiasedness of $\hat{G}$ implies $\mathbb{E}[\nabla_{\theta_t}^T \hat{G}_t(\theta_t)] = ||\nabla_{\theta_t}^T||_2^2$, thus:

$$\mathbb{E}[\mathcal{L}_H(\theta_t)] - \mathbb{E}[\mathcal{L}_H(\theta_{t+1})] \geq \mathbb{E}[\eta_t ||\nabla_\theta||_2^2] - \mathbb{E}[\frac{L\eta_t^2}{2}||\hat{G}_t(\theta_t)||_2^2] := J\,.$$

Above, J is a lower bound on the expected improvement in the loss from one optimization step. Given a fixed choice of $\hat{G}_t(\theta_t)$, how should one pick the learning rate η_t to maximize J and what is the corresponding lower bound on expected improvement?

Optimizing η_t by finding $\eta_t^\star$ s.t. $\frac{dJ}{d\eta_t^\star} = 0$ yields

$$\eta_t^\star = \frac{||\nabla_\theta||_2^2}{L\mathbb{E}[||\hat{G}_t(\theta_t)||_2^2]} \propto \frac{1}{\mathbb{E}[||\hat{G}_t(\theta_t)||_2^2]} \tag{3.7}$$

$$J^\star = \frac{||\nabla_\theta||^4}{2L\mathbb{E}[||\hat{G}_t(\theta_t)||_2^2]} \propto \frac{1}{\mathbb{E}[||\hat{G}_t(\theta_t)||_2^2]} \tag{3.8}$$

This tells us how to choose η_t if we know L, $||\hat{G}||_2^2$, etc. In practice, it is unlikely we know L or even $||\nabla_{\theta_t}||_2$. We might assume we have access to some "reference" learning rate $\bar{\eta_t}$, which has been optimized for use with a "reference" gradient estimator $\bar{G}_t$, with known $\mathbb{E}[||\bar{G}_t||_2^2]$. For example, when using RT estimators, we may have access to learning rates which have been optimized for use with the un-truncated estimator, or can find them via grid search. Instead of directly maximizing J, we choose η_t for $\hat{G}$ by maximizing *improvement relative to the reference estimator* in terms of J, the lower bound on expected improvement.

Assume that $\bar{\eta}_t$ has been set optimally for a problem and reference estimator $\bar{G}$ up to some constant k, i.e.,

$$\bar{\eta}_t = k\frac{||\nabla_{\theta_t}||_2^2}{L\mathbb{E}[||\bar{G}_t(\theta_t)||_2^2]}\,. \tag{3.9}$$

Then the expected improvement $\bar{J}$ obtained by the reference estimator $\bar{G}$ is:

$$\bar{J} = (k - \frac{k^2}{2})\frac{||\nabla_{\theta_t}||^4}{2L\mathbb{E}[||\bar{G}_t(\theta_t)||_2^2]} \tag{3.10}$$

We assume that $0 < k < 2$, such that $\bar{J}$ is positive and the reference has guaranteed expected improvement. Now set the learning rate according to

$$\eta_t = \bar{\eta_t}\frac{\mathbb{E}[||\hat{G}_t||_2^2]}{\mathbb{E}[||\bar{G}_t||_2^2]}\,. \tag{3.11}$$

It follows that the expected improvement $\hat{J}$ obtained by the estimator $\hat{G}$ is

$$\hat{J} = \frac{\mathbb{E}[||\bar{G}_t(\theta_t)||_2^2]}{\mathbb{E}[||\hat{G}_t(\theta_t)||_2^2]}\bar{J} \tag{3.12}$$

Let the expected computation cost of evaluating $\hat{G}$ be $\hat{C}$. We want to maximize $\hat{J}/\hat{C}$. If we use the above method to choose η_t, we have $\frac{\hat{J}}{\hat{C}} \propto \left(\hat{C}\mathbb{E}||\hat{G}_t(\theta_t)||_2^2\right)^{-1}$. We call $\left(\hat{C}\mathbb{E}||\hat{G}_t(\theta_t)||_2^2\right)^{-1}$ the *relative optimization efficiency*, or ROE. We decide between gradient estimators $\hat{G}$ by choosing the one which maximizes the ROE. Once an estimator is chosen, one should choose a learning rate

according to (3.11) relative to a reference learning rate $\bar{\eta}$ and estimator $\bar{G}$.

3.5.2 Optimal weighted sampling for RT estimators

Now that we have an objective for choosing between unbiased stochastic gradient estimators with varying computation, we can consider designing randomized telescope estimators which optimize the ROE. For the classes of single sample and Russian roulette estimators, we prove conditions under which that class maximizes the ROE across an arbitrary choice of RT estimators. We also derive closed-form expressions for the optimal sampling distribution q for each class, under the conditions where that class is optimal.

In this section, we assume that computation can be reused and evaluating $\hat{G}_H = \sum_{n=1}^{N} \Delta_n W(n, N)$ has computation cost $C(N)$. As described in Section 3.1, this is approximately true for many objectives. When it is not, the cost of computing $\sum_{n=1}^{N} \Delta_n W(n, N)$ is $\sum_{n=1}^{N} C(n)\mathbb{1}\{(W(n, N) \neq 0)$ or $(W(n+1, N) \neq 0)\}$. This would penalize the ROE of dense $W(n, N)$ and favor sparse $W(n, N)$, possibly impacting the optimality conditions for RT-RR. We mitigate this inaccuracy by subsequence selection (described in the following subsection), which allows construction of sparse sampling strategies.

We begin with the RT-SS estimator, showing this is minimax-optimal with regards to an adversarial choice of diagonal covariances $\mathrm{Cov}(\Delta_i, \Delta_j)$, and deriving the optimal sampling distribution $q(N)$.

Theorem 3.5.1. ***Optimality of RT-SS under adversarial correlation.*** *Consider the family of estimators presented in Equation 3.2. Assume θ, ∇_θ, and G are univariate. For any fixed sampling distribution q, the single-sample RT estimator RT-SS minimizes the worst-case variance of $\hat{G}$ across an adversarial choice of covariances* $\mathrm{Cov}(\Delta_i, \Delta_j) \leq \sqrt{\mathrm{Var}(\Delta_i)}\sqrt{\mathrm{Var}(\Delta_j)}$.

Theorem 3.5.2. ***Optimal q under adversarial correlation.*** *Consider the family of estimators presented in Equation 3.2. Assume* $\mathrm{Cov}(\Delta_i, \Delta_i)$ *and* $\mathrm{Cov}(\Delta_i, \Delta_j)$ *are diagonal. The RT-SS estimator with $q_n \propto \sqrt{\frac{\mathbb{E}[||\Delta_n||_2^2}{C(n)}}$ maximizes the ROE across an adversarial choice of diagonal covariance matrices* $\mathrm{Cov}(\Delta_i, \Delta_j)_{kk} \leq \sqrt{\mathrm{Cov}(\Delta_i, \Delta_i)_{kk}\mathrm{Cov}(\Delta_j, \Delta_j)_{kk}}$.

We continue with the RT-RR estimator, showing this is optimal when $\mathrm{Cov}(\Delta_i, \Delta_i)$ is diagonal and Δ_i and Δ_j are independent for $j \neq i$, and deriving the optimal sampling distribution $q(N)$.

Theorem 3.5.3. ***Optimality of RT-RR under independence****. Consider the family of estimators presented in Eq. 3.2. Assume the Δ_j are univariate. When the Δ_j are uncorrelated, for any importance sampling distribution q, the Russian roulette estimator achieves the minimum variance in this family and thus maximizes the optimization efficiency lower bound.*

Theorem 3.5.4. ***Optimal q under independence****. Consider the family of estimators presented in Equation 3.2. Assume* $\mathrm{Cov}(\Delta_i, \Delta_i)$ *is diagonal and* Δ_i *and* Δ_j *are independent. The RT-RR estimator with* $Q(i) \propto \sqrt{\frac{\mathbb{E}[||\Delta_i||_2^2}{C(i)-C(i-1)}]}$*, where* $Q(i) = \Pr(n \geq i) = \sum_{j=i}^{H} q(i)$*, maximizes the ROE.*

3.5.3 Subsequence selection

The scheme for designing RT estimators given in the previous subsection contains assumptions which will often not hold in practice. To partially alleviate these concerns, we can design the *sequence of iterates over which we apply the RT estimator* to maximize the ROE.

Some sequences may result in more efficient estimators than others, depending on how the intermediate iterates G_n converge to G. For example, solving an ODE with a number of steps for which the solution is unstable is unlikely to produce gradients which correlate well with the true gradient. The variance of the estimator, and the ROE, will be reduced if we choose a sequence $\mathcal{L}_n$ such that G_n is positively correlated with G for all n.

We begin with a reference sequence $\bar{\mathcal{L}}_i$, $\bar{G}_i$, with cost function $\bar{C}$, where $i, j \in \mathcal{N}$ and $i, j \leq \bar{H}$, and where $\bar{G}_i$ has cost $\bar{c}_i$. We assume knowledge of $\mathbb{E}[||\bar{G}_i - \bar{G}_j||_2^2]$. We aim to find a subsequence $S \in \mathcal{S}$, where $\mathcal{S}$ is the set of subsequences over the integers $1...\bar{H}$ which have final element $S_{-1} = \bar{H}$. Given S, we take $\mathcal{L}_n = \bar{\mathcal{L}}_{S_n}$, $G_n = \bar{G}_{S_n}$, $C(n) = \bar{C}(S_n)$, $H = |S|$, and $\Delta_n = G_n - G_{n-1}$, where $G_0 := 0$.

In practice, we greedily construct S by adding indexes i to the sequence $[\bar{H}]$ or removing indexes i from the sequence $[1, 2, ..., \bar{H}]$. As this step requires minimal computation, we perform both greedy adding and greedy removal and return the S with the best ROE. The minimal subsequence $S = [\bar{H}]$ is always considered, allowing RT estimators to fall back on the original full-horizon estimator.

3.6 Practical implementation

3.6.1 Tuning the estimator

We estimate the expected squared distances $\mathbb{E}[||\bar{G}_i - \bar{G}_j||_2^2]$ by maintaining exponential moving averages. We keep track of the computational budget B used so far by the RT estimator, and "tune" the estimator every $K\bar{C}(\bar{H})$ units of computation, where $\bar{C}(\bar{H})$ is the compute required to evaluate $\bar{G}_{\bar{H}}$, and K is a "tuning frequency" hyperparameter. During tuning, the gradients G_i are computed, the squared norms $||\bar{G}_i - \bar{G}_j||_2^2$ are computed, and the exponential moving averages are updated. At the end of tuning, the estimator is updated using the expected squared norms; i.e. a subsequence is

selected, q is set according to section 5.2 with choice of RT-RR or RT-SS left as a hyperparameter, and the learning rate is adapted according to section 5.1

3.6.2 Controlling sequence length

Tuning and subsequence selection require computation. Consider using RT to optimize an objective with an inner loop of size M. If we let $\bar{G}_i$ be the gradient of the loss after i inner steps, we must maintain $M^2 - M$ exponential moving averages $\mathbb{E}||\bar{G}_i - \bar{G}_j||_2^2$, and compute M gradients $\bar{G}_i$ each time we tune the estimator. The computational cost of the tuning step under this scheme is $\mathcal{O}(M^2)$. This is unacceptable if we wish our method to scale well with the size of loops we might wish to optimize.

To circumvent this, we choose base subsequences such that $\bar{C}_i \propto 2^i$. This ensures that $\bar{H} = \mathcal{O}(\log_2 M)$, where M is the maximum number of steps we wish to unroll. We must maintain $\mathcal{O}(\log_2^2 M)$ exponential moving averages. Computing the gradients $\bar{G}_i$ during each tuning step requires compute $C_{\text{tune}} = \sum_{i=1}^{\bar{H}} k * 2^i$. Noting that $\bar{C}_{\bar{H}} = k * 2^{\bar{H}}$ and that $\sum_{i=1}^{N} 2^i < 2^{N+1} \forall N$ yields $C_{\text{tune}} < 2\bar{C}_{\bar{H}} = 2M$.

3.7 Experiments

For all experiments, we tune learning rates for the full-horizon un-truncated estimator via grid search over all $a \times 10^b$, for $a \in \{1.0, 2.2, 5.5\}$ and $b \in \{0.0, 1.0, 2.0, 3.0, 5.0\}$. The same learning rates are used for the truncated estimators and (as reference learning rates) for the RT estimators. For simplicity, we do not decay the learning rate in any experiments.

We use the same hyperparameters for our online tuning procedure for all experiments: the tuning frequency K is set to 5, and the exponential moving average weight α is set to 0.9. These hyperparameters were not extensively tuned. For each problem, we compare deterministic truncations with RT-SS and RT-RR estimators, each with different truncations.

3.7.1 Lotka-Volterra ODE

We first experiment with variational inference of parameters of a Lotka-Volterra (LV) ODE. LV ODEs are defined by the predator-prey equations:

$$\frac{du_1}{dt} = Au_1 - Bu_1u_2 \qquad \frac{du_2}{dt} = Cu_1u_2 - Du_2$$

u_2 is the predator population, and u_1 is the prey population.

We aim to infer the parameters $\lambda = [u_1(t=0), u_2(t=0), A, B, C, D]$. The true parameters are drawn from $\mathcal{U}([1.0, 0.4, 0.8, 0.4, 1.5, 0.4], [1.5, 0.6, 1.2, 0.6, 2.0, 0.6])$, chosen empirically to ensure stability solving the equations. We generate ground-truth data by solving the equations using RK4 (a common 4th-order Runge Kutta method) from $t = 0$ to $t = 5$ with 10000 steps. The learner is given access to five equally spaced noisy observations $y(t)$, generated according to $y(t) = u(t) + \mathcal{N}(0, 0.1)$.

We place a diagonal Gaussian prior on θ with the same mean and standard deviation as the data-generating distribution. The variational posterior is a diagonal Gaussian $q(\lambda)$ with mean μ and standard deviation σ. The parameters optimized are $\theta = [\tilde{\mu}, \tilde{\sigma}]$. We let $\mu = g(\tilde{\mu})$ and $\sigma = g(\tilde{\sigma})$, where $g(x) = \log(1 + e^{\tilde{x}})$, to ensure positivity. We use a reflecting boundary to ensure positivity of parameter samples from q. The variational posterior is initialized to have mean equal to the prior and standard deviation 0.1.

The loss considered is the negative evidence lower bound (negative ELBO). The ELBO is:

$$\mathrm{ELBO}(q(\theta)) = \mathbb{E}_{q(\theta)} \sum_t \log p(y(t)|u_\theta(t)) + D_{\mathsf{KL}}\big(q(\theta)||p(\theta)\big)$$

Above, $u_\theta(t)$ is the value of the solution u_θ to the LV ODE with parameters θ, evaluated at time t. We consider a sequence $\mathcal{L}_n(\theta)$, where in computing the ELBO, $u_\theta(t)$ is approximated by solving the ODE using RK4 with $2^n + 1$ steps, and linearly interpolating the solution to the 5 observation times. The outer-loop optimization is performed with a batch size of 64 (i.e., 64 samples of θ are performed at each step) and a learning rate of 0.01. Evaluation is performed with a batch size of 512.

Figure 3.1 shows the loss of the different estimators over the course of training. RT-SS estimators outperform the un-truncated estimator without inducing bias. They are competitive with the truncation $H = 6$, while avoiding the bias present with the truncation $H = 4$. By contrast, some RT-RR estimators experience issues with optimization, falling into the same local minimum as the $H = 4$ truncation.

3.7.2 MNIST learning rate

We next experiment with meta-optimization of a learning rate on MNIST. We largely follow the procedure used by Wu et al. (2018). We use a feedforward network with two hidden layers of 100 units, with weights initialized from a Gaussian with standard deviation 0.1, and biases initialized to zero. Optimization is performed with a batch size of 100.

The neural network is trained by SGD with momentum using Polyak averaging, with the

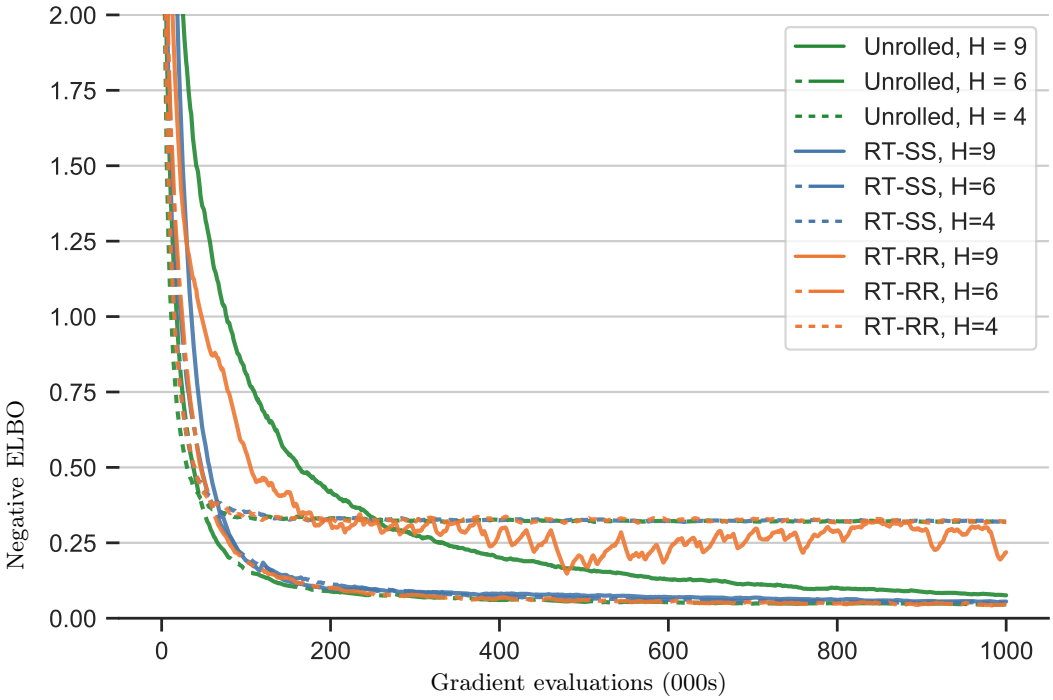


Figure 3.1: Lotka-Volterra parameter inference

momentum parameter fixed to 0.9. We aim to learn a learning rate η_0 and decay λ for the inner-loop optimization. These are initialized to 0.01 and 0.1 respectively. The learning rate for the inner optimization at an inner optimization step t is $\eta_t = \eta_0(1 + \frac{t}{5000})^{-\lambda}$.

As in Wu et al. (2018), we pretrain the net for 50 steps with a learning rate of 0.1. $\mathcal{L}_n$ is the evaluation loss after 2^n+1 training steps with a batch size of 100. The evaluation loss is measured over $2^n + 1$ validation batches or the entire validation set, whichever is smaller. The outer optimization is performed with a learning rate of 0.01.

RT estimators with $H = 9$ or $H = 5$ achieve faster convergence than the fixed-truncation estimators. RT-SS outperforms RT-RR. All estimators with $H = 5$ appear to suffer from some bias. The un-truncated estimator achieves a slightly better loss than the RT estimators, but takes significantly longer to converge.

3.7.3 `enwik8` LSTM

Finally, we study a high-dimensional optimization problem: training an LSTM to model sequences on `enwik8`. These data are the first 100M bytes of a Wikipedia XML dump. There are 205 unique tokens. We use the first 90M, 5M, and 5M characters as the training, evaluation, and test sets.

We build on code from Merity et al. (2017, 2018) found at https://github.com/salesforce/awd-lstm-lm. We train an LSTM with 1000 hidden units and 400-dimensional input and output embeddings.

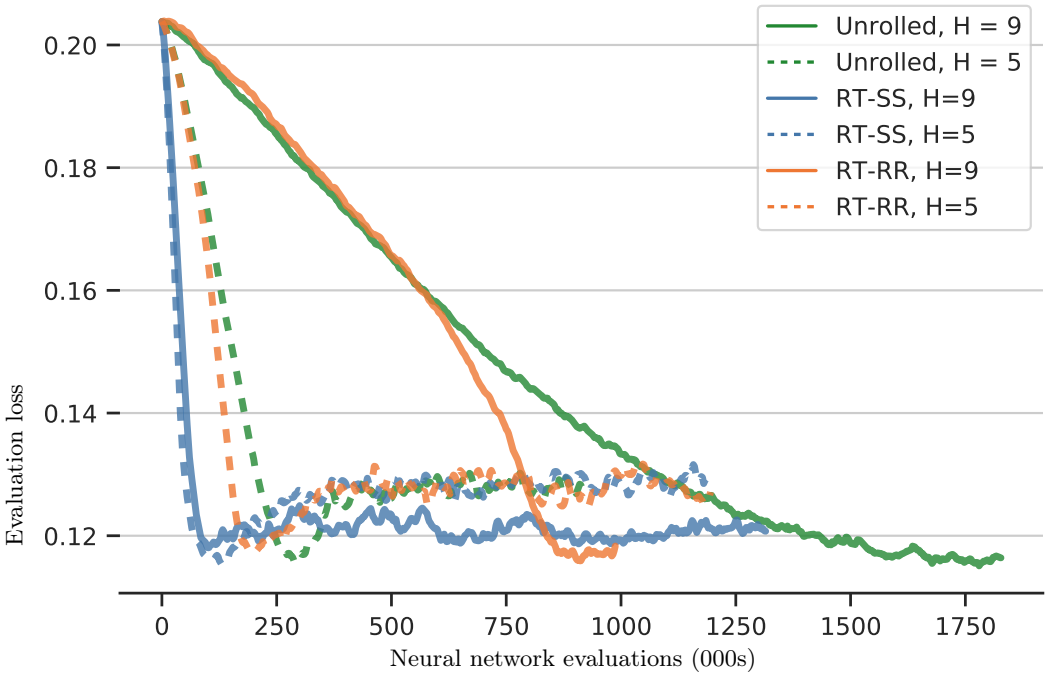


Figure 3.2: MNIST learning rate meta-optimization

The model has 5.9M parameters. The only regularization is an ℓ_2 penalty on the weights with magnitude 10^{-6}. Gradients are clipped to magnitude 1.0 before each optimization step. The optimization is performed with a learning rate of 2.2. This model is not state-of-the-art: our aim to investigate performance of RT estimators for optimizing high-dimensional neural networks, rather than to maximize performance at a language modeling task.

We choose $\mathcal{L}_n$ to be the mean cross-entropy after unrolling the LSTM training for $6 * 2^n + 1$ steps. We choose the horizon $H = 5$, such that the un-truncated loop has 193 steps, chosen to be approximately equal to the 200-length training sequences used by Merity et al. (2018).

Figure 3.3 shows the training bits-per-character (proportional to the training cross-entropy loss). All RT estimators provide some acceleration over the un-truncated $H = 5$ estimator early in training. RT-SS underperforms relative to RT-RR. Within the first 100k cell evaluations, the RT estimators fall back on the un-truncated estimator, subsequently progressing slightly more slowly due to computational cost of tuning. We conjecture that the covariance assumptions in Section 5 are often unsuited to high-dimensional problems, and lead to overly conservative estimators.

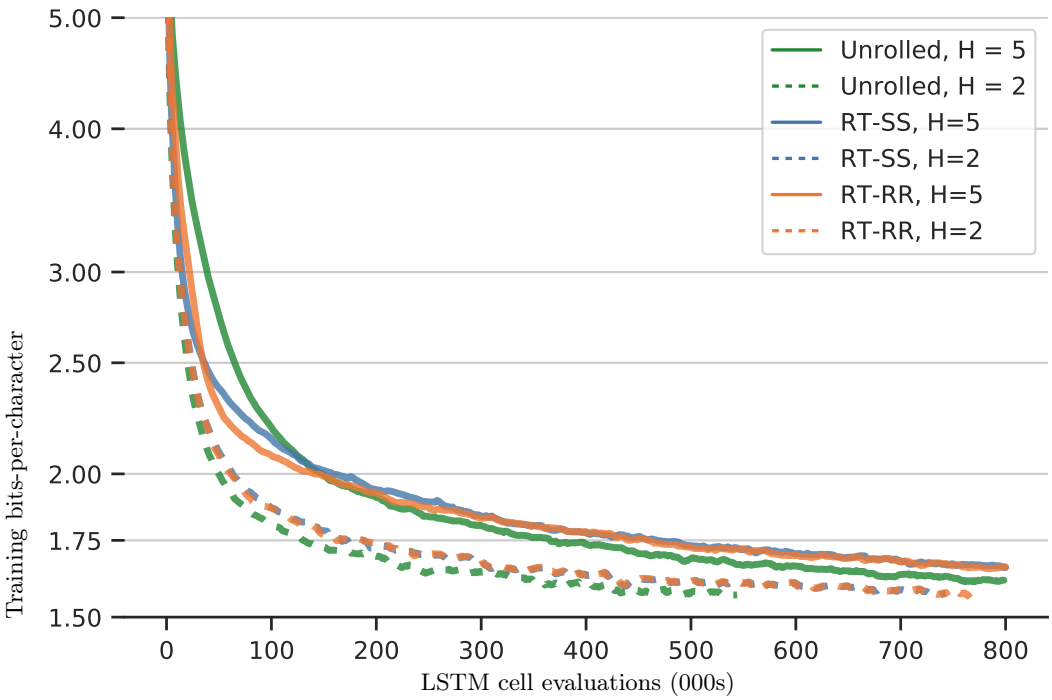


Figure 3.3: LSTM training on `enwik8`

3.8 Limitations and future work

Other optimizers. We develop the lower bound on expected improvement for SGD. Important future directions would investigate adaptive and momentum-based SGD methods such as Adam (Kingma and Ba, 2014).

Tuning step. Our method includes a tuning step which requires computation. It might be possible to remove this tuning step by estimating covariance structure online using just the values of $\hat{G}$ observed during each optimization step.

RT estimators beyond RT-SS and RT-RR. There is a rich family defined by choices of q and $W(n, N)$. The optimal member depends on covariance structure between the G_i. We explore RT-SS and RT-RR under strict covariance assumptions. Could we relax these assumptions and derive or optimize approximately optimal estimators across the wider family? This could improve adaptive estimator performance for high-dimensional problems such as training RNNs.

Predictive models of the sequence limit. Using any sequence G_n with RT yields an unbiased estimator as long as the sequence is consistent, i.e. its limit G is the true gradient. Combining randomized telescopes with predictive models of the gradients (Jaderberg et al., 2017; Weber et al., 2019) might yield a fast-converging sequence, leading to estimators with low computation and variance.

3.9 Conclusion

We investigated the use of randomly truncated unbiased gradient estimators for optimizing objectives which involve loops and limits. We proved these estimators can achieve horizon-independent convergence rates for optimizing loops and limits. We derived adaptive variants which can be tuned online to maximize a lower bound on expected improvement per unit computation. Experimental results matched theoretical intuitions that the single sample estimator is more robust than Russian roulette for optimization. The adaptive RT-SS estimator often significantly accelerates optimization, and can otherwise fall back on the un-truncated estimator.

Chapter 4

Meta-PDE: Learning to solve PDEs quickly without a mesh

4.1 Abstract

Partial differential equations allow us to model and design a wide variety of systems. Solving PDEs with finite element analysis (FEA) can be computationally prohibitive, and in optimization, model-fitting or dynamics problems, the PDE must be solved for many different governing equations, boundary conditions, or geometric domains. Surrogate models allow fast approximate PDE solving, but most surrogates require fixing a vector representation for the PDE's parameters and solution and are not usable when PDEs we encounter may have complex and varying geometry. Meanwhile, neural nets have drawn interest for solving PDEs as they do not require a mesh and allow combining physics with observations, but take far too long to fit to be competitive with FEA.

We use meta-learning to allow an alternative API for surrogate modeling. Meta-PDE takes as input (i) a sampler for points in the PDE domain and (ii) a variational energy density which measures deviation from governing equations. We represent the solution with a neural network, and train an initialization such that it can quickly minimize the variational energy after a few gradient steps. This functional API does not require fixing a parametric basis for the geometry or governing equations. It also does not require supervision from expensive ground-truth or FEA solutions. We apply Meta-PDE to a nonlinear Poisson problem, and show it learns to solve PDEs accurately and quickly across different boundary conditions, governing equations, and problem geometries. The resulting meta-model solves these PDEs much faster than FEA methods which achieve similar

accuracy.

4.2 Introduction

Partial differential equations (PDEs) can be used to model many physical, biological, and mathematical systems, including those governing thermodynamics, continuum mechanics, and electromagnetism, and have applications outside physics in areas such as modeling populations, traffic, optimality of continuous control, and financial markets. Analytical solutions are rarely available for PDEs of practical importance; thus, computational methods to approximate PDE solutions are critical for many problems in science and engineering. One of the most widely used is finite element analysis (FEA). In FEA, the continuous problem is discretized, with the solution represented by a piecewise polynomial on a mesh.

Solving PDEs with FEA can be computationally prohibitive, particularly when the problem geometry requires use of a fine mesh, as the size of the system to be solved grows proportional to the number of mesh cells. The computational expense is exacerbated for parameter identification or design optimization. In this case, the PDE must be solved at each step of a procedure optimizing some set of design or system parameters to maximize a design objective or minimize discrepancy of the solution from data. Given a solution to the PDE, the *adjoint method* (Lions, 1971; Mitusch et al., 2019) may be used to obtain the gradient of an objective computed from the solution with respect to the PDE parameters with cost equivalent to a single solve of the *linearized* PDE. Therefore, the key bottleneck to optimization of PDE parameters is the "forward pass" of obtaining an accurate solution.

Surrogate modeling typically involves fitting a model to map from PDE parameters in a vector basis to coefficients of an approximate solution in another vector basis. The model is trained on a distribution of PDEs to correctly predict their solution or to satisfy the associated PDE constraints. These bases are fixed across the class of problems to be amortized. However, different problems may require different meshes to represent the solution, source terms, or boundary conditions, or may demand different representations for the geometry itself. Even generating a mesh which can adequately represent the geometry and solution of a single problem can be difficult. Surrogate modeling approaches are usually therefore restricted to scenarios where we can fix a mesh or at least represent geometry, parameters, and solution with fixed coefficient vectors.

Meanwhile, neural networks have long been researched as a basis with which to solve PDEs, and have seen considerable recent interest. Mesh-free methods such as neural networks remove some of

the difficulties with generating meshes and bases to model complex geometry. Neural networks also alow blending observations with PDE constraints to approximate a solution field even when the data does not come in a form amenable to being imposed as boundary conditions (Raissi et al., 2019). However, neural networks take far too long to optimize to fit a given PDE to be competitive with finite-element methods.

We use meta-learning to accelerate fitting neural networks to satisfy PDE constraints. This lets us develop a new, "functional" API for surrogate modeling, which can handle arbitrary geometries and removes the need to fix a mesh or to fix vector bases for the PDE parameters or solution. For a given PDE, our surrogate model takes as input (i) samplers which can sample points uniformly on each region of the domain, and (ii) a loss function encoding the PDE constraint or boundary condition for each such region. Combining these allows unbiased estimation of a variational energy which measures deviation of a given solution field from the governing equations. We use a neural network to model the solution field, and train a neural network initialization to converge quickly across a distribution of tasks a la MAML, Finn et al. (2017); in our case each task in the distribution is minimizing the variational energy for a PDE with given domain, boundary conditions and governing equations.

Our scheme has several important properties. It does not require supervised data provided by expensive PDE solvers. It does not place any assumptions on the structure of the geometry, and does not require geometry to be fixed across PDEs or for the user to define a parametric representation of varying geometry. Similarly, it does not place any assmptions on the structure of the PDE constraints and boundary conditions. Geometry and governing equations are free to vary as long as the user can supply an appropriate sampler or loss function. It also provides the first way to train a neural network to satisfy a PDE with competitive or faster speed to finite element analysis.

4.3 Finite element analysis

PDEs are most naturally posed in a *strong form*:

$$\mathcal{F}(u)(x) = 0 \qquad \text{in } \Omega, \tag{4.1}$$

$$\mathcal{G}(u)(x) = 0 \qquad \text{on } \partial\Omega. \tag{4.2}$$

where $\Omega \subset \mathbb{R}^{d_\Omega}$ is the problem domain with boundary $\partial\Omega$, $u : \Omega \to \mathbb{R}^{d_u}$ is the solution, $\mathcal{F} : (\Omega \to \mathbb{R}^{d_u}) \to (\Omega \to \mathbb{R}^{d_\mathcal{F}})$ is a linear or nonlinear operator involving u and its partial derivatives, and $\mathcal{G} : (\Omega \to \mathbb{R}^{u}) \to (\Omega \to \mathbb{R}^{d_\mathcal{G}})$ is an operator enforcing a boundary condition (for example,

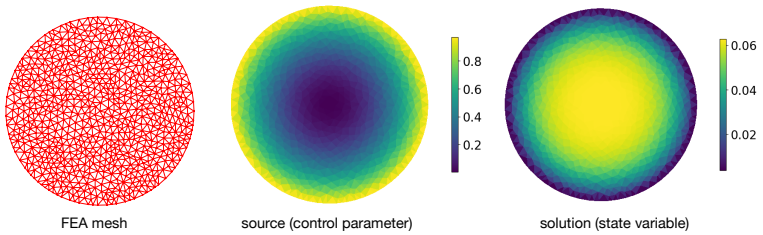


Figure 4.1: Poisson equation on a disc. Figure: Xue et al. (2020a).

$\mathcal{G}(u)(x) = u(x) - b(x)$ is a Dirichlet boundary condition forcing u to be equal to a function b on the boundary).

Finite element analysis involves rewriting the PDE in a *weak form*: find u in a function space $\mathcal{V}$, such that

$$\int_\Omega < \mathcal{F}(u)(x), v(x) > dx + \int_{\partial\Omega} < \mathcal{G}(u)(x), v(x) > dx = 0 \quad \forall v \in \mathcal{V} \tag{4.3}$$

When $\mathcal{V}$ is a suitable infinite-dimensional Sobolev space, we recover the same solution as the strong form. In order to solve the problem numerically with FEA, we let $\mathcal{V}$ be a class of piecewise low-degree polynomials over the domain, parameterized by some finite number of interpolating points. We fix the values of the interpolating points on the boundary, form a set of basis vectors v for $\mathcal{V}$, rewrite the weak form of the PDE as a linear or nonlinear system representing the set of constraints that must be satisfied, and solve the system with an appropriate linear or nonlinear solver. Figure 4.1 shows as an example the Poisson problem on a disc. For this simple problem we have $\mathcal{F}(u) = \nabla\nabla u - f$, for a spatially varying source term f, and $\mathcal{G}(u) = u$, i.e. enforcing $u = 0$ on the boundary.

We observe that finding the solution to the PDE i equivalent to finding the minimizer of the variational energy

$$\mathcal{J}(u) = \int_\Omega ||\mathcal{F}(u)(x)||_2^2 dx + \int_{\partial\Omega} ||\mathcal{G}(u)(x)||_2^2 dx$$

As with Xue et al. (2020a), we use this optimization perspective to derive an efficient training method for our surrogate models.

4.4 Surrogate modeling

Meta-PDE is a surrogate modeling approach. Surrogate models are useful when one will need to evaluate the solution of many PDEs from a similar family and is willing to pay the up-front cost of

training a surrogate in order to solve the downstream probelms with minimal cost-per-PDE. The most popular application is in design or system identification, where a PDE must be solved at each step optimizing the PDE's parameters, boundary conditions, or geometry, and it is convenient to replace the PDE solver with a cheap surrogate (Kochenderfer and Wheeler, 2019). The optimization may be numerical ("PDE-constrained optimization") or by hand. If numerical, gradients can be obtained using the adjoint method with cost equivalent to one solve of a linearized version of the PDE. In both cases, therefore, the optimization bottleneck is in the "forward pass" of solving the PDE. Surrogate models are also particularly relevant in scenarios involving dynamics models (where a PDE must be solved at each timestep), and where real-time analysis is required for the convenience of a human designer or to allow embedding the PDE solver in a control policy.

There are many approaches to surrogate modeling, such as random forests (Criminisi et al., 2011), Gaussian processes (Shahriari et al., 2015), Student-t processes (Shah et al., 2014), and neural networks (Snoek et al., 2015). Regression can be from the PDE's parameters to coefficients of the PDE solution, yielding a general-purpose surrogate for that distribution of PDEs, or from PDE parameters to an objective value for a specific optimization problem. Most surrogate modeling procedures have two key restrictions:

1. They require supervised training data in the form of (PDE parameter, PDE solution) or (PDE parameter, objective value) pairs, which must be generated by an expensive ground-truth PDE solver,

2. They regress from a vector representation of the PDE or its parameters to a vector representaion of the solution; having to fix these vector bases makes it difficult to fit surrogates for distributions of PDEs containin highly varied structure in geometry or governing equations.

Restriction 1 means that in order to be of net computational benefit, the surrogate must be applied to many more downstream problems than the number of data points required to fit a good model. For rich classes of PDEs and expressive models, it can take tens or hundreds of thousands of data points to fit a model which can generalize, so this greatly limits surrogate models' use case.

Restriction 2 is not an issue if the problem can be phrased as regressing from coefficients of a parameter or boundary condition in a piecewise polynomial basis to coefficients of a solution in a similar basis, with the discretization fixed across problems. However, many problems are not amenable to such framing. Problems with different domain geometry or very different parameters and solutions may require very different discretizations to represent them well; generating a good mesh for a given problem can itself be a hard problem and can be a greater bottleneck than the

cost of the base PDE solve. The factors of variation of a PDE (the domain, governing equations, and parameters) are naturally and generally expressed as functions on the domain and as operators on solutions: requiring them to be parameterized by vectors restricts the classes of PDEs to which surrogate models can be applied and greatly limits their use case.

Several recent works have relaxed one of these restrictions. Zhu et al. (2019) amortize the finite differentce method, predicting PDE solutions from parameter fields with a ConvNet by training the ConvNet to produce solutions with minimum variational energy across a distribution of problems. This removes restriction 1 but requiring the discretization to be a fixed uniform grid. Xue et al. (2020a) amortize the finite element method, training a surrogate model to minimize a similar variational energy but use a finite element discretization: their model can handle complicated geometry, and by measuring the variational energy in the finite element basis maintains some desirable properties of finite element analysis. This avoids restriction 1 and allows arbitrary meshes but requires a single mesh be fixed for all PDEs in the distribution to be amortized. Graph Neural Network based approaches (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020), which learn a forward operator in terms of interactions between nearby particles or mesh cells, allow for arbitrary geometries (partially removing restriction 2) but require supervised data and have not yet produced surrogate models which are cheaper to evaluate than ground-truth solvers when tested on equivalent hardware. Neural Operator based approaches (Li et al., 2020b,a) learn a map from initial conditions or parameters to solution which builds in some invariance to mesh *resolution*, but require the mesh to be a uniform grid, and also require expensive supervised data.

4.5 Meta-learning mesh-free PDE operators

We propose Meta-PDE, an approach which meta-learns mesh-free PDE operators and provides a new, functional API for PDE surrogate modeling. Most PDEs can be fully defined by specification of:

- a domain Ω with boundary $\partial\Omega$,
- an operator $\mathcal{F}$ representing governing equations,
- and an operator $\mathcal{G}$ representing boundary conditions.

We aim for a surrogate model with an input schema as close to this general specification as possible. Meta-PDE uses meta-learning, and specifically MAML (), to achieve this. Meta-PDE's parameters are the initial parameters θ_0 for a neural network u_θ representing a function $u : \mathbb{R}^{d_\Omega} \to \mathbb{R}^{d_u}$, and per-parameter per-step learning rates α_k. The geometric dimension $\mathbb{R}^{d_\Omega}$ and solution dimension $\mathbb{R}^{d_u}$

must remain fixed across PDEs in the distribution, even though Ω is allowed to vary. When using Meta-PDE as a surrogate to compute an approximate solution to a given PDE (one 'example' or 'task'), the inputs to the Meta-PDE model are:

- a sampler $s(\Omega)$ which returns points in the domain Ω,
- a sampler $s(\partial\Omega)$ which returns points on the boundary $\partial\Omega$,
- an operator $\mathcal{F}$ representing governing equations,
- and an operator $\mathcal{G}$ representing boundary conditions.

The operators $\mathcal{F}$ and $\mathcal{G}$ may be supplied directly and do not require a particular parametric form. The user must suppy a sampler for the domain and for the boundary of each PDE within the training disribution and for each PDE seen during deployment. A sampler can easily be constructed for any domain for which we have a mesh, but it is also often easier to construct a sampler than to construct an accurate mesh. Finite element models usually use piecewise linear meshes, which can take many elements to accurately represent curved shapes: even when using piecewise polynomially-shaped meshes, the slow rate of convergence of using these meshes to approximate non-polynomial geometry can be a major source of error and/or computational expense for FEA. Given an inside-outside oracle for the domain, it is easy to use rejection sampling to sample from it exactly. Most parametric geometry representations such as those used in computer-aided design also allow exact sampling of the boundary, and even minimal representations such as signed distance functions allow approximate sampling (Brubaker et al., 2012).

The samplers and operators are sufficient to construct an estimator $\hat{\mathcal{J}}$ for the variational energy $\mathcal{J}$:

$$\mathcal{J}(u) = \int_\Omega ||\mathcal{F}(u)(x)||_2^2 dx + \int_{\partial\Omega} ||\mathcal{G}(u)(x)||_2^2 dx$$
$$\hat{\mathcal{J}}(u) = \mathbb{E}_{x\sim s(\Omega)}||\mathcal{F}(u)(x)||_2^2 + \mathbb{E}_{x\sim s(\partial\Omega)} \int_{\partial\Omega} ||\mathcal{G}(u)(x)||_2^2$$

$\hat{\mathcal{J}}(u)$ is unbiased as long as $s()$ return points with uniform probability over their supports, or return batches of points which have uniform probability for any given x aggregated over the batch. Unbiased estimation is not necessarily essential. Note $\hat{\mathcal{J}}(u) > 0$ and $\mathcal{J}(u) > 0\ \forall u$, and the true solution u^* of the PDE achieves $\mathcal{J}(u) = \hat{\mathcal{J}}(u) = 0$. These properties hold if we multiply the integrand in $\mathcal{J}(u)$ by an arbitrary density $\mu > 0$ or if we choose samplers s which have full support but nonuniform

probability on Ω or $\partial\Omega$. Therefore, biased sampling will not change the minimizer of the energy estimator if we have a sufficiently expressive hypothesis class for u.

The "forward pass" computing an approximate solution for a given PDE involves a small number K of inner-loop steps (we use $K = 5$) of stochastic optimization, minimizing the variational energy $\mathcal{J}(u)$ and starting from the initialization θ_0:

$$\theta_k = \theta_{k-1} - \alpha_k \nabla_{\theta_{k-1}} \hat{\mathcal{J}}(u_{\theta_{k-1}}) \quad k = 1..K$$

Meta-PDE returns the approixmate solution u_{θ_K}, the neural network with the final set of parameters.

To train Meta-PDE, we use a distribution of tasks, each specified by samplers and constraint operators for the boundary and loss, each representing a different PDE. We draw a batch of tasks with variational energy estimators $\hat{\mathcal{J}}_i$, $i = 1...n$, and unroll the inner loop to find $u_{\theta_K,i}$. For each task the loss is $\hat{\mathcal{J}}_i(u_{\theta_K,i})$. We backpropogate through the inner loop to find the gradients $\nabla_{\theta_0} \sum_i \frac{1}{n} \hat{\mathcal{J}}_i(u_{\theta_K,i})$ and $\nabla_\alpha \sum_i \frac{1}{n} \hat{\mathcal{J}}_i(u_{\theta_K,i})$, which are used in an outer loop to train the model.

4.6 Experiments

We demonstrate Meta-PDE on a nonlinear Poisson problem with varying source terms, boundary conditions, and geometric domain. The PDE takes the form:

$$\begin{aligned} \nabla \cdot ((1 + 0.1u^2)\nabla u)(x) &= f(x) && x \text{ in } \Omega \\ \nabla \cdot u(x) &= b(x) && x \text{ on } \partial\Omega, \end{aligned}$$

where $u \in \mathbb{R}^1$ and $\Omega \subset \mathbb{R}^2$. Using our notation from the preceding section, this is equivalent to constraining the solution in the domain with an operator $\mathcal{F}(u) = ((1 + 0.1u^2)\nabla u) - f$, and constraining the solution on the boundary with an operator $\mathcal{G}(u) = u - b$.

The domain Ω is a disc-like shape centered at the origin, defined in polar coordinates by and the varying radius about the origin

$$r(\theta) = r_0[1 + c_1 \cos(4\theta) + c_2 \cos(8\theta)],$$

where the varying parameters are $c_1, c_2 \sim \mathcal{U}(-0.2, 0.2)$. The source term f is a sum of radial basis

functions,

$$f(x) = \sum_{i=1}^{3} \beta_i \exp ||x - \mu_i||_2^2,$$

where $\beta_i \in \mathbb{R}^1$ and $\mu_i \in \mathbb{R}^2$ are both drawn from standard normal distributions. The boundary condition b is a periodic function, defined in polar coordinates as

$$b(x) = b_0 + b_1 \cos(\theta) + b_2 \sin(\theta) + b_3 \cos(\theta) + b_4 \sin(\theta),$$

where the parameters $b_{0:4} \sim \mathcal{U}(-1, 1)$.

We train Meta-PDE with a batch size of 16 tasks, with 5 inner steps, and with 256 sampled points on the boundary and in the domain used to evaluate the variational energy at each inner step of optimization for each task. To sample points on the boundary, we construct an evenly spaced interval mesh of angles in $[0, 2 * \pi]$, add uniform noise of the size of one interval to each point, and use the points on the boundary using these angles. To sample points in the domain, we do the same but also draw random radii uniformly in $[0, r(\theta)]$. These samplers are not unbiased for non-circular shapes, but this does not change the optimal solution.

Our model is a three layer NN with sinusoidal activations initialized according to the scheme in Sitzmann et al. (2020), (although we replace $\omega_0 = 30.$ in that paper with $\omega_0 = 3.$ to avoid numerical issues when taking higher-order derivatives of a neural network's input-output function). We initialize the inner-loop learning rate to 1×10^{-4}, and use an outer loop learning rate of 1×10^{-5}. Gradients in both inner and outer loop are clipped to have maximal norm 100. In the inner loop, we use vanilla SGD. In the outer loop, we use the Adam optimizer (Kingma and Ba, 2014). We train for 200,000 outer-loop steps, which takes about 6 hours on one GeForce RTX 2080.

All finite element baselines are implemented in FEniCS (Logg et al., 2012a; Alnæs et al., 2015b). We use the Mumps linear solver backend. Meta-PDE is implemented in Jax (Bradbury et al., 2018).

Meta-PDE learns to quickly find solutions with low error. Table 4.1 shows the mean squared solution error and solution time after zero through five gradient steps for a Meta-PDE model trained to minimize the energy estimate after five gradient steps, as compared to finite element models of varying fidelities. The highest-fidelity finite element model was taken as ground truth and was used to compute errors. Errors and solution times were evaluated using sixteen held-out problems from the same distribution which were not used during training Mean-squared errors are computed between the value of a given approximate solution and the value of the ground truth (highest fidelity finite element solution) at 1024 randomly sampled points within the domain. The relative mean-squared

Method	Resolution	Mean finite element DoFs	Relative MSE	Simulation time (CPU)	Simulation time (GPU)
FEA	1	15	0.28 ± 0.63	0.053s	N/A
FEA	2	53	0.014 ± 0.030	0.13s	N/A
FEA	3	85	0.0065 ± 0.014	0.24s	N/A
FEA	4	178	0.0012 ± 0.0022	0.45s	N/A
FEA	5	222	$7.1 \times 10^{-4} \pm 1.5 \times 10^{-3}$	0.74s	N/A
FEA	6	324	$3.7 \times 10^{-4} \pm 8.7 \times 10^{-4}$	0.83s	N/A
FEA	8	433	$4.0 \times 10^{-5} \pm 6.0 \times 10^{-5}$	1.2s	N/A
FEA	10	568	$2.9 \times 10^{-5} \pm 6.2 \times 10^{-5}$	1.9s	N/A
FEA	12	1246	$5.1 \times 10^{-6} \pm 9.7 \times 10^{-6}$	2.5s	N/A
FEA	16	2163	$5.1 \times 10^{-6} \pm 9.7 \times 10^{-6}$	2.92s	N/A
Meta-PDE	N/A	N/A	$6.2 \times 10^{-5} \pm 1.0 \times 10^{-4}$	0.097s	0.0022s

Table 4.1: Accuracy vs solution time for finite element methods and for meta-PDE. For FEA, a mesh is generated with MSHR using 3x "resolution" points to define the geometry, and "resolution" as an argument to MSHR's automeshing too.

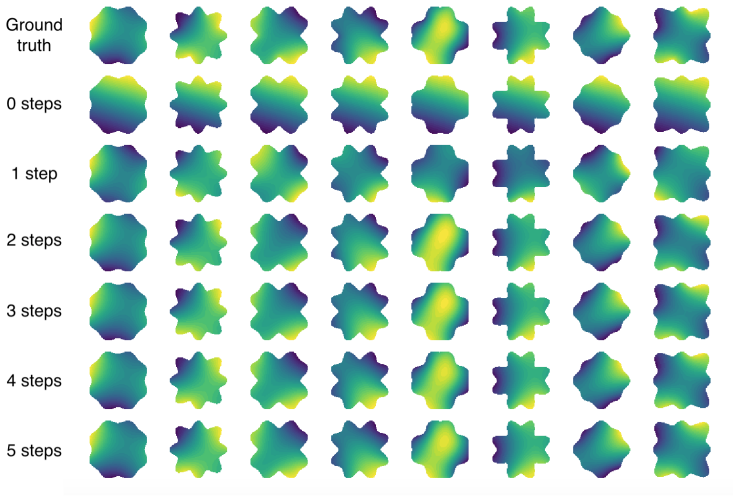


Figure 4.2: Solutions to nonlinear Poisson equations with varying domains, boundary conditions and source terms. Top: ground truth finite element solution. Second row: solution represented by Meta-PDE initial neural network parameters. Third row onwards: solution after each gradient step in the Meta-PDE inner loop.

error is computed by dividing the mean-squared error by the sum of squares of solution values for the ground-truth solution. various fidelities for sampled test problems. Figure 4.2 shows the ground truth and the Meta-PDE solution after zero through five gradient steps minimizing the variational energy for sampled test problems.

We see that Meta-PDE learns to output accurate solutions, and when run on the same CPU (a 2.5GHz Intel i7) is about 4-7x faster than a finite element method with similar accuracy, and about 50x more accurate than a finite element method with the same computation cost. Unlike finite element models, Meta-PDE can be easily accelerated by a GPU, and on GPU we see close to 50x speed up in deployment, leading to a 400x speed increase over similar accuracy finite element models. We expect these gains would only increase if we replaced the vanilla fully-connected neural network with a more tailored NN (such as the attention-like model used to fit PDE solutions in Wang et al.

(2020)), used a lower-variance sampling strategy, or more carefully tailored a meta-learning algorithm to this problem. The speed-accuracy tradeoff could also be tuned by changing the size of the NN used or the number of steps in the inner loop optimization.

4.7 Conclusion

We presented Meta-PDE, a surrogate model uses meta-learning to amortize PDE solving across classes of PDEs with complex and varying geometries and governing equations. Meta-PDE takes as input the governing equations themselves and a sampler for the PDE domain and boundary, thus remaining as close as possible in API to the fundamental representation of the PDE in terms of governing equations and domain. This avoids having to fix a parametric representation of geometry, governing equations and solution for the class of PDEs to be amortized. We show on a nonlinear Poisson problem that Meta-PDE can learn to output accurate solutions with a significantly more favorable accuracy-speed trade-off than a baseline finite element solver.

Chapter 5

Conclusion

In this thesis, we presented several methods using deep learning and stochastic gradient estimation to speed up numerical modeling. There is an increasing and justified interest in using neural networks and other tools from the discipline of machine learning to accelerate numerical procedures (as function approximation has been used to do for decades). As researchers and engineers increasingly integrate machine learning into numerical modeling, it will often be tempting to think about either machine learning or the numerical procedure or both as a black box – to use a numerical method as just a source of parameter vector, solution vector pairs for a regression dataset, or to use machine learning methods as just a tool for doing said regression. In certain scenarios, where data is plentiful and conforms to such a schema, such an approach can be appropriate and can have great results.

However, going beyond this shallow approach will let us develop methods which are of use in a far wider range of scenarios. In this thesis, we showed that tailoring machine learning methods to the computational and physical structure of numerical models can let us train models on cheap simulations and deploy them as surrogates for expensive simulations, or lets us remove the need for supervised data entirely, and lets us develop new efficient methods for optimization and system identification. Perhaps most importantly, synthesis of ML and numerical modeling can allow us to develop methods with new and more flexible APIs, and which allow acceleration of broader classes of numerical models.

Bibliography

Mohammad H Aliabadi. *The boundary element method, volume 2: applications in solids and structures*, volume 2. John Wiley & Sons, 2002.

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015a.

Martin S. Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E. Rognes, and Garth N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015b. doi: 10.11588/ans.2015.100.20553.

James Arvo and David Kirk. Particle transport and image synthesis. *ACM SIGGRAPH Computer Graphics*, 24(4):63–66, 1990.

Lukas Balles, Javier Romero, and Philipp Hennig. Coupling adaptive batch sizes with learning rates. In *Uncertainty in Artificial Intelligence*, 2016.

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Alex Beatson and Ryan P Adams. Efficient optimization of loops and limits with randomized telescoping sums. *arXiv preprint arXiv:1905.07006*, 2019.

Alex Beatson, Zhaoran Wang, and Han Liu. Blind attacks on machine learners. *Advances in Neural Information Processing Systems*, 2016.

Alex Beatson, Jordan Ash, Geoffrey Roeder, Tianju Xue, and Ryan P Adams. Learning composable energy surrogates for pde order reduction. *Advances in Neural Information Processing Systems*, 33, 2020.

Alex Beatson, Sunny T Qin, Nick McGreivy, and Ryan Prescott Adams. Meta-pde: Learning to solve pdes quickly without a mesh. *In preparation*, 2021.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Conference on Optimality in Artificial and Biological Neural Networks*, 1992.

Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, 1993.

Katia Bertoldi, Vincenzo Vitelli, Johan Christensen, and Martin van Hecke. Flexible mechanical metamaterials. *Nature Reviews Materials*, 2(11):1–11, 2017.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

Marcus Brubaker, Mathieu Salzmann, and Raquel Urtasun. A family of mcmc methods on implicitly defined manifolds. In *Artificial intelligence and statistics*, pages 161–172. PMLR, 2012.

Wenshan Cai and Vladimir M Shalaev. *Optical metamaterials*, volume 10. Springer, 2010.

Anindya Chatterjee. An introduction to the proper orthogonal decomposition. *Current science*, pages 808–817, 2000.

Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*, 2018.

Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114*, 5(6):12, 2011.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4278–4287, 2017.

Petros Drineas and Michael W Mahoney. Randnla: randomized numerical linear algebra. *Communications of the ACM*, 59(6):80–90, 2016.

Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on computing*, 36(1):158–183, 2006.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

Paul Fearnhead, Omiros Papaspiliopoulos, and Gareth O Roberts. Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70 (4):755–777, 2008.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1-3):50–79, 2009.

George E Forsythe and Richard A Leibler. Matrix inversion by a Monte Carlo method. *Mathematics of Computation*, 4(31):127–129, 1950.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, 2017.

Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6): 721–741, 1984.

Mark Girolami, Anne-Marie Lyne, Heiko Strathmann, Daniel Simpson, and Yves Atchade. Playing Russian roulette with intractable likelihoods. Technical report, Citeseer, 2013.

Robert J Guyan. Reduction of stiffness and mass matrices. *AIAA journal*, 3(2):380–380, 1965.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

Nicholas J Higham, Mark R Dennis, Paul Glendinning, Paul A Martin, Fadil Santosa, and Jared Tanner. *The Princeton companion to applied mathematics*. Princeton University Press, 2015.

Alan J Hoffman, Michael S Martin, and Donald J Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM Journal on Numerical Analysis*, 10(2):364–369, 1973.

Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch. Metamaterial mechanisms. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 529–539, 2016.

Pierre E Jacob, John O'Leary, and Yves F Atchadé. Unbiased Markov chain Monte Carlo with couplings. *arXiv preprint arXiv:1708.03625*, 2017.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, pages 1627–1635. JMLR. org, 2017.

Antony Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3): 233–260, 1988.

Herman Kahn. Use of different Monte Carlo sampling techniques. 1955.

Yoon Kim, Sam Wiseman, Andrew C Miller, David Sontag, and Alexander M Rush. Semi-amortized variational autoencoders. In *International conference on Machine Learning*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for optimization*. MIT Press, 2019.

Slawomir Koziel and Leifur Leifsson. *Surrogate-based modeling and optimization*. Springer, 2013.

Julius Kuti. Stochastic method for the numerical study of lattice fermions. *Physical Review Letters*, 49(3):183, 1982.

Prem K Kythe, Dongming Wei, and M Okrouhlik. An introduction to linear and nonlinear finite element analysis: a computational approach. *Appl. Mech. Rev.*, 57(5):B25–B25, 2004.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

J.L. Lions. *Optimal control of systems governed by partial differential equations*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1971.

Jun S Liu. The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, 89(427):958–966, 1994.

Anders Logg and Garth N Wells. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):1–28, 2010.

Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012a. doi: 10.1007/978-3-642-23099-8.

Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012b. doi: 10.1007/978-3-642-23099-8.

Anders Logg, Garth N Wells, and Johan Hake. DOLFIN: A C++/Python finite element library. In *Automated solution of differential equations by the finite element method*, pages 173–225. Springer, 2012c.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

Yucen Luo, Alex Beatson, Mohammad Norouzi, Jun Zhu, David Duvenaud, Ryan P Adams, and Ricky TQ Chen. Sumo: Unbiased estimation of log marginal probability for latent variable models. In *ICLR 2020*, 2019a.

Yucen Luo, Alex Beatson, Mohammad Norouzi, Jun Zhu, David Duvenaud, Ryan P Adams, and Ricky TQ Chen. Sumo: Unbiased estimation of log marginal probability for latent variable models. In *ICLR 2020*, 2019b.

Anne-Marie Lyne, Mark Girolami, Yves Atchadé, Heiko Strathmann, Daniel Simpson, et al. On Russian roulette estimates for Bayesian inference with doubly-intractable likelihoods. *Statistical science*, 30(4):443–467, 2015.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

Jay D Martin and Timothy W Simpson. Use of kriging models to approximate deterministic computer models. *AIAA journal*, 43(4):853–863, 2005.

Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3):037106, 2021.

Don McLeish. A general method for debiasing a Monte Carlo estimator. *Monte Carlo Methods and Applications*, 2010.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182*, 2017.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, C Daniel Freeman, and Jascha Sohl-Dickstein. Learned optimizers that outperform SGD on wall-clock and validation loss. *arXiv preprint arXiv:1810.10180*, 2018.

Sebastian Mitusch, Simon Funke, and Jørgen Dokken. dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake. *Journal of Open Source Software*, 4(38):1292, 2019.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, 2018.

Zhenguo Nie, Haoliang Jiang, and Levent Burak Kara. Stress field prediction in cantilevered structures using convolutional neural networks. *Journal of Computing and Information Science in Engineering*, 20(1), 2020.

R.W. Ogden. *Non-linear Elastic Deformations*. Dover Civil and Mechanical Engineering. Dover Publications, 1997.

Deniz Oktay, Nick McGreivy, Joshua Aduol, Alex Beatson, and Ryan P Adams. Randomized automatic differentiation. *ICLR 2021*, 2021.

Naoki Osada. Acceleration methods for vector sequences. *Journal of Computational and Applied Mathematics*, 38(1-3):361–371, 1991.

Johannes TB Overvelde and Katia Bertoldi. Relating pore shape to the non-linear response of periodic elastomeric structures. *Journal of the Mechanics and Physics of Solids*, 64:351–366, 2014.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Alexander Poddubny, Ivan Iorsh, Pavel Belov, and Yuri Kivshar. Hyperbolic metamaterials. *Nature photonics*, 7(12):948, 2013.

Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

Maziar Raissi, Paris Perdikaris, and George 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.

Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. In *ICLR 2019*, 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

Chang-han Rhee and Peter W Glynn. A new approach to unbiased estimation for SDEs. In *Proceedings of the Winter Simulation Conference*, page 17. Winter Simulation Conference, 2012.

Chang-han Rhee and Peter W Glynn. Unbiased estimation with square root convergence for SDE models. *Operations Research*, 63(5):1026–1043, 2015.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

Walter Rudin et al. *Principles of Mathematical Analysis*, volume 3. McGraw-hill New York, 1976.

Tomasz Rychlik. Unbiased nonparametric estimation of the derivative of the mean. *Statistics & probability letters*, 10(4):329–333, 1990.

Tomasz Rychlik. A class of unbiased kernel estimates of a probability density function. *Applicationes Mathematicae*, 22(4):485–497, 1995.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Jörg Schröder. A numerical two-scale homogenization scheme: the fe 2-method. In *Plasticity and beyond*, pages 1–64. Springer, 2014.

Yannick Schroecker and Charles L Isbell. State aware imitation learning. In *Advances in Neural Information Processing Systems*, pages 2911–2920, 2017.

Ari Seff, Alex Beatson, Daniel Suo, and Han Liu. Continual learning in generative adversarial nets. In *arXiv preprint*, 2017.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. *arXiv preprint arXiv:1810.10667*, 2018.

Amar Shah, Andrew Wilson, and Zoubin Ghahramani. Student-t processes as alternatives to Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 877–885, 2014.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Daichao Sheng, Scott W Sloan, and Andrew J Abbo. An automatic Newton–Raphson scheme. *The International Journal Geomechanics*, 2(4):471–502, 2002.

Timothy W Simpson, JD Poplinski, Patrick N Koch, and Janet K Allen. Metamodels for computer-based engineering design: survey and recommendations. *Engineering with computers*, 17(2):129–150, 2001.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

Jerome Spanier and Ely M Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley Publishing Company, 1969.

Endre Süli and David F Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Yee W Teh, David Newman, and Max Welling. A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 1353–1360, 2007.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

Trieu H Trinh, Andrew M Dai, Thang Luong, and Quoc V Le. Learning longer-term dependencies in RNNs with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, pages 5998–6008, 2017.

Wolfgang Wagner. Unbiased Monte Carlo evaluation of certain functional integrals. *Journal of Computational Physics*, 71(1):21–33, 1987.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020.

Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*, 2019.

Colin Wei and Iain Murray. Markov chain truncation for doubly-intractable inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020a.

Tianju Xue, Alex Beatson, Maurizio Chiaramonte, Geoffrey Roeder, Jordan T Ash, Yigit Menguc, Sigrid Adriaenssens, Ryan P Adams, and Sheng Mao. A data-driven computational scheme for the nonlinear mechanical properties of cellular mechanical metamaterials under large deformation. *Soft matter*, 16(32):7524–7534, 2020b.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

Appendix A

List of publications

A.0.1 Chapter 2

- Alex Beatson, Jordan Ash, Geoffrey Roeder, Tianju Xue, and Ryan P Adams. Learning composable energy surrogates for pde order reduction. *Advances in Neural Information Processing Systems*, 33, 2020

A.0.2 Chapter 3

- Alex Beatson and Ryan P Adams. Efficient optimization of loops and limits with randomized telescoping sums. *arXiv preprint arXiv:1905.07006*, 2019

A.0.3 Chapter 4

- Alex Beatson, Sunny T Qin, Nick McGreivy, and Ryan Prescott Adams. Meta-pde: Learning to solve pdes quickly without a mesh. *In preparation*, 2021

A.0.4 Not included in this thesis

- Deniz Oktay, Nick McGreivy, Joshua Aduol, Alex Beatson, and Ryan P Adams. Randomized automatic differentiation. *ICLR 2021*, 2021

- Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020a

- Tianju Xue, Alex Beatson, Maurizio Chiaramonte, Geoffrey Roeder, Jordan T Ash, Yigit Menguc, Sigrid Adriaenssens, Ryan P Adams, and Sheng Mao. A data-driven computational scheme for the nonlinear mechanical properties of cellular mechanical metamaterials under large deformation. *Soft matter*, 16(32):7524–7534, 2020b
- Yucen Luo, Alex Beatson, Mohammad Norouzi, Jun Zhu, David Duvenaud, Ryan P Adams, and Ricky TQ Chen. Sumo: Unbiased estimation of log marginal probability for latent variable models. In *ICLR 2020*, 2019b
- Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. In *ICLR 2019*, 2018
- Ari Seff, Alex Beatson, Daniel Suo, and Han Liu. Continual learning in generative adversarial nets. In *arXiv preprint*, 2017
- Alex Beatson, Zhaoran Wang, and Han Liu. Blind attacks on machine learners. *Advances in Neural Information Processing Systems*, 2016

Appendix B

Appendix for Chapter 2

B.1 Contents

This appendix consists of:

- specification of the data generating distribution and hyperparameters;
- visualization of data generated via Hamiltonian Monte Carlo;
- visualization of data generated via DAgger;
- hyperparameters used for neural network specification and training;
- ablation study of neural network design choices;
- specification of the finite element meshes used as baselines;
- visualization of all solutions found under compression and tension for each pore shape for each baseline mesh and the composed energy surrogate.

B.2 Data generation with Hamiltonian Monte Carlo

We use 100 data collectors, each with randomly drawn hyperparameters, which each terminate (and have their place taken by a newly initialized collector) after sampling 25 data points. We collect 60,000 data points, consisting of a training set of 55,000 and a validation set of 5,000. As our distribution is arbitrary, and as we assume that more data is always a good thing, when a HMC sample is rejected, we still add it to the dataset, but return to the last un-rejected sample to continue the Markov chain.

Hyperparameter distributions are chosen heuristically such that the finite element simulation tends to converge in a reasonable amount of time. The hyperparameter distributions are as follows:

- Leapfrog step size: $\mathcal{U}(0.005, 0.02)$
- Leapfrog path length: $\mathcal{U}(0.05, 0.3)$
- Temperature used to scale the log-probability: $\mathcal{U}([0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1])$
- Standard deviation of the Gaussian from which the Hamiltonian momentum is drawn: $\mathcal{U}(0.01, 0.3)$

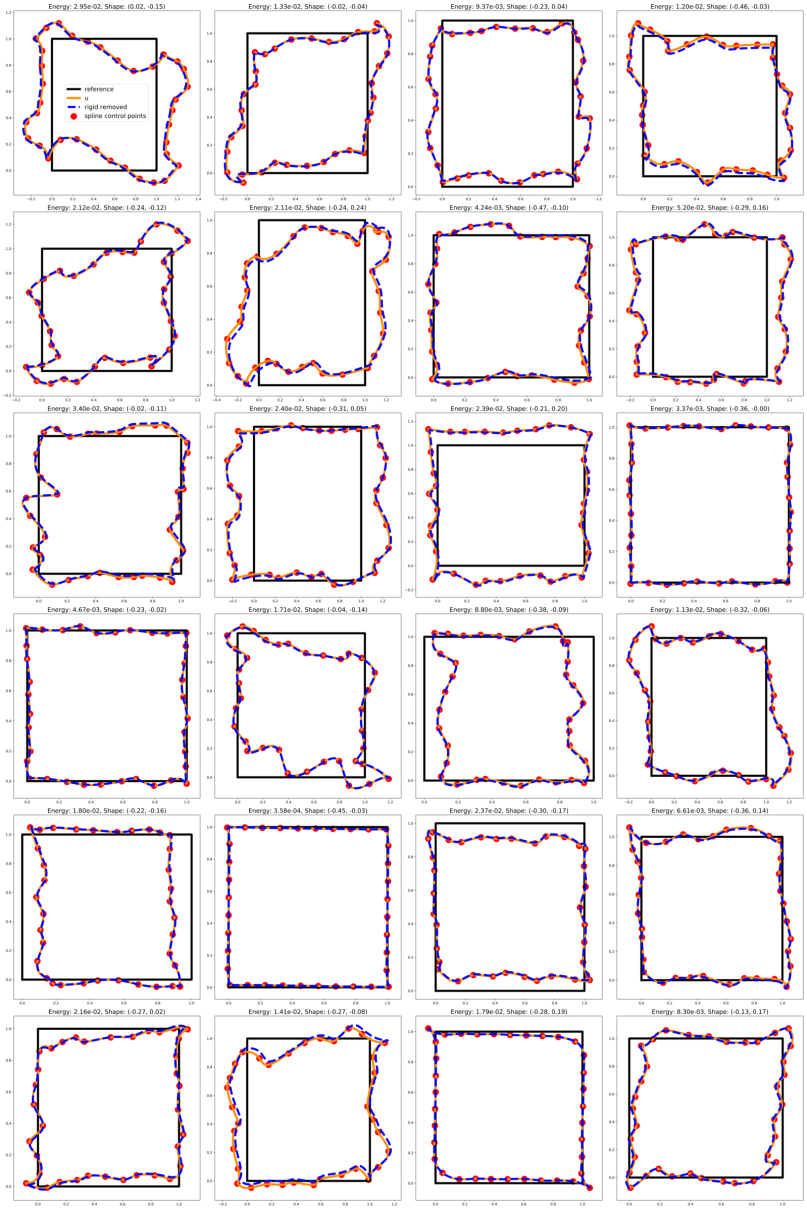
We approximate the macroscopic strain tensor $\bar{x}$ from $\mathbf{u}$ as:

$$\bar{x}(\mathbf{u}) = \frac{1}{N} \begin{bmatrix} \sum_{X \in \text{rhs}} u_1(X) - \sum_{X \in \text{lhs}} u_1(x) & \sum_{X \in \text{rhs}} u_2(X) - \sum_{X \in \text{lhs}} u_2(X) \\ \sum_{X \in \text{top}} u_1(X) - \sum_{X \in \text{bot}} u_1(x) & \sum_{X \in \text{top}} u_2(X) - \sum_{X \in \text{bot}} u_2(X) \end{bmatrix}$$

Above, $u_1(X)$ and $u_2(X)$ are horizontal and vertical displacements defined by $\mathbf{u}$ at a point X, and top, bot, lhs and rhs are the set of control point locations for the splines on the top, bottom, left and right of the component.

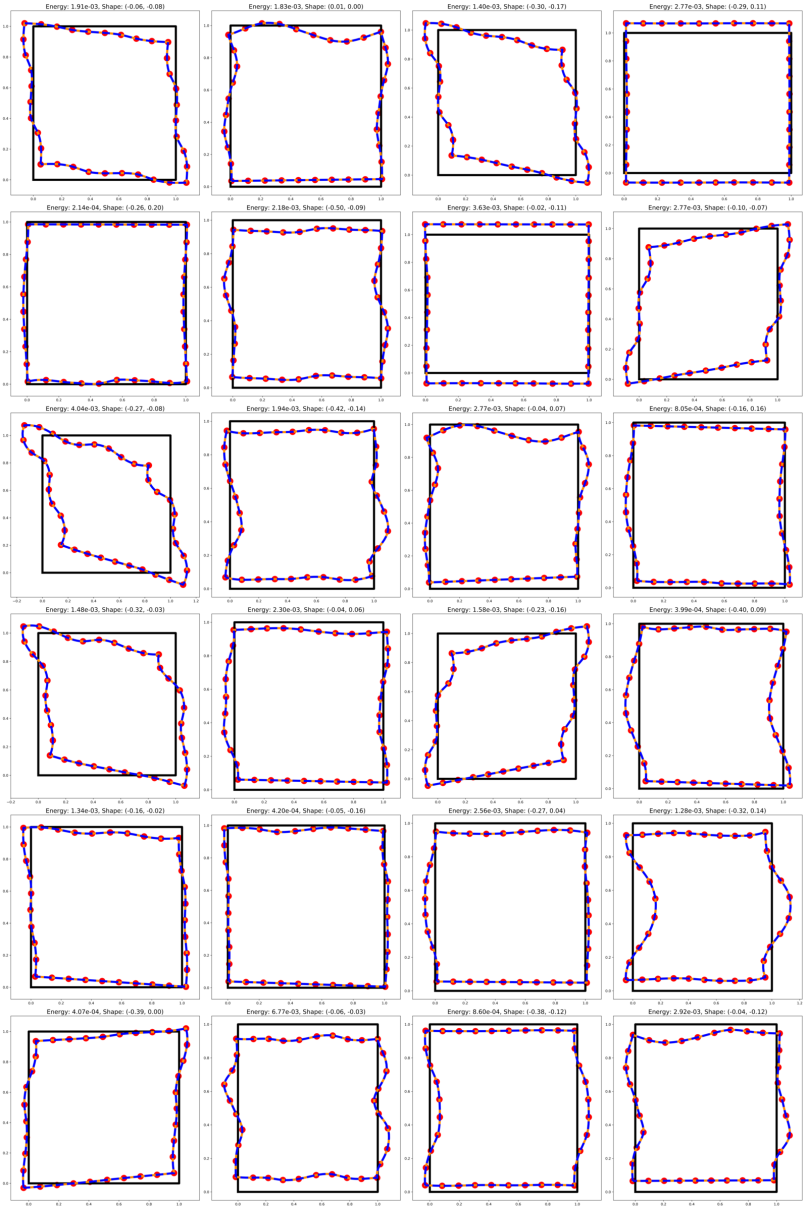
B.3 Visualizing HMC data

Here we display 24 randomly chosen examples from the training set.



B.4 Visualizing DAgger data

Here we display 24 randomly chosen examples from the data collected with DAgger.



B.5 Neural network hyperparameters

We use a fully-connected neural network with three hidden layers of 512 units, Swish nonlinarities, and He initialization. We optimize our neural network using Adam with a learning rate of 3e-4 and a batch size of 512.

B.6 Surrogate design ablation study

We perform an ablation study by switching on and off the following independent variables:

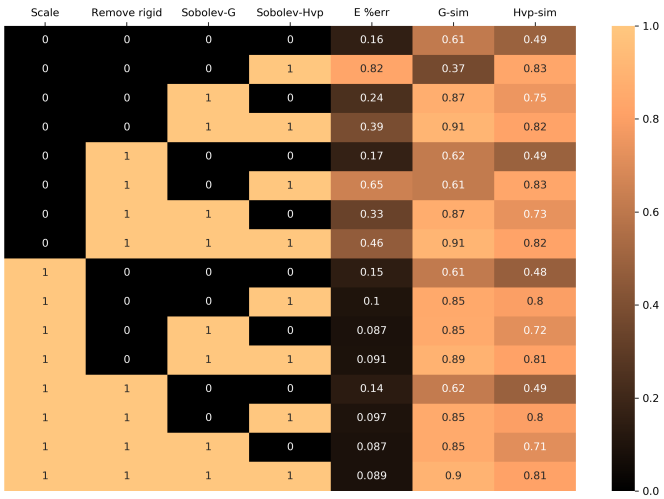
- "Scale": parameterizing the log of the scalar stiffness, vs parameterizing energy directly;
- "Remove rigid": removing rigid body translations from the data via Procrustes analysis;
- "Sobolev-G": Sobolev training on energy gradients;
- "Sobolev-Hvp": Sobolev training on energy Hessian-vector products.

We measure performance after 10,000 training steps (93 epochs) on the training set, without DAgger. We evaluate each model on the validation dataset using the following metrics:

- "E %err": the error in predicted energy, expressed as a percentage of the true energy;
- "G-sim": the cosine similarity between predicted and true gradients;
- "Hvp-sim": the cosine similarity between predicted and true Hessian-vector products.

Results are shown below. For the independent variables, a value of '1' indicates that method or technique was turned on, while a value of '0' indicates it was turned off.

Each design choice improves the validation metrics. "Remove rigid" has marginal impact, as our training displacements contain little rigid body transformation. We leave this feature in as it causes no harm; as it improved performance under earlier dataset creation methods which resulted in more rigid body translation; and as removing translations before computing energy is necessary to be able to compose energy surrogates by tiling.



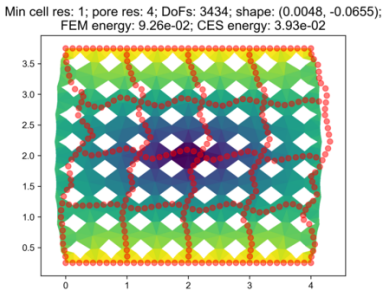
B.7 Finite element baselines

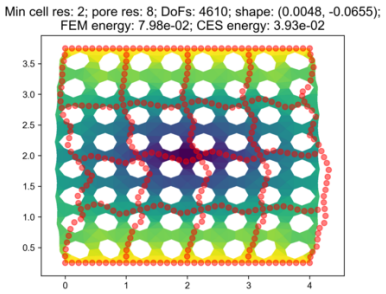
We generate meshes for the finite element baselines using two parameters: pore resolution, and minimum mesh resolution. Firstly, for each pore in the cellular solid, we generate a polygon representing that pore using a number of points equal to pore resolution. We let the material domain be the overall volume of the solid with these polygons subtracted. Next, we generate a mesh over the material domain using MSHR's automated mesh generation routine, passing as resolution minimum mesh resolution multiplied by the number of cells. In MSHR, the resolution parameter controls the maximum cell size, which is the diameter of the domain's bounding circle divided by the resolution. It should be noted that cells can be much smaller than this maximum size, or there can be many more cells than the resolution parameter would imply, as MSHR will place one cell vertex on each point used to construct the domain geometry (i.e. each point in the pore polygon).

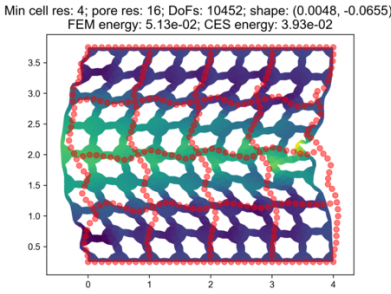
B.8 Benchmark visualizations

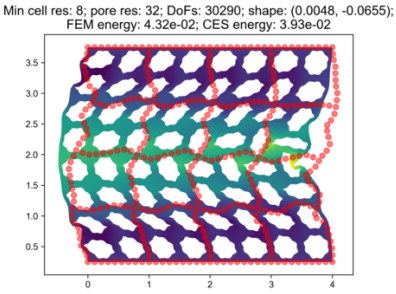
In the following pages we visualize the solutions found for each pore by each FEA mesh and by CES. For each pore we use six different finite element meshes. These respectively used $[4, 8, 16, 32, 48, 64]$ points used to define the geometry of each pore, and minimum of $[1, 2, 4, 8, 12, 16]$ internal mesh vertices along a given axis per pore. Given these parameters and the geometry of the material domain, meshes were created using the automatic mesh generation tool from mshr (the mesh generation component of FEniCS). We include the number of degrees of freedom in the finite element basis in each plot. We superimpose the solution found with CES in red dots on the solution found with FEA. The CES solution has 690 degrees of freedom in all cases.

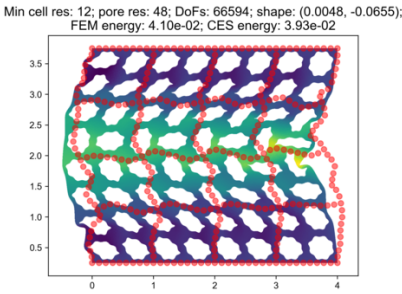
B.8.1 Compression

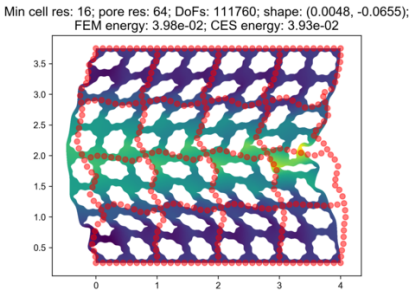




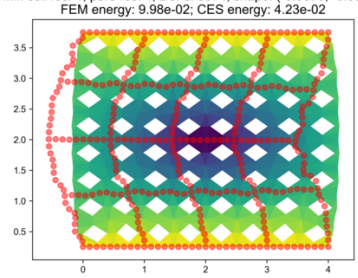




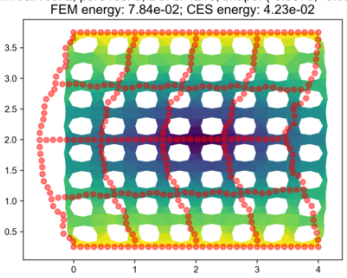




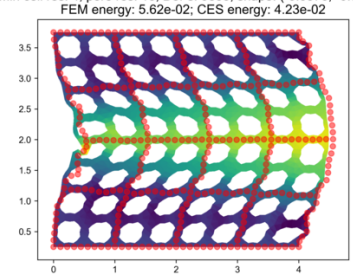
Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.0576, -0.0379);
FEM energy: 9.98e-02; CES energy: 4.23e-02



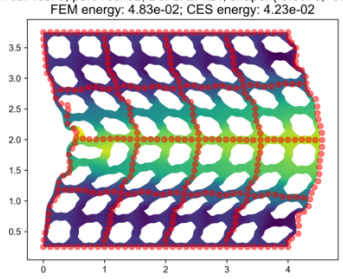
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.0576, -0.0379);
FEM energy: 7.84e-02; CES energy: 4.23e-02



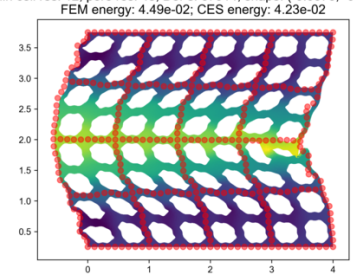
Min cell res: 4; pore res: 16; DoFs: 9858; shape: (-0.0576, -0.0379);
FEM energy: 5.62e-02; CES energy: 4.23e-02



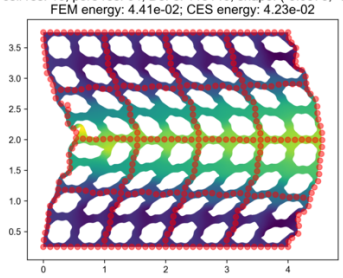
Min cell res: 8; pore res: 32; DoFs: 27724; shape: (-0.0576, -0.0379);
FEM energy: 4.83e-02; CES energy: 4.23e-02



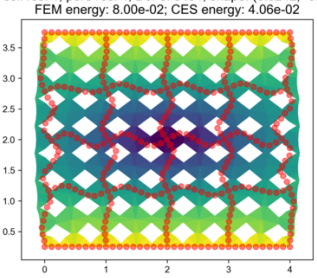
Min cell res: 12; pore res: 48; DoFs: 67914; shape: (-0.0576, -0.0379);
FEM energy: 4.49e-02; CES energy: 4.23e-02



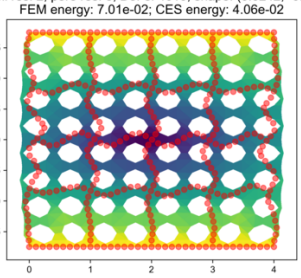
Min cell res: 16; pore res: 64; DoFs: 113146; shape: (-0.0576, -0.0379)
FEM energy: 4.41e-02; CES energy: 4.23e-02



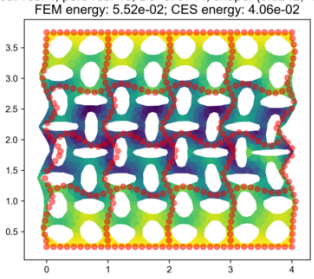
Min cell res: 1; pore res: 4; DoFs: 3434; shape: (0.0242, -0.0153);
FEM energy: 8.00e-02; CES energy: 4.06e-02



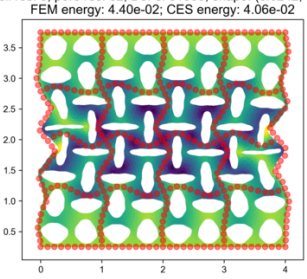
Min cell res: 2; pore res: 8; DoFs: 4610; shape: (0.0242, -0.0153);
FEM energy: 7.01e-02; CES energy: 4.06e-02



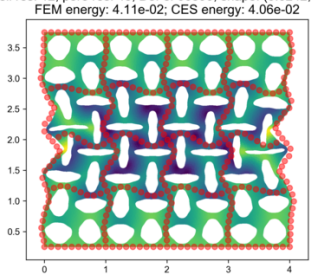
Min cell res: 4; pore res: 16; DoFs: 9744; shape: (0.0242, -0.0153);
FEM energy: 5.52e-02; CES energy: 4.06e-02



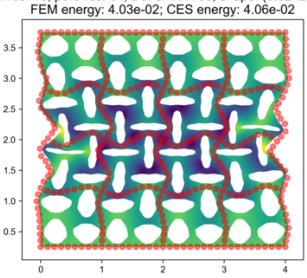
Min cell res: 8; pore res: 32; DoFs: 31906; shape: (0.0242, -0.0153);
FEM energy: 4.40e-02; CES energy: 4.06e-02

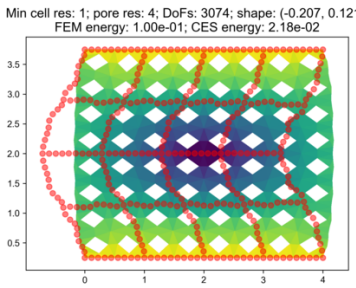


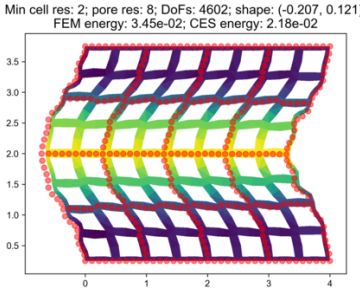
Min cell res: 12; pore res: 48; DoFs: 66890; shape: (0.0242, -0.0153);
FEM energy: 4.11e-02; CES energy: 4.06e-02

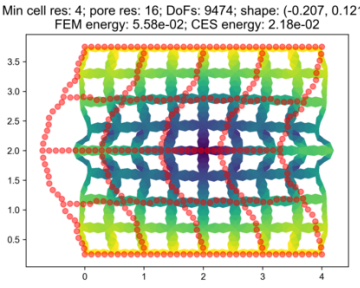


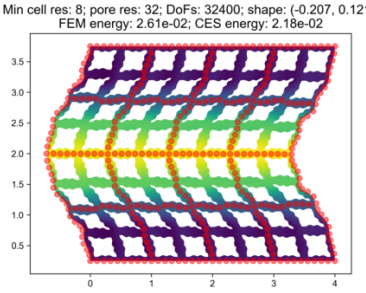
Min cell res: 16; pore res: 64; DoFs: 111766; shape: (0.0242, -0.0153);
FEM energy: 4.03e-02; CES energy: 4.06e-02

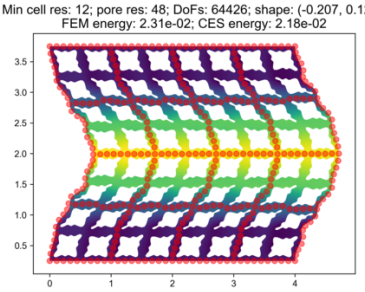


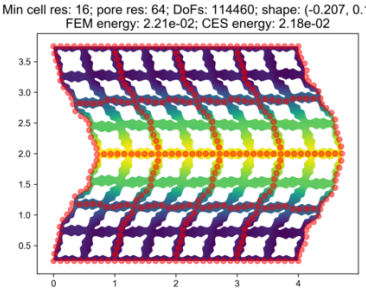

Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.207, 0.121);
FEM energy: 1.00e-01; CES energy: 2.18e-02


Min cell res: 2; pore res: 8; DoFs: 4602; shape: (-0.207, 0.121);
FEM energy: 3.45e-02; CES energy: 2.18e-02

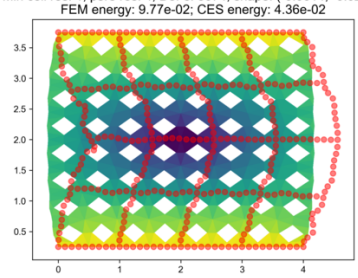

Min cell res: 4; pore res: 16; DoFs: 9474; shape: (-0.207, 0.121);
FEM energy: 5.58e-02; CES energy: 2.18e-02


Min cell res: 8; pore res: 32; DoFs: 32400; shape: (-0.207, 0.121);
FEM energy: 2.61e-02; CES energy: 2.18e-02

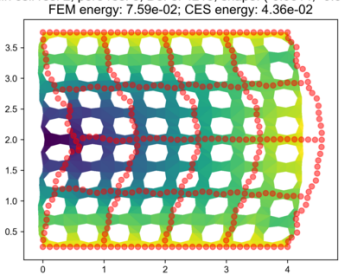

Min cell res: 12; pore res: 48; DoFs: 64426; shape: (-0.207, 0.121);
FEM energy: 2.31e-02; CES energy: 2.18e-02


Min cell res: 16; pore res: 64; DoFs: 114460; shape: (-0.207, 0.121);
FEM energy: 2.21e-02; CES energy: 2.18e-02

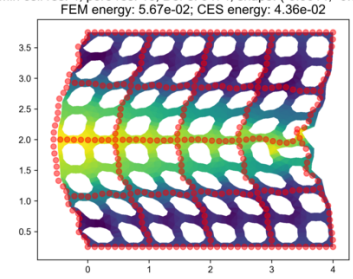
Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.0614, -0.0228);
FEM energy: 9.77e-02; CES energy: 4.36e-02



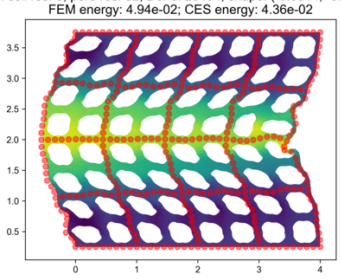
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.0614, -0.0228);
FEM energy: 7.59e-02; CES energy: 4.36e-02



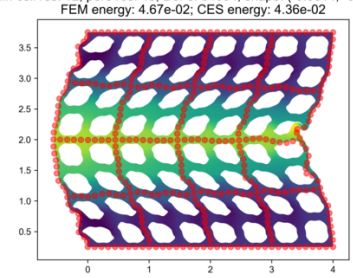
Min cell res: 4; pore res: 16; DoFs: 9474; shape: (-0.0614, -0.0228);
FEM energy: 5.67e-02; CES energy: 4.36e-02



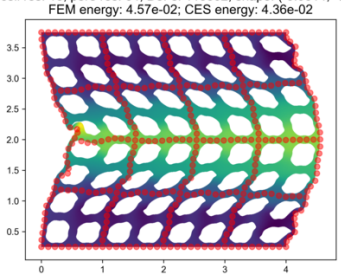
Min cell res: 8; pore res: 32; DoFs: 29164; shape: (-0.0614, -0.0228);
FEM energy: 4.94e-02; CES energy: 4.36e-02

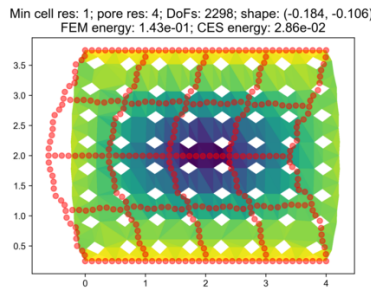


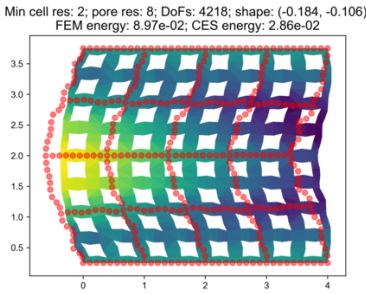
Min cell res: 12; pore res: 48; DoFs: 64634; shape: (-0.0614, -0.0228);
FEM energy: 4.67e-02; CES energy: 4.36e-02

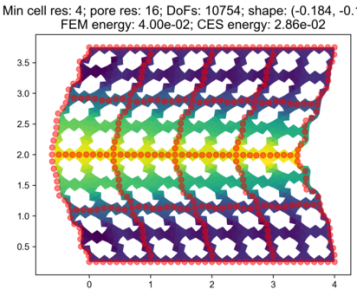


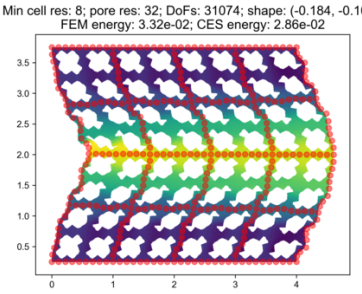
Min cell res: 16; pore res: 64; DoFs: 113802; shape: (-0.0614, -0.0228)
FEM energy: 4.57e-02; CES energy: 4.36e-02

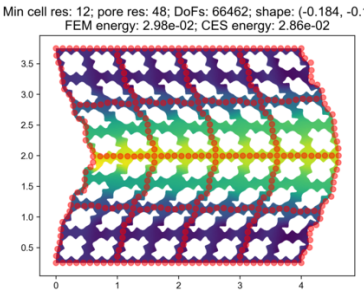


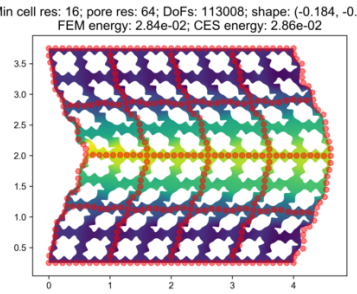

Min cell res: 1; pore res: 4; DoFs: 2298; shape: (-0.184, -0.106);
FEM energy: 1.43e-01; CES energy: 2.86e-02


Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.184, -0.106);
FEM energy: 8.97e-02; CES energy: 2.86e-02

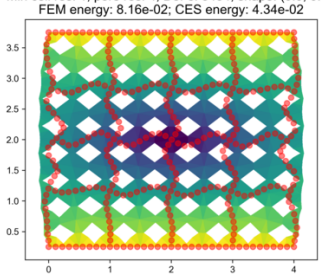

Min cell res: 4; pore res: 16; DoFs: 10754; shape: (-0.184, -0.106);
FEM energy: 4.00e-02; CES energy: 2.86e-02


Min cell res: 8; pore res: 32; DoFs: 31074; shape: (-0.184, -0.106);
FEM energy: 3.32e-02; CES energy: 2.86e-02

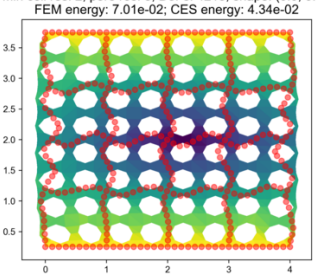

Min cell res: 12; pore res: 48; DoFs: 66462; shape: (-0.184, -0.106);
FEM energy: 2.98e-02; CES energy: 2.86e-02


Min cell res: 16; pore res: 64; DoFs: 113008; shape: (-0.184, -0.106);
FEM energy: 2.84e-02; CES energy: 2.86e-02

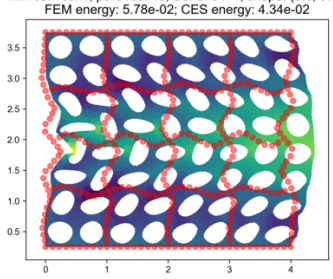
Min cell res: 1; pore res: 4; DoFs: 3434; shape: (0.0, 0.0);
FEM energy: 8.16e-02; CES energy: 4.34e-02



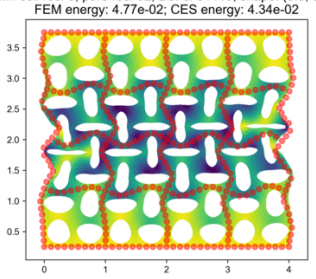
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (0.0, 0.0);
FEM energy: 7.01e-02; CES energy: 4.34e-02



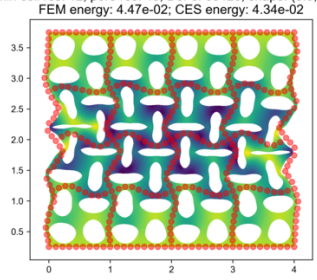
Min cell res: 4; pore res: 16; DoFs: 9474; shape: (0.0, 0.0);
FEM energy: 5.78e-02; CES energy: 4.34e-02



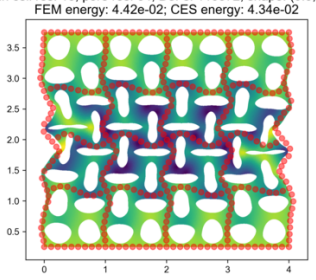
Min cell res: 8; pore res: 32; DoFs: 31410; shape: (0.0, 0.0);
FEM energy: 4.77e-02; CES energy: 4.34e-02



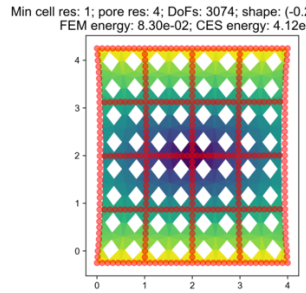
Min cell res: 12; pore res: 48; DoFs: 68426; shape: (0.0, 0.0);
FEM energy: 4.47e-02; CES energy: 4.34e-02

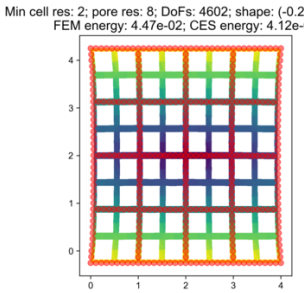


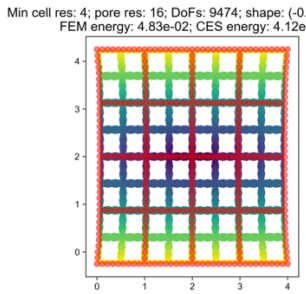
Min cell res: 16; pore res: 64; DoFs: 110372; shape: (0.0, 0.0);
FEM energy: 4.42e-02; CES energy: 4.34e-02

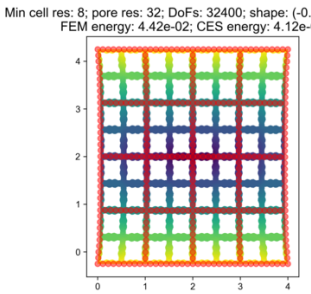


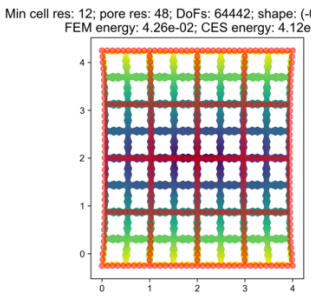
B.8.2 Tension

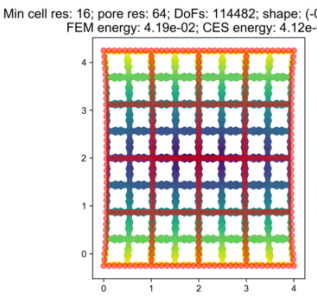

Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.207, 0.121);
FEM energy: 8.30e-02; CES energy: 4.12e-02


Min cell res: 2; pore res: 8; DoFs: 4602; shape: (-0.207, 0.121);
FEM energy: 4.47e-02; CES energy: 4.12e-02

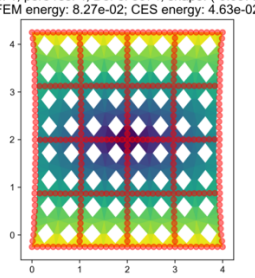

Min cell res: 4; pore res: 16; DoFs: 9474; shape: (-0.207, 0.121);
FEM energy: 4.83e-02; CES energy: 4.12e-02


Min cell res: 8; pore res: 32; DoFs: 32400; shape: (-0.207, 0.121);
FEM energy: 4.42e-02; CES energy: 4.12e-02

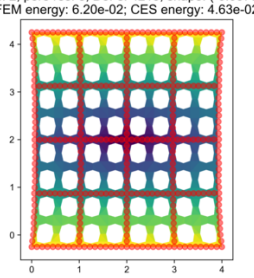

Min cell res: 12; pore res: 48; DoFs: 64442; shape: (-0.207, 0.121);
FEM energy: 4.26e-02; CES energy: 4.12e-02


Min cell res: 16; pore res: 64; DoFs: 114482; shape: (-0.207, 0.121);
FEM energy: 4.19e-02; CES energy: 4.12e-02

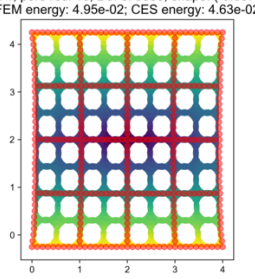
Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.0576, -0.0379);
FEM energy: 8.27e-02; CES energy: 4.63e-02



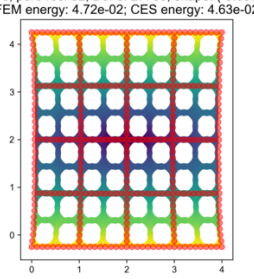
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.0576, -0.0379);
FEM energy: 6.20e-02; CES energy: 4.63e-02



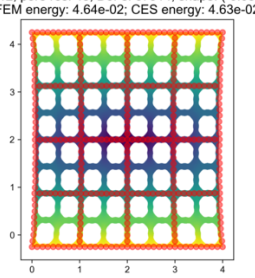
Min cell res: 4; pore res: 16; DoFs: 9858; shape: (-0.0576, -0.0379);
FEM energy: 4.95e-02; CES energy: 4.63e-02



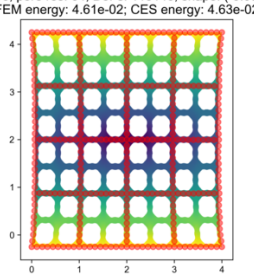
Min cell res: 8; pore res: 32; DoFs: 27730; shape: (-0.0576, -0.0379);
FEM energy: 4.72e-02; CES energy: 4.63e-02



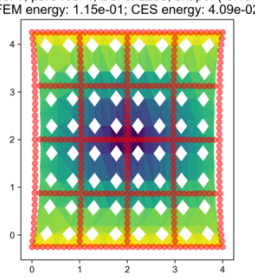
Min cell res: 12; pore res: 48; DoFs: 67914; shape: (-0.0576, -0.0379);
FEM energy: 4.64e-02; CES energy: 4.63e-02



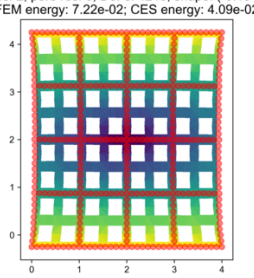
Min cell res: 16; pore res: 64; DoFs: 113146; shape: (-0.0576, -0.0379)
FEM energy: 4.61e-02; CES energy: 4.63e-02



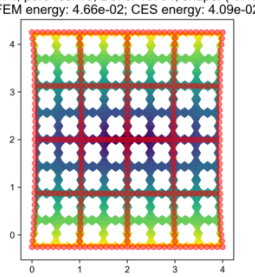
Min cell res: 1; pore res: 4; DoFs: 2298; shape: (-0.184, -0.106);
FEM energy: 1.15e-01; CES energy: 4.09e-02



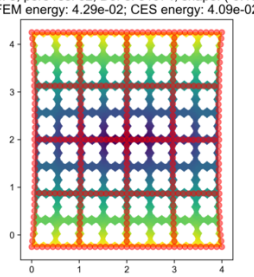
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.184, -0.106);
FEM energy: 7.22e-02; CES energy: 4.09e-02



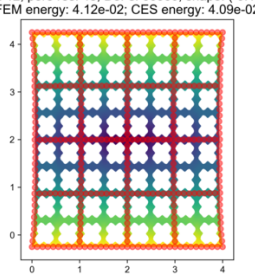
Min cell res: 4; pore res: 16; DoFs: 10754; shape: (-0.184, -0.106);
FEM energy: 4.66e-02; CES energy: 4.09e-02



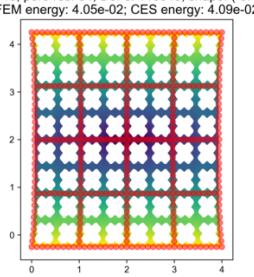
Min cell res: 8; pore res: 32; DoFs: 31074; shape: (-0.184, -0.106);
FEM energy: 4.29e-02; CES energy: 4.09e-02



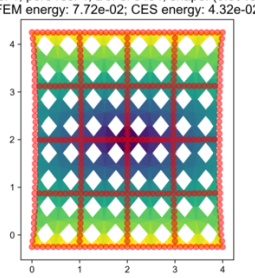
Min cell res: 12; pore res: 48; DoFs: 66560; shape: (-0.184, -0.106);
FEM energy: 4.12e-02; CES energy: 4.09e-02



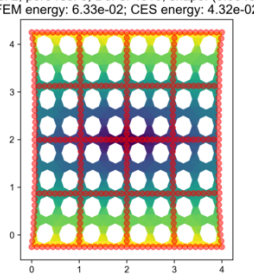
Min cell res: 16; pore res: 64; DoFs: 113048; shape: (-0.184, -0.106);
FEM energy: 4.05e-02; CES energy: 4.09e-02



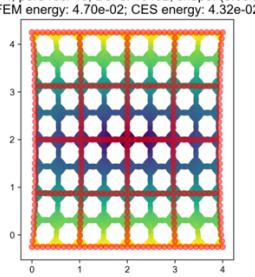
Min cell res: 1; pore res: 4; DoFs: 3434; shape: (0.0048, -0.0655);
FEM energy: 7.72e-02; CES energy: 4.32e-02



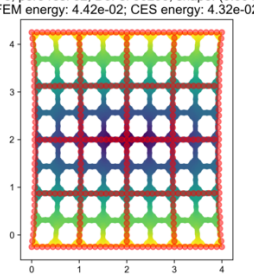
Min cell res: 2; pore res: 8; DoFs: 4610; shape: (0.0048, -0.0655);
FEM energy: 6.33e-02; CES energy: 4.32e-02



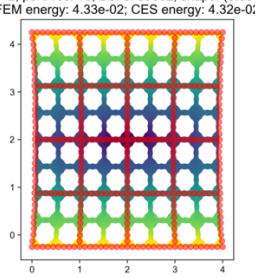
Min cell res: 4; pore res: 16; DoFs: 10452; shape: (0.0048, -0.0655);
FEM energy: 4.70e-02; CES energy: 4.32e-02



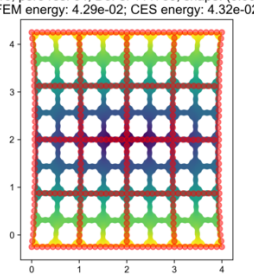
Min cell res: 8; pore res: 32; DoFs: 30290; shape: (0.0048, -0.0655);
FEM energy: 4.42e-02; CES energy: 4.32e-02



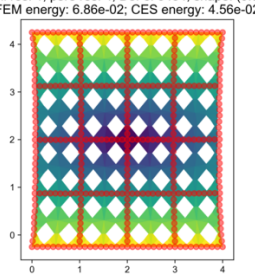
Min cell res: 12; pore res: 48; DoFs: 66602; shape: (0.0048, -0.0655);
FEM energy: 4.33e-02; CES energy: 4.32e-02



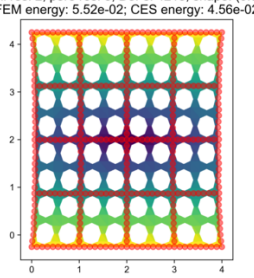
Min cell res: 16; pore res: 64; DoFs: 111760; shape: (0.0048, -0.0655);
FEM energy: 4.29e-02; CES energy: 4.32e-02



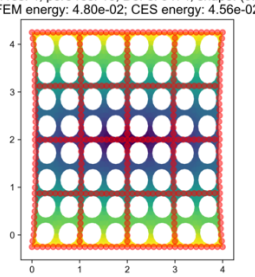
Min cell res: 1; pore res: 4; DoFs: 3434; shape: (0.0, 0.0);
FEM energy: 6.86e-02; CES energy: 4.56e-02



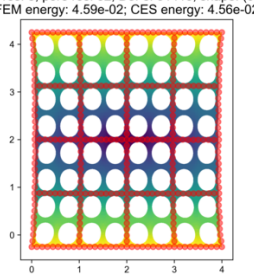
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (0.0, 0.0);
FEM energy: 5.52e-02; CES energy: 4.56e-02



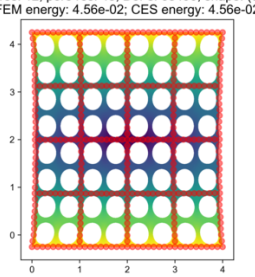
Min cell res: 4; pore res: 16; DoFs: 9474; shape: (0.0, 0.0);
FEM energy: 4.80e-02; CES energy: 4.56e-02



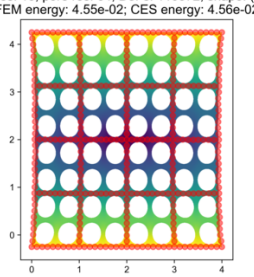
Min cell res: 8; pore res: 32; DoFs: 31418; shape: (0.0, 0.0);
FEM energy: 4.59e-02; CES energy: 4.56e-02



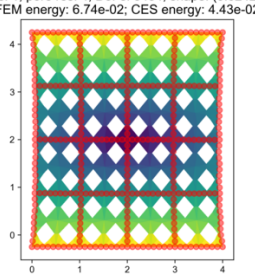
Min cell res: 12; pore res: 48; DoFs: 68490; shape: (0.0, 0.0);
FEM energy: 4.56e-02; CES energy: 4.56e-02



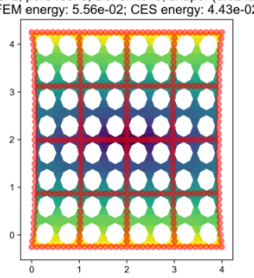
Min cell res: 16; pore res: 64; DoFs: 110372; shape: (0.0, 0.0);
FEM energy: 4.55e-02; CES energy: 4.56e-02



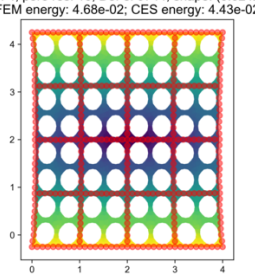
Min cell res: 1; pore res: 4; DoFs: 3434; shape: (0.0242, -0.0153);
FEM energy: 6.74e-02; CES energy: 4.43e-02



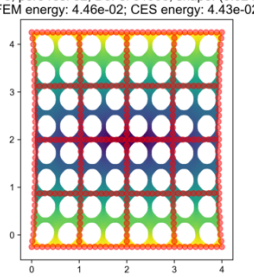
Min cell res: 2; pore res: 8; DoFs: 4610; shape: (0.0242, -0.0153);
FEM energy: 5.56e-02; CES energy: 4.43e-02



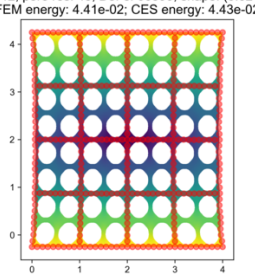
Min cell res: 4; pore res: 16; DoFs: 9744; shape: (0.0242, -0.0153);
FEM energy: 4.68e-02; CES energy: 4.43e-02



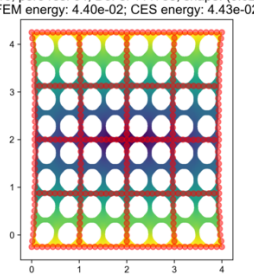
Min cell res: 8; pore res: 32; DoFs: 31906; shape: (0.0242, -0.0153);
FEM energy: 4.46e-02; CES energy: 4.43e-02



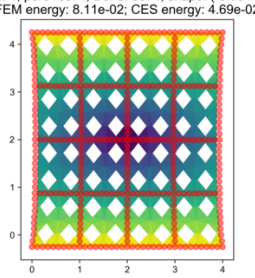
Min cell res: 12; pore res: 48; DoFs: 66890; shape: (0.0242, -0.0153);
FEM energy: 4.41e-02; CES energy: 4.43e-02



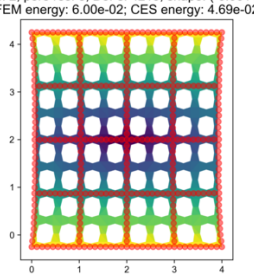
Min cell res: 16; pore res: 64; DoFs: 111758; shape: (0.0242, -0.0153);
FEM energy: 4.40e-02; CES energy: 4.43e-02



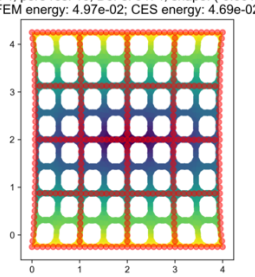
Min cell res: 1; pore res: 4; DoFs: 3074; shape: (-0.0614, -0.0228);
FEM energy: 8.11e-02; CES energy: 4.69e-02



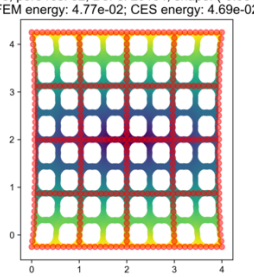
Min cell res: 2; pore res: 8; DoFs: 4218; shape: (-0.0614, -0.0228);
FEM energy: 6.00e-02; CES energy: 4.69e-02



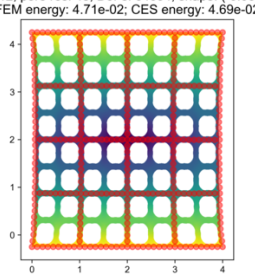
Min cell res: 4; pore res: 16; DoFs: 9474; shape: (-0.0614, -0.0228);
FEM energy: 4.97e-02; CES energy: 4.69e-02



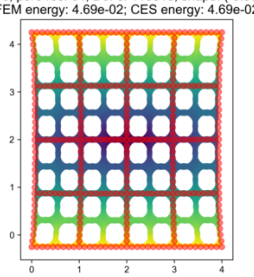
Min cell res: 8; pore res: 32; DoFs: 29164; shape: (-0.0614, -0.0228);
FEM energy: 4.77e-02; CES energy: 4.69e-02



Min cell res: 12; pore res: 48; DoFs: 64634; shape: (-0.0614, -0.0228);
FEM energy: 4.71e-02; CES energy: 4.69e-02



Min cell res: 16; pore res: 64; DoFs: 113818; shape: (-0.0614, -0.0228)
FEM energy: 4.69e-02; CES energy: 4.69e-02



Appendix C

Appendix for Chapter 3

C.1 Algorithm pseudocode

Algorithm 1: Optimization with randomized telescopes

Input: initial parameter θ, gradient routine $g(\theta, i)$ which returns $\bar{G}_i(\theta)$, compute costs $\bar{C}$, exponential decay α, tuning frequency K, horizon $\bar{H}$, reference learning rate $\bar{\eta}$

Initialize $B = 0$, next_tune= 0, $D_{i,j} = 0$

repeat

  if next_tune<= B **then**

    $\bar{D}, q, W, S \leftarrow \text{tune}(\theta, \bar{D}, g, \bar{C}, \alpha, \bar{H})$

    expectedCompute, expectedSquaredNorm = compute_and_variance$(\bar{D}, \bar{C}, S)$

    $\eta \leftarrow \bar{\eta} \frac{\text{expectedSquaredNorm}}{\bar{D}_{0,\bar{H}}}$

    $B+ = \sum_{i=1}^{\bar{H}} \bar{C}(\bar{H})$

    next_tune $+ = \bar{C}(\bar{H})$

  end if

  $N \sim q$

  for $n = 1$ **to** N **do**

    $G_n \leftarrow g(\theta, S[n])$

  end for

  $\hat{G} \leftarrow \sum_{n=1}^{N} G_n W(n, N)$

  $\theta \leftarrow \theta - \eta \hat{G}$

  if compute reused **then**

    $B+ = \bar{C}(S[N])$

  else

    $B+ = \sum_{n=1}^{N} \bar{C}(S[n])$

  end if

until converged

Algorithm 2: tune

Input: current parameter θ, current squared distance estimates $\bar{D}_{i,j}$, gradient routine $g(\theta, i)$ which returns $\bar{G}_i(\theta)$, compute costs $\bar{C}$, exponential decay α, horizon $\bar{H}$

$\bar{G}_0(\theta) \leftarrow 0$

for $i = 1$ **to** $\bar{H}$ **do**

  $\bar{G}_i(\theta) \leftarrow g(\theta, i)$

end for

for $i = 0$ **to** $\bar{H}$ **do**

  for $j = 1$ **to** $\bar{H}$ **do**

    $D_{i,j} \leftarrow ||G_i - G_j||_2^2$

  end for

end for

$\bar{D} \leftarrow \alpha\bar{D} + (1 - \alpha)D$

$S \leftarrow \text{greedy\_subsequence\_select}(\bar{D}, \bar{C})$

$q, W \leftarrow q\_\text{and}\_W(\bar{D}, \bar{C}, S)$

Return: updated estimates $\bar{D}_{i,j}$, sampling distribution q, weight function W, and subsequence S

Algorithm 3: greedy_subsequence_select

Input: Norm estimates $\bar{D}$, compute costs $\bar{C}$

Initialize $N = \text{len}(C)$

Initialize $S^+ = [N]$, $S^- = [1, ..., N]$, converged=FALSE, bestAddCost=cost$(\bar{D}, S^+, \bar{C})$,

bestRemoveCost=cost$(\bar{D}, S^-, \bar{C})$

while not converged **do**

  for $i \in [i$ for $i \in [1...N]$ if not $i \in S^+]$ **do**

    trial$S$ ←sort$(S^+ + [i])$

    trialCost←cost$(\bar{D}, \bar{C}$,trial$S)$

    if trialCost $<$ bestAddCost **then**

      S^+ ←trial$S$

      bestAddCost←trialCost

      converged ← False

      BREAK

    else

      converged ← True

    end if

  end for

end while

converged ← False

while not converged **do**

  for $i \in [i$ for $i \in S^-$ if$i \neq N$ **do**

    trial$S \leftarrow [j$ for $j \in S^-$if$j! = i]$

    trialCost←sequence_cost$(\bar{D}, C, trial\text{S})$

    if trialCost $<$ bestRemoveCost **then**

      S^- ←trial$S$

      bestRemoveCost←trialCost

      converged ← False

      BREAK

    else

      converged ← True

    end if

  end for

end while

if bestRemoveCost> bestAddCost **then**

Algorithm 4: compute_and_variance

Input: Norm estimates $\bar{D}$, compute costs $\bar{C}$, sequence S

q, $W \leftarrow q\_\text{and}\_W(\bar{D}, \bar{C}, S)$

expectedCompute $\leftarrow \sum_{i \in [1 \ldots |S|]} q(S[i]]) \bar{C}(S[i])$

if RT-SS **then**

expectedSquaredNorm $\leftarrow \sum_{i \in [1 \ldots |S|]} q(S[i]]) W(S[i], S[i]) \bar{D}_{S[i-1], S[i]}$

else if RT-RR **then**

expectedSquaredNorm $\leftarrow \sum_{i \in [1 \ldots |S|]} \sum_{j \in [1 \ldots i]} q(S[i]]) W(S[j], S[i]) \bar{D}_{S[j], S[i]}$

else

Undefined: must specify RT-SS or RT-RR

end if

Return: expectedCompute, expectedSquaredNorm

Algorithm 5: sequence_cost

Input: Norm estimates $\bar{D}$, compute costs $\bar{C}$, sequence S

expectedCompute, expectedSquaredNorm = compute_and_variance($\bar{D}, \bar{C}, S$)

Return: expectedCompute * expectedSquaredNorm

Algorithm 6: q_and_W

Input: $\bar{D}$, $\bar{C}$, and S

if RT-SS **then**

$q(N) \leftarrow \sqrt{\frac{\bar{D}_{S[N], S[N-1]}}{\bar{C}(S[n])}}$

$W(n, N) \leftarrow \frac{1}{q(N)} \mathbb{1}\{n = N\}$

else if RT-RR **then**

$\tilde{Q}(N) \leftarrow \sqrt{\frac{\bar{D}_{S[N], S[N-1]}}{\bar{C}(S[n]) - \bar{C}(S[n-1])}}$

$\tilde{(q)}(N) \leftarrow \max(0, \tilde{Q}(N) - \tilde{Q}(N-1))$

$q(N) \leftarrow \frac{\tilde{q}(N)}{\sum_i \tilde{q}(i)}$

$W(n, N) \leftarrow \frac{1}{1 - \sum_i q(i)} \mathbb{1}\{n \leq N\}$

else

Undefined: must specify RT-SS or RT-RR

end if

Return: q, W

C.2 Proofs

C.2.1 Proofs for section 2

Proposition 3.2.1

Unbiasedness of RT estimators. The RT estimators in (3.2) are unbiased estimators of Y_H as long as

$$\mathbb{E}_{N\sim q}[W(n,N)\mathbb{1}\{N \geq n\}] = \sum_{N=n}^{H} W(n,N)q(N) = 1 \quad \forall n\,.$$

Proof. A randomized telescope estimator which satisfies the above linear constraint condition has expectation:

$$\begin{aligned}
\mathbb{E}[\hat{Y}_H] &= \sum_{N=1}^{H} q(N) \sum_{n=1}^{N} W(n,N)\Delta_n \\
&= \sum_{n=1}^{H}\sum_{N=1}^{H} \Delta_n W(n,N)q(N)\mathbb{1}\{n \leq N\} \\
&= \sum_{n=1}^{H} \Delta_n \sum_{N=n}^{H} W(n,N)q(N) = \sum_{n=1}^{H} \Delta_n = Y_H
\end{aligned}$$

□

C.2.2 Proofs for section 4

Theorem 3.4.1

Bounded variance and compute with polynomial convergence of ψ. Assume ψ converges according to $\psi_n \leq \frac{c}{(n)^p}$ or faster, for constants $p > 0$ and $c > 0$. Choose the RT-SS estimator with $q(n) \propto 1/((n)^{p+1/2})$. The resulting estimator $\hat{G}$ achieves expected compute $C \leq (\mathcal{H}_H^{p-\frac{1}{2}})^2$, where $\mathcal{H}_H^i$ is the Hth generalized harmonic number of order i, and expected squared norm $\mathbb{E}[||\hat{G}||_2^2] \leq c_\psi^2(\mathcal{H}_H^{p-\frac{1}{2}})^2 := \tilde{G}^2$. The limit $\lim_{H\to\infty} \mathcal{H}_H^{p-\frac{1}{2}}$ is finite iff $p > \frac{3}{2}$, in which case it is given by the Riemannian zeta function, $\lim_{H\to\infty} \mathcal{H}_H^{p-\frac{1}{2}} = \zeta(p-\frac{1}{2})$. Accordingly, the estimator achieves horizon-agnostic variance and expected compute bounds iff $p > \frac{3}{2}$.

Proof. Begin by noting the RT-SS estimator returns $\frac{\Delta_n}{q_n}$ with probability $q(n)$. Let $\bar{q}(n) = \frac{1}{n^{p+\frac{1}{2}}}$ and $\sum_{n=1}^{H} \bar{q}(n) = Z$, such that $q(n) = \frac{\bar{q}(n)}{Z}$. First, note $Z = \sum_{n=1}^{H} \frac{1}{n^{p+\frac{1}{2}}} = \mathrm{H}_H^{p+\frac{1}{2}}$. Now inspect the

expected squared norm $\mathbb{E}||\hat{G}||_2^2$:

$$\begin{aligned}
\sum_{n=1}^{H} q(n)||\frac{\Delta_n}{q_n}||_2^2 &= \sum_{n=1}^{H} q(n)\frac{||\Delta_n||_2^2}{q_n^2} \\
&= Z\sum_{n=1}^{H} \bar{q}(n)\frac{||\Delta_n||_2^2}{\bar{q}_n^2} \\
&\leq Zc_\psi^2 \sum_{n=1}^{H} \bar{q}(n)\frac{n^{2p+1}}{n^{2p}} \\
&= Zc_\psi^2 \sum_{n=1}^{H} \frac{n^{2p+1}}{n^{3p+\frac{1}{2}}} \\
&= Zc_\psi^2 \sum_{n=1}^{H} \frac{1}{n^{p-\frac{1}{2}}} \\
&= Zc_\psi^2 \mathrm{H}_H^{p-\frac{1}{2}} \\
&= c_\psi^2 \mathrm{H}_H^{p-\frac{1}{2}} \mathrm{H}_H^{p+\frac{1}{2}} \\
&\leq c_\psi^2 (\mathrm{H}_H^{p-\frac{1}{2}})^2
\end{aligned}$$

Now inspect the expected compute, $\mathbb{E}_{n\sim q} n$:

$$\begin{aligned}
\mathbb{E}_{n\sim q} &= \sum_{n=1}^{N} q(n)n \\
&= Z\sum_{n=1}^{H} \frac{n}{n^{p+\frac{1}{2}}} \\
&= Z\sum_{n=1}^{H} \frac{1}{n^{p-\frac{1}{2}}} \\
&= Z\mathrm{H}_H^{p-\frac{1}{2}} \\
&= \mathrm{H}_H^{p-\frac{1}{2}} \mathrm{H}_H^{p+\frac{1}{2}} \\
&\leq (\mathrm{H}_H^{p-\frac{1}{2}})^2
\end{aligned}$$

□

Theorem 3.4.2

Bounded variance and compute with geometric convergence of ψ. Assume ψ_n converges according to $\psi_n \leq cp^n$, or faster, for $0 < p < 1$. Choose RT-SS and with $q(n) \propto p^n$. The resulting estimator $\hat{G}$ achieves expected compute $C \leq (1-p)^{-2}$ and expected squared norm $||\hat{G}||_2^2 \leq \frac{c}{(1-p)^2} := \tilde{G}^2$. Thus, the estimator achieves horizon-agnostic variance and expected compute bounds for all $0 < p < 1$.

Proof. Let $q(n) = \frac{\bar{q}(n)}{Z}$, for $\bar{q}(n) = p^n$. Note $Z = \sum_{n=1}^{H} p^n = p\frac{1-p^H}{1-p} \leq \frac{1}{1-p}$. Now, note $\psi_n = c_\psi \bar{q}(n)$. It follows

$$\begin{aligned}
\mathbb{E}_{n\sim q}||\frac{\Delta_n}{q(n)}||_2^2 &= \sum_{n=1}^{H} q(n)\frac{||\Delta_n||_2^2}{q(n)^2} \\
&\leq \sum_{n=1}^{H} q(n)\frac{\psi_n^2}{q(n)^2} \\
&=\leq c_\psi^2 \sum_{n=1}^{H} q(n)\frac{\bar{q}(n)^2}{q(n)^2} \\
&= c_\psi^2 Z^2 \sum_{n=1}^{H} q(n) \\
&= c_\psi^2 Z^2
\end{aligned}$$

Now consider the expected compute. We have

$$\begin{aligned}
\mathbb{E}_{n\sim q} n &= \sum_{n=1}^{N} nq(n) \\
&= \sum_{n=1}^{N} \frac{np^n}{Z} \\
&= \frac{1}{Z}\sum_{n=1}^{N} np^n \\
&= p\frac{1}{Z}\frac{1 + Hp^{H+1} - (H+1)p^H}{(1-p)^2} \\
&= \frac{1 + Hp^{H+1} - (H+1)p^H}{(1-p)(1-p^H)} \\
&\leq \frac{1}{(1-p)(1-p^H)} \\
&\leq \frac{1}{(1-p)^2}
\end{aligned}$$

□

Theorem 3.4.3

Asymptotic regret bounds for optimizing infinite-horizon programs. Assume the setting from 3.4.1 or 3.4.2, and the corresponding C and $\tilde{G}$ from those theorems. Let R_t be the instantaneous regret at the tth step of optimization, $R_t = \mathcal{L}(\theta_t) - \min_\theta \mathcal{L}(\theta)$. Let $t(B)$ be the greatest t such that a computational budget B is not exceeded. Use online gradient descent with step size $\eta_t = \frac{D}{\sqrt{t\mathbb{E}[||\hat{G}||_2^2]}}$. As $B \to \infty$, the asymptotic instantaneous regret is bounded by $R_{t(B)} \leq \mathcal{O}(\tilde{G}D\sqrt{\frac{C}{B}})$, independent

of H.

Proof. First, we control $t(B)$ using the central limit theorem. Note $t \to \infty \iff B(t) \to \infty$. Consider B as a function $B(t)$ of t. We have $B(t) = \sum_{\tau=1}^{t} N_t$, where $N \sim q$. Thus, $\frac{B(t)}{t} \to \mathbb{E}_{N\sim q} N$ by the central limit theorem. This implies that in the limit, $t = \frac{B}{C}$.

To complete the proof, plug in $t(B)$ and η_t, as well as the upper bound on squared norm $\mathbb{E}||\hat{G}||_2^2 \leq \tilde{G}^2$ and upper bound on diameter D, into standard results for stochastic gradient descent with convex loss functions (e.g. section 3.4 in Hazan et al. (2016)) □

C.2.3 Proofs for section 5

Theorem 3.5.1

Optimality of RT-SS under adversarial correlation. Consider the family of estimators presented in Equation 3.2. Assume θ, ∇_θ, and G are univariate. For any fixed sampling distribution q, the single-sample RT estimator RT-SS minimizes the worst-case variance of $\hat{G}$ across an adversarial choice of covariances $\text{Cov}(\Delta_i, \Delta_j) \leq \sqrt{\text{Var}(\Delta_i)}\sqrt{\text{Var}(\Delta_j)}$.

Proof. Recall $\hat{G} = \sum_{n=0}^{N} \Delta_n W(n, N)$. Let $\sigma_{i,j}^2 = \text{Cov}(\Delta_i, \Delta_j)$ and $\sigma_i^2 = \text{Var}(\Delta_i)$. The variance of $\hat{G}$ is:

$$\begin{aligned}
\text{Var}(\hat{G}) &= \sum_N q(N)\Big[\sum_{i=0}^{N}\sum_{j=0}^{N} W(i,N)W(j,N)\sigma_{i,j}^2\Big] \\
&\leq \sum_N q(N)\Big[\sum_{i=0}^{N}\sum_{j=0}^{N} W(i,N)W(j,N)\sigma_i\sigma_j\Big] \\
&= \sum_N q(N)\Big(\sum_{n=0}^{N} W(n,N)\sigma_n\Big)^2
\end{aligned}$$

Note the above bound is tight as the adversary can choose $\text{Cov}(\Delta_i, \Delta_j) = \sigma_i\sigma_j$. Introduce $\rho(n, N) = W(n, N)q(N)$, and note that the constraint from proposition 3.2.1 can equivalently be stated as $\sum_{N\geq n} \rho(n, N) = 1 \forall n$. We have the variance:

$$\text{Var}(\hat{G}|N) \leq \sum_N \frac{1}{q(N)}\Big(\sum_{n=0}^{N} \rho(n,N)\sigma_n\Big)^2$$

Consider finding $\rho(n, N)$ which minimizes the variance for an arbitrary q. The constrained optimiza-

tion has the Lagrangian:

$$J = \Big(\sum_N \frac{1}{q(N)} (\sum_{n=0}^{N} \rho(n,N)\sigma_n)^2 \Big) + \sum_n \lambda_n (\sum_{N \geq n} \rho(n,N) - 1)$$

We can accordingly optimize by taking derivatives:

$$\begin{aligned}
\frac{dJ}{d\rho(n,N)} &= 2Cq(N)(\sum_{i=0}^{N} w(i,N)\sigma_i)\sigma_n + \lambda_n \\
\frac{dJ}{d\rho(n,N)} = 0 &\implies \sigma_n q(N) \sum_{i=0}^{N} w(i,N)\sigma_i = k_n \\
&\implies \sigma_n \sum_{i=0}^{N} \rho(i,N)\sigma_i = k_n \forall N \geq n \\
&\implies \rho(n,N) = 0 \forall N > n
\end{aligned}$$

□

Theorem 3.5.2

Optimal q under adversarial correlation. Consider the family of estimators presented in Equation 3.2. Assume $\text{Cov}(\Delta_i, \Delta_i)$ and $\text{Cov}(\Delta_i, \Delta_j)$ are diagonal. The RT-SS estimator with $q_n \propto \sqrt{\frac{\mathbb{E}[||\Delta_n||_2^2}{C(n)}}$ maximizes the ROE across an adversarial choice of diagonal covariance matrices $\text{Cov}(\Delta_i, \Delta_j)_{kk} \leq \sqrt{\text{Cov}(\Delta_i, \Delta_i)_{kk}\text{Cov}($

Proof. First, note that by the assumption of diagonal covariance between all terms, the expected squared norm decomposes over indices k:

$$\mathbb{E}||\hat{G}||_2^2 = \sum_k \mathbb{E}\hat{G}[k]^2$$

For all choices of q, the RT-SS estimator minimizes the worst-case variance and thus (due to unbiasedness) the expected squared value of each entry in $\hat{G}$. Because the squared norm decomposes, the RT-SS estimator minimizes the squared norm for all q.

It remains to optimize q. We know $\rho(n,N) = 0 \forall N > n$. Therefore to satisfy the constraint, we have $\rho(N,N) = 1$. It follows that:

$$\text{ROE}^{-}1 = \big(\sum_N q(N)C(N) \big) \big(\sum_N \frac{\mathbb{E}||\Delta_N||_2^2}{q(N)} \big)$$

We require $\sum_N q(N) = 1$. The constrained optimization has the Lagrangian:

$$J = \Big(\sum_N q(N)C(N)\Big)\Big(\sum_N \frac{\mathbb{E}||\Delta_N||_2^2}{q(N)}\Big) + \lambda(\sum_N q(N) - 1)$$

Let $C = \Big(\sum_N q(N)C(N)\Big)$ and $V = \Big(\sum_N \frac{\mathbb{E}||\Delta_N||_2^2}{q(N)}\Big)$. We optimize $q(N)$ by taking the derivative of the inverse ROE:

$$\begin{aligned}
\frac{d\text{ROE}^{-1}}{dq(N)} &= C(N)V - C\frac{\sigma_N^2}{q(N)^2} \\
\frac{d\text{ROE}^{-1}}{dq(N)} = 0 &\implies q(N)^2 \propto \frac{\mathbb{E}||\Delta_N||_2^2 C}{C(N)V} \\
&\implies q(N) \propto \sqrt{\frac{\mathbb{E}||\Delta_N||_2^2}{C(N)}}
\end{aligned}$$

□

Theorem 3.5.3

Optimality of RT-RR under independence. Consider the family of estimators presented in Eq. 3.2. Assume the Δ_j are univariate. When the Δ_j are uncorrelated, for any importance sampling distribution q, the Russian roulette estimator achieves the minimum variance in this family and thus maximizes the optimization efficiency lower bound.

Proof. By independence, we have $\mathbb{E}\big(\sum_n W(n,N)\Delta_n\big)^2 = \sum_n W(n,N)^2\mathbb{E}\Delta_n^2$. It follows that an RT estimator has variance:

$$\begin{aligned}
\text{Var}(\hat{G}) &= \sum_N q(N) \sum_{n\le N} W(n,N)^2\mathbb{E}\Delta_n^2 \\
&= \sum_N \frac{1}{q(N)} \sum_{n\le N} \rho(n,N)^2\mathbb{E}\Delta_n^2
\end{aligned}$$

Recall the constraint in proposition 3.2.1 requires $\sum_{N\ge n}\rho(n,N) = 1$ for all n. The Lagrangian of the constrained minimization of $\text{Var}(\hat{G})$ with respect to ρ is:

$$J = \text{Var}(\hat{G}) + \sum_n \lambda_n(\sum_{N\ge n}\rho_n - 1)$$

We optimize ρ by finding the minimum of the Lagrangian:

$$\begin{aligned}
\frac{dJ}{d\rho(n,N)} &= \frac{2}{q(N)}\rho(n,N)\mathbb{E}\Delta_n^2 + \lambda_n \\
\frac{dJ}{d\rho(n,N)} = 0 &\implies \frac{\rho(n,N)}{q(N)} = -\frac{\lambda_n}{2\mathbb{E}\Delta_n^2} \\
&\implies W(n,N) = -\frac{\lambda_n}{2\mathbb{E}\Delta_n^2}, \text{ which is independent of } N \\
&\implies W(n,N) = \frac{1}{\sum_{N' \geq n} q(N')} \text{ to fulfill the constraint in proposition 3.2.1}
\end{aligned}$$

□

Theorem 3.5.4

Optimal q under independence. Consider the family of estimators presented in Equation 3.2. Assume $\text{Cov}(\Delta_i, \Delta_i)$ is diagonal and Δ_i and Δ_j are independent. The RT-RR estimator with $Q(i) \propto \sqrt{\frac{\mathbb{E}[||\Delta_i||_2^2}{C(i)-C(i-1)}]}$, where $Q(i) = \Pr(n \geq i) = \sum_{j=i}^{H} q(j)$, maximizes the ROE.

Proof. First note that by theorem 3.5.3, for any q and for each element in the vector $\hat{G}$, the RT-RR estimator minimizes the variance of that element. Now note that due to independence of Δ_i, Δ_j and diagonality of $\text{Cov}(\Delta_i, \Delta_i)$:

$$\begin{aligned}
\mathbb{E}||\sum_{n=1}^{N} W(n,N)\Delta_n||_2^2 &= \sum_{n=1}^{N} W(n,N)\mathbb{E}||\Delta_n||_2^2 \\
&= \sum_k \sum_{n=1}^{N} W(n,N)\mathbb{E}\Delta_n[k]^2 \qquad\qquad = \sum_k \mathbb{E}\hat{G}[k]^2
\end{aligned}$$

As the RT-RR estimator minimizes $\mathbb{E}\hat{G}[k]^2$ for each coordinate k, it also minimizes $\mathbb{E}||\hat{G}||_2^2$. It remains to optimize Q. Consider the inverse ROE of the RT-RR estimator. By independence we have:

$$\text{ROE}(\hat{G})^{-1} = \mathbb{E}||\hat{G}||_2^2\mathbb{E}C = \Big(\sum_N q(N) \sum_{n \leq N} \frac{1}{Q(n)^2}\mathbb{E}||\Delta_n||_2^2\Big)\Big(\sum_N q(N)C(N)\Big)$$

Take the gradient of the inverse optimization efficiency lower bound w.r.t. $q(n)$:

$$\frac{d\text{ROE}(\hat{G})^{-1}}{dq(N)} = C(N)\mathbb{E}||\hat{G}||_2^2 + \sum_{n \leq N} \frac{1}{Q(n)^2}\mathbb{E}||\Delta_n||_2^2 - \sum_i q(i) \sum_{j \leq min(i,N)} \frac{2}{Q(j)^3}\mathbb{E}||\Delta_j||_2^2$$

$$\sum_i q(i) \sum_{j \le min(i,N)} \frac{2}{Q(j)^3} \mathbb{E}||\Delta_j||_2^2 = \sum_{j \le N} \frac{2}{Q(j)^2} \mathbb{E}||\Delta_j||_2^2 \frac{\sum_i q(i) \mathbb{1}\{i \ge j\}}{Q(j)}$$
$$= \sum_{j \le N} \frac{2}{Q(j)^2} \mathbb{E}||\Delta_j||_2^2 \quad \text{by definition of } Q(j)$$

$$\implies \frac{d\text{ROE}(\hat{G})^{-1}}{dq(N)} = C(N)\mathbb{E}||\hat{G}||_2^2 - \sum_{n \le N} \frac{1}{Q(n)^2} \mathbb{E}||\Delta_n||_2^2$$

Now optimize the objective w.r.t. Q by finding the critical point:

$$\frac{d\text{ROE}(\hat{G})^{-1}}{dq(N)} = 0 \implies C(N)\mathbb{E}||\hat{G}||_2^2 = \sum_{n \le N} \frac{1}{Q(n)^2} \mathbb{E}||\Delta_n||_2^2$$
$$\implies \mathbb{E}||\hat{G}||_2^2 \Big(C(N) - C(N-1) \Big) = \frac{1}{2} \frac{\mathbb{E}||\Delta_N||_2^2}{Q(N)^2}$$
$$\implies Q(N)^2 \propto \frac{\mathbb{E}||\Delta_n||_2^2}{C(N) - C(N-1)}$$

□