We need to do better in assessing AI/ML methods for the solution of differential equations

Wolfgang Bangerth (Colorado State University)
Matthew G. Knepley (University at Buffalo)
Kyle T. Mandli (NSF NCAR)
Richard Tran Mills (Argonne National Laboratory)

Introduction

The numerical analysis community has spent decades investigating and developing methods to compute approximate solutions of partial differential equations (PDEs), and so it is appropriate that there is now also a substantial body of literature that investigates the use of neural networks (NNs) for this purpose. Read as a whole, the literature on NN-for-PDE is remarkably bullish: papers routinely report speedups of factors of 10, 100, or more over traditional tools such as the finite element, finite difference, or finite volume methods. Yet, NN-for-PDE methods are not used in practice, and when talking to colleagues who have tried them over coffee or in the hallways of conferences, they typically report that they do not work. This dissonance between reported evidence and lived experience is confusing to many of us.

In a recent paper published in Nature Machine Intelligence, "Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations", Nick McGreivy and Ammar Hakim consider a large sample of papers that solve fluid flow equations with neural networks. Their conclusion, provided in remarkably even language, is that 79% (60 of 76) of the articles in their sample claim to outperform a standard numerical method compared against what they classify as a 'weak baseline'; moreover, they find that reporting biases are widespread.

Numerical analysis has a long history of – and well-established tools for – assessing methods fairly. Nobody benefits in the long-term if a sub-community reports on the methods they develop in ways that do not allow for objective comparisons. As a community, we need to do better than we have over the past decade in assessing these methods, as authors, reviewers, editors, supervisors, and colleagues.

In the following, let us raise two points that we think authors should consider for papers that demonstrate the use of any kind of method (including neural networks) for the solution of partial differential equations, and that we think reviewers and editors should require as a condition for publication.

Show work-accuracy diagrams

Traditionally, the community developing numerical solvers for (ordinary or partial) differential equations has evaluated its methods using "work-accuracy diagrams": Plots in which the

horizontal axis represents a proxy for the computational work necessary to solve a problem, and the vertical axis represents the accuracy or error in the computed solution. The following plots show three widely-used ways of displaying this information.

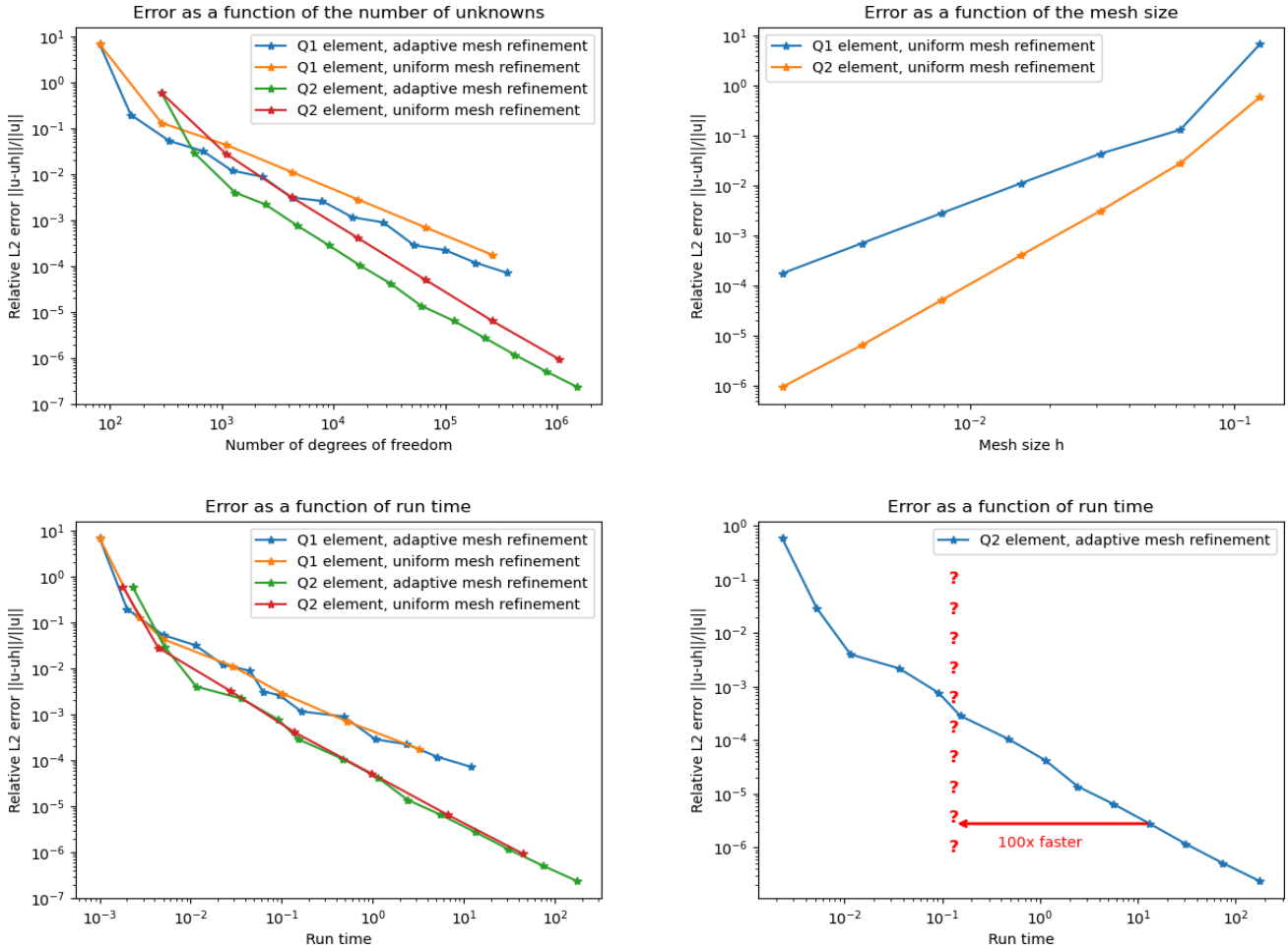


Figure 1: *Three forms of work-accuracy diagrams, taken from the documentation of the step-7 tutorial of the deal.II finite element library. In all cases, accuracy is represented by the error between exact and computed solution. In the top left panel, work is represented by the number N of degrees of freedom. With a good linear solver, the compute time is typically a linear multiple of N. In the top right panel, work is represented via the mesh size h of the mesh on which the solution is computed. On a uniform mesh in d dimensions, the total number of degrees of freedom is given by* $N \propto (1/h)^d$*, and so practitioners understand h as inversely related to work. In these panels, "work" is therefore represented in a way that relates to the way we solve the problem, but not the implementation on a concrete machine. In contrast, in the bottom left panel, work is represented via the CPU time necessary to solve the problem with a concrete implementation on concrete hardware. Bottom right: For a method (in red) that on a given mesh (a data point of the blue curve) is 100x faster than a reference method, but for which we don't provide the error, can be better or worse than the reference.*

Work-accuracy diagrams serve two purposes. First, they illustrate the trade-off between accuracy and work: If one accepts a higher computational cost, one can typically obtain better

accuracy in the output. The plots also allow assessing what accuracy one can expect with a given investment of resources, or conversely what resources one has to put up to achieve a desired accuracy. If several methods are shown on the same plot, this would allow one to determine the Pareto front for the given PDE. Second, work-accuracy diagrams can demonstrate *convergence* of a method if increasing the computational cost consistently reduces the error.

Nearly every paper on numerical PDE solvers published over the past fifty or more years has work-accuracy diagrams. Those that do not may show tables with corresponding information, or at least derive estimates of the form $||u-u_h|| \leq Ch^2 ||\nabla^2 u||$ in which the left hand side is the error between computed and exact solutions and the right hand side, to a practitioner, represents the computational effort because the mesh size h is inversely proportional to the number of mesh points per coordinate direction. For example, in 2d, when using a uniform MxM mesh with $N=M^2$ total points, we have that $h \propto 1/M = 1/\mathrm{sqrt}(N)$ and consequently the error estimate is equivalent to $||u-u_h|| \leq (C ||\nabla^2 u||)/N$ where now N is considered a measure of work.

Given the ubiquity of work-accuracy diagrams in the numerical methods literature over such a long time, it is surprising that they are conspicuously absent from much of the literature that propose neural networks to solve PDEs. Instead, these papers focus by and large on the speed of computing an approximate solution on a given mesh, with statements such as “On a 100x100 mesh, our neural operator solves the problem 80x faster than the finite-element based implementation”. To an uninitiated reader, this sentence suggests that the same solution is computed, just much faster. In practice, however, this is of course not true: the two methods produce *different* approximations, and the statement provides no information about whether the neural-network solution is more or less accurate. Knowing that the method is substantially faster on a 100 x 100 mesh therefore does not, by itself, tell us whether it is actually better because the solution may be anywhere along the line indicated by question marks in the the bottom right panel of Figure 1 above.

To illustrate the issue more poignantly: One can also write a function that computes an approximation u_h=12.345. Because this is just a fixed number, it is very fast: The function will execute in around a nanosecond on modern hardware, vastly faster than any finite element or neural network solver. It is of course also true that this value is in almost all cases a pretty terrible approximation of the *actual* solution of the PDE we want to solve. What the example shows is that performance numbers are only useful if accompanied by statements about the accuracy.

The lack of such statements is not an isolated occurrence in the literature on NN-based PDE solvers. Instead, the paper by McGreivy and Hakim shows that some 80% of papers testing neural networks as PDE solvers have these sorts of issues. Moreover, in our reading, at best a small fraction of the papers on NN-based PDE solvers show results on more than one problem size, leaving it entirely unclear (i) whether the method converges in the sense that with sufficiently more work, any desired error level can be reached, and (ii) for which problem sizes, if any, the proposed method may be better than existing methods. This omission also serves to obscure the setup cost incurred for a mesh of a given size.

This failure of an entire community to provide *interpretable* results about how well these methods do – even though the numerical methods community has a decades-old standard

approach to doing so – lies at the core of the disconnect between the self-reported usefulness of NN-based PDE solvers and the broader community's experience expressed over cups of coffee and in the hallways at conferences. We need to expect better.

Compare costs fairly

Many uses of neural networks to solve PDEs are motivated by applications in which one needs to solve the PDE many times with different right hand sides (or other pieces of input data). As a consequence, what matters is the cost of solving the PDE once (the "online phase"), whereas the (often very large) cost of generating thousands or millions of pieces of training data using a traditional PDE solver, and then actually training the neural network (the "offline phase"), can be neglected: It is implicitly assumed that the entire cost of the offline (or "training") phase becomes insignificant when amortized over the lifetime of the online phase.

Whether this assumption actually corresponds to interesting applications other than the solution of linear time-dependent problems may be debatable. But let us assume that it does. Then a fair comparison with a traditional solver must give the latter the same opportunity to amortize an offline investment.

In such a case, practitioners would typically allow themselves to build expensive preconditioners for their finite element or finite difference solvers. In other words, they would not use a Conjugate Gradient solver with a Jacobi preconditioner as the baseline. Instead, they would perhaps compute a sparse direct decomposition of the system matrix, after which the solution of the PDE comes down to one forward and one backward solve with triangular matrices. If the matrix is too large to compute and store a sparse decomposition, one might use stochastic techniques to project the operator onto a subspace of a few hundred or thousand dimensions, and compute its inverse via a truncated SVD; used as a preconditioner, one should then be able to solve the PDE within at most a handful of CG iterations – and be vastly faster than a simple CG solver with a simplistic preconditioner. In fact, finite element basis functions can be tailored to a specific problem by precomputation, much as neural networks are trained, resulting in vastly more accurate solutions for the same forward cost. The whole field of reduced-order models (ROM) is based on these sorts of premises and has a long history of reporting costs.

The point is that a *fair* comparison of a neural network with traditional solvers needs to allow the latter to make use of the fundamental assumptions of where the proposed methods might be applicable. When pressed, the authors of papers sometimes state that they are not experts in finite-element or finite-difference solvers and consequently lack the knowledge to come up with the relevant techniques. This is not a sufficient reason to compare a proposed method with a sub-par traditional solver, in particular because the relevant knowledge is widely available in the CS&E community, and this excuse would not be accepted in related fields such as computer vision.

Summary

Numerical analysts develop methods because they want to make the solution of concrete problems possible and efficient. But we only advance our field if we describe these methods in

objective and fair ways, and as the work by McGreivy and Hakim shows, a substantial fraction of the literature on the use of neural networks for the solution of partial differential equations has not done so. Their conclusions also match the lived experience of many colleagues who have tried NNs to solve concrete applications for which they need numerical solutions. As a community – as authors, reviewers, editors, colleagues – we need to do better at evaluating these methods: We need to write papers that more objectively assess solvers based on neural networks; as reviewers and editors, we need to insist that authors report enough information to allow *meaningful* comparison.