Accelerating GMRES with Deep Learning in Real-Time

Kevin Luna
kevinluna@math.arizona.edu
University of Arizona

Johannes Blaschke
jpblaschke@lbl.gov
Lawrence Berkeley National Laboratory

ABSTRACT

Deep learning has demonstrated the ability to accelerate scientific simulation codes by learning to replace computationally expensive steps. While deep learning methods have been highly optimized for situations where data is relatively inexpensive to produce in mass quantities, here data is expensive to produce (e.g. by basing the ground on high fidelity solutions). We turn to online learning as a solution: by training in real time – i.e. while a simulation is running – we developed a versatile method of accelerating simulations, while using the smallest data set necessary to train the model. GMRES is a popular solver for the Laplace equation (a common PDE in physical systems). We demonstrate that our method can efficiently train convolutional neural networks to produce tuned preconditioners for GMRES solvers, that provide a roughly 2× acceleration.

1 INTRODUCTION

Partial Differential Equation (PDE) based simulations are expensive in terms of computational resources. On the other hand, machine learning algorithms – and in particular, deep learning algorithms – need to ingest very large data sets in order to produce accurate models. This leads to the situation, that efforts aimed at using machine learning to decrease the computation cost of PDE-solvers will require an enormous up-front investment in the form of large training sets. To alleviate this cost, recent efforts such as in [2] and [4] have proposed deep learning approaches to accelerate iterative linear solvers through the construction of learned preconditoners. In our work we set out to take this idea further by accelerating an iterative linear solver during a simulation in real-time– that is: while a simulation is running – with no human interaction.

While Neural networks are effective at learning, there are some well known challenges when only partially representative data is available for training [3]. Consequently, working with neural networks in real-time with limited data ("online") is challenging. Chief among these issues is so-called catastrophic forgetting where new information interferes with previously learned knowledge of the network. Mindful of these issues with deep learning in an online context, we constructed our approach to deal with these challenges by carefully curating the training set as it grows in real-time so that the neural network is able to improve performance before a representative distribution of data is attained.

2 METHODOLOGY AND ALGORITHMS

To illustrate our approach, consider a sequence of linear problems of the form $Ax_i = b_i$ being solved over the course of a simulation, e.g. once per time step, giving us a "time series"

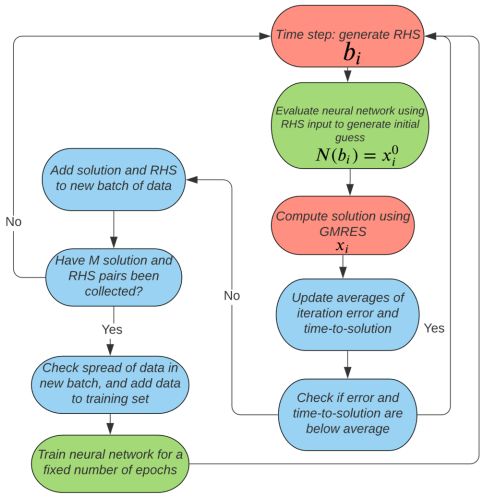

Figure 1: Experiment for accelerating GMRES with neural networks in real-time. Red boxes indicate traditional computational steps, blue boxes data collection, and green boxes indicate deep learning.

of length T. We refer to this sequence as our "simulation". The goal here is to train a neural network $N(b)$ in real-time as linear problems are solved by restarted GMRES. This is accomplished by having the neural network provide an initial guess $N(b_i) = x_i^0$ to the GMRES solver. The training objective is that this initial guess then improves the rate of convergence of the GMRES iterations.

The data set used to train the model consists of RHS-solution pairs $\{(b_i, x_i)\}$. However, unlike traditional deep learning approaches, the goal here is to train the network in real-time while data is being generated from the simulation. This naturally leads to an online supervised learning problem since at a given time $t < T$, we only have access to $N(t)$ number of (b_i, x_i) pairs. In the formulation presented here, at time $t < T$, the training set is $X_t = \{(b_i, x_i)\}_1^{N(t)}$. In order to ensure a high-quality data set some time steps are discarded, while only "high-quality" ones are kept (discussed at the end of this section).

We train the neural network for a fixed number of epochs during the training phase using batched gradient descent where samples are randomly selected from X_t. The network is set up to be relatively shallow so that training can be performed quickly. When A is the discrete Laplace operator networks that are able to learn non-local relationships worked especially well. Good performance was obtained with 4-6 layer CNNs with sufficiently large and dilated kernels. The model is used to generate initial "guesses" for the GMRES solver; with the quality of the prediction being a crucial deciding factor on whether or not to include (x_i, b_i) in the growing

Table 1: Comparison of GMRES solver run times for different grids

	20×20	30×30	40×40
GMRES Time (s)	1735	5484	12803
MLGMRES Time (s)	847	2841	6940
Training Time (s)	155	464	1547
MLGMRES/GMRES	0.49	0.51	0.54

data set. Quality metrics such as the 2-norm of the GMRES residual at a particular iteration and rate-of-convergence for each guess provided by the neural network are tracked as the simulation runs. After M solutions have been computed these metrics are used to decide which of the $\{(b_i, x_i)\}_1^{N(t)}$ pairs are added to the training set for the next training phase (at some time $t' > t$). Only those RHS-solution pairs which increase the quality of the solution, as well as the convergence rate are retained. In practise, this means decision is made after the first few iterations. The flow chart provided in figure 1 outlines the methodology developed.

3 NUMERICAL EXPERIMENTS

Our numerical experiments were performed on a single "Cori-GPU" node at NERSC (Lawrence Berkeley National Laboratory). Each node has two 20-core Intel Skylake processors and 8 NVIDIA V100 GPUs. For the results presented here, we used PyTorch optimized for Cori-GPU's hardware.

As an important application, and highly-relevant benchmark for our approach, we apply this algorithm to the 2D discrete Poisson Problem. For our benchmark the choices of RHS consisted of localized "point-like" sources and fields with linear gradients on the domain $[-1, 1] \times [-1, 1]$. These are motivated by physical applications ranging from particles in fluids, to interacting electrostatic charges. We stress that our approach is not limited to the examples studied here. For every RHS b_i sampled, we measure the time for a direct (non-preconditioned) GMRES implementation to solve the problem and the time for the same GMRES implementation assisted by the neural network. We call the neural network assisted GMRES solver MLGMRES.

For all the RHS choices considered, the results observed were essentially the same. Hence we present data where b_i are spread-out sources that move at random. Table 1 compares the performance of GMRES with MLGMRES for increasing grid resolutions. The primary result is that the neural network provided initial iterates that sped up the time-to-solution by about a factor of two. Figure 2 (left) provides a clearer view of this acceleration by presenting the run-time for each MLGMRES and GMRES call as a function of simulation steps. It's clear that in real-time the neural network is able to produce consecutively improving initial iterates that lead to faster convergence times (lower time-to-solution).

Figure 2 (right) sheds light on how MLGMRES achieves this speedup. Here we see that the first few iterations of the ML-preconditioned GRMES algorithm do indeed converge faster than un-preconditioned GMRES. The effect of this increased convergence rate does wear off, and the algorithm

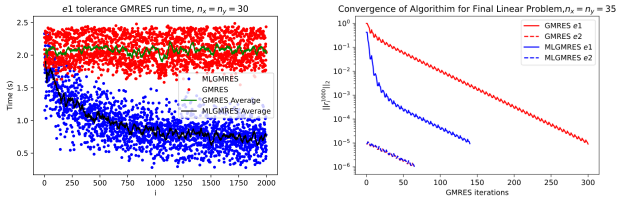

Figure 2: Left: Speed up attained by MLGMRES during the simulation. Right: Improvement in convergence rate of MLGMRES after 200 time steps. The final training set only contains roughly 400 data pairs. Dashed lines indicate restarts at a higher error tolerance e_2.

converges at the same rate as un-preconditioned GMRES. This is important for the real-time setting: even if the neural network does not yet produce solutions of sufficient quality to be useful for scientific simulations, the solver will refine the solution to the desired quality at no additional cost over the un-preconditioned solver. Furthermore, the neural network is doing more than just "tabulating solutions": the first iterate of MLGMRES has only an accuracy of about 10% – the real benefit of MLGMRES is the initial speedup of convergence rates.

4 CONCLUSION

As demonstrated in Table 1 and Figure 2, our real-time methodology to accelerate an iterative linear solver has provided promising results that show potential in future applications. Our approach is flexible in that the neural network model can be constructed from scratch and does not need to be pre-trained on large amounts of expensive simulation data to get a performance boost. The approach presented here only needs the desired iterative linear solver, and a wrapper for that solver that augments the solver calls as outlined in Figure 1 with the user's deep learning framework of choice. Beyond this, the methodology presented here requires no further input from the user as a simulation progresses.

5 ONGOING EFFORTS AND FUTURE WORK

Motivated by positive results in our prototype, work is currently underway to incorporate this methodology to accelerate FHDeX, a mulit-threaded finite volume Stokes flow solver [1] based on the AMReX [5] software suite. The results shown here are with respect to un-preconditioned restarted GMRES, by targeting future acceleration efforts on FHDeX, we will be comparing MLGMRES to preconditioned GMRES. To incorporate this deep learning methodology into our target code base, we are making use of the PyTorch C++ API to take care of the deep learning components of our approach.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy Office of Science, and the Computing Sciences Summer Student program at Berkeley Lab. This research used resources of the National Energy Research Scientific Computing Center

(NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] Mingchao Cai, Andy Nonaka, Boyce E. Griffith, and Aleksandar Donev. 2014. Efficient Variable-Coefficient Finite-Volume Stokes Solvers. Communications in Computational Physics 16 (2014), 1263–1297.

[2] Markus Götz and Hartwig Anzt. 2018. Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation. 2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA) (2018).

[3] German I. Parisi, Kemker Ronald, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual Lifelong Learning with Neural Networks:A Review. Neural Networks 113 (2019), 54–71.

[4] Johannes Sappl, Laurent Seiler, Matthias Harders, and Wolfgang Rauch. 2019. Deep Learning of Preconditioners for Conjugate Gradient Solvers in Urban Water Related Problems. eprint arXiv:1906.06925 (2019).

[5] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. 2019. AMReX: a framework for block-structured adaptive mesh refinement. Journal of Open Source Software 4, 37 (May 2019), 1370. https://doi.org/10.21105/joss.01370