SIAM J. SCI. COMPUT.
Vol. 0, No. 0, pp. S199–S225

LEARNING OPTIMAL MULTIGRID SMOOTHERS VIA NEURAL NETWORKS*

RU HUANG†, RUIPENG LI‡, AND YUANZHE XI†

Abstract. Multigrid methods are one of the most efficient techniques for solving large sparse linear systems arising from partial differential equations (PDEs) and graph Laplacians from machine learning applications. One of the key components of multigrid is smoothing, which aims at reducing high-frequency errors on each grid level. However, finding optimal smoothing algorithms is problem-dependent and can impose challenges for many problems. In this paper, we propose an efficient adaptive framework for learning optimized smoothers from operator stencils in the form of convolutional neural networks (CNNs). The CNNs are trained on small-scale problems from a given type of PDEs based on a supervised loss function derived from multigrid convergence theories and can be applied to large-scale problems of the same class of PDEs. Numerical results on anisotropic rotated Laplacian problems and variable coefficient diffusion problems demonstrate improved convergence rates and solution time compared with classical hand-crafted relaxation methods.

Key words. machine learning, multigrid methods, multigrid smoothers, convolutional neural networks

MSC codes. 65M55, 65F08, 65F10, 15A60

DOI. 10.1137/21M1430030

1. Introduction.

Partial differential equations (PDEs) play important roles in modeling various phenomena in many fields of science and engineering. Their solutions are typically computed numerically when closed-form solutions are not easily available, which leads to large-scale and ill-conditioned sparse linear systems to solve. In machine learning applications such as spectral clustering, graph-based semisupervised learning, Markov chains, and transportation network flows, solving large-scale linear systems associated with graph Laplacians is often needed [39, 14, 30]. The development of efficient linear solvers is still an active research area nowadays [36, 43, 11].

Among many numerical solution schemes, multigrid methods often show superior efficiency and scalability especially for solving elliptic-type PDE and graph Laplacian problems [6, 34, 10, 42]. Fast convergence of multigrid is achieved by exploiting hierarchical grid structures to eliminate errors of all modes by smoothing and coarse-grid correction at each grid level. Thus, the performance of multigrid methods highly depends on the smoothing property of a chosen smoother. However, the design of optimal smoothing algorithm is problem-dependent and often too complex to be achieved even by domain experts. In this paper, we propose an adaptive framework for training optimized smoothers via convolutional neural networks (CNNs), which directly learns a mapping from operator stencils to the inverses of the smoothers. The training process is guided by multigrid convergence theories for good smoothing properties

*Received by the editors June 30, 2021; accepted for publication (in revised form) April 18, 2022; published electronically August 24, 2022.
https://doi.org/10.1137/21M1430030
Funding: This work was supported by NSF grant OAC 2003720. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-833755).

†Department of Mathematics, Emory University, Atlanta, GA 30322 USA (ru.huang@emory.edu, yxi26@emory.edu).
‡Center for Applied Scientic Computing, Lawrence Livermore National Laboratory, P. O. Box 808, L-561, Livermore, CA 94551 USA (li50@llnl.gov).

on eliminating high-frequency errors. Multigrid solvers equipped with the proposed smoothers inherit the convergence guarantees and scalability from standard multigrid algorithms and can show improved performance on anisotropic rotated Laplacian problems that are typically challenging for classical multigrid methods. Numerical results demonstrate that a well-trained CNN-based smoother can damp high-frequency errors more rapidly and thus lead to a faster convergence of multigrid than traditional relaxation-based smoothers. Another appealing property of the proposed smoother and the training framework is the ability to generalize to problems of much larger sizes and more complex geometries.

1.1. Related work. There has been an increasing interest in leveraging machine learning techniques to solve PDEs in the past few years. Several researchers have proposed to use machine learning techniques to directly approximate the solutions of PDEs. For example, [26] first proposed to use neural networks to approximate the solutions for both ordinary differential equations and PDEs with a fixed boundary condition. Later, [41] utilized CNNs to solve Poisson equations with a simple geometry, and [4] extended the techniques to more complex geometries. [21, 38] applied machine learning techniques to solve high dimensional PDEs, and [44] focused on applying reinforcement learning to solve nonlinear PDEs. [40] used parameterized realistic volume conduction models to solve Poisson equations, and [23] trained a neural network to plan optimal trajectories and control the PDE dynamics and showed numerical results for solving incompressible Navier–Stokes equations.

Orthogonal to the above methods, a few studies have focused on leveraging neural networks to improve the performance of existing solvers. For example, [37] developed optimization techniques for geometric multigrid based on evolutionary computation. [29] generalized existing numerical methods as neural networks with a set of trainable parameters. [25] proposed a deep learning method to optimize the parameters of prolongation and restriction matrices in a two-grid geometric multigrid scheme by minimizing the spectral radius of the iteration matrix. [17] used neural networks to learn prolongation matrices in multigrid in order to solve diffusion equations without retraining, and [28] generalized this framework to algebraic multigrid (AMG) for solving unstructured problems.

Meanwhile, researchers have also explored relationships between CNNs and differential equations to design better neural network architectures. For instance, [22] designed MgNet which uses multigrid techniques to improve CNNs. [20, 12] scaled up CNNs by interpreting the forward propagation as nonlinear PDEs.

Here, we would like to highlight the work [24], which proposes to use CNNs and U-net [33] to learn a correction term to the Jacobi method for solving Poisson equations. This approach is shown to preserve convergence guarantees. Since multigrid methods are known to be more scalable than Jacobi, we extend this idea to improve multigrid methods by designing optimal smoothers in this paper. To the best of our knowledge, our approach is the first attempt to use CNNs to learn the smoother at each level of multigrid with more than two levels and exhibits good generalization properties to problems with different sizes, geometries, and variable coefficients.

The outline of the paper is organized as follows. In section 2, we review the background of the multigrid method and its convergence results. In section 3, we propose an adaptive learning framework for learning optimized smoothers for constant coefficient PDEs on structured meshes and extend this framework to variable coefficient problems in section 4. We provide interpretation of the learned smoothers in section 5 and demonstrate the performance of the proposed methods through extensive numerical examples in section 6. Finally, we draw some conclusions in section 7.

2. Preliminaries and theoretical background. In this section, we review the classical convergence theory of iterative methods for solving a linear system of equations,

$$Au = f, \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite (SPD) and $u, f \in \mathbb{R}^n$. Iterative methods generate a sequence of improving approximations to the solution of (2.1), in which the approximate solution u_k at iteration k depends on the previous ones. Formally, an iterative solver can be expressed as

$$u_k = \Phi(u_0, f, k), \tag{2.2}$$

where the solver $\Phi : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{Z}^+ \to \mathbb{R}^n$ is an operator that takes the initial guess u_0, right-hand-side vector f and generates u_k at iteration k.

2.1. Relaxation methods. Iterations based on relaxations can be written as

$$u_{k+1} = \left(I - M^{-1}A\right) u_k + M^{-1}f = Gu_k + M^{-1}f, \quad G = I - M^{-1}A, \tag{2.3}$$

where M is the relaxation matrix and G is the iteration matrix. Standard relaxation approaches include weighted Jacobi method with $M = \omega^{-1}D$, where D denotes the diagonal of A, and the Gauss–Seidel method with $M = D - L$, where $-L$ is the strict lower triangular part of A. Denoting by $e_k = u_* - u_k$ the error at iteration k, where u_* is the exact solution of (2.1), it follows that $e_k = G^k e_0$. The following theorem gives a general convergence result for $\lim_{k\to\infty} e_k = 0$.

THEOREM 2.1 ([35, Theorem 4.1]). *Denote by $\rho(G)$ the spectral radius of G. The iteration* (2.3) *converges for any initial vector u_0 if and only if $\rho(G) < 1$.*

Notice that $\rho(G)$ represents the asymptotic convergence rate, which, however, does not, in general, predict error reduction for a few iterations [10]. When relaxation methods are used as multigrid smoothers, they are typically applied $O(1)$ times in each smoothing step. Thus, the convergent smoothers defined as follows can guarantee a better smoothing effect.

DEFINITION 2.2 (convergent smoother in energy norm). *Assuming A is SPD, relaxation matrix M is called a* convergent smoother *in the energy norm if $\|Ge_k\|_A < \|e_k\|_A$ for all e_k, where $G = I - M^{-1}A$ and $\|x\|_A^2 = x^{\mathsf{T}}Ax$.*

It can be shown that M is a convergent smoother if and only if $\|G\|_A < 1$, where $\|G\|_A = \sup_x\{\|Gx\|_A : \|x\|_A = 1\}$ or $M^{\mathsf{T}} + M - A$ is SPD. Since $\rho(G)$ is easier to compute than $\|G\|_A$ and $\rho(G) < 1$ is a necessary condition for both asymptotic convergence and single-iteration convergence, $\rho(G)$ is still often used as a metric of convergence rate of smoothers.

Though relaxation schemes can have very slow convergence when being used as a solver, they are known to be very efficient for smoothing the error especially for elliptic-type PDEs. That is, after a few iterations, the remaining error varies slowly relative to the mesh grid and thus can be approximated well on a coarser grid. This property is explored in multigrid methods as discussed in the next section.

2.2. Multigrid methods. Multigrid methods exploit a hierarchy of grids with exponentially decreasing numbers of degrees of freedom on coarser levels, starting with the original problem on the finest level. On each level, the computational cost is proportional to the problem size; therefore, the overall complexity is still linear. Smoothing and coarse-grid correction are the two main components of multigrid, designed to

be complementary to each other in order to achieve fast convergence, i.e., aiming at eliminating the "high-frequency" (oscillatory) and "low-frequency" (smooth) errors, respectively, which usually correspond to eigenvectors of $M^{-1}A$ with large and small eigenvalues. Relaxation-based approaches such as weighted Jacobi and Gauss–Seidel are typical choices of multigrid smoothers, as these methods are inexpensive to apply and can effectively remove high-frequency errors for elliptic-type PDEs. On the other hand, the effectiveness of coarse-grid correction on low-frequency errors is due to the fact that smooth errors can be interpolated accurately.

When dealing with hard problems such as ones with irregular anisotropy, anisotropy not aligned along the coordinate axes, or complex geometries, the efficiency of traditional smoothers can deteriorate, in which cases, stronger and often more expensive smoothers are needed such as block smoothers [15, 5], ILU-based smoothers [45], and smoothers based on Krylov methods [2, 27]. Nevertheless, finding robust and efficient smoothers still remains a challenging problem for multigrid methods.

Convergence theory of two-grid methods has been well studied [7, 9, 16, 46] through the error propagation operator E_{TG} of the following form:

$$E_{\mathrm{TG}} = \left(I - M^{-1}A\right)\left(I - P\left(P^{\mathsf{T}}AP\right)^{-1}P^{\mathsf{T}}A\right), \tag{2.4}$$

where M is the smoother, $P \in \mathbb{R}^{n\times n_c}$ is the prolongation operator, P^{T} is typically the restriction operator for symmetric A, and $P^{\mathsf{T}}AP$ is the Galerkin coarse-grid operator. Generally, a smaller $\|E_{\mathrm{TG}}\|_A$ indicates faster convergence for two-grid methods.

In this paper we choose standard prolongation operators P defined in [35] and only focus on using CNNs to parameterize M. The following theorem summarizes the main convergence result in [16] with respect to M and P.

THEOREM 2.3 ([16]). *Assuming $M^{\mathsf{T}} + M - A$ is SPD, denote by*

$$\tilde{M} = M^{\mathsf{T}}(M^{\mathsf{T}} + M - A)^{-1}M \tag{2.5}$$

the symmetrized smoother. Let $R \in \mathbb{R}^{n_c\times n}$ be any matrix such that $RP = I$ and

$$K = \max_{e\neq 0}\frac{\|(I - PR)e\|_{\tilde{M}}^2}{\|e\|_A^2}. \tag{2.6}$$

We have $K \geq 1$ and $\|E_{\mathrm{TG}}\|_A \leq (1 - 1/K)^{1/2}$.

The quantity K in (2.6), corresponding to the so-called *weak approximation property* [8], essentially measures how accurately interpolation approximates the eigenvectors of $M^{-1}A$ proportional to the corresponding eigenvalues. The optimal K yields an *ideal uniform bound of convergence rate*, which is often used to analyze convergence rate of smoothers in two-grid methods [1].

DEFINITION 2.4 (ideal uniform convergence bound). *Suppose P takes form $P = \binom{W}{I}$ as in standard multigrid algorithms, where $R = \begin{pmatrix}0 & I\end{pmatrix}$ and $S^{\mathsf{T}} = \begin{pmatrix}I & 0\end{pmatrix}$. Denoting by K_* the minimum K in (2.6) over P, we define quantity β_* such that*

$$\beta_*^2 = (1 - 1/K_*) = [1 - \lambda_{\min}\left(\left(S^{\mathsf{T}}\tilde{M}S\right)^{-1}\left(S^{\mathsf{T}}AS\right)\right)], \tag{2.7}$$

which can be considered as the ideal uniform bound of convergence rate [16].

Extension from two-grid methods to multigrid methods is straightforward. This can be done by recursively applying two-grid methods on the coarse-grid system; see

Algorithm 3.1 for a brief description of standard multigrid V-cycle. Notice that the smoother $M^{(l)}$ at level l is only required to eliminate errors that are $A^{(l)}$-orthogonal to $\mathrm{Ran}(P^{(l)})$ in order to have fast convergence. This property will be used to design efficient training strategies for learning neural smoothers in the next section.

3. Learning deep neural smoothers for constant coefficient PDEs. The convergence of multigrid V-cycle heavily depends on the choice of smoothers. Classical off-the-shelf smoothers such as weighted Jacobi or Gauss–Seidel exhibit near-optimal performance on simple Poisson equations and generally lose their efficiency on other types of PDEs. In this section, we formulate the design of smoothers as a learning task and train a single neural network to parameterize the action of the inverse of the smoother at a given grid level for constant coefficient PDEs discretized on structured meshes. The learned smoothers are represented as a sequence of convolutional layers and trained in an adaptive way guided by the multigrid convergence theory.

Algorithm 3.1 Multigrid V-cycle for solving $Au = f$

1: Presmoothing: $u^{(l)} = u^{(l)} + (M^{(l)})^{-1}(f^{(l)} - A^{(l)}u^{(l)})$
2: Compute fine-level residual, $r^{(l)} = f^{(l)} - A^{(l)}u^{(l)}$, and restrict it to the coarse level: $r^{(l+1)} = (P^{(l)})^{\mathsf{T}} r^{(l)}$
3: **if** $l+1$ is the last level **then**
4: Solve $A^{(l+1)}u^{(l+1)} = r^{(l+1)}$
5: **else**
6: Call multigrid V-cycle recursively with $l = l+1$, $f^{(l+1)} = r^{(l+1)}$, and $u^{(l+1)} = 0$
7: **end if**
8: Prolongate the coarse-level approximation and correct the fine-level approximation: $u^{(l)} = u^{(l)} + P^{(l)}u^{(l+1)}$
9: Postsmoothing: $u^{(l)} = u^{(l)} + (M^{(l)})^{-1}(f^{(l)} - A^{(l)}u^{(l)})$

3.1. Formulation. We define a PDE problem as the combination of PDE class $\mathcal{A}$, forcing term $\mathcal{F}$, and boundary condition $\mathcal{G}$. To solve the problem numerically on a two dimensional square domain, we discretize it on a grid of size $N \times N$, which leads to solving linear system $Au = f$, where $A \in \mathbb{R}^{N^2 \times N^2}$ and $f \in \mathbb{R}^{N^2}$. Our goal is to train smoothers $M^{(0)}, \ldots, M^{(L-1)}$ on the first L levels of a multigrid solver that has $L+1$ levels. We assume that the multigrid solver uses the same smoother for both the presmoothing and postsmoothing steps (c.f., lines 1 and 9 in Algorithm 3.1, respectively) and uses direct methods as the coarsest-level solver. Denoting by $\Phi^{(0)}$ the multigrid hierarchy from level 0, the training objective for $\Phi^{(0)}$ is to minimize the error

$$\|\Phi^{(0)}(u_0, f, k) - u_*\|_2, \tag{3.1}$$

where u_0 is a given initial guess, u_* is the exact solution, and $u_k = \Phi^{(0)}(u_0, f, k)$ is the approximate solution by performing k steps of V-cycles with $\Phi^{(0)}$.

The advantage of minimizing (3.1) instead of the norm of the associated iteration matrix is that (3.1) can be evaluated and optimized more efficiently. For example, in two-grid methods, $\Phi^{(0)}(u_0, f, k) - u_* = E_{TG}^k e_0$ for each exact solution u_* and an arbitrary initial guess u_0. When multiple initial guesses are used to minimize (3.1) jointly with different iteration number k, the convergence property of the trained smoother can be justified by the following theorem, which shows that when the loss of (3.1) is small, the norm of the associated two-grid operator, E_{TG}, should also be small. It is easy to see that this property also holds true for multigrid operators.

THEOREM 3.1 ([19]). *For any matrix $X \in \mathbb{R}^{n\times n}$ and $z \in \mathbb{R}^n$ that is uniformly distributed on a unit n-sphere, we have*

$$\mathbb{E}(n\|Xz\|_2^2) = \|X\|_F^2.$$

In this paper, we fix $\mathcal{A}$ but vary $\mathcal{F}$ and $\mathcal{G}$, and we learn multigrid smoothers that are appropriate for different PDEs from the same class. Specifically, we train the multigrid solvers on a small set of discretized problems,

$$\mathcal{D} = \bigcup_{j=1}^{Q} \{A, f_j, (u_0)_j, (u_*)_j\}, \tag{3.2}$$

with the presumption that the learned smoothers have good generalization properties: they can perform well on problems with much larger grids of different geometries.

As a motivating example, we consider the following diffusion problem:

$$-\nabla \cdot (g\nabla u(x,y)) = f(x,y), \tag{3.3}$$

where g is assumed to be constant in this section. We will consider the more general form $g(x,y)$ in the next section.

Since the stencils for discretizing (3.3) would be identical for constant g on structured meshes, the dynamics of the problems are spatial invariant and independent of the specific location in the domain. Thus, we can parameterize the action of inverse of the smoother $(M^{(l)})^{-1}$ by one single CNN, $H^{(l)}$, with only convolutional layers. This parameterization has several advantages. First, on an $N \times N$ grid, $H^{(l)}$ only requires $O(N^2)$ computation and has a few parameters. Second, $H^{(l)}$ can be readily applied to problems defined on different grid sizes or geometries. Lastly, which is more important, Theorem 3.2 justifies the use of this parameterization to construct convergent smoothers.

THEOREM 3.2. *For one fixed matrix A, there exists a finite sequence of convolution kernels $\{\omega^{(j)}\}_{j=1}^{J}$ such that the convolutional factorization $H = \omega^{(J)} * \ldots \omega^{(2)} * \omega^{(1)}$ satisfies $\|I - HA\|_A < 1$ indicating H is a convergent smoother.*

Proof. Based on the universality property of deep CNN without fully connected layers [47], we know that H can approximate the linear operator A^{-1} to an arbitrary accuracy measured by some norms when J is large enough. Thus, theoretically, HA can be very close to an identity mapping if parameterized properly. Since all matrix norms are continuous and equivalent, $\|I - HA\|_A$ can be less than 1 for certain J measured in matrix A-norm. ∎

3.2. Training and generalization. In this section, we propose several strategies for training multigrid solvers using CNNs as smoothers. We will also discuss their advantages and disadvantages.

The first training strategy is to train $H^{(l)}$ separately for each multigrid level $l = 0, \ldots, L-1$, where we construct a training set $\mathcal{D}^{(l)}$ similar to (3.2) for the operator $A^{(l)}$. That is, we train $H^{(l)}$ to make iteration (2.3) convergent by minimizing the error between the approximate solution obtained at iteration k and the ground truth solution. As suggested in [24], we also choose different iteration number k, $1 \le k \le b$, in the training, so that $H^{(l)}$ learns to converge at each iteration, where larger b mimics the behavior of solving problems to higher accuracy while smaller b mimics inexpensive smoothing steps in multigrid.

This training strategy is simple, and the trainings on different levels are totally independent. However, we found the obtained $H^{(l)}$ usually does not exhibit a good smoothing property of reducing high-frequency errors, especially when $H^{(l)}$ is a shallow neural network. This phenomenon is expected since the training strategy does not consider the underneath coarser-grid hierarchy and tries to reduce errors over the whole spectrum of $A^{(l)}$. In contrast, a well-trained $H^{(l)}$ with high complexities, deeper in the layers and larger in the convolution kernels, can approximate the action of the inverse of $A^{(l)}$ well, but using it as a smoother is not efficient nonetheless, and moreover, the training cost will be significantly higher. We denote the smoothers learned by this strategy by CNN smoothers.

A second training approach is to optimize the objective function (3.1) directly over $M^{(l)}$ at all levels, $l = 0, \ldots, L-1$. This approach targets optimizing convergence of the overall multigrid V-cycles and considers both the smoothing and the coarse-grid correction. However, training the CNNs at all levels together turns out to be prohibitively expensive. Since this approach is not robust and stable in practice, we exclude this approach in the experiments.

Finally, we propose an efficient adaptive training strategy that can impose the smoothing property by recursively training the smoothing CNN at a fine level. We denote the smoothers learned by this strategy by α-CNN smoothers. The training process starts from the second coarsest level and is repeatedly applied to the finer levels, given that the smoothers at coarser levels have been already trained, so that the solve with the coarse-grid operator can be replaced with a V-cycle using the available multigrid hierarchy at one level down. The adaptive training algorithm is sketched in Algorithm 3.2. Figure 1 illustrates the procedure of adaptively training a 5-level multigrid solver in 4 stages, starting at level 3. The loss is given by

$$L^{(3)} = \sum_{j,k} \| \Phi^{(3)} \left(\left(u_0^{(3)} \right)_j, \left(f^{(3)} \right)_j, k \right) - \left(u_*^{(3)} \right)_j \|_2,$$

where $\Phi^{(3)}$ represents the 2-level multigrid with levels 3 and 4. In the second stage, the training proceeds at level 2 for CNN $H^{(2)}$ utilizing the underlying 2-level hierarchy obtained from the first stage. This procedure continues until $H^{(0)}$ is computed at the finest level and the entire training is completed, so the resulting multigrid hierarchy $\Phi^{(0)}$ can be used for solving systems of equations with $A^{(0)} \equiv A$.

Another appealing property of the proposed training approach is the updatability of smoothers using neural networks. The trained smoothers can be updated in another

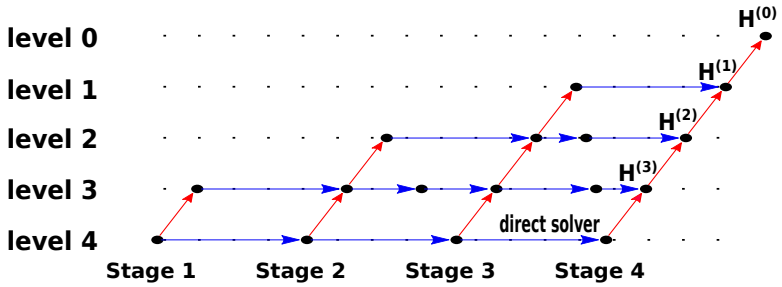


FIG. 1. *The proposed adaptive training strategy for k levels, which starts from level $k-2$ and proceeds upwards. When $H^{(l)}$ is being trained, lower level $H^{(j)}, j = l+1, \ldots, k-2$, is used for the solve with coarse-grid operators and remains unchanged.*

Algorithm 3.2 Adaptive training of multigrid CNN smoothers

1: **Input:** Multigrid hierarchy: number of multigrid levels $L+1$, coarsest-grid solver at level L, namely $\Psi^{(L)}$, coefficient matrix $A^{(l)}$ where $A^{(0)} = A$, and interpolation operator $P^{(l)}$ for $l = 0, \ldots, L-1$. Size of training set Q. Maximum allowed number of smoothing steps b

2: **Output:** Smoothers $H^{(0)}, \ldots H^{(L-1)}$

3: **for** $l = L-1, \ldots, 0$ **do**

4: Construct training set:

$$\mathcal{D}^{(l)} = \bigcup_{j=1}^{Q} \{t_j\}, \quad t_j^{(l)} = \{A^{(l)}, f_j^{(l)}, (u_0^{(l)})_j, (u_*^{(l)})_j\}$$

5: Initialize the weights of $H^{(l)}$

6: Perform stochastic gradient descent to minimize loss function:

$$\sum_{t_j^{(l)} \in \mathcal{D}^{(l)}, k \sim \mathcal{U}(1,b)} \|\Phi^{(l)}((u_0^{(l)})_j, f_j^{(l)}, k) - (u_*^{(l)})_j\|_2$$

With $\Phi(u_0, f, 0) \equiv u_0$, run forward propagation by

$$\begin{aligned}
\Phi^{(l)}(u_0, f, k) &= \Phi^{(l)}(u_0, f, k-1) + \Psi^{(l)}(r_{k-1}),\\
r_{k-1} &:= f - A^{(l)}\Phi(u_0, f, k-1),\\
\Psi^{(l)}(r_{k-1}) &= t_{k-1} + H^{(l)}(r_{k-1} - At_{k-1}),\\
t_{k-1} &:= H^{(l)}(r_{k-1}) + P^{(l)}\Psi^{(l+1)}((P^{(l)})^{\mathsf{T}} s_{k-1}),\\
s_{k-1} &:= r_{k-1} - AH^{(l)}(r_{k-1}),
\end{aligned}$$

and *only* update $H^{(l)}$ by back propagation

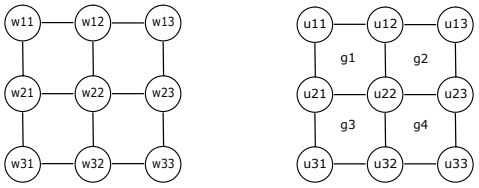
7: **end for**

training process by injecting the errors that cannot be effectively reduced by the current multigrid solver back to the training set. Specifically, to improve the smoothers in a trained multigrid solver $\Phi^{(0)}$, we can first apply $\Phi^{(0)}$ to homogeneous equation $Au = 0$ for k steps with a random initial vector u_0 and get the approximate solution u_k, i.e., $u_k = \Phi^{(0)}(u_0, 0, k)$, then inject the (restricted) residual, $r_k^{(l)} = (P^{(l-1)})^{\mathsf{T}} r_k^{(l-1)}$ with $r_k^{(0)} = -Au_k$, to the training set at each level l, and finally retrain $\Phi^{(0)}$ as before with the new augmented training sets using the existing $H^{(l)}$ in the multigrid hierarchy as the initial values.

4. Learning deep neural smoothers for variable coefficient PDEs. In this section, we extend the adaptive training framework proposed in section 3 to design optimal smoothers for solving variable coefficient PDEs:

$$-\nabla \cdot (g(x,y)\nabla u(x,y)) = f(x,y). \tag{4.1}$$

To better illustrate the difficulty of dealing with variable coefficient PDEs, we simplify our discussion and consider discretizing (4.1) using nine-point stencils with grid spacing h. See the left subfigure of Figure 2 for a demonstration of the 3×3



(a) Weight stencil (b) Grid point

FIG. 2. *Demonstration of weight stencils and grid points.*

neighborhood of the grid point u_{22}. The equation corresponds to the grid point u_{22} reads

$$
\begin{aligned}
&-\frac{1}{3h^2}(g_1u_{11}+g_2u_{13}+g_3u_{31}+g_4u_{33})\\
&-\frac{1}{6h^2}((g_1+g_2)u_{12}+(g_2+g_4)u_{23}+(g_3+g_4)u_{32}+(g_1+g_3)u_{21})\\
&+\frac{2}{3h^2}(g_1+g_2+g_3+g_4)u_{22}=f_{22},
\end{aligned}
$$

which is equivalent to applying a 3×3 weight stencil to the 3×3 neighborhood of u_{22},

$$\sum_{i,j} w_{ij}u_{ij}=f_{22},\qquad i,j\in\{0,1,2\},$$

where w_{ij} is computed according to the function $g(x,y)$ and is shown in the right subfigure of Figure 2. When $g(x,y)$ is constant, the coefficients w_{ij} corresponding to each interior 3×3 stencil are identical. Thus, we can parameterize M^{-1} by a single CNN as a stack of convolution kernels $\{\phi_i\}$. The weights of each convolution kernel ϕ_i are shared over all grid points. However, when $g(x,y)$ is variant, the weight stencils W_{ij} and W_{lm} at two different locations can have completely different dynamics (e.g., W_{ij} can be strong in the x-axis and weak in the y-axis while W_{lm} is strong in the y-axis and weak in the x-axis). In this case, a smoothing kernel H that is learned to smooth the error at one grid point might be ineffective in smoothing the error at another point. As a result, the optimal smoothing kernel H_{ij} associated with each grid point should be conditioned on the location for variable coefficient problems.

In order to generate unshared convolution kernels which are dimension-invariant, we propose to learn a function which can adaptively adjust the kernels based on the spatial information. In particular, we will design neural network architectures which can map each grid representation to a stack of convolution kernels that can be used to efficiently smooth the error at different locations.

We point out the main difference and connection between the design of the smoothers of the variable coefficient case and the constant coefficient case described in the previous section. In the constant coefficient case, the α-CNN smoothers are parameterized by neural networks directly, and the parameters learned by Algorithm 3.2 are the parameters of the smoothers. However, in the variable coefficient case, instead of learning the parameters of the smoothers directly, we use Algorithm 3.2 to learn the parameters of a mapping function parameterized by the neural networks which generates the smoothers. The smoothers generated by the mapping function in the variable coefficient case have the same structure (CNN structure) as in the constant

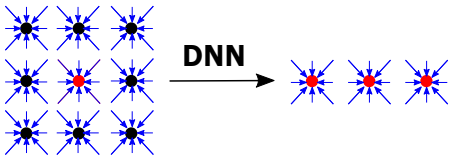


FIG. 3. *The architecture of inferring the smoothing kernels for the central point in the stencil. A fully connected neural network takes nine* 3×3 *stencils as input and outputs three convolution smoothing kernels for the current grid point. DNN stands for deep neural networks.*

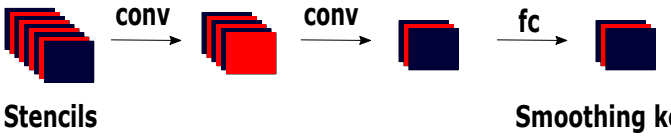


FIG. 4. *The framework of constructing* 3 *smoothing kernels by applying two convolutional layers (conv) and one fully connected layer (fc) to a feature map. The feature maps have* 9,6,3*, and* 3 *channels, respectively, and each channel contains kernels of size* 3×3.

coefficient case. We discuss two different parameterization methods of the mapping function in the following sections.

4.1. Parameterization with fully connected layers. In the first approach, we consider using multiple layer perceptron to construct the mapping from the grid representation to the smoothing kernels at each grid point. Although the stencil at each grid point has already contained the spatial information, we find that only using the stencil information as the representation is not sufficient to learn efficient smoothing kernels and the generalization usually performs poorly. Instead, we suggest incorporating the neighborhood information into the grid representation. More specifically, we construct each grid representation as an 81×1 vector which consists of the stencils in the 3×3 neighborhood of the current point under consideration. In this case, the feature map $\mathbb{M}$ for an $N \times N$ grid has the size of $N^2 \times 81$. The mapping is then parameterized by a fully connected neural network which takes the representation of each grid point as input and infers the weights of the k output smoothing kernels of size 3×3. See Figure 3 for an illustration of this architecture. To smooth the error at the central point in the stencil, we train a fully connected neural network which takes nine 3×3 stencils with 81 parameters in total and outputs three 3×3 convolution kernels that are used to smooth the error at this point. On each level of the multigrid solver, we only construct one such neural network based on the adaptive training strategy discussed in section 3.2.

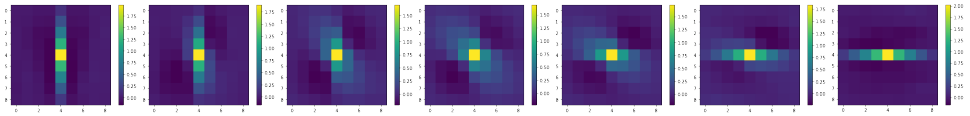
4.2. Parameterization with convolutional layers. Deep neural networks using fully connected layers often require a large amount of parameters in order to well approximate a function and also have high training cost. In order to reduce the training cost, instead of constructing a feature map $\mathbb{M} \in \mathbb{R}^{N^2 \times 81}$ by flattening and stacking the stencils and applying fully connected neural networks, an alternative approach is to feed into the neural network with 9 channels with each channel corresponding to one stencil in the 3×3 neighborhood of the point under consideration. The deep neural network is parameterized by several convolution kernels followed by a fully connected layer. The outputs of the neural network are k smoothing kernels. This architecture is illustrated in Figure 4. We will show in numerical experiments that

this approach can achieve a comparable performance with fully connected layers but requires much fewer parameters.

Remark. The construction of α-CNN is similar to sparse approximate inverse preconditioners [3, 13, 18]. However, instead of using techniques such as Frobenius norm minimization, training of α-CNN is guided by the convergence from lower-level multigrid hierarchy. This generally leads to better smoothing properties in conjunction with other AMG components. For constant coefficient problems, sparse approximate inverse preconditioners can also be computed and applied in stencil forms without forming the whole matrix. On the other hand, for variable coefficients, our approach uses a neural network to produce, at each mesh point, the smoother stencils without storing them. As such, an advantage of the proposed method is to be able to generalize to problems of different sizes without reconstructions.

5. Interpretation of learned smoothers. In this section, we illustrate the patterns of the learned smoothers. Notice that the experiements conducted in this section are for visualization purposes and use a different set of parameters than those used in the experiments in section 6. We consider the anisotropic rotated Laplacian problem (6.1) parameterized by the angle θ of the anisotropy and conductivity ξ. We fix $\xi = 100$ and train smoothers for problems with a variety of $\theta \in \{0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}, \frac{\pi}{2}\}$. For each problem, we use a two-grid solver, and on the fine level we train a smoother which consists of one convolution kernel of size 9×9. We use linear activation in order to illustrate the action of the convolution kernels as the smoothers. The trained convolution kernels corresponding to different θ are shown in Figure 5. The results show that large values in each kernel are gathered symmetrically about the center and the angles of the large values of each kernel also align with the angle of the anisotropy of the problem. These patterns demonstrate that the learned smoothing kernels are able to smooth the error in correct directions, which can be viewed as line smoothers truncated in the convolution windows along the direction of strong couplings.

We also increase the number of convolutional layers and study the impact of each convolutional layer on the final smoother. For each problem, we train three convolution kernels of size 9×9 for better visualization (we use 3×3 kernels in section 6 as they are more efficient in practice) and show the results in Figure 6. The first row shows the kernels of the first convolutional layer for each problem, while the second row and the third row show the second layer and the third layer, respectively. The kernels at different layers exhibit different patterns which indicates that each kernel is responsible for smoothing the error in different regions. Since applying three 9×9 convolution kernels sequentially is equivalent to applying a 25×25 convolution kernel, we illustrate the patterns of the effective 25×25 kernels in the last row of Figure 6. The kernels in the last row display similar patterns as in Figure 5 which perfectly align with the anisotropy of the problem.



(a) $\theta = 0$ (b) $\theta = \frac{\pi}{12}$ (c) $\theta = \frac{\pi}{6}$ (d) $\theta = \frac{\pi}{4}$ (e) $\theta = \frac{\pi}{3}$ (f) $\theta = \frac{5\pi}{12}$ (g) $\theta = \frac{\pi}{2}$

FIG. 5. *Patterns of the trained kernels for* (6.1) *with* $\xi = 100$ *and* $\theta \in \{0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}, \frac{\pi}{2}\}$. *For each problem, the smoother only uses one kernel.*

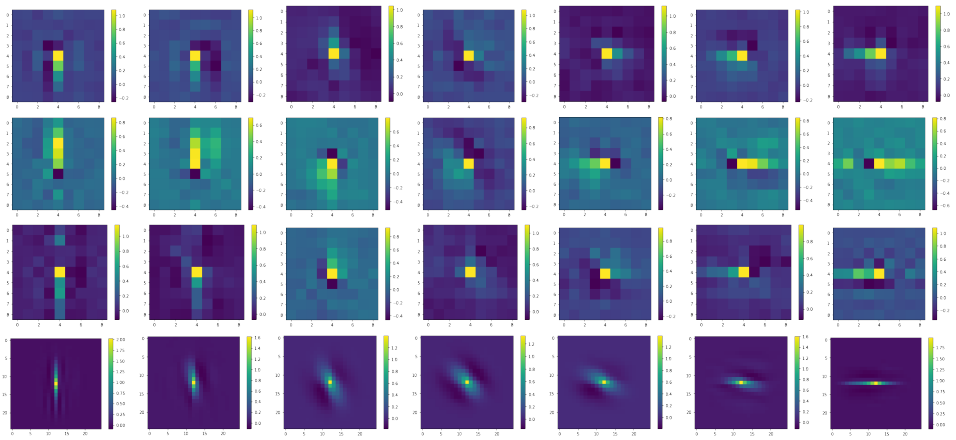


FIG. 6. *Patterns of the trained kernels for* (6.1) *with* $\xi = 100$ *and* $\theta \in \{0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}, \frac{\pi}{2}\}$. *For each problem, the smoother uses three kernels. The first three rows represent the kernels on the first, second, and third layers, respectively, and the last row combines the three kernels into one single kernel for each problem.*

6. Numerical experiments. In this section, we provide numerical examples to demonstrate the smoothing effect of the proposed smoothers. All of the codes were implemented in PyTorch 1.8.1[1] and run on an Intel Core i7-6700 CPU. We use a batch size of 10 and employ the Adam optimizer with a learning rate of 10^{-3} for 500 epochs. The neural network training took roughly 5 hours for each constant coefficient problem and roughly 4 hours for each variable coefficient problem.

6.1. Constant coefficient PDEs. We first consider the following two dimensional anisotropic rotated Laplacian problem:

$$-\nabla \cdot (T \nabla u(x, y)) = f(x, y), \tag{6.1}$$

where 2×2 tensor field T is defined as

$$T = \begin{bmatrix} \cos^2\theta + \xi \sin^2\theta & \cos\theta \sin\theta (1 - \xi) \\ \cos\theta \sin\theta (1 - \xi) & \sin^2\theta + \xi \cos^2\theta \end{bmatrix}, \tag{6.2}$$

where θ is the angle of the anisotropy and ξ is the conductivity. We discretize the operators Δu and u_{xy} in (6.1) using the following stencils, where h is the grid spacing:

$$\frac{1}{4h^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{2h^2} \begin{bmatrix} & -1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & \end{bmatrix}.$$

We use multigrid V-cycles to solve the resulting discretized linear system $Au = f$, where the coefficient matrix A is parameterized with (θ, ξ, n, G), where n is the grid size and G is the geometry of the grid. We show the robustness and efficiency of the proposed neural smoothers on a variety of sets of parameters (θ, ξ, n, G). For each set of the parameters, we train the neural smoothers on a dataset constructed on square domains with small grid size and show that the trained neural smoothers

[1]Code for reproducing the experiments is available at https://github.com/jerryhuangru/Learning-optimal-multigrid-smoothers-via-neural-networks.

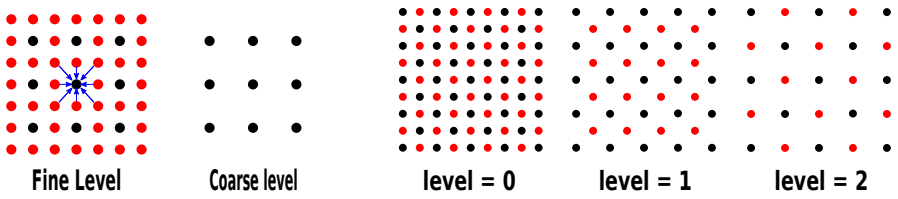


FIG. 7. *Full/red-black coarsening. The fine points are red, and coarse points are black. Full weighting restriction is shown by the arrows to the coarse point at the center.*

can outperform standard ones such as weighted Jacobi. Furthermore, we demonstrate that the trained neural smoothers can be applied to solve much larger problems and problems with more complex geometries without retraining. Since our focus of this work is on smoothers, we adopt standard algorithms for multigrid coarsening and grid-transfer operators. Specifically, we consider full coarsening, which is illustrated in Figure 7 for two dimensional grids, where grid points are coarsened in both x- and y-dimensions. The associated restriction and interpolation are full weighting, a weighted average in a 3×3 neighborhood. The stencils of the restriction and interpolation operators are given by, respectively,

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

We also consider red-black coarsening that has a coarsening factor of about 2 shown in Figure 7 for the first 3 levels. Note that the coarsening on level 0 is essentially a semicoarsening along the 45° angle, and on level 1 the coarsening is performed on the 45°-rotated meshes, which generates the grid on level 2 that amounts to a semicoarsening along the y-dimension. The restriction and interpolation stencils used associated with this coarsening are given by (see [35])

$$\frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}.$$

To evaluate our method, we compare the performance of multigrid using Algorithm 3.1 equipped with convolutional neural smoothers that are trained adaptively (denoted by α-*CNN*), convolutional neural smoothers trained independently (denoted by *CNN*), and weighted Jacobi smoothers (denoted by ω-*Jacobi*) for solving a variety of linear systems. These problems are generated by varying the parameters (ξ, θ, n, G). The weight ω is chosen to be $\frac{2}{3}$ by heuristics for all experiments in this paper.

Training details. First, we train smoothers independently using the first strategy discussed in section 3.2. For each smoother, we construct 50 problem instances of size 16^2. Then, we use the adaptive training framework to train smoothers using Algorithm 3.2. The training process for a 5-level multigrid has 4 stages. At each stage we construct a training data set which contains 50 instances of the problem on each level. All stages have the same size of the coarsest grid. In particular, under the full coarsening scheme, at stage l the problems are constructed on the $(4-l)$th level and have grid size of $(2^{l+2}-1)^2$. Under the red-black coarsening scheme, at stage 1 and stage 2 the problem instances have size of 9^2, and at stage 3 and stage 4 the problems have size of 17^2. This is because when we apply red-black coarsening

to a regular grid, the grid becomes irregular; therefore we need to add zeros to the irregular grid so that we can apply CNNs more efficiently.

Neural networks. We use CNNs to approximate the action of the inverse of the smoothers. In particular, under the full coarsening scheme, for both CNN and α-CNN smoothers, $H^{(l)}$ is parameterized as follows:

$$H^{(l)} = f_5^{(l)}\left(f_4^{(l)}\left(\cdots\left(f_2^{(l)}\left(f_1^{(l)}\right)\right)\cdots\right) + f_6^{(l)}\right), \tag{6.3}$$

where each $f_i^{(l)}$ is parameterized by a 3×3 convolution kernel $\phi_i^{(l)}$. We initialize the weights of $\phi_1^{(l)}, \ldots, \phi_5^{(l)}$ with small values and $\phi_6^{(l)}$ to be the inverse Jacobi stencil so that $H^{(l)}$ is initialized as Jacobi. For red-black coarsening, $H^{(l)}$ is parameterized as

$$H^{(l)} = f_2^{(l)}\left(f_1^{(l)}\right). \tag{6.4}$$

Note that we could use more convolutional layers and also, for each grid point, explore a larger range of the neighborhood, which can typically lead to a faster convergence rate at the price of more computational costs per iteration. The current settings are found to give the best trade-off between convergence rate and time to solution.

Evaluation metrics. We train the smoothers on problems with small grid sizes where the ground truth can be easily obtained. When we test on large-scale problems, it is time consuming to obtain the ground truth. Therefore, when we evaluate the performance, we use the convergence threshold relative residual $\frac{\|f - A\hat{u}\|_2}{\|f\|_2} < 10^{-6}$ as the stopping criterion which can avoid the requirement of exact solutions. We compare both the number of iterations and the runtime for multigrid solvers using different smoothers to reach the same accuracy. To reduce the effect of randomness, for each test problem, we run the multigrid solvers to solve 10 problems with different random right-hand sides and present the averaged numbers.

Convergence rate. Since coarser problems are usually better conditioned, the smoothers on the finest level have the biggest impact on the overall convergence. In this experiment we compare the spectral properties of the smoothers on the finest level. We first compare the spectral radius of the iteration matrices (2.3) constructed by ω-Jacobi smoothers (ω is fixed at $\frac{2}{3}$ in all experiments) and α-CNN smoothers and summarize the results in Table 1. These statistics are calculated on two sets of test problems defined on one 16×16 grid. In the first set, θ is fixed as 0 and $\xi = 100, 200, 300, 400$. In the second set, ξ is fixed at 100 and $\theta = 0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}$. The corresponding comparison of ideal convergence bounds (2.7) on these tests is provided in Table 2. Since we initialize the neural network close to Jacobi, the training is stable. Take the rotated Laplacian problem with $\theta = 0$ and $\xi = 100$ as an example. We use the same learning rate, same number of epochs, and the Adam optimizer to train 20 α-CNN smoothers. The ideal convergence bound has mean of 0.7672 with standard deviation of 0.0042. The spectral radius of iteration matrix has mean of 0.7671 with standard deviation of 0.0041. Since the deviations are typically small, we omit reporting them in the rest of the section.

The results in Table 1 and Table 2 show that for each rotated Laplacian problem, the convergence measure associated with α-CNN smoothers are much smaller than those with ω-Jacobi smoothers and Gauss–Seidel smoothers which indicates a faster convergence can be achieved by multigrid solvers equipped with α-CNN smoothers.

We use the same problem setting as the above tables. We consider the iterative solvers $x_k = Gx_{k-1}$, where G is the 5-level multigrid solver. We compare the spectral

TABLE 1

Spectral radius of iteration matrices (2.3) *of two-grid methods using full coarsening and* ω*-Jacobi with* $\omega = \frac{2}{3}$*, Gauss–Seidel, and* 6*-layered* α*-CNN smoothers for rotated Laplacian problems with different* θ *and* ξ*. The grid size is* 16×16*.*

(θ, ξ)	(0,100)	(0,200)	(0,300)	(0,400)	$(\pi/12, 100)$	$(\pi/6, 100)$	$(\pi/4, 100)$
ω-Jacobi	0.9886	0.9886	0.9886	0.9886	0.9913	0.9934	0.9942
Gauss–Seidel	0.9662	0.9662	0.9662	0.9662	0.9735	0.9797	0.9823
α-CNN	0.7672	0.8060	0.8588	0.7883	0.7743	0.9652	0.9728

TABLE 2

Ideal convergence bound (2.7) *for the same methods and problems in Table* 1*.*

(θ, ξ)	(0,100)	(0,200)	(0,300)	(0,400)	$(\pi/12, 100)$	$(\pi/6, 100)$	$(\pi/4, 100)$
ω-Jacobi	0.9886	0.9886	0.9886	0.9886	0.9913	0.9934	0.9942
Gauss–Seidel	0.9675	0.9675	0.9675	0.9675	0.9748	0.9807	0.9833
α-CNN	0.7671	0.8060	0.8588	0.7883	0.7743	0.9651	0.9728

TABLE 3

Spectral radius of the iteration matrices corresponding to the 5*-level multigrid with full coarsening and* ω*-Jacobi,* $\omega = \frac{2}{3}$*, Gauss–Seidel, and* 6*-layered* α*-CNN smoother for* (6.1) *with different* θ *and* ξ*. The mesh size is* 16×16*.*

(θ, ξ)	(0,100)	(0,200)	(0,300)	(0,400)	$(\pi/12, 100)$	$(\pi/6, 100)$	$(\pi/4, 100)$
ω-Jacobi	0.9853	0.9918	0.9940	0.9951	0.9436	0.8981	0.8837
Gauss–Seidel	0.9564	0.9755	0.9820	0.9853	0.8566	0.7776	0.7643
α-CNN	0.6816	0.8189	0.8805	0.8936	0.4534	0.4547	0.4216

radius of the iteration matrices G of 5-level multigrid solvers equipped with different smoothers and summarize the results in Table 3. The results show that the smoothers can not only efficiently smooth the finest-level errors but also have faster convergence overall as a 5-grid solver compared to ω-Jacobi and Gauss–Seidel. Since ω-Jacobi smoothers have better parallel efficiency than Gauss–Seidel smoothers, we will only compare neural smoothers with ω-Jacobi smoothers in the remaining section.

Smoothing property. To show that our proposed method can learn the optimal smoother with the best smoothing property, for each eigenvector v (that has the unit 2-norm) of the fine-level operator A associated with parameters $\theta = \frac{5\pi}{12}$, $\xi = 100$, $N = 16$ on a square domain, we compute its convergence factor $\|v - H^{(0)}(Av)\|_2$, where $H^{(0)}$ is the smoother on the finest level. An efficient smoother should lead to small convergence factors for eigenvectors associated with large eigenvalues. The results are shown in Figure 8, where the eigenmodes are listed in the descending order of the corresponding eigenvalues. The CNN smoother can reduce low-frequency errors more rapidly than ω-Jacobi; however, both of them have comparable performance for damping high-frequency errors. In contrast, α-CNN has the best performance, which exhibits a superior smoothing property, as the convergence factors corresponding to the large eigenvalues are about 6 times smaller than those with the other two smoothers.

Generalization property. To illustrate that our proposed method is useful, besides showing the statistics, we present the actual iteration numbers and runtime for multigrid solvers to converge. Also for a given PDE problem, we want to only train the neural smoothers once, that is, the neural smoothers need not to be retrained if we increase the grid size or change the geometry of the problem. In this experiment,

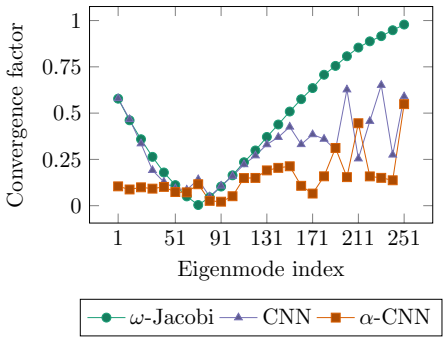


FIG. 8. *Convergence factors of ω-Jacobi with $\omega = \frac{2}{3}$, CNN, and α-CNN smoothers to the eigenvectors of A for (6.1) on a 16×16 grid, where $\theta = \frac{5\pi}{12}$ and $\xi = 100$. The eigenvectors are sorted in descending order of the corresponding eigenvalues.*

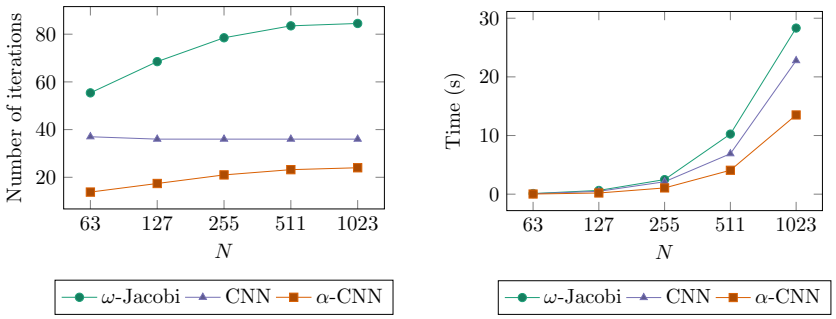


FIG. 9. *Numbers of iterations and runtime required by multigrid with full coarsening to reach convergence tolerance 10^{-6} for solving (6.1) on 63^2, 127^2, 255^2, 511^2, and 1023^2 grids with parameters $\xi = 100$ and $\theta = \frac{5\pi}{12}$ on square domains.*

we first show that the trained smoothers can be generalized to different grid sizes without retraining. We fix the parameter of the problems to be $\xi = 100$ and $\theta = \frac{5\pi}{12}$ on one square domain. We show in Figure 9 that for problems of size 1023^2, multigrid methods using α-CNN smoothers converge faster in terms of the number of iterations than multigrid methods using CNN and ω-Jacobi smoothers by factors of 1.5 and 3.5, respectively. Since the cost of applying α-CNN smoothers is more than ω-Jacobi, the time for iterations of multigrid methods using α-CNN is only faster than that using CNN and ω-Jacobi by factors of 1.68 and 2.1, respectively.

Since the CNN smoothers were trained independently, they are not as successful as α-CNN to capture the smoothing property of reducing errors that cannot be reduced by lower levels of multigrid. Hence, we only compare α-CNN and ω-Jacobi smoothers in the rest of the paper. Next we fix the parameters of the problems to be $\theta = \frac{\pi}{4}$, $\xi = 100$ and show that the trained α-CNN smoothers can be generalized to problems with two different geometries (shown in Figure 10) without retraining.

The results for the two different domains are shown in Figure 11. We can see that since we are using the convolutional layers to approximate the inverse of the smoothers, α-CNN uses the information in the neighborhood information to smooth the error at each grid point, and therefore without retraining, the smoother trained on the square domain can still lead multigrid methods to converge 4.1 times faster

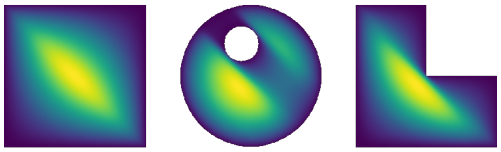


FIG. 10. *Ground truth solutions on square, cylinder, and L-shaped domains.*

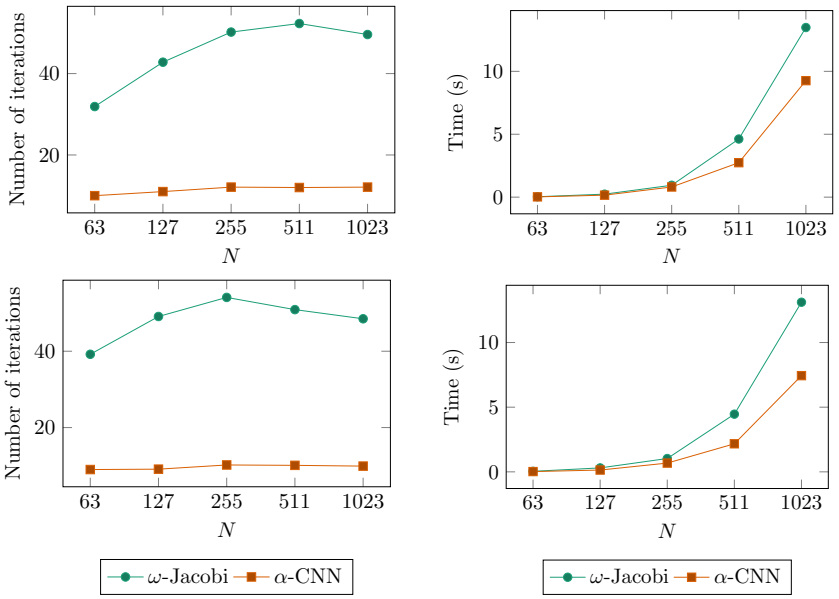


FIG. 11. *Numbers of iterations and runtime required by multigrid solvers for solving* (6.1) *with parameters* $\theta = \frac{\pi}{4}$ *and* $\xi = 100$ *on the cylinder domain (top two subfigures) and the L-shaped domain (bottom two subfigures).*

in terms of the number of iterations and 1.5 times faster in time to solution on the cylinder domain for problems of size 1023^2. On the L-shaped domain for the same sized problem, the performance improvement is 4.9 times and 1.8 times faster in terms of the number of iterations and the time for iterations. We show in Figure 12 that our proposed method can learn optimized smoothers for a variety of problems given by different parameters on the square domain and is not restricted to the choice of coarsening schemes in multigrid. In particular, for $\theta = \frac{5\pi}{12}$, with full coarsening, the multigrid method using α-CNN smoothers is 19.2 times faster in terms of the number of iterations and achieves a speedup of factor 4.4 in the time for iterations. When the red-black coarsening scheme is used, the multigrid solver with α-CNN smoothers can still require much fewer iterations than the one with ω-Jacobi by 1.9 times and converges about 1.3 times faster in time.

Next, we show that a single smoother can be learned that works for all the problems discussed above. Instead of training a smoother for each problem individually, we construct a training set that contains the problems for $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ and $\xi = 100$. We show in Figure 13 that the performance of a single smoother for all the problems is slightly worse than the individual training but still outperforms

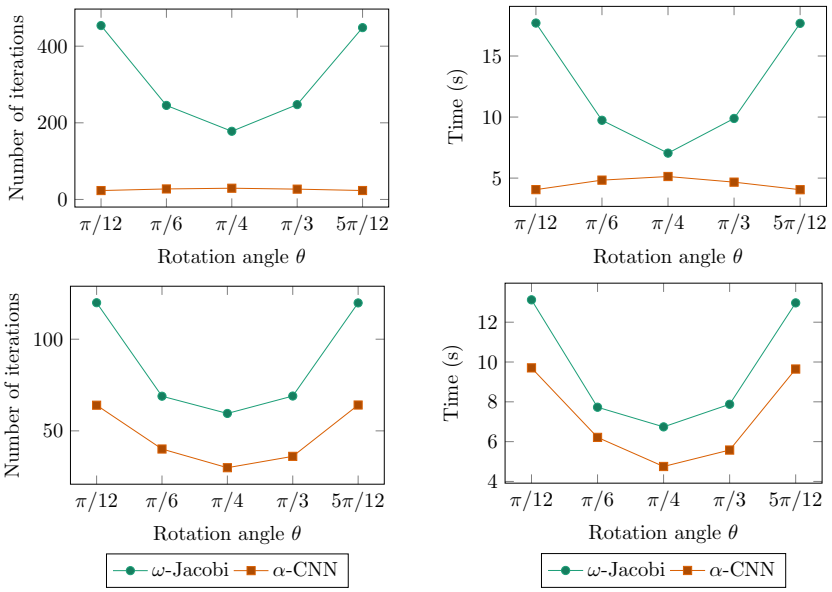


FIG. 12. *Numbers of iterations and runtime required by multigrid solvers for solving* (6.1) *with* $n = 511^2$, $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$, *and* $\xi = 100$. *The top and bottom two subfigures show the performance with full coarsening and red-black coarsening, respectively.*

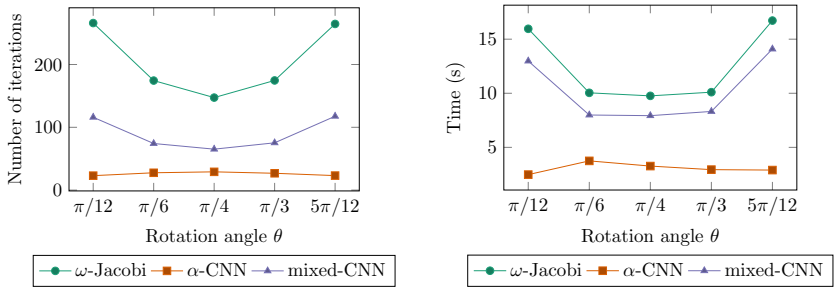


FIG. 13. *Numbers of iterations and runtime required by multigrid solvers with full coarsening for solving* (6.1) *of size* $n = 511^2$ *with* $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ *and* $\xi = 100$. *The smoother is trained from a dataset containing problems with different* θ *and* ξ.

ω-Jacobi. Finally, Figure 14 shows the performance of a 5-level multigrid with ω-Jacobi smoothers and α-CNN smoothers using full and red-black coarsenings with the same problem setting as in Figure 12. However, 6 convolutional layers were used with full coarsening and 2 convolutional layers with red-black coarsening. For fair comparison in terms of computational cost per iteration, in this experiment we run 6 Jacobi steps each iteration for full coarsening and 2 Jacobi steps for red-black coarsening.

6.2. Variable coefficient PDEs. We consider the following variable coefficient problem:

$$-\nabla \cdot ((\sin \kappa \pi x y + 1.1)\nabla u(x, y)) = f(x, y), \tag{6.5}$$

which is determined by the frequency κ.

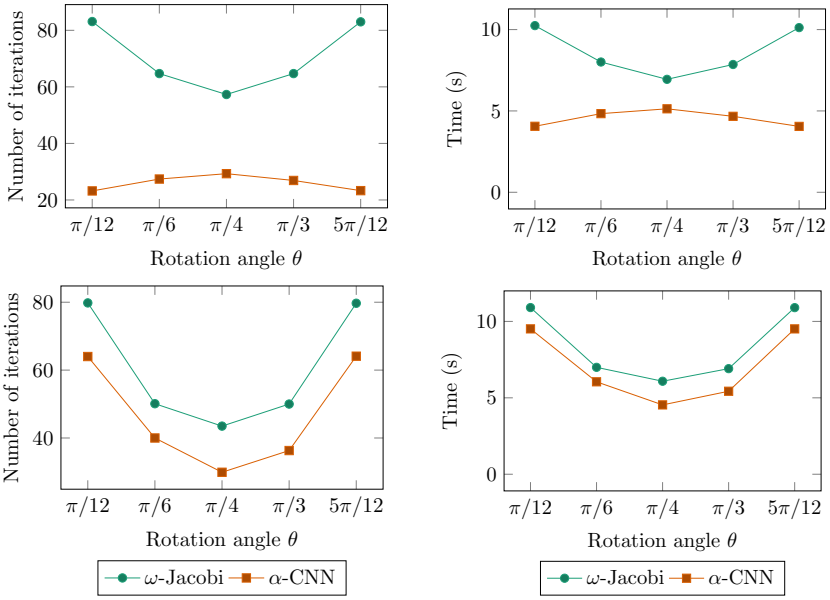


FIG. 14. *Numbers of iterations and runtime required by multigrid for solving* (6.1) *of size* $n = 511^2$ *with* $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ *and* $\xi = 100$. *The top and bottom two subfigures show the performance with full coarsening and red-black coarsening, respectively.*

TABLE 4

Numbers of iterations (runtime in seconds) required by multigrid for solving (6.5) *of size* $n = 255^2$ *with* $\kappa = 0.1, 1, 10, 100$ *using* α*-CNN and* ω*-Jacobi with* $\omega = \frac{2}{3}$.

	$\kappa = 0.1$	$\kappa = 1$	$\kappa = 10$	$\kappa = 100$
ω-Jacobi	17 (0.169)	17(0.167)	20(0.191)	63(0.500)
α-CNN-CNN	6(0.102)	7(0.114)	11(0.151)	30(0.333)
α-FC-CNN	6(0.103)	6(0.101)	10(0.142)	28(0.315)

In this experiment we consider solving the problems determined by $\kappa = 0.1, 1, 10,$ and 100. For each problem we consider a 4-level multigrid solver. We use the two approaches discussed in section 4 to learn one single convolution kernel of size 3×3 used for smoothing. We use 4 fully connected layers with 40 neurons for each layer for the first approach which has $6,800$ parameters to train in total, and we denote this approach by α-FC-CNN. We then use 3 convolutional layers which have 7, 5, and 3 channels for each layer and a fully connected layer of size 27×9 which has 378 parameters to train in total, and we denote this approach by α-CNN-CNN. We use the Leaky ReLu activation function to perform a nonlinear mapping of the stencils to the smoother. We train the smoothers on problems of size 31×31 and test the performance on problems of size 255×255.

We compare the iteration numbers and runtime of using different approaches for learning α-CNN smoothers with weighted Jacobi and show the results in Table 4. We also show the spectral properties of each smoothers in Table 5 and Table 6. We also show that the fully connected approach has similar performance in terms of both iteration number and runtime compared to the convolutional approach while having 17 times more parameters. Both α-CNN approaches can achieve $2\times$ speedup in terms of iteration number and $1.6\times$ speedup in terms of runtime.

TABLE 5
Spectral radius of iteration matrices (2.3) *of two-grid for solving* (6.5) *of size* $n = 16^2$ *using* ω*-Jacobi with* $\omega = \frac{2}{3}$*,* α*-CNN-CNN and* α*-FC-CNN smoothers.*

κ	0.1	1	10	100
ω-Jacobi	0.9951	0.9955	0.9962	0.9962
α-CNN-CNN	0.9856	0.9865	0.9888	0.9887
α-FC-CNN	0.9752	0.9762	0.9796	0.9784

TABLE 6
Ideal convergence bound (2.7) *for the same methods and problems in Table* 5*.*

κ	0.1	1	10	100
ω-Jacobi	0.9952	0.9955	0.9963	0.9962
α-CNN-CNN	0.9858	0.9867	0.9893	0.9895
α-FC-CNN	0.9753	0.9762	0.9798	0.9790

TABLE 7
Numbers of iterations (runtime(s)) required by preconditioned Flexible GMRES to reach tolerance 10^{-6} *for solving* (6.1) *with different* θ *and* ξ*. The grid size is* 511^2*.*

$\xi = 100$	$\theta = \pi/12$	$\theta = \pi/6$	$\theta = \pi/4$	$\theta = \pi/3$	$\theta = 5\pi/12$
ω-Jacobi	37.0(3.48)	30.2(2.86)	28.0(2.74)	30.0(2.65)	37.0(3.46)
α-CNN	11.0(2.32)	12.0(2.52)	13.0(2.56)	12.0(2.47)	11.0(2.29)

6.3. Incorporation with Flexible GMRES. In this section we use multigrid solvers as preconditioners of flexible GMRES (see [35]) on the same group of problems as in Figure 12. Notice that due to the use of nonlinear activation functions in the neural smoothers, it is mandatory to use flexible GMRES instead of standard GMRES as the accelerator. We compare the performance of using the α-CNN smoothers trained before and using the ω-Jacobi smoothers in terms of iteration numbers and running time. We show the results in Table 7 that using α-CNN can achieve up to $3.36\times$ improvement in terms of iteration number and up to $1.5\times$ improvement in terms of time compared to ω-Jacobi.

6.4. Comparison with other smoothers. In this section, we compare the performance of the proposed α-CNN smoothers with other smoothers. We first consider the problem (6.1) with $\theta = 0$ and $\xi = 100$. We train the α-CNN smoothers by applying 3 convolution kernels sequentially on a 31×31 mesh. We compare the performance of α-CNN smoothers with Chebyshev polynomial of degree 3 and 3 iterations of conjugate gradient and show the results on problems of various sizes in Table 8. Note that Chebyshev smoothers require estimates of spectral radius $\lambda^*_{\max}$ and are computed on interval $(\gamma_1\lambda^*_{\max}, \gamma_2\lambda^*_{\max})$. The performance of Chebyshev smoothers can be sensitive to the choice of γ_1 and γ_2. In our experiment, α-CNN smoothers performed better than the Chebyshev smoothers with $\gamma_1 = 1/30$ and $\gamma_2 = 1.1$ that are the default in PyAMG [32].

Next, we consider the following convection diffusion problem:

$$-\nu\Delta u + \vec{v} \cdot \mathbf{grad}(u) = f, \quad -\nu = 10^{-4},\ \vec{v} = [v_x, v_y] = [100, 100]. \tag{6.6}$$

TABLE 8

Number of iterations required by ω-Jacobi, α-CNN smoothers, Chebyshev smoothers, and conjugate gradient (CG) smoothers to reach the convergence tolerance 10^{-6} for solving the rotated Laplacian problems with different grid sizes, where $\theta = 0$ and $\xi = 100$.

	63^2	127^2	255^2	511^2
ω-Jacobi	568.9	599.7	593.5	576.2
Chebyshev ($m = \frac{1}{3}$,$n = 1.1$)	229.6	242.8	240.0	233.0
Chebyshev ($m = \frac{1}{30}$,$n = 1.1$)	92.0	97.1	96.0	93.0
CG	59.5	58.5	60.5	59.5
α-CNN	47.0	48.5	49.1	50.0

TABLE 9

Number of iterations required by ω-Jacobi, α-CNN, and GMRES smoothers to reach the convergence tolerance 10^{-6} for solving the convection-diffusion problem.

	63^2	127^2	255^2	511^2
ω-Jacobi	Div	Div	Div	Div
GMRES	30.4	28.1	37.7	52.5
α-CNN	12.0	11.1	14.0	17.3

We use the upwind finite difference discretization on a regular grid with uniform mesh size h in all directions [31]. The resulting stencil is the following:

$$\begin{bmatrix} & -\frac{\nu}{h^2} & \\ -\frac{\nu}{h^2} - \frac{v_x}{h} & \frac{4\nu}{h^2} + \frac{v_x+v_y}{h} & -\frac{\nu}{h^2} \\ & -\frac{\nu}{h^2} - \frac{v_y}{h} & \end{bmatrix}.$$

We train the α-CNN smoother by applying 2 convolution kernels sequentially to 31×31 problems. Since the matrix is nonsymmetric, Chebyshev smoothers cannot be used. We show the results comparing with GMRES polynomials of degree 2 in Table 9.

7. Conclusion. In this work we propose an efficient framework for training smoothers in the form of multilayered CNNs that can be equipped by multigrid methods for solving linear systems arising from PDE problems. The training process of the proposed smoothing algorithm, called α-CNN, is guided by multigrid convergence theories and has the desired property of minimizing errors that cannot be efficiently annihilated by coarse-grid corrections. Experiments on rotated Laplacian problems show the superior smoothing property of α-CNN smoothers that leads to better performance of multigrid convergence when combined with standard coarsening and interpolation schemes compared with classical relaxation-based smoothers. We also show that well-trained α-CNN smoothers on small problems can be generalized to problems of much larger sizes and different geometries without retraining. The training cost of the proposed approach is still much higher than using standard methods for solving a single PDE problem or a few of them. However, in the context of solving a large number of different problems (potentially with different grid sizes) arising from the same class of PDEs or from the same PDE operator with various right-hand sides, this high training cost can be amortized. Moreover, with the rapid development of deep learning technologies, the training time can be further reduced, and the framework will be more practical in the future. For future work, we plan to use graph convolution networks to extend the current framework to unstructured meshes and study how to optimize other components in multigrid solvers such as coarsening algorithms and grid-transfer operators.

Appendix A. Proof of equivalent conditions for convergent smoothers. The following result shows a sufficient and necessary condition for having a convergent smoother M in the energy norm.

THEOREM A.1. *Assuming A is SPD, each step of iteration* (2.3) *is convergent in the energy norm, i.e., $\|e_{k+1}\|_A = \|Ge_k\|_A < \|e_k\|_A$, if and only if $M + M^{\mathsf{T}} - A$ is SPD.*

Proof. We have the following identity,

$$\begin{aligned}\|e_{k+1}\|_A^2 &= \|(I - M^{-1}A)e_k\|_A^2 \\ &= \|e_k\|_A^2 - 2(e_k, M^{-1}Ae_k)_A + \|M^{-1}Ae_k\|_A^2 \\ &= \|e_k\|_A^2 - ((2M - A)M^{-1}Ae_k, M^{-1}Ae_k),\end{aligned}$$

so $\|e_{k+1}\|_A < \|e_k\|_A$ for any e_k if and only if $2M-A$ is positive definite or equivalently $M + M^{\mathsf{T}} - A$ is SPD. ∎

A similar result can also be shown in the 2-norm.

THEOREM A.2. *Assuming A is SPD, each step of iteration* (2.3) *is convergent in the 2-norm, i.e., $\|e_{k+1}\|_2 = \|Ge_k\|_2 < \|e_k\|_2$ for any e_k if and only if $A^{-1}M + M^{\mathsf{T}}A^{-1} - I$ is SPD.*

Proof. We have the following identity:

$$\begin{aligned}\|e_{k+1}\|_2^2 &= \|e_k\|_2^2 - 2(e_k, M^{-1}Ae_k) + \|M^{-1}Ae_k\|_2^2 \\ &= \|e_k\|_2^2 - \left((2A^{-1}M - I)M^{-1}Ae_k, M^{-1}Ae_k\right).\end{aligned}$$

So, $\|e_{k+1}\| < \|e_k\|$ if and only if $2A^{-1}M - I$ is positive definite, or equivalently $A^{-1}M + M^{\mathsf{T}}A^{-1} - I$ is SPD. ∎

Moreover, we can show that if $A^{-1}M + M^{\mathsf{T}}A^{-\mathsf{T}} - I$ is SPD, the numerical radius $\nu(G) < 1$, for which we first state the following lemma.

LEMMA A.3. *Suppose C is SPD and $(Bx, x) > (\alpha Cx, x)$ for any x and some $\alpha > 0$. Then, $B^{-1} + B^{-\mathsf{T}}$ is SPD and*

$$0 < (\alpha B^{-1}x, x) < (C^{-1}x, x). \tag{A.1}$$

Proof. Since C is SPD and $(Bx, x) > (\alpha Cx, x) > 0$, B is positive definite and so is B^{-1}, i.e., $B^{-1}+B^{-\mathsf{T}}$ is SPD. For (A.1), we have $(\alpha B^{-1}x, x) = (\alpha C^{1/2}B^{-1}x, C^{-1/2}x)$, from which and from the Cauchy–Schwarz inequality, it follows that

$$\begin{aligned}(\alpha B^{-1}x, x)^2 &\le \alpha\|C^{1/2}B^{-1}x\|_2^2\|C^{-1/2}x\|_2^2 \\ &= (\alpha CB^{-1}x, B^{-1}x)(C^{-1}x, x) \\ &< (B^{-1}x, x)(C^{-1}x, x).\end{aligned}$$

The result is given by dividing both sides by $(B^{-1}x, x)$. ∎

THEOREM A.4. *Assuming $A^{-1}M+M^{\mathsf{T}}A^{-\mathsf{T}}-I$ is SPD, then the numerical radius $\nu(G) < 1$.*

Proof. By the assumption $(A^{-1}Mx, x) > (x/2, x)$, so by Lemma A.3, we have $0 < (M^{-1}Ax, x) < 2(x, x)$, i.e., the numerical radius $\nu(G) < 1$. ∎

COROLLARY A.5. *Assuming $A^{-1}M+M^{\mathsf{T}}A^{-\mathsf{T}}-I$ is SPD, then the spectral radius $\rho(G) < 1$.*

Appendix B. Proof of convolutional network as convergent smoother.

LEMMA B.1. *Norms are continuous functions in* $\mathbb{R}^n$.

Proof. Suppose x and y are two vectors in $\mathbb{R}^n$. Define $u = x - y$, and write u in the canonical basis as $u = \sum_{i=1}^n \delta_i e_i$. Then we have

$$\|u\| = \|\sum_{i=1}^n \delta_i e_i\| \leq \sum_{i=1}^n |\delta_i| \|e_i\| \leq \max |\delta_i| \|e_i\|.$$

Setting $M = \sum_{i=1}^n \|e_i\|$ we get

$$\|u\| \leq M \max |\delta_i| = M \|x - y\|_\infty.$$

Let ϵ be given, and take x, y such that $\|x - y\|_\infty \leq \frac{\epsilon}{M}$. Then, by using the triangle inequality we obtain

$$|\|x\| - \|y\|| \leq \|x - y\| \leq M \max \delta_i \leq M \frac{\epsilon}{M} = \epsilon.$$

This means that we can make $\|y\|$ arbitrarily close to $\|x\|$ by making y close enough to x in the sense of the defined metric. Therefore, norms are continuous functions. ∎

LEMMA B.2. *In* $\mathbb{R}^n$ *all norms are equivalent.*

Proof. We only need to show that for $\Phi_1 = \|\cdot\|$ some norm and $\Phi_2 = \|\cdot\|_\infty$. All other cases will follow from this one.

First we can show that for some scalar α we have $\|x\| \leq \alpha \|x\|_\infty$. Express x in the canonical basis of $\mathbb{R}^n$ as $x = \sum_{i=1}^n x_i e_i$. Then

$$\|x\| = \|\sum_{i=1}^n x_i e_i\| \leq \sum_{i=1}^n |x_i| \|e_i\| \leq \max |x_i| \sum_{i=1}^n \|e_i\| \leq \|x\|_\infty \alpha,$$

where $\alpha = \sum_{i=1}^n \|e_i\|$.

We then show that there is a β such that $\|x\| \geq \beta \|x\|_\infty$. Assume $x \neq 0$, and consider $u = x/\|x\|_\infty$. Note that u has infinity norm equal to one. Therefore it belongs to the closed and bounded set $S_\infty = \{v | \|v\|_\infty = 1\}$. Based on Lemma B.1, the minimum of the norm $\|u\|$ for all u's is reachable in the sense that there is a $u_0 \in S_\infty$ such that

$$\min_{u \in S_\infty} \|u\| = \|u_0\|.$$

Let us call β this minimum value. Since this value cannot be zero, we then have

$$\|\frac{x}{\|x\|_\infty}\| \geq \beta.$$

This implies that $\|x\| \geq \beta \|x\|_\infty$. This completes the proof. ∎

THEOREM B.3. *Suppose* H *is a convolutional network with* k *convolutional layers. For one fixed matrix* A*, if parameterized properly,* H *can be a convergent smoother.*

Proof. Based on the universality property of deep CNNs without fully connected layers [47], we know that H can approximate the linear operator A^{-1} to an arbitrary accuracy measured by some norms when k is large enough. Thus, theoretically, HA can be very close to an identity mapping if parameterized properly. Based on Lemmas B.1 and B.2, we know matrix induced norms are continuous functions; thus $\|I - HA\|_A$ can be less than 1 for certain k measured in matrix A-norm. ∎

Appendix C. Finite difference discretization of PDEs. We discretize the operators Δu and u_{xy} in (6.1) using the following stencils:

$$\frac{1}{4h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{2h^2}\begin{bmatrix} & -1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & \end{bmatrix},$$

where h is the grid spacing.

Appendix D. Numerical results of spectral radius. We fix the parameters of the rotated Laplacian problems to be $\theta = 0$, and we train the neural smoothers for $\xi = 100, 200, 300, 400$ on the square domain. Then we fix the parameters to be

TABLE 10

Spectral radius of the iteration matrices corresponding to the 5-level multigrid methods with full coarsening and ω-Jacobi and 6-layered α-CNN smoother for rotated Laplacian problems with different θ and ξ. The mesh size is 16×16.

$\theta = 0$	$\xi = 100$	$\xi = 200$	$\xi = 300$	$\xi = 400$
ω-Jacobi	0.9853	0.9918	0.9940	0.9951
α-CNN	0.6816	0.8189	0.8805	0.8936

$\xi = 100$	$\theta = 0$	$\theta = \pi/12$	$\theta = \pi/6$	$\theta = \pi/4$
ω-Jacobi	0.9853	0.9436	0.8981	0.8837
α-CNN	0.6816	0.4534	0.4547	0.4216

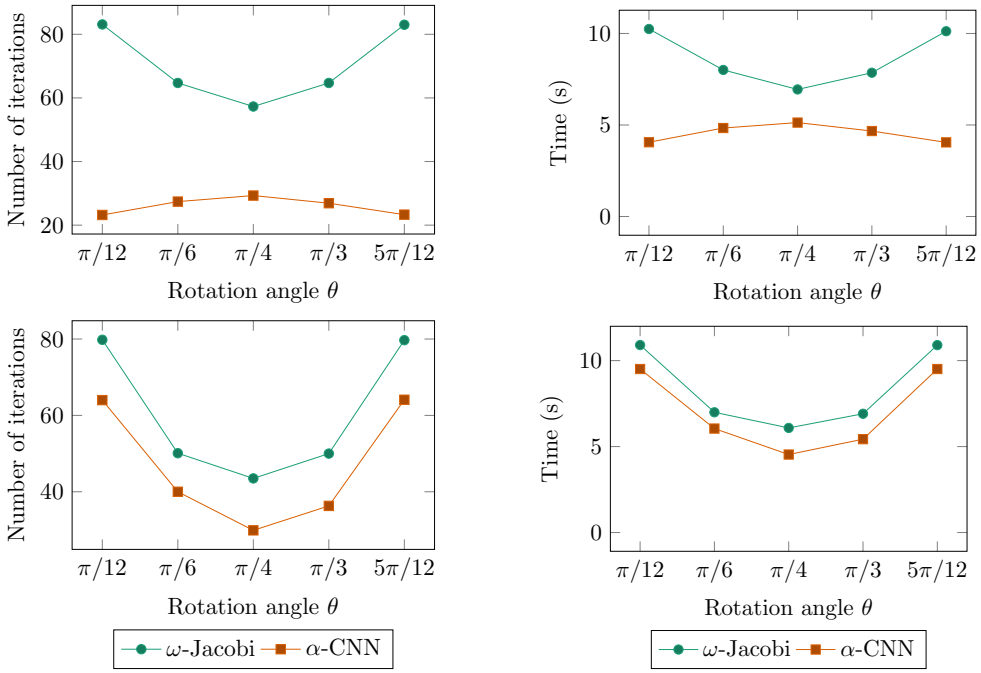


FIG. 15. *Numbers of iterations and runtime required by multigrid solvers for solving the rotated Laplacian problems of size $n = 511^2$ with $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ and $\xi = 100$. The top two subfigures show the performance of multigrid with full coarsening, and the bottom two subfigures show the performance of multigrid with red-black coarsening.*

$\xi = 100$, and we train the neural smoothers for $\theta = 0, \ldots, \frac{\pi}{4}$. We compare the spectral radius of the iteration matrices of 5-level multigrid solvers equipped with different smoothers and summarize the results in Table 10.

Appendix E. More numerical results.

Figure 15 shows the performance of 5-level multigrid with ω-Jacobi smoothers and α-CNN smoothers using full coarsening and red-black coarsening with the same problem setting as in Figure 12. However, here we are using 6 convolutional layers with full coarsening and 2 convolutional layers with red-black coarsening. For fair comparison in terms of computational flops per iteration, in this experiment we run 6 Jacobi steps each iteration for full coarsening and 2 Jacobi steps for red-black coarsening.

Acknowledgments. We would like to acknowledge the fruitful discussions with the *hypre* team at Lawrence Livermore National Laboratory, which prompted the exploration of interpretability of the learned smoothers in section 5.

REFERENCES

[1] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, *Multigrid smoothers for ultraparallel computing*, SIAM J. Sci. Comput., 33 (2011), pp. 2864–2887.

[2] R. E. Bank and C. C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM J. Numer. Anal., 22 (1985), pp. 617–633.

[3] M. Benzi and M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Anal., 30 (1999), pp. 305–340.

[4] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing, 317 (2018), pp. 28–41.

[5] M. Bolten and K. Kahl, *Using block smoothers in multigrid methods*, PAMM. Proc. Appl. Math. Mech., 12 (2012), pp. 645–646.

[6] A. Brandt, *Algebraic multigrid (AMG) for sparse matrix eqations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, UK, 1984, pp. 257–284.

[7] A. Brandt, *Algebraic multigrid theory: The symmetric case*, Appl. Math. Comput., 19 (1986), pp. 23–56.

[8] A. Brandt, S. McCormick, and J. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, UK, 1985, pp. 257–284.

[9] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM J. Sci. Comput., 22 (2001), pp. 1570–1592.

[10] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM, Philadelphia, 2000.

[11] D. Cai, E. Chow, L. Erlandson, Y. Saad, and Y. Xi, *SMASH: Structured matrix approximation by separation and hierarchy*, Numer. Linear Algebra Appl., 25 (2018), e2204.

[12] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, *Multi-level residual networks from dynamical systems view*, in Proceedings of the International Conference on Learning Representations, 2018.

[13] J. Cosgrove, J. Diaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems*, Int. J. Comput. Math., 44 (1992), pp. 91–110.

[14] H. De Sterck, T. A. Manteuffel, S. F. McCormick, K. Miller, J. Ruge, and G. Sanders, *Algebraic multigrid for markov chains*, SIAM J. Sci. Comput., 32 (2010), pp. 544–562.

[15] D. J. Evans and W. S. Yousif, *The explicit block relaxation method as a grid smoother in the multigrid v-cycle scheme*, Int. J. Comput. Math., 34 (1990), pp. 71–78.

[16] R. D. Falgout and P. S. Vassilevski, *On generalizing the algebraic multigrid framework*, SIAM J. Numer. Anal., 42 (2004), pp. 1669–1693.

[17] D. Greenfeld, M. Galun, R. Basri, I. Yavneh, and R. Kimmel, *Learning to optimize multigrid PDE solvers*, in Proceedings of the International Conference on Machine Learning, 2019, pp. 2415–2423.

[18] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[19] T. Gudmundsson, C. S. Kenney, and A. J. Laub, *Small-sample statistical estimates for matrix norms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 776–792.
[20] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, *Learning across scales—multiscale methods for convolution neural networks*, in Proceedings of the AAAI Conference on Artificial Intelligence, 2018.
[21] J. Han, A. Jentzen, and E. Weinan, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510.
[22] J. He and J. Xu, *MgNet: A unified framework of multigrid and convolutional neural network*, Sci. China Math., 62 (2019), pp. 1331–1354.
[23] P. Holl, N. Thuerey, and V. Koltun, *Learning to control PDEs with differentiable physics*, in Proceedings of the International Conference on Learning Representations, 2019.
[24] J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, and S. Ermon, *Learning neural PDE solvers with convergence guarantees*, in Proceedings of the International Conference on Learning Representations, 2019.
[25] A. Katrutsa, T. Daulbaev, and I. Oseledets, *Deep Multigrid: Learning Prolongation and Restriction Matrices*, preprint, arXiv:1711.03825 [math.NA], 2017.
[26] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw., 9 (1998), pp. 987–1000.
[27] P. T. Lin, J. N. Shadid, and P. H. Tsuji, *Krylov Smoothing for Fully-Coupled AMG Preconditioners for VMS Resistive MHD*, Springer International Publishing, Cham, 2020, pp. 277–286.
[28] I. Luz, M. Galun, H. Maron, R. Basri, and I. Yavneh, *Learning algebraic multigrid using graph neural networks*, in Proceedings of the International Conference on Machine Learning, 2020, pp. 6489–6499.
[29] S. Mishra, *A machine learning framework for data driven acceleration of computations of differential equations*, Math. Engrg., 1 (2018), pp. 118–146.
[30] E. Nathan, G. Sanders, V. E. Henson, and D. A. Bader, *Numerically approximating centrality for graph ranking guarantees*, J. Comput. Sci., 26 (2018), pp. 205–216.
[31] Y. Notay, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM J. Comput. Sci., 34 (2012), pp. A2288–A2316.
[32] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic Multigrid Solvers in Python v*4.0, 2018, https://github.com/pyamg/pyamg.
[33] O. Ronneberger, P.Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in Medical Image Computing and Computer-Assisted Intervention (MICCAI), Lecture Notes in Comput. Sci. 9351, Springer, 2015, pp. 234–241.
[34] J. W. Ruge, *Algebraic multigrid (AMG) for geodetic survey problems*, in Preliminary Proceedings of the International Multigrid Conference, Fort Collins, CO, 1983.
[35] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[36] Y. Saad, *Iterative Methods for Linear Systems of Equations: A Brief Historical Journey*, preprint, arXiv:1908.01083 [math.HO], 2020.
[37] J. Schmitt, S. Kuckuk, and H. Köstler, *Optimizing Geometric Multigrid Methods with Evolutionary Computation*, preprint, arXiv:1910.02749 [math.NA], 2019.
[38] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.
[39] H. D. Sterck, V. E. Henson, and G. Sanders, *Multilevel aggregation methods for small-world graphs with application to random-walk ranking*, Comput. Informatics, 30 (2011), pp. 225–246.
[40] M. Sun, X. Yan, and R. Sclabassi, *Solving partial differential equations in real-time using artificial neural network signal processing as an alternative to finite-element analysis*, in Proceedings of the International Conference on Neural Networks and Signal Processing, IEEE, 2003, pp. 381–384.
[41] W. Tang, T. Shan, X. Dang, M. Li, F. Yang, S. Xu, and J. Wu, *Study on a Poisson's equation solver based on deep learning technique*, in Proceedings of the IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), IEEE, 2017, pp. 1–3.
[42] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*, Elsevier, New York, 2000.
[43] A. J. Wathen, *Preconditioning*, Acta Numer., 24 (2015), pp. 329–376.
[44] S. Wei, X. Jin, and H. Li, *General solutions for nonlinear differential equations: A rule-based self-learning approach using deep reinforcement learning*, Comput. Mech., 64 (2019), pp. 1361–1374.

[45] G. Wittum, *On the robustness of ILU smoothing*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 699–717.
[46] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numer., 26 (2017), p. 591–721.
[47] D.-X. Zhou, *Universality of deep convolutional neural networks*, Appl. Comput. Harmon. Anal., 48 (2020), pp. 787–794.