Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Ru Huang Date

Improving Multigrid Methods with Deep Neural Networks

By

Ru Huang

Doctor of Philosophy

Mathematics

Yuanzhe Xi

Advisor

James G. Nagy

Committee Member

Lars Ruthotto

Committee Member

Ruipeng Li

Committee Member

Accepted:

Kimberly Jacob Arriola, Ph.D., MPH

Dean of the James T. Laney School of Graduate Studies

Date

Improving Multigrid Methods with Deep Neural Networks

By

Ru Huang
B.S., Fudan University (P. R. China), 2016

Advisor: Yuanzhe Xi, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Mathematics
2022

Abstract

Improving Multigrid Methods with Deep Neural Networks

By Ru Huang

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. There are two key components of multigrid, *smoothing* which aims at reducing high-frequency errors on each grid level and *coarse grid correction* which interpolates the solution at the coarse grid. However, finding optimal smoothing algorithms is problem-dependent and can impose challenges for many problems. Meanwhile, as the multigrid hierarchy is formed, coarse-grid operators have significantly more nonzeros per row than the original fine-grid operator, which generates high parallel communication costs on coarse-levels. In this thesis, I first propose an efficient adaptive framework for learning optimal 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. I also propose a deep learning framework for sparsifying coarse grid operators. Two neural networks are constructed to learn the sparsity pattern and the corresponding values, respectively. The learned sparser operator has the same interpolation accuracy on algebraic smooth basis. Numerical results on challenging anisotropic rotated Laplacian problems, variable coefficient diffusion problems and linear elasticity problems demonstrate the superior performance of the proposed framework over classical hand-crafted methods.

Improving Multigrid Methods with Deep Neural Networks

By

Ru Huang
B.S., Fudan University (P. R. China), 2016

Advisor: Yuanzhe Xi, Ph.D.

A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Mathematics
2022

Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. Yuanzhe Xi. In academia, Yuanzhe inspired me with new research ideas, supported my Ph.D study and related research, helped me with the writing of this thesis. Yuanzhe has also given me lots of opportunities to connect with experts in academy and he always encouraged me to give talks which makes me more well-rounded as a Ph.D student. I could not have imagined completing my degree without him.

I also want to show my great thankfulness to Dr. Ruipeng Li for his continuous support for the related research, for his patience and motivation, for his expertise which has improved the quality of this thesis.

I would like to thank the rest of my thesis committee: Prof. James Nagy and Prof. Lars Ruthotto for their insightful comments and suggestions which helped me to widen my research from various perspectives.

In addition, I would like to express my appreciation to Prof. Michele Benzi who guided me at the beginning of my Ph.D career. His guidance helped me in all the time of research and Ph.D life which leads me to the right track of the career path.

Finally, I would like to thank my friends and family for supporting me in life in general, for encouraging me when I encounter difficulties, for listening to all my complaints.

Contents

List of Figures

List of Tables

Chapter 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 that need to be solved. In machine learning applications such as spectral clustering, graph-based semi-supervised learning, Markov chains and transportation network flows, solving large-scale linear systems associated with graph Laplacians is often needed [18, 39, 49]. Hence, the development of efficient linear solvers remains an active research area [16, 45, 56].

Among many numerical solution schemes, multigrid methods often show superior efficiency and scalability especially for solving elliptic-type PDE and graph Laplacian problems [9, 15, 43, 54]. 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. On the other hand, the time complexity of multigrid methods depends not only on convergence rate but also on the computa-

tional cost performed at each step. Multigrid methods suffer from increasing density on coarse grids. Although coarse grids are smaller in size and have fewer number of nonzero entries, the increasing number of nonzeros per row is the bottleneck in parallel computing.

1.1 Related work

There is 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, [34] first proposed to use neural networks (NNs) to approximate the solutions for both Ordinary Differential Equations (ODEs) and PDEs with a fixed boundary condition. Later, [51] utilized Convolutional Neural Networks (CNNs) to solve Poisson equations with a simple geometry and [4] extended the techniques to more complex geometries. [28,48] applied machine learning techniques to solve high dimensional PDEs, and [57] focused on applying reinforcement learning to solve nonlinear PDEs. [50] used parameterized realistic volume conduction models to solve Poisson equations and [30] 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, [47] developed optimization techniques for geometric multigrid based on evolutionary computation. [38] generalized existing numerical methods as NNs with a set of trainable parameters. [33] proposed a deep learning method to optimize the parameters of prolongation and restriction matrices (introduced in Section 3.4) in a two-grid geometric multigrid scheme by minimizing the spectral radius of the iteration matrix. [25] used NNs to learn prolongation matrices in multigrid in order to solve diffusion equations

without retraining and [37] 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 NN architectures. For instance, [29] designed MgNet which uses multigrid techniques to improve CNNs. [17, 27] scaled up CNNs by interpreting the forward propagation as nonlinear PDEs.

Here, we would like to highlight the work by Hsieh [31], which proposes to use CNNs and U-net [42] to learn a correction term to 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 thesis.

The problem of increasing density on coarse grids was first addressed in [19, 24]. [53, 58] control the coarse grid sparsity pattern explicitly by exploring the approximations of the fine grid operator through algebraically smooth basis vectors. [22, 52] sparsified the coarse grid operator directly via first determining the sparsity patterns heuristically and then computing the values based on spectral equivalence between coarse grid operator and its sparse version. [5] proposed new algorithms for reducing communications.

Sparsification using deep learning has been well studied in graph learning [36, 61]. Graph sparsifications focus on only dropping edges in graph while not modifying the values. The goal of graph sparsification is usually to reduce noise in the graph and lets the graph data less sensitive to the subsequent sparse approximation operations. On contrary, Multigrid sparsification requires not only to determine the sparsity pattern but also compute the optimal values. Spectral equivalence is often used the key metric in multigrid sparsification to preserve the convergence.

1.2 Contributions of Work

Multigrid methods have two key components, the smoothers and the coarse grid operator. We leverage deep learning techniques to improve the efficiency of multigrid methods by

- developing smoothers which have better convergence, and
- controlling the sparsity pattern in multigrid hierarchy to reduce cost while not influencing convergence.

In particular, we first 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 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 of generalization to problems of much larger sizes and more complex geometries. We then propose a deep learning framework for sparsifying the coarse grid operator in multigrid hierarchy while not influencing the convergence to improve the parallel efficiency in practice. We utilize two deep neural networks to learn the desired sparsity pattern and the values respectively. We exploit sparsification on algebraic smooth basis to train spectral equivalent sparse stencils. Numerical results show that the proposed model can be generalized to problems of much larger sizes and problems with different parameters. To the best of our knowledge, our work

is the first attempt to use CNNs to learn the smoother at each level of multigrid with more than two levels and the first to use deep learning in multigrid sparsification which exhibits good generalization properties to problems with different sizes, geometries and variable coefficients.

1.3 Outline of Thesis

The thesis is organized as follows. In Chapter 2, we review the finite difference discretization scheme for numerically solving partial differential equations (PDEs). Several iterative methods for solving sparse linear systems arise from discretized PDEs, including relaxation methods, polynomial based methods, GMRES, and multigrid (which is the focus of this thesis), are introduced in Chapter 3. In Chapter 4, we describe an efficient adaptive framework for learning smoothers using CNNs. In Chapter 5 we move further to use a deep learning framework to sparsify the coarse grid operator. In both Chapter 4 and Chapter 5 we mainly deal with stencil-based problems. Conclusions and future work are described in Chapter 6.

Chapter 2

Background on PDEs

Partial Differential Equations (PDEs) are widely used to model real world problems that are related to physical phenomena. However, they often don't have closed-form solutions and require numerical methods to obtain approximate solutions. In this chapter, we provide a quick review of Poisson's equation (see Section 2.1) and the finite difference discretization scheme (see Section 2.2) for numerically solving PDEs [44]. We also use Poisson's equation as an example for illustrating the use of finite difference on PDEs.

2.1 Poisson's equation

Poisson's equation has the following form

$$-\Delta u(\vec{x}) = f(\vec{x}), \quad \vec{x} \in \Omega,$$

where Ω is a bounded domain, and Δ is the Laplace operator defined as the sum of second order partial derivatives of u with respect to each independent variable. Poisson's equation is a generalization of Laplace's equation, which can be obtained from Poisson's equation by setting the right hand side function $f = 0$.

In the one dimensional case where Ω is an interval, Poisson's equation can be written as

$$-u''(x) = f(x), \quad x \in (a, b).$$

In the two dimensional case where Ω is an area on the 2D plane, Poisson's equation can be written as

$$-(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}) = f(x_1, x_2), \quad (x_1, x_2) \in \Omega.$$

Three types of boundary conditions are often considered with Poisson's equation:

1. Dirichlet condition:
$$u(x) = \phi(x);$$

2. Neumann condition:
$$\frac{\partial u}{\partial \overrightarrow{n}}(x) = 0;$$

3. Cauchy condition:
$$\frac{\partial u}{\partial \overrightarrow{n}}(x) + \alpha(x)u(x) = \gamma(x),$$

where $\overrightarrow{n}$ is the outward-pointing unit vector normal to the boundary surface $\partial\Omega$.

2.2 Finite difference discretization

A popular method for the discretization of PDEs is the finite difference method, which is based on local approximations of the partial derivatives derived from Taylor series expansions. In this subsection, we briefly review a few finite difference schemes.

We first consider the one-dimensional problem with Dirichlet boundary condition:

$$\begin{aligned} -u''(x) &= f(x), \quad x \in (0,1) \\ u(0) &= u(1) = 0 \end{aligned} \tag{2.1}$$

with one space variable x on the interval $(0,1)$. The first derivative of the function u at the point x can be approximated by the function values at points that are closed to x (i.e., $x + \delta x$ and $x - \delta x$) via the following approximation formulae,

- Forward difference:

$$(\frac{du}{dx})(x) \approx \frac{u(x+\delta x) - u(x)}{\delta x}, \tag{2.2}$$

- Backward difference:

$$(\frac{du}{dx})(x) \approx \frac{u(x) - u(x-\delta x)}{\delta x}, \tag{2.3}$$

- Centered difference

$$(\frac{du}{dx})(x) \approx \frac{u(x+\delta x) - u(x-\delta x)}{2\delta x}. \tag{2.4}$$

Similarly, the second derivative of u can be approximated by the following centered difference formula:

$$(\frac{d^2u}{dx^2})(x) \approx \frac{u(x+\delta x) - 2u(x) + u(x-\delta x)}{(\delta x)^2}.$$

If the difference δx is small enough, the approximations of the derivatives are close to the ground truth values. It can be shown that forward difference scheme (2.2) and backward difference scheme (2.3) are first order accurate whose errors are $O(\delta x)$ while centered difference scheme is second order accurate whose error is $O((\delta x)^2)$.

When solving the Equation (2.1) numerically, one usually seeks to approximate the solution at a set of discrete points (known as mesh points) in the solution domain. For example, a common discretization for the one-dimensional problem is to discretize the interval $(0, 1)$ uniformly with constant mesh size h which results in $(n+2)$ points with

$$x_i = ih, \quad i = 0, \ldots, n+1,$$

so that $x_0 = 0$, $x_{n+1} = 1$, and $n = \frac{1}{h} - 1$. From now on, we use $\tilde{u}_i$ to denote the approximate value of u at x_i, i.e., $\tilde{u}_i \approx u(x_i)$, and f_i to denote $f(x_i)$.

When the centered difference formula is used to approximate the second derivative in Equation (2.1), the $\tilde{u}_i$'s at three consecutive mesh points satisfy the following equation:

$$-\tilde{u}_{i-1} + 2\tilde{u}_i - \tilde{u}_{i+1} = h^2 f_i.$$

Since $\tilde{u}_0$ and $\tilde{u}_{n+1}$ are determined by the boundary condition and are zero in this case, the rest approximations at the n points can be obtained by solving the following linear system

$$A\tilde{u} = f,$$

where

$$A \in \mathbb{R}^{n\times n} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}, \quad \tilde{u} = \begin{bmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{bmatrix}, \quad \text{and} \quad f = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

The coefficients describing the relationships between the values of function u at different grid points can be represented concisely with stencils. The corresponding stencil of the above linear system is given in Figure 2.1. Note that stencils usually

represent the non-zero entries of each row in the coefficient matrix A except for the rows that correspond to the boundary points.

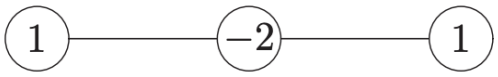


Figure 2.1: The 3-point stencil of second derivative approximation using centered difference formula

The two-dimensional Possion's equation with the Dirichlet boundary condition has the form:

$$-(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}) = f(x_1, x_2), \quad (x_1, x_2) \in \Omega = (0,1) \times (0,1)$$
$$u(x_1, x_2) = 0, \quad (x_1, x_2) \in \partial\Omega, \tag{2.5}$$

where two space variables x_1 and x_2 are defined on a domain Ω.

Suppose the same mesh size h is used along both directions to discretize the problem and the centered difference formula is applied to approximate second order derivatives $\frac{\partial^2}{\partial x_1^2}$ and $\frac{\partial^2}{\partial x_2^2}$. Let $\tilde{u}_{i,j}$ denote the approximate value of $u(x_{1i}, x_{2j})$ and $f_{i,j} = f(x_{1i}, x_{2j})$, where

$$x_{1i} = ih, \quad x_{2j} = jh, \quad \text{for} \quad i, j = 0, \ldots, n+1,$$

with $n = \frac{1}{h} - 1$. Then the $\tilde{u}_{i.j}$'s at consecutive mesh points satisfy the following equations:

$$\tilde{u}_{0,j} = \tilde{u}_{n+1,j} = \tilde{u}_{i,0} = \tilde{u}_{i,n+1} = 0 \quad , \quad \text{for} \quad i, j = 0, \ldots n+1$$
$$-(\tilde{u}_{i+1,j} + \tilde{u}_{i,j+1} - 4\tilde{u}_{i,j} + \tilde{u}_{i,j-1} + \tilde{u}_{i-1,j}) = h^2 f_{i,j} \quad , \quad \text{for} \quad i, j = 1, \ldots n.$$

This relationship can be represented with the following linear system:

$$A\tilde{u} = f, \tag{2.6}$$

where

$$A = \frac{1}{h^2}\begin{bmatrix} B & -I & 0 & \cdots & 0 \\ -I & B & -I & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -I & B & -I \\ 0 & \cdots & 0 & -I & B \end{bmatrix},$$

with B and $I \in \mathbb{R}^{n\times n}$, and

$$B = \begin{bmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{bmatrix}.$$

The above linear system corresponds to the 5-point stencils given in Figure 2.2.

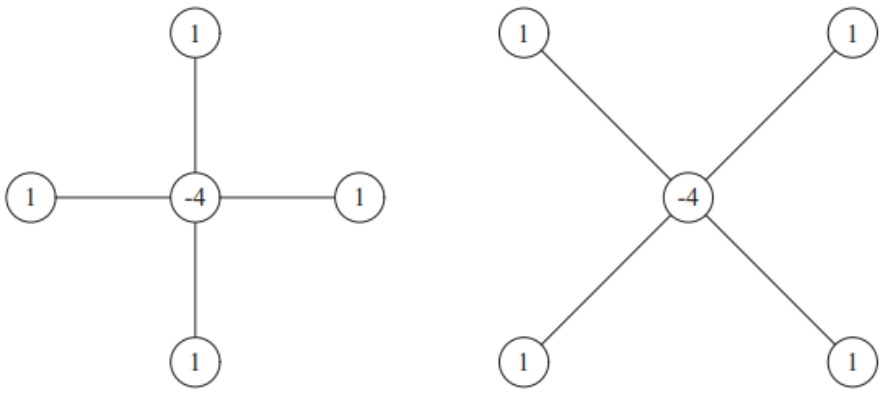


Figure 2.2: Left: the standard five-point stencil Right: the skewed five-point stencil

Chapter 3

Iterative methods for PDEs

In this chapter, we review several iterative methods for solving linear systems that arise in the discretization of PDEs. We start with the introduction of some basic iterative methods, namely, relaxation methods (Section 3.1), polynomial based methods (Section 3.2), and GMRES (Section 3.3). Their convergence properties are also introduced. We then describe multigrid methods in Section 3.4, which are the main focus of this thesis. Convergence theories of multigrid methods will also be discussed in detail as they provide the theoretical guidance for the design of the efficient algorithms proposed in this thesis.

3.1 Relaxation methods

In this section, we review the relaxation methods for solving the linear system

$$Au = f, \tag{3.1}$$

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 (3.1), in which the approximate solution u_k at iteration k depends on the previous ones.

Formally, an iterative solver can be expressed as:

$$u_k = \Phi(u_0, f, k), \tag{3.2}$$

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. Iterations based on relaxation schemes can be written as

$$\begin{aligned} u_{k+1} &= (I - M^{-1}A)u_k + M^{-1}f \\ &= Gu_k + M^{-1}f, \quad G = I - M^{-1}A, \end{aligned} \tag{3.3}$$

where M is the relaxation matrix and G is the iteration matrix.

Consider the splitting of A so that $A = D + L + U$, where

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad \text{and} \quad L + U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}.$$

Standard relaxation approaches include the weighted Jacobi method which follows

$$M = \omega^{-1}D,$$

where D is the diagonal component of A and ω is a weight between 0 and 1, and the Gauss-Seidel method, for which

$$M = D - L,$$

where L is the strict lower triangular part of A.

Consider the discretization of Poisson's equation (2.5) with the standard five point stencil shown in Figure 2.2 on a regular grid. The iteration matrix G of the weighted Jacobi method and Gauss-Seidel method in this case both correspond to the following stencil:

$$\begin{bmatrix} & \frac{\omega}{4} & \\ \frac{\omega}{4} & & \frac{\omega}{4} \\ & \frac{\omega}{4} & \end{bmatrix}.$$

However, weighted Jacobi method applies the stencil simultaneously for each grid point and therefore can be implemented in parallel while Gauss-Seidel method applies this stencil sequentially. This is because the point is updated using the updated values of the processed points in Gauss-Seidel method and therefore Gauss-Seidel has less parallel efficiency than Jacobi.

Denoting by $e_k = u_* - u_k$ the error at iteration k, where u_* is the exact solution of (3.1), it follows that $e_k = G^k e_0$. Theorem 3.1.1 gives a general convergence result for $\lim_{k\to\infty} e_k = 0$.

Theorem 3.1.1 ([44, Theorem 4.1])**.** *Denote by $\rho(G)$ the spectral radius of G. The iteration* (3.3) *converges for any initial vector u_0 if and only if $\rho(G) < 1$.*

In particular, the weighted Jacobi method converges if

$$0 < \omega < 2/\rho(D^{-1}A).$$

Notice that $\rho(G)$ represents the asymptotic convergence rate, which, however, does not predict error reduction for a few iterations [15] in general. When relaxation methods are used as multigrid smoothers, they are typically applied $O(1)$ times in each smoothing step. Thus, a class of relaxation methods called *convergent smoothers* are often used in multigrid methods in order to guarantee a fast convergence.

Definition (Convergent smoother in energy norm)**.** Assuming A is SPD, relaxation

matrix M is called a *convergent smoother* in the energy norm if $\|Ge_k\|_A < \|e_k\|_A, \forall e_k$, where $G = I - M^{-1}A$ and $\|x\|_A^2 = x^\mathsf{T} Ax$.

Theorem 3.1.2 shows sufficient and necessary conditions for having a convergent smoother M in the energy norm and 2-norm.

Theorem 3.1.2. *Assuming A is SPD, each step of iteration* (3.3) *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. Each step of iteration* (3.3) *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.*

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 for a certain class of PDE problems (e.g. elliptical PDEs), they are known to be very efficient for smoothing the error. 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 Section 3.4.

3.2 Polynomial based methods

In this section we review the polynomial based methods for solving linear systems

$$Au = f,$$

where A is a matrix of size $n \times n$. The solution u can be approximated by

$$A^{-1}f \approx p_m(A)f$$

where p_m is a polynomial of degree m.

When the coefficient matrix A is SPD, the polynomial p_m can be constructed by solving the following optimization problem:

$$\min_{p} \max_{\lambda \in \sigma(A)} |1 - \lambda p(\lambda)|,$$

where $\sigma(A)$ is the spectrum of A. The above optimization problem requires knowing all the eigenvalues of A, which is unrealistic for most applications. Therefore, a more viable approach is to solve the above optimization problem on a continuous interval (a, b) that contains the spectrum of A. This leads to the following optimization problem:

$$\min_{s} \max_{\lambda \in (a,b)} |1 - \lambda p(\lambda)|. \tag{3.4}$$

The solution to (3.4) is the polynomial

$$T_k = \frac{1}{\sigma_k} C_k\left(\frac{b + a - 2t}{b - a}\right), \tag{3.5}$$

with

$$\sigma_k = C_k\left(\frac{b + a}{b - a}\right),$$

where C_k is the Chebyshev polynomial of the first kind of degree k defined as

$$C_k(t) = \cos\left[k \cos^{-1}(t)\right], \quad \text{for} \quad -1 \leq t \leq 1.$$

Chebyshev polynomials of the first kind can be efficiently computed through three term recurrence. Letting $\theta = \frac{a+b}{2}$ and $\delta = \frac{b-a}{2}$, the solution (3.5) to the equation

(3.4) can be computed as

$$\rho_k = \frac{1}{2\sigma_1 - \rho_{k-1}},$$
$$T_{k+1}(t) = \rho_k[2(\sigma_1 - \frac{t}{\delta})T_k(t) - \rho_{k-1}T_{k-1}(t)], \quad k \geq 1$$

with initial conditions:

$$\sigma_1 = \frac{\theta}{\delta}, \quad \sigma_0 = 1, \quad T_1(t) = 1 - \frac{t}{\theta}, \quad T_0(t) = 1.$$

Since the Chebyshev polynomial approximation $p_m(A)$ does not require the computation of inner products when being applied to vectors, they can be efficiently implemented on modern heterogeneous computing architectures. The drawbacks of this polynomial approximation is that their performance is very sensitive to the estimation of the spectrum and might have very undesirable performance if hyperparameters are not chosen correctly.

3.3 GMRES

When the coefficient matrix A is nonsymmetric or indefinite, Generalized Minimum Residual Method (GMRES) method [46] is a popular method for solving the corresponding linear system. Given an initial guess u_0, GMRES first computes the initial residual $r_0 = f - Au_0$. Let $v = \frac{r_0}{\|r_0\|}$. At the mth iteration, GMRES computes an approximation

$$u_m \in u_0 + \mathcal{K}_m(A, v),$$

which minimizes

$$L(y) = \|f - Au\|_2 = \|f - A(u_0 + V_m y)\|_2$$

over the Krylov subspace

$$\mathcal{K}_m(A, v) \equiv \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}.$$

Here

$$V_m = [v_0, v_1, \ldots, v_{m-1}]$$

denotes the orthonormal basis of the Krylov subspace $\mathcal{K}_m$.

Since the approximate solution u_m returned by GMRES can be written as $u = u_0 + p_m(A)v$ for some polynomial p_m, this shows that GMRES also implicitly defines a polynomial approximation for A^{-1}. Both Chebyshev polynomial and GMRES can be used as efficient smoothers in multigrid methods for solving large scale linear systems that are discretized from PDEs.

3.4 Multigrid methods

Multigrid methods exploit a hierarchy of grids with exponentially decreasing numbers of degrees of freedom on coarser levels to speed up the convergence of simple iterative methods. Since the computational cost is proportional to the problem size on each level, the overall complexity of multigrid methods can be kept linear for solving certain elliptic PDEs.

Smoothing and coarse-grid correction are the two main components of multigrid, which are designed to be complementary to each other in order to achieve fast convergence, i.e., they aim at eliminating "high-frequency" (oscillatory) and "low-frequency" (smooth) errors respectively, where high- and low-frequency errors 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 because these methods are inexpensive to apply and can ef-

fectively remove high-frequency errors for elliptic type PDEs. On the other hand, coarse-grid correction on low-frequency errors is effective since smooth errors can be interpolated accurately. For instance, interpolation operators in standard AMG can interpolate constant vectors exactly, and methods like smooth aggregation AMG construct interpolation matrices from the smoothed errors directly. When dealing with hard problems such as those with irregular anisotropy, anisotropy not aligned along the coordinate axes, or complex geometries, efficiency of traditional smoothers can deteriorate, in which cases, stronger and often more expensive smoothers are needed such as block smoothers [8, 21], ILU-based smoothers [59], hierarchical matrix based smoothers [16, 20] and smoothers based on Krylov methods [3, 35]. Nevertheless, finding robust and efficient smoothers still remains a challenging problem for multigrid.

There are many different ways to construct multigrid mesh hierarchy, depending on specific problems. Taking the full coarsening scheme as an example, which is illustrated in Figure 3.1 for 2D grids. Given a fine level grid, the coarse level grid is constructed by selecting the black grid points with equal space on both x and y dimension, the rest red grid points will be coarsened and the value will be aggregated to the black points. The mesh size on the coarse level grid is doubled and the problem size of the linear system is reduced by a factor of four.

Next, we review the basic ingredients in multigrid methods and relevant convergence theories [44].

3.4.1 Prolongation

A prolongation operator takes a vector on the coarse level grid Ω_H with mesh size H and transforms it to a vector on the fine grid Ω_h with a smaller mesh size h. One common idea to define a prolongation operator is through linear interpolation:

$$I_H^h : \Omega_H \to \Omega_h.$$

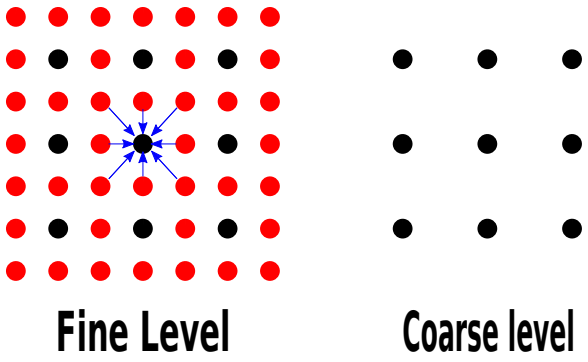


Figure 3.1: Full coarsening. The fine points are red and coarse points are black. Full weighting restriction is shown by the arrows to the coarse point at the center.

We first consider the one-dimensional case where we have $n+2$ discretization points $x_0, x_1, \ldots, x_{n+1}$ that are equally spaced on the fine grid where x_0 and x_{n+1} are two boundary points and the mesh size is h. Assume n is odd so that the coarse grid points are $x_0, x_2, \ldots, x_{n-1}, x_{n+1}$ with mesh size $2h$. Then, given a coarse grid vector $(v_i^{2h})_{i=0,\ldots,(n+1)/2} \in \Omega_{2h}$, the fine grid vector $v^h = I_{2h}^h v^{2h}$ in Ω_h constructed by the prolongation operator can be defined as follows:

$$\left\{ \begin{array}{c} v_{2j}^h = v_j^{2h} \\ v_{2j+1}^h = (v_j^{2h} + v_{j+1}^{2h})/2 \end{array} \right\} \quad j = 0, \ldots, \frac{n+1}{2}. \tag{3.6}$$

The prolongation can also be written in the following matrix vector multiplication:

$$v^h = \frac{1}{2}\begin{bmatrix} 1 & & & \\ 2 & & & \\ 1 & 1 & & \\ & 2 & & \\ & 1 & 1 & \\ & & \vdots & \\ & & \vdots & \\ & & 1 & 1 \\ & & & 2 \\ & & & 1 \end{bmatrix} v^{2h}.$$

The two dimensional case is similar, and the specific grid points can be written as the following:

$$\left\{\begin{array}{c} v^h_{2i,2j} = v^{2h}_{ij}, \\ v^h_{2i+1,2j} = (v^{2h}_{ij} + v^{2h}_{i+1,j})/2, \\ v^h_{2i,2j+1} = (v^{2h}_{ij} + v^{2h}_{i,j+1})/2, \\ v^h_{2i+1,2j+1} = (v^{2h}_{ij} + v^{2h}_{i+1,j} + v^{2h}_{i,j+1} + v^{2h}_{i+1,j+1})/4. \end{array}\right\},$$

for $i = 0, \ldots, \frac{n+1}{2}$ and $j = 0, \ldots, \frac{m+1}{2}$. In 2D, the prolongation operator can be defined as a tensor product of two one dimensional prolongation operators:

$$I^h_{2h} = I^h_{y,2h} \bigotimes I^h_{x,2h},$$

where $I^h_{x,2h}$ denotes the prolongation in the x direction only and $I^h_{y,2h}$ denotes the prolongation in the y direction only.

Similar to a finite-difference discretized linear system, the above interpolation

operators can also be equivalently expressed using stencils, i.e., the prolongation stencil in 1D is:

$$\Big]\ \tfrac{1}{2}\quad 1\quad \tfrac{1}{2}\ \Big[,$$

while the prolongation stencil in 2D is

$$\frac{1}{4}\Big]\begin{matrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{matrix}\Big[.$$

Note that the notation][means the operators should be interpreted as fan-out operations rather than fan-in.

3.4.2 Restriction

The restriction operation I_h^H is the inverse of prolongation, which maps a fine grid vector $v^h \in \Omega_h$ to a coarse grid vector $v_H \in \Omega_H$:

$$I_h^H : \Omega_h \to \Omega_H.$$

Using the same grids as before with mesh sizes $H = 2h$, the restriction operator in the one dimensional case can be written as the following matrix:

$$v^H = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & 1 & 2 & 1 & & \\ & & & & \cdots & \cdots & \cdots & & \\ & & & & & & & 1 & 2 & 1 \end{bmatrix} v^{2h}.$$

In particular,

$$I_{2h}^{h} = 2(I_{h}^{2h})^{T}.$$

The corresponding restriction stencil can be written as:

$$\frac{1}{4}\left[\begin{array}{ccc} 1 & 2 & 1 \end{array}\right],$$

where the closed bracket means that the stencil should be interpreted as fan-in operation.

Similarly, in the two dimensional case, the restriction stencil can be written as:

$$\frac{1}{16}\left[\begin{array}{ccc} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{array}\right].$$

The 2D prolongation operator and the 2D restriction operator satisfy the following relationship:

$$I_{2h}^{h} = 4(I_{h}^{2h})^{T}.$$

Theorem 3.4.1 shows that the problem is defined by a 3×3 stencil on the finest grid, then, when using full coarsening with linear interpolation, the size of the stencils in multigrid hierarchy will remain identical.

Theorem 3.4.1. *If the stencil of the restriction operator R is $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ and the prolongation operator $P = 4R^{T}$, and if the stencil on the fine level is $\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$,*

then the stencil on the coarse level is

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix},$$

where Please check the statement. you have coarse level twice no fine level.

$w_1 = 1/4a_1 + 1/16a_2 + 1/16a_4 + 1/64a_5,$

$w_2 = 1/4a_1 + 6/16a_2 + 1/4a_3 + 1/16a_4 + 3/32a_5 + 1/16a_6,$

$w_3 = 1/16a_2 + 1/4a_3 + 1/64a_5 + 1/16a_6,$

$w_4 = 1/4a_1 + 1/16a_2 + 6/16a_4 + 3/32a_5 + 1/4a_7 + 1/16a_8,$

$w_5 = 1/4a_1 + 6/16a_2 + 1/4a_3 + 6/16a_4 + 9/16a_5 + 6/16a_6 + 1/4a_7 + 6/16a_8 + 1/4a_9,$

$w_6 = 1/16a_2 + 1/4a_3 + 3/32a_5 + 6/16a_6 + 1/16a_8 + 1/4a_9,$

$w_7 = 1/16a_3 + 1/64a_5 + 1/4a_7 + 1/16a_8,$

$w_8 = 1/16a_4 + 3/32a_5 + 1/16a_6 + 1/4a_7 + 6/16a_8 + 1/4a_9,$

$w_9 = 1/64a_5 + 1/16a_6 + 1/16a_8 + 1/4a_9.$

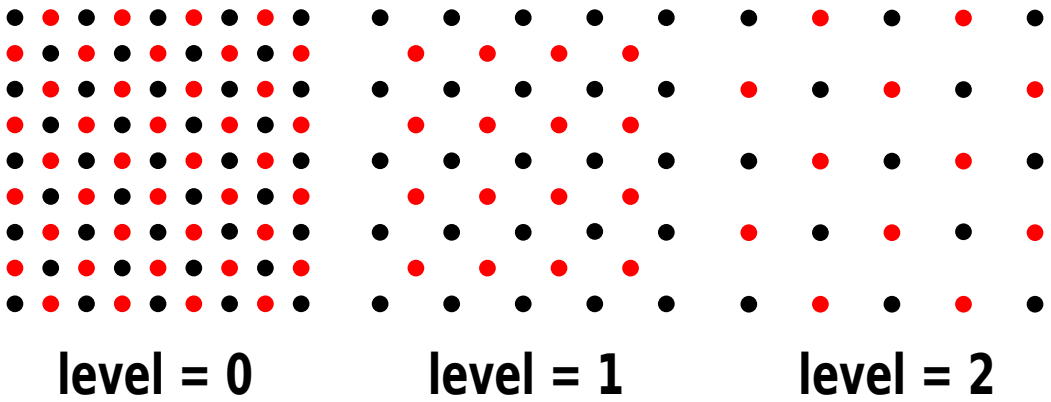


Figure 3.2: Red-black coarsening. The fine points are red and coarse points are black.

To show that our method is not restricted to a certain coarsening scheme, in this thesis we also consider the red-black coarsening scheme that has a coarsening factor about 2 shown in Figure 3.2 for the first 3 levels. Note that the coarsening on level 0 is essentially a semi-coarsening 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 semi-coarsening along the y-dimension. The restriction and interpolation stencils used associated with this coarsening are given by (see [44])

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

3.4.3 Multigrid

Consider the discretized linear system

$$Au = f \tag{3.7}$$

from the one dimensional Poisson's equation, where

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

Let u be a vector whose components are $\sin\theta, \sin 2\theta, \ldots, \sin n\theta$. Then matrix A satisfies

$$Au = \sin\left((n+1)\theta\right)q_n + 2(1-\cos\theta)u,$$

where q_n has 1 at the last entry and 0 elsewhere. Note that for $\theta_k = \frac{k\pi}{n+1}$, we have $\sin(n+1)\theta = 0$ for any integer k. So the eigenvalues of A are

$$\lambda_k = 2(1 - \cos\theta_k) = 4\sin^2\frac{\theta_k}{2}$$

with corresponding eigenvectors:

$$v_k = \begin{pmatrix} \sin\theta_k \\ \sin 2\theta_k \\ \vdots \\ \sin n\theta_k \end{pmatrix}.$$

We now discuss the ability of smoothers to smooth errors in detail and show why they are important in multigrid methods. Take weighted Jacobi as an example, the relaxation step can be written as

$$u_{k+1} = (I - \omega D^{-1}A)u_k + \omega D^{-1}f.$$

Note that for the problem considered (3.7), the diagonal matrix is in fact $D = 2I$. Therefore, we can simply denote $\omega D^{-1}A$ by $\frac{\omega}{2}A$, so the iteration matrix can be written as:

$$S_\omega = I - \frac{\omega}{2}A.$$

Then, the eigenvalues of S_ω are $1 - \frac{\omega}{2}\lambda_k$ where λ_k are the eigenvalues of A. Denote the ground truth solution by u_*, then the error of the approximation at j step of the relaxation method, denote by $e_j \equiv u_* - u_j$, satisfies

$$e_j = S_\omega^j e_0. \tag{3.8}$$

Let's expand the initial error vector e_0 using the eigenbasis of the iteration matrix S_ω

and coefficients η_i so that

$$e_0 = \sum_{i=1}^{n} \eta_i v_i. \tag{3.9}$$

By (3.8) and (3.9), we have

$$\begin{aligned} e_j &= \sum_{i=1}^{n} (1 - \frac{\omega}{2}\lambda_i)^j \eta_i v_i \\ &= \sum_{i=1}^{n} (1 - 2\omega \sin^2 \frac{i\pi}{2(n+1)})^j \eta_i v_i. \end{aligned} \tag{3.10}$$

From (3.10) we can see that the convergence speed is not similar for all components. The corresponding error reduction of each component depends on the magnitude of the eigenvalues. The components corresponding to large eigenvalues, with $i > \frac{n}{2}$, converge faster than the components corresponds to small eigenvalues to 0. These components are called *high frequency* components and the part of the error is called the *oscillatory part* because of trigonometric properties.

The relaxation methods can rapidly reduce the oscillatory part of the error corresponding to the high frequency components, but they may have slow progress for the remaining low frequency components. Multigrid methods favor the smoothing ability of relaxation methods and use them as "smoothers" to smooth the errors. The smoothed error is then injected to a coarse grid and can be reduced further.

Now consider the fine grid Ω_h, where the points on the fine grid are $x_i^h = i * h$ for $i = 1, \ldots n + 2$ with $h = \frac{1}{n+1}$, and n is odd. Consider the coarse grid Ω_{2h}, where the discretization points are $x_i^{2h} = i * 2h, i = 1, \ldots \frac{n+3}{2}$. Base on the simple fact that $x_i^{2h} = x_{2i}^h$, the components on the grids follow the following relationship:

$$v_k^h(x_{2i}^h) = \sin(k\pi x_{2i}^h) = \sin(k\pi x_i^{2h}) = v_k^{2h}(x_i^{2h}).$$

Some of the low frequency components on the fine grid become high frequency on the coarse grid, for example, the component $v_{\frac{n+3}{2}}^h$ become the highest component

on the coarse grid. The error corresponding to those components can therefore be eliminated on the coarse grid by smoothers. In fact, multigrid requires going back and forth through the grids to obtain an accurate solution at the end.

Convergence theory of two-grid (TG) methods has been well studied [10,12,23,60] through the error propagation operator E_{TG} of the form:

$$E_{\mathrm{TG}} = (I - M^{-1}A)(I - P(P^{\mathsf{T}}AP)^{-1}P^{\mathsf{T}}A), \tag{3.11}$$

where M is the smoother, $P \in \mathbb{R}^{n\times n_c}$ is the prolongation operator, P^{T} is typically used as the restriction operator for symmetric problems, and $P^{\mathsf{T}}AP$ is the Galerkin coarse-grid operator. In general, smaller $\|E_{\mathrm{TG}}\|_A$ indicates faster convergence for two-grid methods on $Ran(P)^{\perp_A}$ and has the kernel space $Ran(P)$.

In this thesis, we choose standard prolongation operators P (see [44]), and focus on smoothers in Chapter 4 and coarse grid operators in Chapter 5. Theorem 3.4.2 summarizes the main convergence result in [23] with respect to M and P.

Theorem 3.4.2 ([23]). *Assuming $M^{\mathsf{T}} + M - A$ is SPD, denote $\tilde{M}$ by*

$$\tilde{M} = M^{\mathsf{T}}(M^{\mathsf{T}} + M - A)^{-1}M, \tag{3.12}$$

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

$$K = \max_{e\neq 0} \frac{\|(I - PR)e\|_{\tilde{M}}^2}{\|e\|_A^2}. \tag{3.13}$$

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

The quantity K in (3.13), which is the so-called *weak approximation property* [11], essentially measures how accurately interpolation approximates the eigenvectors of $M^{-1}A$ proportional to the corresponding eigenvalues.

Notice that compared with relaxation matrix in (3.3), efficient smoothers M in

(3.11) only need to eliminate errors that are A-orthogonal to $Ran(P)$, which correspond to high-frequency errors, in order to have fast convergence.

Theorem 3.4.3 ([23, Theorem 3.1]). *Let R be any matrix such that $RP = I$ and $S \in \mathbb{R}^{n \times n_f}$ such that $RS = 0$, and define*

$$K_* = \min_P \max_{e \neq 0} \frac{\|(I - PR)e\|_{\tilde{M}}^2}{\|e\|_A^2} \ .$$

The minimum K_* is given by

$$K_* = \frac{1}{\lambda_{\min}((S^\mathsf{T}\tilde{M}S)^{-1}(S^\mathsf{T}AS))} \ , \tag{3.14}$$

and the corresponding minimizer is

$$P_* = (I - S(S^\mathsf{T}AS)^{-1}S^\mathsf{T}A)R^\mathsf{T}. \tag{3.15}$$

Interpolation P_* in (3.15) is often referred to as the ideal interpolation operator. Note that in practice it is typically too expensive to use P_*, since $(S^\mathsf{T}AS)^{-1}$ is generally not sparse. However, it is still reasonable to use K_* to analyze the convergence of the smoothers. Theorem 3.4.4 gives a bound of the constant K when $P \neq P_*$.

Theorem 3.4.4 ([23, Theorem 4.2]). *Assuming $\|PR\|_A^2 < \eta$, K is as defined in* (3.13) *and K_* in* (3.14)*, we have*

$$K \leq \eta K_* \ . \tag{3.16}$$

The importance implification of Theorem 3.4.4 is that neither P_* nor η depends on M (whereas K_* does), which implies that the procedure of developing smoothers can be done independently from selecting coarse grids and computing interpolation operators.

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 [2], which can be computed explicitly.

Definition (Ideal uniform bound of convergence rate)**.** Suppose P takes the form of $P = \binom{W}{I}$ as in standard multigrid algorithms. Let $R = \begin{pmatrix} 0 & I \end{pmatrix}$ and $S^{\mathsf{T}} = \begin{pmatrix} I & 0 \end{pmatrix}$. we define quantity β_* such that

$$\beta_*^2 = (1 - 1/K_*) = [1 - \lambda_{\min}((S^{\mathsf{T}}\tilde{M}S)^{-1}(S^{\mathsf{T}}AS))], \tag{3.17}$$

which can be considered as the ideal uniform bound of convergence rate [23]. This quantity is often used to analyze convergence rate of smoothers in two-grid methods [2].

In Section 4.4, we will use (3.4.3) to measure the convergence rate of two-grid methods with the proposed smoothers.

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 1 for a brief description of standard multigrid V-cycle. An illustration is given in Figure 3.3. Notice that the smoother $M^{(l)}$ at level l is only required to eliminate errors that are $A^{(l)}$-orthogonal to $Ran(P^{(l)})$ in order to have fast convergence. This property will be used to design efficient training strategies for learning neural smoothers in Chapter 4.

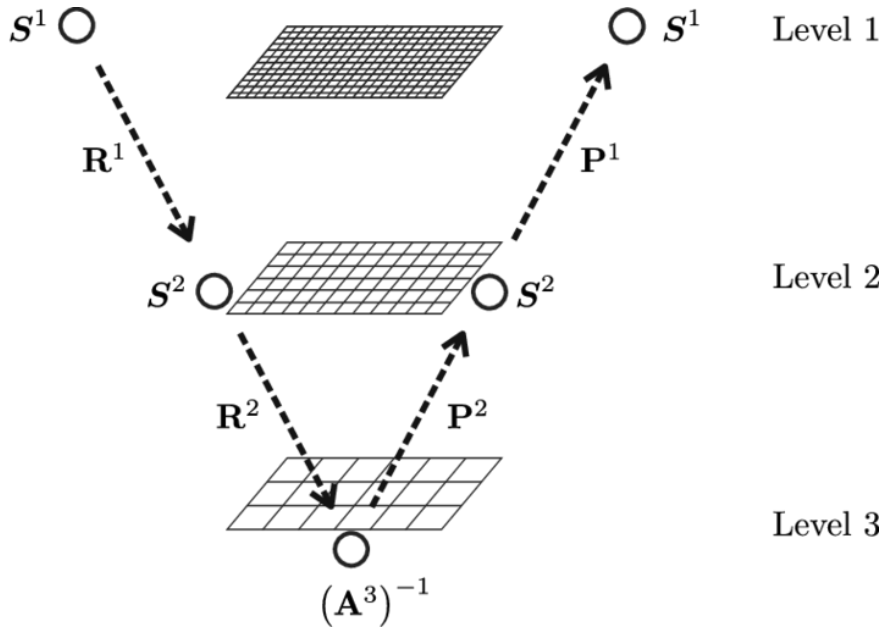


Figure 3.3: Graph illustration of Multigrid V-cycle

Algorithm 1 Multigrid V-cycle for solving $Au = f$

Input: Linear system $Au = f$ with initial guess u_0. Multigrid hierarchy: for each level l, coefficient matrix $A^{(l)}$, approximation $u^{(l)}$, right-hand side $f^{(l)}$, smoother $M^{(l)}$ and prolongation matrix $P^{(l)}$, where $A^{(0)} = A$, $f^{(0)} = f$ and $u^{(0)} = u_0$
Output: The approximated solution u_k after k steps of multigrid V-cycle
Pre-smoothing: $u^{(l)} = u^{(l)} + (M^{(l)})^{-1}(f^{(l)} - A^{(l)}u^{(l)})$
Compute fine-level residual: $r^{(l)} = f^{(l)} - A^{(l)}u^{(l)}$, and restrict it to the coarse level: $r^{(l+1)} = (P^{(l)})^{\mathsf{T}} r^{(l)}$
if $l + 1$ is the last level **then**
    Solve $A^{(l+1)}u^{(l+1)} = r^{(l+1)}$
else
    Call multigrid V-cycle recursively with $l = l + 1$, $f^{(l+1)} = r^{(l+1)}$ and $u^{(l+1)} = 0$
end if
Prolongate the coarse-level approximation and correct the fine-level approximation: $u^{(l)} = u^{(l)} + P^{(l)}u^{(l+1)}$
Post-smoothing: $u^{(l)} = u^{(l)} + (M^{(l)})^{-1}(f^{(l)} - A^{(l)}u^{(l)})$

The multigrid V-cycle iteration can also be regarded as the following updating step:

$$u_{k+1} = MG^{(\ell)}u_k, \tag{3.18}$$

where we refer $MG^{(\ell)}$ as the multigrid iteration matrix with $\ell + 1$ levels.

Starting with $MG^{(0)} = (A^{(0)})^{-1}$ where $A^{(0)}$ is the problem matrix at the coarsest level. The iteration matrix $MG^{(\ell)}$ of multigrid V-cycle with $\ell + 1$ levels can be computed by the following recursion:

$$I^{(\ell)} - MG^{(\ell)}A^{(\ell)} = S^{(\ell)}(I^{(\ell)} - P^{(\ell)}MG^{(\ell-1)}R^{(\ell)}A^{(\ell)})S^{(\ell)}, \quad \ell = 1, \ldots, n,$$

where $A^{(\ell)}$ is the problem matrix at ℓth level, $S^{(\ell)}$ is the smoothing matrix, $P^{(\ell)}$ is the prolongation matrix and $R^{(\ell)}$ is the restriction matrix. The error propagation matrix can be written as $E = I - M^{(\ell)}A^{(\ell)}$, then the updating scheme (3.18) converges if $\rho(E) < 1$.

3.5 Relationship between PDEs and CNNs

Convolutional neural networks (CNNs) is constructed with convolutional layers, and have been widely used in the application of image processing. Each convolutional layer is constructed by the shared-weight architecture of convolution kernels. Convolutions have close relationship with PDEs. In fact, many operations in PDEs can be written as convolutions.

Let the filter be $W \in \mathcal{R}^{F\times F\times d} = [w_{i,j,k}]$ and the feature matrix be $H \in \mathcal{R}^{L\times B\times d} = [h_{i,j,k}]$. Then the general form for convolution with stride S is defined as [1]:

$$h_{ijp} = \sum_{r=1}^{F}\sum_{s=1}^{F}\sum_{k=1}^{d} w_{rsk}h_{i+r-1,j+s-1,k}, \tag{3.19}$$

for $i = 1, 1+S, 1+2S, \ldots, L-F+1$, $j = 1, 1+S, 1+2S, \ldots, B-F+1$, $p = 1, \ldots, d$.

CNNs are well-known as shift-invariant or space-invariant neural networks, which can efficiently aggregate local information. Some PDEs like Poisson's equation also consider local information because of the physical properties. Therefore, it is natural

to relate PDEs with CNNs. In fact, the operations involving stencils described above in Chapter 2 can be equivalently written as convolution operations.

The linear system of a discretized PDE can be written as convolution with stride 1:

$$\mathcal{A} * u = f,$$

where $\mathcal{A}$ is the stencil, $u \in \mathbb{R}^{n \times n}$ and $f \in \mathbb{R}^{n \times n}$. The restriction operator can be written as convolution with stride 2 as shown in Figure 3.4.

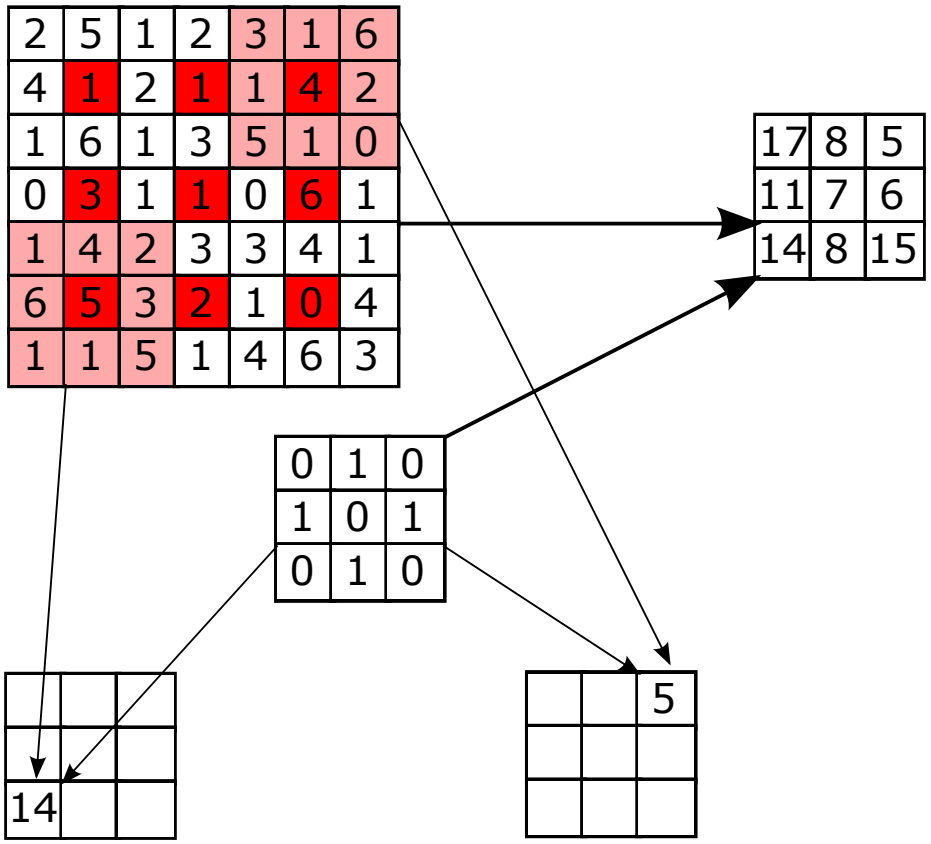


Figure 3.4: An example of convolving a 7×7 matrix by a 3×3 kernel with stride of 2.

Chapter 4

Learning deep neural smoothers

As we can see from the analysis described in Chapter 3, 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. The design of smoothers for a certain type of PDEs is often problem-dependent and requires expert knowledge. In this chapter, we propose an adaptive framework for learning smoothers for both constant coefficient PDEs (Section 4.1) and variable coefficient PDEs (Section 4.3). We use the same learning framework discussed in Section 4.1.2 for both types of PDEs. The construction of the neural smoothers is different because of the nature of the problems. We 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 in Section 4.1.1. The learned smoothers are represented as a sequence of convolutional layers. The interpretation of the learned smoothers is described in Section 4.2. In Section 4.3, instead of directly parameterizing the smoothers using neural networks as in constant coefficient case, we train neural networks which generate the weights of the smoothers in the variable coefficient case. Numerical experiments for both constant coefficient problems and variable coefficients

problems are describe in Section 4.4. Concluding remarks are given in Section 4.5.

4.1 Learning deep neural smoothers for constant coefficient PDEs

4.1.1 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 2-D square domain, we discretize it on a regular 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 neural smoothers $M^{(0)}, \ldots, M^{(L-1)}$ on the first L levels of a multigrid solver that has $L+1$ levels. Without loss of generality, we assume that the multigrid solver uses the same smoother for both the pre-smoothing and post-smoothing steps (c.f., lines 1 and 9 in Algorithm 1, respectively), and uses direct methods (e.g.LU decomposition) 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

$$\|\Phi^{(0)}(u_0, f, k) - u_*\|_2, \tag{4.1}$$

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 multigrid V-cycles with $\Phi^{(0)}$.

Although multigrid convergence theories depend on the spectral properties of the associated iteration matrix, it has been shown in [55] that backpropagation through eigendecomposition is unstable. Therefore, we instead minimize the loss (4.1) since it 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 (4.1) 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 (4.1) 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 4.1.1 ([26]). *For any matrix $X \in \mathbb{R}^{n\times n}$ and $z \in \mathbb{R}^n$ that is uniformly distributed on unit n-sphere, we have*

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

In this thesis, we fix the PDE class $\mathcal{A}$ but vary the forcing term $\mathcal{F}$ and the boundary condition $\mathcal{G}$, and 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

$$\mathcal{D} = \bigcup_{j=1}^{Q} \{A, f_j, (u_0)_j, (u_*)_j\} \tag{4.2}$$

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:

$$-\nabla \cdot (g\nabla u(x, y)) = f(x, y), \tag{4.3}$$

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 (4.3) 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 in-

verse of the smoother $(M^{(l)})^{-1}$ by one single convolutional neural network, $H^{(l)}$, 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 4.1.2 justifies the use of this parameterization to construct convergent smoothers.

Theorem 4.1.2. *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 convolutional neural networks without fully connected layers [62], 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. □

4.1.2 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 $\ell = 0, \ldots, L-1$, where we construct a training set $\mathcal{D}^{(l)}$ similar to (4.2) for the operator $A^{(l)}$. That is, we train $H^{(l)}$ to make iteration (3.3) convergent by minimizing the error between the approximate solution obtained at iteration k and the ground truth solution. In particular, on each level ℓ of the multigrid hierarchy, we construct

an individual training set

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

and we minimize the following loss function:

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

where we consider the smoother

$$\Psi^{(\ell)}(u^{(\ell)}, f^{(\ell)}, 1) = u^{(\ell)} + H^{(\ell)}(f^{(\ell)} - A^{(\ell)}u^{(\ell)})$$

as the solver on this level. As suggested in [31], we also choose different iteration number k, $1 \leq k \leq b$ in the training, so that $H^{(\ell)}$ 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 straightforward and the loss function is not complicated to optimize. The trainings on different levels are totally independent and therefore can be done efficiently in parallel. However, we found that the obtained $H^{(\ell)}$ usually do not exhibit good smoothing property of reducing high-frequency errors, especially when $H^{(\ell)}$ 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 $A^{(\ell)}$. In contrast, a well-trained $H^{(\ell)}$ with high complexities, deeper in the layers and larger in the convolution kernels, can approximate the action of the inverse of $A^{(\ell)}$ 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 (4.1) directly over

$M^{(\ell)}$ at all levels, $\ell = 0, \ldots, L-1$. This approach targets at optimizing convergence of the overall multigrid V-cycles and considers both the smoothing and the coarse-grid correction. Since the entire multigrid hierarchy is considered during training, the trained smoothers should have desired smoothing property if the global minimum of the loss function is found. 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.

To overcome the weakness of the previous two approaches, we propose an efficient adaptive training strategy that can impose the smoothing property during training. Instead of treating the smoothers as solvers to decrease the error at each level, for a fixed coarsest level, we use the multigrid solver starting at the corresponding level in the loss function.

This is done in a reverse order. 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 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 2. Figure 4.1 illustrates the procedure of adaptively training a 5-level multigrid solver in 4 stages, starting at level 3. The loss is given by

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

where $\Phi^{(3)}$ represents the two-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 $A^{(0)} \equiv A$. We denote the

smoothers learned by this strategy by α-CNN smoothers.

Algorithm 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 $A^{(\ell)}$ where $A^{(0)} = A$, and interpolation operator $P^{(\ell)}$, for $\ell = 0, \ldots, L-1$. Size of training set Q. Maximum allowed number of smoothing steps b

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

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

4: Construct training set:

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

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

6: Perform stochastic gradient descent (SGD) to minimize loss function:

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

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

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

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

7: **end for**

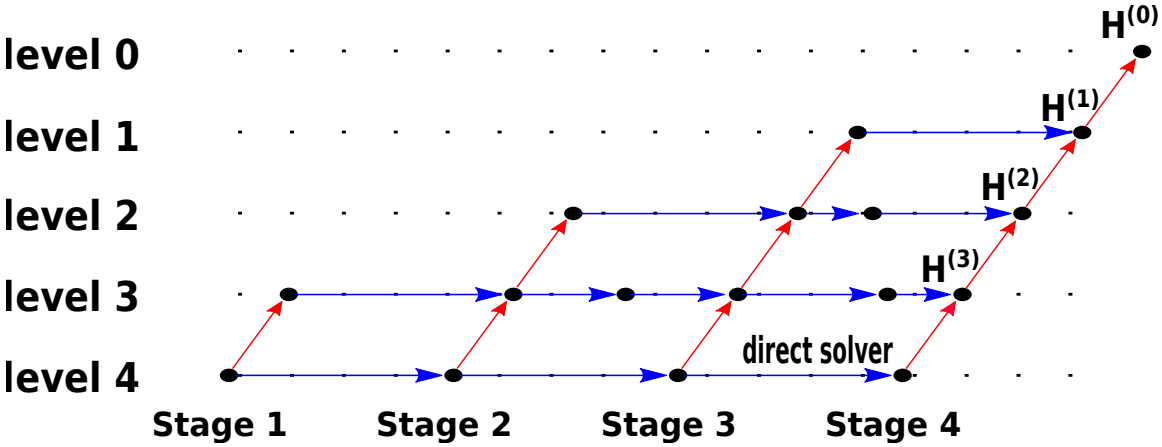


Figure 4.1: The proposed adaptive training strategy for k levels, which starts from level $k-2$ and proceeds upwards. When $H^{(l)}$ is being trained, lower level $H^{(j)}, j = l+1, \ldots, k-2$ are used for the solve with coarse-grid operators and remain unchanged.

Since the learned smoothers are constructed by a stack of convolutional layers which can gather the information of the neighborhood around each grid point to smooth the solution, this invariant property guarantee that they are not restricted to a certain grid size or geometry.

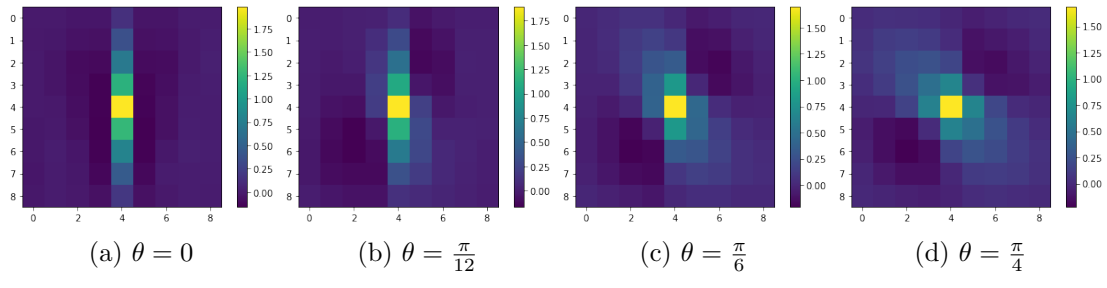
Another appealing property of the proposed training approach is the updatability of smoothers using neural networks. The trained smoothers can be updated in another training process by injecting the errors that cannot be effectively reduced by the current multigrid solver back to the training set. Specifically, 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 re-train $\Phi^{(0)}$ as before with the new augmented training sets using the existing $H^{(l)}$ in the multigrid hierarchy as the initial values.

4.2 Interpretation of learned smoothers

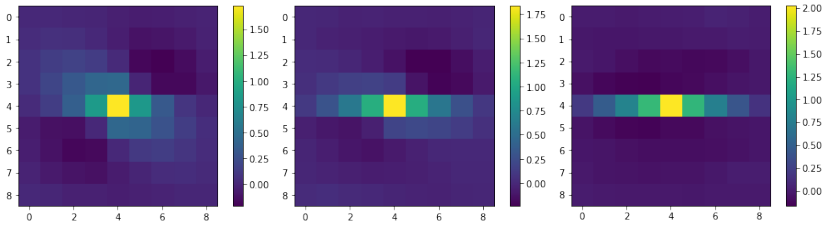
In this section, we illustrate the patterns of the learned smoothers. Notice that the experiements conducted in this section are of visualization purpose and use a different set of parameters as those used in the experiments in Section 4.4. We consider the anisotropic rotated Laplacian problem (5.11) 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 4.2. 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 in the most relevant regions.

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 4.4 as they are more efficient in practice and show the results in Figure 4.3. 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 column of Figure 4.3. The kernels in the last column display similar patterns as in Figure 4.2

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) $\theta = \frac{\pi}{2}$

Figure 4.2: Patterns of the trained kernels for (5.11) 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.

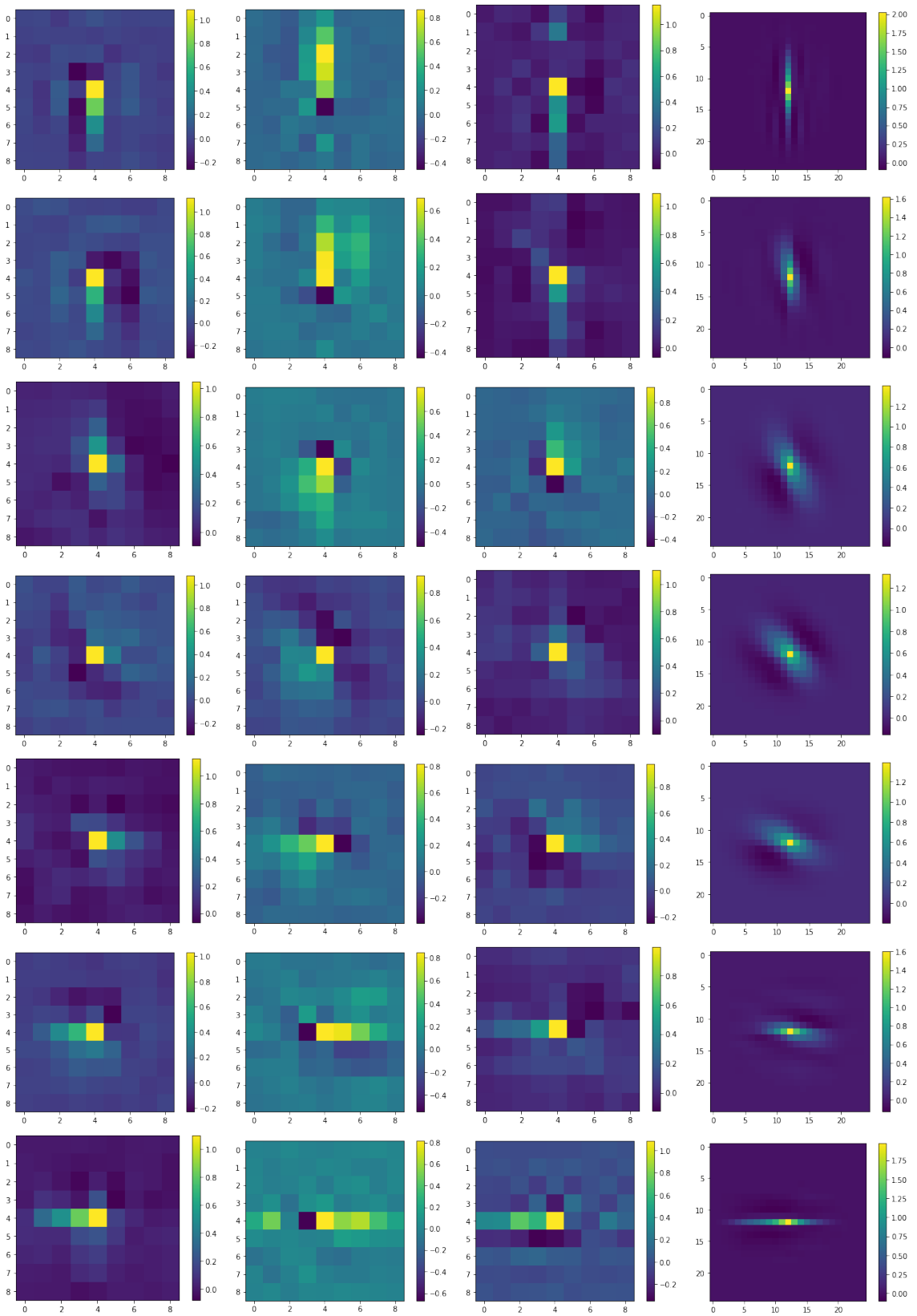


Figure 4.3: Patterns of the trained kernels for (5.11) with $\xi = 100$ and $\theta \in \{0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}, \frac{\pi}{2}\}$. For each problem, the smoother uses three kernels. The first three rows represent the kernels on the first, second and third layers respectively and the last row combines the three kernels into one single kernel for each problem.

4.3 Learning deep neural smoothers for variable coefficient PDEs

In this section, we extend the adaptive training framework proposed in Section 4.1 to design optimal neural smoothers for solving variable coefficient PDEs where the coefficient function g is no longer constant:

$$-\nabla \cdot (g(x,y)\nabla u(x,y)) = f(x,y). \tag{4.4}$$

To better illustrate the difficulty of dealing with variable coefficient PDEs, we simplify our discussion and consider discretizing (4.4) using nine-point stencils with grid spacing h. See the left subfigure of 4.4 for a demonstration of 3×3 neighborhood of the grid point u_{22}. The equation corresponds to the grid point u_{22} reads:

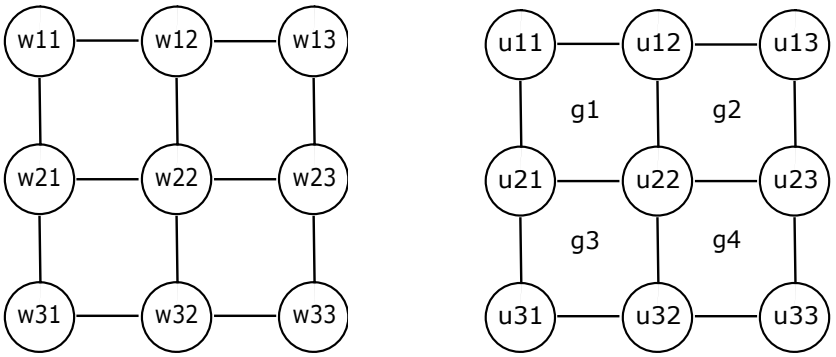
$$\begin{aligned}
&-\frac{1}{3h^2}(g_1 u_{11} + g_2 u_{13} + g_3 u_{31} + g_4 u_{33}) \\
&-\frac{1}{6h^2}((g_1+g_2)u_{12} + (g_2+g_4)u_{23} + (g_3+g_4)u_{32} + (g_1+g_3)u_{21}) \\
&+\frac{2}{3h^2}(g_1+g_2+g_3+g_4)u_{22} = f_{22},
\end{aligned}$$

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

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

where w_{ij} are computed according to the function $g(x,y)$ and is shown in the right subfigure of 4.4. When $g(x,y)$ is constant, the coefficients w_{ij} correspond to each interior 3×3 stencil are identical. Thus, we can parameterize M^{-1} by a single convolutional neural network 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 x-axis and weak in y-axis while W_{lm} is strong in y-axis and weak in 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.



(a) Weight stencil (b) Grid point

Figure 4.4: Demonstration of weight stencils and grid points.

In order to generate unshared smoothing convolution kernels which are dimension-invariant, instead of directly learning the kernels, 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.

To reduce confusion, we point out the main difference and connection between the design of learning smoothers in 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 2 are the parameters of the smoothers. However, in the variable coefficient case, instead of learning the parameters of the smoothers directly, we 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 coefficient case. In both cases, we use the same learning strategy described in Algorithm 2. We discuss two different parameterization methods of the mapping function in the following sections.

4.3.1 Parameterization with fully connected layers

In the first approach, we consider using multiple layer perceptron (MLP) to construct the mapping from the grid representation to the smoothing convolution 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 enough to learn efficient smoothing kernels and the generalization usually performs poorly. Instead, we suggest to incorporate the neighborhood information into the grid representation. More specifically, we construct each grid representation as an 81×1 vector which consists of the 3×3 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 4.5 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 multigrid solver, we only construct one such neural network based on the adaptive training strategy discussed in Section 4.1.2.

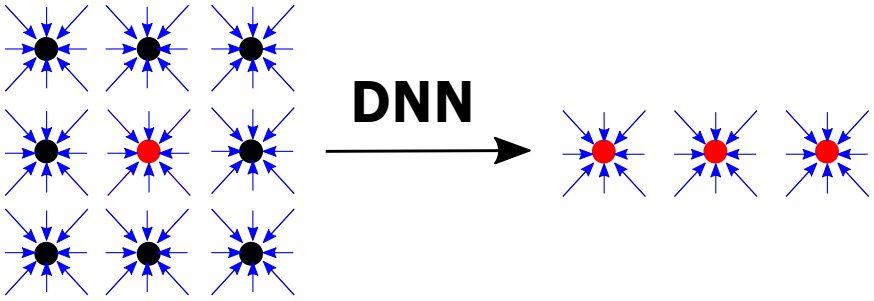


Figure 4.5: The architecture of inferring the smoothing kernels for the central point in the stencil. A fully connected neural network takes nine 3×3 stencils as input and outputs three convolution smoothing kernels for the current grid point.

4.3.2 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 4.6. We will show in numerical experiments that this approach can achieve a comparable performance with fully connected layers but requires much fewer parameters.

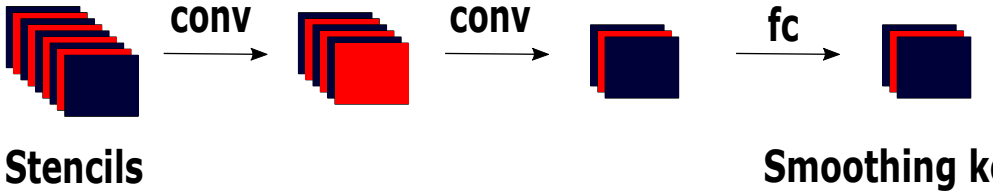


Figure 4.6: The framework of constructing 3 smoothing kernels by applying two convolutional layers and one fully connected layer to a feature map. The feature maps have 9,6,3 and 3 channels, respectively and each channel contains kernels of size 3×3.

4.4 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.

4.4.1 Constant coefficient PDEs

We first consider the following two dimensional anisotropic rotated Laplacian problem:

$$-\nabla \cdot (T \nabla u(x, y)) = f(x, y), \tag{4.5}$$

where 2×2 tensor field T is defined as

$$T = \begin{bmatrix} \cos^2\theta + \xi \sin^2\theta & \cos\theta \sin\theta (1-\xi) \\ \cos\theta \sin\theta (1-\xi) & \sin^2\theta + \xi \cos^2\theta \end{bmatrix}, \tag{4.6}$$

where θ is the angle of the anisotropy and ξ is the conductivity. We discretize the operators Δu and u_{xy} in (4.3) 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

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

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 dataset constructed on square domains with small grid size, and show that the trained neural smoothers 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.

To evaluate our method, we compare the performance of multigrid using Algorithm 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 smoother (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 thesis.

4.4.2 Training details

First, we train smoothers independently using the first strategy discussed in Section 4.1.2. For each smoother, we construct 50 problem instances of size 16^2. Then, we use the adaptive training framework to train smoothers using Algorithm 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 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 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 full coarsening scheme, for both CNN and α-CNN smoothers, $H^{(l)}$ is parameterized as follows:

$$H^{(l)} = f_5^{(l)}(f_4^{(l)}(\cdots(f_2^{(l)}(f_1^{(l)}))\cdots) + f_6^{(l)}, \tag{4.7}$$

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

$$H^{(l)} = f_2^{(l)}(f_1^{(l)}). \tag{4.8}$$

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 3.3 constructed by ω-Jacobi smoothers (ω is fixed at $\frac{2}{3}$ in all experiments) and α-CNN smoothers and summarize the results in Tables 4.1 and 4.2. 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 (3.17) on these tests is provided in Tables 4.3 and 4.4. 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 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 report them in the rest of the section.

ξ	100	200	300	400
ω-Jacobi	0.9886	0.9886	0.9886	0.9886
Gauss-Seidel	0.9662	0.9662	0.9662	0.9662
α-CNN	0.7672	0.8060	0.8588	0.7883

Table 4.1: Spectral radius of iteration matrices 3.3 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 fixed $\theta = 0$ and different ξ. The grid size is 16×16.

The results in Tables 4.1, 4.2, 4.3 and 4.4 show that for each rotated Laplacian problem, the convergence measures associated with α-CNN smoothers are much smaller than those with ω-Jacobi smoothers and Gauss-Seidel smoothers which indi-

θ	$\pi/12$	$\pi/6$	$\pi/4$
ω-Jacobi	0.9913	0.9934	0.9942
Gauss-Seidel	0.9735	0.9797	0.9823
α-CNN	0.7743	0.9652	0.9728

Table 4.2: Spectral radius of iteration matrices 3.3 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 fixed $\xi = 100$ and different θ. The grid size is 16×16.

ξ	100	200	300	400
ω-Jacobi	0.9886	0.9886	0.9886	0.9886
Gauss-Seidel	0.9675	0.9675	0.9675	0.9675
α-CNN	0.7671	0.8060	0.8588	0.7883

Table 4.3: Ideal convergence bound (3.17) for the same methods and problems in Table 4.1.

θ	$\pi/12$	$\pi/6$	$\pi/4$
ω-Jacobi	0.9913	0.9934	0.9942
Gauss-Seidel	0.9748	0.9807	0.9833
α-CNN	0.7743	0.9651	0.9728

Table 4.4: Ideal convergence bound (3.17) for the same methods and problems in Table 4.2.

cates 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 radius of the iteration matrices G of 5-level multigrid solvers equipped with different smoothers and summarize the results in Table 4.5. 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.

ξ	100	200	300	400
ω-Jacobi	0.9853	0.9918	0.9940	0.9951
Gauss-Seidel	0.9564	0.9755	0.9820	0.9853
α-CNN	0.6816	0.8189	0.8805	0.8936

Table 4.5: 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 (5.11) with fixed $\theta = 0$ and different ξ. The mesh size is 16×16.

θ	$\pi/12$	$\pi/6$	$\pi/4$
ω-Jacobi	0.9436	0.8981	0.8837
Gauss-Seidel	0.8566	0.7776	0.7643
α-CNN	0.4534	0.4547	0.4216

Table 4.6: 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 (5.11) with fixed $\xi = 100$ and different θ. The mesh size is 16×16.

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 4.7, 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.

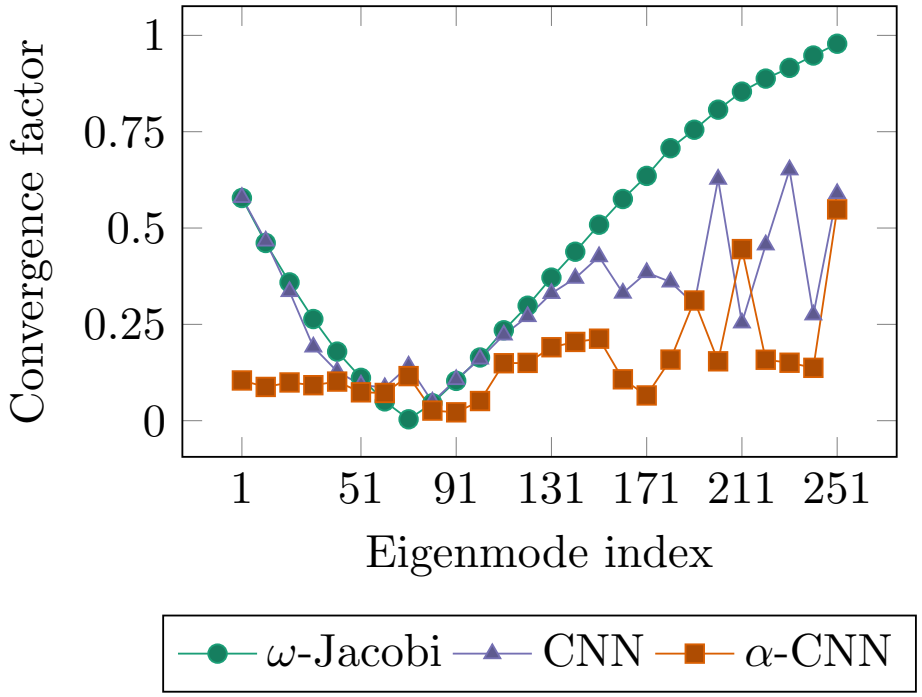


Figure 4.7: Convergence factors of ω-Jacobi with $\omega = \frac{2}{3}$, CNN and α-CNN smoothers to the eigenvectors of A for (5.11) on a 16×16 grid, where $\theta = \frac{5\pi}{12}$ and $\xi = 100$. The eigenvectors are sorted in the descending order of the corresponding eigenvalues.

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, 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 4.8 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.

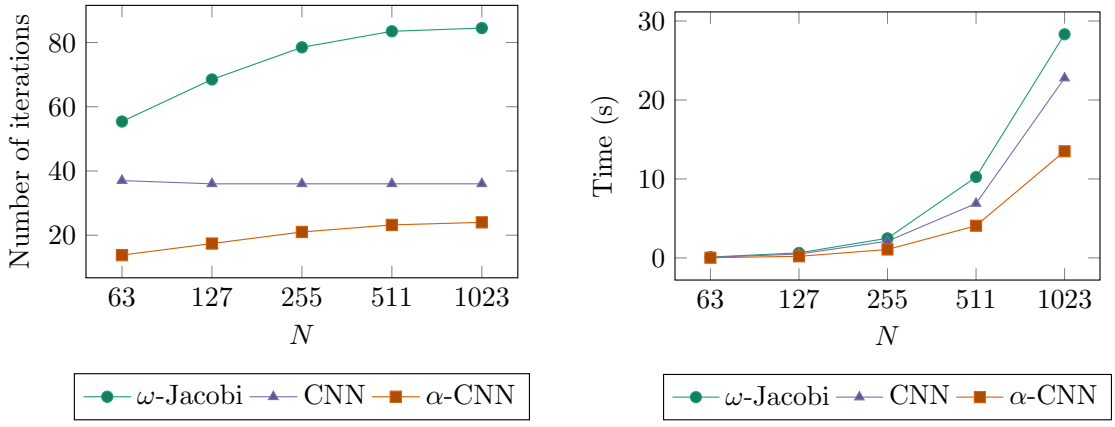


Figure 4.8: Numbers of iterations and runtime required by multigrid with full coarsening to reach convergence tolerance 10^{-6} for solving (5.11) on 63^2, 127^2, 255^2, 511^2 and 1023^2 grids, with parameters $\xi = 100$ and $\theta = \frac{5\pi}{12}$ on square domains.

Since 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 section. 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 4.9) without retraining.

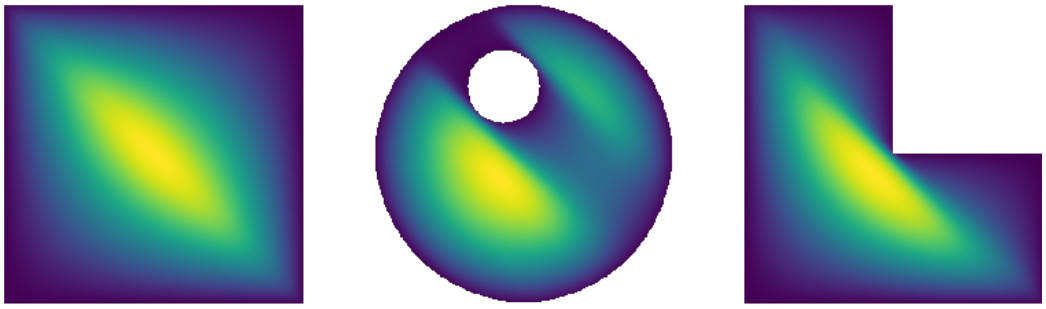


Figure 4.9: Ground truth solutions on square, cylinder and L-shaped domains.

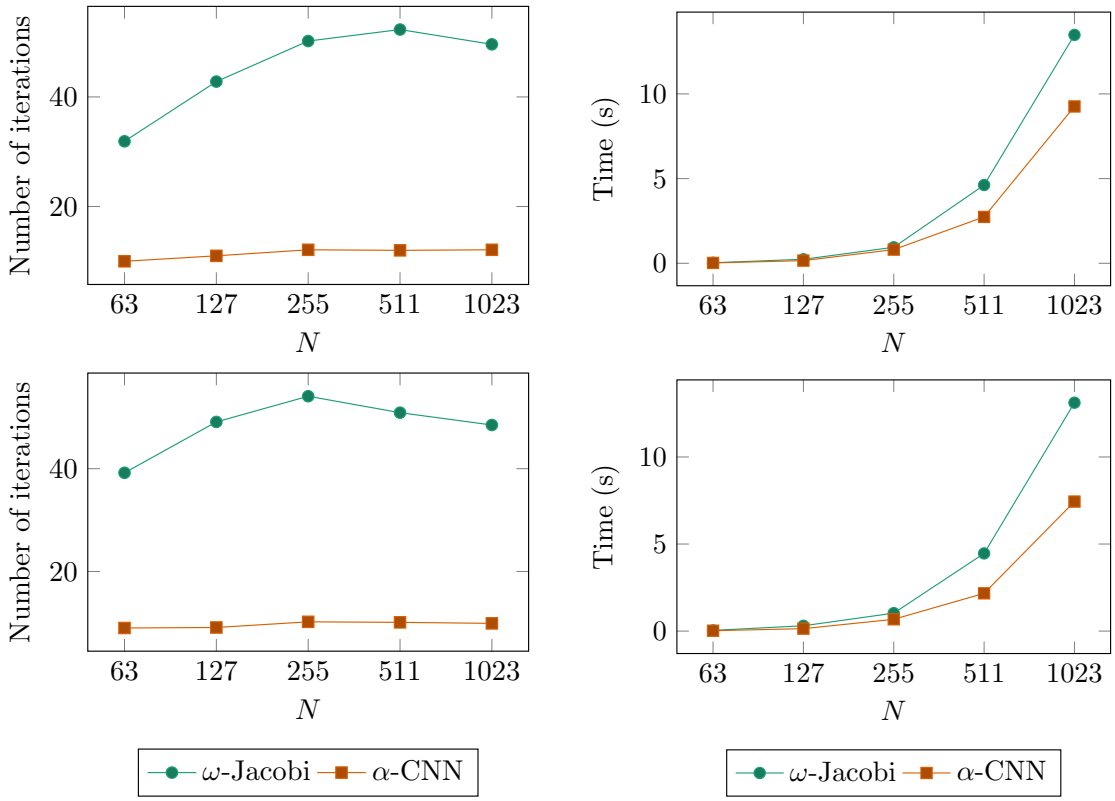


Figure 4.10: Numbers of iterations and runtime required by multigrid solvers for solving (5.11) with parameters $\theta = \frac{\pi}{4}$ and $\xi = 100$ on the cylinder domain (top two figures) and the L-shaped domain (bottom two figures).

The results for the two different domains are shown in Figure 4.10. We can see that since we are using the convolutional layers to approximate the inverse of the smoothers, α-CNN use the information in the neighborhood information to smooth

the error at each grid point and therefore without retraining, the smoother trained on square domain can still lead multigrid methods to converge 4.1 times faster 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 4.11 that our proposed method can learn optimized smoothers for a variety of problems given by different parameters on 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 red-black coarsening scheme is used, 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.

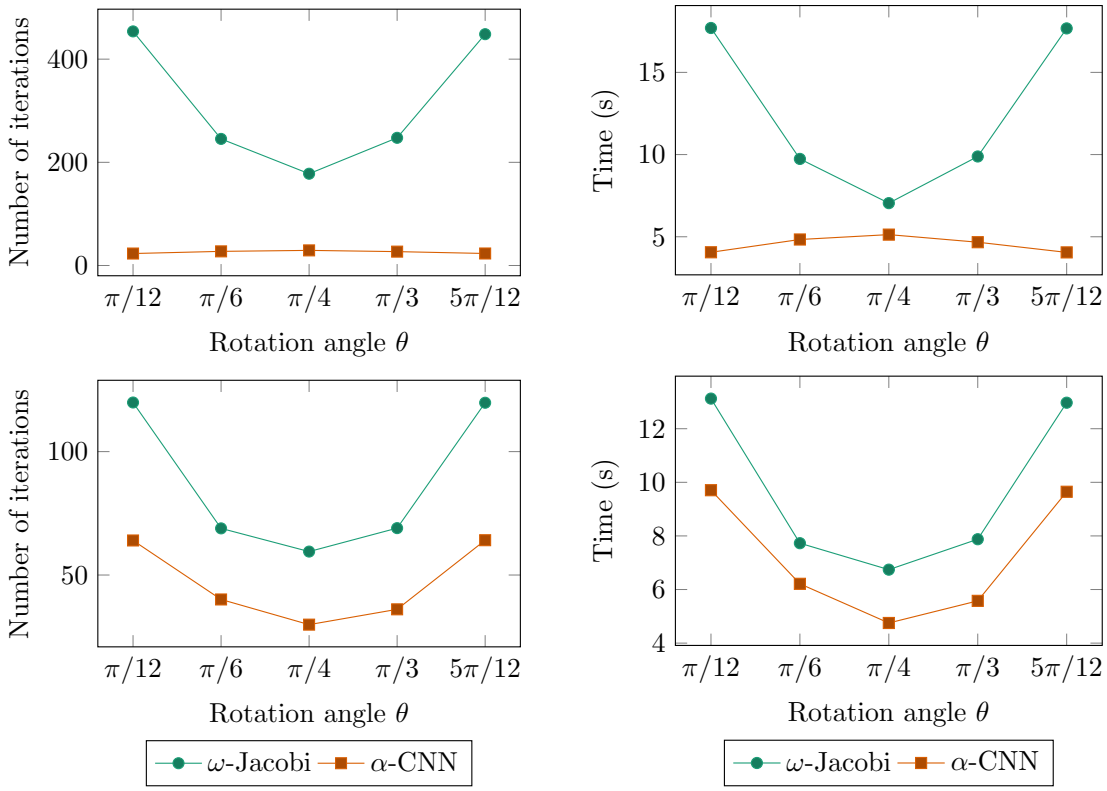


Figure 4.11: Numbers of iterations and runtime required by multigrid solvers for solving (5.11) with $n = 511^2$, $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ and $\xi = 100$. The top and bottom two figures show the performance with full coarsening and red-black coarsening respectively.

Next, we show that a single smoother can be learnt 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 4.12 that the performance of a single smoother for all the problems is slightly worse than the individual training but still outperforms ω-Jacobi. Finally, Figure 4.13 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 4.11. 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.

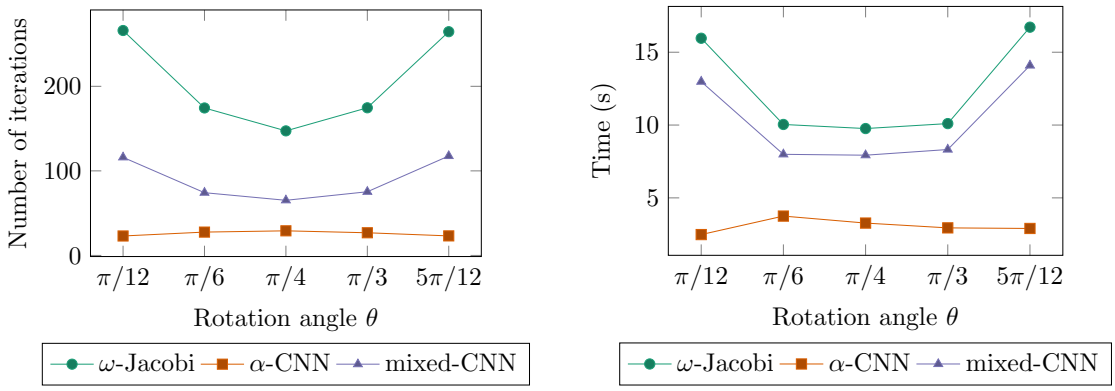


Figure 4.12: Numbers of iterations and runtime required by multigrid solvers with full coarsening for solving (5.11) of size $n = 511^2$ with $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ and $\xi = 100$. The smoother is trained from a dataset containing problems with different θ and ξ.

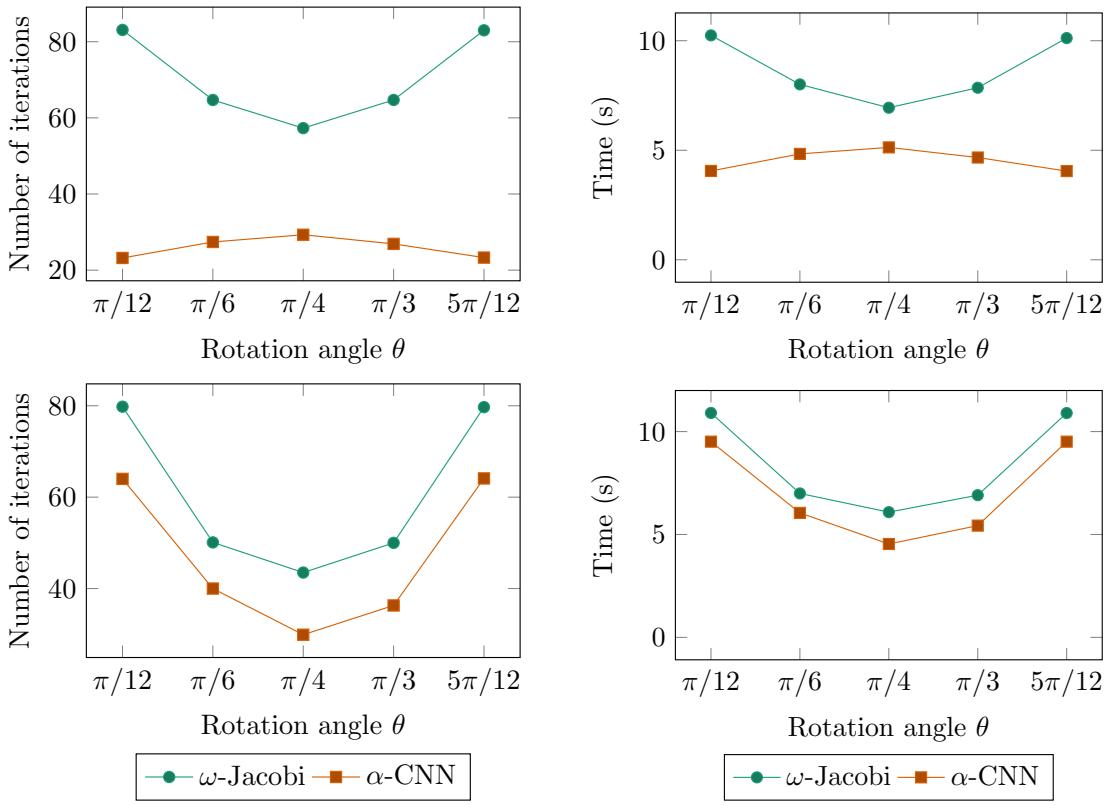


Figure 4.13: Numbers of iterations and runtime required by multigrid for solving (5.11) of size $n = 511^2$ with $\theta = [\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5\pi}{12}]$ and $\xi = 100$. The top and bottom two figures show the performance with full coarsening and red-black coarsening respectively.

4.4.3 Variable coefficient PDEs

We consider the variable coefficient problem:

$$-\nabla \cdot ((\sin \kappa \pi x y + 1.1)\nabla u(x, y)) = f(x, y), \tag{4.9}$$

which is determined by the frequency κ.

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 4.3 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 has 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 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 run time of using different approaches for learning α-CNN smoothers with weighted Jacobi and show the results in Tables 4.7 and 4.8. We also show the spectral properties of each smoother in Tables 4.9 and 4.10 . We also show the 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.

	$\kappa = 0.1$	$\kappa = 1$	$\kappa = 10$	$\kappa = 100$
ω-Jacobi	17	17	20	63
α-CNN-CNN	6	7	11	30
α-FC-CNN	6	6	10	28

Table 4.7: Numbers of iterations required by multigrid for solving (4.9) 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	0.169	0.167	0.191	0.500
α-CNN-CNN	0.102	0.114	0.151	0.333
α-FC-CNN	0.103	0.101	0.142	0.315

Table 4.8: Run time (in seconds) required by multigrid for solving (4.9) of size $n = 255^2$ with $\kappa = 0.1, 1, 10, 100$ using α-CNN and ω-Jacobi with $\omega = \frac{2}{3}$.

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 4.9: Spectral radius of iteration matrices (3.3) of two-grid for solving (4.9) 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	0.9952	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 4.10: Ideal convergence bound (3.17) for the same methods and problems in (4.9).

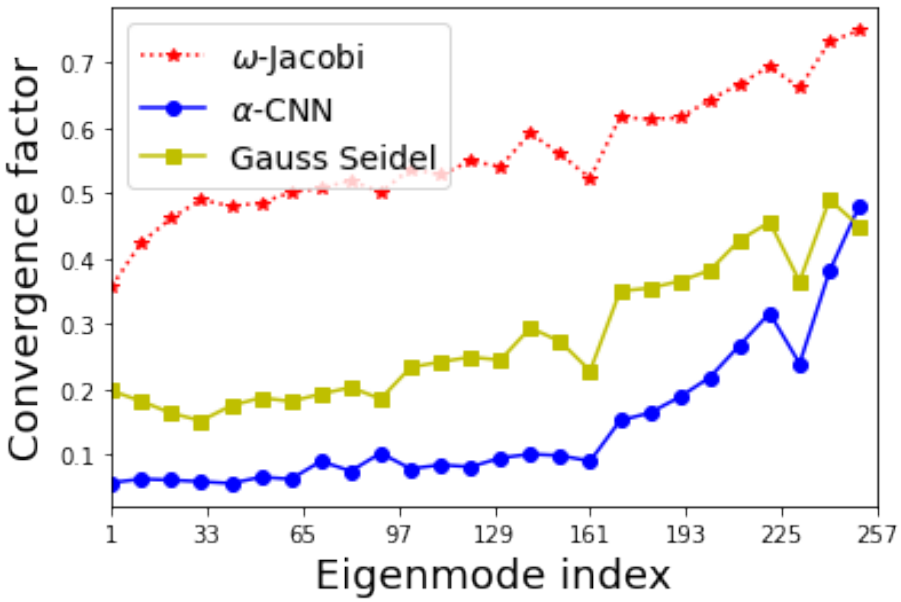


Figure 4.14: Convergence factors of ω-Jacobi with $\omega = \frac{2}{3}$, CNN and α-CNN smoothers to the eigenvectors of A for (5.11) on a 16×16 grid, where $\theta = \frac{5\pi}{12}$ and $\xi = 100$. The eigenvectors are sorted in the descending order of the corresponding eigenvalues.

4.4.4 Incorporation with FGMRES

A multigrid step can also be written as an iterative step with

$$u_{k+1} = MG(u_k).$$

In the case where linear activation is used in the network. The iteration is linear, and the iteration is equivalent to applying a matrix

$$u_{k+1} = Gu_k.$$

Then instead of solving the linear system

$$Au = f,$$

we can solve the following linear system using GMRES

$$AGx = f$$

where G is the multigrid precondioner and the solution $u = Gx$,.

The right preconditioned GMRES is convergent only when the preconditioner G is fixed during iteration. In the case when non linear activations are used in the network, the multigrid iteration is not linear and the multigrid iteration MG can not be viewed as a fixed iteration for each step and the right preconditioned GMRES is not applicable. Nevertheless, flexible GMRES (FGMRES) can be invoked in this case.

We use multigrid solvers as preconditioners of flexible GMRES (see [44]) on the same group of problems as in Table 4.11. 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 Tables 4.11 and 4.12 that using α-CNN can achieve up to 3.36× improvement in terms of iteration number and up to 1.5× improvement in terms of time compare to ω-Jacobi.

$\xi = 100$	$\theta = \pi/12$	$\theta = \pi/6$	$\theta = \pi/4$	$\theta = \pi/3$	$\theta = 5\pi/12$
ω-Jacobi	37.0	30.2	28.0	30.0	37.0
α-CNN	11.0	12.0	13.0	12.0	11.0

Table 4.11: Numbers of iterations required by preconditioned FGMRES to reach tolerance 10^{-6} for solving (5.11) 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	3.48	2.86	2.74	2.65	3.46
α-CNN	2.32	2.52	2.56	2.47	2.29

Table 4.12: Run time(in seconds) required by preconditioned FGMRES to reach tolerance 10^{-6} for solving (5.11) with different θ and ξ. The grid size is 511^2.

4.4.5 Comparison with Chebyshev smoothers

In this section, we consider problem (5.11) 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 show the results on problems of various sizes in Table 4.13. Note that Chebyshev smoothers require estimates of spectral radius $\lambda^*_{\max}$ and are computed on interval $(\gamma_1 \times \lambda^*_{\max}, \gamma_2 \times \lambda^*_{\max})$. 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 [41].

	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
α-CNN	51.0	54.0	53.0	52.0

Table 4.13: Number of iterations required by ω-Jacobi, α-CNN smoothers and Chebyshev smoothers to reach the convergence tolerance 10^{-6} for solving the rotated Laplacian problems with different grid sizes where $\theta = 0$ and $\xi = 100$.

4.4.6 Comparison with GMRES smoothers

In this section, we consider the following convection diffusion problem:

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

We use the upwind finite difference discretization on a regular grid with uniform mesh size h in all directions [40]. 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 4.14.

	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

Table 4.14: 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.

4.5 Conclusion

In this chapter we propose an efficient framework for training smoothers in the form of multi-layered 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 have the desired property of minimizing errors that cannot be efficiently annihilated by coarse-grid corrections. Experiments on rotated Laplacian problems show 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.

Chapter 5

Learning sparsified coarse-grid operator

Recall in Chapter 3 we have discussed the basic ingredients, smoothers and coarse grid operators, of multigrid methods. In Chapter 4, we have proposed an efficient adaptive learning algorithm which can learn smoothers parameterized by neural networks that can improve the convergence of multigrid. In this chapter, we focus on sparsifying the coarse grid operators in order to improve the parallel scalability of multigrid methods. This chapter is organized as follows. We describe the problem of multigrid with increasing density in Section 5.1. We propose a machine learning framework to control the sparsity in multigrid hierarchy without influencing the convergence in Section 5.2. Numerical results on anisotropic rotated Laplacian problems and linear elasticity problems are given in Section 5.3 and conclusions are given in Section 5.4.

5.1 Motivation

Although the coarse grid operators in multigrid methods have smaller sizes, they often have a decreased sparsity.

Consider the linear system from the finite difference discretization of the 3D Pois-

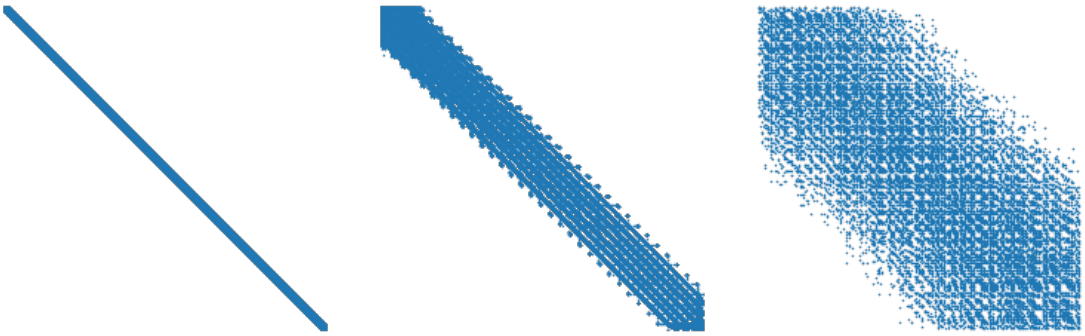


Figure 5.1: [5] Matrix sparsity pattern using classical Multigrid method for three levels in the hierarchy: $\ell = 0, 3, 5$

son's equation

$$-\Delta u = f \tag{5.1}$$

with a 7-point stencil on a $100 \times 100 \times 100$ grid. Figure 5.1 shows the sparsity pattern in classical multigrid hierarchy for solving (5.1). As can be seen that the bandwidth increases in the hierarchy. Table 5.1 shows that as the problem size decreases along the multigrid hierarchy, the average number of nonzeros increases dramatically. This decreased sparsity in coarse grid operators causes an increase in parallel communication costs, which leads to much longer solution time on coarse-levels than the time spent working on the original, fine-level problem. Figure 5.2 shows the total costs partitioned into the local computation cost and vector communication cost in multigrid hierarchies for solving (5.1). While having fewer nonezero entries, the increasing density on coarse level introduces extra communication cost which leads to increasing total cost.

level ℓ	matrix size n	nonzeros nnz	nonzeros per row nnz/n
0	1,000,000	6,940,000	7
1	4,190,209	9,320,600	19
2	83,338	2,775,206	35
3	13,363	745,531	67

Table 5.1: Matrix properties using classical Multigrid method for a 3D Poisson problem (5.1).

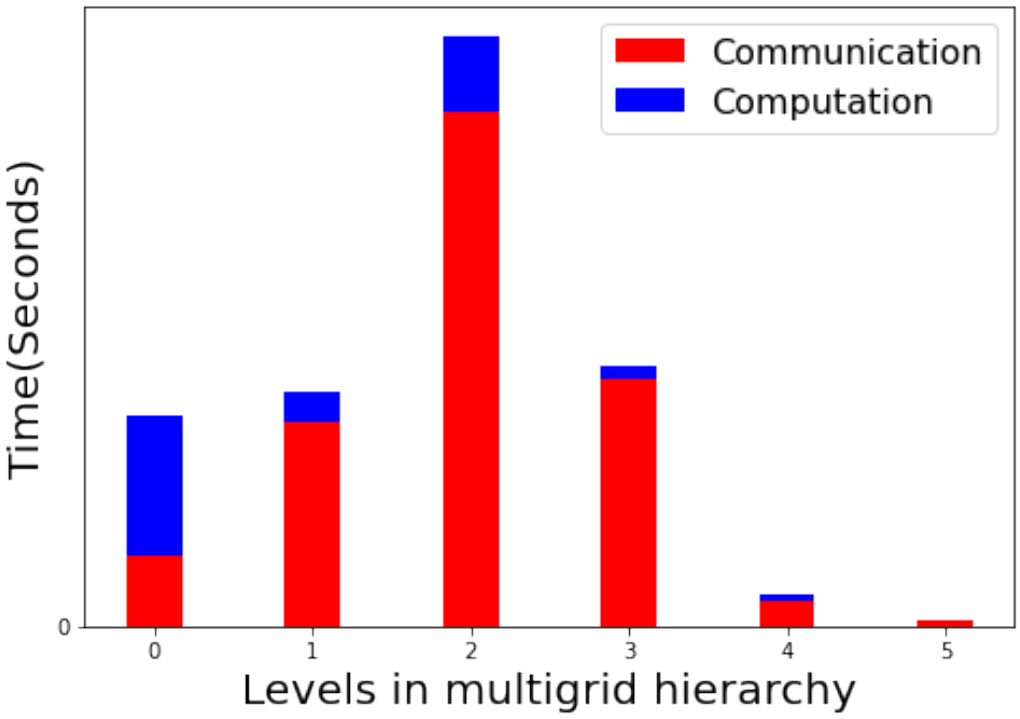


Figure 5.2: The local computation cost (in blue) and communication cost (in red) for smoothed aggregation multigrid hierarchies to solve (5.1) [6].

5.1.1 Theoretical considerations

One approach to address the decreasing sparsity in multigrid methods is to simply remove a few nonzeros in coarse grid operators. However, if some essential entries are removed, the convergence can deteriorate. In this section, we review a few theoretical results which study the convergence of multigrid method with sparsified coarse grid operators. In this thesis we will focus on sparsifying stencils in stencil-based problems. In the rest of this chapter, we will use stencils to define problems and construct sparsifications, we will use matrices to compute eigenvalues and eigenvectors. To avoid confusion with stencils and matrices, we will use A to denote matrices and use $\mathcal{A}$ to denote corresponding stencils.

Let A be the matrix for the fine grid problem and P be the prolongation operator, denote $P^{\mathsf{T}}AP$ by A_g and the sparsified A_g by A_c. Consider the two-grid error

propagation operator E_{G} of the form

$$E_{\mathrm{G}} = (I - M^{-1}A)(I - PA_g^{-1}P^{\mathsf{T}}A)(I - M^{-1}A),$$

where M is the smoother, $P \in \mathbb{R}^{n \times n_c}$ is the prolongation operator. Replace A_g with A_c we have

$$E_{\mathrm{C}} = (I - M^{-1}A)(I - PA_c^{-1}P^{\mathsf{T}}A)(I - M^{-1}A).$$

Spectral equivalence between A_c and A_g is often used as the metric to study the convergence of E_{C}. The definition of spectral equivalence between two stencils is given as follows.

Definition. Let $\{A_j\}_j$ and $\{B_j\}_j$ be two sequences of SPD matrices associated with $\mathcal{A}_g$ and $\mathcal{A}_c$ respectively with increasing size of $N_j \times N_j$. If all the eigenvalues of $B_j^{-1}A_j$ satisfy:

$$0 < \alpha < \lambda(B_j^{-1}A_j) \leq \beta < \infty$$

for all j where α and β are mesh independent, then $\mathcal{A}_g$ and $\mathcal{A}_c$ are spectrally equivalent stencils.

Matrices associated with spectrally equivalent stencils have similar convergence behavior in iterative methods. Apparently, a sparser one will definitely have a smaller computational cost at each iteration.

The following theorem shows that a structured 9-point stencil is spectrally equivalent to a sparser 5-point stencil (see [7]).

Theorem 5.1.1 (Spectral equivalent stencils [7])**.** *Let*

$$\begin{bmatrix} c & b & c \\ a & -2(a+b)-4c & a \\ c & b & c \end{bmatrix}$$

be an arbitrary 9-point stencil in 2D such that the associated generating symbol has a unique single zero at the origin. The following associated 5-point stencil is spectral equivalent:

$$\begin{bmatrix} & b+2c & \\ a+2c & -2(a+b)-8c & a+2c \\ & b+2c & \end{bmatrix}$$

Theorem 5.1.1 shows the possibility of replacing a dense stencil with a sparse one without affecting the convergence of multigrid. In fact, we will show in Section 5.3 that our proposed method can reproduce almost the same results as shown in the theorem.

The next theorem shows that two-grid convergence rate can be analyzed in terms of the convergence rate of the standard Galerkin coarse grid and $\|I - A_c A_g^{-1}\|_2$, a term that measures the spectral equivalence between A_c and A_g.

Theorem 5.1.2 (Conditioning and spectral radius [22])**.** *Let* $B_{\mathrm{G}} = A(I - E_{\mathrm{G}})^{-1}$ *and* $B_{\mathrm{C}} = A(I - E_{\mathrm{C}})^{-1}$. *Define*

$$\phi = \|I - A_c A_g^{-1}\|_2. \tag{5.2}$$

Assume A_c *and* A_g *are both SPD, if* $\phi < 1$, *then:*

$$\kappa(B_{\mathrm{C}}^{-1}A) \leq (\frac{1+\phi}{1-\phi})\kappa(B_{\mathrm{G}}^{-1}A),$$

and

$$\rho(E_{\mathrm{C}}) \leq \max(\lambda_{\max}(B_{\mathrm{G}}^{-1}A) \cdot \frac{1}{1-\phi} - 1, 1 - \lambda_{\min}(B_{\mathrm{G}}^{-1}A) \cdot \frac{1}{1+\phi}).$$

The sparsification algorithm proposed in [22] is based on Theorem 5.1.2. It follows two separate phases. In the first phase, it initializes a sparsity pattern using the matrix graph of P and the fine grid operator A to construct sufficient connections in A_c in order to approximate the Galerkin matrix stencil. In the second phase, it uses

heuristic to minimize ϕ in (5.2) in order to preserve the spectral equivalence between A_g and A_c.

Although the condition $\phi < 1$ in Theorem 5.1.2 is not a sufficient condition for the two-grid method using A_c to converge, ϕ can still be used as a good metric to check the spectral equivalence. This is demonstrated in the next corollary.

Corollary 5.1.3. *If $\phi < 1 - \lambda_{\max}(B_{\mathrm{G}}^{-1}A)/2$, two-grid method with A_c converges. Moreover, if two grid method with A_g converges and $\phi < 1/2$, then two-grid method with A_c is guaranteed to converge.*

Corollary 5.1.3 gives a sufficient condition of the value (5.2) for convergence.

Finally, we provide an example to show a simple heuristic sparsification approach by dropping small entries in the dense stencil $\mathcal{A}_g$ and adding the dropped entries back to the center point might deteriorate the convergence. Note that although this simple approach guarantees the row sums of the corresponding matrices A_c and A_g are the same, as such A_c preserves the action of A_g on the constant vector, this might still fail to preserve the spectral equivalence.

The example comes from the following anisotropic diffusion problem:

$$-u_{xx} - \epsilon u_{yy} = f.$$

We use standard the standard five-point finite-difference stencil:

$$\mathcal{A} = \begin{bmatrix} & -\epsilon & \\ -1 & (2+2\epsilon) & -1 \\ & -\epsilon & \end{bmatrix}$$

Using semi-coarsening in the x direction only yields the coarse grid stencil:

$$\mathcal{A}_g = \begin{bmatrix} -\frac{\epsilon}{4} & -\frac{3}{2}\epsilon & -\frac{\epsilon}{4} \\ -\frac{1}{2}+\frac{\epsilon}{2} & (1+3\epsilon) & -\frac{1}{2}+\frac{\epsilon}{2} \\ -\frac{\epsilon}{4} & -\frac{3}{2}\epsilon & -\frac{\epsilon}{4} \end{bmatrix}. \tag{5.3}$$

Dropping the small values in $\mathcal{A}_c$ associated with ϵ results in

$$\mathcal{A}_c = \begin{bmatrix} & 0 & \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ & 0 & \end{bmatrix}. \tag{5.4}$$

However, it has been shown in [22] that the upper bound of $\lambda({A_c}^{-1}A_g)$ is mesh-dependent. When the mesh size is small, since the value (5.2) is close to 1, in theory, this indicates spectral in-equivalence. In practice, replacing the stencil 5.3 by the stencil 5.4 in multigrid even leads to divergence.

The ideal sparsified stencil $\mathcal{A}_c^{ideal}$ of $\mathcal{A}_g$ can be obtained via rediscretizing the problem on the coarse grid for $\mathcal{A}_g$:

$$\mathcal{A}_c^{ideal} = \begin{bmatrix} & -2\epsilon & \\ -\frac{1}{2} & (1+4\epsilon) & -\frac{1}{2} \\ & -2\epsilon & \end{bmatrix}.$$

As can be seen, the two small values -2ϵ still appear in the spectral equivalent stencil.

5.2 Sparsification with machine learning

In this section, we propose a machine learning framework to sparsify coarse grid operators without affecting the convergence too much. Given the coarse grid operator $\mathcal{A}_g$, the construction of the sparse version $\mathcal{A}_c$ follows two phases: 1) selecting the

locations of nonzero entries $\mathcal{A}_c$ and 2) calculating the values of the nonzero entries. We utilize one neural network for each phase. The framework of sparsifying a nine point stencil to a five point stencil is illustrated in Figure 5.3.

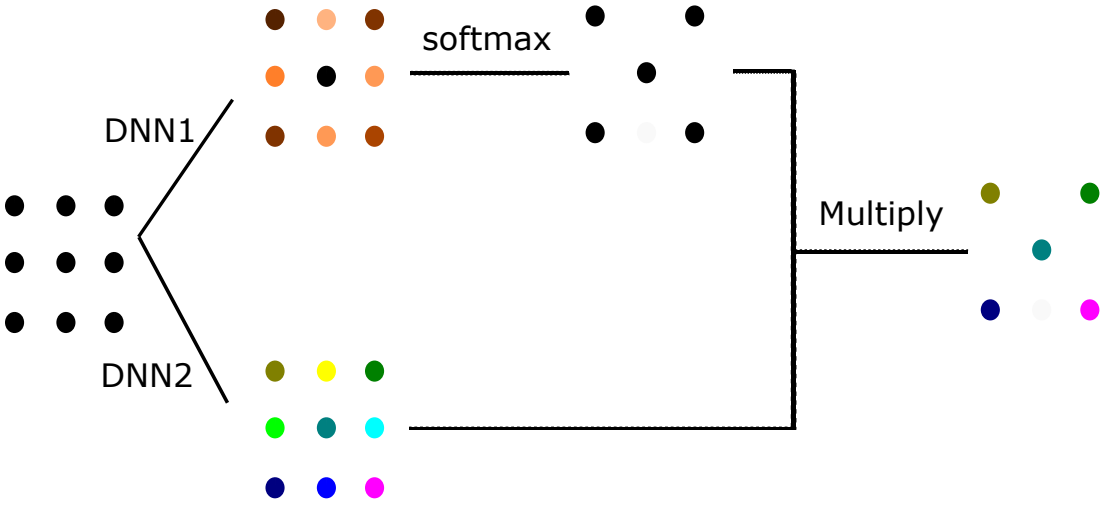


Figure 5.3: The framework of learning a five point stencil from a nine point stencil using two neural networks.

The first neural network learns a nine point stencil where each entry of the stencil represents the probability of keeping the value at the location. Set values at the locations with highest probabilities to 1 and the values at remaining locations to 0. This results in a mask matrix $Z \in \{0,1\}^{3\times 3}$. The second neural network learns the values $\mathcal{Y}$ of the stencil. Then the sparse stencil $\mathcal{A}_c$ is computed as $\mathcal{Y} \odot Z$.

Note that when measuring spectral equivalence, evaluating the value (5.2) on a single mesh size is not enough, it is important that the value (5.2) is bounded mesh-independently. To demonstrate this issue, we take the Poisson's equation with the 5-point stencil (5.5) as an example:

$$\mathcal{A} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}. \tag{5.5}$$

The stencil on the coarse grid is:

$$\mathcal{A}_g = \frac{1}{16}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{bmatrix}. \tag{5.6}$$

Rediscretizing the problem on the coarse grid yields

$$\mathcal{A}_c^{ideal} = \frac{1}{16}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}. \tag{5.7}$$

Let a five point stencil be defined as:

$$\mathcal{A}_c = \begin{bmatrix} 0 & -0.2615 & 0 \\ -0.1622 & 0.7329 & 0 \\ 0 & -0.1493 & -0.1599 \end{bmatrix}. \tag{5.8}$$

The value ϕ (5.2) for different grid sizes is shown in Table 5.2. We can see that the value ϕ for (5.7) remains almost the same while the value ϕ for (5.8), smaller than the value of (5.7) for certain grid sizes, increases rapidly as grid size increases.

Grid size	15	31	63	127
$\|I - A_c^{ideal} A_g^{-1}\|_2$	0.7451	0.7487	0.7496	0.7499
$\|I - A_c A_g^{-1}\|_2$	0.4155	0.5588	0.9953	1.9429

Table 5.2: The value (5.7) computed by A_g (5.6), A_c^{ideal} (5.7) and A_c (5.8).

As described in Chapter 4, when applying machine learning techniques, to reduce training cost, we train models on problems of small grid sizes and use the models on problems of larger grid sizes. Therefore, as we can see from Table 5.2, minimizing (5.2) on problems of size 15×15 and 31×31 might lead to a spectral in-equivalence

result. Instead of using (5.2) as the loss function, we describe a more appropriate approach based on multigrid convergence theory. Recall that the goal of sparsification is to replace the dense coarse grid operator A_g with one sparse one A_c without affecting the convergence of multigrid. This can be done by enforcing similar accuracy of A_c and A_g for the so called *algebraic smooth basis vectors* as explained in the next.

Consider at current multigrid step, the residual on fine grid is $r = f - Ax$, assume the residual r is in the range of P with $r = Pr_c$ for some coarse vector r_c. Then consider the two-grid method, if we use A_c instead of A_g to eliminate the residual r, we have

$$\begin{aligned} r_{\text{new}} &= (I - P(A_c)^{-1}P^{\mathsf{T}}A)r \\ &= (I - P(A_c)^{-1}P^{\mathsf{T}}A)Pr_c \\ &= P(I - (A_c)^{-1}A_g)r_c \\ &= P(A_c)^{-1}(A_c - A_g)r_c \end{aligned} \tag{5.9}$$

Ideally, if $A_c r_c = A_g r_c$, then the residual after the two-grid method is 0. Since A_c is sparser than A_g, it is not possible to require $A_c r_c = A_g r_c$ hold true for any vector r_c. Nevertheless, recall in multigrid methods, smoothers are applied to reduce the high frequency errors, the remaining errors are usually smooth. Hence in literature [14, 58], researchers suggest to define A_c such that it has similar behaviour with A_g when being applied to algebraic smooth vectors. Such construction of A_c can yield similar coarse grid corrections as A_g and the entire multigrid process will not be influenced due to the use of A_c instead of A_g.

The constructions of the algebraic smooth basis usually depend on problems. In this work we choose to use generalized eigenvectors associated with k smallest eigenvlues (k should be no less than the maximum nonzeros per row) as basis. These eigenvectors are related to the sought algebraically smooth modes ([14]). The procedure for constructing algebraic smooth basis is presented in Algorithm 3.

Algorithm 3 Generating algebraic smooth basis

Input: Prolongation operator P, coarse grid operator A_g, number of basis vectors: k

Output: Algebraic smooth basis $v_i, i = 1, \ldots, k$

1: Compute $T = P^\top P$
2: Compute generalized eigen pairs $A_g v_i = \lambda_i T v_i$.
3: Return $v_i, i = 1, \ldots, k$ with k smallest $\lambda_i, i = 1, \ldots, k$

Given a set of algebraic smooth vectors $v_i^{(j)}, i = 1, \ldots, k$ for each training instance, we minimize the loss function (5.10) to learn two neural networks f_θ and g_ψ which can generate the sparse stencil for unseen coarse grid stencil $\mathcal{A}_g$:

$$L(\{\mathcal{A}_g^j, \mathcal{A}_c^j, \{v_i^j\}_{i=1,\ldots,k}\}_{j=1\ldots,m}) = \sum_{j=1}^{m} \sum_{i=1}^{k} ||A_g^j v_i^j - A_c^j v_i^j||_2^2. \tag{5.10}$$

The entire learning procedure for sparsification is summarized in Algorithm 4.

Algorithm 4 Learning to sparsify

Input: Prolongation operator P, m coarse grid stencils $\mathcal{A}_g^j$ $(j = 1, \ldots, m)$, number of basis vectors: k, sparsification ratio ρ

Output: Two neural networks f_θ and g_ψ

1: Use Algorithm 3 to generate $v_i^j, i = 1, \ldots, k$ for each $\mathcal{A}_g^j$
2: Initialize two neural networks f_θ and g_ψ
3: $\{\mathcal{A}_c^j\}_{j=1,\ldots,m}$ =Sparsify$(\{\mathcal{A}_g^j\}_{j=1,\ldots,m}, f_\theta, g_\psi, \rho)$
4: Perform stochastic gradient descent (SGD) to minimize loss function:

$$\ell = \sum_{j=1}^{m} \sum_{i=1}^{k} ||A_g^j v_i - A_c^j v_i||_2^2$$

5: Update the weights of f_θ and g_ψ

Algorithm 5 Sparsify$(\mathcal{A}_g, f_\theta, g_\psi, \rho)$

1: Compute $\mathcal{Y} = f_\theta(\mathcal{A}_g)$ and $Z = g_\psi(\mathcal{A}_g)$
2: Set the top ρ percent largest values in Z be 1 and the remaining values to 0
3: Compute $\mathcal{A}_c = \mathcal{Y} \odot Z$
4: Return $\mathcal{A}_c$

5.3 Numerical Experiements

In this section, we provide numerical examples to demonstrate the ability of our proposed method for sparsifying stencils. All of the codes were implemented in PyTorch 1.8.1 and run on an Intel Core i7-6700 CPU. The neural network training took roughly 1 minute for each constant coefficient problem and roughly 5 minutes for linear elasticity problem.

5.3.1 Circulant stencil

We first use a small example to show that our proposed method can reproduce the results from Theorem 5.1.1.

We generate a random circulant matrix A corresponding to a 9-point stencil. We use full coarsening to obtain the coarse grid operator

$$\mathcal{A}_g : \begin{bmatrix} -0.0081 & -0.0095 & -0.0081 \\ -0.0062 & 0.0637 & -0.0062 \\ -0.0081 & -0.0095 & -0.0081 \end{bmatrix},$$

which satisfies the condition of Theorem 5.1.1. The spectrally equivalent stencil gby t is

$$\mathcal{A}_c^{ideal} : \begin{bmatrix} 0 & -0.0257 & 0 \\ -0.0224 & 0.0962 & -0.0224 \\ 0 & -0.0257 & 0 \end{bmatrix}$$

The stencil learned by our proposed method is

$$\mathcal{A}_c : \begin{bmatrix} 0 & -0.0231 & 0 \\ -0.0207 & 0.0874 & -0.0206 \\ 0 & -0.0230 & 0 \end{bmatrix}.$$

We can see that the proposed method can exactly capture the correct sparsity pattern and learns almost the same values. When using A_g, A_c^{ideal}, A_c as the coarse grid operator, two-grid method will exhibit exactly the same convergence behavior.

Next we test our proposed method on two classes of problems, the 2D anisotropic rotated Laplacian problem which we have described in the previous chapter and the 2D linear elasticity problem. To evaluate our method, similar as before, we compare the performance of using two-grid method with the dense coarse grid operator A_g and its coarse version A_c obtained by our method. We use the convergence threshold relative residual $\frac{\|f-A\hat{u}\|_2}{\|f\|_2} < 10^{-6}$ as the stopping criterion.

5.3.2 Rotated Laplacian

Recall the 2D anisotropic rotated Laplacian problem:

$$-\nabla \cdot (T\nabla u(x,y)) = f(x,y), \tag{5.11}$$

where 2×2 tensor field T is defined as

$$T = \begin{bmatrix} \cos^2\theta + \xi \sin^2\theta & \cos\theta \sin\theta(1-\xi) \\ \cos\theta \sin\theta(1-\xi) & \sin^2\theta + \xi\cos^2\theta \end{bmatrix} \tag{5.12}$$

We first fixed $\xi = 10$ and test the generalization ability of our proposed method when varying θ. We construct 4 training data points with $\theta = (\frac{3\pi}{12}, \frac{4\pi}{12}, \frac{5\pi}{12}, \frac{6\pi}{12})$. We use full coarsening scheme which produces a 9-point stencil on the coarse grid. We then use Algorithm 4 to train our model on problems of size 15×15 to generate sparse 5-point stencils. Then we use Algorithm 5 to test 10 problems of size 127×127 that are not shown in training set with $\xi = 10$ and random θ on each interval. The numbers of iterations for two-grid method to converge using $\mathcal{A}_g$ and $\mathcal{A}_c$ are summarized in Table 5.3. We can see that even when the coarse grid operator has nearly a half density as $\mathcal{A}_g$, the convergence is barely influenced.

$\xi = 10, \theta$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{4\pi}{12}, \frac{5\pi}{12})$	$(\frac{5\pi}{12}, \frac{6\pi}{12})$
$\mathcal{A}_g$	13.4	20.5	31.1
$\mathcal{A}_c$	13.9	20.6	31.1

Table 5.3: Averaged number of iterations required by two-grid method for solving (5.11) of size $n = 127^2$ with $\xi = 10$ and random θ on each interval using $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencils.

Then we fixed $\theta = \frac{\pi}{4}$ and test the generalization ability of our proposed method when varying ξ. We construct 4 training data points with $\xi = 0.1, 0.13, 0.16, 0.19$. We use the same training setting as before. The numbers of iterations for two-grid method to converge using $\mathcal{A}_g$ and $\mathcal{A}_c$ are summarized in Table 5.4. We can see that for $\xi \in (0.1, 0.13)$, even the iteration number associated with $\mathcal{A}_c$ is significantly larger than that of $\mathcal{A}_g$, it is acceptable since $\mathcal{A}_c$ only has nearly a half of the number of nonzeros compared to $\mathcal{A}_g$.

$\theta = \frac{\pi}{4}, \xi$	(0.1,0.13)	(0.13,0.16)	(0.16,0.19)
$\mathcal{A}_g$	13.1	11.4	10.2
$\mathcal{A}_c$	17.7	11.8	10.4

Table 5.4: Averaged number of iterations required by two-grid method for solving (5.11) of size $n = 127^2$ with $\theta = \frac{\pi}{4}$ and random ξ on each interval using $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencils.

We also compute the values $\phi = \|I - A_c A_g^{-1}\|_2$ (5.2) for each problem and report the results in Tables 5.5 and 5.6. All of the values are less than 1 indicating a similar spectrum between the learned stencil and the original stencil.

$\xi = 10, \theta$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{4\pi}{12}, \frac{5\pi}{12})$	$(\frac{5\pi}{12}, \frac{6\pi}{12})$
ϕ	0.6152	0.1581	0.3157

Table 5.5: The value ϕ (5.2) for $\xi = 10$ and random θ on each interval.

$\theta = \frac{\pi}{4}, \xi$	(0.1,0.13)	(0.13,0.16)	(0.16,0.19)
ϕ	0.3257	0.3363	0.3592

Table 5.6: The value ϕ (5.2) for $\theta = \frac{\pi}{4}$ and random ξ on each interval.

In particular, we take a specific problem with $\xi = 10$ and $\theta = \frac{\pi}{3}$ as an example. We show the eigenvalues of $A_c^{-1}A_g$ for different mesh sizes in Figure 5.4. We can see that the distribution of spectrum is mesh independent indicating spectral equivalence of A_c and A_g.

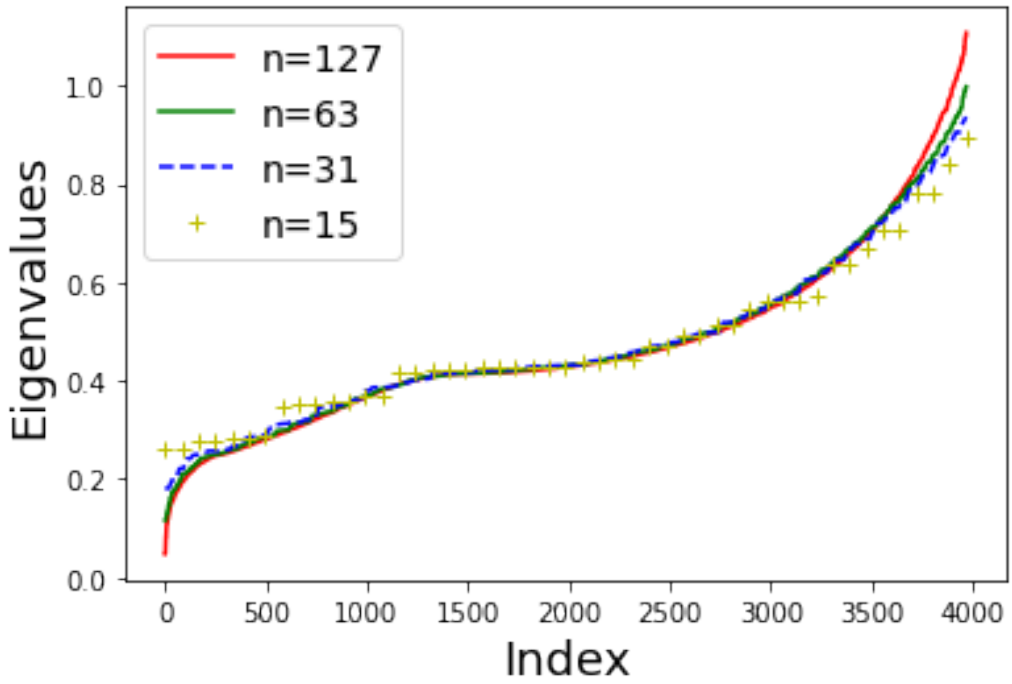


Figure 5.4: The eigenvalues of $A_c^{-1}A_g$ with different grid sizes for solving rotated Laplacian problem with $\xi = 10$ and $\theta = \frac{\pi}{3}$.

An example of the approximate solutions after the same number of iterations of using two-grid method with $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencil is shown in Figure 5.5.

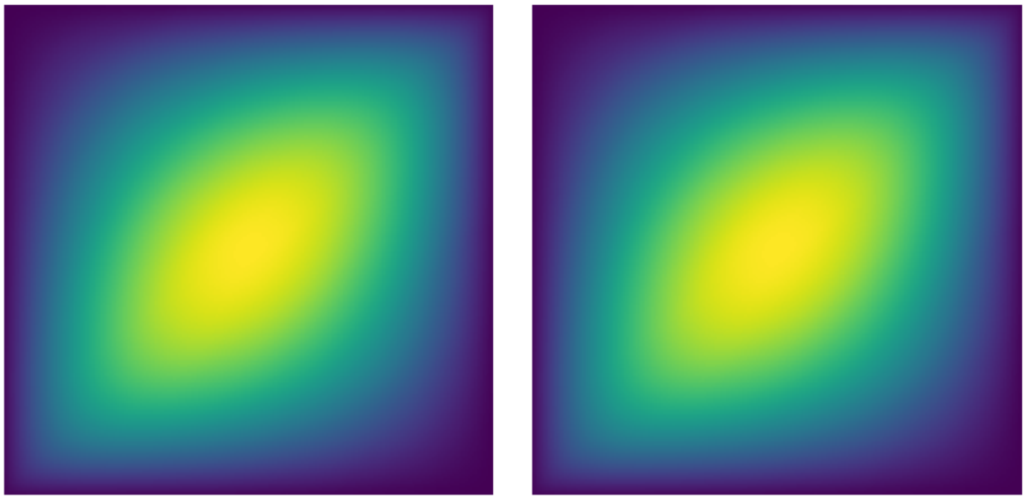


Figure 5.5: The approximate solutions (left: $\mathcal{A}_g$, right: $\mathcal{A}_c$) after 10 iterations of two-grid method with two coarse grid operators for solving rotated Laplacian problem with $\xi = 0.1147$ and $\theta = \frac{\pi}{4}$.

Finally we show in Figure 5.6 the convergence of using different number of nonzeros in $\mathcal{A}_c$ for the rotated Laplacian problem with $\xi = 10$ and $\theta = \frac{\pi}{4}$. Figure 5.6 shows that the convergence of twogrid method with 5 and 6 nonzeros in $\mathcal{A}_c$ is similar to that with $\mathcal{A}_g$. On the other hand, the convergence of tworid method with 4 or fewer nonzeros in $\mathcal{A}_c$ is much worse. This indicates 5 might be the minimum number required by $\mathcal{A}_c$ for this problem.

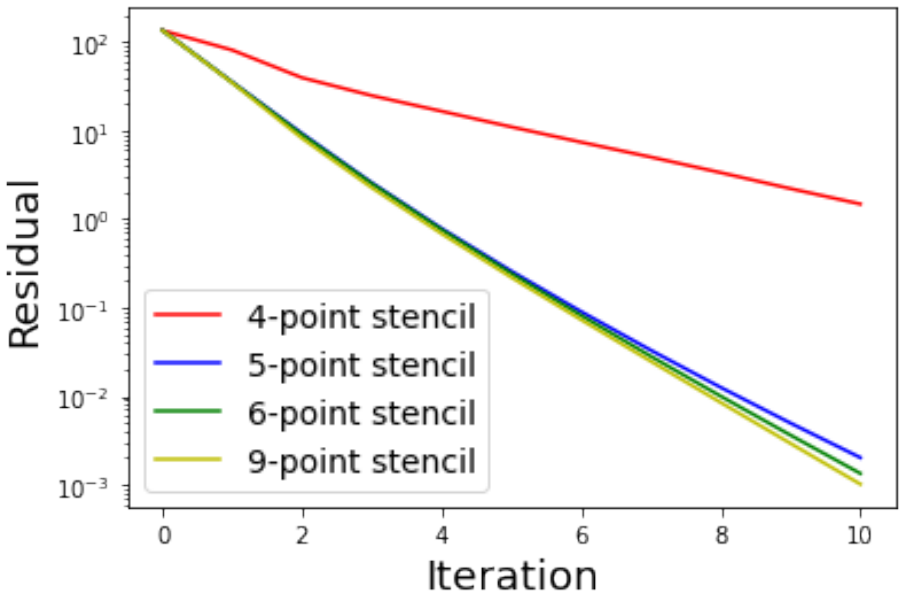


Figure 5.6: The residuals of using two-grid method with different coarse grid stencils for solving rotated Laplacian problem with $\xi = 10$ and $\theta = \frac{\pi}{4}$.

5.3.3 2-D elasticity problem

In this section we consider the 2-D linear time-independent elasticity problem. The problem in an isotropic homogeneous medium is described by the following PDEs:

$$\mu\nabla^2(u) + (\mu+\lambda)(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y}) + f_x = 0, \tag{5.13}$$

$$\mu\nabla^2(v) + (\mu+\lambda)(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y}) + f_y = 0, \tag{5.14}$$

where u and v are the solution of x and y directions respectively, f_x and f_y are forcing terms, μ and λ are Lame coefficients that are given by Young's modulus E and Poisson's ratio ν:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}.$$

We use the following 9-point discretization stencils with a mesh size h on x direc-

tion and mesh size $b_y h$ on y direction as described in [32].

$$\mathcal{A}_{uu} = \begin{bmatrix} \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{(b_y^2-1)\lambda + 2(b_y^2-2)\mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ -\frac{2\lambda b_y^2 + 4\mu b_y^2 + \lambda + \mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 1 & -\frac{2\lambda b_y^2 + 4\mu b_y^2 + \lambda + \mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{(b_y^2-1)\lambda + 2(b_y^2-2)\mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \end{bmatrix}$$

$$\mathcal{A}_{uv} = \begin{bmatrix} \frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 0 & -\frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ 0 & 0 & 0 \\ -\frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 0 & \frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \end{bmatrix}$$

Then the discretized linear system is

$$Ax = \begin{bmatrix} A_{uu} & A_{uv} \\ A_{uv}^{\mathsf{T}} & A_{uu}^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} \bar{u} \\ \bar{v} \end{bmatrix} = \begin{bmatrix} \bar{f}_x \\ \bar{f}_y \end{bmatrix}. \tag{5.15}$$

The prolongation operator for linear elasticity problem is also constructed blockwisely as:

$$\begin{bmatrix} P_{uu} & P_{uv} \\ P_{uv}^{\mathsf{T}} & P_{uu} \end{bmatrix},$$

and the restriction operator R is the transpose of the prolongation operator. We use red-black coarsening scheme for each block. The restriction and interpolation stencils for $u-u$ and $v-v$ connections associated with this coarsening scheme are given by

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

the restriction and interpolation stencils for $u - v$ connection are given by

$$\frac{1}{8}\begin{bmatrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{bmatrix},$$

and the restriction and interpolation stencils for $v - u$ connection are given by

$$\frac{1}{8}\begin{bmatrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{bmatrix}.$$

Note that as stated in [13], multigrid method requires to have one-row-sum for the $u-u$ and $v-v$ interpolation weights and zero-row-sum for the $u-v$ and $v-u$ weights to converge.

We choose Young's modulus $E = 10^{-5}$ and test the generalization ability of our method for various Poisson's ratio ν. We construct 4 training data points with $\nu = 0.1, 0.2, 0.3, 0.4$ of size 9×9 and choose the sparsification ratio $\rho = 0.5$. Note that for linear elasticity problems, we have four blocks of stencils on the coarse grid and two distinct stencils because of symmetry. We combine the two stencils before feeding into the neural networks so that the framework can learn a better balance between $u - u$ connection and $u - v$ connection rather than learning them separately. The numbers of iterations for two-grid method to converge using $\mathcal{A}_g$ and $\mathcal{A}_c$ for problems of size 65×65 are summarized in Table 5.7. We can see from the results that the convergence is barely influenced.

ν	(0.1,0.2)	(0.2,0.3)	(0.3,0.4)
$\mathcal{A}_g$	10.1	10.2	10.6
$\mathcal{A}_c$	11.0	10.7	11.5

Table 5.7: Averaged numbers of iterations required by two-grid method for solving (5.15) of size $n = 65^2$ with random ν on each interval using $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencils.

The values $\phi = \|I - A_c A_g^{-1}\|_2$ (5.2) for each problem are shown in Table 5.8. All of the values are less than 1.

ν	(0.1,0.2)	(0.2,0.3)	(0.3,0.4)
ϕ	0.7014	0.6889	0.6890

Table 5.8: The value ϕ (5.2) for $E = 10^{-5}$ and random ν on each interval.

We take a specific problem with $\nu = 0.3$ and $E = 10^{-5}$ as an example. We show the eigenvalues of $A_c^{-1} A_g$ for different mesh sizes in Figure 5.7. We can see that the distribution of spectrum is mesh independent indicating spectral equivalence of A_c and A_g.

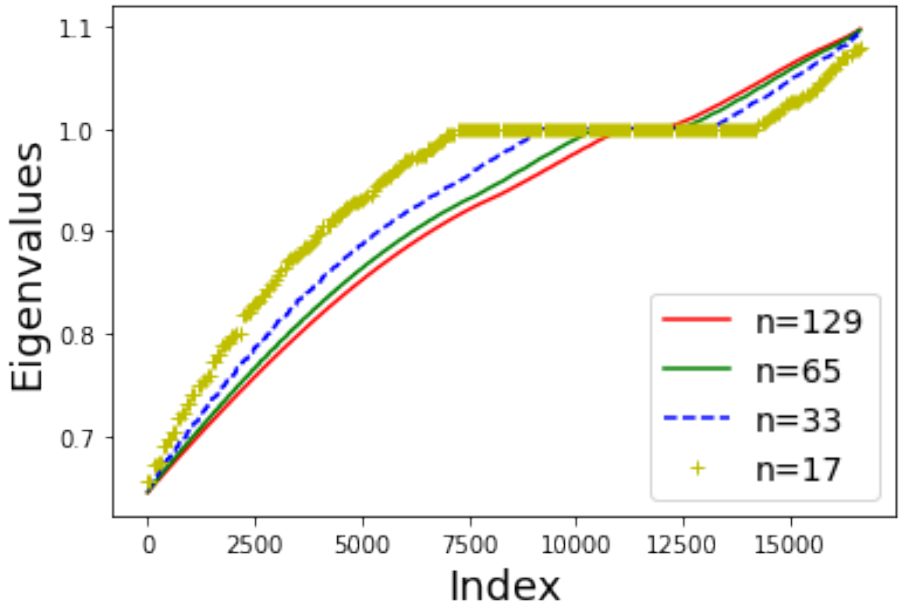


Figure 5.7: The eigenvalues of $A_c^{-1} A_g$ of different sizes for solving linear elasticity problem with $E = 10^{-5}$ and $\nu = 0.3$.

An example of the approximate solutions after the same number of iterations of using two-grid method with $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencils is shown in Figure 5.8.

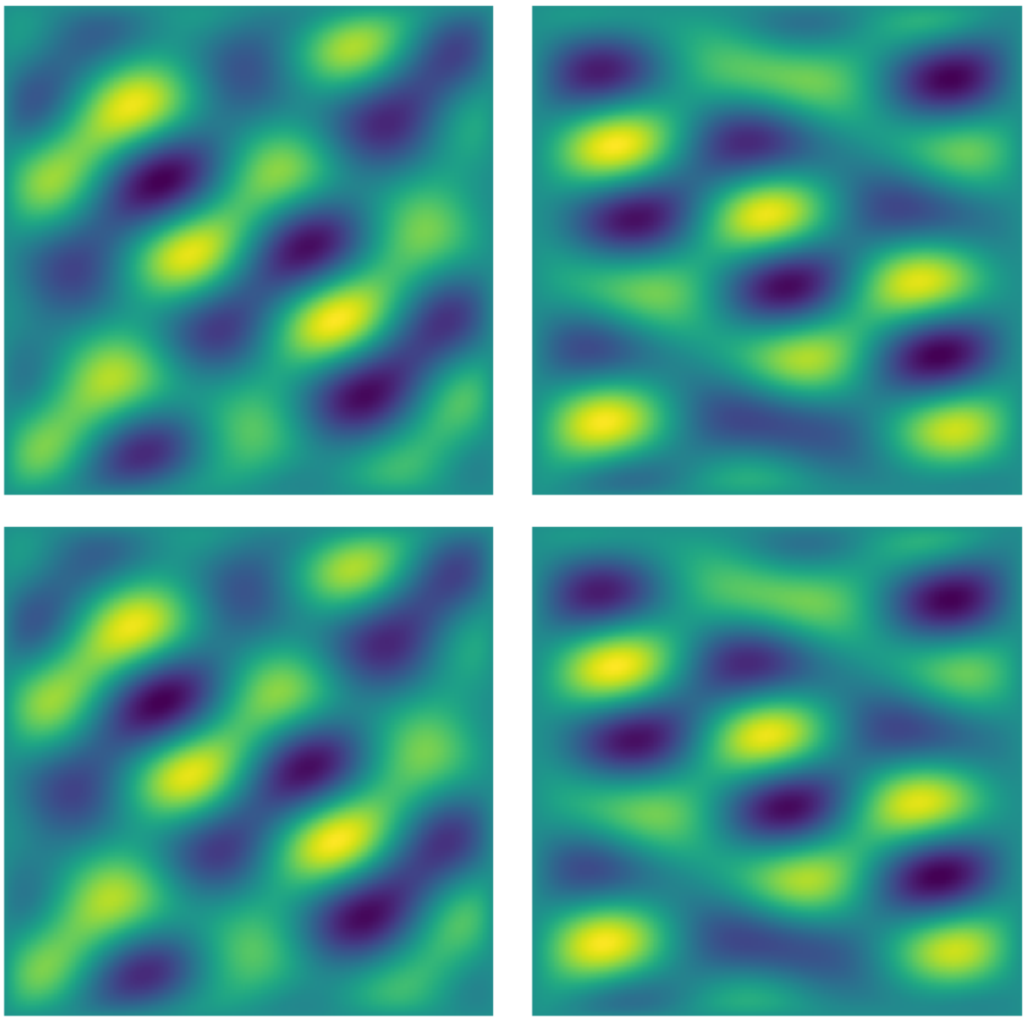


Figure 5.8: The approximate solutions (left: u, right: v, top: A_g, bottom: A_c) after 10 iterations of two-grid method with two coarse grid operators for solving linear elasticity problem with $E = 10^{-5}$ and $\nu = 0.1409$.

5.4 Conclusion

In this chapter we propose a machine learning framework for sparsifying the coarse grid operator A_g in multigrid methods to improve parallel efficiency. The sparsifi-

cation process has two major steps: choosing the sparsity pattern and deciding the values. For each step we utilize one neural network and combine the results from the two steps to construct A_c which has fewer number of nonzeros than A_g. We minimize the error of applying A_g and A_c on algebraic smooth basis during training so that multigrid with A_c, though being sparse, has similar convergence as that with A_g.

Experiments on rotated Laplacian problems and linear elasticity problems show that our framework can sparsify the coarse grid operator by a factor of 2 while the iteration number for multigrid method to converge remains almost the same.

Chapter 6

Conclusions and Future Work

Multigrid methods are one of the most efficient methods for solving large-scale sparse linear systems arising from discretized PDEs. Error propagation operator (6.1) of two grid is often used for convergence analysis:

$$E_{\mathrm{TG}} = (I - M^{-1}A)(I - P(P^{\mathsf{T}}AP)^{-1}P^{\mathsf{T}}A). \tag{6.1}$$

In this thesis, we propose deep learning frameworks to learn the smoother M and the coarse grid operator $A_g = P^{\mathsf{T}}AP$ which are two key components in (6.1). In particular, we parameterize M^{-1} by multi-layered CNNs and propose an adaptive framework guided by multigrid convergence theories. The training process of the proposed algorithm, called α-CNN, can learn the smoothers which have the desired smoothing property of reducing high frequency errors and lead to better convergence compared to classical smoothers. We further improve the efficiency of multigrid methods by sparsifying A_g. We use one deep neural network to design the sparsity pattern and the other deep neural network to compute the values. Combining the sparsity pattern and the values leads to A_c which is a sparser version of A_g. Multigrid methods with trained A_c, being sparser, have similar convergence compared to multigrid methods with A_g and therefore have better efficiency.

In this thesis, we have worked with stencil-based 2D problems on regular grids such as rotated Laplacian problems and linear elasticity problems. We can also extend the frameworks to 3-D problems or other more complicated problems. Since all the models and operations of learning can either be written as stencil-operations or be directly applied to stencils, a more general and more challenging situation to explore is to deal with irregular grids where stencils are not available, in which case general linear systems need to be considered. A straightforward idea is to treat the sparse linear systems as sparse graphs and utilize graph neural networks. Furthermore, convergence theories need to be developed when combining deep learning with multigrid. To conclude, lots of inspiring and exciting work can be done to combine multigrid methods with deep neural networks in the future.

Bibliography

[1] C. C. Aggarwal et al., *Neural networks and deep learning*, Springer, 10 (2018), pp. 978–3.

[2] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, *Multigrid smoothers for ultraparallel computing*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2864–2887.

[3] R. E. Bank and C. C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM Journal on Numerical Analysis, 22 (1985), pp. 617–633.

[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] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM Journal on Scientific Computing, 38 (2016), pp. S332–S357.

[6] A. Bienz, W. D. Gropp, and L. N. Olson, *Reducing communication in algebraic multigrid with multi-step node aware communication*, The International Journal of High Performance Computing Applications, 34 (2020), pp. 547–561.

[7] M. Bolten and A. Frommer, *Structured grid amg with stencil-collapsing for d-level circulant matrices*, (2007).

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

[9] A. Brandt, *Algebraic multigrid (amg) for sparse matrix eqations*, Sparsity and its Applications, (1984), pp. 257–284.

[10] A. Brandt, *Algebraic multigrid theory: The symmetric case*, Applied Mathematics and Computation, 19 (1986), pp. 23–56.

[11] 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, 1985, pp. 257–284.

[12] 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 Journal on Scientific Computing, 22 (2001), pp. 1570–1592.

[13] 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 Journal on Scientific Computing, 22 (2001), pp. 1570–1592.

[14] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (α sa)*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1896–1920.

[15] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, 2000.

[16] 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).

[17] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, *Multi-level residual networks from dynamical systems view*, arXiv preprint arXiv:1710.10348, (2017).

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

[19] H. De Sterck, U. M. Yang, and J. J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1019–1039.

[20] L. Erlandson, D. Cai, Y. Xi, and E. Chow, *Accelerating parallel hierarchical matrix-vector products via data-driven sampling*, in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 749–758.

[21] D. J. Evans and W. S. Yousif, *The explicit block relaxation method as a grid smoother in the multigrid v-cycle scheme*, International Journal of Computer Mathematics, 34 (1990), pp. 71–78.

[22] R. D. Falgout and J. B. Schroder, *Non-galerkin coarse grids for algebraic multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C309–C334.

[23] R. D. Falgout and P. S. Vassilevski, *On generalizing the algebraic multigrid framework*, SIAM Journal on Numerical Analysis, 42 (2004), pp. 1669–1693.

[24] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp, *Modeling the performance of an algebraic multigrid cycle on hpc*

platforms, in Proceedings of the international conference on Supercomputing, 2011, pp. 172–181.

[25] D. Greenfeld, M. Galun, R. Basri, I. Yavneh, and R. Kimmel, *Learning to optimize multigrid pde solvers*, in International Conference on Machine Learning, PMLR, 2019, pp. 2415–2423.

[26] T. Gudmundsson, C. S. Kenney, and A. J. Laub, *Small-sample statistical estimates for matrix norms*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 776–792.

[27] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, *Learning across scales—multiscale methods for convolution neural networks*, in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[28] J. Han, A. Jentzen, and E. Weinan, *Solving high-dimensional partial differential equations using deep learning*, Proceedings of the National Academy of Sciences, 115 (2018), pp. 8505–8510.

[29] J. He and J. Xu, *Mgnet: A unified framework of multigrid and convolutional neural network*, Science china mathematics, 62 (2019), pp. 1331–1354.

[30] P. Holl, V. Koltun, and N. Thuerey, *Learning to control pdes with differentiable physics*, arXiv preprint arXiv:2001.07457, (2020).

[31] J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, and S. Ermon, *Learning neural PDE solvers with convergence guarantees*, in International Conference on Learning Representations, 2019.

[32] A. Idesman and B. Dey, *Compact high-order stencils with optimal accuracy for numerical solutions of 2-d time-independent elasticity equations*, Computer Methods in Applied Mechanics and Engineering, 360 (2020), p. 112699.

[33] A. Katrutsa, T. Daulbaev, and I. Oseledets, *Deep multigrid: learning prolongation and restriction matrices*, arXiv preprint arXiv:1711.03825, (2017).

[34] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks, 9 (1998), pp. 987–1000.

[35] 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.

[36] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, *Learning to drop: Robust graph neural network via topological denoising*, in Proceedings of the 14th ACM International Conference on Web Search and Data Mining, 2021, pp. 779–787.

[37] I. Luz, M. Galun, H. Maron, R. Basri, and I. Yavneh, *Learning algebraic multigrid using graph neural networks*, in International Conference on Machine Learning, PMLR, 2020, pp. 6489–6499.

[38] S. Mishra, *A machine learning framework for data driven acceleration of computations of differential equations*, arXiv preprint arXiv:1807.09519, (2018).

[39] E. Nathan, G. Sanders, D. A. Bader, et al., *Numerically approximating centrality for graph ranking guarantees*, Journal of computational science, 26 (2018), pp. 205–216.

[40] Y. Notay, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM journal on scientific computing, 34 (2012), pp. A2288–A2316.

[41] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic multigrid solvers in Python v4.0*, 2018. Release 4.0.

[42] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.

[43] J. W. Ruge, *Algebraic multigrid (amg) for geodetic survey problems*, in Prelimary Proc. Internat. Multigrid Conference, Fort Collins, CO, 1983.

[44] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, second ed., 2003.

[45] ——, *Iterative methods for linear systems of equations: A brief historical journey*, Brenner, SC, Shparlinski, I., Shu, C.-W., Szyld, DB (eds.), 75 (2020), pp. 197–216.

[46] Y. Saad and M. H. Schultz, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), p. 856–869.

[47] J. Schmitt, S. Kuckuk, and H. Köstler, *Optimizing geometric multigrid methods with evolutionary computation*, arXiv preprint arXiv:1910.02749, (2019).

[48] J. Sirignano and K. Spiliopoulos, *Dgm: A deep learning algorithm for solving partial differential equations*, Journal of computational physics, 375 (2018), pp. 1339–1364.

[49] 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.

[50] 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 International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003, vol. 1, IEEE, 2003, pp. 381–384.

[51] 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 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), IEEE, 2017, pp. 1–3.

[52] E. Treister and I. Yavneh, *Non-galerkin multigrid based on sparsified smoothed aggregation*, SIAM Journal on Scientific Computing, 37 (2015), pp. A30–A54.

[53] E. TREISTER, R. Zemach, and I. YAVNEH, *Algebraic collocation coarse approximation (acca) multigrid*, in 12th Copper Mountain Conference on Iterative Methods, 2012.

[54] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*, Elsevier, 2000.

[55] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, *Backpropagation-friendly eigendecomposition*, Advances in Neural Information Processing Systems, 32 (2019).

[56] A. J. Wathen, *Preconditioning*, Acta Numer., 24 (2015), pp. 329–376.

[57] S. Wei, X. Jin, and H. Li, *General solutions for nonlinear differential equations: a rule-based self-learning approach using deep reinforcement learning*, Computational Mechanics, 64 (2019), pp. 1361–1374.

[58] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM Journal on Scientific Computing, 31 (2009), pp. 3643–3660.

[59] G. Wittum, *On the robustness of ilu smoothing*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 699–717.

[60] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numerica, 26 (2017), p. 591–721.

[61] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, *Robust graph representation learning via neural sparsification*, in International Conference on Machine Learning, PMLR, 2020, pp. 11458–11468.

[62] D.-X. Zhou, *Universality of deep convolutional neural networks*, Applied and Computational Harmonic Analysis, 48 (2020), pp. 787–794.