

Neural Multigrid Architectures

Vladimir Fanaskov

Center for Design, Manufacturing, and Materials
Skolkovo Institute of Science and Technology
Moscow, Russia
vladimir.fanaskov@skoltech.ru

Abstract—We propose a convenient matrix-free neural architecture for the multigrid method. The architecture is simple enough to be implemented in less than fifty lines of code, yet it encompasses a large number of distinct multigrid solvers. We argue that a fixed neural network without dense layers can not realize an efficient iterative method. Because of that, standard training protocols do not lead to competitive solvers. To overcome this difficulty, we use parameter sharing and serialization of layers. The resulting network can be trained on linear problems with thousands of unknowns and retains its efficiency on problems with millions of unknowns. From the point of view of numerical linear algebra network’s training corresponds to finding optimal smoothers for the geometric multigrid method. We demonstrate our approach on a few second-order elliptic equations. For tested linear systems, we obtain from two to five times smaller spectral radius of the error propagation matrix compare to a basic linear multigrid with Jacobi smoother.

I. INTRODUCTION

In this article, we describe how neural networks can be used to solve a system of linear equations

$$Ax = b, \quad x \in \mathbb{R}^n, \quad b \in \mathbb{R}^n, \quad A \in \mathbb{R}^{n \times n}, \quad (1)$$

in a particular case when A results from the discretization of PDE. Since A is typically large and sparse, iterative methods are preferable to direct ones [1, Section 8]. It is known that arbitrary linear iterative method has a form

$$x^{(m+1)} = x^{(m)} + N \left(b - Ax^{(m)} \right), \quad \det(N) \neq 0, \quad (2)$$

where N is an approximate inverse of A , and m is iteration number [2, Section 2.2.2]. To solve equation (1), we represent N as linear neural network $\mathcal{N}(\omega)$ and tune parameters ω to improve convergence speed, i.e., to obtain as small spectral radius of error propagation matrix $I - NA$ as possible.

The architecture of $\mathcal{N}(\omega)$ is chosen so that it corresponds to a particular geometric multigrid solver. For the article to be self-consistent, we provide a brief review of multigrid techniques in Section II. The resulting network $\mathcal{N}(\omega)$ consists of convolutional layers and does not depend on a matrix of a linear operator so that it can be conveniently implemented and applied to a variety of linear problems. The architecture can be found in Section III.

The loss function that we use for unsupervised training is described in Section IV. It is the same function that was used in [3] for a black-box optimization of multigrid solvers.

Next, in Section V we explain, that the training of $\mathcal{N}(\omega)$ is not straightforward because A^{-1} is non-local, and we typically

train on small problems. Namely, it is not enough to find ω that results in the small value of loss function for a fixed grid (the number of grid points controls the size of the matrix A). In addition to that, one needs to present a mechanism that enlarges the network when the grid is refined. If this is not done, the performance of a solver based on realistic neural networks becomes arbitrary bad for a sufficiently fine grid. To overcome this, we use the serialization of layers. Serialization performs well for some architectures, but it does not completely resolve the problem.

Concrete architectures that we test and a baseline model can be found in Section VI.

As test linear operators, we use five-point, nine-point, and Mehrstellen discretizations of the Poisson equation in $D = 2$ as well as anisotropic problem and a problem with mixed derivatives. The description of equations and learning results can be found in Section VII. In short, we obtain about five times smaller spectral radius of the error propagation matrix on train set ($n = (2^5 - 1)^2$), and from two to four times smaller spectral radius on test set ($n = (2^{11} - 1)^2$).

We conclude with an overview of related works in Section IX and a short summary of the paper in Section X.

All results can be reproduced (see Section VIII for details).

II. MULTIGRID METHOD

We start by giving a short introduction to the multigrid method. Multigrid is a multilevel iterative method that solves linear system (1) with large sparse matrix A . Two components crucial to fast convergence are smoother, and restriction operators [4, Section 1.5.1].

The smoother is a cheap linear iteration (2) that effectively reduces error in a subspace $W \subset \mathbb{R}^n$. The overall efficiency and a subspace are controlled by the choice of matrix N , and the last condition is (2) ensures consistency.

The role of the restriction operator $P \in \mathbb{R}^{k \times n}, k < n$ is to perform dimension reduction. Ideally $W \perp \text{range}(P)$, so after smoothing $e^{(n+1)} \in \text{range}(P)$. That means we can project on $\text{range}(P)$, reduce the number of unknowns, and retain all information about the solution.

Having restriction and smoothing operators, we can con-

struct a two-grid cycle:

$$\begin{aligned}
x^{(n+1/3)} &= x^{(n)} + N \left(b - Ax^{(n)} \right), \\
(PAP^T) e^{(n+1/3)} &= P \left(b - Ax^{(n+1/3)} \right), \\
x^{(n+2/3)} &= x^{(n+1/3)} + P^T e^{(n+1/3)} \\
x^{(n+1)} &= x^{(n+2/3)} + N \left(b - Ax^{(n+2/3)} \right).
\end{aligned} \tag{3}$$

The first line in (3) corresponds to smoothing, the second line is a coarse-grid equation, the third line is an error correction, and the last line is a smoothing again. Scheme (3) is preferred compare to (1) because PAP^T is a $k \times k$ matrix, that is, it is smaller, meaning cheaper to invert.

A multigrid method is a two-grid cycle, applied recursively, i.e., the two-grid cycle is used to solve the second line in (3). This procedure is repeated until we reach a small enough matrix that can be inverted by direct methods, f.e., LU factorization.

In the case of a simplest geometric multigrid in $D = 1$, P is a convolution with stride 2, and a kernel $[1/2 \quad 1/4 \quad 1/2]$ (direct product of convolutions in higher dimensions). Smoother is chosen to be either some variant of damped Gauss-Seidel (first line) or damped Jacobi (second line) methods:

$$\begin{aligned}
x^{(n+1)} &= x^{(n)} + \omega L(A)^{-1} \left(b - Ax^{(n)} \right); \\
x^{(n+1)} &= x^{(n)} + \omega D(A)^{-1} \left(b - Ax^{(n)} \right),
\end{aligned} \tag{4}$$

where $D(A)$ is a diagonal part of A and $L(A)$ is a lower triangular (including the diagonal) part of A , and $\omega \in \mathbb{R}$ is chosen based on A .

Intuition about the role of smoothers and restriction operators can be gained in the simplest case of Poisson equation [5, Chapter 13]. It can be shown that Jacobi smoother averages error. As a result, error considered as a function on a fine grid lacks high-frequency components and, as a result, can be well represented on a coarser grid. This is achieved by convolution operator P , which coincides with a low-pass filter combined with subsampling.

III. MATRIX-FREE MULTIGRID ARCHITECTURE

To describe our architecture, we need to introduce a few matrices. For each level $k \geq 1$ we use A_k, P_k to describe matrix of linear operator and the restriction matrix. According to two-grid cycle (3) the following relation holds $A_{k+1} = P_k A_k P_k^T$. For level $k = 1$, matrix A_1 should be given either explicitly or as a linear operator, i.e., the black-box function that computes $A_1 x$ for any given x suffices.

On each level k we need to implement two-grid cycle (3) as neural network. There are four operations we need to consider: computation of residual $r_k \equiv b_k - A_k x_k$, restriction $P r_k$, prolongation (interpolation) $P^T e_k$, and smoothing $N_k r_k$.

The simplest operations are restriction and prolongation that can be considered convolution with at least one stride > 1 , and a transpose to this operation.

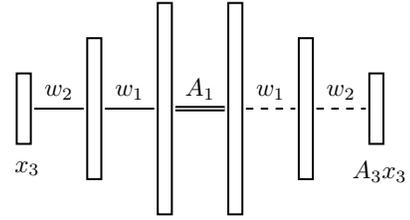


Fig. 1. Because $A_{i+1} = P_i A_i P_i^T$, product $A_3 x_3$ can be computed as a set of convolutions (dashed lines), and transposed convolutions (solid lines), and an application of operator on the fine grid (double line); w_i corresponds to convolution kernels.

Computation of the residual is straightforward too. Because $A_{k+1} = P_k A_k P_k^T$, any product can be computed recursively:

$$A_m x_m = \left(\prod_{l=m-1, \dots, 1} P_l \right) A_1 \left(\prod_{l=1, \dots, m-1} P_l^T \right) x_m. \tag{5}$$

This procedure is illustrated for $k = 3$ on Fig. 1.

The situation with smoothers is less straightforward. Not all smoothers can be considered in a matrix-free framework. For example, Gauss-Seidel smoother explicitly requires a lower triangular part of the matrix, which can be hard to extract. However, there is a family of smoothers, known as polynomial smoothers [6, Section 3], that are better suited for our purposes. Polynomial smoothers take a form

$$x^{(n+1)} = x^{(n)} + p(A) \left(b - Ax^{(n)} \right), \quad p(A) = \sum_{i=0}^D \alpha_i A^i, \tag{6}$$

where $\alpha_i, i = 0, \dots, D$ are parameters of the smoother chosen based on matrix A . Since (6) contains only vector-matrix products, we can apply the smoother using (5).

As a rule, polynomial smoothers are applied to the matrix with 1 on diagonal, i.e., the diagonal rescaling $D(A)^{-1}$ is explicitly introduced. We hide this additional factor in convolution operation.

Algorithm 1 specifies the smoothers that we use. In lines 2 and 5, $A_k(x)$ uses kernels w_i , fine-grid operator A and should be computed as in (5), $\text{conv}_{\tilde{w}_{i-1}}$ in line 4 should preserve the size of the input vector, so all strides equal 1.

To summarize, for a given level k , we implement a two-grid cycle (3) as a convolutional neural network with the following adjustments:

Algorithm 1 Polynomial smoothing.

- 1: **Input:** x_k, b_k , kernels $w_i, i = k-1, \dots, 1$ corresponding to convolutions $P_l, l = k-1, \dots, 1$, fine grid operator A , kernels $\tilde{w}_j, j = 0, \dots, D$ that are used to compute $p(A)$.

- 2: $r \leftarrow b_k - A_k(x)$
- 3: **for** $i = 1 : (D + 1)$ **do**
- 4: $x_k \leftarrow x_k + \text{conv}_{\tilde{w}_{i-1}}(r)$
- 5: $r \leftarrow A_k(r)$
- 6: **end for**

- 1) w_k, s_k are kernel and strides (at least one stride should be > 1) that implement convolution and transposed convolution that corresponds to P and P^T ;
- 2) $\tilde{w}_k^{(i)}, i = 0, \dots, D$ are kernels that are used in Algorithm 1 that substitutes the first and the last lines in (3);
- 3) all convolutions are with zero biases and without nonlinearities,
- 4) any matrix-vector product is computed according to (5) (see also Fig. 1).

The whole multigrid architecture can be constructed by recursive application of two-grid layers. To imitate matrix inversion on the coarsest grid, we use a few additional convolutions.

The presence of residuals makes it hard to draw the resulting architecture. But, in general, a neural network that imitates multigrid resembles U-Net [7]. The one crucial difference is that U-Net contains only one ascending and one descending branches, whereas our architecture contains an additional Λ -shaped network at each place where the residual $b_k - A_k x_k$ is needed.

Our approach offers the following advantages:

- Currently, no major machine learning framework supports sparse-sparse matrix multiplication, so PAP^T can not be computed efficiently.
- The architecture is agnostic to the sparsity pattern of A and P so that they can be changed easily. This can be especially useful when graph neural networks are used to learn the coarsening strategy.
- Since the network consists of convolution layers, one can benefit from using GPU.
- There is a one-to-one correspondence between some multigrid schemes and proposed architecture. This improves interpretability.

The main disadvantage is the additional operations we need to perform to compute $A_k x_k$. However, on modern GPUs, training on matrices with $n = (2^5 - 1)^2$ takes a few minutes, so the overhead seems to be justified by the overall convenience of the architecture.

IV. LOSS FUNCTION

To find optimal parameters of the neural network described in Section III, we introduce a loss function.

Let x^* be the exact solution to (1). It is known that for an arbitrary linear iterative method (2) with symmetric $I - NA$, the following is true

$$\left\| e^{(n+1)} \right\| \leq \rho(I - NA) \left\| e^{(n)} \right\|, \quad (7)$$

where $e^{(n)} = x^{(n)} - x^*$ is an error, $\rho(I - NA)$ is a spectral radius, and $\|\cdot\|$ is an arbitrary norm [2, Section 2.2.6].

Since neural multigrid architecture can be used as a linear iterative method, upper bound (7) suggests that $\rho(I - NA)$ is a good loss function.

Because $\rho(I - NA)$ is not readily available, it is a custom to use an approximation or upper bound to the spectral radius. Following [3], we use Gelfand formula [8], and stochastic trace

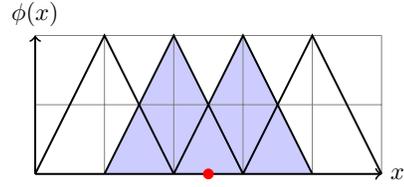


Fig. 2. When delta-function is presented as a right-hand side of a continuous problem, the weak form results in a sparse right-hand side because only a small number of (shaded) tent functions feels the presence of the source (denoted by a point).

estimation [9] to derive the following approximation to the spectral radius:

$$\rho(B) \simeq \rho_1(B, k, N_{\text{batch}}) \equiv \left(\frac{1}{N_{\text{batch}}} \sum_{j=1}^{N_{\text{batch}}} \|B^k z_j\|_2^2 \right)^{1/2k}, \quad (8)$$

where B is an arbitrary matrix, and each $z_j, j = 1, \dots, N_{\text{batch}}$ is a random vector with components i.i.d. according to Rademacher distribution. In all our experiments we use $k = N_{\text{batch}} = 10$.

V. RESTRICTION ON ARCHITECTURE FOR LINEAR ITERATIVE METHODS

Standard machine learning pipeline consists of choosing an appropriate architecture, training (supervised or unsupervised) with a given loss function, and applying trained model to unseen data [10, Chapter 11]. In this section, we argue that this approach is insufficient for training specific neural networks if we are to use them as iterative methods.

To make an argument, we consider the following boundary value problem:

$$\Delta u(x, y, z) = -\delta(x)\delta(y)\delta(z), \quad u(x, y, z)|_{x^2+y^2+z^2=R^2} = 0, \quad (9)$$

that is, a 3D Poisson equation with a point source at the origin, considered inside a sphere of radius R . The solution is easily obtained from Green function [11, Section 1.10]

$$u(x, y, z) = \frac{1}{4\pi} \left(\frac{1}{\sqrt{x^2 + y^2 + z^2}} - \frac{1}{R} \right). \quad (10)$$

One way to solve (9) numerically is to use finite element method (see [12] for introduction). For a suitable defined mesh (for example the mesh as in Fig. 3 can be used), we introduce a set of piecewise linear functions $\phi_i(x, y, z)$ that possess cardinality property: $\phi_i(x_j, y_j, z_j) = \delta_{ij}$, where (x_j, y_j, z_j) is a fixed grid point. The solution is approximated as $u(x, y, z) = \sum_i \phi_i(x, y, z) u_i$, and PDE is enforced in a weak form by Petrov-Galerkin condition $\int dx dy dz \phi_i(x, y, z) (\Delta u(x, y, z) + \delta(x)\delta(y)\delta(z)) = 0$. That gives us a system of linear equations (1) with sparse matrix and sparse right-hand side. The sparsity of the right-hand side is illustrated by Fig. 2.

Let \mathcal{U} be a space of functions on a finite 3D grid with spacing $\simeq H$, and \mathcal{N} be a linear neural network $\mathcal{U} \xrightarrow{\mathcal{N}} \mathcal{U}$ that

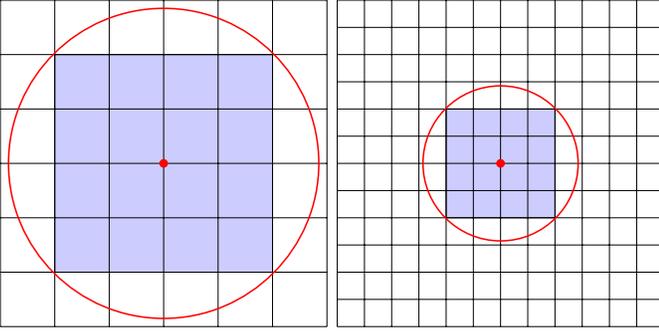


Fig. 3. The figure shows how the receptive field of fixed architecture with only local layers (shaded) changes after refinement. Since convolutions are performed on discrete data, information from the dot in the middle can spread over the smaller region (enclosed by the circle) in physical space. This limits the ability to generalize for a neural network with fixed architecture.

acts like linear operator on space \mathcal{U} . Let $u_k \in \mathcal{U}$ be a function equals 1 at point k and 0 at all other points. Because the grid is finite, it is possible to find a minimal radius R_k such that all nonzero elements of $\mathcal{N}(u_k)$ are inside the ball with radius R_k centered at point k . We define the radius of influence of a given network \mathcal{N} as

$$r_H(\mathcal{N}) = \max_k R_k.$$

The example of this radius is given in Fig. 3 for convolution with 5×5 kernel.

Now, if refinement is performed and the architecture of the network does not contain dense layers, the radius of influence shrinks as explained in the same Fig. 3. This fact is used to prove the following statement.

Proposition. *Let A_H be a matrix of linear problem (9) obtained using finite element method on a given grid with spacing $\simeq H$. Let \mathcal{N} be a neural network, that consists on finite number of (local) convolutional layers¹, and used as N in linear iterative method (2). Suppose that the network has been trained to provide a good convergence for grid H , that is, $\rho(I - \mathcal{N}_H A_H) = \epsilon \ll 1$. It is always possible to find a grid with spacing $\simeq h < H$ such that $\rho(I - \mathcal{N}_h A_h)$ is arbitrary close to 1.*

Proof. Without loss of generality we can assume that for grid H the radius of influence $r_H(\mathcal{N})$ is smaller than a grid size in a physical space, which is R for our problem. For $h = H/2^p$, $p > 1$ the radius of influence is $r_h(\mathcal{N}) = r_H(\mathcal{N})/2^p$. Let b_h be a discrete right hand side corresponding to a delta function in equation (9). If we start from zero initial guess $x^{(0)} = 0$, an estimation to the initial error in L_2 norm reads

$$h^3 \left\| e^{(0)} \right\|_2^2 \simeq 4\pi \int_0^R dr r^2 u(r)^2 = R/(12\pi), \quad (11)$$

¹We exclude nonlocal convolutions based on graph Laplacian as in [13].

and a lower bound on error for step $n = K$ ($Kr_h(\mathcal{N}) < R$) reads

$$\begin{aligned} h^3 \left\| e^{(K)} \right\|_2^2 &\geq 4\pi \int_{Kr_h(\mathcal{N})}^R dr r^2 u(r)^2 = \\ &= (R/(12\pi)) \left(1 - \frac{Kr_h(\mathcal{N})}{R} \right)^3. \end{aligned} \quad (12)$$

To derive equation (12), we assumed that our iterative method recovers the exact solution for all points that the network reached. Note that this argument is valid only because b_h is a sparse vector.

From (7) we conclude

$$\rho(I - \mathcal{N}_h A)^K \geq \left\| e^{(K)} \right\|_2 / \left\| e^{(0)} \right\|_2 \geq \left(1 - \frac{Kr_h(\mathcal{N})}{R} \right)^{3/2}. \quad (13)$$

□

Because r_h can be arbitrary small for sufficiently small $h = H/2^p$, the expression in the brackets above can be arbitrary close to 1, which signifies arbitrary slow convergence.

Remark 1. *The proposition above holds for networks that consist of (local) convolutional layers. We exclude networks with dense and nonlocal layers because they require $\simeq O(N^2)$ (N is a number of inputs) flops, which is unacceptable for iterative methods. On the other hand, convolutional neural networks require $\simeq O(N)$ flops and can be applied on grids with different sizes and geometries.*

Remark 2. *Our “radius of influence” is similar to the “domain of dependence” used to analyze convergence of numerical methods [14, Section 10.7]. Also, there is an evident parallel with CFL condition [15].*

Corollary. *Let A_H be a matrix of linear problem (9) obtained using finite element method on a given grid with spacing $\simeq H$. Let \mathcal{N} be a neural network, that consists of finite number of (local) convolutional layers. It is not possible to have $\|I - \mathcal{N}_h A_h\| \leq \epsilon \ll 1$, with ϵ independent on $h < H$. In other words, it is impossible to uniformly approximate inverses to A_h using fixed architecture with local layers.*

Proof. Since $\rho(I - \mathcal{N}_h A_h) \leq \|I - \mathcal{N}_h A_h\|$ for any matrix norm, the statement can be proven by contradiction. □

Remark 3. *It is crucial that operator A_h^{-1} is nonlocal. For example, A_h from the statement is equivalent to \mathcal{N} with a single convolutional layer.*

Remark 4. *It is known that neural network can approximate arbitrary nonlinear operator [16]. The corollary above does not contradict this result because it is restricted to neural networks with a finite number of layers.*

VI. ARCHITECTURES AND A BASELINE SOLVER

Architecture that we propose in Section III is a convolutional neural network. We want to train this architecture on small linear problems with a number of variables $n \simeq 2^{10}$ and apply it on large linear problems with $n \simeq 2^{20}$. According

to the result in the previous section, it is necessary to enlarge the network when we refine the grid. The simplest strategy is a serialization of layers. By serialization, we mean that an additional layer uses parameters from a previous layer. Here we formulate a few concrete architectures that we are going to compare in Section VII.

A. LMG

As a baseline model we use multigrid with linear interpolation and two pre-smoothing and two post-smoothing Jacobi sweeps (second line of equation (4)) with $\omega = 4/5$ (this ω is optimal for five-point discretization of Poisson equation in 2D [4, Section 2.1.2]). Linear interpolation means that P corresponds to convolution with strides $(2, 2)$ with the kernel

$$k_{\text{linear}} = \frac{1}{2} \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}. \quad (14)$$

B. s1MG(rs)

The name of the model derived from the fact that it is a neural multigrid (MG) architecture with a single serialized layer (s1), which contain adjustable restriction and smoothing operators (rs), with weights w and \tilde{w} respectively. To have the same number of floating-point operations as a baseline model, we use smoothing (Algorithm 1) with $D = 0$. Both w and \tilde{w} represents kernels of sizes 3×3 , which initially coincide with linear interpolation (14). Convolutional layer with kernel w has strides $(2, 2)$, and the layer with kernel \tilde{w} has strides $(1, 1)$. For this model, we use exact matrix inversion as a coarse-grid correction. This is possible because we can always stack enough layers to have a single unknown on a coarse grid for considered model problems.

C. s1MG(s)

This model is the same as the previous one but with two differences. First, the restriction operator is fixed to be linear interpolation (14), and only the smoothing operator is learned. Second, we explicitly incorporate diagonal rescaling with $D(A)^{-1}$ on each level. This is possible because restriction operators are fixed, so all diagonal can be computed in advance.

D. s3MG(s)

The model is the same as a previous one, but now we train three distinct layers with $\tilde{w}_1, \tilde{w}_2, \tilde{w}_3$. The serialization is performed as follows:

$$\begin{aligned} \text{layer 1 : } & \tilde{w}_1; \text{ layer 2 : } \tilde{w}_2; \text{ layer 3 : } \tilde{w}_3; \\ \text{layer 4 : } & \tilde{w}_1; \text{ layer 5 : } \tilde{w}_2; \text{ layer 6 : } \tilde{w}_3; \\ \text{layer 7 : } & \tilde{w}_1; \dots \end{aligned} \quad (15)$$

E. U-Net

This is an attempt to reproduce results from [17].² We use architecture presented in Fig. 4, but with 5 layers. U-Net is

²Which is nontrivial because the code is absent and the architecture of the model is unspecified. We cannot also use results from the article because they are scarce, and authors measure performance relative to the multigrid method, which they did not bother to describe in detail.

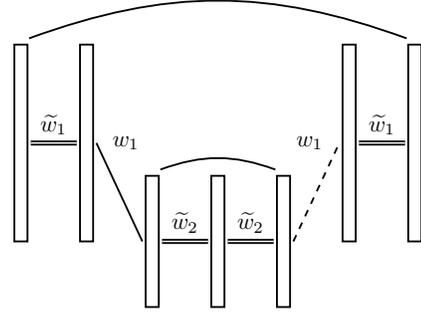


Fig. 4. U-Net architecture with two layers. Convolutions with all strides equal 1 are denoted by double lines (they correspond to smoothing in multigrid architecture), the single line represents convolution with at least one stride > 1 , dashed line is a transpose to this operation, a curved line is a skip connection (copy and add).

used as N in linear iteration (14). Parameters for all layers are distinct, and no serialization is performed.

F. fMG

This is another model without serialization. We use 5 layers with distinct restriction w and smoothing \tilde{w} operators. Two convolutions are used as a coarse grid correction. All kernels are initialised as bilinear interpolation (14).

VII. EXPERIMENTS

We start by defining the model equation and then comment on the performance of the models. All of the equations below correspond to the following boundary value problem

$$\begin{aligned} \left(a \frac{\partial^2}{\partial x^2} + b \frac{\partial^2}{\partial y^2} + c \frac{\partial^2}{\partial x \partial y} \right) u(x, y) &= f(x, y), \\ x, y \in (0, 1)^2 \equiv \Gamma, \quad u(x, y)|_{\partial\Gamma} &= 0, \end{aligned} \quad (16)$$

that is, a second-order equation with Dirichlet boundary conditions. For all discrete problems, we also perform a Jacobi preconditioning step $A \rightarrow D(A)^{-1/2} A D(A)^{-1/2}$ [18, Section 3.1].

A. Model equations

1) *Poisson equation*: Here $a = b = -1$, $c = 0$, and the corresponding kernels are

$$k_{P(5)} = \begin{bmatrix} 0 & -1/4 & 0 \\ -1/4 & 1 & -1/4 \\ 0 & -1/4 & 0 \end{bmatrix}, \quad (17)$$

$$k_{P(9)} = \begin{bmatrix} 0 & 0 & 1/60 & 0 & 0 \\ 0 & 0 & -4/15 & 0 & 0 \\ 1/60 & -4/15 & 1 & -4/15 & 1/60 \\ 0 & 0 & -4/15 & 0 & 0 \\ 0 & 0 & 1/60 & 0 & 0 \end{bmatrix}, \quad (18)$$

$$k_{P(M)} = \begin{bmatrix} -1/20 & -1/5 & -1/20 \\ -1/20 & 1 & -1/20 \\ -1/20 & -1/5 & -1/20 \end{bmatrix}, \quad (19)$$

which correspond to second-order, and two distinct fourth-order schemes. The last discretization is known as Mehrstellen

TABLE I
 $k_{p(5)}$, EQUATION (17), $\rho(I - \mathcal{N}A)$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.11	0.047	0.046	0.028	0.50	0.031
4	0.13	0.051	0.050	0.035	0.54	0.034
5	0.15	0.057	0.058	0.041	0.58	0.037
6	0.16	0.19	0.066	0.050	0.92	0.37
7	0.17	0.58	0.073	0.059	—	0.80
8	0.19	—	0.080	0.065	—	—
9	0.20	—	0.088	0.073	—	—
10	0.21	—	0.094	0.083	—	—
11	0.23	—	0.10	0.092	—	—

and can be used to construct sixth-order accurate discretization for sufficiently smooth right-hand side and boundary data [19].

2) *Anisotropic Poisson equation*: In this case $a = -\epsilon$, $b = -1$, $c = 0$ and the kernel reads

$$k_A = \begin{bmatrix} 0 & -1/(2+2\epsilon) & 0 \\ -\epsilon/(2+2\epsilon) & 1 & -\epsilon/(2+2\epsilon) \\ 0 & -1/(2+2\epsilon) & 0 \end{bmatrix}, \quad (20)$$

and we use $\epsilon = 2$ and $\epsilon = 10$.

3) *Mixed derivative*: Here $a = b = -1$ and $c = 2\tau$. The kernel is

$$k_M = \begin{bmatrix} -\tau/8 & -1/4 & \tau/8 \\ -1/4 & 1 & -1/4 \\ \tau/8 & -1/4 & -\tau/8 \end{bmatrix}, \quad (21)$$

and we test for $\tau = 1/4$ and $\tau = 3/4$.

B. Results

Results are gathered in Tables I–VII. Each table contains $\rho(I - \mathcal{N}A)$ approximated by equation (8) for a given architecture \mathcal{N} , J fixes the number of grid points along each direction $n_x = (2^J - 1)$, $n_y = (2^J - 1)$, and a total number of points $n = n_x n_y$. Symbol “—” means that $\rho_1((I - \mathcal{N}A), 10, 10) \geq 1$ (see (8))³. Each model that uses serialization applied with J layers, U-Net and fMG both contain ≤ 5 layers for all grids. The training is done for $J \leq 5$, then we test for $J \in [6, 11]$.

³This fact does not automatically mean that the actual spectral radius is greater than one. It might as well be merely close to one. In any case $\rho_1((I - \mathcal{N}A), 10, 10) \geq 1$ implies a significant deterioration of the solver.

TABLE II
 $k_{p(9)}$, EQUATION (18), $\rho(I - \mathcal{N}A)$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.16	0.058	0.069	0.042	0.53	0.030
4	0.22	0.063	0.073	0.041	0.58	0.031
5	0.25	0.070	0.079	0.049	0.62	0.038
6	0.28	0.081	0.088	0.086	—	0.54
7	0.30	0.33	0.096	0.11	—	0.89
8	0.32	0.82	0.10	0.12	—	—
9	0.35	—	0.11	0.13	—	—
10	0.37	—	0.12	0.14	—	—
11	0.40	—	0.13	0.15	—	—

TABLE III
 $k_{p(M)}$, EQUATION (19), $\rho(I - \mathcal{N}A)$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.073	0.030	0.036	0.022	0.40	0.017
4	0.094	0.034	0.041	0.028	0.44	0.019
5	0.11	0.041	0.048	0.02	0.60	0.022
6	0.12	0.13	0.058	0.042	0.96	0.52
7	0.13	0.40	0.065	0.051	—	0.99
8	0.14	0.78	0.071	0.057	—	—
9	0.15	0.99	0.077	0.063	—	—
10	0.16	—	0.083	0.069	—	—
11	0.17	—	0.090	0.076	—	—

1) *Poisson equation*: (Tables I–III) For all discretizations of the Poisson equation, we can see that architectures U-Net, fMG, and s1MG(rs) fail to provide a good solver for $J \geq 6$.

Presumably, the spectral radius of error propagation matrices corresponding to U-Net and fMG architectures deteriorates because neural networks have fixed sizes.

This explanation does not work for s1MG(rs) because of the serialization performed. We can conjure that because both restriction and smoothing operators are optimized, s1MG(rs) is getting tuned to the spectrum of the matrix with $n = (2^5 - 1)^2$, since the spectrum changes when J increases, the solver ceases to be efficient. It is evident from other examples that the naive serialization does not seem to work when both restriction and smoothing operators are optimized.

The only two solvers (besides a baseline model) that retain their efficiency are s1MG(s) and s3MG(s). The latter is better than the former for five-point (17) and Mehrstellen (19) discretizations, but for the long stencil (18) s1MG(s) is superior.

We can conclude that for the Poisson equation, U-Net is the weakest model, fMG and s1MG(rs) fail to generalize on the test set, and both s1MG(s) and s3MG(s) can generalize and outperform a baseline model on a test set.

2) *Anisotropic Poisson equation*: (Tables IV, V) For equation (20), the trend is largely the same. That is, fMG, s1MG(rs) and U-Net lose their efficiency, s1MG(s) and s3MG(s) are robust and outperform a baseline model.

It is instructive to discuss results for anisotropic equation with $\epsilon = 10$. First, we can see that all solvers are relatively inefficient. The reason is a full coarsening that we applied. If one uses semicoarsening instead, the results would be the

TABLE IV
 k_A , EQUATION (20), $\epsilon = 2$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.23	0.071	0.076	0.047	0.65	0.060
4	0.29	0.076	0.085	0.048	0.70	0.067
5	0.31	0.084	0.093	0.054	0.76	0.075
6	0.33	0.31	0.10	0.066	—	0.39
7	0.36	0.74	0.11	0.089	—	0.74
8	0.38	—	0.12	0.10	—	—
9	0.41	—	0.13	0.11	—	—
10	0.44	—	0.15	0.12	—	—
11	0.47	—	0.16	0.13	—	—

TABLE V
 k_A , EQUATION (20), $\epsilon = 10$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.59	0.40	0.44	0.42	0.91	0.44
4	0.73	0.42	0.47	0.42	0.99	0.47
5	0.81	0.49	0.53	0.49	—	0.53
6	0.88	—	0.59	0.53	—	—
7	0.94	—	0.63	0.57	—	—
8	—	—	0.68	0.61	—	—
9	—	—	0.73	0.65	—	—
10	—	—	0.78	0.70	—	—
11	—	—	0.83	0.75	—	—

same as for the isotropic Poisson equation. Because of the full coarsening, the U-Net solver fails already on a train set. If one further increases ϵ , our networks would not be able to provide efficient solvers unless strides are chosen appropriately. If strides and sizes of filters are considered as hyperparameters, it should be possible to apply Bayesian optimization [20], reinforcement learning [21], or genetic programming [22] to construct optimal solver.

3) *Mixed derivative*: (Tables VI-VII) Equation with mixed derivative changes type from elliptic to hyperbolic when τ crosses 1. It is interesting to look at how our models behave when τ approach 1.

For $\tau = 1/4$ architectures s1MG(s), s3MG(s) produces more efficient solvers than the standard multigrid with two Jacobi sweeps. On the other hand, U-Net is of no use even on the test set, and fMG and s1MG(rs) deteriorate rapidly for $J > 5$.

We can see that for $\tau = 3/4$ s3MG(s) performs substantially better than s1MG(s) on the train set. However, on the test set, it results in only a marginally smaller spectral radius. This means that the training and serialization strategies are not ideal. It should be possible to use additional parameters of s3MG(s) more efficiently. Other architectures behave similarly to the case $\tau = 1/4$.

VIII. REPRODUCIBILITY

To ensure complete reproducibility, we share a set of Jupyter notebooks that contain all models, linear equations, and training loops: <https://github.com/VLSF/nmg>.

TABLE VI
 k_M , EQUATION (21), $\tau = 1/4$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.11	0.040	0.049	0.029	0.49	0.043
4	0.14	0.049	0.056	0.032	0.54	0.048
5	0.15	0.061	0.065	0.037	0.58	0.054
6	0.17	0.27	0.073	0.045	0.88	0.34
7	0.18	0.67	0.081	0.052	—	0.78
8	0.19	—	0.088	0.062	—	0.98
9	0.21	—	0.095	0.072	—	—
10	0.22	—	0.10	0.083	—	—
11	0.24	—	0.11	0.092	—	—

TABLE VII
 k_M , EQUATION (21), $\tau = 3/4$

J	LMG	s1MG(rs)	s1MG(s)	s3MG(s)	U-Net	fMG
3	0.24	0.097	0.15	0.055	0.51	0.062
4	0.35	0.11	0.16	0.071	0.56	0.069
5	0.41	0.12	0.19	0.087	0.61	0.082
6	0.46	0.24	0.23	0.19	0.79	0.33
7	0.50	—	0.25	0.25	0.99	0.72
8	0.54	—	0.28	0.28	—	0.98
9	0.59	—	0.31	0.30	—	—
10	0.63	—	0.33	0.33	—	—
11	0.68	—	0.36	0.35	—	—

IX. RELATED WORK

Here we discuss a few related attempts to improve the multigrid method with machine learning tools. In the already mentioned paper [3], authors use stochastic gradient-based optimization to learn optimal multigrid solvers. This work roughly corresponds to architecture s1MG(rs), and our training strategy is exactly the same as in [3]. From the results, (Tables I–VII) we can conclude that simultaneous optimization of restriction and smoothing operators does not lead to a robust solver.

The other multilevel solver that we tried is a U-Net from [17]. As we pointed in Section VI, we cannot be sure that we reproduce results from [17] because the paper contains omissions. However, U-Net architecture from Fig. 4 fails on the test set and even unable to work on the train set for the anisotropic Poisson equation. It is not hard to see that U-Net architecture is just a slightly generalized filtering preconditioner [23]. Given that, the whole scheme from [17] is a generalized Richardson iteration for the preconditioned system. An optimal spectral radius for the preconditioner Richardson method is $(\kappa(NA) - 1) / (\kappa(NA) + 1)$, where $\kappa(NA)$ is a condition number, so to match multigrid $\kappa(NA)$ should be about 1.5. This means N should be much better than (optimal) Schwarz preconditioners for Poisson equation [24].

Two articles that firmly demonstrate that machine learning is a valuable tool for the construction of multigrid solvers are [25] (geometric multigrid), [26] (algebraic multigrid). In both cases, the authors take Gauss-Seidel smoother and focus on restriction weights. In some sense, our contribution is complementary because we focus on finding optimal smoothers and use bilinear interpolation as P^T . We can speculate that in both cases, scalable solvers are obtained in part because authors utilize ready-made coarsening strategies and robust smoother. Namely, in [25] a strategy from algebraic multigrid (AMG) is used to restore the solution on the fine grid in such a way that $b - Ax = 0$ for red points in a red-black pattern, and in [26] authors completely rely on AMG coarsening strategy. An attractive alternative would be to use kriging to perform coarsening as explained in [27].

Other related areas are Bootstrap AMG [28] and optimization based on local Fourier analysis [30], [29].

X. CONCLUSION

We introduce a convenient architecture that represents multigrid as a convolutional neural network. Using the simple 3D Poisson equation, we argue that the training of linear solver should be supplemented by a mechanism that enlarges the network's size. The simplest possible solution based on serialization of layers performs well, i.e., result in a robust solver competitive with a baseline model, but only for some architectures. Sadly, serialization does not work for the most promising architecture that combines optimization of smoothing and restriction operators. In our opinion, this problem can be solved either by a modification of the loss function or by an introduction of an additional mechanism that assembles multigrid based on pretrained layers. This is the focus of our current research.

REFERENCES

- [1] Y. Saad, "Iterative methods for linear systems of equations: A brief historical journey," in 75 years of mathematics of computation, ser. Contemp. Math. Amer. Math. Soc., Providence, RI, 2020, vol. 754, pp. 197–215.
- [2] W. Hackbusch, Iterative solution of large sparse systems of equations, 2nd ed., ser. Applied Mathematical Sciences. Springer, [Cham], 2016, vol. 95.
- [3] A. Katrutsa, T. Daulbaev, and I. Oseledets, "Black-box learning of multigrid parameters," J. Comput. Appl. Math., vol. 368, pp. 112–524, 12, 2020.
- [4] U. Trottenberg, C. W. Oosterlee, and A. Schüller, Multigrid. Academic Press, Inc., San Diego, CA, 2001, with contributions by A. Brandt, P. Oswald and K. Stüben.
- [5] Y. Saad, Iterative methods for sparse linear systems, 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003
- [6] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, "Parallel multigrid smoothing: polynomial versus Gauss-Seidel," J. Comput. Phys., vol. 188, no. 2, pp. 593–610, 2003.
- [7] 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, [Cham], 2015, pp. 234–241
- [8] V. Kozyakin, "On accuracy of approximation of the spectral radius by the Gelfand formula," Linear Algebra Appl., vol. 431, no. 11, pp. 2134–2141, 2009.
- [9] H. Avron and S. Toledo, "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix," J. ACM, vol. 58, no. 2, pp. Art. 8, 17, 2011.
- [10] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning, ser. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2016.
- [11] J. D. Jackson, Classical electrodynamics, 3rd ed. John Wiley & Sons, Inc., New York-London-Sydney, 1998.
- [12] P. G. Ciarlet, The finite element method for elliptic problems, ser. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002, vol. 40, reprint of the 1978 original.
- [13] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in International Conference on Learning Representations, CBL, 2014.
- [14] R. J. LeVeque, Finite difference methods for ordinary and partial differential equations. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.
- [15] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," IBM J. Res. Develop., vol. 11, pp. 215–234, 1967
- [16] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," IEEE Transactions on Neural Networks, vol. 6, no. 4, pp. 911–917, 1995.
- [17] 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.
- [18] A. J. Wathen, "Preconditioning," Acta Numerica, vol. 24, p. 329–376, 2015.
- [19] J. B. Rosser, "Nine-point difference solutions for Poisson's equation," in Computers and mathematics with applications, 1976, pp. 351–360.
- [20] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," Proceedings of the IEEE, vol. 104, no. 1, pp. 148–175, 2016.
- [21] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," Journal of Machine Learning Research, vol. 18, no. 185, pp. 1–52, 2018.
- [22] J. Schmitt, S. Kuckuk, and H. Köstler, "Constructing efficient multigrid solvers with genetic programming," in Proceedings of the 2020 Genetic and Evolutionary Computation Conference, ser. GECCO'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1012–1020.
- [23] C. H. Tong, T. F. Chan, and C.-C. J. Kuo, "Multilevel filtering preconditioners: extensions to more general elliptic problems," SIAM J. Sci. Statist. Comput., vol. 13, no. 1, pp. 227–242, 1992
- [24] X. Zhang, "Multilevel Schwarz methods," Numer. Math., vol. 63, no. 4, pp. 521–539, 1992.
- [25] D. Greenfeld, M. Galun, R. Basri, I. Yavneh, and R. Kimmel, "Learning to optimize multigrid PDE solvers," in Proceedings of the 36th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 2415–2423.
- [26] I. Luz, M. Galun, H. Maron, R. Basri, and I. Yavneh, "Learning algebraic multigrid using graph neural networks," in Proceedings of the 37th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 13–18 Jul 2020, pp. 6489–6499.
- [27] H. Gottschalk and K. Kahl, "Coarsening in algebraic multigrid using-gaussian processes," arXiv preprint arXiv:2004.11427, 2020.
- [28] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, "Bootstrap AMG," SIAM J. Sci. Comput., vol. 33, no. 2, pp. 612–632, 2011.
- [29] J. Schmitt, S. Kuckuk, and H. Köstler, "Optimizing geometric multigrid methods with evolutionary computation," ArXiv: 1910.02749, 2019.
- [30] R. Wienands and W. Joppich, Practical Fourier analysis for multigrid methods, ser. Numerical Insights. Chapman & Hall/CRC, Boca Raton, FL, 2005, vol. 4.