A SUPERVISED LEARNING APPROACH TO PREDICTING MULTIGRID CONVERGENCE

NICOLAS NYTKO[*]

In collaboration with: Matthew West, Luke Olson, Scott MacLachlan

Abstract. Classical AMG solvers often require careful parameter tuning to achieve optimal convergence, and the way these parameters affect performance can be unpredictable in practice. Evidence is presented that supervised learning techniques are able to learn certain characteristics of two-level multigrid solvers, particularly the rate of convergence and optimal relaxation weight for a given coarse/fine mesh splitting. Random perturbations of C/F splittings are generated and evaluated in a multigrid solver to train a convolutional neural network (CNN) in order to predict convergence and a relaxation weight for the 1D variable coefficient Poisson equation, and to predict the convergence rate for a specific 2D convection-diffusion problem. Additionally for the 2D problem, the use of graph nets is explored for use on general finite-element meshes.

Key words. convergence, relaxation weight, machine learning, supervised learning, graph net, cnn, convolutional network

1. Introduction.

Multigrid methods are some of the most widely used linear solvers for sparse systems, particularly ones that arise from the discretization of partial differential equations. By "downsampling" the original problem to a series of smaller and smaller problems, fast convergence can be obtained with minimal work [2]. Optimal "downsampling" is critical to the success of the method; classical Algebraic Multigrid (AMG) techniques attempt to create an interpolation operator by exploiting the structure of the original system [13]. However, these methods often require careful parameter tweaking to obtain a sensible operator and C/F splittings and often the best way to test the parameter space is to actually run the multigrid iterations.

In this paper, the following research question is addressed: can convergence rates of multigrid solvers be learned, and are neural networks an effective way of learning this attribute on the solver? The ability to predict this would aid in the use and study of classical AMG methods by allowing users to estimate efficacy ahead of time: for example being used to differentiate between different or the most optimal AMG methods for a specific problem. We address this question by introducing a method of generating data and training neural networks to predict convergence rates for a specific PDE or class of equations. Two problems are discussed here: (1) a 1D variable coefficient Poisson equation, and (2) a 2D recirculating flow convection-diffusion problem with specific parameters. Sections 2.1 and 2.3 describe how these random grid splittings are generated and evaluated in a multigrid solver.

For the 1D poisson case, convolutional neural networks (CNNs) are trained to predict the convergence rate and optimal relaxation weight (Section 2.2). For the 2D convection-diffusion problem, a CNN was first trained to predict convergence (Section 2.4), and then two different graph nets were trained to extrapolate to differently sized inputs and non-grid meshes (Sections 2.5, 2.6). Numerical results consisting of the network error performance and sample predictions are shown in Section 3. Finally, further discussion and possible directions of future work are given in Section 4.

[*]University of Illinois at Urbana-Champaign, (nnytko2@illinois.edu).

2. Methods. The methods described here will be split into two subsections describing each problem that was explored. First, the Poisson case and the data generation is discussed (Section 2.1) followed by the respective convolutional neural network that was trained (Section 2.2). Afterwards, an overview of the data generation for the 2D convection-diffusion problem is given (Section 2.3) and then followed by a description of the convolutional network that was trained (Section 2.4). The use of two graph nets is also described: one that performs a simple edge convolution for each node (Section 2.5), and another that implements "message passing" and learns optimal edge weighting of the convolution (Section 2.6).

2.1. 1D Poisson. Formally, the problem being solved is the Poisson equation in one dimension with variable coefficients

$$-\nabla\cdot\left(k\left(\boldsymbol{x}\right)\nabla\boldsymbol{u}\right)=f, \tag{2.1}$$

with the right-hand side $f\left(\boldsymbol{x}\right)$ being arbitrarily chosen. Eqn. (2.1) is discretized using finite differences on a grid of $N = 31$ internal points on the domain $\Omega = [-1, 1]$ and Dirichlet boundary conditions, $\partial\Omega = 0$. To preserve the symmetric, positive definite properties of the resulting linear system, the $k\left(\boldsymbol{x}\right)$ function is discretized on the grid *midpoints* [1].

Before the neural network can be trained, a dataset of approximately 300,000 random C/F grid splittings and their computed convergence rate and optimal relaxation weight was generated. A set of 6 reference grids were first created according to various "coarsening" factors:

$$r=\begin{Bmatrix}2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\end{Bmatrix}. \tag{2.2}$$

Each of the values in 2.2, say $r_i = j$ refers to a grid in which each jth point is coarse, and the rest fine. So, a coarsening by 3 is a grid that has roughly $\frac{1}{3}$ of its distribution of points as coarse and all others fine. Each of these reference grids was randomly permuted such that each grid point had a random probability of being flipped to the opposite value: coarse point to fine and fine point to coarse. For each reference grid, approximately 1000 random trials of each following probability were run:

$$p=\begin{Bmatrix}0.01 & 0.05 & 0.1 & 0.25 & 0.5 & 0.75\end{Bmatrix} \tag{2.3}$$

Additionally, each trial was run with a randomly-chosen function for the variable coefficients $k\left(\boldsymbol{x}\right)$. The function is randomly chosen from one of four functional forms

$$k\left(\boldsymbol{x}\right)=\begin{cases}\alpha & 0<\alpha<10\\ \text{rand}()\left(\alpha+1\right) & 0<\alpha<10\\ \alpha\cos\left(\pi x\beta\right)+\gamma & 0<\alpha<10, 0<\beta<10, \alpha<\gamma<10\\ \left|\sum_{i=1}^{5}\alpha_i x^i\right|+0.01 & -10<\alpha<10,\end{cases} \tag{2.4}$$

with coefficient values specifically picked to prevent the function from being non-positive at any value in Ω.

The full set of C/F splittings was then given to a 2 level V-cycle Multigrid solver that was run for a maximum of 15 iterations. The solver is composed of one round of weighted Jacobi pre-smoothing, a coarse error correction, then another round of Jacobi post-smoothing. At each iteration, the absolute error between the approximation and the "exact" solution (pre-computed via sparse linear solve) was found and saved:

$e_i = \left\| \boldsymbol{A}^{-1}\boldsymbol{f} - \boldsymbol{u}_i \right\|$. This sequence of errors was then used to compute the average convergence rate. To acquire the optimal relaxation weight, the smoother was run through a bracketed numerical optimization method with the assumption that the convergence rate is unimodal as a function of relaxation weight. [1]

2.2. Poisson CNN. Two separate deep convolutional networks with residual connections were trained to separately predict the optimal relaxation weight and convergence rate when given a linear C/F splitting. The architecture for both networks is identical and a textual overview is given here:

1. 6 1D CNN layers of kernel size 7, input 2 channels output 7 channels. Padding by three on each side to keep dimensions static.
2. 6 CNN layers of kernel size 5, input/output 7 channels. Padding by two on each side.
3. 6 CNN layers of kernel size 3, input/output 7 channels. Padding by 1 on each side.
4. Max-pooling layer of kernel 2, stride 2. Effectively reduces input size by half.
5. 8 Fully-connected layers to gradually reduce output to a scalar describing the convergence or relaxation weight.

Each layer is followed by an implicit ReLU nonlinear activation function. Because the convolutional layers (except for the first) keep the input and output size static, we are able to push residual values and skip layers similarly to a ResNet [8, 6]. Even numbered layers n take as input both the output of layers $n-1$ and $n-2$, while odd-numbered layers only use the output from layer $n-1$.

The C/F splitting is remapped such that a coarse point is given a value of 1 and a fine point the value of -1. The variable coefficients were also re-discretized to be defined on the nodal points instead of midpoints in order for the splitting and coefficients to be represented by vectors of same length. These two were then stacked into a two-channel tensor for input into the CNN. When training, input values (convergence rate and relaxation weight) are normalized to be within the range of $[0, 1]$. This normalization is undone when the output is displayed.

2.3. 2D Convection-Diffusion. This specific problem models what is called the *double glazing problem*, modeling the temperature distribution of a cavity with a single "hot" wall. This is given by the differential equation:

$$-k\nabla^2 u + \boldsymbol{w} \cdot \nabla u = f. \tag{2.5}$$

The wind velocity function, $\boldsymbol{w}(x, y)$ is defined as $\boldsymbol{w}(x, y) =$ $\begin{bmatrix} 2y\left(1 - x^2\right) & 2x\left(1 - y^2\right) \end{bmatrix}$. The domain is a square of side length two centered at the origin, $\Omega = [-1, 1] \times [-1, 1]$. Dirichlet boundary conditions are defined on $\partial\Omega$, with the one "hot" wall defined on $x = 1$ with value $\partial\Omega_H = 1$. The other boundaries are "cold" walls with $\partial\Omega_c = 0$. A diffusivity constant of $k = 0.1$ is used. This problem is derived from an example by Elman, Silvester, and Wathen [4].

This PDE is discretized using finite-elements on a structured grid of 25x25 internal points using the Firedrake software for FEM discretizations [3, 12, 7, 10]. Using a grid as a basis for the discretization allows use of both CNN and more sophisticated graph convolutional techniques.

Generating a dataset of mesh splittings and convergence rates was done in an overall similar way to the 1D Poisson equation with a few notable differences. Again,

[1]Experimental testing generally asserts this to be true, however this will remain a conjecture for now.

a set of reference C/F splittings were generated that are later permuted. For the convection-diffusion case, the following reference splittings were used:

1. All fine points
2. All coarse points
3. Splitting as given by Ruge-Stüben AMG ($\theta = 0.25$) [13, 11]
4. Coarsening in each direction by 2
5. Coarsening in each direction by 3
6. Coarsening in each direction by 4
7. Coarsening in each direction by 5

The entries of each individual reference splitting were then randomly permuted according to a defined probability. The probability values used are the same as those in the Poisson case, repeated here for convenience:

$$p = \begin{Bmatrix} 0.01 & 0.05 & 0.1 & 0.25 & 0.5 & 0.75 \end{Bmatrix}. \tag{2.6}$$

Generated splittings that are unsolveable (i.e., consist of no coarse points) were rejected and their trial re-run. This generated set of C/F splittings was passed along to another 2 level V-cycle multigrid solver run for a maximum of 50 iterations. The interpolation operator was formed by means of *direct interpolation*, the implementation of which gratiously taken from the PyAMG [11] library. This solver performs two rounds of Jacobi pre-and-post relaxation, with a coarse error correction between the relaxation steps. The absolute error between the approximation at each iteration and the "exact" solution was computed and saved, with the full sequence used to find the average convergence rate. Note the optimal Jacobi relaxation weight was not found here, as experimentation found that the optimal weight would nearly always be 1.

The convection-diffusion problem was additionally re-discretized at 4 different mesh sizes: 15×15, 25×25, 35×35, 50×50. The above process was re-run for each mesh size to generate a new dataset for use in the graph nets. To distinguish between the two datasets, they will hereby be referred to as the *statically-sized* (only containing 25×25 mesh and splittings) and the *variably-sized* datasets.

2.4. Convection-Diffusion CNN. A 2D convolutional network was trained on the statically-sized dataset to predict convergence when given a C/F splitting for the specific recirculating flow problem. Since the input parameters are slightly less complex, a less deep (*shallower*, if you will) network was trained:

1. 3 2D CNN layers of kernel 7, input two channels output 7 channels. Padding by three on each side to keep dimensions.
2. 3 CNN layers of kernel 5, input/output 7 channels. Padding by two on each side.
3. 3 CNN layers of kernel 3, input/output 7 channels. Padding by 1 on each side.
4. 2D Max-pooling layer of kernel 2, stride 2. Effectively reduces input size by half.
5. 1 Fully-connected layer to reduce output to a scalar predicting convergence rate.

Each layer is followed by a ReLU nonlinear activation function. Odd layers are fed the output of the previous two layers, while even layers are fed the input of only the previous layer, similarly to a ResNet [8, 6]. C/F splittings are mapped so that coarse points have value 1 and fine points have value -1. The CNN thus has a $25 \times 25 = 625$-length vector as input. Interestingly, normalization of the convergence rates is unneeded as the minimum and maximum recorded values are already close to

0 and 1, respectively.

2.5. Convection-Diffusion Graph Convolutional Network (GCN). The main downside of using traditional convolutional layers is that they are useful only on structured, grid-like inputs. Grid-based convolution is ineffective on the more complex meshes that may arise from finite-element discretizations. By treating the input mesh as a graph and using graph-based convolution techniques, we may get around this roadblock. The first graphnet that was tried uses the GCN layer introduced by Kipf and Welling [9] and is described here, the other uses an Edge-Conditioned Convolution (ECC)/message passing layer and is described in Section 2.6.

For the graphnets, the sparse FEM system was re-interpreted as a graph by assigning each row/column to be a node and defining connectivity between nodes as nonzero matrix entries. Edge weights were taken to be the entry values themselves, although normalized to be within the range of $[0, 1]$. This network was trained on the variably-sized dataset; as will be seen, the lack of any fully-connected layers allows for any arbitrary input.

The Graph Convolutional Network (GCN) layer is defined as an operator on the graph Laplacian, using the following propagation rule:

$$\boldsymbol{H}^{(i)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(i-1)}\boldsymbol{W}^{(i)} + \boldsymbol{b}^{(i)}\right). \tag{2.7}$$

With $\boldsymbol{H}^{(i)}$ being the i'th hidden layer, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ the adjacency matrix with added self loops, $\tilde{d}_{ii} = \sum_j \tilde{a}_{ij}$ a diagonal matrix consisting of the sum of outgoing edge weights, $\boldsymbol{W}^{(i)}$ the weight matrix at layer i, $\boldsymbol{b}^{(i)}$ a learned bias vector with same shape as $\boldsymbol{H}^{(i)}$, and $\sigma(\cdot)$ some nonlinear activation function. Hidden layers have dimensions $n \times f$, with n the number of graph nodes and f the number of features at each node. Thus, using only GCN layers it is not possible to reduce the number of rows in hidden layers (or the input layer for that matter).

Assume $\sigma(\cdot) = \text{ReLU}(\cdot) = \max\{\cdot, 0\}$ for all activation functions. Let $\boldsymbol{X}$ be the $n \times 1$ vector containing C/F splitting values for each node. The network that was trained for this application consisted of the following architecture:

$$\begin{aligned}
\boldsymbol{H}^{(1)} &= \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right) \in \mathbb{R}^{n\times 2}\\
\boldsymbol{H}^{(2)} &= \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}\right) \in \mathbb{R}^{n\times 3}\\
\boldsymbol{H}^{(3)} &= \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(2)}\boldsymbol{W}^{(3)} + \boldsymbol{b}^{(3)}\right) \in \mathbb{R}^{n\times 2}\\
\boldsymbol{H}^{(4)} &= \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(3)}\boldsymbol{W}^{(4)} + \boldsymbol{b}^{(4)}\right) \in \mathbb{R}^{n\times 1}\\
\boldsymbol{R} &= \begin{bmatrix} \cdots & \boldsymbol{X}^T & \cdots \\ \cdots & \left(\boldsymbol{H}^{(1)}\right)^T & \cdots \\ \cdots & \left(\boldsymbol{H}^{(2)}\right)^T & \cdots \\ \cdots & \left(\boldsymbol{H}^{(3)}\right)^T & \cdots \\ \cdots & \left(\boldsymbol{H}^{(4)}\right)^T & \cdots \end{bmatrix} \in \mathbb{R}^{9\times n}\\
y &= \sum_{j=1}^{n} \sigma\left(\sigma\left(\boldsymbol{r}_j\boldsymbol{W}^{(5)} + \boldsymbol{b}^{(5)}\right)\boldsymbol{W}^{(6)} + \boldsymbol{b}^{(6)}\right) \in \mathbb{R}
\end{aligned}$$

Model	Metric	Dataset	Value
Jacobi Weight	MSE	Training	1.8331×10^{-3}
Jacobi Weight	MSE	Testing	1.8396×10^{-3}
Jacobi Weight	L1	Training	2.9254×10^{-2}
Jacobi Weight	L1	Testing	2.9271×10^{-2}
Convergence Factor	MSE	Training	1.4839×10^{-3}
Convergence Factor	MSE	Testing	1.5171×10^{-3}
Convergence Factor	L1	Training	2.3908×10^{-2}
Convergence Factor	L1	Testing	2.3865×10^{-2}

Table 3.1: Final training and testing Mean Squared Error (MSE)/L^1-norm loss values for the two Poisson models. Lower values correspond to higher model accuracy.

Note that the matrix $\boldsymbol{R}$ is formed whose columns are the propagation history of each node; this is an attempt to emulate traditional ResNet architectures. The final scalar output is computed by summing each column of $\boldsymbol{R}$ through a two layer fully-connected neural network, with the first layer taking as input 9 features and outputting 5 features, and the second layer taking as input 5 features and outputting 1 feature. This local transformation of each nodal value following by a global averaging removes any dependency for a fixed input size – instead any sized graph and splitting could (in theory) be used.

2.6. Convection-Diffusion Message Passing Network (MPNN). A downside of the GCN layer, while being simple to implement and understand, is that it does not effectively learn edge features between nodes and thus is only an approximation of a convolution on a graph. The Edge-Conditioned Convolution (ECC) layer, as defined by Simonovsky and Komodakis [14], attempts to combat this by learning optimal edge weights given an arbitrary node-edge neighborhood. This approach is also referred to as a "message-passing network", as each connected neighbor attempts to learn a "message" that is passed to the original node [5]. The convolution operation is defined as

$$\left(\boldsymbol{H}^{(i)}\right)_j = \frac{1}{|\mathcal{N}(j)|} \sum_{k \in \mathcal{N}(j)} F^{(i)}\left(e_{j,k}\right) \left(\boldsymbol{H}^{(i-1)}\right)_k + \boldsymbol{b}^{(i)} \tag{2.8}$$

where $\boldsymbol{H}^{(i)} \in \mathbb{R}^{n \times f_i}$ is the i'th hidden layer, $\mathcal{N}(i)$ is a map returning the neighborhood of vertex i (including itself), $F^{(i)} : E \mapsto \mathbb{R}^{f_i \times f_{i-1}}$ is some function (in our case a small fully-connected neural network) that outputs a learned weight matrix given an edge, and $\boldsymbol{b}^{(i)}$ is a bias term for layer i. The notation $\left(\boldsymbol{H}^{(i)}\right)_j$ here denotes the j'th row of the i'th hidden layer.

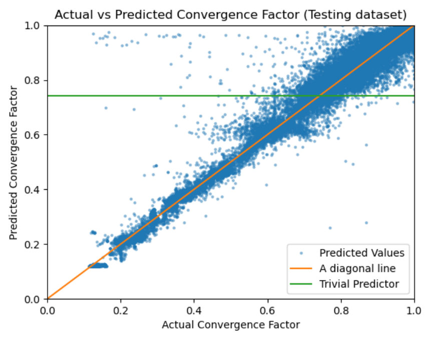
Using the ECC layer, another graph net was trained to predict the convergence rate given some C/F mesh splitting. The architecture is identical to that in Section 2.5 with the exception that the propagation layers $\boldsymbol{H}^{(i)} : \; i = 1 \ldots 4$ are replaced with (2.8) and surrounded with a ReLU activation layer. Each $F^{(i)}$ is a two-layer fully-connected network taking some edge $\boldsymbol{e} \in \mathbb{R}^{f_i}$ and outputting a weight matrix $\boldsymbol{\Theta} \in \mathbb{R}^{f_i \times f_{i-1}}$.

3. Numerical Results.

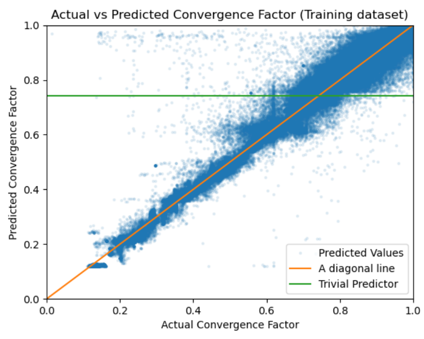
3.1. 1D Poisson. A total of 336,000 C/F "grid" splittings were randomly generated for the variable-coefficient Poisson model. Of these, a random 85% of data items were designated as a training set and the remaining 15% reserved for a testing set. The relaxation and convergence CNNs were trained for a total of 30 epochs each, where each epoch is a total pass over all training set entries. Models were trained using an MSE loss, and final loss values for both networks are detailed in Table 3.1.

Overall, the resulting data is encouraging and seems to indicate that both convergence factors and relaxation weights for the Poisson problem are predictable using a convolution-based network. Sample convergence rate predictions on the training and testing subsets are shown in Figure 3.1; these plots display predicted rates as a function of the true rates, so error is indicated by deviation from the diagonal of the plots. It is interesting to visually see that the network has an easier time predicting more convergent grids ($c < 0.6$ or so) than more degenerate grids. A lack of predictions less than ≈ 0.1 is a deficiency of the input data not having grids with those convergence values.

Predictions for relaxation weights on both training and testing subsets are shown in Figure 3.2. Like the previous convergence plots, more accurate predictions are closer to the plot diagonals. There is a positive correlation between the predictions and the true values, though unlike the convergence rates there is no obvious "split" in the prediction data. The lack of predictions below 0.1 is likely to also be caused by the input data.



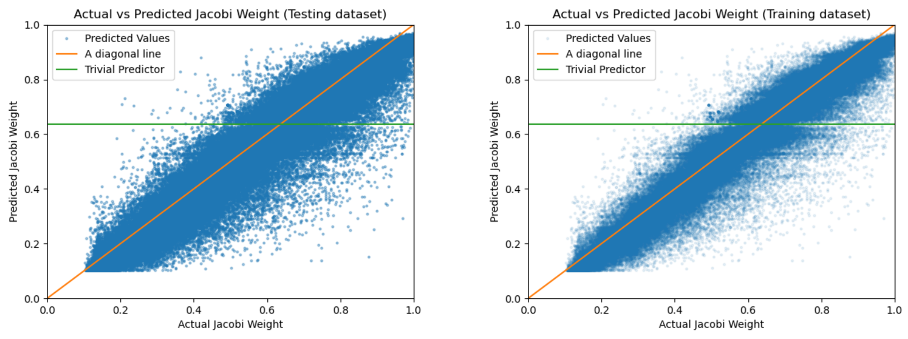
(a) Testing predictions



(b) Training predictions

Fig. 3.1: Predicted convergence values vs. true convergence values on (3.1a) testing and (3.1b) training datasets for Poisson equation. Values closer to the diagonal represent more accurate predictions.

3.2. 2D Convection-Diffusion. Two datasets were created for the 2D recirculating flow problem: a set of *statically-sized* C/F splittings and a set of *variably-sized* splittings; both datasets consisted of 84 000 elements. The variably-sized dataset was evenly divided into the four mesh sizes: 15×15, 25×25, 35×35, 50×50. Each dataset was then split 85%-15% into training and testing subsets. All neural networks were trained for 10 epochs each on an MSE loss, loss history for both datasets is displayed in Figure 3.3. The recirculating flow CNN was trained and tested on the static dataset, while the two graph nets were trained on the variable dataset and

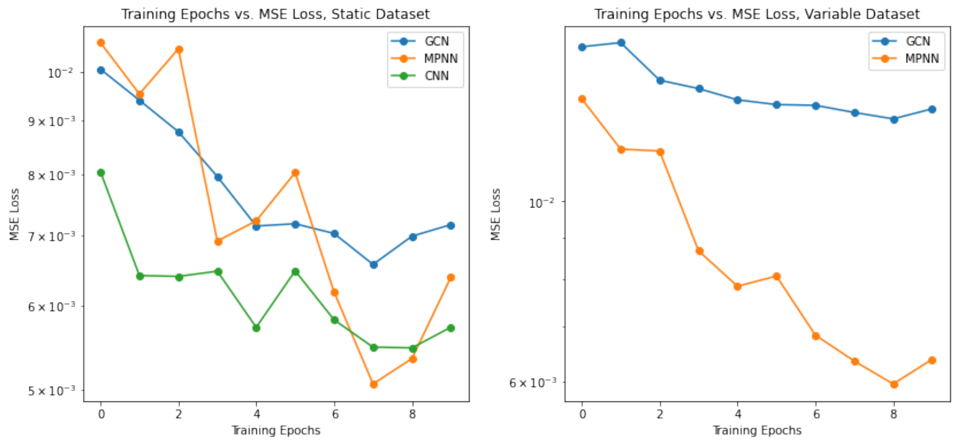


(a) Testing predictions (b) Training predictions

Fig. 3.2: Predicted relaxation weights vs. true relaxation weights on (3.2a) testing and (3.2b) training datasets for Poisson equation. Values closer to the diagonal represent more accurate predictions.

evaluted on both datasets.

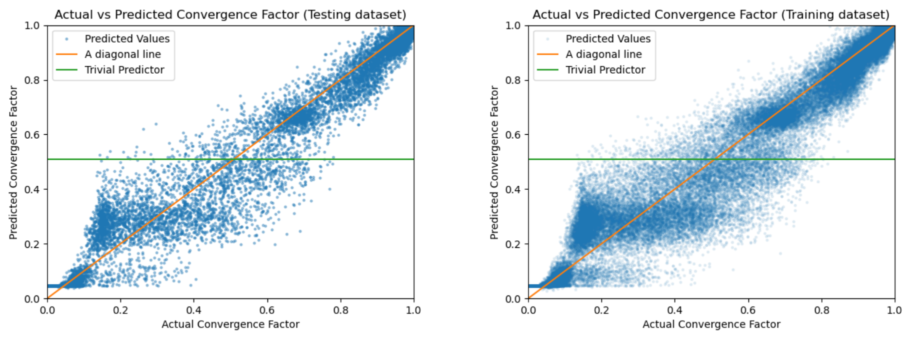
Again, results show that networks are able to learn convergence values for the particular multigrid solver and recirculating-flow problem. Graph nets and convolutional networks seem to have more-or-less the same predictive power, with the graph nets having the obvious upside of being cable of handling non-rectangular grid inputs. Almost expectedly, the message-passing network (ECC) was able to more effectively learn the problem space than the GCN network, likely due to it learning edge weightings as well.



(a) Static dataset (b) Variable dataset

Fig. 3.3: MSE loss history on (3.3a) static and (3.3b) variable datasets. Network types are differentiated between each other and lower values correspond with lower prediction error against the entire dataset.

Predicted convergence values by the CNN over the static dataset are displayed in Figure 3.4. The subplots detail predicted convergence rates vs their true value for each C/F splitting. Almost opposite to the phenomenon observed in the Poisson CNN (Figure 3.1, the network predicts more poorly converging grids accurately. This could perhaps be explained by the data more densely populated with the poor convergence grids, and has a distinct lack of optimal grids.



(a) Testing predictions (b) Training predictions

Fig. 3.4: Predicted convergence rates vs. true convergence rates on (3.4a) testing and (3.4b) training datasets for model convection-diffusion problem, using a convolutional network. Values closer to the diagonal represent more accurate predictions.

Convergence predictions by the ECC network over the variable-size dataset are given in Figure 3.5. Various horizontal trends of predictions are visible in the plots, with distinct subsets of convergence factors all being predicted as the same value.

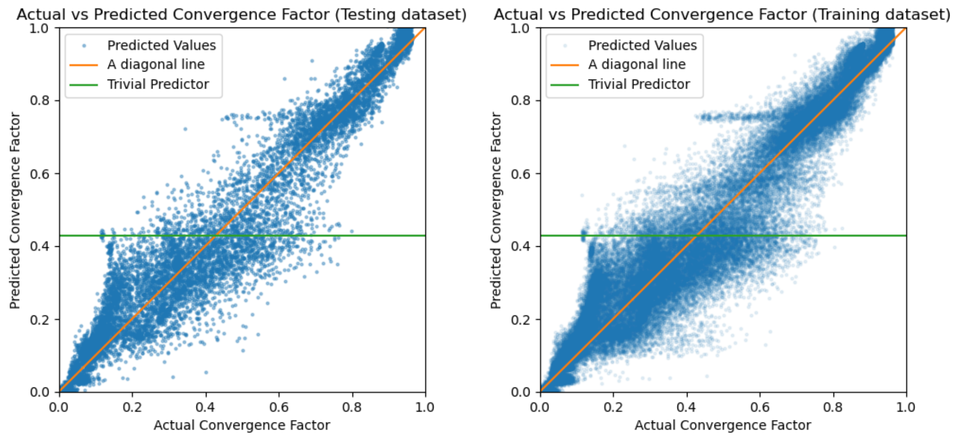
4. Discussion and Conclusions.

In this paper, a method for training convolutional and graph nets was presented to predict convergence of a two-level multigrid solver when given an input C/F mesh splitting. Results indicate that neural networks can indeed predict convergence for a multigrid solver. Convolutional networks are effective at learning attributes on structured grid-like 1D or 2D meshes, and graph nets are able to generalize on variably-sized structured meshes. The graph networks could be further developed and trained on more general classes of problems, instead of the specific convection-diffusion problem or the variable-coefficent Poisson equation.

Future works could build on the networks trained here and explore their applications, using them for example to pick between different AMG aggregation methods to find the most optimal for a specific use case. Another more ambitious future work could be to use the neural networks in an optimization method to obtain the most convergent or most work efficient C/F splitting for a problem.

REFERENCES

[1] J. Adler, H. D. Sterck, S. MacLachlan, and L. Olson, *Numerical partial differential equations*. Draft, 2020.
[2] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM, 2000.
[3] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, *Parallel distributed computing using Python*, Advances in Water Resources, 34 (2011), pp. 1124–1139, https://doi.org/http:



(a) Testing predictions

(b) Training predictions

Fig. 3.5: Predicted convergence rates vs. true convergence rates on (3.5a) testing and (3.5b) training datasets for model convection-diffusion problem, using an Edge-Conditioned Convolution network. Values closer to the diagonal represent more accurate predictions.

//dx.doi.org/10.1016/j.advwatres.2011.04.013. New Computational Methods and Software Tools.

[4] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers With Applications in Incompressible Fluid Dynamics*, Oxford University Press, 2014.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural message passing for quantum chemistry*, 2017, https://arxiv.org/abs/1704.01212.

[6] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015, https://arxiv.org/abs/1512.03385.

[7] M. Homolya and D. A. Ham, *A parallel edge orientation algorithm for quadrilateral meshes*, SIAM Journal on Scientific Computing, 38 (2016), pp. S48–S61, https://doi.org/10.1137/15M1021325, http://arxiv.org/abs/1505.03357, https://arxiv.org/abs/1505.03357.

[8] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2018, https://arxiv.org/abs/1608.06993.

[9] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017, https://arxiv.org/abs/1609.02907.

[10] A. T. T. McRae, G.-T. Bercea, L. Mitchell, D. A. Ham, and C. J. Cotter, *Automated generation and symbolic manipulation of tensor product finite elements*, SIAM Journal on Scientific Computing, 38 (2016), pp. S25–S47, https://doi.org/10.1137/15M1021167, http://arxiv.org/abs/1411.2940, https://arxiv.org/abs/1411.2940.

[11] L. N. Olson and J. B. Schroder, *Pyamg: Algebraic multigrid solvers in python v4.0*, 2018, https://github.com/pyamg/pyamg. Release 4.0.

[12] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: automating the finite element method by composing abstractions*, ACM Trans. Math. Softw., 43 (2016), pp. 24:1–24:27, https://doi.org/10.1145/2998441, http://arxiv.org/abs/1501.01809, https://arxiv.org/abs/1501.01809.

[13] J. Ruge and K. Steuben, *Multigrid Methods*, SIAM, 1987, ch. 4. Algebraic Multigrid.

[14] M. Simonovsky and N. Komodakis, *Dynamic edge-conditioned filters in convolutional neural networks on graphs*, 2017, https://arxiv.org/abs/1704.02901.