A Supervised Learning Approach to Predicting Multigrid Convergence

Nicolas Nytko

Matthew West, Luke Olson, Scott MacLachlan

March 18, 2021

Introduction

todo: insert introduction

Poisson Problem

- Look at the 1D variable coefficients case w/ homogeneous Dirichlet conditions

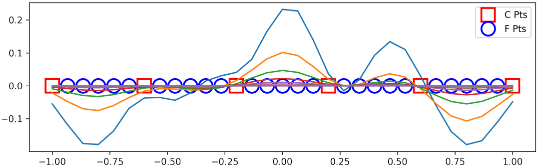
$$-\nabla \cdot \left(k\left(\mathbf{x}\right)\nabla \mathbf{y}\right) = f$$

$$\Omega = [-1, 1] \quad \partial\Omega = 0$$

- Discretized on $N = 31$ internal points using finite differences, $k\left(\mathbf{x}\right)$ is discretized on midpoints to preserve symmetry.
- For arbitrary C/F splitting, can we predict convergence rate and optimal relaxation weight?

Training Dataset

- For “traditional” machine learning we need a dataset.
- Idea: Run a *whole lot* of multigrid iterations.
- Run multigrid iterations and record convergence rate and relaxation weight for randomly generated C/F splittings and problem setups.



Dataset Generation

- Start from "reference" splittings, which are evenly spaced coarse points on a grid.
- Randomly perturb each reference in several trials, according to

$$p = \{0.01 \quad 0.05 \quad 0.1 \quad 0.25 \quad 0.5 \quad 0.75\}.$$

- Generate variable coefficients such that

$$k(\mathbf{x}) = \begin{cases} \alpha \\ \text{rand()}\,(\alpha + 1) \\ \alpha \cos(\pi x \beta) + \gamma \\ \left|\sum_{i=1}^{5} \alpha_i x^i\right| + 0.01 \end{cases}.$$

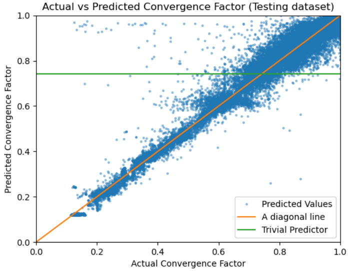
Convolutional Network

- Take the C/F splittings, run in multigrid solver to find convergence rate and relaxation weight that maximizes the former.
- Use the data to train a *1D convolutional network* that predicts convergence, Jacobi relaxation.

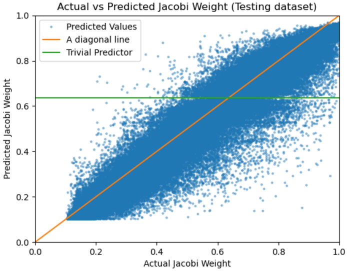
Model	Dataset	Value
Jacobi Weight	Training	1.8331×10^{-3}
Jacobi Weight	Testing	1.8396×10^{-3}
Convergence Factor	Training	1.4839×10^{-3}
Convergence Factor	Testing	1.5171×10^{-3}

Table: Mean squared error (MSE) between predicted and true Jacobi weight, convergence factor.

CNN Performance



(a) Testing predictions



(b) Training predictions

Figure: (a) Predicted convergence values vs. true convergence values, (b) Predicted relaxation weights vs true relaxation weights. Values closer to the diagonal represent more accurate predictions.

What we learned: Poisson is too easy!

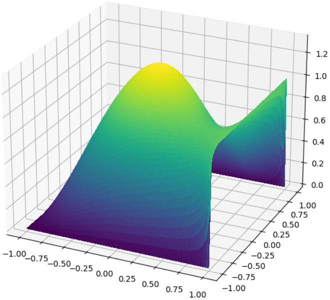
Let's try learning a more difficult problem.

Convection-Diffusion Problem

Try out a specific convection-diffusion problem,

$$\mathbf{w} \cdot \nabla u - k\nabla^2 u = f,$$

on 2D square, $\Omega = [-1, 1]^2$, discretized as quadrilateral finite elements. Use $k = 0.1$, $\mathbf{w} = \begin{bmatrix} 2y(1-x^2) & 2x(1-y^2) \end{bmatrix}$. Apply Dirichlet conditions as one "hot" wall and three "cold" walls.



Dataset Generation, Convection-Diffusion

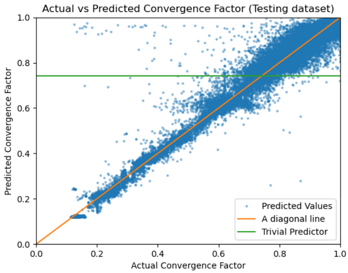
- Discretize on a 25×25 structured grid.
- Start from "reference" splittings, all fine, all coarse, AMG output, etc.
- Randomly perturb each reference in several trials, according to

$$p = \{0.01 \quad 0.05 \quad 0.1 \quad 0.25 \quad 0.5 \quad 0.75\}.$$

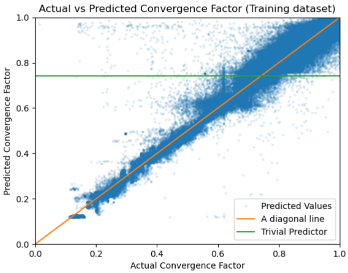
- Don't generate coefficient values for now.
- Take output and run through 50 iteration multigrid solver to find convergence rate.

Convection-Diffusion Convolution

- 2D structured grid ⇒ train 2D convolutional network to predict convergence.



(a) Testing predictions



(b) Training predictions

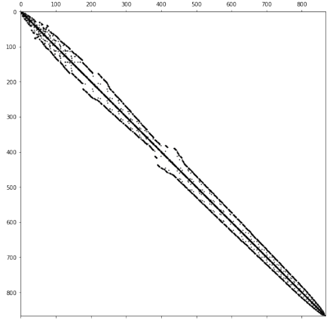
Figure: Predicted convergence values vs. true convergence values on (2a) testing and (2b) training datasets for Poisson equation. Values closer to the diagonal represent more accurate predictions.

CNN to GNN

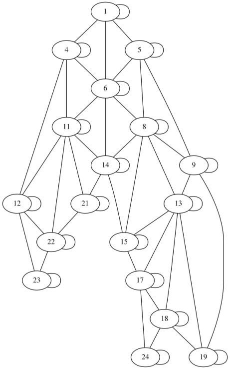
- Classical convolution techniques work okay on structured, grid-like inputs.
- Very restrictive in terms of mesh data we can use for FEM solvers.
- Take a look at some network architectures that allow for unstructured data: introduce *graph-nets*.

Graphnets Motivation

Take FEM matrix, convert to graph and try to learn properties about the system.



$\Longrightarrow$



Message-Passing Graph Convolutions

- Many graph convolution implementations, one such is the *Message-Passing Graph* layer.
- Nodes learn optimal "messages" to pass via edges. Each node passes this message to other nodes in its neighborhood.
- Update given by:

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

- Stacking multiple of these layers approximates traditional grid-based convolution.

Message-Passing Architecture

- Pass node C/F values, $\mathbf{X}$, through 4 MPN layers to get $\mathbf{H}^{(i)},\ i \in \{1,2,3,4\}$.
- Stack historical values to both simulate residual-style networks and give more information to aggregator:

$$\mathbf{R} = \begin{bmatrix} \dots & \mathbf{X}^T & \dots \\ \dots & \left(\mathbf{H}^{(1)}\right)^T & \dots \\ \dots & \left(\mathbf{H}^{(2)}\right)^T & \dots \\ \dots & \left(\mathbf{H}^{(3)}\right)^T & \dots \\ \dots & \left(\mathbf{H}^{(4)}\right)^T & \dots \end{bmatrix}$$

- Run each set of nodal values through linear NN, take average for final convergence rate:

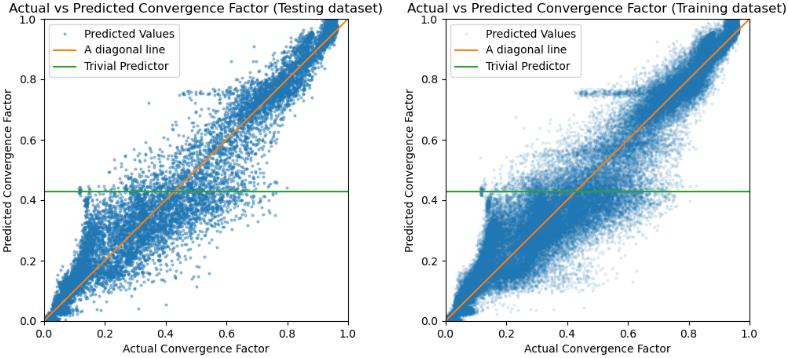
$$y = \frac{1}{N} \sum_j \sigma\left(\left(\mathbf{r}_j \mathbf{W}^{(5)} + \mathbf{b}^{(5)}\right) \mathbf{W}^{(6)} + \mathbf{b}^{(6)}\right)$$

Message-Passing Dataset

- Decoupled the neural network from a fixed input size due to final aggregation step.
- Can have variable-sized input. Now generate and train on a variably-sized dataset of four mesh sizes:

$$\{15 \times 15 \quad 25 \times 25 \quad 35 \times 35 \quad 50 \times 50\}$$

Message-Passing Performance

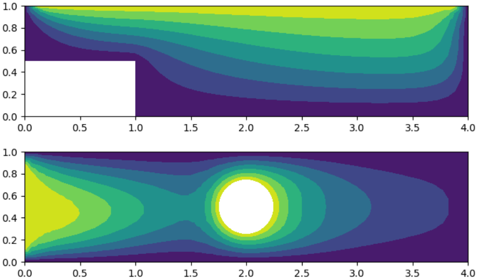


(a) Testing predictions (b) Training predictions

Figure: Predicted convergence rates vs. true convergence rates on (4a) testing and (4b) training datasets for model convection-diffusion problem, using an Edge-Conditioned Convolution network. Values closer to the diagonal represent more accurate predictions.

Future Directions

- Try out some more interesting problems:



- Pick between different AMG methods with predictions.
- Use predictions in an optimization routine to find most convergent C/F splitting.