Approximating the Inverse of a Sparse Linear Operator With Graph Neural Networks

Jie Chen

MIT-IBM Watson AI Lab, IBM Research

Presented at Preconditioning Conference, June 12, 2024
(Updated October 2, 2024)

Introduction

Preconditioning is at the heart of iterative solutions of large, sparse linear systems of equations.

Is the field dead? – No, ill-conditioned problems are still very challenging.

We aim at developing a new kind of algebraic preconditioner by using neural networks.

Why neural networks? – Time to exploit these universal function approximators.

We develop a neural network $\mathbf{M}$ that directly approximates $\mathbf{A}^{-1}$. Distinctions from existing methods:

- No PDE. Purely algebraic.
- Some neural network approaches exist to learn the nonzeros in ILU or approximate inverse.
- We exploit nothing that neural operators heavily depend on (e.g., spatial coordinates, smoothness, decay of Green's function, etc).

We evaluate on over 800 matrices and 50 application areas – widest coverage seen in the literature.

Stay tuned and we'll see the advantages of the proposed preconditioner!

Flexible Prconditioning

Algorithm 1 FGMRES(m) with $\mathbf{M} \approx \mathbf{A}^{-1}$ being a nonlinear operator

Let $\mathbf{x}_0$ be given. Define $\overline{\mathbf{H}}_m \in \mathbb{R}^{(m+1)\times m}$ and initialize all its entries h_{ij} to zero
loop until `maxiters` is reached
    Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$
    for $j = 1, \ldots, m$ **do**
        Compute $\mathbf{z}_j = \mathbf{M}(\mathbf{v}_j)$ and $\mathbf{w} = \mathbf{A}\mathbf{z}_j$
        for $i = 1, \ldots, j$ **do**
            Compute $h_{ij} = \mathbf{w}^\top \mathbf{v}_i$ and $\mathbf{w} \leftarrow \mathbf{w} - h_{ij}\mathbf{v}_i$
        end for
        Compute $h_{j+1,j} = \|\mathbf{w}\|_2$ and $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
    end for
    Define $\mathbf{Z}_m = [\mathbf{z}_1, \ldots, \mathbf{z}_m]$ and compute $\mathbf{x}_m = \mathbf{x}_0 + \mathbf{Z}_m \mathbf{y}_m$ where $\mathbf{y}_m = \operatorname{argmin}_{\mathbf{y}} \|\beta \mathbf{e}_1 - \overline{\mathbf{H}}_m \mathbf{y}\|_2$
    If $\|\mathbf{b} - \mathbf{A}\mathbf{x}_m\|_2 < \texttt{tol}$, exit the loop; otherwise, set $\mathbf{x}_0 \leftarrow \mathbf{x}_m$
end loop

$$\mathbf{A}\mathbf{Z}_m = \mathbf{V}_{m+1}\overline{\mathbf{H}}_m$$

Some Theory

Theorem

Assume that FGMRES is run without restart and without breakdown. On completion, let $\mathbf{H}_n$ *be diagonalizable, as in* $\mathbf{Y}^{-1}\mathbf{H}_n\mathbf{Y} = \mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$. *Then, the residual norm satisfies*

$$\|\mathbf{r}_m\|_2 \leq \kappa_2(\mathbf{Y})\ \epsilon^{(m)}(\mathbf{\Sigma})\ \|\mathbf{r}_0\|_2,$$

where κ_2 *denotes the 2-norm condition number,* $\mathbb{P}_m$ *denotes the space of degree-*m *polynomials, and*

$$\epsilon^{(m)}(\mathbf{\Sigma}) = \min_{p \in \mathbb{P}_m,\, p(0)=1} \max_{i=1,\ldots,n} |p(\sigma_i)|.$$

Graph Neural Network / Graph Convolutional Network

A sparse matrix $\mathbf{A}$ admits a graph interpretation, similar to how AMG interprets the coefficient matrix.

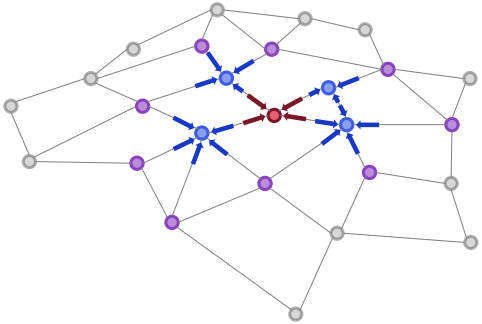
Treating $\mathbf{A}$ as the graph adjacency matrix allows us to use GNNs to parameterize the preconditioner.

Graph convolutional networks

$$\mathbf{Y} = \mathrm{softmax}(\widehat{\mathbf{A}} \cdot \mathrm{ReLU}(\underbrace{\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}}) \cdot \mathbf{W}^{(1)})$$

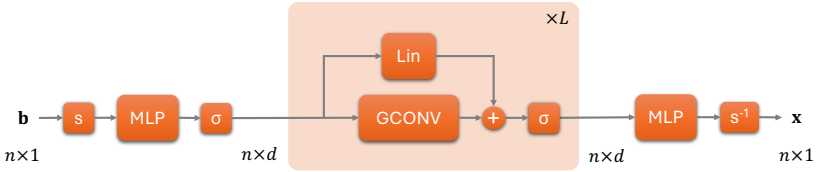
aggregate purple to blue

aggregate blue to red



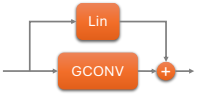
A GCN layer is defined as $\mathrm{GCONV}(\mathbf{X}) = \mathrm{ReLU}(\widehat{\mathbf{A}}\mathbf{X}\mathbf{W})$, where $\widehat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ is some normalization of $\mathbf{A}$, $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{in}}}$ is input data, and $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is a learnable parameter.

Graph Neural Preconditioner (GNP)



GCONV Normalize $\mathbf{A}$ by an upper bound of the spectral radius:

$$\widehat{\mathbf{A}} = \mathbf{A}/\gamma \quad \text{where} \quad \gamma = \min\left\{ \max_i \left\{ \sum_j |a|_{ij} \right\}, \max_j \left\{ \sum_i |a|_{ij} \right\} \right\}$$



Use residual connections to stack a deeper network:

$$\text{Res-GCONV}(\mathbf{X}) = \text{ReLU}(\mathbf{X}\mathbf{U} + \widehat{\mathbf{A}}\mathbf{X}\mathbf{W})$$

s s^{-1} Ensure scale-equivariance $\mathbf{M}(\alpha\mathbf{b}) = \alpha\mathbf{M}(\mathbf{b})$:

$$s(\cdot) = \frac{\sqrt{n}}{\tau} \cdot \quad \text{and} \quad s^{-1}(\cdot) = \frac{\tau}{\sqrt{n}} \cdot, \quad \text{where } \tau = \|\mathbf{b}\|_2$$

Training Data Generation

Want to sample $(\mathbf{b}, \mathbf{x})$ pairs to train $\mathbf{M}$.

Option 1: $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. Drawback: Input space $\mathbf{b} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}\mathbf{A}^\top)$ misses data along the bottom eigen-subspace of $\mathbf{A}\mathbf{A}^\top$. Easy to train but hard to generalize.

Option 2: $\mathbf{b} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. Drawback: Output space $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}^{-1}\mathbf{A}^{-\top})$ is skewed; hard to train.

Training Data Generation

Want to sample $(\mathbf{b}, \mathbf{x})$ pairs to train $\mathbf{M}$.

Option 1: $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. Drawback: Input space $\mathbf{b} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}\mathbf{A}^\top)$ misses data along the bottom eigen-subspace of $\mathbf{A}\mathbf{A}^\top$. Easy to train but hard to generalize.

Option 2: $\mathbf{b} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. Drawback: Output space $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}^{-1}\mathbf{A}^{-\top})$ is skewed; hard to train.

Our proposal:

1. Run m-step Arnoldi without preconditioner $\mathbf{A}\mathbf{V}_m = \mathbf{V}_{m+1}\overline{\mathbf{H}}_m$
2. Perform SVD $\overline{\mathbf{H}}_m = \mathbf{W}_m\mathbf{S}_m\mathbf{Z}_m^\top$
3. Define $\mathbf{x} = \mathbf{V}_m\mathbf{Z}_m\mathbf{S}_m^{-1}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_m)$

One sees that

$$\begin{aligned} &\mathbf{x} \in \mathrm{range}(\mathbf{V}_m), \qquad &&\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_m^{\mathbf{x}}), \qquad &&\boldsymbol{\Sigma}_m^{\mathbf{x}} = (\mathbf{V}_m\overline{\mathbf{H}}_m^{+})(\mathbf{V}_m\overline{\mathbf{H}}_m^{+})^\top, \\ &\mathbf{b} \in \mathrm{range}(\mathbf{V}_{m+1}), \qquad &&\mathbf{b} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_m^{\mathbf{b}}), \qquad &&\boldsymbol{\Sigma}_m^{\mathbf{b}} = (\mathbf{V}_{m+1}\mathbf{W}_m)(\mathbf{V}_{m+1}\mathbf{W}_m)^\top. \end{aligned}$$

In practice, we sample half the batch $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_m^{\mathbf{x}})$ and half the batch $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$.

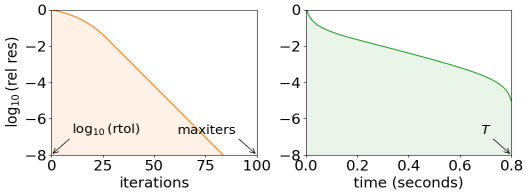
Experiment Setting: Problems

Widest coverage in the literature: 867 matrices from SuiteSparse

- All square, real-valued, and non-SPD matrices ...
- ... whose number of rows falls between 1K and 100K and
- ... whose number of nonzeros is fewer than 2M

1. 2D/3D problem
2. acoustics problem
3. chemical process simulation problem (sequence)
4. combinatorial problem
5. (duplicate) model reduction problem
6. (duplicate) structural problem (sequence)
7. electromagnetics problem
8. frequency-domain circuit simulation problem
9. materials problem
10. (subsequent) circuit simulation problem (sequence)
11. (subsequent) computational fluid dynamics problem (sequence)
12. (subsequent) power network problem (sequence)
13. (subsequent) semiconductor device problem (sequence)
14. (subsequent) theoretical/quantum chemistry problem (sequence)
15. thermal problem
16. directed (weighted) temporal (multi)graph
17. (un)directed multigraph
18. (un)directed weighted graph (sequence)
19. (un)directed weighted random graph
20. linear programming problem
21. optimal control problem
22. (subsequent) optimization problem (sequence)
23. counter-example problem
24. economic problem
25. statistical/mathematical problem

Experiment Setting: Metrics



Area under the relative residual norm curve with respect to iterations

$$\text{Iter-AUC} = \sum_{i=0}^{\text{iters}} \log_{10} r_i - \log_{10} \texttt{rtol}, \quad r_i = \|\mathbf{b} - \mathbf{A}\mathbf{x}_i\|_2 / \|\mathbf{b}\|_2$$

Area under the relative residual norm curve with respect to time (using `timeout` to stop)

$$\text{Time-AUC} = \int_0^T [\log_{10} r(t) - \log_{10} \texttt{rtol}]\, dt \approx \sum_{i=1}^{\text{iters}} [\log_{10} r_i - \log_{10} \texttt{rtol}](t_i - t_{i-1})$$

Experiment Setting: Additional Details

Solver: FGMRES(10), $\mathbf{x}_0 = \mathbf{0}$

Stopping criteria: `rtol = 1e-8`, `maxiters = 100`, no timeout

Compared preconditioners (Python): (1) GMRES(10); (2) ILU `scipy.sparse.linalg.spilu`; (3) AMG `pyamg.blackbox.solver().aspreconditioner` (all parameters using default choices)

Neural network (no hyperparameter tuning):

- Res-GCONV layers: 8
- layer in/out dimension: 16
- MLP layer: 2
- MLP hidden dimension: 32
- optimizer: Adam
- learning rate: 1e-3
- steps: 2000
- batch size: 16
- dropout: 0
- weight decay: 0
- model: best training
- Arnoldi steps: 40

Training loss: ℓ_1 residual norm $\|\mathbf{AM}(\mathbf{b}) - \mathbf{Ax}\|_1$

Compute: one Tesla V100(16GB) GPU, 96 Intel Xeon 2.40GHz cores, 386GB main memory

Results 1/5

Question 1: How does GNP perform compared with traditional preconditioners?

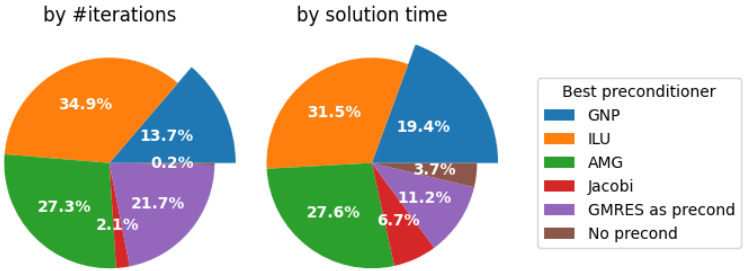


Figure: Percentage of problems on which each preconditioner is the best.

	by #iter	by time
25%	1.91e+00	1.18e+00
50%	6.74e+00	2.33e+00
75%	6.78e+03	8.16e+01
100%	1.89e+10	1.91e+10

Table: Distribution of the residual-norm ratio between the second best preconditioner and GNP, when GNP performs the best. Distribution is described by using percentiles.

Results 1/5

Question 1: How does GNP perform compared with traditional preconditioners?

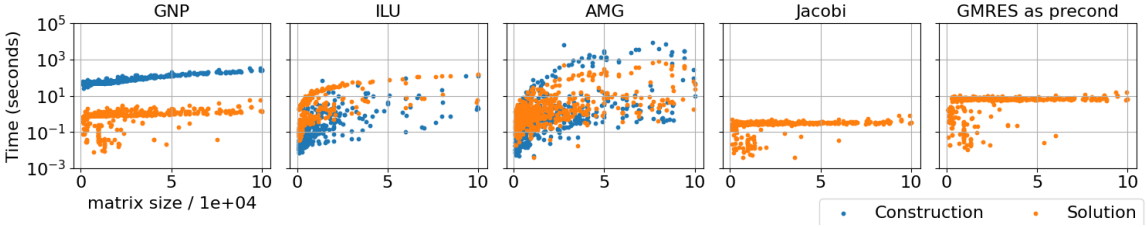


Figure: Preconditioner construction time and solution time (using `maxiters` to stop). The construction time of Jacobi is negligible and not shown. GMRES does not require construction.

Results 2/5

Question 2: On what problems does GNP perform the best?

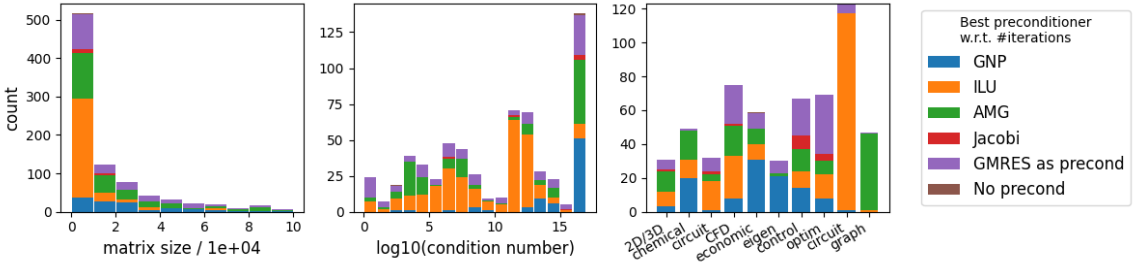


Figure: Breakdown of best preconditioners with respect to matrix sizes, condition numbers, and application areas. Only the application areas with the top number of problems are shown. The last bar in the middle plot is for condition number $\geq 10^{16}$.

Results 3/5

Question 3: How robust is GNP?

Table: Failures of preconditioners (count and proportion).

	GNP	ILU	AMG	Jacobi	GMRES as precond
Construction failure	0 (0.00%)	348 (40.14%)	62 (7.15%)	N/A	N/A
Solution failure	1 (0.12%)	61 (7.04%)	5 (0.58%)	53 (6.11%)	2 (0.23%)

- Common failures of ILU construction: "(f)actor is exactly singular" and "matrix is singular ... in file ilu_dpivotL.c"
- Common failures of AMG construction: "array ... contain(s) infs or NaNs".
- Solution failures occur when the residual norm tracked by QR($\overline{\mathbf{H}}_m$) fails to match the actual residual norm.

Results 4/5

Question 4: What does the convergence history look like with GNP?

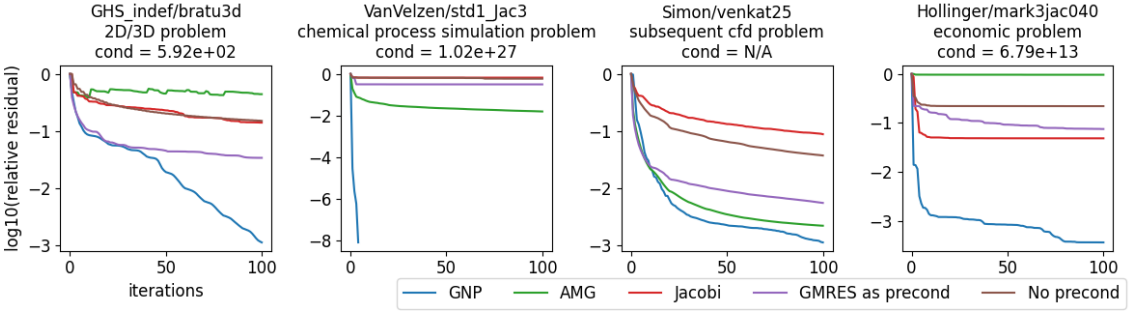


Figure: Example convergence histories.

Results 4/5

Question 4: What does the convergence history look like with GNP?

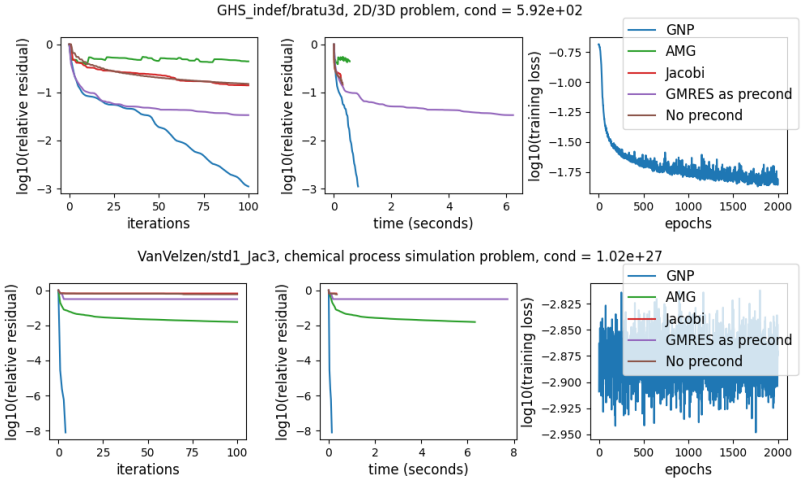


Figure: Convergence of the linear system solutions and training history of the preconditioners.

Results 4/5

Question 4: What does the convergence history look like with GNP?

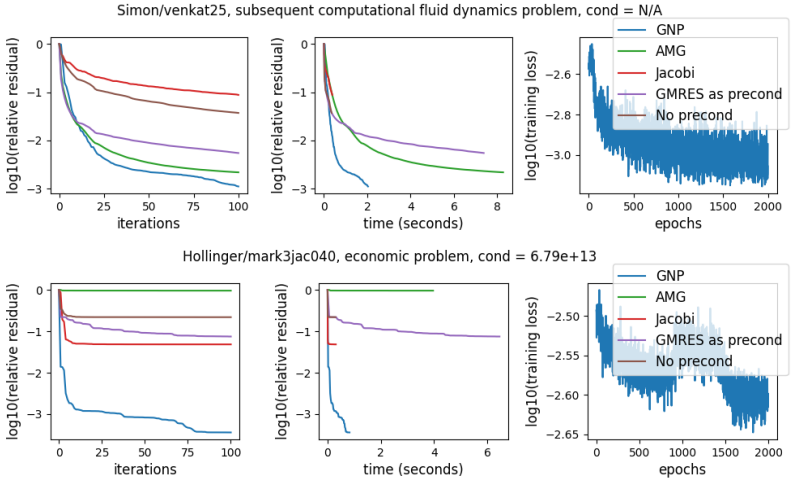


Figure: Convergence of the linear system solutions and training history of the preconditioners.

Results 5/5

Question 5: Are the proposed training-data generation and the scale-equivariance design necessary?

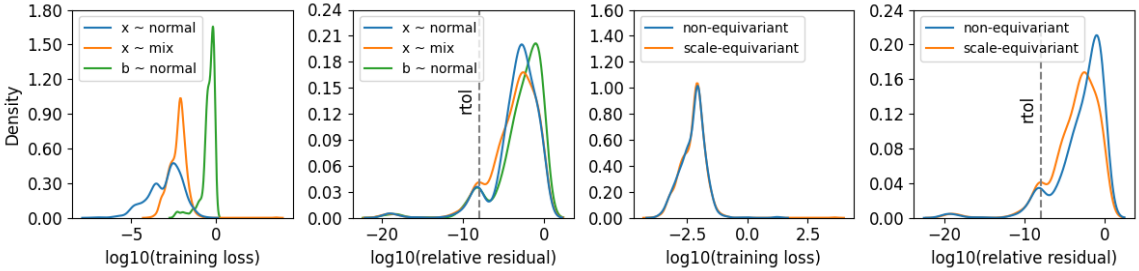


Figure: Left: comparison of training data generation; right: comparison of scale-equivariance.

Conclusions

- Graph neural network has a strong potential to serve as a general-purpose preconditioner
- GNP performs competitively for ill-conditioned problems
- GNP is robust with predictable construction costs (more predictable than ILU and AMG)
- GNP is faster than GMRES (which may be bottlenecked by orthogonalization)

Future Directions

- Preconditioning SPD matrices. (flexible CG + split preconditioner?)
- Preconditioning a sequence of evolving matrices. (continual learning)
- Scalability. (distributed and/or multi-GPU training of GNN)
- Problem-specific hyperparameter tuning and architecture tuning. (applications!)