Graph Neural Preconditioners for Iterative Solutions of Sparse Linear Systems

Jie Chen

MIT-IBM Watson AI Lab, IBM Research

The Thirteenth International Conference on Learning Representations (ICLR 2025)

Introduction

Krylov methods are commonly used to solve large, sparse linear systems

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{A} \in \mathbb{R}^{n \times n}.$$

The effectiveness of Krylov methods heavily depends on the conditioning of the matrix $\mathbf{A}$.

Preconditioning amounts to using a matrix $\mathbf{M} \approx \mathbf{A}^{-1}$ to solve an easier problem $\mathbf{A}\mathbf{M}\mathbf{u} = \mathbf{b}$ with the new unknown $\mathbf{u}$ and recover the solution $\mathbf{x} = \mathbf{M}\mathbf{u}$.

Main question

Can we design a neural network $\mathbb{R}^n \to \mathbb{R}^n$ as a preconditioner $\mathbf{M}$ (i.e., $\mathbf{M} \approx \mathbf{A}^{-1}$)?

- PINN/NO approaches: They need PDEs and exploit additional information (e.g., spatial coordinates, smoothness, decay of Green's function, etc).
- Neural incomplete-factorization approaches: There have the same drawback as incomplete factorization – nonzero factorization error.
- We aim at a general-purpose preconditioner: No PDE; purely algebraic.

Example problems and application areas

867 non-SPD matrices $\mathbf{A}^{-1}$ from SuiteSparse. $1\text{K} \leq n \leq 100\text{K}$, $\text{nnz} \leq 2\text{M}$.
They come from 50 application areas.

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

Graph neural network

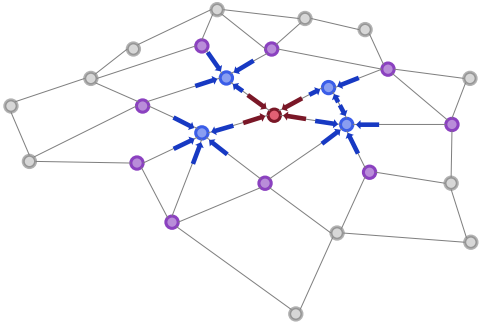
Treating $\mathbf{A}$ as the graph adjacency matrix allows us to use GNNs to parameterize the preconditioner.

Graph convolutional networks

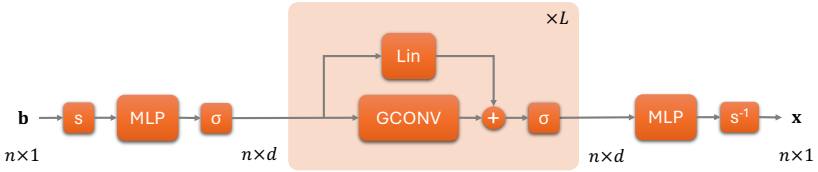
$$\mathbf{Y} = \mathrm{softmax}(\widehat{\mathbf{A}} \cdot \mathrm{ReLU}(\underbrace{\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}}) \cdot \mathbf{W}^{(1)})$$

aggregate purple to blue

aggregate blue to red

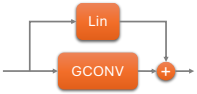


Graph neural preconditioner (GNP)



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 $\mathbf{M}(\alpha\mathbf{b}) = \alpha\mathbf{M}(\mathbf{b})$:

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

Nuts and bolts: Training data generation

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.

Nuts and bolts: Training data generation

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 $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_m)$

It is easy to see that

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

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

Nuts and bolts: evaluation

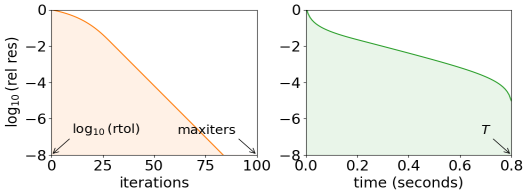
How to evaluate a general-purpose preconditioner?

- Need a broad coverage (as opposed to one or two PDEs)
- Cannot just stare at the convergence plot
- Need a metric

Good evaluation metrics are hard to define!

- Comparing the iteration count is insufficient (per-iteration time of each method is different)
- Comparing time is tricky. If using `rtol` as a stopping criterion, hard to set the same `rtol` for diverse problems
- If using `timeout` as a stopping criterion, hard to compare reaching `maxiters` with poor residual versus reaching `timeout` with good residual

Nuts and bolts: evaluation



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 findings 1/4

GNP performs the best for a substantial portion of the problems.

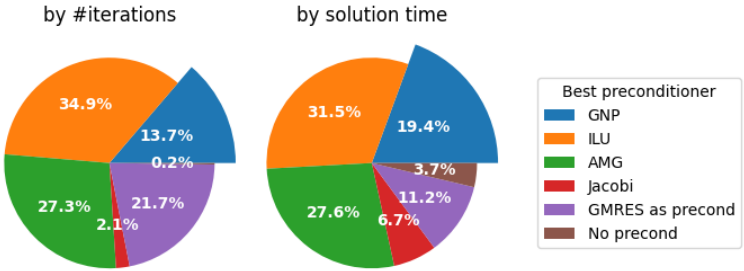


Figure: Percentage of problems on which each preconditioner is the best.

	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.

Experiment findings 2/4

The construction time of GNP is predictable, while that of ILU and AMG is not.
GNP sometimes is much faster to construct than ILU and AMG.
GNP is faster in execution than GMRES as a preconditioner.

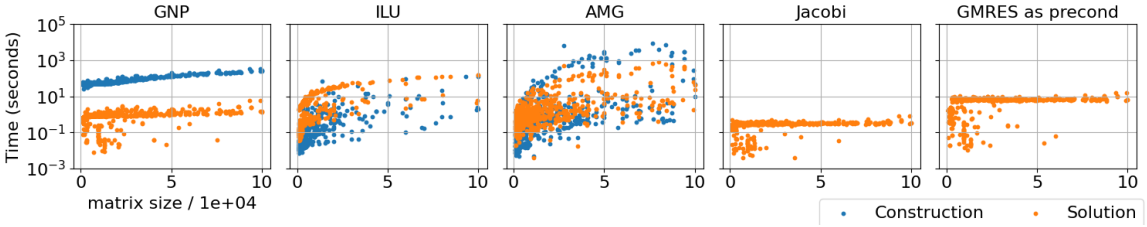


Figure: Preconditioner construction time and solution time (using `maxiters` to stop). The construction time of Jacobi is negligible and not shown. GMRES does not require construction.

Experiment findings 3/4

GNP performs competitively for ill-conditioned problems.
GNP is useful in some areas: chemical process simulation, economics, and eigenvalue/model reduction.

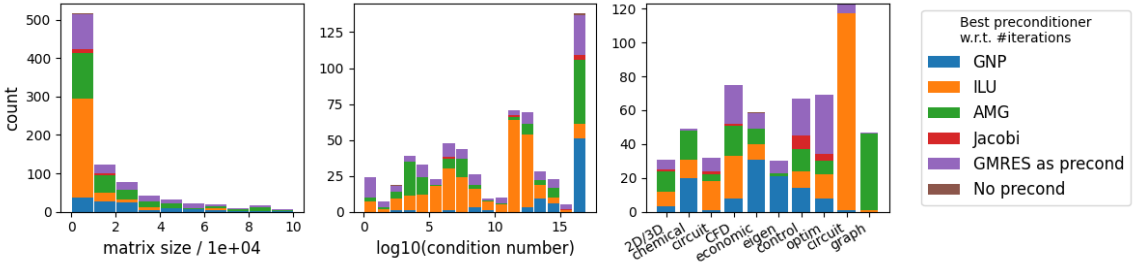


Figure: Breakdown of best preconditioners with respect to matrix sizes, condition numbers, and application areas. Only the application areas with the top number of problems are shown. The last bar in the middle plot is for condition number $\geq 10^{16}$.

Experiment findings 4/4

GNP is very robust, while ILU and AMG often fail.

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.

Conclusions, paper and code

Graph neural network has a strong potential to serve as a general-purpose preconditioner

- mitigating the restrictive modeling of the sparsity pattern (as in ILU and approximate inverse)
- mitigating the insufficient quality of polynomial approximation (as in inner-outer GMRES)
- mitigating the challenge of smoothing over a non-geometric mesh (as in AMG)

Paper: https://jiechenjiechen.github.io/pub/GNP.pdf

Code and demo: https://github.com/jiechenjiechen/GNP