Efficient preconditioning for iterative methods with graph neural networks

Vladislav Trifonov[a], **Alexander Rudikov**[b], **Oleg Iliev**[c], **Yuri M. Laevsky**[d], **Ivan Oseledets**[b], **Ekaterina Muravleva**[a]

[a] *Sberbank of Russia, AI4S Center, Moscow, Russian Federation*
Skolkovo Institute of Science and Technology, Moscow, Russia
vladislav.trifonov@skoltech.ru, e.muravleva@skoltech.ru

[b] *Artificial Intelligence Research Institute (AIRI), Moscow, Russia*
Skolkovo Institute of Science and Technology, Moscow, Russia
rudikov@airi.net, oseledets@airi.net

[c] *Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany*
oleg.iliev@itwm.fraunhofer.de

[d] *Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia*
laev@labchem.sscc.ru

1. Abstract

In this paper we present a novel GNN-based approach to construct preconditioners for the iterative methods of solving large sparse linear systems of algebraic equations - the PreCorrector. The PreCorrector is based on the idea that classical preconditioners of the ILU family provide reliable preconditioners for the CG method. Although the algorithm for their construction is known, they are not unique and one can try to train a neural network to construct preconditioners of the same sparsity pattern with a better effect on the spectrum. This can be shown by inplace updating of IC(0) factors with gradient descent for a single matrix (Figure 1).

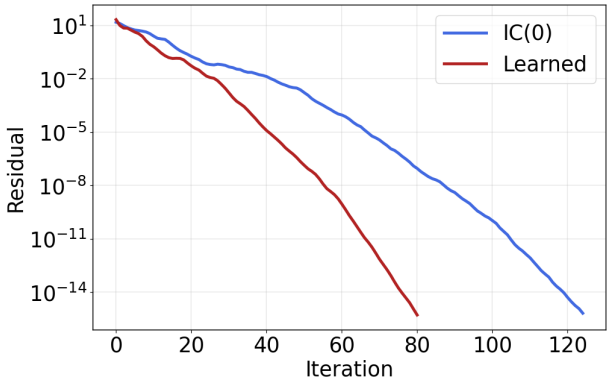

Fig. 1: Inplace updating of IC(0) factor allows to obtain better preconditioner.

2. Introduction

Modern problems in computational science and engineering are often based on parametric partial differential equations (PDEs). Researchers are required to use numerical methods to solve PDEs that result in a system of linear algebraic equations $Ax = b,\ A \in \mathbb{R}^{n\times n},\ x \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$. These systems are usually sparse, i.e. the number of non-zero elements is $\ll n^2$.

To solve such systems, the Krylov subspace iterative methods are used, which search for the solution in $\mathcal{K}_r(A, b) = \text{span}\{b,\ Ab,\ A^2b, \ldots,\ A^{r-1}\}$. Typically, the application of parametric PDEs produces large linear systems, often with entries of varying scale, resulting in ill-conditioned matrices with large condition numbers $\kappa(A)$, which complicates the solution process. A well-designed preconditioner P that $\kappa(P^{-1}A) \ll \kappa(A)$ is a crucial element in solving large linear systems.

Recently, GNNs have been shown to be capable of designing preconditioners for iterative methods [1, 2]. However, preconditioners designed with these approaches cannot outperform ones designed with classical methods in terms of the effect on the spectrum and hence the number of iterations in CG.

We present PreCorrector (short for Preconditioner Corrector), an approach that combines graph neural network and classical linear algebra to construct efficient preconditioners. The preconditioners constructed with PreCorrector reduce $\kappa(A)$ more significantly than classical preconditioners, leading to faster convergence of iterative methods.

3. Neural design of preconditioner

3.1 Problem statement

We consider systems of linear algebraic equations from the discretization of differential operators formed with a symmetric positive definite (SPD) matrix $A \succ 0$. The conjugate gradient (CG) method is used to solve large sparse systems with SPD matrices [3, 4]. The convergence rate of CG is determined by $\sqrt{\kappa(A)}$, which makes it crucial to obtain a preconditioner P such that the preconditioned linear system $P^{-1}Ax = P^{-1}b$ has a lower condition number than the initial system. If one knows the sparsity pattern of A, then possible options are incomplete Cholesky decomposition (IC) [3]. Additional information about these preconditioners can be found in the Appendix A.

Table 1: Comparison of the classical algorithms, IC(0) and ICt(1), and PreCorrector with a corresponding initial factorization, PreCor$\big[$IC(0)$\big]$ and PreCor$\big[$ICt(1)$\big]$. The size of a linear system is $6.6 \cdot 10^4$. Lower is better, the best results are in bold. Pre-time stands for precomputation time.

Method	Pre-time	Time (iters) to 10^{-3}	Time (iters) to 10^{-6}	Time (iters) to 10^{-9}	Time (iters) to 10^{-12}
IC(0)	$1.8 \cdot 10^{-3} \pm 6.0 \cdot 10^{-5}$	8.638 ± 2.000 (345 ± 13.5)	10.635 ± 2.355 (425 ± 14.9)	12.249 ± 2.654 (490 ± 14.9)	13.608 ± 2.905 (544 ± 14.4)
PreCor$\big[$IC(0)$\big]$	$3.3 \cdot 10^{-3} \pm 6.1 \cdot 10^{-4}$	**3.388** ± 0.544 (144 ± 12.6)	**4.199** ± 0.663 (179 ± 15.5)	**4.943** ± 0.775 (211 ± 17.7)	**5.637** ± 0.875 (241 ± 19.6)
ICt(1)	$1.2 \cdot 10^{-2} \pm 1.8 \cdot 10^{-4}$	5.278 ± 0.593 (209 ± 8.4)	6.515 ± 0.709 (259 ± 9.0)	7.511 ± 0.809 (298 ± 9.1)	8.340 ± 0.890 (332 ± 8.8)
PreCor$\big[$ICt(1)$\big]$	$1.4 \cdot 10^{-2} \pm 6.8 \cdot 10^{-4}$	**2.719** ± 0.381 (105 ± 8.3)	**3.378** ± 0.462 (131 ± 9.8)	**3.976** ± 0.539 (154 ± 11.2)	**4.534** ± 0.608 (176 ± 12.4)

The duality between sparse matrices and graphs is used to obtain vertices and edges, such as $Ax = b \rightarrow \mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $a_{i,j} = e_{i,j} \in \mathcal{E}, b_i = v_i \in \mathcal{V}$. Then the message-passing architecture [5] can be used to update the edges of the matrix A preserving the sparsity pattern and output the updated lower triangular part $L(\theta)$. Finally, one can form preconditioner in the form of a Cholesky decomposition [6] $P(\theta) = L(\theta)L(\theta)^\top$.

3.2 Related work

Several previous papers proposed to use convolutional neural networks with sparse convolutions [7, 8] and graph neural networks [1, 2]. The latter approaches use shallow GNNs and typically require a single inference before the iteration process. GNNs take the initial left hand side matrix A and right hand side vector b as input and construct preconditioners in the form of a Cholesky decomposition $L(\theta) = \text{GNN}(\theta, A, b)$. We propose to learn corrections to classical preconditioner to obtain a better effect on the spectrum than classical preconditioners while maintaining the same sparsity.

3.3 PreCorrector

Instead of passing the left hand side matrix A as input to GNN, we propose: (i) to pass L from the IC decomposition to the GNN and (ii) to train GNN to predict a correction for this decomposition:

$$L(\theta) = L + \alpha \cdot \text{GNN}(\theta, L)\,. \tag{1}$$

The correction coefficient α is also a learning parameter that is updated during training. At the beginning of training, we set $\alpha = 0$ to ensure that the first gradient updates come from pure IC factorization. Moreover, GNN in (1) takes as input the lower-triangular matrix L from IC instead of A, so we are not anchored to a single specific sparse pattern of A and we can: (i) omit half of the graph and speed up the training process and (ii) use different sparsity patterns to obtain even better preconditioners. One can find details on PreCorrector architecture in Appendix B.

4. Results and Discussion

The main comparison of preconditioners designed with different algorithms is made by comparing total time, including the preconditioner construction time, and the number of CG iterations to achieve a given tolerance. One can find details on how to compare different preconditioner in Appendix C.

We test PreCorrector on SPD matrices obtained by discretization of elliptic equations. We consider a 2D diffusion equation:

$$\begin{aligned} -\nabla \cdot \big(k(x)\nabla u(x)\big) &= f(x), \text{ in } \Omega \\ u(x)\Big|_{x\in\partial\Omega} &= 0 \end{aligned}\,, \tag{2}$$

Preconditioners constructed with PreCorrector outperform classical algorithms with the same sparsity pattern in both total time and effect on the spectrum, i.e. CG iterations (Table 1). The difference between the inference of the PreCorrector and the construction of the IC is negligible and, most importantly, contributes little to the final time-to-solution. The CG with PreCor$\big[$IC(0)$\big]$ requires fewer iterations to achieve the required tolerance and thus has a better effect on the spectrum of A than IC(0).

Since PreCorrector takes L from any classical preconditioner, one can obtain even better preconditioners with more advanced starting preconditioners. The proposed approach based on the ICt(1) preconditioner, PreCor$\big[$ICt(1)$\big]$, also has a better effect on the spectrum of the initial A than the classical ICt(1).

Compared to the CG with PreCorrector, the CG with GNN from the previous work [1] cannot converge to the required tolerance. We observe that training a GNN from scratch can be unstable, resulting in preconditioners that have a weaker effect on the spectrum than their classical analogues. Learning corrections to classical methods mitigates this problem.

References

[1] Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. Learning preconditioners for conjugate gradient pde solvers. In *International Conference on Machine Learning*, pages 19425–19439. PMLR, 2023.

[2] Paul Häusner, Ozan Öktem, and Jens Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. *arXiv preprint arXiv:2305.16368*, 2023.

[3] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[4] Owe Axelsson. *Iterative solution methods*. Cambridge university press, 1996.

[5] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[6] Lloyd N Trefethen and David Bau. *Numerical linear algebra*. SIAM, 2022.

[7] Johannes Sappl, Laurent Seiler, Matthias Harders, and Wolfgang Rauch. Deep learning of preconditioners for conjugate gradient solvers in urban water related problems. *arXiv preprint arXiv:1906.06925*, 2019.

[8] Salvatore Calì, Daniel C Hackett, Yin Lin, Phiala E Shanahan, and Brian Xiao. Neural-network preconditioners for solving the dirac equation in lattice gauge theory. *Physical Review D*, 107(3):034508, 2023.

Appendix A. ILU preconditioner

Full LU decomposition [6] for a square non-singular matrix defined as the product of the lower and upper triangular matrices $B = L_B U_B$. In general, these matrices have no restriction on the position and number of elements within their triangular structure and can even be dense for a sparse matrix B. On the other hand, an ILU is an approximate LU factorization:

$$A \approx LU, \tag{A1}$$

where $LU - A$ satisfies certain constraints.

Zero fill-in ILU, denoted ILU(0), is an approximate LU factorization $A \approx L_0 U_0$ in such a way that L_0 has exactly the same sparsity pattern as the lower part of A and U_0 has exactly the same sparsity pattern as the upper part of A. For the ILU(p) decomposition the level of fill-in is defined hierarchically. The product of the factors of the ILU(0) decomposition produces a new matrix B with a larger number of non-zero elements. The factors of the ILU(1) factorization have the same sparsity patterns as lower and upper parts of the sparsity pattern of the matrix B. With this recursion one gets a pattern of the ILU(p) factorization with p-level of fill-in. ILU(0) is a typical choice to precondition iterative solvers and relies only on the levels of fill-in, e.g. sparsity patterns [3]. One can obtain better approximation with ILU by using incomplete factorizations with thresholding.

One such technique is the ILU factorization with thresholding (ILUt(p)). The parameter p defines the number of additional non-zeros allowed per column in the resulting factorization. For the ILUt(p) decomposition, the algorithm is more complex and involves both dropping values by some predefined threshold and controlling the number of possible non-zero values in the factorization. In the case of ILUt(p), the value p represents additional non-zero values allowed in the factorization per row. The thresholding algorithm provides a more flexible and effective way to approximate the inverse of a matrix, especially for realistic problems where the numerical values of the matrix elements are important.

The complexity of solving sparse linear systems with matrices in the form of the Cholesky decomposition defined by the number of non-zero elements $\mathcal{O}(\text{nnz})$. This value also defines the storage complexity and the complexity of preconditioner construction.

In this paper we focus on the SPD matrices so instead of ILU, ILU(p) and ILUt(p) we use the incomplete Cholesky factorization IC, IC(p) and ICt(p).

Appendix B. PreCorrector architecture

We use ubiquitous encode-process-decode [**?**] architecture for the GNN, where multilayer perceptrons are used for encoders and decoders. Process-block consists of message-passing layer that increase receptive field of the GNN with multiple rounds. Before encoder and after decoder we apply normalization and renormalization for the edges respectively.

Note that in (1) we omit the right-hand side vector b and do not use it as input for GNN. In our experiments, we observe that GNN with the right-hand side as node input does not effect resulting preconditioner quality, while it suffers from sensitivity to the right-hand side vector. Although we can either use a different representation of the node features (e.g., as in [2]) or use diagonal elements of the matrix. Instead we set vector $\mathbf{1} = [1, \ldots, 1]^T$ as the node input to the message passing layer, which allowed us to reduce the number of parameters by a factor of 2.3 while maintaining the quality of the resulting preconditioner.

By using $\mathbf{1}$ as an input nodes to the processor block, we discard the node encoder MLP. We also notice that node model in processor does not contribute when there are no explicit nodes. We discard it either to use simple aggregation as the node

update model and to further reduce number of parameters. Moreover, we make our approach consistent with classical methods by passing only the left-side matrix A to the preconditioner construction routine. We use the `max` aggregation function, although our experiments showed that the resulting preconditioner quality does not depend on the type of aggregation function.

We also investigate which protocol to follow to increase the receptive field. It is possible to follow the original message passing paper [5] and perform T rounds of message passing with shared learnable functions during each round. Another option is to follow classic GNN's multi-block protocol and combine multiple message-passing layers to increase the receptive field with independent weights. During the PreCorrecotr ablation study we observed no difference in resulting preconditioner between the protocols, so we use message-passing with shared layer to further simplify the model.

Appendix C. Preconditioner comparison

For construction time, we report averaged values over 200 runs of preconditioner construction and for CG time and iterations we report averaged values over the test set as well as standard deviations for the average values. Construction time for PreCorrector is reported including construction time of classical preconditioners.

The GNNs is chosen to construct preconditioners because it allows preserve the sparsity pattern. Therefore, the algorithmic complexity of using preconditioners (matrix-vector product) is the same when using preconditioners with the same sparsity pattern. This allows a fair evaluation of the quality of neural preconditioners with the same sparsity pattern only in terms of the number of CG iterations. Furthermore, all approaches to IC decomposition with the same sparsity pattern, including the classical ones, compete with each other in terms of construction time, effect on the spectrum (i.e., number of CG iterations) and generalization ability.