Neural incomplete factorization: learning preconditioners for the conjugate gradient method

Paul Häusner[1], Ozan Öktem[2], Jens Sjölund[1]

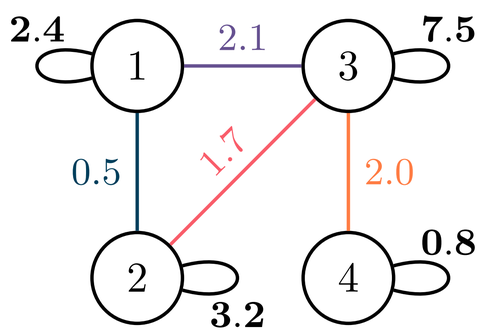
[1]Department of Information Technology, Uppsala University, Sweden
[2]Department of Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden

Overview

We accelerate the conjugate gradient method using graph neural networks for solving large-scale linear equation systems

$$Ax = b$$

$$\begin{bmatrix} \mathbf{2.4} & 0.5 & 2.1 & 0 \\ 0.5 & \mathbf{3.2} & 1.7 & 0 \\ 2.1 & 1.7 & \mathbf{7.5} & 2.0 \\ 0 & 0 & 2.0 & \mathbf{0.8} \end{bmatrix}$$



- Utilize the connection of graphs and sparse matrices to construct a GNN architecture
- Train the neural network to predict a sparse factorization of the matrix A which is used as a preconditioner for the CG method
- **Advantages**: fast to compute and problem specific preconditioner

NeuralIF preconditioner

- **Replace hand-engineered preconditioners for the conjugate gradient method with outputs produced by a neural network**
- Two design requirements for preconditioners:
  - Symmetric positive definitness (spd) to ensure convergence
  - Sparsity of preconditioner to limit resource requirements
- Output a lower triangular matrix $\Lambda_\theta(\cdot)$ with positive diagonal elements
- Mapping $\Lambda(\cdot)$ is parameterized by graph neural network
- The training objective is to predict an incomplete factorization of the matrix A subject to sparsity constraints:

$$\begin{aligned} min._\theta \quad & \mathbb{E}_A ||\Lambda_\theta(A)\Lambda_\theta(A)^T - A||_F \\ s.t. \quad & \Lambda_\theta(A)_{ij} = 0 \quad \text{if} \quad A_{ij} = 0 \end{aligned}$$

- The lower triangular output matrix can be efficiently inverted using forward-backward substitution
- Additional non-zero elements can be obtained by relaxing the sparsity constraint (e.g. allowing non-zeros for A^2)

Graph neural network architecture

- The problem matrix A is interpreted as the adjacency matrix of the graph (Coates graph representation)
- Implicit node ordering by learned factorization
- Message passing is executed in two steps:
  - Use the lower triangular part of A for message passing
  - Use the upper triangular part of A for message passing
- This aligns the network architecture with the problem structure
- An $exp(\cdot)$ activation function on the diagonal elements is used to ensure positive definiteness

Background: Conjugate gradient method

- Iterative method for symmetric and positive definite (spd) linear equations
- Method of choice for large-scale and sparse problems
- Convergence depends on the spectral properties of the matrix A:

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

- Faster convergence is obtained by solving a preconditioned system:

$$P^{-1}Ax = P^{-1}b$$

  where $P^{-1} \approx A^{-1}$ is the preconditioner
- Trade-off between time required to compute the preconditioner P^{-1} and resulting speedup
- Extreme cases: $P^{-1} = A^{-1}$ (direct method) and $P^{-1} = I$ (no speedup)
- Typical preconditioners are often hand-engineered and domain specific: e.g. Jacobi, incomplete Cholesky, multigrid methods

Experiments & Results

- **Synthetic problem**: Random matrix $AA^T + \alpha I$
- Trade-off between time to compute preconditioner and saved iterations
- Preconditioned CG requires additional matrix multiplications (Total time = P-time + CG-time)

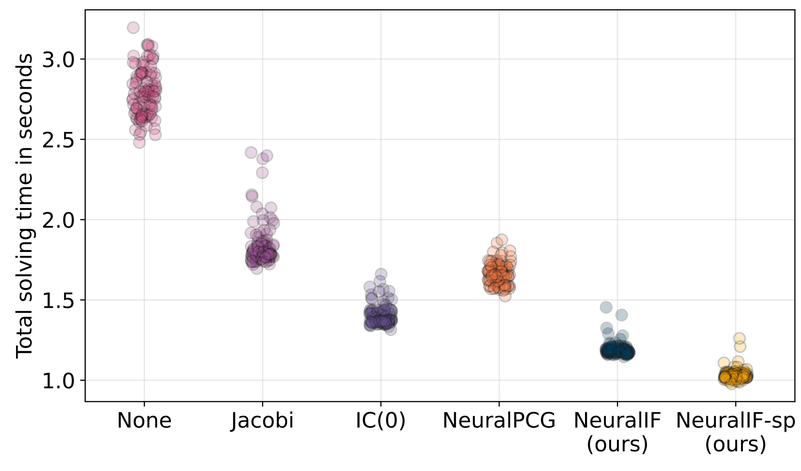


Figure: Comparison of different preconditioner performance on $10\,000 \times 10\,000$ matrices

- **Poisson PDE**: Discretization of the Poisson equation on varying grids
- Generalization to problems up to size $500\,000 \times 500\,000$

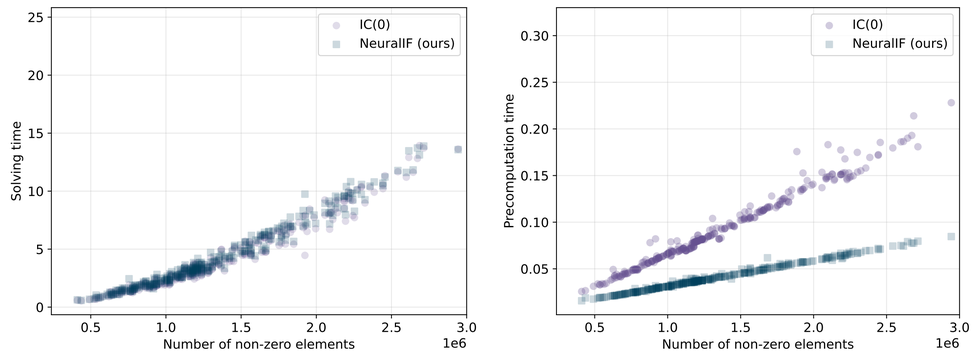


Figure: Precomputation and CG-time vs. the number of non-zero elements in the matrix.

- **The NeuralIF preconditioner is competitive with hand-engineered preconditioners across different tasks**

Summary & Conclusion

- Heuristics allow an easy integration of machine learning and classical optimization
- Learned optimization requires large amounts of data to be efficient (both training and amortizing the cost)
- Graph neural networks are natural computational backends for linear algebra and learned optimization
- Future research includes relaxing the constraints in the optimization problem and extending the setting to more general iterative methods (GMRES)

preprint

WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

paul.hausner@it.uu.se, ozan@kth.se, jens.sjolund@it.uu.se