Neural Network Preconditioners for 2D U(1) Wilson-type Dirac Operators

Brian Xiao, Phiala Shanahan, Daniel Hackett, Salvatore Cali, and Yin Lin

Massachusetts Institute of Technology

Introduction

A large part of the cost of a lattice QCD computation: calculation of propagators

- Requires solving a Dirac operator *D* onto sources
- Large systems → direct solution is impractical; must use iterative solvers (e.g. CG) instead
- Can improve solver performance by preconditioning (e.g. AMG, IC, Jacobi)

Idea: train a neural network to produce sparse preconditioners based on input operators

- One-time high training cost
- Cheap preconditioner generation once network is trained

Preconditioning

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \Longrightarrow \quad \mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$$

$$\kappa(\mathbf{A}) \qquad\qquad \kappa(\mathbf{M}^{-1}\mathbf{A}) \ll \kappa(\mathbf{A})$$

Solver convergence controlled by condition number κ

Neural network: maps $\boldsymbol{A} \rightarrow \boldsymbol{M}^{-1}$

Loss function: $\kappa(\boldsymbol{M}^{-1}\boldsymbol{A})$ or suitable substitute

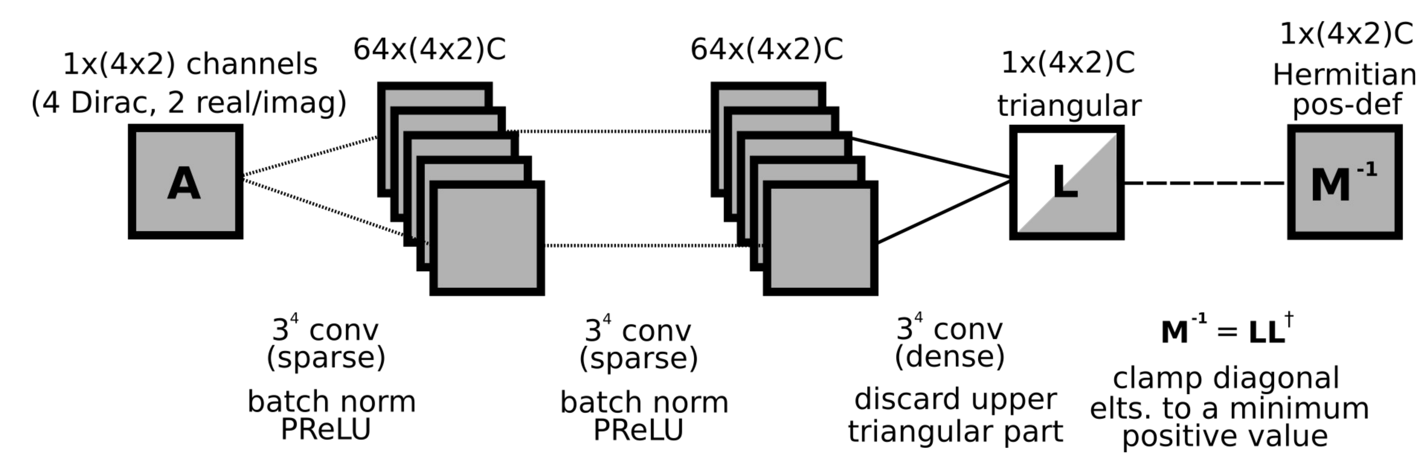
Desirable characteristics for $\boldsymbol{M}^{-1}$:

- approximates inverse of $\boldsymbol{A}$
- computationally cheap to apply (e.g. sparse)

Another simple preconditioning scheme: even-odd. Based on subspace decomposition; cuts problem difficulty by ~50%

Architecture & Training

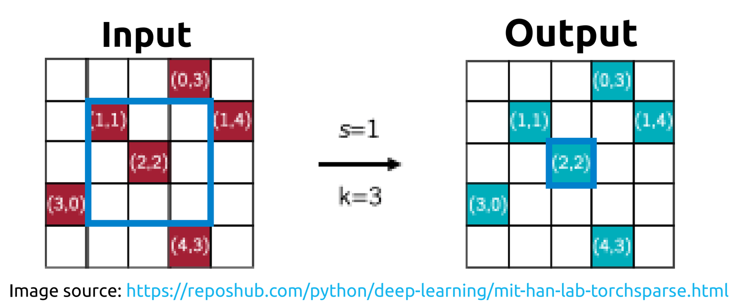
Network architecture:



- 4D convolutions over spatial indices (**x**, **y**); Dirac indices and real/imag parts in channels
- 2D: γ_μ is 2x2 → Dirac indices μ, ν = 0,1

Sparse convolutions:

Each output entry: weighted sum of nearby input entries, parameterized by trainable convolution kernel



Sparse convolutions: only update *nonzero* input entries – preconditioner will be sparse if input Dirac operator is sparse

No fixed size: single network can produce preconditioners for any lattice volume

Loss function: K-condition number; cheaper to compute and better training performance than κ

$$K(\mathbf{A}) = \frac{\operatorname{tr}\mathbf{A}}{n(\det\mathbf{A})^{1/n}} \qquad \operatorname{loss}(\mathbf{A},\mathbf{M}^{-1}) = \frac{K[(\mathbf{M}^{-1})^{\dagger}\mathbf{A}^{\dagger}\mathbf{A}\mathbf{M}^{-1}]}{K(\mathbf{A}^{\dagger}\mathbf{A})} = \frac{\operatorname{tr}[(\mathbf{M}^{-1})^{\dagger}\mathbf{A}^{\dagger}\mathbf{A}\mathbf{M}^{-1}]}{(\operatorname{tr}\mathbf{A}^{\dagger}\mathbf{A})|\det\mathbf{M}^{-1}|^{2/n}}$$

Training parameters:

With and without even-odd preprocessing

Dirac operator *D* from pure-gauge configurations; quenched approximation. Two datasets:

1) κ=0.24: $A = DD^{\dagger}$ (solved w/ CG)

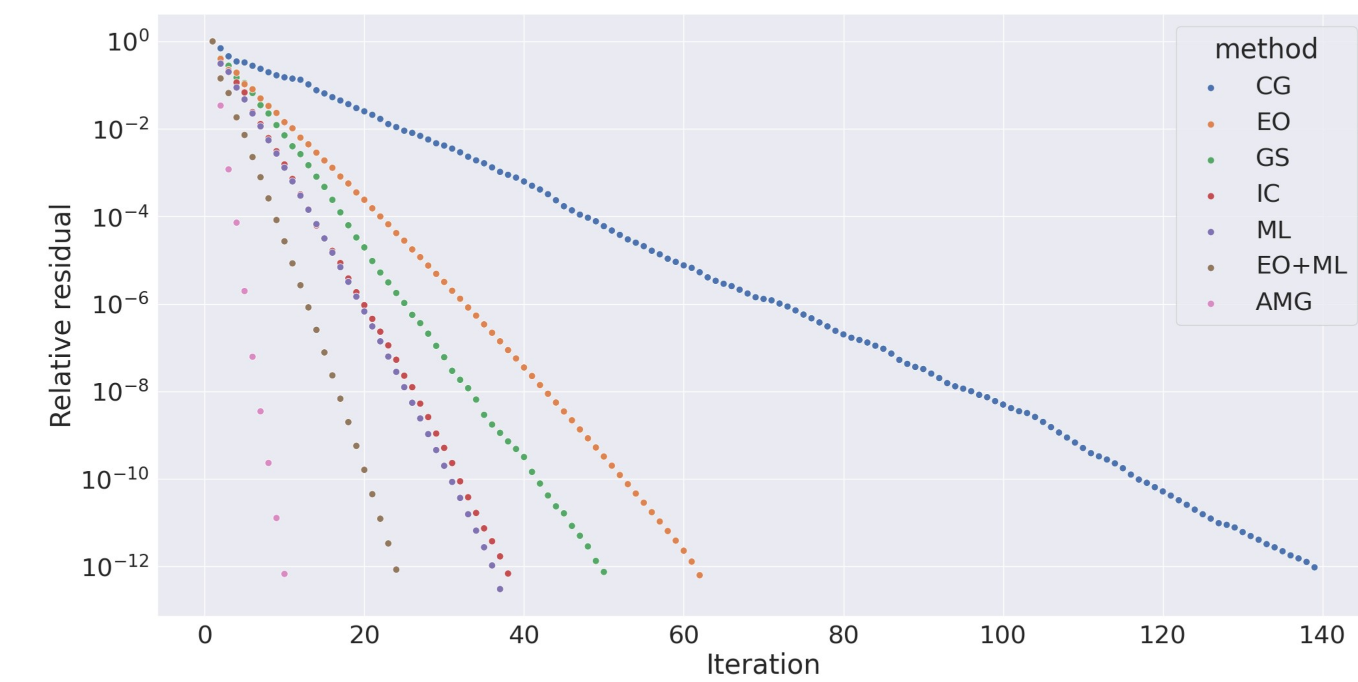
2) κ=0.32: $A = D\gamma_5$ (solved w/ BiCGSTAB)

- 2800 train / 200 validate / 1000 test

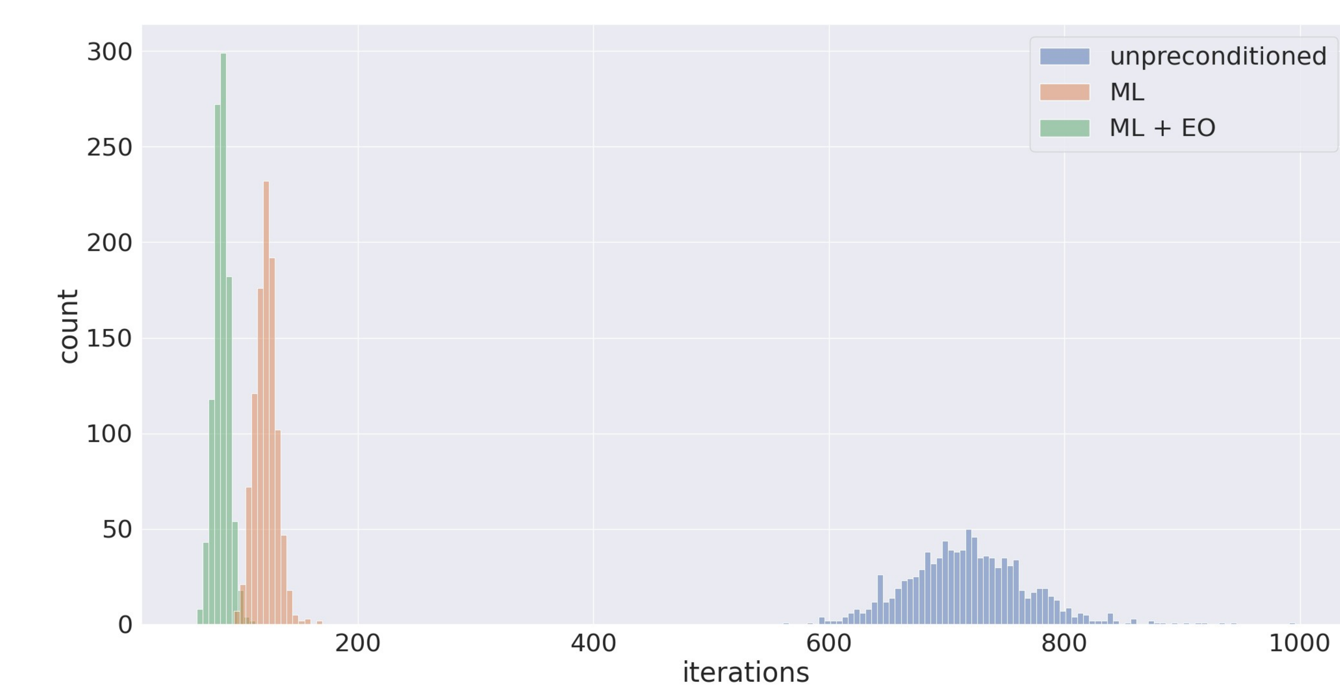
Adam optimizer @ lr 2 x 10^{-5}, weight decay 10^{-5}

50 epochs; batch size 8 x 4 processes

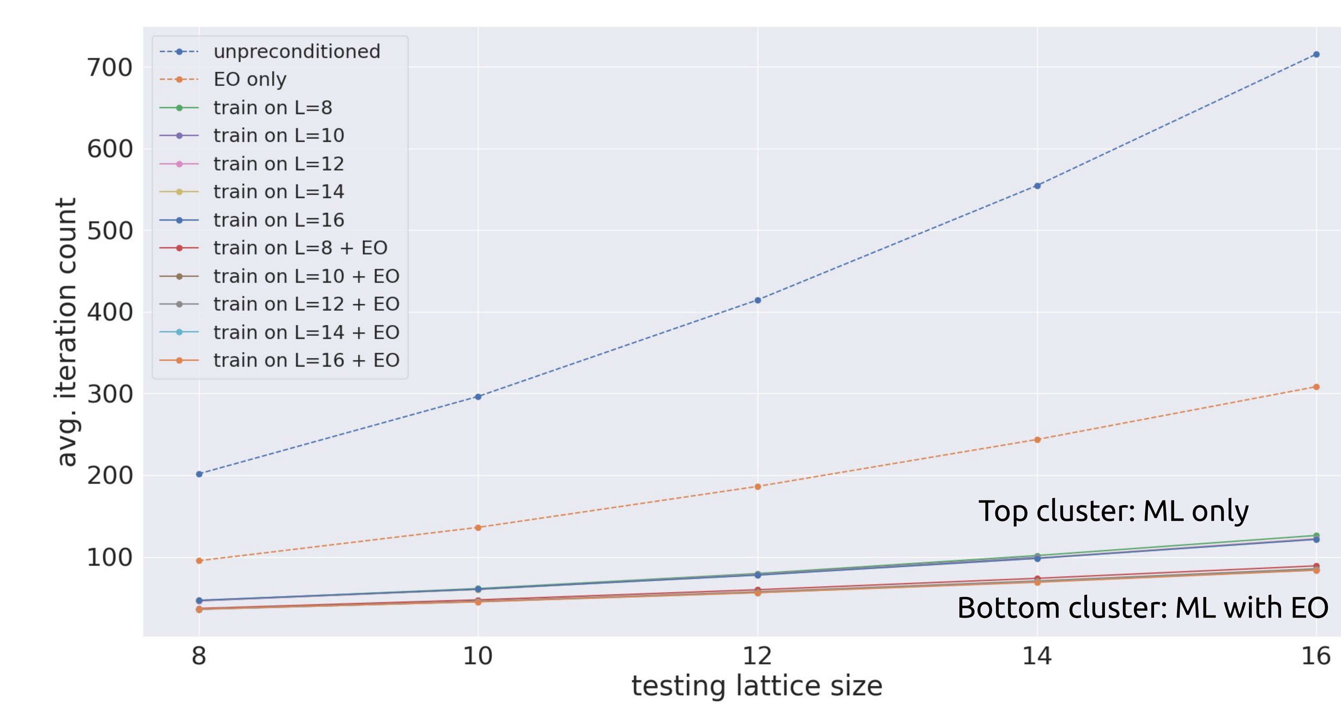
Results



CG convergence, one sample configuration
Dataset: #1; L = 16; solver tolerance 10^{-12}



Iteration count histogram
Dataset: #2; L = 16



Volume scaling: apply network across different lattice sizes without any retraining
Dataset: #2