# Faster Optimization on Sparse Graphs via Neural Reparametrization

**Csaba Both**[*] Northeastern University both.c@northeastern.edu

**Nima Dehmamy**[*] IBM Research nima.dehmamy@ibm.com

**Jianzhi Long** UCSD jlong@ucsd.edu

**Rose Yu** UCSD roseyu@ucsd.edu

## Abstract

Optimization on sparse graphs underlies many scientific challenges, such as heat diffusion in solids and synchronization of nonlinear oscillators. Conventional methods often suffer from slow convergence, especially in complex, random graph structures typical of glassy systems. We introduce a novel approach using Graph Neural Networks (GNN) to reparametrize the state of each node as a network output, transforming the optimization variables into the GNN weights and high-dimensional node embeddings. This reparametrization effectively preconditions the optimization landscape, leveraging the approximate Hessian to accelerate convergence—akin to quasi-Newton methods. Our experiments demonstrate significant speed enhancements in solving the heat equation with boundary conditions on 2D and 3D graphs and in achieving synchronization in the Kuramoto model. Our work both provides a theoretical analysis for the observed performance gains and also a framework for improving optimization techniques in graph-based problems.

## 1 Introduction

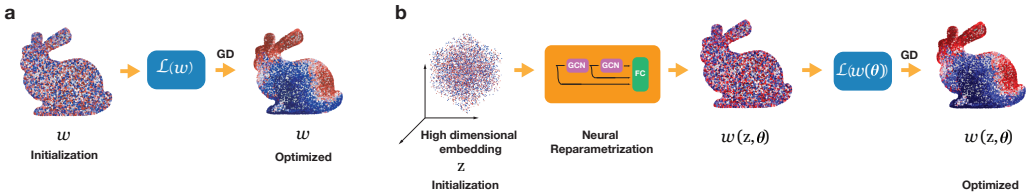


**Figure 1:** Original (a) vs Neural Reparametrization using GNN (b). In problems on sparse graphs, using the Hessian inside the GNN implements a quasi-Newton method and accelerates optimization.

Dynamical processes on graphs are ubiquitous in real-world, such as traffic flow on road networks, epidemic spreading on mobility networks, and heat diffusion in material. They also frequently appear in machine learning applications including in fluid simulations Ummenhofer et al. (2019); Pfaff et al. (2020), topological data analysis Carriere et al. (2021); Birdal et al. (2021), and solving differential equations Zobeiry and Humfeld (2021); He and Pathak (2020); Schnell et al. (2021). Many of these challenges can be formulated as optimization problems on graphs, which are frequently highly non-convex. Gradient-based methods are often used for numerical optimization, but they also suffer from slow convergence on large graphs Chen et al. (2018).

In this paper, we propose a novel *neural reparametrization* which can significantly accelerate optimization on sparse graphs. Our key idea is to reparametrize the optimization variables (state of the nodes) as the output of a graph neural network (GNN) during optimization. Instead of optimizing the original variables of the problem, we optimize a latent embedding for each node and the weights of the GNN. We demonstrate that our neural reparametrization utilizes the approximate Hessian, making

[*]Equal contribution

C. Both et al., Faster Optimization on Sparse Graphs via Neural Reparametrization. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

our approach akin to a quasi-Newton method, resulting in significant speedup. This is particularly advantageous for large, sparse graphs, where GNN reparametrization can be implemented efficiently. We also find comparable speedup using a node-wise neural network without the graph (i.e. using an empty graph in the GNN reparametrization). We demonstrate that for solving heat diffusion on graphs and synchronization of nonlinear oscillators, our method leads to significant speed-up (2-50$\times$ in our experiments).

Since our GNN reparametrization effectively uses the approximate Hessian, it is fair to compare it with other efficient second-order (quasi-Newton) methods. However, existing second-order methods such as KFAC (Martens and Grosse, 2015) and Shampoo (Gupta et al., 2018; Anil et al., 2020) are tailored for deep neural networks. The problems we consider only consist of an input state vector and a loss function (i.e. shallow network). Hence, Shampoo and K-FAC are not expected to be very effective for these sparse graph optimization problems. In our experiments, we compared our model against Shampoo as one of the baselines but did not observe Shampoo to yield speed-up comparable to our neural reparametrization approach.

In summary, we show that

1. In optimization problems on sparse graphs, neural reparametrization can yield significant reduction in convergence time.
2. We show the effectiveness of this method on two scientifically relevant problems: heat diffusion, and synchronization. We observe over an order of magnitude speed-up in graphs of various sizes.
3. We show that in early optimization steps, a GNN reparametrization uses an approximate Hessian, in a way similar to quasi-Newton methods.

## 2 Related Work

**Graph Neural Networks.** Our method demonstrates a novel and unique perspective for GNN. The majority of literature uses GNNs to learn representations from graph data to make predictions, see surveys and the references in Bronstein et al. (2017); Zhang et al. (2018); Wu et al. (2019); Goyal and Ferrara (2018). Recently, (Bapst et al., 2020) showed the power of GNN in predicting long-time behavior of glassy systems, which are notoriously slow and difficult to simulate. Additionally, (Fu et al., 2022) showed that GNN-based models can help speedup simulation of molecular dynamics problems by predicting large time steps ahead. However, we use GNNs to modify the learning dynamics of sparse graph optimization problems. We discover that reparametrizing optimization problems leads to a significant speedup in convergence. Indeed, we show analytically that a GNN with a certain aggregation rule achieves the same optimal adaptive learning rate as in (Duchi et al., 2011). Thanks to the sparsity of the graph, we can obtain an efficient implementation of the optimizer that mimics the behavior of quasi-Newton methods.

**Neural Reparametrization.** Reparameterizing an optimization problem can reshape the landscape geometry, and change the learning dynamics, hence speeding-up convergence. In linear systems, preconditioning Axelsson (1996); Saad and Van Der Vorst (2000) reparameterizes the problem by multiplying a fixed symmetric positive-definite preconditioner matrix to the original problem. Groeneveld (1994) reparameterizes the covariance matrix to allow the use a faster Quasi-newton algorithm in maximum likelihood estimation. Recently, an *implicit acceleration* has been documented in over-parametrized linear neural networks and analyzed in Arora et al. (2018); Tarmoun et al. (2021). Specifically, Arora et al. (2018) shows that reparametrizing with deep linear networks imposes a preconditioning scheme on gradient descent. Other works (Sosnovik and Oseledets, 2019; Hoyer et al., 2019; Both et al., 2023) have demonstrated that reparametrizing with convolutional neural networks and GNNs can speed-up structure optimization problems (e.g. designing a bridge) and network visualization. However, the theoretical foundation for the improvement is not well understood. To our knowledge, the use of GNNs for reparametrization and acceleration of general graph optimization problems has not been systematically studied.

**PDE Solving.** Our work can also be viewed as finding the steady-state solution of non-linear discretized partial differential equations (PDEs). Traditional finite difference or finite element methods Forsythe and Wasow (1960); Zienkiewicz et al. (2005) for solving PDEs are computationally

challenging. Several recent works use deep learning to solve PDEs in a *supervised* fashion. For example, physics-informed neural network (PINN) Raissi et al. (2019); Greenfeld et al. (2019); Bar and Sochen (2019) parameterize the solution of a PDE with neural networks. Their neural networks take as input the physical domain. Thus, their solution is independent of the mesh but is specific to each parameterization. Neural operators Lu et al. (2021); Li et al. (2020a,b) alleviate this limitation by learning in the space of infinite dimensional functions. However, both class of methods require data from the numerical solver as supervision, whereas our method is completely *unsupervised*. We solve the PDEs by directly minimizing the energy function.

**Combinatorial optimization.** The intersection of Graph Neural Networks (GNNs) and combinatorial optimization has emerged as a transformative approach to solving complex graph-based problems. Several studies have contributed to this field. For example, Khalil et al. (2017) laid the foundation by combining GNNs with reinforcement learning to tackle NP-hard problems. Kool et al. (2018) further advanced this work by incorporating attention mechanisms for the Traveling Salesman Problem. The practical impact of these theoretical developments was highlighted by Mirhoseini et al. (2017), who demonstrated significant performance improvements in chip placement optimization over traditional methods. Additionally, Xu et al. (2019) provided key theoretical insights into the reasoning capabilities of GNNs, formally establishing their computational boundaries. Building on these efforts, Cappart et al. (2023) conducted a comprehensive analysis that unified existing approaches. Their work demonstrated how GNNs can effectively learn and reason about combinatorial optimization problems while preserving problem-specific constraints. This represents a significant step toward developing more efficient and adaptable solutions for complex graph optimization challenges.

## 3 Graph Optimization

**Problem Structure.** Assume we have a graph with $n$ nodes. The edge data is captured in an adjacency matrix $A \in \mathbb{R}^{n\times n}$. We assume each node $i$ has a state (feature) vector $\boldsymbol{w}_i \in \mathbb{R}^d$. The optimization problems we consider are tasked with finding the set of node states $\boldsymbol{w} \in \mathbb{R}^{n\times d}$ that minimize the following energy function:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{ij} A_{ij} f(\|\boldsymbol{w}_i - \boldsymbol{w}_j\|^2) + g(\boldsymbol{w}). \tag{1}$$

where $f$ and $g$ are smooth functions.

The first term represents "diffusion", where neighboring nodes diffuse to each other. The diffusion term explicitly depends the graph structure via $A_{ij}$. The second term may denote any global, graph independent interaction between the nodes or with the environment, e.g. heat bath, external magnetic field, or all-to-all interactions. Since $\|\boldsymbol{w}_i - \boldsymbol{w}_j\| = \|\boldsymbol{w}_j - \boldsymbol{w}_i\|$, only the symmetric part of the $A$ contributes to equation 1 and so we can assume $A_{ij} = A_{ji}$ (i.e. undirected graph). When $f(x) \approx x$, the first term in equation 1 describes a diffusion process on a graph:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) &\approx \sum_{ij} A_{ij}\|\boldsymbol{w}_i - \boldsymbol{w}_j\|^2 + g(\boldsymbol{w}) \\ &= \mathrm{Tr}\left[\boldsymbol{w}^T L \boldsymbol{w}\right] + g(\boldsymbol{w})\end{aligned} \tag{2}$$

where $L = D - A$ is the graph Laplacian, $D_{ij} = \sum_k A_{ik}\delta_{ij}$ the diagonal degree matrix and $\delta_{ij}$ is the Kronecker delta. Problems described by equation 1 include reaction-diffusion dynamics Turing (1990). Next, we consider two examples: heat diffusion processes and the Kuramoto oscillators, which can be highly nonlinear synchronization problems.

### 3.1 Heat Diffusion

The heat equation describes how heat diffuses in a medium Incropera et al. (1996); Du and Zaki (2021); Zhao et al. (2021) with broad applications in fluid mechanics, atmospheric and climate science. It is given by

$$\partial_t \boldsymbol{w}(x,t) = -\varepsilon \nabla^2 \boldsymbol{w}(x,t) + F_{\mathrm{bath}}(\boldsymbol{w}) \tag{3}$$

where $\boldsymbol{w}(x,t)$ represents the temperature at point $x$ at time $t$, and $\varepsilon$ is the heat diffusion constant. $F_{\mathrm{bath}}$ can be a set of constraints from being connected to a heat bath (source or sink of thermal

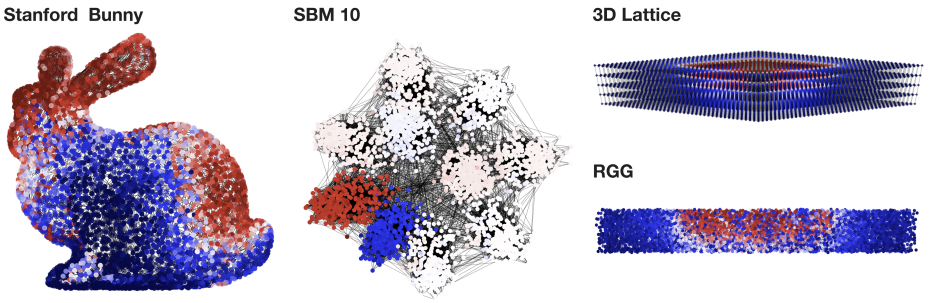


**Figure 2:** Optimized heat distribution on the Stanford bunny, Stochastic Block Model with 10 blocks (SBM 10), 3D lattice and Random Geometric Graphs (RGG).

energy). On a graph, $\boldsymbol{w}(x,t) \in \mathbb{R}^+$ is discretized and replaced by $\boldsymbol{w}_i(t)$, with node $i$ representing the position, and the Laplacian operator $\nabla^2$ becomes the graph Laplacian $L = D - A$. The heat diffusion in equation 3 on graphs becomes $d\boldsymbol{w}/dt = -\varepsilon L\boldsymbol{w} + F_{\text{bath}}(\boldsymbol{w})$. The energy function of the heat diffusion has the form

$$\begin{aligned}\mathcal{L}_{HE}(\boldsymbol{w}) &= \frac{1}{2}\boldsymbol{w}^T L\boldsymbol{w} - V_{\text{Bath}} \\ &= \frac{1}{2}\sum_{ij} A_{ij}\|\boldsymbol{w}_i - \boldsymbol{w}_j\|^2 + V_{\text{bath}}\end{aligned} \tag{4}$$

where $F_{\text{bath}} = -\nabla V_{\text{bath}}$. For instance, if a set of nodes, denoted as **Hot**, are attached to a source that keeps them at temperature $T_{hot}$ and some others, denoted as **Cold** to a cold source at temperature $T_{cold}$, the regularizer can have the form

$$V_{\text{bath}} = \lambda \sum_{i\in\textbf{Hot}} \|\boldsymbol{w}_i - T_{hot}\|^4 + \lambda \sum_{i\in\textbf{Cold}} \|\boldsymbol{w}_i - T_{cold}\|^4 \tag{5}$$

We are interested in the long-term distribution of heat $\boldsymbol{w}(t \to \infty)$, which can be obtained by performing gradient descent (GD) on equation 4. However, finding $\boldsymbol{w}(t \to \infty)$ on large meshes, lattices or amorphous, glassy systems, or systems with bottlenecks (e.g. graph with multiple clusters with bottlenecks between them) can be prohibitively slow Vincent et al. (1997); Rocha et al. (2011). We show that GNN reparametrization can significantly accelerate convergence of GD and finding the steady state solutions $\boldsymbol{w}(t \to \infty)$.

### 3.2 Kuramoto Model and Synchronization

Network synchronization (Pikovsky et al., 2003) optimizes a network of coupled dynamical systems until they reach at the same frequency, known as the *synchronization* phenomena. These problems have a profound impact on engineering, physics, and machine learning Schnell et al. (2021).

An important model for studying the synchronization phenomena is the Kuramoto model (Kuramoto, 1975, 1984). It has had a profound impact on engineering, physics, machine learning Schnell et al.

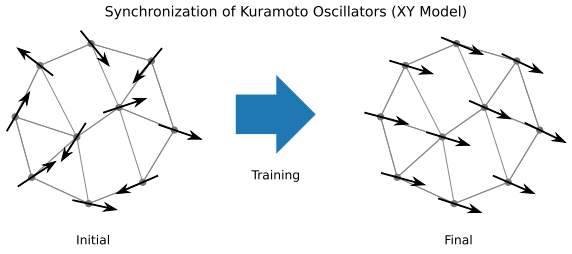


**Figure 3:** Illustration of Kuramoto oscillators. Each node has a phase angle, represented by an arrow. As the energy is minimized, the phase angles at nodes connected to each other start aligning (synchronization).

**input :** Graph $A$, energy function $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{w} \in \mathbb{R}^{n\times d}$, and $t_{MAX}$.
**output :** $\boldsymbol{w} = \text{argmin}_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$
Initialize random latent states $\boldsymbol{z} \in \mathbb{R}^{n\times h}$;
Construct $\mathbf{GNN}(A, \theta) : \mathbb{R}^{n\times h} \to \mathbb{R}^{n\times d}$ with learnable parameters $\theta$ ;
; `/* GCN using A with residual connections. */`
**while** *not* ***GNN_stopping_criterion***$(\mathcal{L}(\boldsymbol{w}_t))$ *and* $t < t_{GNN}$ **do**
  $\boldsymbol{w}_t \leftarrow \mathbf{GNN}(A, \theta_t)(\boldsymbol{z}_t)$;
  ; `/* gradient descent on z and θ */`
  $\boldsymbol{z}_{t+1} \leftarrow \boldsymbol{z}_t - \varepsilon\nabla_{\boldsymbol{z}_t}\mathcal{L}(\boldsymbol{w}_t)$;
  $\theta_{t+1} \leftarrow \theta_t - \varepsilon\nabla_{\theta_t}\mathcal{L}(\boldsymbol{w}_t)$;
**end**
; `/* Switch to GD for w without GNN */`
$\boldsymbol{w}_t \leftarrow \text{GNN}_{\theta_t}(\boldsymbol{z}_t)$; `/* Get last output of GNN */`
**while** *not* ***Stopping_criterion***$(\mathcal{L}(\boldsymbol{w}_t))$ *and* $t < t_{MAX}$ **do**
  $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \varepsilon\nabla_{\boldsymbol{w}_t}\mathcal{L}(\boldsymbol{w}_t)$;
**end**

**Algorithm 1: Neural Reparametrization**

(2021) and network synchronization problems (Pikovsky et al., 2003) in social systems. The loss function for the Kuramoto model is defined as

$$\mathcal{L}_{\text{KM}}(\boldsymbol{w}) = -\sum_{i,j} A_{ji}\cos\Delta_{ij}, \qquad \Delta_{ij} = \boldsymbol{w}_i - \boldsymbol{w}_j. \tag{6}$$

which can be derived from the misalignement $\|x_i - x_j\|^2 = 2\left[1 + \cos(\boldsymbol{w}_i - \boldsymbol{w}_j)\right]$ between unit 2D vectors $x_i$ representing each oscillator. Figure 3 shows a sketch of this model. The oscillators are nodes on a graph and the phases $\boldsymbol{w}_i$ are shown as 2D vectors. The vectors are initially randomly oriented. After synchronization (minimizing $\mathcal{L}$), the vectors become aligned. Thus, the loss function for the Kuramoto model is identical to the energy function of interacting 2D spins, also known as the "XY model" in physics. Minimizing $\mathcal{L}_{\text{KM}}$ using gradient-based methods follows the gradient flow equation $d\boldsymbol{w}_i/dt = -\varepsilon\sum_j A_{ij}\sin\Delta_{ij}$, which is highly nonlinear. We will show in experiments that we observe considerably faster convergence in the Kuramoto model using our method compared to SGD or other adaptive gradient methods.

**The Hopf-Kuramoto Model.** We further consider a more complex version of the Kuramoto model: the Hopf-Kuramoto (HK) model Lauter et al. (2015). This model includes second-neighbor effects via $A_{ij}A_{jk}$ terms. The HK model is interesting for assessing the performance of our method because it includes phases and regimes in which conventional methods become extremely slow. The richness of the phase space of the HK model can be seen in Fig. 7. This diversity of phases allows us to showcase our method's performance in different parameter regimes and in highly nonlinear scenarios. The loss function for the HK model is

$$\mathcal{L}_{\text{HK}} = \sum_{i,j} A_{ji}\left[\sin\Delta_{ij} + s_1\cos\Delta_{ij}\right] + \frac{s_2}{2}\sum_{i,k,j} A_{ij}A_{jk}\left[\cos\left(\Delta_{ji}+\Delta_{jk}\right) + \cos\left(\Delta_{ji}-\Delta_{jk}\right)\right] \tag{7}$$

where $s_1, s_2$ are model constants determining the phases of the system. We observe significant speed-up using our model in most phases of the HK model, as reported in the experiments.

## 4 Neural Reparametrization

Our optimization objective is to minimize the energy function of the form in equation 1. A common approach is to use gradient descent (GD) $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \varepsilon\nabla\mathcal{L}$ to minimize such a loss function. However, GD can take many steps to converge on these graph problems. We describe a novel optimization scheme that reparametrizes the optimization variables $\boldsymbol{w}$ with a GNN. We also show that such a reparametrization has a similar effect to preconditioning.

**GNN reparametrization.** We propose to reparametrize the optimization variables (state of the nodes) $\boldsymbol{w} \in \mathbb{R}^{n\times d}$ as the output of a GNN, $\boldsymbol{w}(z,\theta)$ or $\boldsymbol{w}(\boldsymbol{\theta})$ for short (Algorithm 1). We first randomly initialize a set of latent states $z \in \mathbb{R}^{n\times h}$ as the high-dimensional embedding of the node states. We construct a GNN whose outputs are the optimization variables $\boldsymbol{w}$ and $\theta$ are the weights. After reparametrization, we change the problem from optimization $\boldsymbol{w}$ to optimizing $\boldsymbol{\theta}$. Figure 1 visualizes the pipeline of the proposed approach.

**NoGraph reparametrization.** We introduce a second neural reparametrization, which does not use graph, denoted by "NoGraph", which is similar to our GNN model, but the graph adjacency is set to $A = I$ identity. Next, we describe the details of the algorithm.

**Architecture.** We use graph convolutional networks (GCN) (Kipf and Welling, 2016) with residual connections to build our GNN. We choose GCN because its propagation rule allows us to implement polynomials of Hessian easily. The idea is that incorporating the Hessian can have an effect similar to quasi-Newton method as we discuss in the next section. To minimize the computational overhead from the GNN, we use only one or two layers of GCN. Our reparametrization begins with a high dimensional embedding $\boldsymbol{z} \in \mathbb{R}^{n\times h}$ for $n$ nodes. The first GCN layer takes $\boldsymbol{z}$ and returns $\sigma(f(A)\boldsymbol{z}\theta)$, where we use $f(A) = A_s \equiv D^{-1/2}AD^{-1/2}$ as the propagation (aggregation) rule. Here $\theta \in \mathbb{R}^{h\times h_1}$ are the learnable weights of the GCN and $\sigma = \text{LeakyReLU}$ is the nonlinearity.

**Early stage Hessian.** The Hessian $\mathcal{H} = \partial^2\mathcal{L}/\partial\boldsymbol{w}^2$ of equation 1 can be sparse when the graph is sparse (see appendix 6.1 for details). Assume we initialize $\boldsymbol{w} \sim \mathcal{N}(0, 1/n)$ as a normal distribution. Using moments of the normal distribution, we can show that the expected value for the components of the Hessian $\mathcal{H}_{ij} = \partial_i\partial_j\mathcal{L}$ of the loss in equation 1, at early stages is given by

$$\mathbb{E}[\mathcal{H}_{ij}] \approx \partial_i\partial_j g(0) - f'(0)L_{ij} + O\left(1/n\right) \tag{8}$$

where $L = D - A$ is the Laplacian. In the problems we consider we expect the diffusion term $A_{ij}f(\Delta_{ij}^2)$ dominates over $g(\boldsymbol{w})$ for random $\boldsymbol{w}$, up to a constant factor, $\mathcal{H} \approx L = D - A$ from equation 8. To avoid divergences we can normalize by the degree $\boldsymbol{H} \equiv D^{-1/2}\mathcal{H}D^{-1/2} \approx I - A_s$. Hence, since the propagation rule of our GNN is $A_s$, it effectively uses the Hessian at initialization.

**Two-stage optimization.** As we showed above, the GNN effectively uses the Hessian at early stages. However, the Hessian changes during optimization, making the initial Hessian less relevant. This, combined with the extra computation cost of the forward pass through GNN led us to adopt a two-stage optimization. After optimizing the reparametrized $\boldsymbol{w}(\boldsymbol{\theta})$ until the change in loss becomes small, we remove the GNN reparametrization and switch to simple GD on $\boldsymbol{w}$, initialized using the final value of $\boldsymbol{w}(\boldsymbol{\theta})$, to fine-tune. This hybrid approach works very well in our experiments and led to significant speedup.

**Per-step Time Complexity.** Let $\boldsymbol{w} \in \mathbb{R}^{n\times d}$ with $d \ll n$ and let the average degree of each node be $k = \sum_{ij} A_{ij}/n$. For a sparse graph we have $k \ll n$. Assuming the leading term in the loss $\mathcal{L}(\boldsymbol{w})$ is as in equation 2, the complexity of computing $\nabla\mathcal{L}(\boldsymbol{w})$ is at least $O(dn^2k)$, because of the matrix product $L\boldsymbol{w}$. When we reparametrize to $\theta \in \mathbb{R}^{n\times h}$ with $h \ll n$, passign through each layer of GCN has complexity $O(hn^2k)$. The complexity of GD on the reparametrized model with $l$ GCN layers is $O((lh + d)n^2k)$. Thus, as long as $lh$ is not too big, the reparametrization slows down each iteration by a constant factor of $\sim 1 + lh/d$. Next, we provide a theoretical argument for why our GNN reparametrization should lead to speedup.

### 4.1 Theoretical Analysis

We now show that the GNN reparametrization has the effect of preconditioning. Moreover, we show that in the graph problems we consider here, this preconditioner is a function of the approximate Hessian of the loss. We also show that in certain cases, this reparametrization is equivalent to a quasi-Newton methods.

**Reparametrization and to Preconditioning.** We are performing GD on the parameters $\theta$ of the GNN. Denoting $\delta\boldsymbol{\theta} = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$, the GD update rule given by

$$\delta\boldsymbol{\theta}_a = -\sum_b \hat{\varepsilon}_{ab}\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}_b} = -\sum_b \hat{\varepsilon}_{ab}\sum_i \frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_i}\frac{\partial\boldsymbol{w}_i}{\partial\boldsymbol{\theta}_b} = -\left[\hat{\varepsilon}J\nabla_{\boldsymbol{w}}\mathcal{L}\right]_a \tag{9}$$

where $J = \partial\boldsymbol{w}/\partial\boldsymbol{\theta}$ is the Jacobian. The consequence of equation 9 is that the $\boldsymbol{w}$ update rule changes from $\delta\boldsymbol{w} = -\varepsilon\nabla\mathcal{L}$ to

$$\delta\boldsymbol{w} = \frac{\partial\boldsymbol{w}}{\partial\boldsymbol{\theta}}^T\frac{d\boldsymbol{\theta}}{dt} = -J^T\hat{\varepsilon}J\nabla_{\boldsymbol{w}}\mathcal{L} \tag{10}$$

meaning that reparametrization is preconditioning the GD equation of $\boldsymbol{w}$ by the Jacobian of the reparametrization neural network. As we argued above, when diffusion on the graph is a dominant term in the loss, the Hessian is related to the graph structure as $\boldsymbol{H} \approx I - A_s$. Hence, the preconditioner resulting from the GNN reparametrization is a function of the approximate Hessian. Next, we show that, for a specific set of weights, the GNN preconditioner can approximate $\mathcal{H}^{-1}$ in the early stages of the optimization, meaning it can act as a quasi-Newton method.

**Relation to Quasi-Newton methods.** Recall that our GNN consists of GCN layers performing $\sigma(A_s\boldsymbol{z}\theta$ and $\boldsymbol{H} \approx I - A_s$. In equation 10, if the preconditioner $J^T\varepsilon J \approx \boldsymbol{H}^{-1}$, the GNN will essentially have the same effect using Newton's method. Assuming $\varepsilon = 1$, to get a quasi-Newton method we'd need $J \approx \boldsymbol{H}^{-1/2}$. Next, we note that using the binomial expansion $\boldsymbol{H}^{-1/2} \approx I - \frac{1}{2}A_s - \frac{3}{4}A_s^2 + \ldots$. This binomial expansion can be implemented using GCN layers. A linear GCN layer with residual connections represents the polynomial $F(\boldsymbol{z}) = \sum_k f(A)^k\boldsymbol{z}V_k$ (Dehmamy et al., 2019). Hence, to get an order $q$ approximation for $J = \boldsymbol{H}^{-1/2}$ we can use a $q$-layer GNN and we can implement the $O(A_s^2)$ approximation $J \approx \boldsymbol{H}^{-1/2}$ using a two-layer GCN with pre-specified weights. However, the Hessian of the loss is not constant and changes during GD. To account for small changes in the Hessian during optimization, instead of fixing GCN weights to an approximation of $J \approx \boldsymbol{H}^{-1/2}$, we the GCN weights be trainable. Finally, we need to address the issue of computational complexity, discussed next.

**Exploiting Sparsity.** In the context of sparse graphs, our GNN reparametrization does not significantly increase computational complexity, even when approximating the inverse Hessian, $\mathcal{H}^{-1}$. While computing $\mathcal{H}^{-1}$ for a general $n \times n$ matrix is $O(n^3)$, the situation is more favorable for sparse matrices. Despite $\mathcal{H}$ being sparse, its inverse may become dense, potentially increasing the cost of applying Newton's method. However, computations within a $q$-layer GNN corresponds to evaluating a polynomial of order $q$ on the graph, where each layer's complexity mirrors that of multiplying sparse matrices. If $A$ is dense, the complexity under naive multiplication would be $O(qn^3)$, though in practice, algorithms like Strassen's reduce this cost. For a sparse $A$ with $k$ nonzeros per row, the complexity for $A^2$ is $O(nk^2)$, and extending to $A^q$ gives a pessimistic estimate of $O(nk^q)$. However, the growth in nonzeros often saturates, keeping the complexity manageable. For a two-layer GNN using such a sparse $A$, the complexity remains $O(nk^2)$, which is linear in $n$ when $k \ll n$. This supports our assertion that GNN reparametrization on sparse graphs introduces only a modest overhead, scaling linearly with $n$. Next, we present experimental results validating the benefit of our algorithm.

## 5 Experiments

We showcase the acceleration of neural reparametrization on two graph optimization problems: heat diffusion on a graph; and synchronization of oscillators. More experimental details, parameter sweeps and extra experiments on persistent Homology can be found in the appendix. We use the Adam Kingma and Ba (2014) optimizer and compare different reparametrization models. We implemented all models with Pytorch. Figure 4a summarizes the speedups (wall clock time to run original problem divided by time of the GNN model) we observed in all our experiments. We explain the two problems used in the experiments next.

### 5.1 Heat Diffusion

We perform experiments for heat diffusion on different graphs, including the Stanford Bunny, Stochastic Block Model (SBM), 2D and 3D lattices, and Random Geometric Graphs (RGG) Penrose

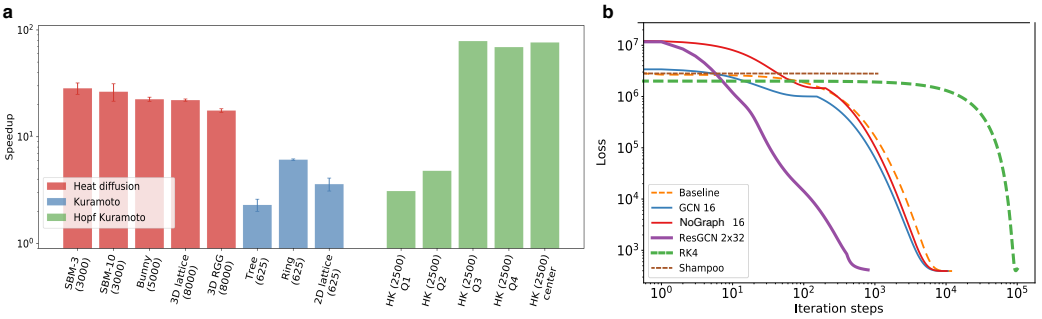


**Figure 4:** a) Summary of the speedup (wall-clock time for GD divided by our method) in different experiments. In all experiments, GD and our method used the same optimizers (Adam). The colors indicate the type of loss function, and the labels explain the graph structure. The numbers in parentheses are the number of vertices. For solving the heat diffusion, we used GCN reparametrization, while for the synchronization problem, we used GCN with residual connection. b) The loss curves for solving heat diffusion on a 2D lattice (2500) by the different models. It shows the advantages of reparametrization, and at the same time, it reveals the bottleneck of the Shampoo optimizer that cannot converge in a comparable time with other models.

(2003); Karrer and Newman (2011). For all graphs, we pick 10% of nodes and connect them to a hot source using the regularizer $\|\boldsymbol{w}_i - T_{hot}\|^4$, and 10% to the cold source using $\|\boldsymbol{w}_i - T_{cold}\|^4$ in equation 5. We also compare against different baselines, choosing a 2D lattice of size $50 \times 50$ as the graph, described in Fig. 4b.

**Results for heat diffusion.** Figure 4 summarizes the observed speedups. We find that on all these graphs, our method can speed up finding the final $\boldsymbol{w}$ by over an order of magnitude. Figure 2 shows the final temperature distribution in some examples of our experiments. SBM is model where the probability of $A_{ij} = 1$ is drawn from a block diagonal matrix. It represents a graphs with multiple clusters (diagonal block in $A_{ij}$) where nodes within a cluster are more likely to be connected to each other than to other clusters (Fig. 2, SBM 10). RGG are graphs where the nodes are distributed in space (2D or higher) and nodes are more likely to connect to nearby nodes. In our experimental setup, we use the earlier introduced hybrid optimization model with automatic switching criteria between the two optimization stages.

Figure 5 shows the results of more detailed experiments for the heat equation. All our experiments are compared against the simple GD baseline, without reparametrization. We use the Adam optimizer with a 0.02 learning rate for each simulation. Additionally, we compare both the speedup (wall-clock time for the linear model divided by the neural model) as well as the the final loss (relative to the linear baseline) on 2D lattice, SBM and RGG. Our neural reprameterization models consist of multi-layer GCN ("GCN": feedforward; "GCN Res": including residual connections to last layer), and a "NoGraph" model (also with and without residual connections), which is similar to our GCN models, but the graph adjacency is set to $A = I$ identity. We run these models with dimensions 16, 32, 64, and 128 (both for the initial latent vectors $\theta$ and the hidden widths). Finally, we also compare with the Shampoo Gupta et al. (2018) preconditioner used on the baseline model. We also

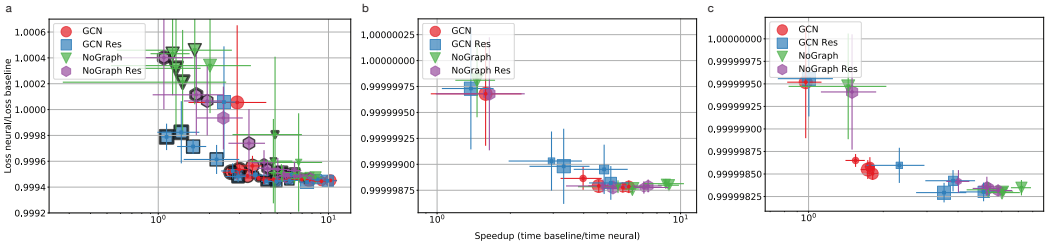


**Figure 5: Heat diffusion on graphs** ($N = 2500$)**.** Comparing the speedup (wall clock time) and the final energy (ratio to baseline model energy) for the following graphs: a) 2D lattice, b) random graph, and c) random geometric graph. a) Symbol border thickness shows model depth (i.e. no border is single layer, thin is 2 layers and thick is 3 layers). The node size increases with latent dimensions: 16, 32, 64, and 128. The x-axis shows the speedup (ratio of wall-clock times; higher is better), and the y-axis the loss relative to the baseline (lower is better). Another baseline, Runge Kutta (RK4), is more than 10 times slower than simple GD with Adam and we do not show it in this plot. In b and c, we only show the single-layer models.

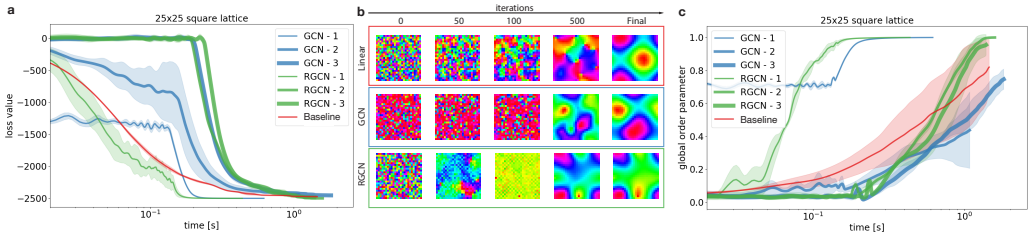


**Figure 6:** Kuramoto model on a $25 \times 25$ lattice (a) Loss over run time for different methods. (b) Evolution of $\boldsymbol{w}$ over iterations. (c) Level of synchronization, measured by global order parameter $\rho$ over time. Neural reparametrization achieves the highest speedup.

run 4th order Runge-Kutta (RK4) as a baseline, which is the standard way coupled ODE such as heat diffusion on graphs is solved.

We observe that GCN and GCN Res yields the highest speedup (more than 10 times faster than baselines) with a slightly higher loss for 2D lattice. Also note, the baseline (i.e. simple GD with Adam) is over 10 times faster than RK4. The node sizes in the Fig. 5 indicate the number of latent dimensions. Single layer models are less computationally expensive and generally yield more speedup. All reparametrization models outperform the baseline model in terms of energy and running time. GCN Res and NoGraph Res achieve the best performances, outperforming each other by a small margin depending on the graph topology. We do not show RK4 in the figure, because it is more than ten times slower than the GD baseline.

Figure 4b shows the loss curves for the different models. It shows the advantages of parametrization, and at the same time, it reveals the bottleneck of the Shampoo optimizer that cannot converge in a comparable time with other models. Note that Shampoo and K-FAC are designed for deep networks. Similarly, the shallow architecture of the baseline does not make the K-FAC preconditioner applicable to this problem. All three reparametrization models (NoGraph, NoGraph Res, GCN, GCN Res) outperform the baseline model in terms of energy and running time.

### 5.2 Synchronization

We also evaluate on the network synchronization problems. In all experiments, we use the same lattice size $50 \times 50$, with the same stopping criteria (10 patience steps and $10^{-10}$ loss error limit). For early stages, we use a GCN with the aggregation function derived from the Hessian. For the Kuramoto model, it is $\mathcal{H}_{ij}(0) = \partial^2 \mathcal{L} / \partial \boldsymbol{w}_i \partial \boldsymbol{w}_j |_{\boldsymbol{w} \to 0} = A_{ij} - \sum_k A_{ik} \delta_{ij} = -L_{ij}$ ($L = D - A$ being the Laplacian). We use neural reparametrization in the first 100 iterations and then switch to the original optimization (referred to as baseline) afterwards. We experimented with three different graph structures: square lattice, circle graph, and tree graph. The phases $\boldsymbol{w}$ are randomly initialized between 0 and $2\pi$ from a uniform distribution. We let the models run until the loss converges (10 patience steps for early stopping, $10^{-15}$ loss fluctuation limit).

**Results for the Kuramoto Model.** Figure 6 shows the results of Kuramoto model on a square lattice. Additional results on circle graph, and tree graph can be found in Appendix 7. Figure 6 (a) shows that our method with one-layer GCN (GCN-1) and GCN with residual connection (RGCN-1) achieves significant speedup. In particular, we found $3.6 \pm .5$ speed improvement for the lattice, $6.1 \pm .1$ for the circle graph and $2.7 \pm .3$ for tree graphs. We also experimented with two layer (GCN/RGCN-2) and three layer (GCN/RGCN-3) GCNs. As expected, the overhead of deeper GCN models slows

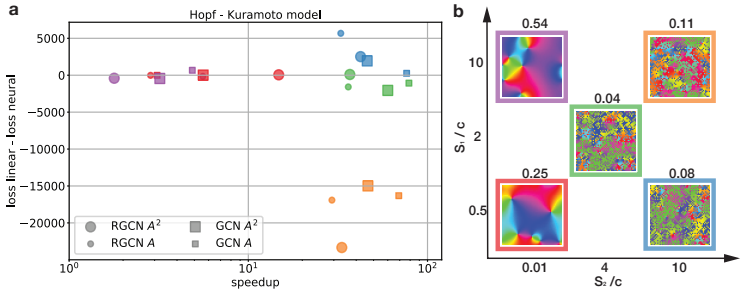


**Figure 7:** Hopf-Kuramoto model on a square lattice ($50 \times 50$). a) Speedup in the final loss value difference function. Points color correspond to the regions of the phase diagram (b), also, the number above each phase pattern are the global order parameters. c) Coupled oscillator system.

down optimization and offsets the speedup gains. Figure 6 (b) shows the evolution of $\boldsymbol{w}_i$ on a square lattice. Although different GNNs reach the same loss value, the final solutions are quite different. The baseline model (without GNN) arrives at the final solution smoothly, while GNN models form dense clusters at the initial steps and reach an organized state before 100 steps. To quantify the level of synchronization, we measure a quantity $\rho$ known as the "global order parameter" Sarkar and Gupte (2021): $\rho = \frac{1}{N}\left|\sum_j e^{i\boldsymbol{w}_j}\right|$. Figure 6 (c) shows the convergence of $\rho$ over time. We can see that one-layer GCN and RGCN give the most acceleration, driving the system to synchronization.

**Results for the Hopf-Kuramoto Model.** We report the comparison on synchronizing more complex Hopf-Kuramoto dynamics. According to the Lauter et al. (2015) paper, we identify two different main patterns on the phase diagram Fig. 7 (b): ordered (small $s_2/c$, smooth patterns) and disordered (large $s_2/c$, noisy) phases ($c = 1$). Figuer 7a shows the loss at convergence versus the speedup. We compare different GCN models and observe that GCN with $A^2$ as the propagation rule achieves the highest speedup. This is not surprising, as from equation 7 the Hessian for HK contains $O(A^2)$ terms. Also, we can see that we have different speedups in each region, especially in the disordered phases. Furthermore, we observed that the Linear and GCN models converge into different minima in a few cases. However, the patterns of $\boldsymbol{w}$ remain the same. Interestingly, in the disordered phase, we observe the highest speedup (Fig. 4)

### Conclusion

We propose a novel neural reparametrization scheme using GNN to accelerate a large class of graph optimization problems. The effect of the neural reparametrization is a preconditioning similar to quasi-Newton methods as it uses the Hessian. We demonstrate our method on optimizing heat diffusion and network synchronization problems and observe between over ten fold faster convergence to loss minima, in some cases. We believe such neural reparametrization using the Hessian may also benefit other optimization problems.

## References


Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019. 1

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020. 1

Mathieu Carriere, Frédéric Chazal, Marc Glisse, Yuichi Ike, Hariprasad Kannan, and Yuhei Umeda. Optimizing persistent homology based functions. In *International Conference on Machine Learning*, pages 1294–1303. PMLR, 2021. 1, 18, 19

Tolga Birdal, Aaron Lou, Leonidas J Guibas, and Umut Simsekli. Intrinsic dimension, persistent homology and generalization in neural networks. *Advances in Neural Information Processing Systems*, 34, 2021. 1, 18

Navid Zobeiry and Keith D Humfeld. A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications. *Engineering Applications of Artificial Intelligence*, 101:104232, 2021. 1

Haiyang He and Jay Pathak. An unsupervised learning approach to solving heat equations on chip based on auto encoder and image gradient. *arXiv preprint arXiv:2007.09684*, 2020. 1

Patrick Schnell, Philipp Holl, and Nils Thuerey. Half-inverse gradients for physical deep learning. In *International Conference on Learning Representations*, 2021. 1, 4, 14

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018. 1

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015. 2

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pages 1842–1850. PMLR, 2018. 2, 8

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020. 2

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 2

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018. 2

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019. 2

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. 2

Victor Bapst, Thomas Keck, A Grabska-Barwińska, Craig Donner, Ekin Dogus Cubuk, Samuel S Schoenholz, Annette Obika, Alexander WR Nelson, Trevor Back, Demis Hassabis, et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020. 2

Xiang Fu, Tian Xie, Nathan J Rebello, Bradley D Olsen, and Tommi Jaakkola. Simulate time-integrated coarse-grained molecular dynamics with geometric machine learning. *arXiv preprint arXiv:2204.10348*, 2022. 2

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2

Owe Axelsson. *Iterative solution methods*. Cambridge university press, 1996. 2

Yousef Saad and Henk A Van Der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):1–33, 2000. 2

E Groeneveld. A reparameterization to improve numerical optimization in multivariate reml (co) variance component estimation. *Genetics Selection Evolution*, 26(6):537–545, 1994. 2

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018. 2

Salma Tarmoun, Guilherme Franca, Benjamin D Haeffele, and Rene Vidal. Understanding the dynamics of gradient flow in overparameterized linear models. In *International Conference on Machine Learning*, pages 10153–10161. PMLR, 2021. 2

Ivan Sosnovik and Ivan Oseledets. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 34(4):215–223, 2019. 2

Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019. 2

Csaba Both, Nima Dehmamy, Rose Yu, and Albert-László Barabási. Accelerating network layouts using graph neural networks. *Nature communications*, 14(1):1560, 2023. 2

George E Forsythe and Wolfgang R Wasow. Finite-difference methods for partial differential equations. *Applied Mathematics Series*, 1960. 2

Olek C Zienkiewicz, Robert Leroy Taylor, and Jian Z Zhu. *The finite element method: its basis and fundamentals*. Elsevier, 2005. 2

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. 3

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pages 2415–2423. PMLR, 2019. 3

Leah Bar and Nir Sochen. Unsupervised deep learning algorithm for pde-based forward and inverse problems. *arXiv preprint arXiv:1904.05417*, 2019. 3

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. 3

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020a. 3

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020b. 3

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017. 3

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018. 3

Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International conference on machine learning*, pages 2430–2439. PMLR, 2017. 3

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019. 3

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. 3

Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52 (1):153–197, 1990. 3

Frank P Incropera, David P DeWitt, Theodore L Bergman, Adrienne S Lavine, et al. *Fundamentals of heat and mass transfer*, volume 6. Wiley New York, 1996. 3

Yifan Du and Tamer A Zaki. Evolutional deep neural network. *Physical Review E*, 104(4):045303, 2021. 3

Jialin Zhao, Yuxiao Dong, Ming Ding, Evgeny Kharlamov, and Jie Tang. Adaptive diffusion in graph neural networks. *Advances in Neural Information Processing Systems*, 34:23321–23333, 2021. 3

Eric Vincent, Jacques Hammann, Miguel Ocio, Jean-Philippe Bouchaud, and Leticia F Cugliandolo. Slow dynamics and aging in spin glasses. In *Complex Behaviour of Glassy Systems*, pages 184–219. Springer, 1997. 4

Luis EC Rocha, Fredrik Liljeros, and Petter Holme. Simulated epidemics in an empirical spatiotemporal network of 50,185 sexual contacts. *PLoS computational biology*, 7(3):e1001109, 2011. 4

Arkady Pikovsky, Jurgen Kurths, Michael Rosenblum, and Jürgen Kurths. *Synchronization: a universal concept in nonlinear sciences*. Number 12. Cambridge university press, 2003. 4, 5, 14

Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International symposium on mathematical problems in theoretical physics*, pages 420–422. Springer, 1975. 4, 14

Yoshiki Kuramoto. Chemical turbulence. In *Chemical Oscillations, Waves, and Turbulence*, pages 111–140. Springer, 1984. 4, 14

Roland Lauter, Christian Brendel, Steven JM Habraken, and Florian Marquardt. Pattern phase diagram for two-dimensional arrays of coupled limit-cycle oscillators. *Physical Review E*, 92(1): 012902, 2015. 5, 10, 15

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 6, 21

Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. *Advances in Neural Information Processing Systems*, 2019. 7

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

Mathew Penrose. *Random geometric graphs*, volume 5. OUP Oxford, 2003. 7

Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011. 8

Mrinal Sarkar and Neelima Gupte. Phase synchronization in the two-dimensional kuramoto model: Vortices and duality. *Physical Review E*, 103(3):032204, 2021. 10

John Michael Kosterlitz and David James Thouless. Ordering, metastability and phase transitions in two-dimensional systems. *Journal of Physics C: Solid State Physics*, 6(7):1181, 1973. 15

Nina Otter, Mason A Porter, Ulrike Tillmann, Peter Grindod, and Heather A Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(17), 2017. 18

Rickard Brüel Gabrielsson, Bradley J Nelson, Anjan Dwaraknath, and Primoz Skraba. A topology layer for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1553–1563. PMLR, 2020. 18, 19

Herbert Edelsbrunner, John Harer, et al. Persistent homology-a survey. *Contemporary mathematics*, 453:257–282, 2008. 18

Christoph Hofer, Roland Kwitt, Marc Niethammer, and Mandar Dixit. Connectivity-optimized representation learning via persistent homology. In *International Conference on Machine Learning*, pages 2751–2760. PMLR, 2019. 18

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 21

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arxiv 2018. *arXiv preprint arXiv:1806.01261*, 2018. 21

# 6 Extended Derivations

## 6.1 Hessian in early stage

We assume we initialize $\boldsymbol{w} \sim \mathcal{N}(0, 1/n)$ as a normal distribution. We want to estimate the expected value of the Hessian for the loss functions in equation 1 at initialization. For brevity, let $\Delta_{ij} = \boldsymbol{w}_i - \boldsymbol{w}_j$ and $f_{ij} \equiv f(\|\Delta_{ij}\|^2)$. The Hessian is given by

$$\begin{aligned}\mathcal{H}_{ij} =& \partial_i \partial_j \mathcal{L} = \partial_i \partial_j g + \partial_i \sum_k A_{jk} \Delta_{jk} f'_{jk} \\ =& \partial_i \partial_j g + A_{ji} f'_{ij} - \delta_{ij} \sum_k A_{jk} f'_{ik} + A_{ji} \|\Delta_{ji}\|^2 f''_{ji} - \delta_{ij} \sum_k A_{jk} \|\Delta_{jk}\|^2 f''_{jk} \end{aligned} \tag{11}$$

With this initialization, $\Delta_{ij} \sim \mathcal{N}(0, 2/n)$ will also be normally distributed. We can use a Taylor expansion to estimate the expected Hessian. Assuming $n \gg 1$, we want to find the leading order terms in $\mathbb{E}[\mathcal{H}]$. We have ($\text{even}(p) = 1$ if $p$ is even, and 0 when it's odd)

$$\begin{aligned}\mathbb{E}\left[\prod_{a=1}^{p} \Delta_{i_a j_a}\right] =& \delta_{i_1 \cdots i_p} \delta_{j_1 \cdots j_p} \frac{(p-1)!!}{(n/2)^{p/2}} \text{even}(p) \\ \mathbb{E}[\mathcal{H}] \approx& \sum_{p=0}^{\infty} \frac{\mathbb{E}\left[\prod_{a=1}^{p} \Delta_{i_a j_a}\right]}{p!} \left.\frac{\partial^p \mathcal{H}}{\prod_{a=1}^{p} \partial \Delta_{i_a j_a}}\right|_{\Delta \to 0} = \mathcal{H}(\Delta \to 0) + O\left(\frac{1}{n}\right) \\ \mathbb{E}[\mathcal{H}_{ij}] \approx& \partial_i \partial_j g(0) - f'(0) L_{ij} + O\left(\frac{1}{n}\right) \end{aligned} \tag{12}$$

where we used the degree matrix $D_{ij} = \delta_{ij} \sum_k A_{jk}$, the Laplacian $L = D - A$, and the assumption of undirectedness stated for equation 1 ($A_{ij} = A_{ji}$). Because the Hessian contains the graph Laplacian, which is $n \times n$, and so computing $\mathcal{H}^{-1}$ is $O(n^3)$.

## 6.2 Network Synchronization

The HK model's dynamics are follows:

$$\frac{d\boldsymbol{w}_i}{dt} = c \sum_j A_{ji} \left[\cos \Delta_{ij} - s_1 \sin \Delta_{ij}\right] + s_2 \sum_{k,j} A_{ij} A_{jk} \left[\sin\left(\Delta_{ji} + \Delta_{jk}\right) - \sin\left(\Delta_{ji} - \Delta_{jk}\right)\right] + A_{ij} A_{ik} \sin\left(\Delta_{ji} + \Delta_{ki}\right). \tag{13}$$

where $c$, $s_1$, $s_2$ are the governing parameters. Equation 13 is the GD equation for the following loss function (found by integrating Equation 13):

$$\mathcal{L}(\boldsymbol{w}) = \frac{c}{\varepsilon} \sum_{i,j} A_{ji} \left[\sin \Delta_{ij} + s_1 \cos \Delta_{ij}\right] + \frac{s_2}{2\varepsilon} \sum_{i,k,j} A_{ij} A_{jk} \left[\cos\left(\Delta_{ji} + \Delta_{jk}\right) + \cos\left(\Delta_{ji} - \Delta_{jk}\right)\right] \tag{14}$$

# 7 Experiment details and additional results

## 7.1 Network Synchronization

Network synchronization (Pikovsky et al., 2003) optimizes a network of coupled oscillators until they reach at the same frequency, known as synchronization. Kuramoto model Kuramoto (1975, 1984) are widely used for synchronization problems, which have profound impact on engineering, physics and machine learning Schnell et al. (2021).

As shown in Fig. 3, Kuramoto model describes the behavior of a large set of coupled oscillators. Each oscillator is defined by an angle $\theta_i = \omega_i t + \boldsymbol{w}_i$, where $\omega_i$ is the frequency and $\boldsymbol{w}_i$ is the phase. We consider the case where $\omega_i = 0$. The coupling strength is represented by a graph $A_{ij} \in \mathbb{R}$. Defining $\Delta_{ij} \equiv \boldsymbol{w}_i - \boldsymbol{w}_j$, the dynamics of the phases $\boldsymbol{w}_i(t)$ in the Kuramoto model follows the following equations:

$$\frac{d\boldsymbol{w}_i}{dt} = -\varepsilon \sum_{j=1}^{n} A_{ji} \sin \Delta_{ij}, \;\; \mathcal{L}(\boldsymbol{w}) = \sum_{i,j=1}^{n} A_{ji} \cos \Delta_{ij}. \tag{15}$$

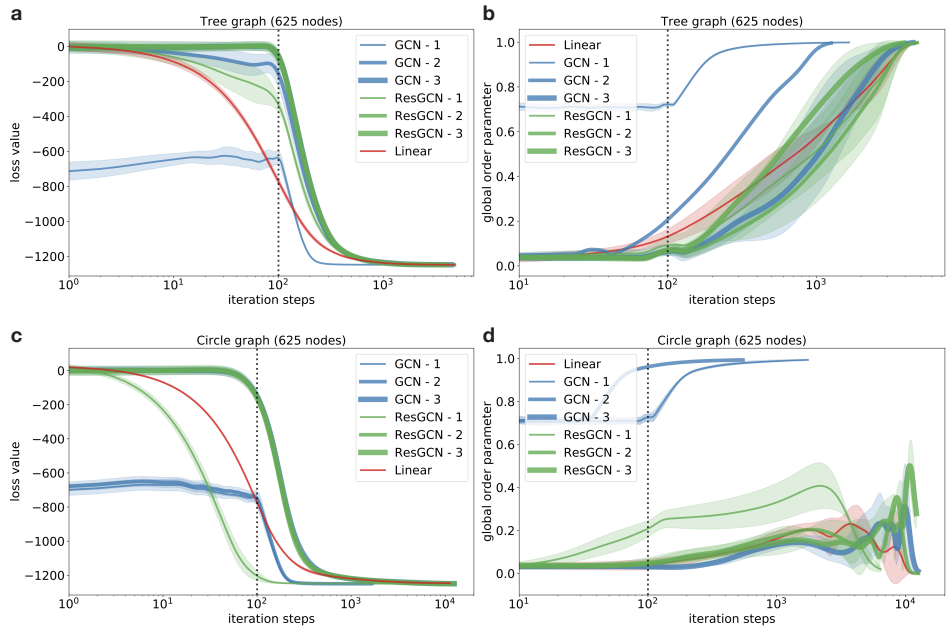


**Figure 8:** Kuramoto model on (a,b) tree and (c,d) circle graph. a,c) loss curves evolution, while b,d) global order parameter in each iteration steps

Our goal is to minimize the phase drift $d\boldsymbol{w}_i/dt$ such that the oscillators are synchronized. We further consider a more general version of the Kuramoto model: Hopf-Kuramoto (HK) model Lauter et al. (2015), which includes second-order interactions.

We experiment with both the Kuramoto model and Hopf-Kuramoto model. Existing numerical methods directly optimize the loss $\mathcal{L}(\boldsymbol{w})$ with gradient-based algorithms, which we refer to as *linear*. We apply our method to reparametrize the phase variables $\boldsymbol{w}$ and speed up convergence towards synchronization.

**Implementation.** For early stages, we use a GCN with the aggregation function derived from the Hessian which for the Kuramoto model simply becomes $\mathcal{H}_{ij}(0) = \partial^2\mathcal{L}/\partial\boldsymbol{w}_i\partial\boldsymbol{w}_j|_{\boldsymbol{w}\to 0} = A_{ij} - \sum_k A_{ik}\delta_{ij} = -L_{ij}$, where $L = D - A$ is the graph Laplacian of $A$. We found that NR in the early stages of the optimization gives more speed up. We implemented the hybrid optimization described earlier, where we reparemtrize the problem in the first 100 iterations and then switch to the original *linear* optimization for the rest of the optimization.

We experimented with three Kuramoto oscillator systems with different coupling structures: square lattice, circle graph, and tree graph. For each system, the phases are randomly initialized between 0 and $2\pi$ from uniform distribution. We let the different models run until the loss converges (10 patience steps for early stopping, $10^{-15}$ loss fluctuation limit).

### 7.2 Kuramoto Oscillator

**Relation to the XY model.** The loss equation 6 is also identical to the Hamiltonian (energy function) of a the classical XY model (Kosterlitz and Thouless (1973)), a set of 2D spins $s_i$ with interaction energy given by $\mathcal{L} = \sum_{i,j} A_{ij} s_i \cdot s_j = \sum_{i,j} A_{ji} \cos(\boldsymbol{w}_j - \boldsymbol{w}_i)$. In the XY model, we are also interested in the minima of the energy.

**Method.** In our model, we first initialize random phases between 0 and $2\pi$ from a uniform distribution for each oscillator in a $h$ dimensional space that results in $N \times h$ dimensional vector $N$ is the number of oscillators. Then we use this vector as input to the GCN model, which applies $D^{-1/2}\hat{A}D^{-1/2}$ , $\hat{A} = A + I$ propagation rule, with LeakyRelu activation. The final output dimension is $N \times 1$, where the elements are the phases of oscillators constrained between 0 and $2\pi$. In all experiments for $h$ hyperparameter we chose 10. Different $h$ values for different graph sizes may give

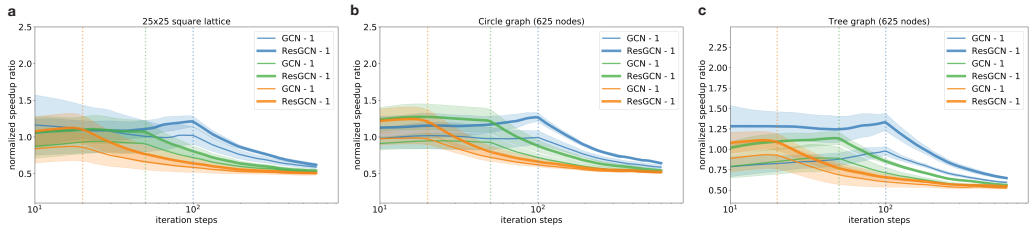


**Figure 9:** Kuramoto-model on different graph structures. We use the GCN model and switch to the Linear model after different iteration steps: orange - 20 steps, green - 50 steps, and blue - 100 steps. The plot shows that each GCN iteration step takes longer.

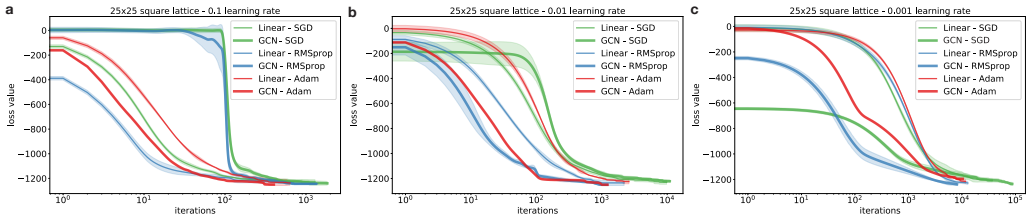


**Figure 10:** Kuramoto-model on $25 \times 25$ square lattice. We use the GCN and Linear model with different optimizers and learning rates. The Adam optimizer in all cases over-performs the other optimizers.

different results. Choosing large $h$ reduces the speedup significantly. We used Adam optimizer by $0.01$ learning rate.

### 7.3 MNIST image classification

Here, we introduce our reparametrization model for image classification on the MNIST image dataset. First, we test a simple linear model as a baseline and compare the performance to the GCN model. We use a cross-entropy loss function, Adam optimizer with a $0.001$ learning rate in the experiments, and Softmax nonlinear activation function in the GCN model. We train our models on $100$ batch size and $20$ epochs. In the GCN model, we build the $\boldsymbol{H} = (1 - \xi)\mathcal{H}/h_{max}$ matrix from the covariance matrix of images and use it as a propagation rule. In the early stages of the optimization, we use the GCN model until a plateau appears on the loss curve then train the model further by a linear model. We found that the optimal GCN to linear model transition is around $50$ iterations steps. Also, we discovered that wider GCN layers achieve better performance; thus, we chose $500$ for the initially hidden dimension. According to the previous experiments, the GCN model, persistent homology (7.4), and Kuramoto (7.2) model speedups the convergence in the early stages (Fig. 12)

### 7.4 Persistent Homology

**Overview.** Homology describes the general characteristics of data in a metric space, and is categorized by the order of its features. Zero order features correspond to connected components, first order features have shapes like "holes" and higher order features are described as "voids".

A practical way to compute homology of a topological space is through forming simplicial complexes from its points. This enables not only fast homology computation with linear algebra, but also approximating the topological space with its subsets.

In order to construct a simplicial complex, a filtration parameter is needed to specify the scope of connectivity. Intuitively, this defines the resolution of the homological features obtained. A feature is considered "persistent" if it exists across a wide range of filtration values. In order words, persistent homology seeks features that are scale-invariant, which serve as the best descriptors of the topological space.

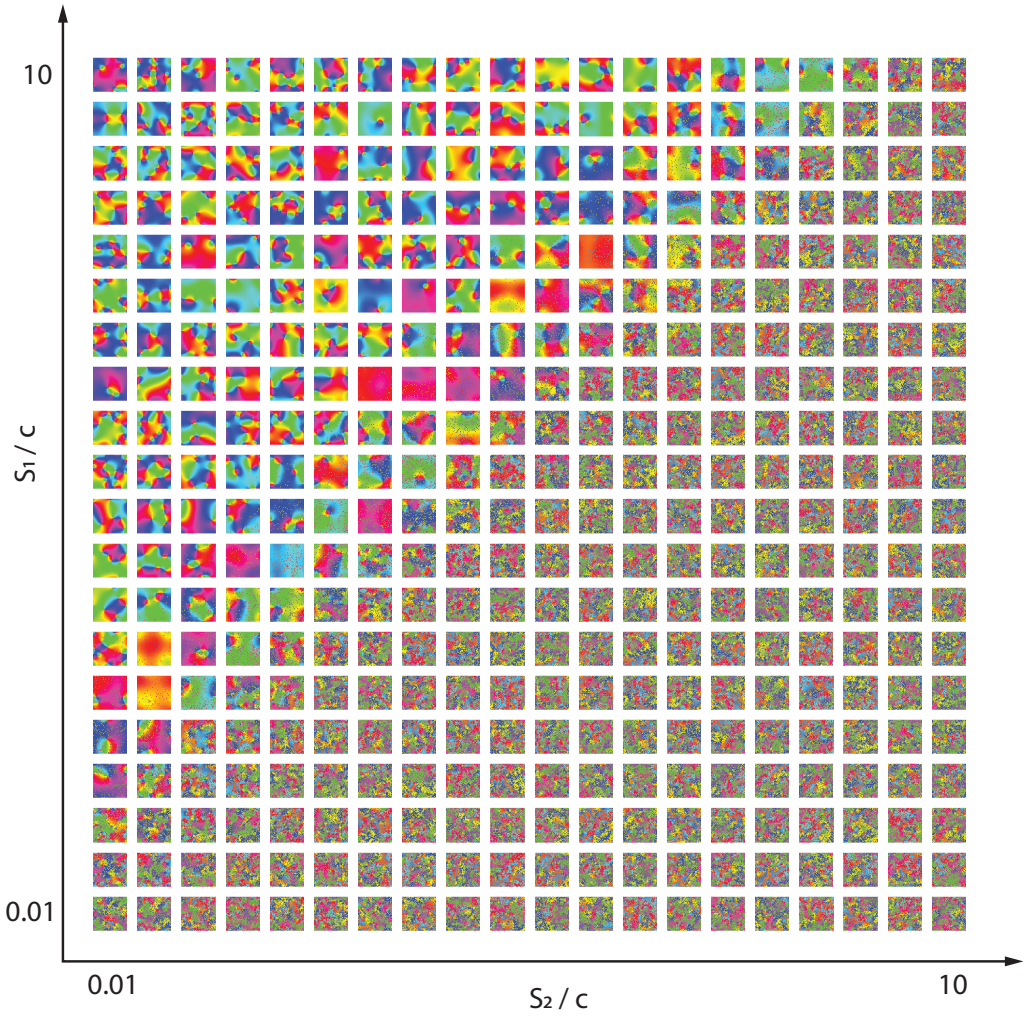


**Figure 11:** Hopf-Kuramoto model on a square lattice ($50 \times 50$) - phase pattern. We can distinguish two main patterns - organized states are on the left part while disorganized states are on the right part of the figure. In the experiments $c = 1$ for the simplicity.

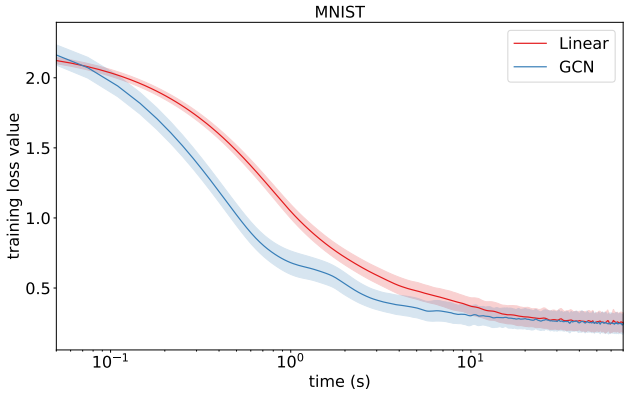


**Figure 12:** MNIST image classification. Red curve shows a linear model ($92.68\%$ accuracy). Blue curve is the reparametrized model, where at step $50$ we switch from GCN to Linear model ($92.71\%$ accuracy).

There are different ways to build simplicial complexes from given data points and filtration values. Czech complex, the most classic model, guarantees approximation of a topological space with a subset of points. However, it is computationally heavy and thus rarely used in practice. Instead, other models like the Vietoris-Rips complex, which approximates the Czech complex, are preferred for their efficiency (Otter et al., 2017). Vietoris-Rips complex is also used in the point cloud optimization experiment of ours and Carriere et al. (2021); Gabrielsson et al. (2020).

### 7.5 Persistent homology

Persistent homology Edelsbrunner et al. (2008) is an algebraic tool for measuring topological features of shapes and functions. Recently, it has found many applications in machine learning Birdal et al. (2021); Gabrielsson et al. (2020); Hofer et al. (2019). Persistent homology is computable with linear algebra and robust to perturbation of input data (Otter et al., 2017), see more details in Appendix 7.4. An example application of persistent homology is point cloud optimization (Carriere et al., 2021; Gabrielsson et al., 2020). As shown in Fig. 13 left, given a random point cloud $\boldsymbol{w}$ that lacks any observable characteristics, we aim to produce persistent homological features by optimizing the position of data.

$$\mathcal{L}(\boldsymbol{w}) = -\sum_{p \in D} \|p - \pi_{\Delta}(p)\|_{\infty}^{2} + \sum_{i=1}^{n} \|\boldsymbol{w}_i - r\|^2 \tag{16}$$

where $p \in D = \{(b_i, d_i)\}_{i \in I_k}$ denotes the homological features in the the persistence diagram $D$, consisting of all the pairs of birth $b_i$ and death $d_i$ filtration values of the set of k-dimensional homological features $I_k$. $\pi_{\Delta}$ is the projection onto the diagonal $\Delta$ and $\sum_i d(\boldsymbol{w}_i, S)$ constrains the points within the a square centered at the origin with length $2r$ (denote $r$ as range of the point cloud). Carriere et al. (2021) optimizes the point cloud positions directly with gradient-based optimization (we refer to it as "linear").

**Algorithm Implementation.** Instead of optimizing the coordinates of the point cloud directly, we reparameterize the point cloud as the output of the GCN model. To optimize the network weights, we chose identity matrix with dimension of the point cloud size as the fixed input.

To apply GCN, we need the adjacency matrix of the point cloud. Even though the point cloud does not have any edges, we can manually generate edges by constructing a simplicial complex from it. The filtration value is chosen around the midpoint between the maximum and minimum of the feature birth filtration value of the initial random point cloud, which works well in practice.

Before the optimization process begins, we first fit the network to re-produce the initial random point cloud distribution. This is done by minimizing MSE loss on the network output and the regression target.

Then, we begin to optimize the output with the same loss function in Carriere et al. (2021); Gabrielsson et al. (2020), which consists of topological and distance penalties. The GCN model can significantly accelerates convergence at the start of training, but this effect diminishes quickly. Therefore, we switch the GCN to the linear model once the its acceleration slows down. We used this hybrid approach in all of our experiments.

**Hyperparameter Tuning.** We conducted extensive hyperparameter search to fine tune the GCN model, in terms of varying hidden dimensions, learning rates and optimizers. We chose the setting of 200 point cloud with range 2.0 for all the tuning experiments.

Fig. 14 shows the model convergence with different hidden dimensions. We see that loss converges faster with one layer of GCN instead of two. Also, convergence is delayed when the dimension of GCN becomes too large. Overall, one layer GCN model with $h1, h2 = 8, 6$ generally excels in performance, and is used in all other experiments.

Fig. 15,16,17 shows the performance of the GCN model with different prefit learning rates, train learning rates and optimizers. From the results, a lower prefit learning rate of 0.01 combined with a training learning rate below 0.01 generally converges to lower loss and yields better speedup. For all the optimizers, default parameters from the Tensorflow model are used alongside varying learning rates and the same optimizer is used in both training and prefitting. Adam optimizer is much more effective than RSMProp and SGD on accelerating convergence. For SGD, prefitting with learning

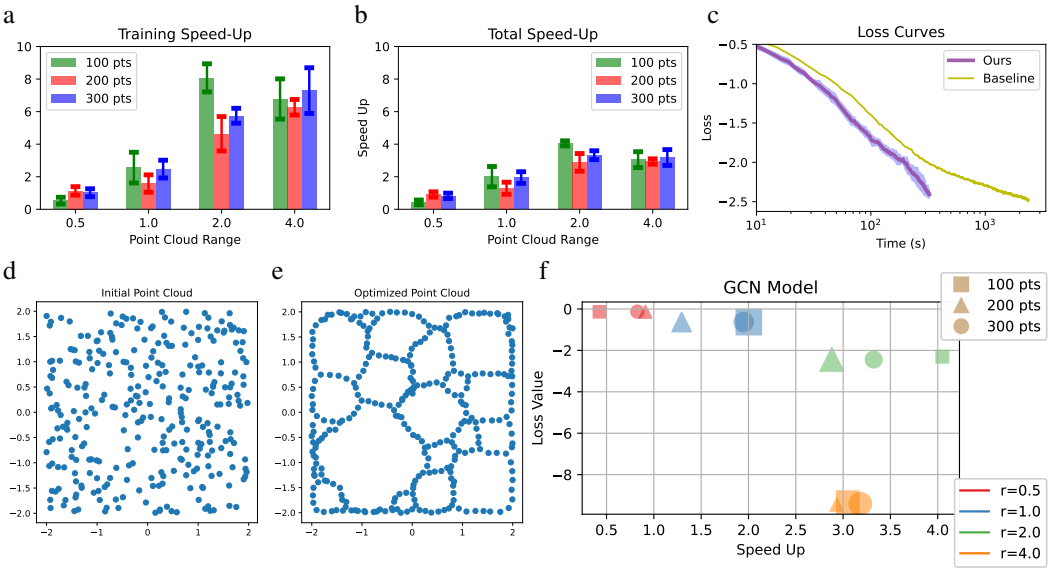


**Figure 13:** Speedup: a,b) Training and total time speedup; c) GCN speeds up convergence. d) Initial random point cloud and e) optimized point cloud. f) Loss vs speed-up of GCN model. The final loss depends only on the point cloud range, while the speedup is affected by both range and size.

rate 0.05 and 0.1 causes the loss to explode in a few iterations, thus the corresponding results are left as blank spaces.

**Detailed Runtime Comparison.** Fig. 18 and 19 shows how training, initial point cloud fitting and total time evolve over different point cloud sizes and ranges. Training time decreases significantly with increasing range, especially from 1.0 to 2.0. This effect becomes more obvious with density normalized runtime. On the other hand, prefitting time increases exponentially with both point cloud range and size. Overall, the total time matches the trend of training time, however the speed-up is halved compared to training due to the addition of prefitting time.

**Implementation.** We used the same Gudhi library for computing persistence diagram as Carriere et al. (2021); Gabrielsson et al. (2020). The run time of learning persistent homology is dominated by computing persistence diagram in every iteration, which has the time complexity of $O(n^3)$. Thus, the run time per iteration for GCN model and linear model are very similar, and we find that the GCN model can reduce convergence time by a factor of $\sim 4$ (Fig. 13,13). We ran the experiments for point cloud of 100,200,300 points, with ranges of 0.5,1.0,2.0,4.0. The hyperparameters of the GCN model are kept constant, including network dimensions. The result for each setting is averaged from 5 consecutive runs.

**Results for Persistent Homology.** Figure 13 right shows that the speedup of the GCN model is related to point cloud density. In this problem, the initial position of the point cloud determines the topology features. Therefore, we need to make sure the GCN models also yield the same positions as used in the linear model. Therefore, we first run a “training" step, where we use MSE to match the $\boldsymbol{w}(\theta)$ or GCN to the initial $\boldsymbol{w}$ used in the linear model. Training converges faster as the point cloud becomes more sparse, but the speedup gain saturates as point cloud density decreases. On the other hand, time required for initial point cloud fitting increases significantly with the range of point cloud. Consequently, the overall speedup peaks when the range of point cloud is around 4 times larger than what is used be in Carriere et al. (2021); Gabrielsson et al. (2020), which spans over an area 16 times larger. Further increase in point cloud range causes the speedup to drop as the extra time of initial point cloud fitting outweighs the reduced training time. The loss curve plot in Fig. 13 shows the convergence of training loss of the GCN model and the baseline model in one of the settings when GCN is performing well. Fig. 13 shows the initial random point cloud and the output from the GCN model. In the Appendix 7.4, we included the results of GCN model hyperparameter search and a runtime comparison of the GCN model under all experiment settings.

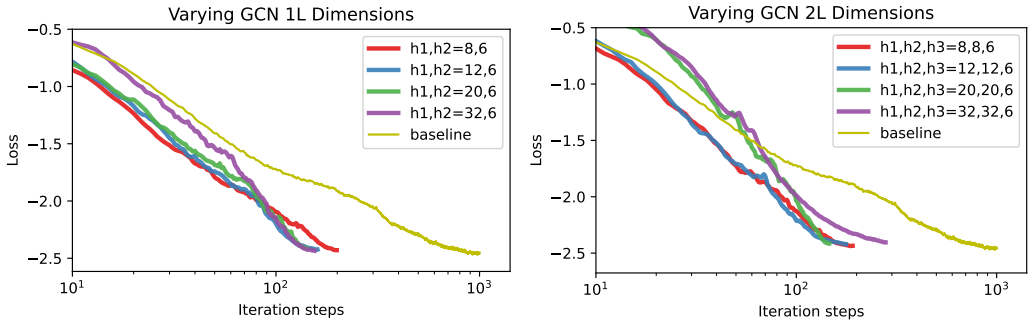


**Figure 14:** GCN Hyperparameter Comparison. We recommend using one layer GCN model with $h1, h2 = 8, 6$.

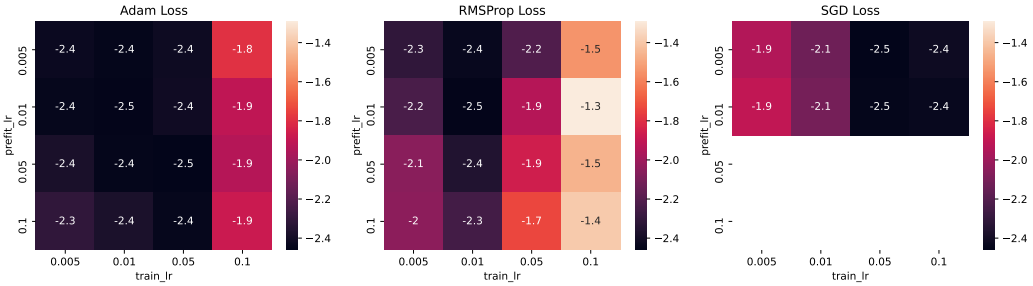


**Figure 15:** Converged loss of different learning rates and optimizers.

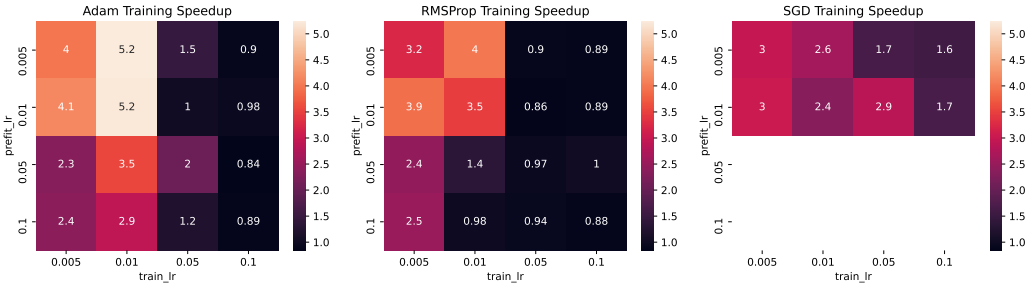


**Figure 16:** Training Speedup of different learning rates and optimizers.

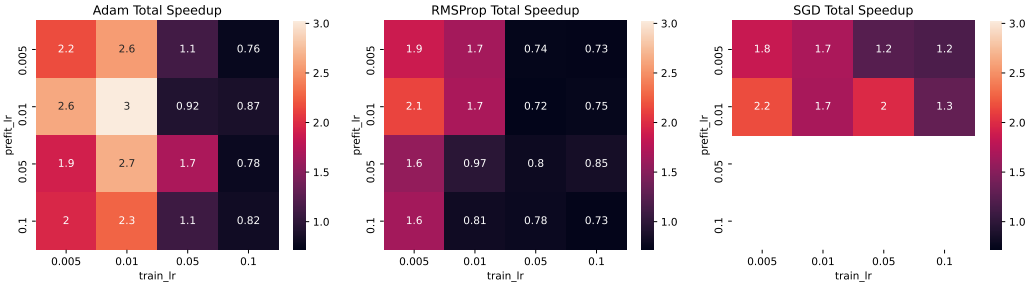


**Figure 17:** Total Speedup of different learning rates and optimizers.

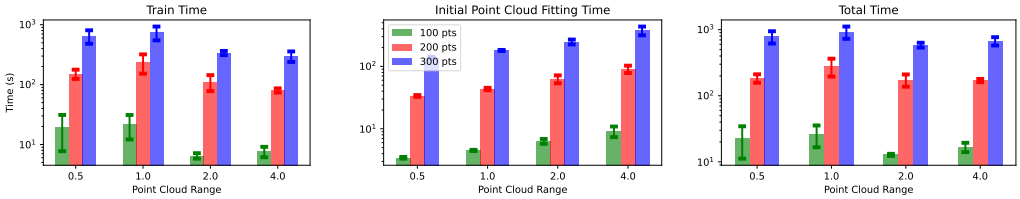


**Figure 18:** Training, prefitting and total time

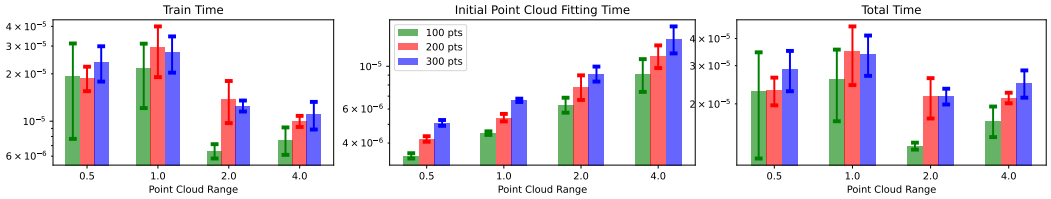


**Figure 19:** Density Normalized training, prefitting and total time. We normalize the runtime by $N^3$, where $N$ is the point cloud size. This is because persistence diagram computation has time complexity of $\mathcal{O}(N^3)$

.

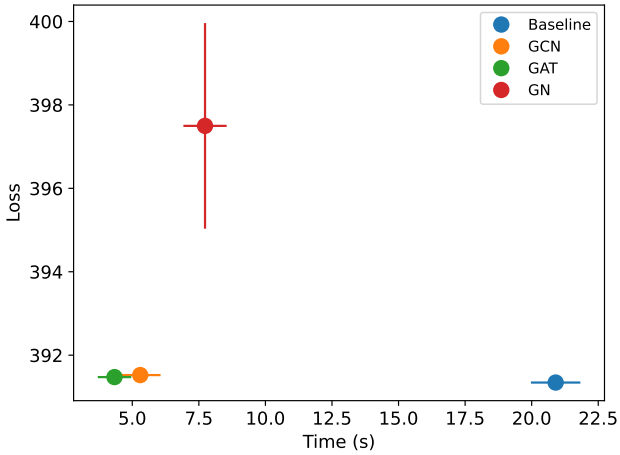


**Figure 20:** Heat diffusion - GCN/GAT/GN reparametrization. Conducted experiments with GCN Kipf and Welling (2016), GAT Veličković et al. (2017), and GN Battaglia et al. (2018) models on the heat diffusion problem. Specifically, we used a 75x75 triangular lattice configuration (2926 nodes and 8550 edges) with a hot heat bath at the top and a cold one at the bottom. Our findings indicate that all three methods outperformed the baseline GD method in wall clock time. While GN showed slightly less favorable energy and running time performance compared to GAT and GCN, the difference between GAT and GCN was not significant. These simulation results represent averages over five random seeds.