ACCELERATING OPTIMIZATION USING NEURAL REPARAMETRIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We tackle the problem of accelerating certain optimization problems related to steady states in ODE and energy minimization problems common in physics. We reparametrize the optimization variables as the output of a neural network. We then find the conditions under which this neural reparametrization could speed up convergence rates during gradient descent. We find that to get the maximum speed up the neural network needs to be a special graph convolutional network (GCN) with its aggregation function constructed from the gradients of the loss function. We show the utility of our method on two different optimization problems on graphs and point-clouds.

Central Questions

1. Concentration of states in infinite-width networks.
2. Relation between width and structure of input data

Todo

1. Topology optimization
2. NeuralSync
3. Generative modeling problem ??

1 INTRODUCTION

2 THEORETICAL ANALYSIS

We begin by considering a class of unsupervised optimization problems. The problem is defined as finding a local minimum of a given smooth, lower-bounded loss function $\mathcal{L}(\boldsymbol{w}) \in \mathbb{R}$, where $\boldsymbol{w} \in \mathbb{R}^n$ [ND: this is general enough to include $\mathbb{R}^{n\times m}$, but we need to explain that later.] is a high-dimensional vector of optimization parameters. Many systems in physics and chemistry are of this nature. For instance, $\mathcal{L}$ can be the free energy of a set of spins, as in the Ising model or Spin glasses Bapst et al. (2020) [ND: cite]. Note that these optimization problems do not involve an input dataset. [ND: Can we generalize to supervised problems?] We are interested in the dynamics of the loss, $d\mathcal{L}/dt$ during the optimization process. Using the chain rule we have

$$\frac{d\mathcal{L}(\boldsymbol{w})}{dt} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} \frac{d\boldsymbol{w}_i}{dt} \tag{1}$$

When we solve such an optimization problem numerically, we need to have bounds on the magnitude of $\|d\boldsymbol{w}/dt\|^2$ in order to avoid numerical instabilities. An immediate result from this is that the gradient is the direction of steepest descent in the parameter space. This means that setting

$$\frac{d\boldsymbol{w}_i}{dt} = -\varepsilon \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i}. \tag{2}$$

yields the largest $|d\mathcal{L}/dt|$ for a fixed $\|d\boldsymbol{w}/dt\|^2$. Therefore, we focus on using gradient descent (GD) to find a local minimum of $\mathcal{L}(\boldsymbol{w})$. Note that for continuous t GD equation 2 is an Ordinary Differential Equation (ODE). Alternatively, a first order ODE of the form $d\boldsymbol{w}/dt = F(\boldsymbol{w})$ is equivalent

to GD if there exists a function $\mathcal{L}(\boldsymbol{w})$ such that $F(\boldsymbol{w}) = -\varepsilon\partial\mathcal{L}(\boldsymbol{w})/\partial\boldsymbol{w}$. The task of GD is to find local minima of $\mathcal{L}(\boldsymbol{w})$, which are stationary solutions of the ODE, meaning $d\boldsymbol{w}/dt = 0$.

We are interested in the rate of convergence to a local minimum and how it can be accelerated. Calculating the rate of the loss drop, $d\mathcal{L}/dt$, when using GD to optimize we have

$$\frac{d\mathcal{L}(\boldsymbol{w})}{dt} = \sum_i \frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_i}\frac{d\boldsymbol{w}_i}{dt} = -\varepsilon\left\|\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}\right\|^2 = -\varepsilon\,\mathrm{Tr}[M], \qquad M_{ij} \equiv \frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_i}\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_j}. \tag{3}$$

The symmetric matrix M, which we will call "squared gradients", encodes the gradients of $\mathcal{L}(\boldsymbol{w})$ with respect to its inputs. Note that $M(\boldsymbol{w}(t))$ is time-dependent and can change during optimization. Also, note that equation 3 is *not* the convergence rate. For that we need to estimate how long it would take the system to get close to a local minimum. The dynamics of $\boldsymbol{w}$ is not the same in all directions of the parameter space, with some directions moving faster relative to others, as we discuss next.

2.1 Fast and slow modes

We can show that the eigenvalues of M determine the rate of change of their corresponding eigenvectors:

Proposition 1 (Fast and slow modes)**.** *M is Hermitian with a spectral expansion $M = \sum_i m_i\psi_i\psi_i^T$. We have*

$$\textit{if: } m_i \geq m_j \Rightarrow \left\|\psi_i^T\frac{d\boldsymbol{w}}{dt}\right\|^2 \geq \left\|\psi_j^T\frac{d\boldsymbol{w}}{dt}\right\| \tag{4}$$

Proof. M being positive semi-definite is because for any vector $\boldsymbol{v} \in \mathbb{R}^n$ we have $\boldsymbol{v}^T M\boldsymbol{v} = \|\boldsymbol{v}^T D\|^2 \geq 0$ where $D_i \equiv \partial\mathcal{L}/\partial\boldsymbol{w}_i$. Using equation 2, we have

$$\left\|\psi_i^T\frac{d\boldsymbol{w}}{dt}\right\|^2 = \varepsilon^2\psi_i^T\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}^T\psi_i = \varepsilon^2\psi_i^T M\psi_i = \varepsilon^2 m_i \tag{5}$$

which proves equation 4. □

Thus, the eigenvalues of M determine the rate of evolution of the overlaps along each of its eigenvectors. In particular, if some eigenvalue $m_{slow} \ll \mathrm{mean}_i[m_i]$ is much smaller than the mean, the evolution of $\boldsymbol{w}$ along ψ_{slow} will be much slower than other directions. Therefore, we will refer to all ψ_{slow} as the *slow modes* of the optimization problem. Conversely, the ψ_{fast} where $m_{fast} \gg \mathrm{mean}_i[m_i]$ will be referred to as the *fast modes*.

Note that the dynamics of $\boldsymbol{w}$ along different ψ_i are orthogonal to each other, since

$$\left[\psi_i^T\frac{d\boldsymbol{w}}{dt}\right]\left[\psi_j^T\frac{d\boldsymbol{w}}{dt}\right]^T = \varepsilon^2\psi_i^T M\psi_j = \varepsilon^2 m_i\delta_{ij} \tag{6}$$

with δ_{ij} being the Kronecker delta. When numerically running GD, the maximum change in $\boldsymbol{w}$ is bounded to ensure numerical stability. because of the orthogonality equation 6, we can enforce a numerical bound η via

$$\max_i \|\psi_i^T d\boldsymbol{w}/dt\| < \eta \qquad \Rightarrow \qquad \varepsilon < \frac{\eta}{\sqrt{m_{max}}} \tag{7}$$

where $m_{max} = \max_i m_i$ is the largest eigenvalue of M. Therefore, the choice of the learning rate ε is constrained by the largest eigenvalues (fastest modes). This further constrains how fast the slow modes can evolve because for $m_{min} = \min_i[m_i]$ it leads to [ND: find ref]

$$\left\|\psi_{min}^T\frac{d\boldsymbol{w}}{dt}\right\|^2 = \varepsilon^2 m_{min} < \eta^2\frac{m_{min}}{m_{max}}. \tag{8}$$

Again, note that because M is time-dependent, the constraint equation 8 defines a time-dependent learning rate $\varepsilon(t)$ for a fixed η. Assuming that the distance from the initial $\boldsymbol{w}(t=0)$ to the local minimum $\boldsymbol{w}^*$ is on average similar in all ψ_i directions, meaning $\|\psi_i^T(\boldsymbol{w}(0) - \boldsymbol{w}^*)\|$ is similar for all i. If so, the constraint on the rate of convergence stems from the ratio m_{min}/m_{max}. Thus, in order to speed up the *convergence* we need to speed up how fast the slowest modes evolve. We will show below that a reparametrization of $\boldsymbol{w}$ can make this possible. [ND: find similarity with preconditioning]

2.2 NEURAL PARAMETRIZATION OF OPTIMIZATION PROBLEMS

A number of recent works have used neural networks to reparametrize optimization problems (Hoyer et al., 2019; Sosnovik & Oseledets, 2019) [ND: more?]. The idea is that the optimization variable $\boldsymbol{w}$ can be defined as the output of a neural network $\boldsymbol{w}(\theta) \equiv f(\theta)$ with trainable parameters θ. Then, instead of optimizing over $\boldsymbol{w}$ directly, we optimize over the neural network parameters θ. We will take this idea further and ask whether there exist neural network architectures that we could use to accelerate the optimization process. It follows that after the "neural reparametrization" the loss drop can be expressed in terms f the Neural Tangent Kernel (NTK) (Jacot et al., 2018):

Theorem 1 (Rate of Loss Drop)**.** *In the unsupervised optimization using GD with $\mathcal{L}(\boldsymbol{w})$ as above, using neural network $\boldsymbol{w}(\theta) = f(\theta)$ to reparametrize the optimization variable $\boldsymbol{w}$ yields*

$$\frac{d\mathcal{L}(\boldsymbol{w}(\theta))}{dt} = -\varepsilon \operatorname{Tr}\left[MK^T\right], \qquad K_{ij} \equiv \sum_a \frac{\partial \boldsymbol{w}_i}{\partial \theta_a}\frac{\partial \boldsymbol{w}_j}{\partial \theta_a} \tag{9}$$

where K is the neural tangent kernel (NTK) of $\boldsymbol{w}(\theta)$, and squared gradients M (equation 3).

Proof. To measure acceleration, we rewrite the rate of convergence for this setup. First, note that now the GD is done for θ with the update rule for component θ_a being

$$\frac{d\theta_a}{dt} = -\varepsilon \frac{\partial \mathcal{L}}{\partial \theta_a} = -\varepsilon \sum_i \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i}\frac{\partial \boldsymbol{w}_i}{\partial \theta_a} \tag{10}$$

The convergence rate is then given by

$$\begin{aligned}\frac{d\mathcal{L}}{dt} &= \sum_{i,a} \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i}\frac{\partial \boldsymbol{w}_i}{\partial \theta_a}\frac{d\theta_a}{dt} = -\varepsilon \sum_{i,j} \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i}\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j}\sum_a \frac{\partial \boldsymbol{w}_i}{\partial \theta_a}\frac{\partial \boldsymbol{w}_j}{\partial \theta_a} \\ &= -\varepsilon \sum_{i,j} M_{ij}K_{ij} = -\varepsilon \operatorname{Tr}\left[MK^T\right]\end{aligned} \tag{11}$$

□

Note that, like M, the NTK K is positive-semi-definite because $K = BB^T$ with $B_{ia} \equiv \partial \boldsymbol{w}_i / \partial \theta_a$, and so for any vector $\boldsymbol{v} \in \mathbb{R}^n$ we have $\boldsymbol{v}^T K \boldsymbol{v} = \|\boldsymbol{v}^T B\|^2 \geq 0$. To see that K is indeed the NTK, note that in supervised problems on a dataset $\mathcal{X} = \{x_1, \dots\}$ (x_n includes both inputs and labels) the neural network output is $\boldsymbol{w}(\theta; x_n)$ a function of x_n. Therefore, the supervised NTK has two more indices $K_{ij,nm} = \sum_a \partial \boldsymbol{w}_i(\theta; x_n)/\partial \theta_a \partial \boldsymbol{w}_j(\theta; x_m)/\partial \theta_a$. Our optimization problem is unsupervised and so there are no sample indices n, m. $M(\boldsymbol{w})$ is purely a function of the optimization loss function and the current value of $\boldsymbol{w}$, and does not directly depend on the architecture of the neural network. $K(\theta)$, on the other hand, directly depends on the neural network structure and parameters.

Similar to equation 4, we can compute the rate of change of $\boldsymbol{w}(\theta)$ along the previous fast and slow modes, which are eigenvectors ψ_i of M. Using equation 11 and equation 10 we have

$$\psi_i^T \frac{d\boldsymbol{w}}{dt} = \psi_i^T \sum_a \frac{\partial \boldsymbol{w}}{\partial \theta_a}\frac{d\theta_a}{dt} = -\varepsilon \psi_i^T \sum_a \frac{\partial \boldsymbol{w}}{\partial \theta_a}\sum_j \frac{\partial \boldsymbol{w}_j}{\partial \theta_a}\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j} = -\varepsilon \psi_i^T K \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \tag{12}$$

where $\partial \mathcal{L}/\partial \boldsymbol{w} = \nabla_{\boldsymbol{w}} \mathcal{L}$ is the gradient vector. Multiplying equation 12 by its transpose we get

$$\left\| \psi_i^T \frac{d\boldsymbol{w}}{dt} \right\|^2 = \varepsilon^2 \psi_i^T K M K^T \psi_i = \varepsilon^2 \sum_j m_j \left(\psi_i^T K \psi_j\right)^2. \tag{13}$$

Now we can see how the neural reparametrization may speed up the convergence. First, note that the neural reparametrization changes the rate of evolution of modes ψ_i in equation 8. We can quantify this change for the special case where $[K, M] = 0$, as follows.

Proposition 2. *Given the constraint $\max_i \|\psi_i^T d\boldsymbol{w}/dt\| < \eta$, if $[K, M] = 0$ we have*

$$\left\| \psi_i^T \frac{d\boldsymbol{w}}{dt} \right\|^2 = \varepsilon^2 k_i^2 m_i < \eta^2 \frac{m_i k_i^2}{m_{max}}. \tag{14}$$

where k_i is the eigenvalue of K for the shared eigenvector ψ_i between M and K.

Proof. This follows from plugging the spectral expansion $K = \sum_i k_i \psi_i \psi_i^T$ into equation 13 and using $\varepsilon < \eta/\sqrt{m_{max}}$ (equation 7). □

Thus, one way to speed up the convergence, is to have $[K, M] =$ and have K increase the slowest rates $\|\psi_{slow} d\boldsymbol{w}/dt\|$, while keeping the fastest rate mostly unchanged. To make all modes ψ_i reach the ideal maximum rate $\max_i \|\psi_i^T d\boldsymbol{w}/dt\| < \eta$, we should have $k_i^2 \approx m_{max}/m_i$ (for the nonzero m_i). This means that, when $[K, M] = 0$, maximum speedup can be achieved if

$$K \approx (M/m_{max})^{-1/2} \tag{15}$$

with $\max_i k_i = 1$. How can we achieve this in practice? What architecture for the neural network $\boldsymbol{w}(\theta) = f(\theta)$ will result in an NTK given by equation 15? We will discuss this next.

2.3 Practical Implementation

To find a neural network model that satisfies equation 15 we need to consider the fact that M changes during the optimization process. To avoid overhead from constructing $\boldsymbol{w}(\theta)$ as a function of M, the architecture has to be as simple as possible and involve steps which do not increase the time complexity of the optimization process. We will now briefly discuss when such neural reparametrization is feasible and how to get equation 15 in practice.

Per-step time complexity Since $\boldsymbol{w} \in \mathbb{R}^n$, the complexity of computing $\mathcal{L}(\boldsymbol{w})$ is $O(np)$ where p depends on the structure of $\mathcal{L}$. For example if $\mathcal{L} \sim \|\boldsymbol{w}\|^2$, $p \sim O(1)$, while if $\mathcal{L} \sim \boldsymbol{w}^T L \boldsymbol{w}$, we may have $p \sim O(n)$ when L is a dense matrix. M is an $n \times n$ matrix and computing $M^{-1/2}$ is $O(n^3)$ (using eigen-decomposition), which is very expensive when $n \gg 1$. We may instead try to approximate $M^{-1/2}$ and m_{max} iteratively with complexity $O(qn^2)$ where $q \ll n$ is the number of iterations. Hence we note that using equation 15 to speedup the optimization may generally be useful if the the per step complexity of GD on $\mathcal{L}$ has a time complexity of least $O(qn^2)$.

Simple implementation The simplest architecture we can have is a linear model $\boldsymbol{w}_i = \sum_a \kappa_{ia}\theta_a$, which yields $K = \kappa\kappa^T \approx (M/m_{max})^{-1/2}$. Since we the ψ_i with the smallest m_i have the largest eigenvalues in $(M/m_{max})^{-1/2}$, it may seem that the $n \times q$ matrix κ can be the q principal components (PC) of $(M/m_{max})^{-1/2}$. However, this will not for our problem because according to equation 13 any ψ_i that has eigenvalue zero in K will not evolve at all. Therefore, we need K to be a full rank matrix, meaning that κ needs to be at least $n \times n$. Since we need the least computationally expensive K, we choose $\theta \in \mathbb{R}^n$, and let κ be a symmetric $n \times n$ matrix. This way, $\kappa = \sqrt{K} = (M/m_{max})^{-1/4}$.

Relation to GCN Evaluating this κ may in practice be a challenge, which we will address next. But before that, let us interpret the constraints we have found so far on the architecture of the neural network. We have $\boldsymbol{w} = \kappa\theta$ where $\kappa = (M/m_{max})^{-1/4}$ is a matrix fixed by the loss function $\mathcal{L}$. Since both $\theta \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$, technically, our architecture is the same as a Graph Convolutional Network (GCN) (Kipf & Welling, 2016) with the aggregation rule $(M/m_{max})^{-1/4}$. Indeed, we can think of M as the weighted adjacency matrix for a graph with n vertices (nodes) and θ and $\boldsymbol{w}$ are node attributes. In fact, all of our derivations remain unchanged if $\boldsymbol{w} \in \mathbb{R}^{n\times d}$ and $\theta \in \mathbb{R}^{n\times m}$ (i.e. if there are d and m features per node in $\boldsymbol{w}$ and θ).

Thus, our claim is that we can speed up the optimization process by reparametrizing $\boldsymbol{w}$ using a GCN with linear activation, as $\boldsymbol{w} = \kappa\theta$, where θ are trainable and $\kappa = (M/m_{max})^{-1/4}$ is the aggregation function derived from the loss gradients. However, evaluating this κ during GD can be quite expensive if n is large. As we discussed above, each step of GD may be $O(n^2)$ before neural reparametrization, while evaluating $\kappa = (M/m_{max})^{-1/4}$ is typically $O(n^3)$. Even if we do not recalculate κ at every step of GD, this is prohibitively expensive. Ideally we want an approximation for $(M/m_{max})^{-1/4}$ which is $O(qn^2)$ with $q \sim O(1)$, which we discuss next.

Computationally efficient implementation Define $M_0 \equiv M/m_{max}$. To get an $O(qn^2)$ approximation for $\kappa = M_0^{1/4}$ we can take the first q terms in the binomial expansion

$$M_0^{-1/4} = [I + (M_0 - I)]^{-1/4} = \sum_{m=0}^{\infty} \frac{\prod_{i=0}^{m}(-1/4 - i)}{m!}(M_0 - I)^m$$
$$\approx \frac{-1}{4} \sum_{m=0}^{\infty} (I - M_0)^m \tag{16}$$

where since $1/4 \ll 1$, we used $\prod_{i=0}^{m}(-1/4 - i) \approx \frac{1}{4}(-1)^{m+1}m!$. Since M_0 is positive semi-definite and its largest eigenvalue is one, the sum in equation 16 can be truncated after $q \sim O(1)$ terms. [ND: should we change to $M/(m_{max}+\xi)$ for a small $\xi \ll 1$ to argue for truncating the sum? (since technically we need all eigenvalues smaller than 1)] The first q terms of equation 16 can be implemented as a q layer GCN with aggregation function $f(M) = I - M_0$ and residual connections. We choose q to be as small as 1 or 2, as larger q may actually slow down the optimization. [ND: move following to appendix?] Note that computing $f(M)^q\theta$ is $O(qn^2)$ because we do not need to first compute $f(M)^q$ (which is $O(qn^3)$) and instead use $\boldsymbol{v}_{i+1} f(M)\boldsymbol{v}_i$ ($O(n^2)$) q times with $\boldsymbol{v}_1 = \theta$. [ND: discuss architecture details.]

Lastly, to evaluate M_0 we need to estimate the leading eigenvalue m_{max}. We can use a similar expansion for approximating m_{max}. Given any vector $\boldsymbol{v} \in \mathbb{R}^n$ and using the spectral expansion we have

$$M^q \boldsymbol{v} = \sum_i m_i^q (\psi_i^T \boldsymbol{v})\psi_i \tag{17}$$

Since M is positive semi-definite, for $q > 1$ the leading eigenvector ψ_{max} quickly dominates the sum and we have $M^q\boldsymbol{v} \approx m_{max}^q(\psi_{max}\boldsymbol{v})\psi_{max}$. Additionally, using the generalized Perron-Frobenius theorem[ND: cite], the components of the leading eigenvector ψ_{max} should be mostly positive. Therefore, a good choice for $\boldsymbol{v}$, which may be close to the actual ψ_{max}, is $\boldsymbol{v} = \mathbf{1}/\sqrt{n}$, where $\mathbf{1} = (1, \ldots, 1)$. This yields

$$\mathbf{1}^T M^p \mathbf{1} \approx m_{max}^p (\psi_i^T \mathbf{1})^2, \qquad m_{max}^{p-1} \approx \frac{\mathbf{1}^T M^p \mathbf{1}}{\mathbf{1}^T M \mathbf{1}} \tag{18}$$

This is a crude approximation, but it serves our goal by being computationally efficient. Note that, while computing M^q is $O(qn^3)$, because $M\mathbf{1}$ is $O(n^2)$, computing $M^q\mathbf{1}$ is $O(qn^2)$. Therefore, we choose $p = 2$ and write

$$m_{max} \approx \frac{\mathbf{1}^T M^2 \mathbf{1}}{\mathbf{1}^T M \mathbf{1}} = \frac{\sum_{ij} M_{ij}^2}{\sum_{ij} M_{ij}} \tag{19}$$

It is worth noting that if we think of M as the weighted adjacency matrix of a graph with n vertices, the vector component $D_i = [M\mathbf{1}]_i = \sum_j M_{ij}$ is the weighted degree of node i. Hence equation 19 states $M/m_{max} \approx \left(\sum_i D_i/\|D\|^2\right) M$. This is reminiscent of the graph diffusion operator $D^{-1}M$ and $D^{-1/2}MD^{-1/2}$ which are used as the aggregation functions in GCN (here $D_{ij} = D_i\delta_{ij}$ is the diagonal degree matrix).

[ND: Note! All these derivations are for the non-zero eigenvalues of M. So in the ratio m_{min}/m_{max}, m_{min} is the smallest **nonzero** eigenvalue of M.]

[ND: Todo: Check if $[M, K] = 0$ yields maximum speedup.]

[ND: This actually works great for our GNN argument. As I showed for the convex case, $M \approx LL^T$, which means we get max speedup if $K \sim L^{-1}$ or its approximations, which is similar to $K \approx I - D^{-1/2}LD^{-1/2} = D^{-1/2}AD^{-1/2}$. We can show this explicitly for some graphs.]

Equation 11 decomposes the rate of convergence into a part M arising from the loss function and a part K capturing the effect of the neural architecture. To understand the interplay between these two matrices, we will first work out the toy example of convex optimization.

2.3.1 EXAMPLE OF CONVEX OPTIMIZATION

Assume we have the convex optimization problem of minimizing $\mathcal{L}(x) = \boldsymbol{w}^T L \boldsymbol{w}$, with $L \in \mathbb{R}^{d\times d}$ being a positive semi-definite matrix and $\boldsymbol{w} \in \mathbb{R}^d$. In this case we have

$$M_{ij} = \sum_k L_{ik}\boldsymbol{w}_k \sum_m L_{jm}\boldsymbol{w}_m = [L\boldsymbol{w}\boldsymbol{w}^T L]_{ij} \tag{20}$$

Clearly, M depends on the value of $\boldsymbol{w}$. But in the early stages of the optimization problem the $\boldsymbol{w}$ are mostly random and we can obtain an expected value for M, averaged over different initializations. Assuming a random initialization for the θ in $\boldsymbol{w}(\theta)$, we can assume that in the early stages $\boldsymbol{w}_i$ have a random normal distribution. This means that the expectation value $\mathbb{E}[\boldsymbol{w}_k\boldsymbol{w}_m] = \frac{1}{d}\|\boldsymbol{w}\|^2\delta_{km}$ and that for other powers $\mathbb{E}[\boldsymbol{w}^n] = 0$ for all $n \neq 2$. Denoting $\overline{M} \equiv \mathbb{E}[M]$ over initializations[RY: we only have one instance], we have

$$\overline{M} = L\mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T]L^T = LL^T \tag{21}$$

We want to see whether some choices for the neural network architecture in $\boldsymbol{w}(\theta)$ can yield a K which would accelerate GD in this convex problem. Let's consider a case where the neural network has an NTK which commutes with $\overline{M}$, meaning $[\overline{M}, K] = \overline{M}K - K\overline{M} = 0$. In this case, we can simultaneously diagonalize $\overline{M}$ and K and let ψ_α denote this shared eigenbasis. Since $\overline{M} = LL^T$ and because L is a symmetric square matrix (since it's positive semi-definite), the eigenvectors ψ_α are also L. Thus we can write

$$K = \sum_\alpha k_\alpha \psi_\alpha \psi_\alpha^T, \qquad L = \sum_\alpha \lambda_\alpha \psi_\alpha \psi_\alpha^T, \qquad \overline{M} = \sum_\alpha m_\alpha \psi_\alpha \psi_\alpha^T, \qquad m_\alpha = \lambda_\alpha^2. \tag{22}$$

If we assume that for a given initialization the value of M is close to $\overline{M}$ [ND: can we assume this? What are the corrections? of order σ_M?], we can obtain an approximate approximate value for the drop in loss using $M \approx baM$. Plugging equation 22 into equation 11 we have

$$\begin{aligned}\frac{d\mathcal{L}}{dt} &= -\varepsilon\,\mathrm{Tr}[M^T K] \approx -\varepsilon\,\mathrm{Tr}[\overline{M}^T K] \\ &= \sum_{\alpha,\beta} \lambda_\alpha^2 k_\beta [\psi_\alpha^T\psi_\beta][\psi_\beta^T\psi_\alpha] = \sum_\alpha \lambda_\alpha^2 k_\alpha\end{aligned} \tag{23}$$

where we used $\psi_\alpha^T\psi_\beta = \delta_{\alpha\beta}$ [RY: why this notation?].

Accelerating Convex Optimization Let's interpret the consequences of equation 23. If the NTK was trivial, like $K \propto I$, which satisfies $[\overline{M}, K] = 0$, equation 23 states that the sum of the squared eigenvalues of L determine the rate by which the loss drops. How can K accelerate this process. Obviously, multiplying K by a number larger than 1 will increase $d\mathcal{L}/dt$, but that simply amounts to increasing the learning rate ε and may lead to numerical instabilities in a discrete GD implementation. To avoid this trivial case, let's say under the constraint that $\mathrm{Tr}[K]$ is constant [ND: or determinant?], what can K accelerate the problem? One possibility is that some λ_α are much smaller than others. Let $m = \mathrm{Tr}[\overline{M}] = \sum_\alpha m_\alpha = \sum_\alpha \lambda_\alpha^2$ and $\mathrm{Tr}[K] = \sum_\alpha k_\alpha = 1$. The condition for K to accelerate the drop in loss is to have

$$\sum_\alpha m_\alpha k_\alpha > \sum_\alpha m_\alpha = m \qquad\qquad \Rightarrow \sum_\alpha \frac{\lambda_\alpha^2}{m} k_\alpha > 1 \tag{24}$$

There are many ways to satisfy equation 24. For example, k_α can be larger for larger λ_α, increasing their weight in the sum. One issue with this choice is that it will also increase the learning rate for the evolution of $\boldsymbol{w}$ along ψ_α, because

$$\psi_\alpha^T \frac{d\boldsymbol{w}}{dt} = -\varepsilon\psi_\alpha^T \frac{\partial\mathcal{L}}{\boldsymbol{w}} = -\varepsilon\psi_\alpha^T L\boldsymbol{w} = -\varepsilon\lambda_\alpha\psi_\alpha^T\boldsymbol{w} \tag{25}$$

Thus, to avoid numerical instabilities during GD, we need to reduce the learning rate ε accordingly to ε/k_α. This not only effectively cancels the advantage gained by k_α, but it also reduces the rate of change of all ψ_β with $\lambda_\beta < \lambda_\alpha$, slowing down the optimization even more. The takeaway from this example is that instead of magnifying the larger λ_α, we should focus on magnifying *smaller* λ_α. meaning k_α should be *larger* when λ_α is *smaller*. For example, we may have $k_\alpha = 1/(\lambda_\alpha + c)$, but

this penalizes larger λ_α a lot and may end up slowing down the optimization. Another choice could be $k_\alpha = c + \max_\beta[\lambda_\beta^p] - \lambda_\alpha^p$ for some exponent $p \geq 1$ and $c > 0$.

Hence, to accelerate the optimization, all we need is a neural network for $\boldsymbol{w}(\theta)$ whose NTK has eigenvalues which are positive and where k_α is a decreasing function of λ_α. We will now discuss how to construct such a neural network explicitly. The key is that its NTK must commute with $\overline{M} = L^2$ and hence with L. Let's begin with a linear ansatz for the neural network, yielding

$$\boldsymbol{w}_i(\theta) = \sum_a A_{ia}\theta_a, \qquad \Rightarrow K_{ij} = \sum_a A_{ia}A_{ja} = [AA^T]_{ij} \tag{26}$$

We need K to commute with $\overline{M} = LL^T$. An easy way to achieve this is if A is a function of L. For instance, let $\theta \in \mathbb{R}^d$, so that A is a $d \times d$ matrix. Choosing $A = (m + c)I - \overline{M}$ for $0 < c \ll m = \mathrm{Tr}[\overline{M}]$, and using $K = AA^T$, the eigenvalue of K for eigenvector ψ_α becomes

$$k_\alpha = \left(m + c - \lambda_\alpha^2\right)^2, \qquad m = \sum_\beta \lambda_\beta^2 \tag{27}$$

Thus, the choice $\boldsymbol{w} = ((m + c)I - L^2)\theta$ yields k_α which is a decreasing function of λ. If we had $\sum_\alpha k_\alpha = 1$, this choice for $\boldsymbol{w}$ would satisfy the condition in equation 24, but it currently does not. [ND: Does a degree-normalized version satisfy it? To say that we still have the same maximum descent rate, we want $k_\alpha = 1$ for the max λ_α. Similarly for the smallest eigenvalue λ_β, we want $k_\beta \lambda\beta^2 \sim \lambda_\alpha^2$. So, we roughly want $k_\alpha \approx \max_\beta \lambda_\beta^2/\lambda_\alpha^2$, meaning $K \sim L^{-2}$ Since inverting L may be expensive (if it's a large matrix), we can approximate the inverse using $1/a \approx \sum_n (1-a)^n$. So to invert L^2, we can use $A = \sum_p (I - L^2)^p$]

2.4 More General ODE

In general, the ODE can be highly nonlinear, meaning $\partial\mathcal{L}/\partial\boldsymbol{w}$ contains terms which are $O(\boldsymbol{w}^p)$ with $p > 1$. If $\mathcal{L}$ is a real analytic function, we can express it as a Taylor series around some local minimum $\boldsymbol{w}_0$. We will use the notation $\partial_i\mathcal{L} \equiv \partial\mathcal{L}/\partial\boldsymbol{w}^i$, where $\boldsymbol{w}^i$ is the ith component of the vector $\boldsymbol{w}$. We will also use Einstein summation rule where an upper and a lower index are summed over $A \cdot B = A^iB_i = \sum_i A^iB_i$. Denoting $\Delta\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}_0$, the Taylor expansion of $\mathcal{L}$ around $\boldsymbol{w}_0$ is given by

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) =& \mathcal{L}(\boldsymbol{w}_0) + \partial_i\mathcal{L}(\boldsymbol{w}_0)\Delta\boldsymbol{w}^i + \frac{1}{2}\partial_i\partial_j\mathcal{L}(\boldsymbol{w}_0)\Delta\boldsymbol{w}^i\Delta\boldsymbol{w}^j \\ &+ \sum_{p\geq 3}\frac{1}{p!}\partial_{i_1}\cdots\partial_{i_p}\mathcal{L}(\boldsymbol{w}_0)\Delta\boldsymbol{w}^{i_1}\cdots\Delta\boldsymbol{w}^{i_p}\end{aligned} \tag{28}$$

where, for brevity, we have used the notation $\partial_i \cdot \partial_j\mathcal{L}(\boldsymbol{w}_0) \equiv \partial_i \cdot \partial_j\mathcal{L}(\boldsymbol{w})|_{\boldsymbol{w}\to\boldsymbol{w}_0}$. Since $\boldsymbol{w}_0$ is a local minimum, $\partial_i\mathcal{L}(\boldsymbol{w}_0) = 0$. Using equation 28, we can rewrite equation 2 as

$$\frac{-1}{\epsilon}\frac{d\boldsymbol{w}_i}{dt} = \partial_i\partial_j\mathcal{L}(\boldsymbol{w}_0)(\boldsymbol{w} - \boldsymbol{w}_0) + O((\Delta\boldsymbol{w})^2) \tag{29}$$

For a general, non-convex $\mathcal{L}$ this Taylor expansion cannot be used during optimization because the local minimum $\boldsymbol{w}_0$ is not known. Yet, if the higher order terms in $\mathcal{L}$ satisfy certain conditions, we may be able to go further. For example, if $\mathcal{L}$ is a function of pair-wise distances $|\boldsymbol{w}_i - \boldsymbol{w}_j|$

[ND: Derive conditions for quadratic term to be larger than the rest during the majority of the optimization process. Use the intuition of weakness of nonlinear terms in FDL as a guide and see what more general conditions need to b satisfied for the quadratic term to dominate.]

3 Background

We first describe the general optimization problem on graphs, followed by several examples.

3.1 OPTIMIZATION PROBLEM

Given a graph with N nodes and each node i has input features $\mathbf{x}_i \in \mathbb{R}^D$. Consider the following optimization problem over the matrix $X := (\mathbf{x}_1, \cdots, \mathbf{x}_N)$

$$\min_{\mathbf{x}} \mathcal{L}(X, A), \quad \text{s.t.} \quad C(X) \leq 0 \tag{30}$$

Where $\mathcal{L}$ is the loss function, A is the graph adjacency matrix, and C is the set of constraints.

Ising Model. Ising model plays an important role in probabilistic graphical model Koller & Friedman (2009). It defines the probabilistic distribution over discrete variables $\mathbf{x}_i \in \{+1, 1\}$ as :

$$p(X) = \exp(-E(X)), \quad E(X) = \sum_{ij} K_{ij}\mathbf{x}_i\mathbf{x}_j + \mu \sum_j h_j \mathbf{x}_j \tag{31}$$

where $E(X)$ is the energy potential. Maximizing the likelihood of the Ising model is equivariant to minimizing the energy potential $E(X)$, hence $\mathcal{L}(X) = E(X)$.

Synchronization Problems [ND: Also talk about the XY model] [RY: add an illustration fig] Synchronization of phase oscillators has profound impact on engineering, physics and social systems (Pikovsky et al. (2003)). One well-known example is the *Kuramoto model*, which couples together each phase oscillator by the sine of their phase differences (Kuramoto (1975), Kuramoto (1984)). Consider N homogeneous oscillators with the same frequency, the governing equation and the optimization problem is the following:

$$\frac{d\mathbf{x}_i}{dt} = \sum_{j=1}^{N} A_{ji} \sin(\mathbf{x}_j - \mathbf{x}_i) = \frac{dL}{dx_i}, \quad \mathcal{L} = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ji} \cos(\mathbf{x}_j - \mathbf{x}_i) \tag{32}$$

Here $\mathbf{x}_i \in [0, 2\pi]$is the phase of the oscillator i, $A_{ji} = 1$ if j is coupled to oscillator i. Townsend et al. (2020) shows that if all interactions are symmetric, aka $A_{ij} = A_{ji}$, this is a gradient system. It means that all the attractors are equilibrium points, and to investigate the system's long-term dynamics and pattern formation, it suffices to analyze these equilibrium states.

On a two-dimensional square lattice, Lauter et al. (2015) showed how identical limit-cycle oscillators synchronize and create deterministic pattern. Considering the oscillators are weakly coupled, we can obtain an equation for the resonator phases called *Hopf-Kuramoto model*:

$$\frac{d\mathbf{x}_i}{dt} = C \sum_{<ji>} \cos(\mathbf{x}_j - \mathbf{x}_i) + S_1 \sum_{<ji>} \sin(\mathbf{x}_j - \mathbf{x}_i) +$$

$$+S_2\{\sum_{<ji>}\sum_{<kj>} [\sin(2\mathbf{x}_j - \mathbf{x}_k - \mathbf{x}_i) - \sin(\mathbf{x}_k - \mathbf{x}_i)] + \sum_{<ji>}\sum_{<kj>} \sin(\mathbf{x}_k + \mathbf{x}_j - 2\mathbf{x}_i)\} \tag{33}$$

Here $< ij >$ means the nearest neighbors. It may not reach a unique stable state, but having different regimes of behavior with phase transitions. Despite the fact that this system may never reach a stationary state, it is not a gradient system, we can explore its phase field [RY: change to different language]. Hence, similar to the Kuramoto-model, we can introduce a loss function by integrating Eqn. equation 33:

$$\mathcal{L}(X) = -C \sum_i \sum_{<ji>} \sin(\mathbf{x}_j - \mathbf{x}_i) + S_1 \sum_i \sum_{<ji>} \cos(\mathbf{x}_j - \mathbf{x}_i) +$$

$$+S_2 \sum_i \{\sum_{<ji>}\sum_{<kj>} [\cos(2\mathbf{x}_j - \mathbf{x}_k - \mathbf{x}_i) - \cos(\mathbf{x}_k - \mathbf{x}_i)] + \frac{1}{2}\sum_{<ji>}\sum_{<ki>} \cos(\mathbf{x}_k + \mathbf{x}_j - 2\mathbf{x}_j)\}$$

We know that running a dynamics in a large lattice are computationally expensive especially close to the phase transition points where difficult to separate the slow and fast modes.[RY: what is the traditional method for solving synchronization]

Persistent Homology Persistent homology Edelsbrunner et al. (2008) is an algebraic tool for measuring topological features of shapes and functions. It provides precise definition of qualitative features, is computable with linear algebra and robust to perturbation of input data otter2017roadmap.

Homology describes the general characteristics of data in a metric space, and is categorized 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 simplicial complexes. Building simplicial complexes enable not only fast homology computation with linear algebra, but also approximating the topological space with its subsets.

In order to obtain a robust, and especially scale-invariant homology characterization, the concept of filtration is introduced to form simplicial complexes that reflect homological features at different scale resolutions. A feature is considered "persistent" if it continue to exist across a wide range of scales.

There are different ways to build simplicial complexes from given data points and resolutions. 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.

An example of persistent homology is point clouds optimization Gabrielsson et al. (2020); Carriere et al. (2021). Given point clouds $\mathbf{x}$, we aim to find the coordinates of the points that maximize the homology.

$$\mathcal{L}(X) = -\sum_{p \in D} \|p - \pi_\Delta(p)\|_\infty^2 + \sum_i \|\mathbf{x}_i - 1\|^2 \tag{34}$$

where $D = \{(b_i, d_i)\}_{i \in I_k}$ is the persistence diagram [RY: what is I_k], consisting of pairs of birth and death parameter of a homology class. π_Δ is the projection onto the diagonal Δ and $\sum_i d(\mathbf{x}_i, S)$ constrains the points within the unit square.

For large graphs, traditional methods suffer from computational bottlenecks. For example, solving persistent homology [RY: numbers, big o notation]. Next, we describe a novel reparametrization to signficantly speed up the optimization process.

4 METHODOLOGY

4.1 REPARAMETERIZATION WITH GNN

Due to the non-convex nature of the optimization problem, reaching a good local optimum or an equilibrium state can be computationally expensive. The core idea of our approach to accelerate optimization is to parametrize the node features X as the output of a deep neural network.

Specifically, [RY: generalize the notation to multiple layers] we start with a high-dimensional random embedding $G_0 \in \mathbb{R}^{N \times h_0}$. Then we apply GNN layers to the random embedding sequentially:

$$G_1 = \sigma(f(A)ZW^{(1)}), \quad G_2 = \sigma(f(A)G_1W^{(2)}), \quad X = \sigma(GW + b) \tag{35}$$

where $f(A) = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is the propagation rule in the GNN Kipf & Welling (2016). Propagation rule is critical to the representation power of GNNs Dehmamy et al. (2019). Here we use the normalized adjacency matrix $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix of $\tilde{A}$. The new embeddings $G_1 \in \mathbb{R}^{N \times h_1}$ and $G_2 \in \mathbb{R}^{N \times h_2}$ are reparametrized via GNNs. $W^{(1)}$ and $W^{(2)}$ are the trainable weight matrices. We concatenate G_0, G_1 and G_2 along the feature dimension to obtain the node embedding $G = [G_0, G_1, G_2] \in \mathbb{R}^{N \times h}$ where $h = h_0 + h_2 + h_2$.

Finally, we apply a fully connected network to project the h-dimensional embedding G to the original d dimension features X. We allow G_0 to be freely trainable and $W \in \mathbb{R}^{h \times d}$ is the projection weight matrix. The set of trainable parameters is $\theta = \{G_0, W^{(1)}, W^{(2)}, W, b\}$. In this way, the original optimization problem over X becomes the optimization over θ. We initialize all the parameters θ randomly and optimize them through back-propagation.

[RY: include model architecture]

5 EXPERIMENTAL RESULTS

5.1 NETWORK OF KURAMOTO OSCILLATORS

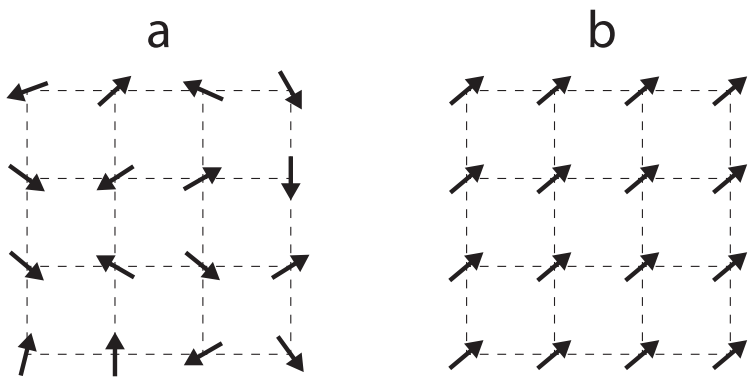

Figure 1: a) disorganized b) organized phase

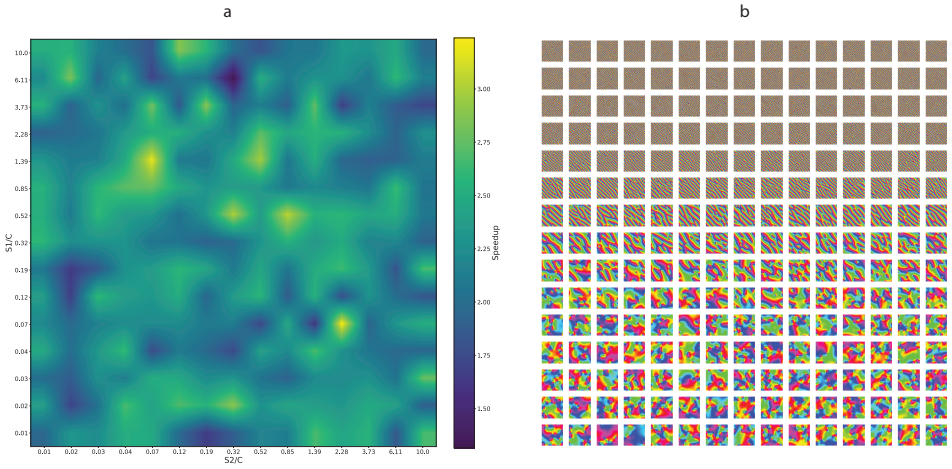

Figure 2: Speedup 2d lattice

5.2 POINT CLOUD OPTIMIZATION

[Jianzhi]

6 RELATED WORK

Graph Neural Network GNNs received considerable attention in recent years, see surveys and the references therein Bronstein et al. (2017); Zhang et al. (2018); Wu et al. (2019); Goyal & Ferrara (2018).

Topological Optimization Sosnovik & Oseledets (2019) leverage convolutional neural networks as the efficient pixel-wise image labeling technique to perform the topology optimization. Hoyer et al. (2019) propose using neural networks as prior to improve the parameterization of structural optimization.

REFERENCES

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.

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.

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

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.

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

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*, pp. 1553–1563. PMLR, 2020.

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

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

Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018.

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

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

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

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

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.

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

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

Alex Townsend, Michael Stillman, and Steven H Strogatz. Dense networks that do not synchronize and sparse ones that do¡? a3b2 show [editpick]?¿. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(8):083142, 2020.

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.

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

What type of problems are good fits What is the scaling factor Spin-glass [RY: Related to pre-conditioning in numerical optimization?]

Theorem 2 (Maximum loss drop)**.** *After the reparametrization $\boldsymbol{w}(\theta)$, $d\mathcal{L}(\boldsymbol{w}(\theta))/dt$ is maximized when $[K, M] = 0$.*

To show this, we use the positive semi-definiteness of M and K and show that $\mathrm{Tr}[MK^T]$ has an upper bound $\sum_i m_i k_i$ which is achieved when $[M, K] = 0$.

Lemma 2.1. *Both M and K are symmetric and positive semi-definite.*

Proof. Define $D_i \equiv \partial\mathcal{L}/\partial\boldsymbol{w}_i$ and $B_{ia} \equiv \partial\boldsymbol{w}_i/\partial\theta_a$. We then have $M = DD^T$ and $K = BB^T$. Therefore, for any vector $\boldsymbol{v} \in \mathbb{R}^d$ we have

$$\boldsymbol{v}^T M \boldsymbol{v} = \|\boldsymbol{v}^T D\|^2 \geq 0 \qquad \boldsymbol{v}^T K \boldsymbol{v} = \|\boldsymbol{v}^T B\|^2 \geq 0 \tag{36}$$

□

As a result, the spectral expansions $M = \sum_i m_i \psi_i \psi_i^T$ and $K = \sum_a k_i \phi_i \phi_i^T$ yield two complete bases ψ_i and ϕ_i for $\mathbb{R}^d$ and a set of eigenvalues $m_i \geq 0$ and $k_i \geq 0$. We have

Writing out $\mathrm{Tr}[MK^T]$ in terms of these spectral expansions we have

$$\mathrm{Tr}\left[MK^T\right] = \sum_i m_i \psi_i^T K \psi_i = \sum_{i,j} m_i k_j (\psi_i^T \phi_j)^2 = \sum_{i,j} m_i k_j c_{ij}^2 \tag{37}$$

We can show that equation 37 has an upper bound:

Lemma 2.2 (Upper bound)**.** $\mathrm{Tr}\left[MK^T\right] \leq \sum_i m_i k_i$

Proof. Index the spectral expansion of M and K such that for $i < j$, $m_i \geq m_j$ and $k_i \geq k_j$. We can show that the sum in $\mathrm{Tr}[MK^T]$ is larger when the larger eigenvalues m_i and k_i are matched to each other, meaning if $\psi_i^T \phi_i = 0$. To see this, consider the terms in $\mathrm{Tr}[MK^T]$ involving only i, j for a fixed $i < j$

$$T_i \equiv m_i \left(k_i c_{ii}^2 + k_j c_{ij}^2\right) = m_i k_i + m_i k_i \left(c_{ii}^2 + c_{ij}^2 - 1\right) + m_i (k_j - k_i) c_{ij}^2 \tag{38}$$

and the same for T_j with $i \leftrightarrow j$. Note that because $k_j - k_i \leq 0$ and $c_{ii}^2 + c_{ij}^2 \leq 1$. Using the completeness of the bases ψ_i and ϕ_i, we have

$$\sum_i c_{ij}^2 = \sum_i (\psi_i^T \phi_j)^2 = \|\phi_j\|^2 = 1 \tag{39}$$

and, similarly, $\sum_j c_{ij}^2 = 1$. Therefore,

$$c_{ii}^2 = 1 - c_{ij}^2 - \sum_{q \neq i,j} c_{iq}^2 = 1 - c_{ji}^2 - \sum_{q \neq i,j} c_{qi}^2 \tag{40}$$

Using this we can take the derivative of T_i and T_j with respect to c_{ij}^2 and c_{ji}^2, which yields

$$\frac{\partial T_i}{\partial c_{ij}^2} = m_i (k_j - k_i) \qquad \frac{\partial T_i}{\partial c_{ji}^2} = -m_i k_i \tag{41}$$

and similarly for T_j. Putting these together, we have

$$\begin{aligned} \frac{\partial (T_i + T_j)}{\partial (c_{ij}^2 + c_{ji}^2)} &= m_i (k_j - k_i) - m_i k_i + m_j (k_i - k_j) - m_j k_j \\ &= -(m_i - m_j)(k_i - k_j) - m_i k_i - m_j k_j \leq 0 \end{aligned} \tag{42}$$

Since $c_{ij}^2 + c_{ji}^2 \geq 0$, it follows that $T_i + T_j$ has a maximum when $c_{ij}^2 + c_{ji}^2 = 0$, meaning when $\psi_i^T \phi_j = \psi_j^T \phi_i = 0$. This result does not state what the value of the rest of c_{iq} for $q \neq i, j$ should be. But starting from $i = 1$, we find that $\psi_i^T \phi_j = 0$ for all $j > 1$, meaning $\phi_1 = \pm\psi_1$. Since ϕ_1 and ψ_i are matched together, we are left with the subspace spanned by $i = 2, \ldots d$. Using induction, the

same argument yields $\phi_i = \pm\psi_i$. Hence, using the maximum of $\mathrm{Tr}[MK^T]$ occurs when $\phi_i = \pm\phi_i$, which yields

$$\mathrm{Tr}[MK^T] = \sum_{i,j} m_i k_j (\psi_i^T \phi_j)^2 \leq= \sum_{i,j} m_i k_j \delta_{ij}^2 = \sum_i m_i k_i \tag{43}$$

with δ_{ij} being the Kronecker delta. □

Lemma 2.2 proves that $|d\mathcal{L}/dt| = \varepsilon\,\mathrm{Tr}[MK^T]$ is maximized when M and K share the same eigenvectors, which means when they commute, $[M, K] = 0$, which proves Theorem 2.

Graph Neural Tangent Kernel For sufficiently over-parametrized neural network $f(\theta, \mathbf{x})$ [RY: how is SGD being used?], Jacot et al. (2018) shows that the gradients of the network with respect to its parameters remains constant during training:

$$H(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \frac{\partial f(\theta, \mathbf{x}_i)}{\partial \theta}, \frac{\partial f(\theta, \mathbf{x}_j)}{\partial \theta} \right\rangle \tag{44}$$

where the loss $\mathcal{L}(X) = \sum_i (f(\theta, \mathbf{x}_i) - y_i)^2$[RY: this is convex]

Fast and slow mode are good problem

[ND: What is the main question we want to tackle? We want to show that certain optimization problems, in general, could be speed up using the trick we realized. The key is there must exist a hierarchy in the rate of evolution of different modes. If those modes can somehow be identified, we could be able to use the GCN trick.

]