How Graph Neural Networks Learn: Lessons from Training Dynamics

Chenxiao Yang [1 *] **Qitian Wu** [1] **David Wipf** [2] **Ruoyu Sun** [3 4] **Junchi Yan** [1]

Abstract

A long-standing goal in deep learning has been to characterize the learning behavior of black-box models in a more interpretable manner. For graph neural networks (GNNs), considerable advances have been made in formalizing what functions they can represent, but whether GNNs will learn desired functions during the optimization process remains less clear. To fill this gap, we study their training dynamics in function space. In particular, we find that the gradient descent optimization of GNNs implicitly leverages the graph structure to update the learned function, as can be quantified by a phenomenon which we call *kernel-graph alignment*. We provide theoretical explanations for the emergence of this phenomenon in the over-parameterized regime and empirically validate it on real-world GNNs. This finding offers new interpretable insights into when and why the learned GNN functions generalize, highlighting their limitations in heterophilic graphs. Practically, we propose a parameter-free algorithm that directly uses a sparse matrix (i.e. graph adjacency) to update the learned function. We demonstrate that this embarrassingly simple approach can be as effective as GNNs while being orders-of-magnitude faster.

1. Introduction

Graph Neural Networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008; Bruna et al., 2014; Kipf & Welling, 2017) represent network architectures for learning on entities with relations and interactions. In addition to their empirical success, the pursuit of theoretical understanding has also led researchers to dissect GNNs in terms of their representation powers (a.k.a. expressiveness) (Maron et al., 2019; Xu et al., 2019; Oono & Suzuki, 2019; Chen et al., 2019), which aim to answer what families of functions GNNs can represent or approximate. However, beyond these studies, it remains unclear whether GNNs will indeed learn the desired function during the training process. Filling this gap calls for more attention to the optimization of GNNs and effects of graph structures that implicitly bias the learning process towards certain solutions. In this paper, we analyze the training dynamics of GNNs in function space, aiming to answer

Do GNNs indeed learn the desired function that can generalize during the training process, and if so, how?

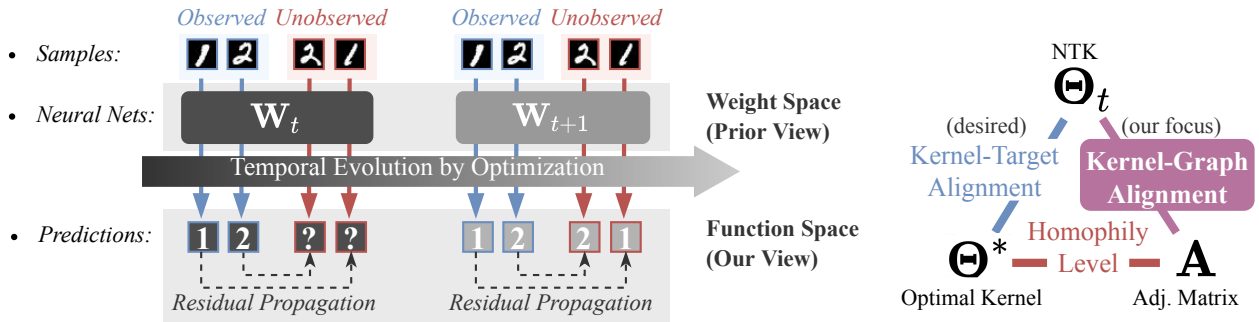
We note that the exact mathematical characterization of the optimization of neural networks, even for shallow ones, is prohibitive or even impossible due to their non-linearity. Existing work thus resorts to simplifications, such as removing activations (e.g., Xu et al. (2021a) for GNNs) or training a single neuron (e.g., Bai & Lee (2020); Frei et al. (2020)). Moreover, even with these simplifications, there is generally no single straightforward answer to the above question; only insights from different angles exist for illuminating certain mechanisms in this complex process (e.g., Cao et al. (2019); Papyan et al. (2020); He & Su (2020)). In this work, we provide initial answers to this open question by studying the role of graph structures in the training process. Particularly, we have the following informal statement:

The optimization of GNNs implicitly leverages the graph structure, as is quantified by kernel-graph alignment, to learn functions that can generalize.

More formally, the so-called *Kernel-Graph Alignment* characterizes such a graph implicit bias during training, where the so-called *Neural Tangent Kernel (NTK)* (Jacot et al., 2018) that controls the evolution of the learned GNN function tends to align with the message passing matrix (typically implemented as graph adjacency) used in GNNs' forward pass. This result is not only useful for understanding the optimization of GNNs but also allows us to explain a series of previously unresolved theoretical questions and develop practically useful algorithms in a principled manner. Our contributions are summarized as follows:

[*] Work was partially done during an internship at Amazon Web Services. [1]School of Artificial Intelligence & Department of Computer Science and Engineering & MoE Lab of AI, Shanghai Jiao Tong University [2]Amazon Web Services [3]School of Data Science, The Chinese University of Hong Kong, Shenzhen [4]Shenzhen International Center for Industrial and Applied Mathematics, Shenzhen Research Institute of Big Data. Correspondence to: Junchi Yan <yanjunchi@sjtu.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).



(a) Training dynamics

(b) Alignment of matrices

Figure 1: **(a)** Training dynamics of GNNs in function space where residuals (i.e. difference between labels and predictions) propagate from observed to unobserved samples based on a kernel similarity measure. **(b)** The kernel matrix $\mathbf{\Theta}_t$ naturally aligns with the adjacency matrix $\mathbf{A}$, which is favorable for generalization if $\mathbf{A}$ is inherently close to the optimal kernel $\mathbf{\Theta}^*$.

Warm-Up Example: Residual Propagation. As a warm-up example, we propose a class of learning algorithms that replace the NTK matrix in the training dynamics with a sparse adjacency matrix, representing an extreme case where the kernel perfectly aligns with the graph. We dub this method *Residual Propagation* since it simultaneously minimizes the loss and yields predictions by purely propagating residuals (a.k.a. errors) without the need for trainable parameters or back-propagation. The algorithm has interesting connections with two classic non-parametric algorithms: label propagation (Zhou et al., 2003) and kernel regression (Shawe-Taylor & Cristianini, 2004). Strong empirical evidence across 15 benchmarks, including three OGBN datasets, demonstrates that this embarrassingly simple method has surprisingly good generalization performance, competing with non-linear GNNs. Additionally, compared with GNNs, it offers up to thousands of times speedup and requires ten times less memory due to its linear complexity. This algorithm serves as a simple example illustrating the inner mechanism of GNNs' optimization. (**Section 3**)

Implicit Leverage of Graphs in Optimization. Next, we formally study how the graph structure is leveraged in the optimization of GNNs. To make the analysis tractable, we examine the training dynamics of GNNs in the overparameterized regime (Jacot et al., 2018; Lee et al., 2019), where the model asymptotically converges to its first-order Taylor expansion around its initialization as an approximation. Despite this approximation, the message passing modules and non-linearities in the model, which are critical for the success of GNNs, are preserved. A rich body of literature (e.g. see Golikov et al. (2022) and references therein) also conduct analysis in this regime for gaining practical insights into otherwise prohibitive problems. Using this approach, we provide explicit mathematical formulas for recurrently computing the NTK of GNNs (which we call node-level GNTK) with arbitrary model depth and inputs. The formula demonstrates how the graph structure is naturally integrated into the kernel function, inducing the kernel-graph alignment phenomenon. Additional examples of shallow (two- and single-layer) GNNs with fixed inputs further show cases where the kernel function becomes equivalent to special forms of adjacency. (**Section 4**)

Generalization and Failure Modes. Stepping further, we provide interpretable explanations of how the graph implicit bias in optimization leads to generalization of the learned GNN function, and why GNNs struggle with certain graph learning problems such as heterophily. As illustrated in Fig. 1(b), we introduce another matrix called the optimal kernel matrix, which denotes if a pair of instances share the same label. The alignment of this matrix with the adjacency matrix quantifies the homophily level of the (Zhu et al., 2020), a data characteristic that has been empirically shown to be highly relevant to GNNs' generalization performance. Intuitively, for homophilic graphs, a larger homophily level with good kernel-graph alignment indicates that the kernel function approaches the optimal one (the so-called kernel-target alignment (Cristianini et al., 2001)); this is desired for favorable generalization, as labels of training instances tend to flow to testing instances with the same label during optimization. Theoretically, we establish a strong correlation between generalization and homophily by deriving a data-dependent generalization bound that highly depends on the graph heterophily, and also showing that GNNs are the Bayesian optimal prior model architecture that minimizes the population risk on homophilic graphs. during optimization on datasets with diverse characteristics. (**Section 5**)

Empirical Verification. To further verify the theory, we numerically study the evolution of real-world GNNs during the GD-based training process on synthetic and real-world datasets. We found that their NTKs indeed align

with the message passing matrix used in the forward pass. On homophilic graphs, alignment with the graph promotes alignment with the optimal kernel matrix and thus improves generalization, whereas on heterophilic graphs, it adversely affects generalization. Additionally, we observed that GNNs are capable of gradually adapting themselves to align with the optimal kernel regardless of different graph structures, which might be of independent interest for understanding feature learning in GNNs. (**Section 6**)

Finally, we note that our analysis may only explain a limited part of an under-explored complex problem, and hope our analytical framework to pave the way for dissecting other graph learning tasks (such as link prediction and representation learning), and to be used to gain insights into, and methodology to solve other practically relevant GNN issues. We conclude our paper by discussing such possibilities and more related works. Our codes are available at `https://github.com/chr26195/ResidualPropagation`. (**Section 7**)

2. Preliminary

Notation and Setup. Given a training set with n_l labeled instances $\mathbf{X} = \{\boldsymbol{x}_i\}_{i=1}^{n_l} \in \mathbb{R}^{n_l \times d}$ and $\mathbf{Y} = \{y_i\}_{i=1}^{n_l} \in \mathbb{R}^{n_l}$, we aim to learn a predictive function $f(\boldsymbol{x})$ parameterized by weights $\mathbf{W}$. For (semi-)supervised learning, we minimize the squared loss $\mathcal{L}$ using *Gradient Descent (GD)*,

$$\mathcal{L} = \|\mathbf{F}_t - \mathbf{Y}\|^2/2, \quad \partial \mathbf{W}_t/\partial t = -\eta \nabla_{\mathbf{W}} \mathcal{L}, \tag{1}$$

where $\mathbf{W}_t$ and $\mathbf{F}_t = \{f_t(\boldsymbol{x})\}_{\boldsymbol{x} \in \mathbf{X}} \in \mathbb{R}^{n_l}$ are weights and predictions for the training set at optimization time index t, and η is the learning rate. Temporal discretization of this gradient flow system with time-step $\Delta t = 1$ yields the fixed step-size GD algorithm commonly used in practice. Let also $\mathbf{X}'$ and $\mathbf{Y}'$ denote testing instances, and $\bar{\mathbf{X}} = [\mathbf{X}, \mathbf{X}'] \in \mathbb{R}^{n \times d}$ (resp. $\bar{\mathbf{Y}}$) the concatenation of training and testing inputs (resp. labels), where n is the full dataset size. For convenience, we generally refer to $f_t(\boldsymbol{x})$ as prediction for a single data point, and allow it to depend also on other nodes' information. $\mathbf{F}_t$ and $\mathbf{F}'_t$ are predictions for the training and testing sets. This setup can be extended to other loss functions and multi-dimensional output (see discussions in Appendix. C.2, C.3).

Similar to (Xu et al., 2021a), we focus on learning node representations, where instances (i.e. nodes) and their relations (i.e. edges) are described by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$. The graph defines a symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ where $\mathbf{A}_{ij} = 1$ for a pair of connected nodes $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ otherwise 0. Based on the data split, we denote submatrices of $\mathbf{A}$ using $\mathbf{A}_{\mathbf{XX}}$ and $\mathbf{A}_{\mathbf{X}'\mathbf{X}}$. Our insights apply to both transductive (i.e. semi-supervised learning) and inductive settings (see Appendix. C.1), but will focus on the former case unless stated otherwise.

Graph Neural Networks (GNNs) are a class of network architectures for learning representations on graphs. For GNNs with ReLU activation, each layer can be written as

$$\mathbf{Z}^{(\ell)} = \mathrm{ReLU}(\mathbf{A}\mathbf{Z}^{(\ell-1)}\mathbf{W}^{(\ell)}) \in \mathbb{R}^{n \times m}, \tag{2}$$

where $\mathbf{Z}^{(\ell)}$ are node representations at the ℓ-th layer with $\mathbf{Z}^{(0)} = \bar{\mathbf{X}}$, and m is the model width. The definition of $\mathbf{A}$ could differ for different GNNs. Our analysis applies to arbitrary $\mathbf{A}$ and will not differentiate adjacency and the actual message passing matrix used in the implementation. We denote GNN prediction for a single data point as $f(\boldsymbol{x}; \mathbf{A})$.

Label Propagation (LP) represents a class of algorithms for semi-supervised learning, where labels $\mathbf{Y}$ propagate along edges to efficiently predict $\mathbf{F}'$. From (Zhou et al., 2003), the LP update equation can be written as $\mathrm{LP}(\mathbf{Y}; k, \alpha) =$

$$[\mathbf{F}_k, \mathbf{F}'_k] = \alpha \mathbf{A}\, [\mathbf{F}_{k-1}, \mathbf{F}'_{k-1}] + (1-\alpha)[\mathbf{Y}, \mathbf{0}], \tag{3}$$

where $[\cdot, \cdot]$ is concatenation, k is the iteration number, and α is a hyperparameter. As initialization $[\mathbf{F}_0, \mathbf{F}'_0] = [\mathbf{Y}, \mathbf{0}]$, and after convergence $[\mathbf{F}_\infty, \mathbf{F}'_\infty] \propto (\mathbf{I}_n - \alpha\mathbf{A})^{-1}[\mathbf{Y}, \mathbf{0}]$. LP algorithms have found wide applicability due to their superior efficiency and scalability.

3. Graph Implicit Bias in Training

We commence by providing a label propagation perspective on the evolution of a general parameterized model during GD-based optimization, whereby we propose a simple non-parametric algorithm for semi-supervised learning. The algorithm shows that explicitly leveraging graph structure in the training dynamics to update the learned function leads to satisfactory generalization performance that is comparable to non-linear GNNs. This serves as an illustrative example, later contributing to our understanding of the optimization of GNNs where they implicitly leverage graph structures.

3.1. Label Propagation View of Gradient Descent

On the training set, one can characterize the evolution of a general parameterized model (with no restriction on model architecture) $f(\cdot)$, induced by GD-based optimization that continuously updates the weights $\mathbf{W}_t$ as (Jacot et al., 2018):

$$\begin{aligned} \partial \mathbf{F}_t/\partial t &= \eta\, \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X})\mathbf{R}_t \\ \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}) &\triangleq \nabla_{\mathbf{W}} \mathbf{F}_t^\top \nabla_{\mathbf{W}} \mathbf{F}_t \in \mathbb{R}^{n_l \times n_l}, \end{aligned} \tag{4}$$

where $\mathbf{R}_t = \mathbf{Y} - \mathbf{F}_t \in \mathbb{R}^{n_l}$ denotes *residuals* (Hastie et al., 2009) (a.k.a. errors), the difference between ground-truth labels and model predictions. The so-called *Neural Tangent Kernel (NTK)* (Jacot et al., 2018) $\mathbf{\Theta}_t(\mathbf{X}, \mathbf{X})$ is produced by the product of Jacobians, which is dependent on the network architecture and evolves over time due to its association with time-varying weights. Intuitively, it quantifies

similarity between instances based on how differently their outputs change by an infinitesimal perturbation of weights. Specially, if the kernel is constant (such as inner-product kernel for linear models), (4) reduces to the training dynamics of kernel regression (Shawe-Taylor & Cristianini, 2004). The derivation of (4) mainly relies on the chain rule; we reproduce it in Appendix B.1 for self-containment.

Residual Dynamics. While (4) has found usage for analyzing the convergence of empirical risk (Du et al., 2019a; Arora et al., 2019) and the spectral bias of deep learning (Mei et al., 2019; Cao et al., 2019), it is restricted to a limited set of samples (i.e. training set). To see how the model evolves on arbitrary inputs for fully characterizing the learned function, we extend (4) to accommodate unseen samples (which could be chosen arbitrarily). Specifically, let $\mathbf{R}'_t = \mathbf{Y}' - \mathbf{F}'_t$ denote residuals for the testing set, and after temporal discretization, the ODE in (4) can be rewritten neatly using a single variable residual $\mathbf{R}$ (see derivation in Appendix B.1),

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta\, \mathbf{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}})\, [\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t]\,, \quad (5)$$

where $\mathbf{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}}) \triangleq \nabla_{\mathbf{W}}[\mathbf{F}_t, \mathbf{F}'_t]^\top \nabla_{\mathbf{W}}[\mathbf{F}_t, \mathbf{F}'_t] \in \mathbb{R}^{n\times n}$ is the NTK matrix between training and testing sets. The $n \times n$ matrix will be henceforth abbreviated as $\mathbf{\Theta}_t$. The equation can be viewed as propagating residuals unidirectionally from training to arbitrary unseen samples based on a similarity measure, controlling the evolution of the learned function.

To provide more intuitions into the inner mechanism of (5), we rewrite it for an arbitrary unseen data point $\boldsymbol{x}'$:

(Ground-truth) label propagation in optimization

$$f_{t+1}(\boldsymbol{x}') = f_t(\boldsymbol{x}') + \eta \sum_{\boldsymbol{x}\in\mathbf{X}} \mathbf{\Theta}_t(\boldsymbol{x}, \boldsymbol{x}')(\, y(\boldsymbol{x}) - f_t(\boldsymbol{x})),$$

where $y(\boldsymbol{x})$ is the ground-truth label for $\boldsymbol{x}$, and $\mathbf{\Theta}_t(\boldsymbol{x}, \boldsymbol{x}') = \nabla_{\mathbf{W}} f(\boldsymbol{x})^\top \nabla_{\mathbf{W}} f(\boldsymbol{x}')$. Intuitively, for an unseen instance $\boldsymbol{x}'$ that is more 'similar' to $\boldsymbol{x}$, more ground-truth label information $y(\boldsymbol{x})$ will then propagate to $\boldsymbol{x}'$, and vice versa (illustrated in Fig. 1(a)). For $f_t(\boldsymbol{x}) \neq 0$, the ground-truth label is adjusted by subtracting current model prediction, i.e. $y(\boldsymbol{x}) - f_t(\boldsymbol{x})$, enabling the propagation process to diminish progressively as errors or residuals are minimized.

Drawing upon an analogy between (5) induced by optimization, and instance-wise propagation schemes commonly seen in graph learning, we have the following hypothesis: *For GNNs defined in Section 2, its optimization process implicitly leverages the graph structure to update the learned function by aligning their $\mathbf{\Theta}_t$ with certain forms of $\mathbf{A}$, which is also a key factor contributing to their good generalization performance.*

3.2. Residual Propagation

To test this hypothesis, we propose a toy algorithm that explicitly introduces graph structure information into training dynamics in (5), by replacing the NTK matrix $\mathbf{\Theta}_t$ with high-order graph adjacency matrix $\mathbf{A}^K$. While original (5) is expensive to run, such replacement allows us to actually implement it as a practically useful semi-supervised algorithm that can efficiently run, by taking advantage of the fact that $\mathbf{A}$ is sparse. We dub this algorithm as *Residual Propagation (RP)*, whose update equation is

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^K [\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t]\,. \quad (6)$$

At initialization, $\mathbf{F}_0$ and $\mathbf{F}'_0$ are defined as $\mathbf{0}$, and unknown testing labels are defined as $\mathbf{Y}' = \mathbf{0}$. One can conveniently convert $\mathbf{R}'_t$ back to predictions at a certain time step. More generally, one can flexibly replace the term $\mathbf{A}^K$ with other matrices indicating similarity of samples to broaden the use cases of the algorithm.

Intriguingly, we show the RP algorithm has interesting connections with various classic methods including LP (Zhou et al., 2003) and kernel regression (Shawe-Taylor & Cristianini, 2004), though they emerge from very different contexts.

Proposition 3.1 (Connection with Label Propagation)**.** *The first step of RP in (6) yields identical classification results as LP in (3) (with $\alpha = 1$ and $k = K$):*

$$\begin{aligned} \textit{(First Step of RP):}\quad & [\mathbf{F}_1, \mathbf{F}'_1] = \eta \mathbf{A}^K [\mathbf{Y}, \mathbf{0}], \\ \textit{(Label Propagation):}\quad & \mathrm{LP}(\mathbf{Y}; K, 1) = \mathbf{A}^K [\mathbf{Y}, \mathbf{0}], \end{aligned} \quad (7)$$

and each of subsequent step of RP can also be viewed as LP on adjusted ground-truth labels, i.e. $\mathbf{Y} - \mathbf{F}_t = \mathbf{R}_t$.

Besides the first step, each of subsequent step of RP can also be viewed as LP on adjusted ground-truth labels, i.e. $\mathbf{Y} - \mathbf{F}_t = \mathbf{R}_t$. This result shows RP encompasses LP as a special case; such a connection further motivates a generalized version of RP by combining with other off-the-shelf LP variants (see references in Appendix A.3):

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta\, \mathrm{LP}^*(\mathbf{R}_t) + [\mathbf{R}_t, \mathbf{R}'_t]\,. \quad (8)$$

where $\mathrm{LP}^*(\cdot) : \mathbb{R}^{n_l} \to \mathbb{R}^n$ is a general LP function that takes as input ground-truth labels and outputs predictions.

Theorem 3.2 (Convergence & Connection with Kernel Regression)**.** *For RP in (6) and sufficiently small step size $\eta < 2\sigma_{max}^{-1}[\mathbf{A}^K_{\mathbf{XX}}]$, where $\mathbf{A}^K_{\mathbf{XX}}$ is a submatrix of $\mathbf{A}^K$, and σ_{max} is its largest eigenvalue, $\mathbf{R}_t$ and $\mathbf{R}'_t$ converge for positive definite $\mathbf{A}^K_{\mathbf{XX}}$ or positive semi-definite $\mathbf{A}^K$. Upon convergence in the former case, the predictions are equivalent to the kernel regression solution w.r.t. kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') \triangleq \mathbf{A}^K_{\boldsymbol{x}\boldsymbol{x}'}$*

$$\mathbf{F}_\infty = \mathbf{Y}, \quad \mathbf{F}'_\infty = \mathbf{A}^K_{\mathbf{X}'\mathbf{X}} (\mathbf{A}^K_{\mathbf{XX}})^{-1} \mathbf{Y}. \quad (9)$$

Table 1: Empirical evaluation of RP on `OGB` datasets. Accuracy is reported for `Arxiv` and `Products`, and ROC-AUC for `Proteins`. Last three rows compare RP against full-batch GNN. Results of some baseline are from the official leaderboard.

Model	Feat.	Arxiv		Proteins		Products		# Param.
		Validation	Test	Validation	Test	Validation	Test	
MLP	**X**	57.65 ± 0.12	55.50 ± 0.23	77.06 ± 0.14	72.04 ± 0.48	75.54 ± 0.14	61.06 ± 0.08	$O(\ell m^2)$
LinearGNN	**X**, **A**	70.67 ± 0.02	69.39 ± 0.11	66.11 ± 0.87	62.89 ± 0.11	88.97 ± 0.01	74.21 ± 0.04	$O(dc)$
GNN	**X**, **A**	73.00 ± 0.17	71.74 ± 0.29	79.21 ± 0.18	72.51 ± 0.35	92.00 ± 0.03	75.64 ± 0.21	$O(\ell m^2)$
LP	**A**	70.14 ± 0.00	68.32 ± 0.00	83.02 ± 0.00	74.73 ± 0.00	90.91 ± 0.00	74.34 ± 0.00	0
RP (ours)	**A**	71.37 ± 0.00	70.06 ± 0.00	85.19 ± 0.00	78.17 ± 0.00	91.31 ± 0.00	78.25 ± 0.00	0
Speedup / step		$\times$ 14.48		$\times$ 14.00		$\times$ 12.46		
Time to Acc.		$\times$ 0.01461		$\times$ 0.00008		$\times$ 0.00427		
Memory		$\times$ 0.094		$\times$ 0.363		$\times$ 0.151		

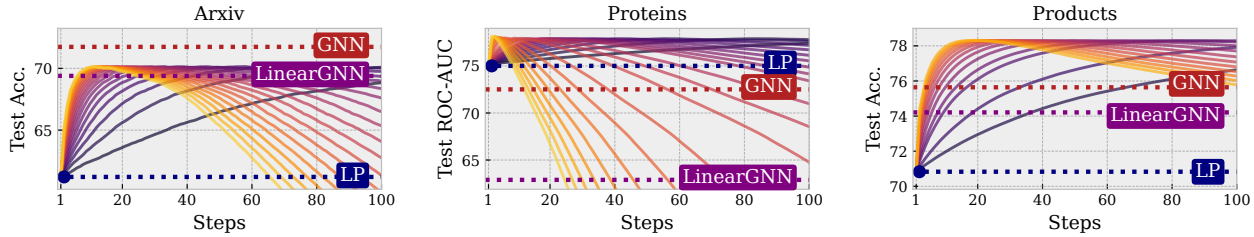


Figure 2: Learning curves of RP and comparison with the performance of LP ($\alpha = 1$), linear GNN and deep GNN. Transition from yellow to purple denotes RP with decreasing step size η.

See proof and a more comprehensive discussion of the convergence of RP and its connection with classic algorithms in Appendix B.2. Different from kernel regression, RP might not necessarily converge since it does not restrict the propagation matrix to be PSD or symmetric. But intriguingly, it can still achieve satisfactory generalization performance by stopping at the step with peak validation performance.

3.3. Preliminary Empirical Verification

Since RP represents an extreme case where the kernel function perfectly aligns with the graph, we test its performance against GNNs to empirically verify whether the graph implicit bias in optimization alone is crucial for satisfactory performance.

Setup. We compare RP with some standard GNN architectures (LinearGNN Wu et al. (2019) and GCN Kipf & Welling (2017)) on a diverse set of 15 datasets, including three challenging `OGB` (Hu et al., 2020) datasets `Arxiv`, `Proteins`, `Products` with up to millions of nodes and edges. Due to space limit, we report results on OGB datasets in the main text, and defer the rest and experimental details to Appendix E and D.1.

Results. In Table 1, the proposed RP demonstrates competitive performance. As depicted in Fig. 2, RP achieves the same performance as LP using one step, and quickly increases until reaching its peak performance, which often surpasses GNNs. Specifically, on `Proteins` where the graph contains relatively richer structural information, a single step of RP exceeds a well-trained deep GNN, while in `Products`, merely four steps of RP exceeds the GNN. Moreover, RP does not require trainable parameters and boosts speed, with each step being more than 10 times faster than each gradient descent step for training GNN. Furthermore, RP can achieve its peak performance using less steps, and thus overall takes significantly less time (up to 1000 times less) to attain GNN-level performance. RP also inherits the scalability of LP and only requires storage space for predictions; thus it consumes up to 10 times less memory in practice.

In Appendices E.1 and E.2 respectively, we discuss generalized RP that combines with kernels (e.g. Gaussian kernel) to incorporate node features. The corresponding update equation can be re-written as $\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] =$

$$-\eta \mathbf{A}^K \mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) \mathbf{A}^K [\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t], \qquad (10)$$

where $\mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$ could be specified as arbitrary kernel functions (such as Sigmoid, Gaussian, etc.) Additionally, we test 12 datasets including homophilic (`Cora`, `Citeseer`, `Pubmed`, `Computer`, `Photo`, `CS`, `Physics`) and heterophilic (`roman-empire`, `amazon-ratings`, `minesweeper`, `tolokers`, `questions`) ones where we find RP can still outperform most popular GNNs.

4. Analysis in Overparameterized Regime

We next theoretically verify that the optimization of GNNs indeed implicitly leverage graph structure for updating the learned function. Further empirical verification will be deferred to the next section.

4.1. Insights from Explicit Formula of GNNs' NTK

Similar to the training dynamics of general parameterized models in (5), the training dynamics of GNNs in node-level tasks is characterized by their NTK defined as follows:

Definition 4.1 (Node-Level GNTK)**.** For a ℓ-layer GNN in node-level tasks defined in Sec. 2, the NTK is defined as

$$\mathbf{\Theta}_t^{(\ell)}(\boldsymbol{x}, \boldsymbol{x}'; \mathbf{A}) = \nabla_{\mathbf{W}} f(\boldsymbol{x}; \mathbf{A})^\top \nabla_{\mathbf{W}} f(\boldsymbol{x}'; \mathbf{A}), \quad (11)$$

which we refer to as Node-Level Graph Neural Tangent Kernel (GNTK), or simply NTK of GNNs, to differentiate it with the graph-level GNTK for graph-level tasks initially proposed in Du et al. (2019b).

How GNNs evolve during training also follows (5). Ideally, one might want to show the precise mathematical characterization of (11) and its connection with $\mathbf{A}$, which however is impossible in principle due the complexity of optimizing non-linear neural networks. To make the analysis of non-linear models tractable, a previous work (Xu et al., 2021a) removes all activations such that the GNN boils down to a linear model. In comparison, we adopt an assumption that makes the analysis tractable in a way that still preserves model's non-linearity. Specifically, we study node-level GNTK in overparameterized regimes, where the model width m tends to infinity, and consequently, neural networks asymptotically converge to its first order Taylor expansion around its initialization (e.g. Jacot et al. (2018); Lee et al. (2019)), i.e. a kernel regression predictor. In this case, the node-level GNTK is a constant kernel $\mathbf{\Theta}^{(\ell)}$ (without subscript t). We next represent the explicit mathematical formula, which applies to arbitrary layer number ℓ, input features $\mathbf{X}$, graph structure $\mathcal{G}$, and various GNN architectures that conform to the definition in Section 2.

Similar to fully-connected neural networks, the NTK of a ℓ-layer GNN can be recursively computed based on the NTK of a $(\ell-1)$-layer GNNs. The layer-wise formula can be decomposed into two steps that respectively correspond to *Transformation* (i.e. $\mathbf{Z} \leftarrow \mathrm{ReLU}(\mathbf{ZW})$) and *Propagation* (i.e. $\mathbf{Z} \leftarrow \mathbf{AZ}$) in the GNN architecture. As additional elements in the computation, we denote a node-level GNTK for GNN without propagation at ℓ-th layer as $\bar{\mathbf{\Theta}}^{(\ell)}$, the covariance matrix of the ℓ-th layer's outputs with (and w/o) propagation as $\mathbf{\Sigma}^{(\ell)}$ (and $\bar{\mathbf{\Sigma}}^{(\ell)}$), and the covariance matrix of the derivative to the ℓ-th layer as $\dot{\mathbf{\Sigma}}^{(\ell)}$. Then the propagation and transformation steps in each layer respectively correspond to (we concisely show the key steps here and defer the complete formula to Appendix B.3):

$$(\textit{Transformation}) \quad \bar{\mathbf{\Theta}}^{(\ell)} = \mathbf{\Theta}^{(\ell-1)} \odot \dot{\mathbf{\Sigma}}^{(\ell)} + \bar{\mathbf{\Sigma}}^{(\ell)} \quad (12)$$

$$(\textit{Propagation}) \quad \begin{cases} \mathbf{\Sigma}^{(\ell)} = \mathbf{A}\, \bar{\mathbf{\Sigma}}^{(\ell)} \mathbf{A} \\ \mathbf{\Theta}^{(\ell)} = \mathbf{A}\, \bar{\mathbf{\Theta}}^{(\ell)} \mathbf{A}. \end{cases} \quad (13)$$

Implications. Compared with the computation for NTK of a fully-connected neural network (Jacot et al., 2018), the node-level GNTK has an equivalent transformation step, while its uniqueness stems from the propagation step, whereby the adjacency matrix $\mathbf{A}$ (or propagation matrix more generally) naturally integrates into the kernel similarity measure. Consequently, this kernel function inherently accommodates a graph implicit bias, and thus the gradient descent optimization of GNNs also tends to follow the trajectory regulated by the graph, similar to the behavior of the RP algorithm. To give more concrete examples, we provide case studies and show how the training dynamics of certain shallow GNNs, given fixed inputs (that are practically used), can be exactly described by the framework of generalized RP in (8).

We will further provide analysis for real-world finite width GNNs' NTK in Section 5, to empirically corroborate our insights. We believe similar results can be potentially obtained using more advanced techniques, e.g. (Bai & Lee, 2020), which we leave as future works.

4.2. Illustrative Examples

Given that our primary focus centers on the role of graphs in GNNs (as without them, GNNs are largely equivalent to MLPs), we exclude external node features and instead define inputs as either: 1) an identity matrix $\bar{\mathbf{X}} \triangleq \mathbf{I}_n$ that assigns each node a one-hot vector as indication of its unique identity (as sometimes assumed in practice (Kipf & Welling, 2017; Zhu et al., 2021)), which can be viewed as learning a unique embedding for each node by treating the first-layer weights as an embedding table; 2) fixed node embeddings from graph spectral decomposition $\bar{\mathbf{X}} \triangleq \arg\min_{\mathbf{B}} \|\mathbf{A} - \mathbf{B}\mathbf{B}^\top\|_F^2$, which aligns with various network embedding approaches based on definitions of $\mathbf{A}$ (Qiu et al., 2018).

Two-Layer GNN. Following the setup from prior work on fully-connected neural networks (Arora et al., 2019), we consider the training of two-layer GNNs where first-layer weights are optimized:

Theorem 4.2 (Two-Layer GNN)**.** *For an infinite width two-layer GNN defined as* $[\mathbf{F}, \mathbf{F}'] = \mathbf{A}\,\mathrm{ReLU}(\mathbf{A}\bar{\mathbf{X}}\mathbf{W}^{(1)})\mathbf{W}^{(2)}/\sqrt{m}$ *with* $\bar{\mathbf{X}} = \mathbf{I}_n$ *as inputs and standard NTK parameterization, its training dynamics by optimizing* $\mathbf{W}^{(1)}$ *can be written as a*

generalized RP process

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}(\mathbf{A}^2 \odot \mathbf{S})\mathbf{A}[\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t]\,,$$
$$\mathbf{S}_{ij} = \left(\pi - \arccos(\tfrac{\mathbf{A}_i^\top \mathbf{A}_j}{\|\mathbf{A}_i\|\|\mathbf{A}_j\|})\right)/2\pi. \qquad (14)$$

The matrix $\mathbf{S}$ *reweights each element in* $\mathbf{A}^2$ *by the similarity of neighborhood distributions of two nodes. For* $\bar{\mathbf{X}} = \arg\min_{\mathbf{B}} \|\mathbf{A} - \mathbf{B}\mathbf{B}^\top\|_F^2$*, the propagation matrix is replaced by* $\mathbf{A}(\mathbf{A}^3 \odot \tilde{\mathbf{S}})\mathbf{A}$ *where* $\tilde{\mathbf{S}}$ *is another reweighting matrix (details and proof in Appendix B.4).*

Deep and Wide GNNs. Pushing further, we can also characterize the evolution of arbitrarily deep GNNs where feature propagation is applied at the last layer (e.g. Klicpera et al. (2019); Liu et al. (2020); Spinelli et al. (2020); Chien et al. (2021)): , i.e. $f(\mathbf{X}; \mathbf{A}, \mathbf{W}) = \mathbf{A}^\ell \,\mathrm{MLP}(\mathbf{X}; \mathbf{W})$.

Theorem 4.3 (Deep and Wide GNN Dynamics)**.** *For arbitrarily deep and infinitely-wide GNNs with feature propagation deferred to the last layer, i.e.* $[\mathbf{F}, \mathbf{F}'] = \mathbf{A}^\ell \,\mathrm{MLP}(\bar{\mathbf{X}})$ *with* $\bar{\mathbf{X}} = \mathbf{I}_n$*, the training dynamics that result from optimizing MLP weights can be written as the generalized RP process* $\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] =$

$$-\eta \mathbf{A}^\ell(\mathbf{I}_n + c\mathbf{1}\mathbf{1}^\top)\mathbf{A}^\ell[\mathbf{R}_t, \mathbf{0}] \;+\; [\mathbf{R}_t, \mathbf{R}'_t]\,, \qquad (15)$$

where $c \geq 0$ *is a constant determined by the model depth, and* $\mathbf{1}$ *is an all-*1 *column vector.*

Linear GNNs. Another simple and interesting example is linear GNN (specifically SGC Wu et al. (2019)), whose training dynamics is equivalent to a special case of RP in (6):

Corollary 4.4 (One-Layer GNN)**.** *The training dynamics of the linear GNN* $[\mathbf{F}, \mathbf{F}'] = \mathbf{A}^\ell \bar{\mathbf{X}}\mathbf{W}$ *is identical to the basic version of RP in (6) with* $K = 2\ell$ *for input features* $\bar{\mathbf{X}} = \mathbf{I}_n$*, and* $K = 2\ell + 1$ *for* $\bar{\mathbf{X}} = \arg\min_{\mathbf{B}} \|\mathbf{A} - \mathbf{B}\mathbf{B}^\top\|_F^2$ *and positive semi-definite* $\mathbf{A}$*. (Proof in Appendix B.6)*

Remark. Despite this equivalence, it is important to note that this specific linear GNN (on our given input features) is not as lightweight as it may appear, since its parameter number scales with the size of dataset (even reaching orders of magnitude larger than deep GNN models), and the full-rank spectral decomposition of $\mathbf{A}$ is computationally very expensive (i.e. n^3) for large graphs. In contrast, RP efficiently yields identical results to this heavily parameterized GNN without actually training parameters or decomposing matrix. In practice, RP also consistently performs better than linear GNN, since real-world input feature is commonly a tall matrix (i.e. $n > d$), which makes linear models sub-optimal (see details in Appendix).

Note that there might exist many other examples where the NTK of GNNs is identical to special forms of adjacency. An exhaustive list of them is prohibitive. We refer readers to Appendix B.5 for analysis of another type of deep GNN where feature propagation is applied at the last layer (e.g. APPNP Klicpera et al. (2019)).

5. Generalization and Heterophily

In this section, we offer interpretable explanations of "when and why GNNs successfully generalize" and their pathological training behavior on heterophilic graphs. We also study the time evolution of real-world GNN NTKs to further empirically verify our theoretical results.

5.1. When and Why GNNs Generalize?

Our previous discussions have revolved around two matrices, namely the graph adjacency matrix $\mathbf{A}$ and the NTK matrix $\mathbf{\Theta}_t$.[1] To complete the theoretical picture, we introduce another matrix called the ideal *or optimal kernel matrix* (Cristianini et al., 2001), defined as $\mathbf{\Theta}^* \triangleq \bar{\mathbf{Y}}\bar{\mathbf{Y}}^\top \in \mathbb{R}^{n\times n}$ to indicate whether two instances have the same label, and a metric to quantify alignment of (kernel) matrices:

Definition 5.1 (Alignment, Cristianini et al. (2001))**.** Given two (kernel) matrices $\mathbf{K}_1$ and $\mathbf{K}_2$, their alignment is defined as $A(\mathbf{K}_1, \mathbf{K}_2) \triangleq \langle \mathbf{K}_1, \mathbf{K}_2\rangle_F / (\|\mathbf{K}_1\|_F \|\mathbf{K}_2\|_F) \in [0, 1]$. This is a generalization of cosine similarity from vectors to matrices, $\arccos$ of which satisfies the triangle inequality.

○ *Homophily Level*: $A(\mathbf{A}, \mathbf{\Theta}^*)$. The alignment between $\mathbf{A}$ and $\mathbf{\Theta}^*$ quantifies the *homophily level* of graph structure, i.e. whether two connected nodes indeed have the same label, and is determined the dataset. While many empirical results (e.g. Zhu et al. (2020)) suggest high homophily level is important for the performance of GNNs, deeper theoretical understandings are mostly lacking.

○ *Kernel-Target Alignment*: $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*)$. The alignment between kernel matrix and optimal $\mathbf{\Theta}^*$ has been widely studied and used as an objective for learning kernels functions (Cristianini et al., 2001; Kwok & Tsang, 2003; Lanckriet et al., 2004; Gönen & Alpaydın, 2011). Better kernel-target alignment has been recognized as a critical factor that leads to favorable generalization for classic non-parametric models. For intuition in our case, one can quickly verify that substituting $\mathbf{\Theta}^*$ to the training dynamics in (5) leads to perfect generalization performance (since ground-truth labels only propagate to unseen instances with the same label).

○ *Kernel-Graph Alignment*: $A(\mathbf{\Theta}_t, \mathbf{A})$. The alignment between NTK and graph is a novel notion in our work, as prior sections have shown that GNN NTK matrices naturally tend to align with $\mathbf{A}$. The RP algorithm (and variants thereof) serve as an extreme case with two identical matrices.

[1] We here refer to $\mathbf{A}$ as a class of similarity matrices based on original $\mathbf{A}$ in a general sense, such as $\mathbf{A}^K$ etc.

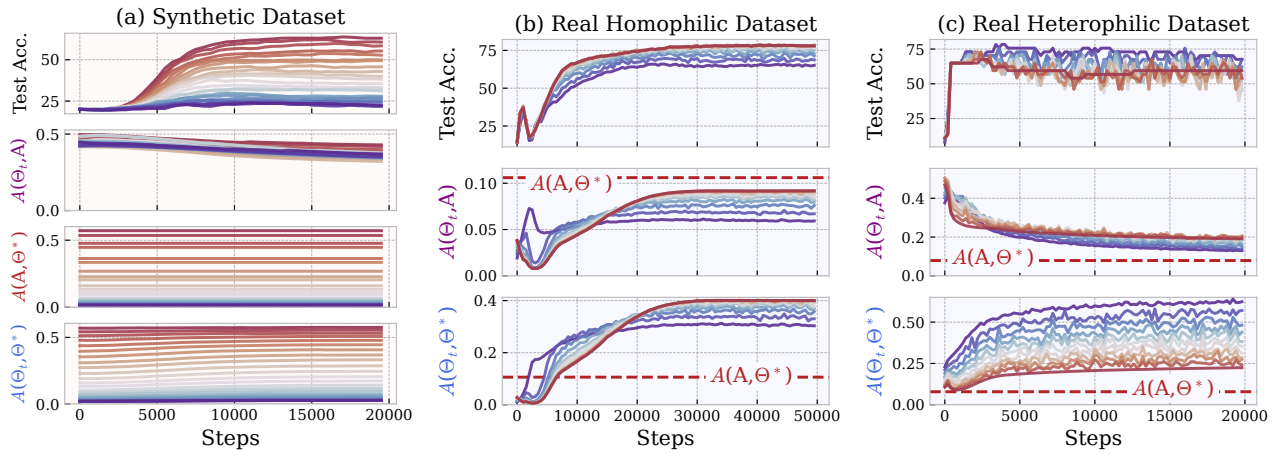


Figure 3: Evolution of NTK matrix $\mathbf{\Theta}_t$ of GCN during training, reflected by matrix alignment. **(a)** Synthetic dataset generated by a stochastic block model, where the homophily level gradually decreases by altering edge probabilities, i.e. homophilic $\rightarrow$ heterophilic; **(b & c)** Real-world homophilic (`Cora`) and heterophilic (`Texas`) datasets, where the graph is gradually coarsened until there is no edge left when evaluating $\mathbf{\Theta}_t$, i.e. more graph $\rightarrow$ less graph. (Details in Appendix D.2)

Implications. We consider two cases. For *homophilic* graphs where $A(\mathbf{A}, \mathbf{\Theta}^*) \uparrow$ is naturally large, better kernel-graph alignment $A(\mathbf{\Theta}_t, \mathbf{A}) \uparrow$ consequently leads to better kernel-target alignment $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*) \uparrow$. In other words, the NTK of GNNs naturally approaches the optimum as the graph structure possesses homophily property, and leveraging it in the optimization process (5) encourages training residuals to flow to unseen samples with the same label and thus better generalization; In contrast, for *heterophilic* graphs where $A(\mathbf{A}, \mathbf{\Theta}^*)$ is small, better kernel-graph alignment will hinder kernel-target alignment, explaining the pathological learning behavior of GNNs when dealing with heterophilic graphs in an interpretable manner.

5.2. Theoretical Results

To support this interpretation, we examine the generalization behavior of infinitely-wide neural networks in the extreme case where its NTK matrix is governed by the graph, say, $\lim_{k\to\infty} \sum_{i=0}^{k} (\alpha \mathbf{A})^i$ with $\alpha \in (0, 1)$ as adopted by the converged LP algorithm in (3). With a common assumption that training instances are drawn i.i.d. from a distribution $\mathcal{P}$, and based on the Rademacher complexity generalization bound for kernel regression (Bartlett & Mendelson, 2002; Arora et al., 2019; Du et al., 2019b), we have a label-dependent high-probability (at least $1 - \delta$) upper bound on population risk (derivation in Appendix B.7) $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}} \left[l\left(f(\boldsymbol{x}), y\right)\right] =$

$$\mathcal{O}\left(\sqrt{1 - c n_l^{-1} A(\mathbf{A}, \mathbf{\Theta}^*)} + \sqrt{n_l^{-1} \log(\delta^{-1})}\right), \quad (16)$$

where $c = \alpha \|\mathbf{\Theta}^*\|_F \|\mathbf{A}\|_F$ is a constant, $A(\mathbf{A}, \mathbf{\Theta}^*)$ is the homophily level for the training set (with slight abuse of notation). This bound can also be viewed as a theoretical guarantee for the converged (generalized) RP algorithm and clearly demonstrates that: *for $A(\mathbf{\Theta}_t, \mathbf{A})$ fixed as 1, higher level of graph homophily $A(\mathbf{A}, \mathbf{\Theta}^*)$ plays a dominant role in better generalization*. The above analysis can also be potentially extended to infinitely-wide GNNs discussed in Sec. 4 with further assumptions on the condition number of its NTK matrix, which will lead to similar (but less straightforward) results.

Pushing further, in a complementary setting to the above, where the objective is to find the optimal a priori kernel for directly minimizing the population risk (but without access to any input features or ground-truth labels), we demonstrate that when homophily assumptions are imposed on the graph, infinitely-wide GNNs and the RP algorithm will yield the optimal kernel regression predictor with the provably best generalization performance:

Theorem 5.2 (Bayesian Optimality of GNN)**.** *Assume that the underlying data generation distribution $\mathcal{P}$ is such that the probability of a pair of instances having the same label $P(y(\boldsymbol{x}_i) = y(\boldsymbol{x}_j))$ is proportional to $\mathbf{A}_{ij} - 1/2$. Then the optimal kernel regression predictor that minimizes the population risk with squared loss has kernel matrix $\mathbf{A}$.*

Remark. See proof in Appendix B.8. The matrix $\mathbf{A}$ from the above result could vary depending on different assumptions on $P(y(\boldsymbol{x}_i) = y(\boldsymbol{x}_j))$, such as $\mathbf{A}^K$ for the basic version of RP, or $\mathbf{A}\left(\mathbf{A}^2 \odot \mathbf{S}\right)\mathbf{A}$ for infinitely-wide two-layer GNNs in Theorem 4.2. And ideally, if one has privileged access to labels of all instances, this optimal matrix is exactly the optimal (posterior) kernel $\mathbf{\Theta}^*$ discussed above and perfect generalization will be achieved. This result further verifies our interpretation of generalization by showing that *for $A(\mathbf{A}, \mathbf{\Theta}^*)$ fixed to be large, better $A(\mathbf{\Theta}_t, \mathbf{A})$ (e.g. GNN*

and RP) leads to favorable generalization.

6. Empirical Verification

We next empirically verify whether real-world GNN NTKs indeed align with the graph during training and its effect on generalization. Figure 3 plots the results reflected by alignment between $\mathbf{\Theta}_t$, $\mathbf{A}$ and $\mathbf{\Theta}^*$.

Synthetic Dataset. To demonstrate the effects of different homophily levels, we use stochastic block models (Holland et al., 1983) to generate a synthetic dataset. As shown in Fig. 3(a), the kernel-graph alignment $A(\mathbf{\Theta}_t, \mathbf{A})$ for GNNs stays at a high level regardless of different graph structures, as a natural result of the network architecture (Sec. 4). Consequently, as we gradually alter edge probabilities to decrease the homophily level $A(\mathbf{A}, \mathbf{\Theta}^*)$, the kernel-target alignment $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*)$ also decreases and the testing accuracy drops dramatically (Sec. 5.1).

Real-World Datasets. On real-world homophilic and heterophilic datasets, we progressively coarsen the graph until the GNN degrades to an MLP, allowing us to analyze the effects of feature propagation in the model. For now, let us first focus on comparing red and blue lines in Fig. 3(b,c). We found the kernel-graph alignment $A(\mathbf{\Theta}_t, \mathbf{A})$ overall decreases with less graph structure, again verifying results in Sec. 4. However, the impact of the graph structure depends on different homophily levels $A(\mathbf{A}, \mathbf{\Theta}^*)$: on the homophilic dataset, more graph structure and better $A(\mathbf{\Theta}_t, \mathbf{A})$ optimize the NTK matrix as reflected by better kernel-target alignment $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*)$, but worsens it on the heterophilic one. This results in distinct generalization behavior of GNNs reflected by the testing accuracy.

Can and How (G)NNs Handle Heterophily? Recently, there has been a growing discussion on whether standard GNNs are capable of handling heterophily, with empirical results pointing to diverging conclusions (Zhu et al., 2020; Ma et al., 2022; Luan et al., 2022; Platonov et al., 2023). From the training dynamics perspective, we add new insights to this debate:

○ **1)** While we have shown that infinitely-wide GNNs are sub-optimal on heterophilic graphs, MLPs in contrast have no guarantee of kernel-target alignment (since both $A(\mathbf{A}, \mathbf{\Theta}^*)$ and $A(\mathbf{\Theta}_t, \mathbf{A})$ are not well-aligned), indicating that *they could either generalize better or worse without clearcut (dis)advantages, as opposed to the case on homophilic graphs where GNN is provably at an advantage*, which explains the diverse empirical results when comparing GNNs with MLPs in prior work.

○ **2)** As we now turn to the NTK's time evolution in Fig. 3, an additional phenomenon we found across all datasets is the overall increase of kernel-target alignment $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*)$ during the training process. This indicates that the training process enables real-world GNNs to adjust their NTK's feature space such that the kernel matrix leans towards an ultimately homoplilic graph structure (i.e. $\mathbf{\Theta}^*$) to adapt to heterophilic datasets (and consequently different evolutionary trends of $A(\mathbf{\Theta}_t, \mathbf{A})$ for hemophiliac and heterophilic datasets). Such a phenomenon has also been found for other models in vision tasks (Baratin et al., 2021).

Additionally, since our analysis also applies to other forms of $\mathbf{A}$ for feature propagation in GNNs, it could also potentially explain how some specialized models with different instantiations of $\mathbf{A}$ (e.g. signed propagation) can mitigate the heterophily issue. We also refer readers to Appendix C.4 for a discussion of how our insights can be potentially used to guide designing new GNN architectures for handling heterophily.

7. Discussions

Abridged Related Work. Most existing work on theoretical aspects of GNNs focuses on representation and generalization (see Jegelka (2022) and references therein) while their optimization properties remains underexplored (Zhang et al., 2020; Xu et al., 2021a; Yadati, 2022; Huang et al., 2023). Specifically, existing work in representation (or expressiveness) does not provide answers to what exactly GNN functions are found during the optimization process. For generalization, prior work is insufficient to explain the effects of training, which is widely-recognized as a crucial ingredient, and does not connect to heterophily, a relevant aspect in practice. (See a comprehensive discussion of related work in Appendix A)

Applicability and Future Directions. Our insights apply to both transductive and inductive settings (Appendix C.1), other loss functions (Appendix C.2), multi-dimensional outputs (Appendix C.3), different GNN (or graph Transformer) architectures that conform to our definition. The analytical framework could also be extended to other tasks (which are left as future work); for instances, for graph classification or regression, the definition of GNTK in (5) should be modified to the graph-level one (Du et al., 2019b); for link prediction and self-supervised learning, the loss function in the derivation of (5) should be adjusted accordingly. A similar message passing process in function space would still hold in these settings. The framework could also be potentially adapted to analyze other GNN issues, such as over-smoothing and over-squashing, though specific considerations may be required. While this paper focuses specifically on studying the role of graph structures in optimization, the framework can potentially also be used to study the effects of different model architectures by noting that the definition of $\mathbf{A}$ in the paper is associated with GNN architectures (2).

Acknowledgements

The authors from SJTU were in part supported by NSFC (92370201, 62222607, 72342023). The author from CUHKSZ was supported by NSFC (12326608), the National Key Research and Development Project under grant 2022YFA1003900 and Hetao Shenzhen-Hong Kong Science and Technology Innovation Cooperation Zone Project (No.HZQSWS-KCCYB-2024016).

Impact Statement

Theoretically, our analytical framework from the training dynamics perspective might have broader impacts beyond graph learning as it exemplifies using kernel learning to justify model architectures. And practically, the proposed RP also unlocks the possibility of leveraging sparse structures in training dynamics to develop efficient algorithms. Similar to other works in machine learning, this work may inevitably have negative society impacts if machine learning were to be employed for unethical purposes.

References

Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332. PMLR, 2019.

Bai, Y. and Lee, J. D. Beyond linearization: On quadratic and higher-order approximation of wide neural networks. *International Conference on Learning Representations*, 2020.

Baranwal, A., Fountoulakis, K., and Jagannath, A. Graph convolution for semi-supervised classification: Improved linear separability and out-of-distribution generalization. *arXiv preprint arXiv:2102.06966*, 2021.

Baratin, A., George, T., Laurent, C., Hjelm, R. D., Lajoie, G., Vincent, P., and Lacoste-Julien, S. Implicit regularization via neural feature alignment. In *International Conference on Artificial Intelligence and Statistics*, pp. 2269–2277. PMLR, 2021.

Bartlett, P. L. and Mendelson, S. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, 2021.

Boyd, S. P. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., and Gu, Q. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

Chapelle, O., Scholkopf, B., and Zien, A. Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3): 542–542, 2009.

Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. *International Conference on Learning Representations*, 2021.

Cong, W., Ramezani, M., and Mahdavi, M. On provable benefits of depth in training graph convolutional networks. *Advances in Neural Information Processing Systems*, 34: 9936–9949, 2021.

Cortes, C., Mohri, M., and Rostamizadeh, A. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

Cristianini, N., Shawe-Taylor, J., Elisseeff, A., and Kandola, J. On kernel-target alignment. *Advances in neural information processing systems*, 14, 2001.

Dehmamy, N., Barabási, A.-L., and Yu, R. Understanding the representation power of graph neural networks in learning graph topology. *Advances in Neural Information Processing Systems*, 32, 2019.

Du, L., Shi, X., Fu, Q., Ma, X., Liu, H., Han, S., and Zhang, D. Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In *Proceedings of the ACM Web Conference 2022*, pp. 1550–1558, 2022.

Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pp. 1675–1685. PMLR, 2019a.

Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019b.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. *International Conference on Learning Representations*, 2019c.

Frei, S., Cao, Y., and Gu, Q. Agnostic learning of a single neuron with gradient descent. *Advances in Neural Information Processing Systems*, 33:5417–5428, 2020.

Garg, V., Jegelka, S., and Jaakkola, T. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pp. 3419–3430. PMLR, 2020.

Gatterbauer, W., Günnemann, S., Koutra, D., and Faloutsos, C. Linearized and single-pass belief propagation. *Proceedings of the VLDB Endowment*, 8(5):581–592, 2015.

Golikov, E., Pokonechnyy, E., and Korviakov, V. Neural tangent kernel: A survey. *arXiv preprint arXiv:2208.13614*, 2022.

Gönen, M. and Alpaydın, E. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12: 2211–2268, 2011.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

He, H. and Su, W. J. The local elasticity of neural networks. *International Conference on Learning Representations*, 2020.

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Huang, W., Cao, Y., Wang, H., Cao, X., and Suzuki, T. Graph neural networks provably benefit from structural information: A feature learning perspective. *arXiv preprint arXiv:2306.13926*, 2023.

Iscen, A., Tolias, G., Avrithis, Y., and Chum, O. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5070–5079, 2019.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jegelka, S. Theory of graph neural networks: Representation and learning. *arXiv preprint arXiv:2204.07697*, 2022.

Jia, J. and Benson, A. R. A unifying generative model for graph learning algorithms: Label propagation, graph convolutions, and combinations. *SIAM Journal on Mathematics of Data Science*, 4(1):100–125, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *International Conference on Learning Representations*, 2019.

Koutra, D., Ke, T.-Y., Kang, U., Chau, D. H., Pao, H.-K. K., and Faloutsos, C. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II 22*, pp. 245–260. Springer, 2011.

Kwok, J. T. and Tsang, I. W. Learning with idealized kernels. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 400–407, 2003.

Lanckriet, G. R., Cristianini, N., Bartlett, P., Ghaoui, L. E., and Jordan, M. I. Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72, 2004.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lee, M.-C., Shekhar, S., Yoo, J., and Faloutsos, C. Ultraprop: Principled and explainable propagation on large graphs. *arXiv preprint arXiv:2301.00270*, 2022.

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pp. 13242–13256. PMLR, 2022.

Liao, R., Urtasun, R., and Zemel, R. A pac-bayesian approach to generalization bounds for graph neural networks. *arXiv preprint arXiv:2012.07690*, 2020.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020.

Loukas, A. How hard is to distinguish graphs with graph neural networks? *Advances in neural information processing systems*, 33:3465–3476, 2020.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting heterophily for graph neural networks. *Advances in neural information processing systems*, 35:1362–1375, 2022.

Ma, J., Deng, J., and Mei, Q. Subgroup generalization and fairness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:1048–1061, 2021.

Ma, Y., Liu, X., Shah, N., and Tang, J. Is homophily a necessity for graph neural networks? *International Conference on Learning Representations*, 2022.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Maurya, S. K., Liu, X., and Murata, T. Simplifying approach to node classification in graph neural networks. *Journal of Computational Science*, 62:101695, 2022.

Mei, S., Misiakiewicz, T., and Montanari, A. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. In *Conference on Learning Theory*, pp. 2388–2464. PMLR, 2019.

Mohamadi, M. A. and Sutherland, D. J. A fast, well-founded approximation to the empirical neural tangent kernel. *International Conference on Machine Learning*, 2023.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Papyan, V., Han, X., and Donoho, D. L. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of gnns under heterophily: are we really making progress? *International Conference on Learning Representations*, 2023.

Pukdee, R., Sam, D., Balcan, M.-F., and Ravikumar, P. Label propagation with weak supervision. *International Conference on Learning Representations*, 2023.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pp. 459–467, 2018.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Scarselli, F., Tsoi, A. C., and Hagenbuchner, M. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018.

Shawe-Taylor, J. and Cristianini, N. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Spinelli, I., Scardapane, S., and Uncini, A. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4755–4760, 2020.

Szummer, M. and Jaakkola, T. Partially labeled classification with markov random walks. *Advances in neural information processing systems*, 14, 2001.

Verma, S. and Zhang, Z.-L. Stability and generalization of graph convolutional neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1539–1548, 2019.

Wang, H. and Leskovec, J. Combining graph convolutional neural networks and label propagation. *ACM Transactions on Information Systems (TOIS)*, 40(4):1–27, 2021.

Wang, X. and Zhang, M. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pp. 23341–23362. PMLR, 2022.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pp. 6861–6871. PMLR, 2019.

Wu, Q., Zhang, H., Yan, J., and Wipf, D. Handling distribution shifts on graphs: An invariance perspective. *International Conference on Learning Representations*, 2022.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

Xu, K., Zhang, M., Jegelka, S., and Kawaguchi, K. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *International Conference on Machine Learning*, pp. 11592–11602. PMLR, 2021a.

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. *International Conference on Learning Representations*, 2021b.

Yadati, N. A convex formulation for graph convolutional training: Two layer case. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1281–1286. IEEE, 2022.

Yamaguchi, Y., Faloutsos, C., and Kitagawa, H. Camlp: Confidence-aware modulated label propagation. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 513–521. SIAM, 2016.

Yang, C., Wu, Q., Wang, J., and Yan, J. Graph neural networks are inherently good generalizers: Insights by bridging gnns and mlps. *International Conference on Learning Representations*, 2023.

Yehudai, G., Fetaya, E., Meirom, E., Chechik, G., and Maron, H. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pp. 11975–11986. PMLR, 2021.

Zhang, S., Wang, M., Liu, S., Chen, P.-Y., and Xiong, J. Fast learning of graph neural networks with guaranteed generalizability: one-hidden-layer case. In *International Conference on Machine Learning*, pp. 11268–11277. PMLR, 2020.

Zheng, X., Liu, Y., Pan, S., Zhang, M., Jin, D., and Yu, P. S. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

Zhou, D., Bousquet, O., Lal, T., Weston, J., and Schölkopf, B. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

Zhou, X. and Wang, H. The generalization error of graph convolutional networks may enlarge with more layers. *Neurocomputing*, 424:97–106, 2021.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems*, 33:7793–7804, 2020.

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily. In *Proceedings of the AAAI conference on artificial intelligence*, 2021.

Zhu, X. and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation. *ProQuest Number: INFORMATION TO ALL USERS*, 2002.

Zhu, X., Ghahramani, Z., and Lafferty, J. D. Semi-supervised learning using gaussian fields and harmonic functions. In *International conference on Machine learning*, pp. 912–919, 2003.

A. Unabridged Related Work

This section discusses more related works that are not covered in the main text and those related works that are already covered but in greater depth.

A.1. Optimization and Training Dynamics

Towards deeper understanding of the success and limitation of GNNs, many works focus on the representation power of GNNs (Maron et al., 2019; Xu et al., 2019; Oono & Suzuki, 2019; Chen et al., 2019; Dehmamy et al., 2019; Sato et al., 2019; Loukas, 2020). While these works formalize what functions a GNN can possibly represent, they do not provide answers to which specific GNN function will be found during optimization process or whether the learned GNN function will successfully generalize. In contrast, theoretical understandings of the optimization properties of GNNs are scarce. Specifically, Zhang et al. (2020) prove the global convergence of a one-layer GNN with assumptions on the training algorithm built on tensor initialization and accelerated gradient descent; Xu et al. (2021a) analyze the convergence rate of linearized GNNs with focus on training dynamics in weight space and empirical risk; Yadati (2022) analyze optimization properties of a two-layer GCN by introducing a convex program; Huang et al. (2023) characterize signal learning and noise memorization in two-layer GCN and compare it to CNNs. However, none of existing studies study the training dynamics of GNNs in function space, which is our focus and could lead to a wealth of new insights that are theoretically and practically valuable.

A.2. In- and Out-of-Distribution Generalization

Existing works in generalization of GNNs focus on their in-distribution (Scarselli et al., 2018; Verma & Zhang, 2019; Du et al., 2019b; Liao et al., 2020) and out-of-distribution generalization properties (Yehudai et al., 2021; Xu et al., 2021b; Wu et al., 2022; Yang et al., 2023). For *node-level tasks* which are challenging due to the dependency between samples, most prior art study the generalization bound of GNNs based on complexity of model class (Scarselli et al., 2018; Baranwal et al., 2021; Garg et al., 2020; Ma et al., 2021) or algorithmic stability (Verma & Zhang, 2019; Zhou & Wang, 2021; Cong et al., 2021). The former line of works do not consider the optimization, which however is a critical ingredient of finding generalizable solutions; the latter line of works consider the training algorithm, but their bounds as the number of epochs becomes large. Moreover, there is no existing work (to the best of our knowledge) formally connecting generalization and heterophily, though their connections are tacitly implied in many empirical results, e.g. (Zhu et al., 2020; 2021; Zheng et al., 2022). The most related work in GNN generalization is (Yang et al., 2023) wherein the authors found vanilla MLPs with test-time message passing operations can be as competitive as different GNN counterparts, and then analyzed generalization of GNNs using node-level GNTK with attention to feature-wise extrapolation. Our results agree with and complement (Yang et al., 2023) by noting that our analysis is applicable for all inductive, transductive and training without graph settings (cf. Appendix C.1).

A.3. Graph-Based Semi-Supervised Learning (Label Propagation)

One of most popular type of semi-supervised learning methods is graph-based methods (Chapelle et al., 2009), where label propagation (Szummer & Jaakkola, 2001; Zhu & Ghahramani, 2002; Zhu et al., 2003; Zhou et al., 2003; Chapelle et al., 2009; Koutra et al., 2011; Gatterbauer et al., 2015; Yamaguchi et al., 2016; Iscen et al., 2019) is one of the most classic and widely-used algorithms. The algorithm relies on external data structure, usually represented by a graph adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, to propagate ground-truth labels of labeled samples to infer unlabeled ones, and it is still under active research nowadays, e.g. (Pukdee et al., 2023; Lee et al., 2022), mainly due to its efficiency and scalability. The label propagation algorithm can usually be induced from minimization of a quadratic objective (Zhou et al., 2003):

$$\mathcal{E} = \|\mathbf{F} - \mathbf{Y}\|^2 + \lambda \operatorname{Tr}\left[[\mathbf{F}, \mathbf{F}']^\top (\mathbf{I}_n - \mathbf{A}) [\mathbf{F}, \mathbf{F}']\right], \tag{17}$$

motivated from the Dirichlet energy to enforce smoothness of predictions according to sample relations. Notice the minima of this objective does not minimize the squared loss part $\|\mathbf{F} - \mathbf{Y}\|^2$ due to the regularization of the second term. In contrast, the proposed RP algorithm has similar smoothness effects but minimizes the squared loss. Moreover, some previous works attempt to explore the interconnections between LP and GNNs from different perspectives such as feature/label influence (Wang & Leskovec, 2021), generative model (Jia & Benson, 2022). Compared with them, we first show the exact equivalence of classification results between LP and the first step training of GNN (on node embeddings and with squared loss).

B. Theoretical Results

B.1. Derivation of (4) and (5): Training Dynamics of General Parameterized Model

Recall that for supervised learning, one is interested in minimizing the squared error $\mathcal{L}$ using *gradient descent (GD)*,

$$\mathcal{L} = \frac{1}{2}\|\mathbf{F}_t - \mathbf{Y}\|^2, \quad \frac{\partial \mathbf{W}_t}{\partial t} = -\eta \nabla_{\mathbf{W}} \mathcal{L}. \tag{18}$$

Let $\mathbf{R}_t = \mathbf{Y} - \mathbf{F}_t$ and $\mathbf{R}'_t = \mathbf{Y}' - \mathbf{F}'_t$ denote residuals for the labeled and unlabeled sets.

For the training set, we have

$$\begin{aligned}
& \frac{\partial \mathbf{F}_t}{\partial t} && && (19)\\
= \; & \frac{\partial \mathbf{F}_t}{\partial \mathbf{W}_t} \frac{\partial \mathbf{W}_t}{\partial t} && \textit{(Chain rule)} && (20)\\
= \; & -\eta \frac{\partial \mathbf{F}_t}{\partial \mathbf{W}_t} \nabla_{\mathbf{W}} \mathcal{L} && \textit{(GD training)} && (21)\\
= \; & -\eta \frac{\partial \mathbf{F}_t}{\partial \mathbf{W}_t} \nabla_{\mathbf{W}} \mathbf{F}_t \nabla_{\mathbf{F}_t} \mathcal{L} && \textit{(Chain rule)} && (22)\\
= \; & -\eta \nabla_{\mathbf{W}} \mathbf{F}_t{}^{\top} \nabla_{\mathbf{W}} \mathbf{F}_t \nabla_{\mathbf{F}_t} \mathcal{L} && \textit{(Change of notation)} && (23)\\
= \; & -\eta \, \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}) \nabla_{\mathbf{F}_t} \mathcal{L} && \textit{(Neural tangent kernel)} && (24)\\
= \; & \eta \, \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}) \, \mathbf{R}_t && \textit{(Loss function)} && (25)
\end{aligned}$$

Incorporating the test set, we have

$$\begin{aligned}
& \frac{\partial [\mathbf{R}_t, \mathbf{R}'_t]}{\partial t} && && (26)\\
= \; & \frac{\partial [\mathbf{R}_t, \mathbf{R}'_t]}{\partial \mathbf{W}_t} \frac{\partial \mathbf{W}_t}{\partial t} && \textit{(Chain rule)} && (27)\\
= \; & -\eta \frac{\partial [\mathbf{R}_t, \mathbf{R}'_t]}{\partial \mathbf{W}_t} \nabla_{\mathbf{W}} \mathcal{L} && \textit{(GD training)} && (28)\\
= \; & -\eta \frac{\partial [\mathbf{R}_t, \mathbf{R}'_t]}{\partial \mathbf{W}_t} \nabla_{\mathbf{W}} [\mathbf{F}_t, \mathbf{F}'_t] \nabla_{[\mathbf{F}_t, \mathbf{F}'_t]} \mathcal{L} && \textit{(Chain rule)} && (29)\\
= \; & \eta \nabla_{\mathbf{W}} [\mathbf{F}_t, \mathbf{F}'_t]^{\top} \nabla_{\mathbf{W}} [\mathbf{F}_t, \mathbf{F}'_t] \nabla_{[\mathbf{F}_t, \mathbf{F}'_t]} \mathcal{L} && \textit{(Change of notation)} && (30)\\
= \; & \eta \, \mathbf{\Theta}_t([\mathbf{X}, \mathbf{X}'], [\mathbf{X}, \mathbf{X}']) \nabla_{[\mathbf{F}_t, \mathbf{F}'_t]} \mathcal{L} && \textit{(Neural tangent kernel)} && (31)\\
= \; & \eta \, \mathbf{\Theta}_t([\mathbf{X}, \mathbf{X}'], [\mathbf{X}, \mathbf{X}']) \nabla_{[\mathbf{F}_t, \mathbf{F}'_t]} \frac{1}{2}\|\mathbf{F}_t - \mathbf{Y}\|_F^2 && \textit{(Loss function)} && (32)\\
= \; & -\eta \, \mathbf{\Theta}_t([\mathbf{X}, \mathbf{X}'], [\mathbf{X}, \mathbf{X}']) \, [\mathbf{R}_t, \mathbf{0}] && \textit{(Compute gradient)} && (33)
\end{aligned}$$

where

$$\mathbf{\Theta}_t([\mathbf{X}, \mathbf{X}'], [\mathbf{X}, \mathbf{X}']) \triangleq \begin{bmatrix} \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}) & \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}') \\ \mathbf{\Theta}_t(\mathbf{X}', \mathbf{X}) & \mathbf{\Theta}_t(\mathbf{X}', \mathbf{X}') \end{bmatrix} \tag{34}$$

is the NTK matrix in $\mathbb{R}^{n \times n}$. Discretizing the residual dynamics with step size $\Delta t = 1$ and rearranging the equation gives

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \begin{bmatrix} \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}) & \mathbf{\Theta}_t(\mathbf{X}, \mathbf{X}') \\ \mathbf{\Theta}_t(\mathbf{X}', \mathbf{X}) & \mathbf{\Theta}_t(\mathbf{X}', \mathbf{X}') \end{bmatrix} [\mathbf{R}_t, \mathbf{0}] \; + \; [\mathbf{R}_t, \mathbf{R}'_t] \, . \tag{35}$$

or

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \, \mathbf{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}}) [\mathbf{R}_t, \mathbf{0}] \; + \; [\mathbf{R}_t, \mathbf{R}'_t] \, . \tag{36}$$

B.2. Proof of Theorem 3.2: Convergence of RP, Connection with Kernel Regression

In this proof we examine generic RP iterations of the form given by

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{S}[\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t], \tag{37}$$

where $\mathbf{S}$ is an arbitrary symmetric matrix as similarity measure ($\mathbf{S} = \mathbf{A}^K$ for the basic version of RP in (6)) with block structure aligned with the dimensions of $\mathbf{R}_t$ and $\mathbf{R}'_t$ respectively

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{\mathbf{XX}} & \mathbf{S}_{\mathbf{XX}'} \\ \mathbf{S}_{\mathbf{X}'\mathbf{X}} & \mathbf{S}_{\mathbf{X}'\mathbf{X}'} \end{bmatrix}, \tag{38}$$

where $\mathbf{S}_{\mathbf{XX}} \in \mathbb{R}^{n_l \times n_l}$ is a principal submatrix of $\mathbf{S} \in \mathbb{R}^{n \times n}$ corresponding to the training set. According to (37), the explicit form of training residuals can be written as

$$\begin{aligned} \mathbf{R}_{t+1} &= (\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}}) \mathbf{R}_t \\ &= (\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}})^{t+1} \mathbf{Y} \end{aligned} \tag{39}$$

and testing residuals can be written as

$$\begin{aligned} \mathbf{R}'_{t+1} &= -\eta \mathbf{S}_{\mathbf{X}'\mathbf{X}} \mathbf{R}_t + \mathbf{R}'_t = -\eta \sum_{i=0}^{t} \mathbf{S}_{\mathbf{X}'\mathbf{X}} \mathbf{R}_i \\ &= -\eta \mathbf{S}_{\mathbf{X}'\mathbf{X}} \sum_{i=0}^{t} (\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}})^i \mathbf{Y} \end{aligned} \tag{40}$$

To analyze their convergence, we consider three variants based on the property of $\mathbf{S}_{\mathbf{XX}}$: 1) $\mathbf{S}_{\mathbf{XX}}$ is positive definite; 2) $\mathbf{S}_{\mathbf{XX}}$ is positive semi-definite (but not positive definite); 3) $\mathbf{S}_{\mathbf{XX}}$ is not positive semi-definite.

For the first variant, we stipulate $\mathbf{S}_{\mathbf{XX}}$ is positive definite. In this case, sufficiently small step size $\eta < 2/\sigma_{max}[\mathbf{S}_{\mathbf{XX}}]$, where $\sigma_{max}[\mathbf{S}_{\mathbf{XX}}]$ is the largest eigenvalue of $\mathbf{S}_{\mathbf{XX}}$, ensures each diagonal element in $\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}}$ to lie between $(-1, 1)$. Therefore, as $t \to \infty$, the power and geometric series of $\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}}$ converge to

$$\begin{aligned} &(\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}})^{t+1} \to \mathbf{Q}(\mathbf{I}_{n_l} - \eta \mathbf{\Lambda}[\mathbf{S}_{\mathbf{XX}}])^{\infty} \mathbf{Q}^{-1} = \mathbf{0}, \\ &\sum_{i=0}^{t} (\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}})^i \to (\mathbf{I}_{n_l} - (\mathbf{I}_{n_l} - \eta \mathbf{S}_{\mathbf{XX}}))^{-1} = (\eta \mathbf{S}_{\mathbf{XX}})^{-1}. \end{aligned} \tag{41}$$

Plugging them back into (39) and (40) gives us

$$\mathbf{R}_t \to \mathbf{0}, \quad \mathbf{R}'_t \to -\mathbf{S}_{\mathbf{X}'\mathbf{X}} \mathbf{S}_{\mathbf{XX}}^{-1} \mathbf{Y}. \tag{42}$$

Correspondingly, by noting that $\mathbf{Y}' = \mathbf{0}$ at initialization, we have

$$\mathbf{F}_t \to \mathbf{Y}, \quad \mathbf{F}'_t \to \mathbf{S}_{\mathbf{X}'\mathbf{X}} \mathbf{S}_{\mathbf{XX}}^{-1} \mathbf{Y}. \tag{43}$$

Namely, for positive definite $\mathbf{S}_{\mathbf{XX}}$, the converged model predictions $\mathbf{F}_\infty$ perfectly fit the training labels $\mathbf{Y}$, and we further assume that $\mathbf{S}$ is positive definite, (43) is equivalent to the solution of kernel regression with respect to the kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \mathbf{A}^K_{\boldsymbol{x}\boldsymbol{x}'}$.

For the second variant, we stipulate that $\mathbf{S}_{\mathbf{XX}}$ is positive semi-definite (but not positive definite), which implies that $\mathbf{S}_{\mathbf{XX}} = \mathbf{B}\mathbf{B}^\top$ for some tall matrix $\mathbf{B} \in \mathbb{R}^{n_l \times r}$ that is full column rank. Suppose $\mathbf{P}_{C(\mathbf{B})} = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top$ and $\mathbf{P}_{N(\mathbf{B}^\top)} = \mathbf{I}_{n_l} - \mathbf{P}_{C(\mathbf{B})}$ denotes the projection matrices onto the column space of $\mathbf{B}$ and null space of $\mathbf{B}^\top$. By their construction,

$$\mathbf{P}_{C(\mathbf{B})} \mathbf{B} = \mathbf{B}, \quad \mathbf{P}_{N(\mathbf{B}^\top)} \mathbf{B} = \mathbf{0}. \tag{44}$$

For training residuals, we may form the decomposition

$$\begin{aligned} \mathbf{R}_{t+1} &= (\mathbf{I}_{n_l} - \eta \mathbf{B}\mathbf{B}^\top)^{t+1} \mathbf{Y} \\ &= (\mathbf{I}_{n_l} - \eta \mathbf{B}\mathbf{B}^\top)^{t+1} \mathbf{P}_{C(\mathbf{B})} \mathbf{Y} + (\mathbf{I}_{n_l} - \eta \mathbf{B}\mathbf{B}^\top)^{t+1} \mathbf{P}_{N(\mathbf{B}^\top)} \mathbf{Y} \end{aligned} \tag{45}$$

Table 2: Summary of the convergence of residual propagation. The convergence of model predictions can be inferred by $\mathbf{F}_t = \mathbf{Y} - \mathbf{R}_t$ and $\mathbf{F}'_t = -\mathbf{R}'_t$.

$\mathbf{S_{XX}}$	$\mathbf{S}$	Convergence of $\mathbf{R}_t$	Convergence of $\mathbf{R}'_t$	Counterpart
PD	PSD	$\mathbf{0}$	$-\mathbf{S_{X'X}}\mathbf{S_{XX}^{-1}}\mathbf{Y}$	Kernel Regression
PD	not PSD	$\mathbf{0}$	$-\mathbf{S_{X'X}}\mathbf{S_{XX}^{-1}}\mathbf{Y}$	Unique
PSD (but not PD)	PSD	$\left(\mathbf{I}_{n_l} - \mathbf{B}(\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top\right)\mathbf{Y}$	$-\mathbf{B}'(\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{Y}$	Linear Regression
PSD (but not PD)	not PSD	$\left(\mathbf{I}_{n_l} - \mathbf{B}(\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top\right)\mathbf{Y}$	Not Converge	Unique
not PSD	not PSD	Not Converge	Not Converge	Unique

It follows that, for the first term on the RHS of (45),

$$\begin{aligned}
(\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top)^{t+1}\mathbf{P}_{C(\mathbf{B})}\mathbf{Y} &= (\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top)^{t+1}\mathbf{B}\mathbf{B}^\dagger\mathbf{Y} \\
&= (\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top)^{t}(\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top)\mathbf{B}\mathbf{B}^\dagger\mathbf{Y} \\
&= (\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top)^{t}\mathbf{B}(\mathbf{I}_r - \eta\mathbf{B}^\top\mathbf{B})\mathbf{B}^\dagger\mathbf{Y} \\
&= \mathbf{B}(\mathbf{I}_r - \eta\mathbf{B}^\top\mathbf{B})^{t+1}\mathbf{B}^\dagger\mathbf{Y} \\
&\to \mathbf{0}, \qquad (46)
\end{aligned}$$

where $\mathbf{B}^\dagger = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top$ is the pseudo inverse of $\mathbf{B}$. The convergence also requires $\eta < 2/\sigma_{max}[\mathbf{S_{XX}}]$.

For the second term on the RHS of (45), we have

$$\begin{aligned}
(\mathbf{I} - \eta\mathbf{B}\mathbf{B}^\top)^{t+1}\mathbf{P}_{N(\mathbf{B}^\top)}\mathbf{Y} &= (\mathbf{I} - \eta\mathbf{B}\mathbf{B}^\top)^{t}(\mathbf{P}_{N(\mathbf{B}^\top)} - \eta\mathbf{B}(\mathbf{P}_{N(\mathbf{B}^\top)}\mathbf{B})^\top)\mathbf{Y} \\
&= (\mathbf{I} - \eta\mathbf{B}\mathbf{B}^\top)^{t}\mathbf{P}_{N(\mathbf{B}^\top)}\mathbf{Y} \\
&= \mathbf{P}_{N(\mathbf{B}^\top)}\mathbf{Y} \qquad (47)
\end{aligned}$$

It follows that

$$\mathbf{R}_t \to \mathbf{P}_{N(\mathbf{B}^\top)}\mathbf{Y} = \left(\mathbf{I}_{n_l} - \mathbf{B}(\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top\right)\mathbf{Y}, \qquad (48)$$

which is equivalent to optimal training residuals of linear regression (Boyd & Vandenberghe, 2004).

However, in our case, the testing residuals will not necessarily converge for arbitrary $\mathbf{S_{X'X}}$ since the geometric series in (40) diverges outside $(-1, 1)$. Nevertheless, we can further stipulate $\mathbf{S}$ is also positive semi-definite. In this case, we have $\mathbf{S_{X'X}} = \mathbf{B}'\mathbf{B}^\top$ for another tall matrix $\mathbf{B}' \in \mathbb{R}^{n_u \times r}$. Correspondingly, the testing residuals can be written as

$$\begin{aligned}
\mathbf{R}'_{t+1} &= -\eta\mathbf{B}'\mathbf{B}^\top\sum_{i=0}^{t}\left(\mathbf{I}_{n_l} - \eta\mathbf{B}\mathbf{B}^\top\right)^i\mathbf{Y} \\
&= -\eta\mathbf{B}'\sum_{i=0}^{t}\left(\mathbf{I}_r - \eta\mathbf{B}^\top\mathbf{B}\right)^i\mathbf{B}^\top\mathbf{Y} \\
&= -\eta\mathbf{B}'\sum_{i=0}^{t}\left(\mathbf{I}_r - \eta\mathbf{B}^\top\mathbf{B}\right)^i\mathbf{B}^\top\mathbf{Y} \\
&\to -\eta\mathbf{B}'(\mathbf{I}_r - (\mathbf{I}_r - \eta\mathbf{B}^\top\mathbf{B}))^{-1}\mathbf{B}^\top\mathbf{Y} \\
&= -\mathbf{B}'(\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{Y}. \qquad (49)
\end{aligned}$$

For the last variant, if $\mathbf{S_{XX}}$ is not positive semi-definite, it is no longer possible to guarantee convergence from arbitrary initializations. Rather we can only establish that solutions of the form described above can serve as fixed points of the iterations.

To summarize, for the RP algorithm in (6) where $\mathbf{S} = \mathbf{A}^K$, if step size is sufficiently small $\eta < 2\sigma_{max}^{-1}[\mathbf{A}^K_{\mathbf{XX}}]$, both $\mathbf{R}_t$ and $\mathbf{R}'_t$ converge as $t \to \infty$ for positive definite $\mathbf{A}^K_{\mathbf{XX}}$ or positive semi-definite $\mathbf{A}^K$. In practice, one can choose K as an even

number or letting $\mathbf{A} \leftarrow \alpha\mathbf{A} + (1-\alpha)\mathbf{I}_n$ for $\alpha >= \frac{1}{2}$ to enforce positive semi-definiteness of $\mathbf{A}^K$. However, in practice, we did not find that test performance was compromised when the algorithm does not converge. Table 2 gives a clear overview of the convergence of RP and connection with existing learning algorithms.

B.3. Computation of Node-Level GNTK

We present the recurrent formula for computing of GNTK in node-level tasks. Suppose the GNN is denoted as $f(\boldsymbol{x}_i; \mathbf{A}) \in \mathbb{R}$ where the input is node feature $\boldsymbol{x}_i$, the graph structure $\mathbf{A}$ is used for cross-instance feature propagation at each layer, weights are $\mathbf{W}$. The GNTK in node-level tasks is defined as

$$\mathbf{\Theta}_t(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) = \left\langle \frac{\partial f_t(\boldsymbol{x}_i; \mathbf{A})}{\partial \mathbf{W}_t}, \frac{\partial f_t(\boldsymbol{x}_j; \mathbf{A})}{\partial \mathbf{W}_t} \right\rangle, \tag{50}$$

for a pair of nodes (i.e. data points) at optimization time index t. Intuitively, the kernel function measures quantifies similarity between seen and unseen instances based on how differently their outputs given by the GNN change by an infinitesimal perturbation of weights (which we will show is biased by the adjacency matrix used for feature propagation in GNNs).

Note that we consider transductive learning here, which is more convenient. For inductive learning, one should replace $\bar{\mathbf{X}}$ with $\mathbf{X}$, and $\mathbf{A}$ with its submatrix $\mathbf{A}_{\mathbf{XX}}$. Let us denote GNTK with/without feature propagation at ℓ-th layer as $\mathbf{\Theta}^{(\ell)}$/$\bar{\mathbf{\Theta}}^{(\ell)}$, covariance matrix of outputs of ℓ-th layer with/without feature propagation as $\mathbf{\Sigma}^{(\ell)}/\bar{\mathbf{\Sigma}}^{(\ell)}$, covariance matrix of the derivative to ℓ-th layer layer as $\dot{\mathbf{\Sigma}}^{(\ell)}$. The recurrent formula for computing node-level GNTK for infinitely-wide GNNs can be written as the following. The initialization of GNTK for graph regression is given as

$$\bar{\mathbf{\Theta}}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) = \bar{\mathbf{\Sigma}}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) = \boldsymbol{x}_i^\top \boldsymbol{x}_j, \tag{51}$$

We also write matrix form of computing GNTK here in order to give clearer intuitions of how the adjacency matrix can be naturally encoded into the computation of $\mathbf{\Theta}$ in node regression tasks:

$$\bar{\mathbf{\Theta}}^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}) = \bar{\mathbf{\Sigma}}^{(1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) = \bar{\mathbf{X}}^\top \bar{\mathbf{X}}. \tag{52}$$

Propagation. The feature propagation operation (i.e. $\mathbf{Z} \leftarrow \mathbf{AZ}$) at each layer corresponds to

$$\begin{aligned} \mathbf{\Sigma}^{(\ell-1)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) &= \sum_{i' \in \mathcal{N}_i} \sum_{j' \in \mathcal{N}_j} \mathbf{A}_{ii'} \mathbf{A}_{jj'} \bar{\mathbf{\Sigma}}^{(\ell-1)}(\boldsymbol{x}_{i'}, \boldsymbol{x}_{j'}; \mathbf{A}) \\ \mathbf{\Theta}^{(\ell-1)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) &= \sum_{i' \in \mathcal{N}_i} \sum_{j' \in \mathcal{N}_j} \mathbf{A}_{ii'} \mathbf{A}_{jj'} \bar{\mathbf{\Theta}}^{(\ell-1)}(\boldsymbol{x}_{i'}, \boldsymbol{x}_{j'}; \mathbf{A}), \end{aligned} \tag{53}$$

where $\mathcal{N}_i$ denote neighboring nodes of $\boldsymbol{x}_i$ including $\boldsymbol{x}_i$ itself. For GCN where $\mathbf{A}$ is defined as a symmetric normalized adjacency matrix, we have $\mathbf{A}_{ij} = \mathbf{A}_{ji} = 1/\sqrt{d_i d_j}$. The above equation can be compactly described in a matrix form:

$$\begin{aligned} \mathbf{\Sigma}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) &= \mathbf{A}\, \bar{\mathbf{\Sigma}}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) \mathbf{A} \\ \mathbf{\Theta}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) &= \mathbf{A}\, \bar{\mathbf{\Theta}}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) \mathbf{A}. \end{aligned} \tag{54}$$

The multiplication of $\mathbf{A}$ before $\bar{\mathbf{\Theta}}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right)$ gives row-wise weighted summation and after $\bar{\mathbf{\Theta}}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right)$ gives column-wise weighted summation.

Transformation. Let $\mathcal{T}$ and $\dot{\mathcal{T}}$ be functions from 2×2 positive semi-definite matrices $\mathbf{\Lambda}$ to $\mathbb{R}$ given by

$$\begin{cases} \mathcal{T}(\mathbf{\Lambda}) = \mathbb{E}[\sigma(a)\sigma(b)] \\ \dot{\mathcal{T}}(\mathbf{\Lambda}) = \mathbb{E}\left[\sigma'(a)\sigma'(b)\right] \end{cases} \quad (a, b) \sim \mathcal{N}(0, \mathbf{\Lambda}) \tag{55}$$

where σ' is the derivative of the activation function. Then, the feature transformation process (i.e. $\mathbf{Z} = \sigma(\mathbf{ZW})$) then corresponds to:

$$\begin{aligned}\bar{\boldsymbol{\Sigma}}^{(\ell)}\left(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}\right) &= c_\sigma \mathcal{T}\begin{pmatrix} \boldsymbol{\Sigma}^{(\ell-1)}(\boldsymbol{x}_i, \boldsymbol{x}_i; \mathbf{A}) & \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}\right) \\ \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_j, \boldsymbol{x}_i; \mathbf{A}\right) & \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_j, \boldsymbol{x}_j; \mathbf{A}\right) \end{pmatrix}, \\ \dot{\boldsymbol{\Sigma}}^{(\ell)}\left(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}\right) &= c_\sigma \dot{\mathcal{T}}\begin{pmatrix} \boldsymbol{\Sigma}^{(\ell-1)}(\boldsymbol{x}_i, \boldsymbol{x}_i; \mathbf{A}) & \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}\right) \\ \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_j, \boldsymbol{x}_i; \mathbf{A}\right) & \boldsymbol{\Sigma}^{(\ell-1)}\left(\boldsymbol{x}_j, \boldsymbol{x}_j; \mathbf{A}\right) \end{pmatrix}.\end{aligned} \tag{56}$$

The layer-wise computation for $\boldsymbol{\Theta}$ is given as

$$\bar{\boldsymbol{\Theta}}^{(\ell)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) = \boldsymbol{\Theta}^{(\ell-1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) \odot \dot{\boldsymbol{\Sigma}}^{(\ell)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) + \bar{\boldsymbol{\Sigma}}^{(\ell)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right), \tag{57}$$

where $\odot$ denotes Hadamard product. For a ℓ-layer GNN, the corresponding GNTK is given by $\boldsymbol{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A})$, which is abbreviated as $\boldsymbol{\Theta}^{(\ell)}$ in the main text.

B.4. Proof of Theorem 4.2: Two-Layer GNN

Consider a two-layer GNN defined as

$$\{f(\boldsymbol{x}; \mathbf{A})\}_{\boldsymbol{x} \in \bar{\mathbf{X}}} = [\mathbf{F}, \mathbf{F}'] = \mathbf{A}\frac{1}{\sqrt{m}}\sigma\left(\mathbf{A}\bar{\mathbf{X}}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)} \tag{58}$$

where m is the width and σ is ReLU activation. Following previous works that analyze two-layer fully-connected neural networks (Arora et al., 2019; Du et al., 2019c), we consider optimization of the first layer weights $\mathbf{W}^{(1)}$ while fixing the second-layer weights. In this case, the two-layer GNTK is defined as

$$\boldsymbol{\Theta}^{(2)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) = \left\langle \frac{\partial f(\boldsymbol{x}_i; \mathbf{A}, \mathbf{W})}{\partial \mathbf{W}^{(1)}}, \frac{\partial f(\boldsymbol{x}_j; \mathbf{A}, \mathbf{W})}{\partial \mathbf{W}^{(1)}} \right\rangle. \tag{59}$$

We can derive the the explicit form formula for computing it in overparameterized regime based on the general formula for arbitrarily deep GNNs given in Appendix. B.3 (or directly calculating $\partial f(\boldsymbol{x}_i; \mathbf{A}, \mathbf{W})/\partial \mathbf{W}^{(1)}$ by using chain rule). Consequently, we have

$$\begin{aligned}\boldsymbol{\Theta}^{(2)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) = \sum_{i' \in \mathcal{N}(i)} \sum_{j' \in \mathcal{N}(j)} & \mathbf{A}_{ii'}\mathbf{A}_{jj'}\left([\mathbf{A}\bar{\mathbf{X}}]_{i'}^\top [\mathbf{A}\bar{\mathbf{X}}]_{j'}\right) \\ & \mathbb{E}_{\boldsymbol{w} \sim \mathcal{N}(0,1)}\left[\mathbb{1}\left\{\boldsymbol{w}^\top [\mathbf{A}\bar{\mathbf{X}}]_{i'} \geq 0, \boldsymbol{w}^\top [\mathbf{A}\bar{\mathbf{X}}]_{j'} \geq 0\right\}\right]\end{aligned} \tag{60}$$

where $\mathcal{N}(i)$ denote the set of neighboring nodes, $\boldsymbol{w} \in \mathbb{R}^d$ is sampled from Gaussian distribution, and $\mathbb{1}$ is indicator function.

Case 1. If we stipulate the input features are represented by an identity matrix, i.e. $\bar{\mathbf{X}} = \mathbf{I}_n$, it follows that

$$\begin{aligned}\boldsymbol{\Theta}^{(2)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) &= \sum_{i' \in \mathcal{N}(i)} \sum_{j' \in \mathcal{N}(j)} \mathbf{A}_{ii'}\mathbf{A}_{jj'}\left(\mathbf{A}_{i'}^\top \mathbf{A}_{j'}\right)\mathbb{E}_{\boldsymbol{w} \sim \mathcal{N}(0,1)}\left[\mathbb{1}\left\{\boldsymbol{w}^\top \mathbf{A}_{i'} \geq 0, \boldsymbol{w}^\top \mathbf{A}_{j'} \geq 0\right\}\right] \\ &= \sum_{i' \in \mathcal{N}(i)} \sum_{j' \in \mathcal{N}(j)} \mathbf{A}_{ii'}\mathbf{A}_{jj'} \frac{\mathbf{A}_{i'}^\top \mathbf{A}_{j'}(\pi - \arccos(\frac{\mathbf{A}_{i'}^\top \mathbf{A}_{j'}}{\|\mathbf{A}_{i'}\|\|\mathbf{A}_{j'}\|}))}{2\pi}.\end{aligned} \tag{61}$$

The above equation can be written neatly in the matrix form

$$\boldsymbol{\Theta}^{(2)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}) = \mathbf{A}(\mathbf{A}^2 \odot \mathbf{S})\mathbf{A}, \quad \text{where} \quad \mathbf{S}_{ij} = (\pi - \arccos(\frac{\mathbf{A}_i^\top \mathbf{A}_j}{\|\mathbf{A}_i\|\|\mathbf{A}_j\|}))/2\pi. \tag{62}$$

The matrix $\mathbf{S}$ reweights each entry in $\mathbf{A}^2$ by the similarity of neighborhood patterns of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Substituting it to (5) gives us the training dynamics of overparameterized two-layer GNN

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta\mathbf{A}(\mathbf{A}^2 \odot \mathbf{S})\mathbf{A}[\mathbf{R}_t, \mathbf{0}] \ + \ [\mathbf{R}_t, \mathbf{R}'_t]. \tag{63}$$

Case 2. If we stipulate the input features are node embeddings from spectral decomposition of a full-rank adjacency matrix, i.e. $\bar{\mathbf{X}} \triangleq \arg\min_{\mathbf{B}} \|\mathbf{A} - \mathbf{B}\mathbf{B}^\top\|_F^2$. We have

$$\begin{aligned}
&\mathbf{\Theta}^{(2)}(\boldsymbol{x}_i, \boldsymbol{x}_j; \mathbf{A}) \\
&= \sum_{i' \in \mathcal{N}(i)} \sum_{j' \in \mathcal{N}(j)} \mathbf{A}_{ii'}\mathbf{A}_{jj'}\mathbf{A}^3_{i'j'} \mathbb{E}_{\boldsymbol{w}\sim\mathcal{N}(0,1)} \left[\mathbb{1}\left\{\boldsymbol{w}^\top[\mathbf{A}\bar{\mathbf{X}}]_{i'} \geq 0, \boldsymbol{w}^\top[\mathbf{A}\bar{\mathbf{X}}]_{j'} \geq 0\right\}\right] \\
&= \sum_{i' \in \mathcal{N}(i)} \sum_{j' \in \mathcal{N}(j)} \mathbf{A}_{ii'}\mathbf{A}_{jj'} \frac{\mathbf{A}^3_{i'j'}(\pi - \arccos(\frac{[\mathbf{A}\bar{\mathbf{X}}]_{i'}^\top[\mathbf{A}\bar{\mathbf{X}}]_{j'}}{\|[\mathbf{A}\bar{\mathbf{X}}]_{i'}\|\|[\mathbf{A}\bar{\mathbf{X}}]_{j'}\|}))}{2\pi}.
\end{aligned} \tag{64}$$

The above equation can be written neatly in the matrix form

$$\mathbf{\Theta}^{(2)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}) = \mathbf{A}(\mathbf{A}^3 \odot \tilde{\mathbf{S}})\mathbf{A}, \quad \text{where} \quad \tilde{\mathbf{S}}_{ij} = (\pi - \arccos(\frac{[\mathbf{A}\bar{\mathbf{X}}]_i^\top[\mathbf{A}\bar{\mathbf{X}}]_j}{\|[\mathbf{A}\bar{\mathbf{X}}]_i\|\|[\mathbf{A}\bar{\mathbf{X}}]_j\|}))/2\pi. \tag{65}$$

The matrix $\tilde{\mathbf{S}}$ reweights each entry in $\mathbf{A}^3$ by the similarity of aggregated node embeddings $[\mathbf{A}\bar{\mathbf{X}}]_i$ and $[\mathbf{A}\bar{\mathbf{X}}]_j$. Substituting it to (5) gives us the training dynamics of overparameterized two-layer GNN

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta\mathbf{A}(\mathbf{A}^3 \odot \tilde{\mathbf{S}})\mathbf{A}[\mathbf{R}_t, \mathbf{0}] \ + \ [\mathbf{R}_t, \mathbf{R}'_t]\,. \tag{66}$$

B.5. Proof: Arbitrarily Deep Decoupled GNN

Before analyzing the training dynamics of deep GNNs, let us first consider an arbitrarily deep standard fully-connected neural networks, which is denoted as $\mathrm{MLP}(\bar{\mathbf{X}})$ with weights $\mathbf{W}$. We also denote its corresponding NTK in overparameterized regime as $\mathbf{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$. For input features as an identity matrix $\bar{\mathbf{X}} = \mathbf{I}_n$, we have the following permutation invariant property of $\mathbf{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$.

Lemma B.1 (Permutation Invariance of NTK)**.** *For arbitrarily deep and infinitely wide fully-connected neural networks* $\mathrm{MLP}(\bar{\mathbf{X}})$ *with ReLU activation and standard NTK parameterization, the corresponding NTK* $\mathbf{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$ *for onehot vector inputs is permutation invariant. Namely for arbitrary permutation function* ψ *on input set* $\bar{\mathbf{X}}$

$$\mathbf{\Theta}^{(\ell)}(\psi(\bar{\mathbf{X}}), \psi(\bar{\mathbf{X}})) = \mathbf{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}). \tag{67}$$

Proof. Recall that in the explicit form computation for NTK of fully-connected neural networks (Jacot et al., 2018; Lee et al., 2019), the NTK and NNGP for a $(\ell+1)$-layer network is produced from the NTK and NNGP for a ℓ-layer network. Therefore, to prove the permutation invariance of NTK of an arbitrarily deep fully-connected neural network, we need only to prove the permutation invariance of $\mathbf{\Theta}^{(1)}$ and $\mathbf{\Sigma}^{(1)}$, which are defined as the inner product of input features

$$\mathbf{\Theta}^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) = \mathbf{\Sigma}^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) = \bar{\mathbf{X}}\bar{\mathbf{X}}^\top. \tag{68}$$

Since arbitrary permutation function ψ on $\bar{\mathbf{X}}$ can be expressed as

$$\psi(\bar{\mathbf{X}}) = \prod_{i=0}^{c} \mathbf{T}_i\bar{\mathbf{X}} \tag{69}$$

for some row-interchanging elementary matrices $\{\mathbf{T}_i\}_{i=0}^c$. It follows that

$$\begin{aligned}
\mathbf{\Theta}^{(1)}(\psi(\bar{\mathbf{X}}), \psi(\bar{\mathbf{X}})) &= \prod_{i=0}^{c} \mathbf{T}_i\bar{\mathbf{X}}(\prod_{i=0}^{c} \mathbf{T}_i\bar{\mathbf{X}})^\top \\
&= \prod_{i=0}^{c} \mathbf{T}_i\bar{\mathbf{X}}\bar{\mathbf{X}}^\top \prod_{i=0}^{c} \mathbf{T}_{c-i}^\top \\
&= \mathbf{T}_0 \cdots \mathbf{T}_c\mathbf{T}_c^\top \cdots \mathbf{T}_0^\top \\
&= \mathbf{I}_n \\
&= \mathbf{\Theta}^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}).
\end{aligned} \tag{70}$$

By induction, it follows that $\mathbf{\Theta}^{(\ell)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$ is also permutation invariant. □

Now, let us consider a deep GNN defined as

$$[\mathbf{F}, \mathbf{F}'] = \mathbf{A}^{\ell}\,\mathrm{MLP}(\bar{\mathbf{X}}) \tag{71}$$

where $\mathrm{MLP}(\bar{\mathbf{X}})$ is a L-layer infinitely-wide fully-connected neural network with ReLU activation. Then, the corresponding node-level GNTK matrix is defined as

$$\left\langle \frac{\partial \mathbf{A}^{\ell}\,\mathrm{MLP}(\bar{\mathbf{X}})}{\partial \mathbf{W}}, \frac{\partial \mathbf{A}^{\ell}\,\mathrm{MLP}(\bar{\mathbf{X}})}{\partial \mathbf{W}} \right\rangle = \mathbf{A}^{\ell}\mathbf{\Theta}^{(L)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})\mathbf{A}^{\ell}. \tag{72}$$

Since the NTK matrix $\mathbf{\Theta}^{(L)}(\bar{\mathbf{X}}, \bar{\mathbf{X}})$ is permutation invariant with diagonal values being larger than non-diagonal values, we can write it as

$$\mathbf{\Theta}^{(L)}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) = c'(\mathbf{I} + c\mathbf{1}\mathbf{1}^{\top}) \tag{73}$$

for some constants c' and c determined by the depth of the network L. Substituting it to (5) gives us the training dynamics of infinitely-wide and arbitrarily-deep GNN

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^{\ell}(\mathbf{I} + c\mathbf{1}\mathbf{1}^{\top})\mathbf{A}^{\ell}[\mathbf{R}_t, \mathbf{0}] \;+\; [\mathbf{R}_t, \mathbf{R}'_t]\,. \tag{74}$$

B.6. Proof of Corollary 4.4: Linear GNN

For linear GNN defined as $[\mathbf{F}, \mathbf{F}'] = \mathbf{A}^{\ell}\bar{\mathbf{X}}\mathbf{W}$. The corresponding node-level GNTK $\mathbf{\Theta}_t^{(1)}$ for linear GNN is naturally constant and can be computed as

$$\begin{aligned} \mathbf{\Theta}_t^{(1)}\left(\bar{\mathbf{X}}, \bar{\mathbf{X}}; \mathbf{A}\right) &= \nabla_{\mathbf{W}}[\mathbf{F}, \mathbf{F}']^{\top}\nabla_{\mathbf{W}}[\mathbf{F}, \mathbf{F}'] \\ &= \mathbf{A}^{\ell}\bar{\mathbf{X}}(\mathbf{A}^{\ell}\bar{\mathbf{X}})^{\top}. \end{aligned} \tag{75}$$

Case 1. When the input $\bar{\mathbf{X}}$ is defined as an identity matrix, we have

$$\mathbf{A}^{\ell}\bar{\mathbf{X}}(\mathbf{A}^{\ell}\bar{\mathbf{X}})^{\top} = \mathbf{A}^{2\ell} \tag{76}$$

which is also a special case of Theorem 4.3 where the backbone MLP model is one layer and consequently $c = 0$. Based on (5), the training dynamics of linear GNN can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^{2\ell}[\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t] \tag{77}$$

which is equivalent to the basic version of RP in (6) with $K = 2\ell$.

Case 2. When the input $\bar{\mathbf{X}}$ is obtained from (full-rank) graph spectral decomposition, we have

$$\mathbf{A}^{\ell}\bar{\mathbf{X}}(\mathbf{A}^{\ell}\bar{\mathbf{X}})^{\top} = \mathbf{A}^{\ell}\bar{\mathbf{X}}\bar{\mathbf{X}}^{\top}\mathbf{A}^{\ell} = \mathbf{A}^{2\ell+1}. \tag{78}$$

In this case, the training dynamics of linear GNN can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^{2\ell+1}[\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t] \tag{79}$$

which is equivalent to the basic version of RP in (6) with $K = 2\ell + 1$. For non-PSD adjacency matrix $\mathbf{A}$, we also have the approximation $\mathbf{A}^{\ell}\bar{\mathbf{X}}^{\top}\bar{\mathbf{X}}\mathbf{A}^{\ell} \approx \mathbf{A}^{2\ell+1}$. However, in this case, the basic RP algorithm in (6) can not be implemented by the conventional (deep) learning framework (cf. Theorem 3.2).

B.7. Derivation of (16): Generalization Bound

Based on the Rademacher complexity-based generalization bound for kernel regression in (Bartlett & Mendelson, 2002), for training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_l}$ drawn i.i.d. from an underlying distribution $\mathcal{P}$, arbitrary loss function $l : \mathbb{R} \times \mathbb{R} \to [0, 1]$ that is 1-Lipschitz in the first argument such that $l(y, y) = 0$, we have that with probability at least $1 - \delta$, the population risk of kernel regression with respect to the limit NTK $\mathbf{\Theta}$ has upper bound (see proof in Bartlett & Mendelson (2002); Arora et al. (2019); Du et al. (2019b))

$$\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\left[l\left(f(\boldsymbol{x}), y\right)\right] = O\left(\frac{\sqrt{\mathbf{Y}^{\top}\mathbf{\Theta}^{-1}\mathbf{Y}\cdot\mathrm{Tr}(\mathbf{\Theta})}}{n_l} + \sqrt{\frac{\log(1/\delta)}{n_l}}\right). \tag{80}$$

We now assume an extreme case where the limit NTK matrix is determined by the graph, i.e. $\mathbf{\Theta} = (\mathbf{I} - \alpha\mathbf{A})^{-1} = \lim_{k\to\infty}\sum_{i=0}^{k}(\alpha\mathbf{A})^i$, which is equivalent to the propagation matrix adopted by the converged LP algorithm,[2] and is a valid kernel matrix by noting that the spectral radius $\rho(\mathbf{A}) \leq 1$ for normalized adjacency matrix $\mathbf{A}$ and $\alpha < 1$. For $\mathbf{Y}^\top\mathbf{\Theta}^{-1}\mathbf{Y}$, we have

$$\begin{aligned}\mathbf{Y}^\top\mathbf{\Theta}^{-1}\mathbf{Y} &= \mathbf{Y}^\top\mathbf{Y} - \alpha\mathbf{Y}^\top\mathbf{A}\mathbf{Y} \\ &= n_l - \alpha\langle\mathbf{Y}\mathbf{Y}^\top, \mathbf{A}\rangle_F \\ &= n_l - cA(\mathbf{\Theta}^*, \mathbf{A})\end{aligned} \tag{81}$$

where $c = \alpha\|\mathbf{\Theta}^*\|_F\|\mathbf{A}\|_F$ is a constant. For $\mathrm{Tr}(\mathbf{\Theta})$, we have

$$\mathrm{Tr}(\mathbf{\Theta}) = \sum_{i=1}^{n_l}\frac{1}{1-\alpha\sigma_i(\mathbf{A})} \leq \frac{n_l}{1-\alpha\sigma_{max}(\mathbf{A})} = O(n_l), \tag{82}$$

where $\sigma_{max}(\mathbf{A}) \leq 1$ is the maximal eigenvalue of $\mathbf{A}$. It follow that the population risk has upper bound

$$\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\left[l\left(f(\boldsymbol{x}), y\right)\right] = O\left(\sqrt{\frac{n_l - cA(\mathbf{\Theta}^*, \mathbf{A})}{n_l}} + \sqrt{\frac{\log(1/\delta)}{n_l}}\right). \tag{83}$$

B.8. Proof of Theorem 5.2: Bayesian Optimality of GNN

Setup. In this proof, we consider the following setup. Suppose all possible samples are given by inputs $\mathbf{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ and labels $\mathbf{Y} = \{y(\boldsymbol{x}_i)\}_{i=1}^N$ that are randomly generated from an unknown distribution $\mathcal{P}$, where N could be arbitrarily large and both $\mathbf{X}$ and $\mathbf{Y}$ are assumed to be unobserved. For binary classification, we also have $y \in \{-1, 1\}$ with balanced label distribution, i.e. $\mathbb{E}[y] = 0$. Given that for an arbitrary pair of instances $\boldsymbol{x}$ and $\boldsymbol{x}'$, the probability that they share the same label is defined by a large matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$:

$$P\left(y(\boldsymbol{x}) = y(\boldsymbol{x}')\right) = \mathbf{A}_{\boldsymbol{x}\boldsymbol{x}'}, \tag{84}$$

where $\mathbf{A}_{\boldsymbol{x}\boldsymbol{x}'}$ is an element in $\mathbf{A}$ that corresponds to sample pair $\boldsymbol{x}$ and $\boldsymbol{x}'$. Our task is to find the optimal kernel function whose corresponding kernel regression predictive function $f_{ker}(\cdot)$ minimizes the population risk

$$\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\left[\left(f_{ker}(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2\right]. \tag{85}$$

Proof. Next, we give solution to the optimal kernel function that minimizes the population risk in the above setting. For data randomly generated from $\mathcal{P}$ and conditioned on (84), the random labels have covariance matrix $\mathbf{\Sigma}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N\times N}$ which satisfies

$$\begin{aligned}\mathbf{\Sigma}(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\left[(y(\boldsymbol{x}) - \mathbb{E}[y])(y(\boldsymbol{x}') - \mathbb{E}[y])\right] \\ &= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\left[y(\boldsymbol{x})y(\boldsymbol{x}')\right] \\ &= 2\mathbf{A}_{\boldsymbol{x}\boldsymbol{x}'} - 1.\end{aligned} \tag{86}$$

Let us consider kernel regression w.r.t. a kernel κ, whose corresponding predictive function $f_{ker}(\boldsymbol{x})$ on all possible unobserved samples $\mathbf{X}$ and $\mathbf{Y}$ is given by

$$f_{ker}(\boldsymbol{x}) = \kappa(\boldsymbol{x}, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{Y} = \mathbf{M}_{\boldsymbol{x}}^\top\mathbf{Y}. \tag{87}$$

where $\mathbf{M}_{\boldsymbol{x}} \in \mathbb{R}^N$ is a vector associated with sample $\boldsymbol{x}$ defined as $\mathbf{M}_{\boldsymbol{x}}^\top = \kappa(\boldsymbol{x}, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}$. Note that we do not differentiate training or testing samples here since all inputs and labels are assumed to be unseen. We aim to search the optimal kernel function that minimizes the population risk, which can be achieved if the risk for each sample $\boldsymbol{x}$ is minimized. Specifically, for an arbitrary sample $\boldsymbol{x}$, its risk is

$$\begin{aligned}\mathbb{E}_y\left[\left(f_{ker}(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2\right] &= \mathbb{E}_y\left[\left(\mathbf{M}_{\boldsymbol{x}}^\top\mathbf{Y} - y(\boldsymbol{x})\right)^2\right] \\ &= \mathbf{M}_{\boldsymbol{x}}^\top\mathbf{\Sigma}(\mathbf{X}, \mathbf{X})\mathbf{M}_{\boldsymbol{x}} - 2\mathbf{M}_{\boldsymbol{x}}^\top\mathbf{\Sigma}(\mathbf{X}, \boldsymbol{x}) + \mathbf{\Sigma}(\boldsymbol{x}, \boldsymbol{x}).\end{aligned} \tag{88}$$

[2] With slight abuse of notation, we denote by $\mathbf{A}$ the adjacency matrix for the training set, and $\mathbf{\Theta}^*$ the optimal kernel matrix for the training set.

To find the optimal kernel that minimizes the risk, we differentiate it w.r.t. $\mathbf{M}_{\boldsymbol{x}}$,

$$\nabla_{\mathbf{M}_{\boldsymbol{x}}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}} \left[\left(f_{ker}(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2 \right] = 2\boldsymbol{\Sigma}(\mathbf{X},\mathbf{X})\mathbf{M}_{\boldsymbol{x}} - 2\boldsymbol{\Sigma}(\mathbf{X},\boldsymbol{x}) \tag{89}$$

which gives us the minimizer

$$\mathbf{M}_{\boldsymbol{x}}^{\top} = \boldsymbol{\Sigma}(\boldsymbol{x},\mathbf{X})\boldsymbol{\Sigma}(\mathbf{X},\mathbf{X})^{-1}. \tag{90}$$

And since $\mathbf{M}_{\boldsymbol{x}}^{\top} = \kappa(\boldsymbol{x},\mathbf{X})\mathbf{K}(\mathbf{X},\mathbf{X})^{-1}$, the population risk is minimized on each single data point for optimal kernel κ^* whose kernel matrix is

$$\mathbf{K}^*(\mathbf{X},\mathbf{X}) = 2\mathbf{A} - 1. \tag{91}$$

Remark. The above analysis also generalize to the setting when $\mathbf{A}$ is defined over a subset of all possible samples, in which case the optimal kernel matrix is still $2\mathbf{A} - 1$ but it is no longer possible to minimize population risk outside the coverage of $\mathbf{A}$.

C. Additional Discussions

C.1. Different Settings

Transductive (semi-supervised) and inductive (supervised) learning are two types of common settings in node-level classification tasks. The former incorporates unlabeled nodes (testing samples) in the training process while the latter only has access to unlabeled nodes for inference. Recall that the residual propagation process of general parameterized models with arbitrary unseen testing samples can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \begin{bmatrix} \underbrace{\boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X})}_{n_l\times n_l} & \boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X}') \\ \underbrace{\boldsymbol{\Theta}_t(\mathbf{X}',\mathbf{X})}_{(n-n_l)\times n_l} & \boldsymbol{\Theta}_t(\mathbf{X}',\mathbf{X}') \end{bmatrix} \cdot \underbrace{[\mathbf{R}_t, \mathbf{0}]}_{n} + \underbrace{[\mathbf{R}_t, \mathbf{R}'_t]}_{n}. \tag{92}$$

For node-level tasks, the difference between transductive and inductive settings boils down to different feature map of the kernel function for training and testing sets. To be specific:

Transductive learning. For transductive learning or semi-supervised learning, the residual propagation can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \begin{bmatrix} \underbrace{\boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X};\mathbf{A})}_{n_l\times n_l} & \boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X}';\mathbf{A}) \\ \underbrace{\boldsymbol{\Theta}_t(\mathbf{X}',\mathbf{X};\mathbf{A})}_{(n-n_l)\times n_l} & \boldsymbol{\Theta}_t(\mathbf{X}',\mathbf{X}';\mathbf{A}) \end{bmatrix} \cdot \underbrace{[\mathbf{R}_t, \mathbf{0}]}_{n} + \underbrace{[\mathbf{R}_t, \mathbf{R}'_t]}_{n}. \tag{93}$$

where $\boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X};\mathbf{A})$ is the node-level GNTK defined in Sec. 4. This equation is equivalent to the one presented in the main text, i.e. (5).

Inductive learning. Let us denote $\mathbf{A}_{\mathbf{XX}}$ the submatrix of $\mathbf{A}$ corresponding to the training set. For inductive learning or supervised learning, the residual propagation can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \begin{bmatrix} \underbrace{\boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X};\mathbf{A}_{\mathbf{XX}})}_{n_l\times n_l} & \boldsymbol{\Theta}_t^{(ind)}(\mathbf{X},\mathbf{X}';\mathbf{A}) \\ \underbrace{\boldsymbol{\Theta}_t^{(ind)}(\mathbf{X}',\mathbf{X};\mathbf{A})}_{(n-n_l)\times n_l} & \boldsymbol{\Theta}_t^{(ind)}(\mathbf{X}',\mathbf{X}';\mathbf{A}) \end{bmatrix} \cdot \underbrace{[\mathbf{R}_t, \mathbf{0}]}_{n} + \underbrace{[\mathbf{R}_t, \mathbf{R}'_t]}_{n}, \tag{94}$$

Let us denote predictions (for the training set) given by the GNN model with matrix $\mathbf{A}_{\mathbf{XX}}$ as $\mathbf{F}_{ind}$, and the predictions (for the testing set) given by the GNN model with matrix $\mathbf{A}$ as $\mathbf{F}'_{trans}$, we have

$$\begin{aligned} \boldsymbol{\Theta}_t(\mathbf{X},\mathbf{X};\mathbf{A}_{\mathbf{XX}}) &= \nabla_{\mathbf{W}}\mathbf{F}_{ind}^{\top}\nabla_{\mathbf{W}}\mathbf{F}_{ind}, \\ \boldsymbol{\Theta}_t^{(ind)}(\mathbf{X}',\mathbf{X};\mathbf{A}) &= \nabla_{\mathbf{W}}\mathbf{F}'^{\top}_{trans}\nabla_{\mathbf{W}}\mathbf{F}_{ind}. \end{aligned} \tag{95}$$

In this case, the $n \times n$ matrix is still a valid kernel matrix with graph implicit bias (cf. Appendix B.3), and thus our insights from the transductive setting still holds.

Training without graph. Another interesting setting studied recently (Yang et al., 2023) is using vanilla MLP for training and then adopts the GNN architecture in inference, which leads to the so-called PMLP that can accelerate training without deteriorating the generalization performance. In this setting, the residual propagation can be written as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \begin{bmatrix} \underbrace{\boldsymbol{\Theta}_t(\mathbf{X}, \mathbf{X})}_{n_l \times n_l} & \boldsymbol{\Theta}_t^{(pmlp)}(\mathbf{X}, \mathbf{X}'; \mathbf{A}) \\ \underbrace{\boldsymbol{\Theta}_t^{(pmlp)}(\mathbf{X}', \mathbf{X}; \mathbf{A})}_{(n-n_l)\times n_l} & \boldsymbol{\Theta}_t^{(pmlp)}(\mathbf{X}', \mathbf{X}'; \mathbf{A}) \end{bmatrix} \cdot \underbrace{[\mathbf{R}_t, \mathbf{0}]}_{n} \; + \; \underbrace{[\mathbf{R}_t, \mathbf{R}'_t]}_{n}, \tag{96}$$

where $\boldsymbol{\Theta}_t(\mathbf{X}, \mathbf{X})$ is equivalent to the NTK matrix of fully-connected neural networks. This is equivalent to the inductive setting where the graph adjacency matrix used in training is an identity matrix: let us denote predictions (for the training set) given by MLP as $\mathbf{F}_{mlp}$, we have

$$\begin{aligned} \boldsymbol{\Theta}_t(\mathbf{X}, \mathbf{X}) &= \nabla_{\mathbf{W}} \mathbf{F}_{mlp}^\top \nabla_{\mathbf{W}} \mathbf{F}_{mlp}, \\ \boldsymbol{\Theta}_t^{(ind)}(\mathbf{X}', \mathbf{X}; \mathbf{A}) &= \nabla_{\mathbf{W}} \mathbf{F}'^\top_{trans} \nabla_{\mathbf{W}} \mathbf{F}_{mlp}. \end{aligned} \tag{97}$$

The success of PMLP is consistent with the insight that the residual flow between training and testing sets with graph implicit bias is important to explain the generalization of GNNs.

C.2. Other Loss Functions

It is worth noting that residual propagation process is not exclusive to the squared loss. While our analysis focuses on the squared loss, similar residual propagation schemes can be obtained with other loss functions that will change the original linear residual propagation process to non-linear. Therefore, the insight still holds for other loss functions.

Mean squared error. For completeness, let us rewrite the residual propagation process for general parameterized model (e.g. linear model, fully-connected neural network, GNN) with squared loss here, with derivation given in Appendix B.1

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \, \boldsymbol{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}})[\mathbf{R}_t, \mathbf{0}] \; + \; [\mathbf{R}_t, \mathbf{R}'_t] \,, \tag{98}$$

which is a linear propagation process, where $\boldsymbol{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}}) = \nabla_{\mathbf{W}}[\mathbf{F}_t, \mathbf{F}'_t]^\top \; \nabla_{\mathbf{W}}[\mathbf{F}_t, \mathbf{F}'_t] \in \mathbb{R}^{n \times n}$.

Mean absolute error. If the loss function is a MAE (a.k.a. L_1) loss $\mathcal{L} = \|\mathbf{F}_t - \mathbf{Y}\|_1$, then the residual propagation process can be revised as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \, \boldsymbol{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}})[\mathrm{sgn}(\mathbf{R}_t), \mathbf{0}] \; + \; [\mathbf{R}_t, \mathbf{R}'_t] \,, \tag{99}$$

where $\mathrm{sgn}(x)$ is the sign function that output 1 is $x > 0$ otherwise -1, and could be thought of as coarsening the residual information to a binary value.

Cross-entropy. For the CE loss commonly used for classification tasks, we define the residual as $\mathbf{R}_t = \mathbf{Y} - \sigma \mathbf{F}_t$, where σ here denotes sigmoid activation and applies on every element in vector $\mathbf{F}_t$. The residual propagation process can be revised as

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \, \boldsymbol{\Theta}_t(\bar{\mathbf{X}}, \bar{\mathbf{X}})[c(\mathbf{R}_t), \mathbf{0}] \; + \; [\mathbf{R}_t, \mathbf{R}'_t] \,, \tag{100}$$

where the function $c(\cdot)$ denotes multiplying a scaling factor $1/(1 - \sigma f_t(\boldsymbol{x}_i))\sigma f_t(\boldsymbol{x}_i)$ to each residual for re-weighting. This scaling factor up-weights smaller residuals whose corresponding predictions are more confident, and down-weights larger residuals whose predictions are less confident.

C.3. Multi-Dimensional Output

In this section, we discuss the modification of analysis for training dynamics of GNNs when outputs $f(\boldsymbol{x}) \in \mathbb{R}^c$ is multi-dimensional. For the multi-dimensional output case where $f(\boldsymbol{x}) \in \mathbb{R}^c$ and c is the dimension of outputs, let $\mathbf{F}_t \in \mathbb{R}^{n_l c \times 1}$ and $\mathbf{R}_t \in \mathbb{R}^{n_l c \times 1}$ be model prediction and residuals in vector forms. Then, the residual dynamics for the training set (which

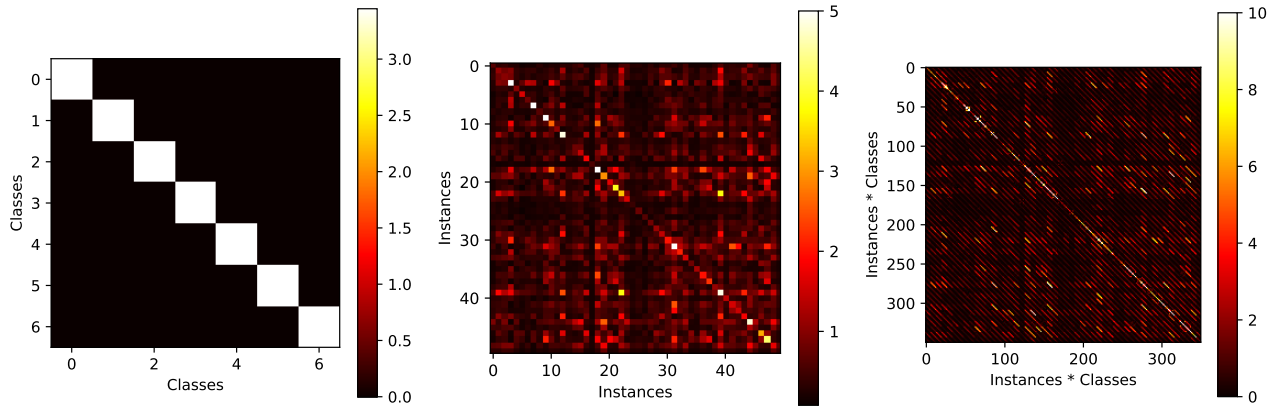


Figure 4: Visualization of NTK of well-trained GCN on a node classification benchmark (Cora). 50 nodes are randomly selected for clearity. From left to right are $c \times c$, $n \times n$, $nc \times nc$ NTK matrices, where the former two matrices are obtained by averaging the $nc \times nc$ NTK matrix at dimension n and c respectively. The diagonal patterns in the first and last matrix verifies that our analysis for finitely-wide GNNs in binary classification also applies to multi-class classification setting.

can straight-forwardly incorporate testing samples similar to the derivation in Appendix B.1) is revised as

$$\frac{\partial \mathbf{R}_t}{\partial t} = \frac{\partial \mathbf{R}_t}{\partial \mathbf{W}_t} \cdot \frac{\partial \mathbf{W}_t}{\partial t} \qquad \text{(Chain rule)} \tag{101}$$

$$= -\eta \cdot \frac{\partial \mathbf{R}_t}{\partial \mathbf{W}_t} \cdot \nabla_{\mathbf{W}} \mathcal{L} \qquad \text{(GD training)} \tag{102}$$

$$= -\eta \cdot \frac{\partial \mathbf{R}_t}{\partial \mathbf{W}_t} \cdot \nabla_{\mathbf{W}} \mathbf{F}_t \cdot \nabla_f \mathcal{L} \qquad \text{(Chain rule)} \tag{103}$$

$$= \eta \cdot \underbrace{\nabla_{\mathcal{W}}^{\top} \mathbf{F}_t}_{n_l c \times |\mathbf{W}|} \cdot \underbrace{\nabla_{\mathbf{W}} \mathbf{F}_t}_{|\mathbf{W}| \times n_l c} \cdot \underbrace{\nabla_f \mathcal{L}}_{n_l c \times 1} \qquad \text{(Change of notation)} \tag{104}$$

$$= \eta \cdot \underbrace{\mathbf{\Theta}_t(\mathbf{X}, \mathbf{X})}_{n_l c \times n_l c} \cdot \underbrace{\nabla_f \mathcal{L}}_{n_l c \times 1} \tag{105}$$

$$= -\eta \cdot \underbrace{\mathbf{\Theta}_t(\mathbf{X}, \mathbf{X})}_{n_l c \times n_l c} \cdot \underbrace{\mathbf{R}_t}_{n_l c \times 1} \qquad \text{(Squared loss).} \tag{106}$$

Compared with residual propagation process for scalar output, the residual propagation process for multi-dimensional output additionally incorporate flow of residual across different dimensions. However, for infinite neural networks, the NTK $\mathbf{\Theta}(\mathbf{X}, \mathbf{X})$ for multi-dimensional output (Jacot et al., 2018) can be written as

$$\underbrace{\mathbf{\Theta}(\mathbf{X}, \mathbf{X})}_{n_l c \times n_l c} = \underbrace{\bar{\mathbf{\Theta}}(\mathbf{X}, \mathbf{X})}_{n_l \times n_l} \otimes \underbrace{\mathbf{I}_c}_{c \times c}. \tag{107}$$

Namely, there is no cross-dimension flow in residual propagation and our analysis for scalar output can be trivially adapted to the multi-dimensional output case. For finite neural networks, while cross-dimension residual propagation is theoretically inevitable, one can still resort the the above decomposition as approximation. For example, we can define pseudo NTK as

$$\bar{\mathbf{\Theta}}_t(\mathbf{X}, \mathbf{X}) = \left(\nabla_{\mathbf{W}} \sum_{h=1}^{c} [\mathbf{F}_t]_h \right)^{\top} \left(\nabla_{\mathbf{W}} \sum_{h=1}^{c} [\mathbf{F}_t]_h \right), \tag{108}$$

and recent works (Mohamadi & Sutherland, 2023) has proved that such a kernel function can be used to approximate the original $n_l c \times n_l c$ kernel matrix:

$$\frac{\left\| \bar{\mathbf{\Theta}}_t(\boldsymbol{x}_i, \boldsymbol{x}_j) \otimes \mathbf{I}_c - \mathbf{\Theta}_t(\boldsymbol{x}_i, \boldsymbol{x}_j) \right\|_F}{\left\| \mathbf{\Theta}_t(\boldsymbol{x}_i, \boldsymbol{x}_j) \right\|_F} \in \tilde{\mathcal{O}}\left(n_l^{-\frac{1}{2}} \right). \tag{109}$$

To empirically verify this for finitely-wide GNNs, we visualize the NTK matrix for real-world GNNs in Cora dataset in Fig. 4. Therefore, our insights for both infinitely-wide GNNs and finitely-wide GNNs still hold in multi-dimensional case, which are also verified by our experiments.

C.4. Practical Implications for Heterophily

The analysis in works can be used to provide high-level design principles for handling heterophily. For instances:

Increasing $A(\mathbf{A}, \mathbf{\Theta}^*)$ by designing propagation matrices that deviate from the original adjacency matrix, such that the resulting GNN NTK could better align with the optimal kernel matrix, can lead to better generalization performance. Moreover, this perspective helps to substantiate certain existing heuristics for handling heterophily, such as assigning negative weights to edges;

Decreasing $A(\mathbf{\Theta}_t, \mathbf{\Theta}^*)$ by designing non-standard GNN architectures, i.e., that deviate from our defintion in Section 2, can reduce the degradation of NTK-target alignment caused by heterophily; in doing so, the negative effects of heterophilic graphs on generalization can also be mitigated. Incidently, this also helps to quantify regimes where non-GNN architectures such as MLPs can sometimes perform better.

We note also that even the extremely simple RP algorithm that naturally emerges from our analysis works well in practice handling heterophily. See Appendix E for empirical examples.

D. Implementation Details

D.1. Experiments in Section 3.2

`Arxiv`, `Proteins` and `Products` (Hu et al., 2020) are three relatively large datasets containing 169343, 132534 and 2449029 nodes respectively.

- The `Arxiv` dataset represents the citation network between all computer science arxiv papers. Each node is associated with a feature vector representing the averaged embeddings of words in the title and abstract of that paper and the task is to predict the subject areas.
- For `Proteins` dataset, nodes represent proteins and edges represent biologically significant associations between proteins, categorized by their types. The task is to predict the presence or absence of 112 protein functions as a multi-label binary classification problem.
- The `Products` dataset is an Amazon product co-purchasing network where nodes are products in Amazon and links represent two products are purchased together. The feature for each node is dimensionality-reduced bag-of-words for the product descriptions and the task is predict the category of a product.

We follow the original splitting of (Hu et al., 2020) for evaluation. The statistics of these datasets are shown in Table 3.

We compare GRP with several classic methods for learning on graphs: standard *MLP*, Label Propagation (*LP*) (Zhu et al., 2003), *LinearGNN* (SGC) (Wu et al., 2019), *GNN (GCN)* (Kipf & Welling, 2017). Except the results of linear GNN which are from our reproduction, the results of other baselines align with the results reported in the OGB leaderboard, where the detailed implementation and hyperparameter setup can be found. [3] The LP algorithm reported in Tab. 1 follows the standard implementation that is ran until convergence, while the LP algorithm reported in Fig. 2 does not run until convergence in order to align with the proposed RP algorithm.

For hyperparamter search of RP, we adopt grid search for the RP algorithm with the step size η from $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1\}$, the power K ranging from 1 to 10. For `Arxiv`, `Proteins` and `Products`, K is chosen as 7, 1, 8 respectively. Since both LP and GRP are deterministic algorithms, their standard deviations are 0. All experiments are conducted on Quadro RTX 8000 with 48GB memory.

[3] https://ogb.stanford.edu/docs/leader_nodeprop

Table 3: Statistics of 17 datasets.

Type	Dataset	# Nodes (n)	# Edges (e)	# Features (d)	# Class (c)	# Tasks	Split
OGB	`Arxiv`	169,343	1,166,243	128	40	1	Public
	`Proteins`	132,534	39,561,252	8	2	112	Public
	`Products`	2,449,029	61,859,140	100	47	1	Public
Homophilic	`Cora`	2,708	10,556	1,433	7	1	Public
	`Citeseer`	3,327	9,104	3,703	6	1	Public
	`Pubmed`	19,717	88,648	500	3	1	Public
	`Computers`	13,752	491,722	767	10	1	80%/10%/10%
	`Photo`	7,650	238,162	745	8	1	80%/10%/10%
	`CS`	18,333	163,788	6,805	15	1	80%/10%/10%
	`Physics`	34,493	495,924	8,415	5	1	80%/10%/10%
Heterophilic	`roman-empire`	22662	32927	300	18	1	Public
	`amazon-ratings`	24,492	93,050	300	5	1	Public
	`minesweeper`	10,000	39,402	7	2	1	Public
	`tolokers`	11,758	519,000	10	2	1	Public
	`questions`	48,921	153,540	301	2	1	Public
	`Texas`	183	325	1,703	5	1	Public
Synthetic	`Synthetic`	2,000	8,023 $\sim$ 8,028	100	5	1	100/500/1000

D.2. Experiments in Section 6

We conduct experiments on real-world benchmark datasets `Cora` and `Texas`, and a synthetic dataset generated by the stochastic block model. For the synthetic dataset, we set number of blocks as 5 (i.e. number of classes) with each block having 400 nodes. Each node is associated with a 100-dimensional informative input feature vector. For the homophilic version of the dataset, the 5×5 edge probability matrix is defined as $\mathbf{P} = 0.01 \cdot \mathbf{I}_5$, i.e. nodes in the same block are connected with probability 0.01, and we gradually change this matrix in the generation process until there are only heterophilic edges left in the dataset, i.e. $\mathbf{P} = 0.0025 \cdot \mathbf{1}\mathbf{1}^\top - 0.0025 \cdot \mathbf{I}_5$. The statistics of these datasets are shown in Table 3. Note that we do not consider the large-scale datasets as in Section 3.2, since the computing NTK matrix is extremely costly in memory, especially for GNNs in node-level tasks where the output for an instance is also related to input features of other instances and mini-batch partitioning can not be directly adopted.

For the model, we choose a two-layer GCN with width 16, bias term and ReLU activation for all datasets. In order to control the variable, when comparing NTK of the model using different graph structures, we fix the weights in the model, which is achieved by training on a fixed graph structure but evaluate the NTK matrix (and test performance) using different graph structure. For real-world datasets, we use the original graph for training which is equivalent to the standard training, and for synthetic dataset, we use an identity matrix for training in which case the model is equivalent to the recently proposed PMLP (Yang et al., 2023) model that has shown to as effective as GNNs in the supervised learning setting. The optimization algorithm is gradient descent with learning rates $1e-2$, $3e-4$, $5e-5$ respectively for `Cora`, `Texas`, `Synthetic` respectively, momentum 0.9 and weight decay $5e-4$. The loss function is the standard cross-entropy loss for multi-class classification.

Since the NTK matrix for multi-dimensional output is a $nc \times nc$ matrix, where c is the output dimension, we follow prior work (see Mohamadi & Sutherland (2023) and references therein) and compute the $n \times n$ NTK matrix by averaging over the dimension c. The graph adjacency matrix we consider here is defined as $\mathbf{A}^4$ in order to align with our theoretical result for two-layer infinitely-wide GNN in Theorem 4.2. We also normalize these matrices before computing the alignment following (Cortes et al., 2012; Baratin et al., 2021) as a standard way of preprocess.

D.3. Algorithm Description

Recall the basic version of RP can be described by the following iterative forward propagation process:

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^K [\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t]\,. \tag{110}$$

The intial conditions are

$$[\mathbf{R}_0, \mathbf{R}'_0] = [\mathbf{Y} - \mathbf{F}_0, \mathbf{0} - \mathbf{F}'_0], \tag{111}$$

where $\mathbf{F}_0 = \mathbf{0}$ and $\mathbf{F}'_0 = \mathbf{0}$. For real-world applications where the outputs are usually multi-dimensional, say, multi-class classification with c classes, we define: $\mathbf{Y} = \{\boldsymbol{y}_i\}_{i=1}^{n_l} \in \mathbb{R}^{n_l \times c}$ where $\boldsymbol{y}_i$ is a onehot vector, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the same standard normalized adjacency matrix in GCN (Kipf & Welling, 2017). The pseudo code for the basic version of RP are shown in the following algorithm.

Algorithm 1 Basic version of residual propagation.

Input: Raw graph adjacency matrix $\mathbf{A}$, ground-truth labels for training samples $\mathbf{Y}$, step size η, power K.
Compute the normalized graph adjacency matrix by $\mathbf{A} \leftarrow \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{-\frac{1}{2}}$
Initialize $\mathbf{R}_0 \leftarrow \mathbf{Y} \in \mathbb{R}^{n_l \times c}$
Initialize $\mathbf{R}'_0 \leftarrow \mathbf{0} \in \mathbb{R}^{(n-n_l) \times c}$
while *Validation performance increases* **do**
  Label/residual propagation on the graph:
    $\tilde{\mathbf{R}}_{t-1} \leftarrow [\mathbf{R}_{t-1}, \mathbf{0}] \in \mathbb{R}^{n \times c}$
    for $i \leftarrow 1$ *to* K **do**
      $\tilde{\mathbf{R}}_{t-1} \leftarrow \mathbf{A}\, \tilde{\mathbf{R}}_{t-1}$
  Update residuals $[\mathbf{R}_t, \mathbf{R}'_t] \leftarrow [\mathbf{R}_{t-1}, \mathbf{R}'_{t-1}] - \eta \tilde{\mathbf{R}}_{t-1}$
Output prediction for testing samples $\mathbf{F}' = -\mathbf{R}'$

E. Additional Experiments

E.1. Incorporating Input Features

For smaller datasets where node features are often more useful (i.e. the dimension of node features is closer to the size of dataset, for example in `Citeseer`, the node feature dimension is even larger than the size of dataset), we consider the following generalized RP algorithm that combines kernel methods to leverage node feature information in the propagation process

$$\left[\mathbf{R}_{t+1}, \mathbf{R}'_{t+1}\right] = -\eta \mathbf{A}^K \mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) \mathbf{A}^K [\mathbf{R}_t, \mathbf{0}] + [\mathbf{R}_t, \mathbf{R}'_t], \quad \text{where } \mathbf{R}_0 = \mathbf{Y}, \mathbf{R}'_0 = \mathbf{0}, \tag{112}$$

where $\mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) \in \mathbb{R}^{n \times n}$ could be specified as arbitrary kernel functions (such as Sigmoid kernel, Gaussian kernel, etc.) that is applied to compute pairwise similarities. This variant of RP could also be treated as propagation on kernel's RKHS, i.e.

$$\mathbf{A}^K \mathbf{K}(\bar{\mathbf{X}}, \bar{\mathbf{X}}) \mathbf{A}^K = (\mathbf{A}^K \mathbf{K}(\bar{\mathbf{X}}, \cdot))(\mathbf{A}^K \mathbf{K}(\bar{\mathbf{X}}, \cdot))^\top, \tag{113}$$

which is impossible to directly implement in practice but can be achieved by our proposed RP.

With this implementation, RP can still run efficiently by treating the computation of the kernel matrix as a part of data prepossessing. Specifically, we will test Gaussian kernel which is defined as

$$\mathbf{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2}{2\sigma^2}\right), \tag{114}$$

while in practice one can treat the kernel function as a hyperparameter to tune for even better performance. The pseudo code for this version of RP are shown in the following algorithm.

Algorithm 2 Generalized residual propagation with kernel functions.

Input: Raw graph adjacency matrix $\mathbf{A}$, ground-truth labels for training samples $\mathbf{Y}$, input features $\mathbf{X}$ and $\mathbf{X}'$, step size η, power K.

Compute the normalized graph adjacency matrix by $\mathbf{A} \leftarrow \mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}}$

Compute the kernel matrix based on input features $\mathbf{K}([\mathbf{X},\mathbf{X}'],[\mathbf{X},\mathbf{X}'])$

Initialize $\mathbf{R}_0 \leftarrow \mathbf{Y} \in \mathbb{R}^{n_l \times c}$

Initialize $\mathbf{R}'_0 \leftarrow \mathbf{0} \in \mathbb{R}^{(n-n_l)\times c}$

while *Validation performance increases* **do**

  Label/residual propagation on the graph:

  $\tilde{\mathbf{R}}_{t-1} \leftarrow [\mathbf{R}_{t-1}, \mathbf{0}] \in \mathbb{R}^{n\times c}$

  for $i \leftarrow 1$ *to* K **do**

    $\tilde{\mathbf{R}}_{t-1} \leftarrow \mathbf{A}\,\tilde{\mathbf{R}}_{t-1}$

  $\tilde{\mathbf{R}}_{t-1} \leftarrow \mathbf{K}([\mathbf{X},\mathbf{X}'],[\mathbf{X},\mathbf{X}'])\,\tilde{\mathbf{R}}_{t-1}$

  for $i \leftarrow 1$ *to* K **do**

    $\tilde{\mathbf{R}}_{t-1} \leftarrow \mathbf{A}\,\tilde{\mathbf{R}}_{t-1}$

  Update residuals $[\mathbf{R}_t, \mathbf{R}'_t] \leftarrow [\mathbf{R}_{t-1}, \mathbf{R}'_{t-1}] - \eta\tilde{\mathbf{R}}_{t-1}$

Output prediction for testing samples $\mathbf{F}' = -\mathbf{R}'$

E.2. Homophilic Datasets

To evaluate the generalized RP algorithm in (112), we experiment on 7 more (smaller) datasets: `Cora`, `Citeseer`, `Pubmed`, `Computer`, `Photo`, `CS`, `Physics`. For `Cora`, `Citeseer`, `Pubmed`, we follow the public split, while for other datasets, we randomly split them into training/validation/testing sets based on ratio $8/1/1$. Statistics of these datasets are reported in Table. 3. For RP, we tune the hyperparameters K and σ. For baselines (Wu et al., 2019; Kipf & Welling, 2017; Xu et al., 2018; Klicpera et al., 2019), we tune the hyperparameters provided in their original paper and report mean and standard deviation of testing accuracy with 20 different runs.

The results are reported in Table. 4. We found the proposed RP almost always achieves the best or second best performance, and outperforms GCN in 6 out of 7 datasets even using no learnable parameters. In terms of the average performance, RP achieved the highest ranking out of all popular GNN models considered. Better performance can potentially be achieved by considering more advanced kernel functions or other specialized propagation matrices.

E.3. Heterophilic Datasets

For a comprehensive evaluation, we further consider 5 heterophilic benchmarks `roman-empire`, `amazon-ratings`, `minesweeper`, `tolokers`, `questions` from a recent paper Platonov et al. (2023), which has addressed some drawbacks of existing datasets used for evaluating models designed specifically for heterophily. We use 10 existing standard train/validation/test splits provided in their paper, and statistics of these datasets are also reported in Table. 3. Baselines are recently proposed strong GNN models that are carefully designed to tackle the heterophily problem.

The results are reported in Table. 5. We observed that the proposed RP maintains a strong level of performance when compared to these meticulously designed models, surpassing 6 out of the 9 in terms of average performance. It is important to note that RP has been proven to be suboptimal when applied to heterophilic graphs and is orthogonal to various other techniques designed for addressing this challenge. These observations suggest that there is significant untapped potential for further improvement of the algorithm to address this issue.

Table 4: Performance of generalized RP on homophilic datasets. We mark the first and second place with gold and silver, and compare its performance with GCN by Δ_{GCN}.

Model	Cora	Citeseer	Pubmed	Computer	Photo	CS	Physics	Avg.
MLP	59.7 ± 1.0	57.1 ± 0.5	68.4 ± 0.5	85.42 ± 0.51	92.91 ± 0.48	95.97 ± 0.22	96.90 ± 0.27	79.49
SGC	81.0 ± 0.5	71.9 ± 0.5	78.9 ± 0.4	89.92 ± 0.37	94.35 ± 0.19	94.00 ± 0.30	96.19 ± 0.13	86.61
GCN	81.9 ± 0.5	71.6 ± 0.4	79.3 ± 0.3	92.25 ± 0.61	95.16 ± 0.92	94.10 ± 0.34	96.64 ± 0.36	87.28
JKNet	81.3 ± 0.5	69.7 ± 0.2	78.9 ± 0.6	91.25 ± 0.76	94.82 ± 0.22	93.57 ± 0.49	96.31 ± 0.29	86.55
APPNP	82.6 ± 0.2	71.7 ± 0.5	80.3 ± 0.1	91.81 ± 0.78	95.84 ± 0.34	94.41 ± 0.29	96.84 ± 0.26	87.64
RP (Ours)	82.7 ± 0.0	73.0 ± 0.0	80.1 ± 0.0	92.00 ± 0.00	95.55 ± 0.00	94.60 ± 0.00	96.75 ± 0.00	87.81
Δ_{GCN}	+ 0.8	+ 1.4	+ 1.2	- 0.25	+ 0.39	+ 0.19	+ 0.11	+ 0.53

Table 5: Performance of generalized RP on heterophilic datasets. The first three datasets use Accuracy, and the last two datasets use ROC-AUC. We report the ranking of RP among all baselines.

Model	roman-empire	amazon-ratings	minesweeper	tolokers	questions	Avg.
ResNet	65.88 ± 0.38	45.90 ± 0.52	50.89 ± 1.39	72.95 ± 1.06	70.34 ± 0.76	61.39
H2GCN (Zhu et al., 2020)	60.11 ± 0.52	36.47 ± 0.23	89.71 ± 0.31	73.35 ± 1.01	63.59 ± 1.46	64.64
CPGNN (Zhu et al., 2021)	63.96 ± 0.62	39.79 ± 0.77	52.03 ± 5.46	73.36 ± 1.01	65.96 ± 1.95	59.02
GPR-GNN (Chien et al., 2021)	64.85 ± 0.27	44.88 ± 0.34	86.24 ± 0.61	72.94 ± 0.97	55.48 ± 0.91	64.88
FSGNN (Maurya et al., 2022)	79.92 ± 0.56	52.74 ± 0.83	90.08 ± 0.70	82.76 ± 0.61	78.86 ± 0.92	76.87
GloGNN (Li et al., 2022)	59.63 ± 0.69	36.89 ± 0.14	51.08 ± 1.23	73.39 ± 1.17	65.74 ± 1.19	57.35
FAGCN (Bo et al., 2021)	65.22 ± 0.56	44.12 ± 0.30	88.17 ± 0.73	77.75 ± 1.05	77.24 ± 1.26	70.50
GBK-GNN (Du et al., 2022)	74.57 ± 0.47	45.98 ± 0.71	90.85 ± 0.58	81.01 ± 0.67	74.47 ± 0.86	73.58
JacobiConv (Wang & Zhang, 2022)	71.14 ± 0.42	43.55 ± 0.48	89.66 ± 0.40	68.66 ± 0.65	73.88 ± 1.16	69.38
RP (Ours)	66.01 ± 0.56	47.95 ± 0.57	80.48 ± 0.76	78.05 ± 0.90	76.39 ± 1.16	69.78
Ranking	**4 / 10**	**2 / 10**	**7 / 10**	**3 / 10**	**3 / 10**	**4 / 10**