THE UNIVERSITY OF CHICAGO

GAUSSIAN PROCESSES AND NEURAL NETWORKS IN GRAPH LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF STATISTICS

BY
ZEHAO NIU

CHICAGO, ILLINOIS
DECEMBER 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Walking through the University of Chicago on a crisp autumn day, the breeze gently caresses my face, bringing back memories of the hopeful aspirations I held during orientation. The ancient trees, sprinkling golden leaves on the century-old main quad, have silently witnessed countless youthful endeavors and moments of growth. Amid the serene beauty of this historic campus, I wish to dedicate this dissertation to commemorate these unforgettable moments and those who accompanied me on this journey.

I want to express my heartfelt gratitude to my advisor, Prof. Mihai Anitescu. This experience would not have been so memorable without your academic guidance, personal care and charismatic presence. I want to appreciate Jie Chen for the inspiration and substantial contributions to my work. Your enthusiasm has encouraged me to push beyond my limits and strive for excellence. I would also like to thank my committee members for their suggestions and comments that helped me improve the quality of this paper.

I am sincerely indebted to my parents, the most steadfast backbone of my journey, allowing me to pursue my dreams. I am also grateful to my lab mates and friends, whose support has opened my heart and enriched my life. I owe my spiritual motivation to artistic masterpieces, such as of *Rachmaninov: Piano Concerto No. 2* performed by Richter and *Millennium Actress* by Satoshi Kon, during moments of melancholy.

The evening bell chimes, serenading our parting, as the sunset over the ivy-clad bell tower paints the sky in hues of nostalgic orange. I realize the time had come for fledglings to leave the nest and ripe fruits to fall from the tree. I only wish to remember the clear and profound sky of this place, so that whenever I look up in the future, the sky will remind me of these cherished feelings.

ABSTRACT

We introduce a novel family of Gaussian Processes (GPs) kernels, derived from the limit of graph neural networks (GNNs) when their layers become infinitely wide. We start with a prominent example of GNNs, the graph convolutional networks (GCNs), and establish the corresponding GP kernel. We conduct an in-depth analysis of the kernel universality and the behavior in the limit of depth. To address computational efficiency, we propose a low-rank approximation of the covariance matrix, enabling scalable posterior inference with large-scale graph data. Our experiments reveal that the kernel achieves competitive classification and regression performance on diverse graph datasets, while offering significant advantages in computation time and memory efficiency.

Expanding on this, we derive various members of this kernel family by extending the equivalence to several interesting GNN architectures. We consider their limits in depth, and demonstrate how to efficiently compose these kernels from the building blocks of GNNs. Our results show that these GNN-based kernels effectively capture rich graph information from graph-structured data, achieving superior performance compared to their GNN counterparts.

We further delve into the dynamic evolution of these kernels during the training process. Specifically, we introduce the Graph Neural Tangent Kernel (GNTK), representing the kernel obtained at the end of training. We show that the GNTK can be computed efficiently using our proposed methods and shares similar performance metrics with GNNGP kernels.

Finally, we conduct a comprehensive performance analysis, investigating how large GNNs scale with respect to model size, dataset size, and training cost. We derive and empirically validate the scaling laws governing these models, providing useful guidance for designing and training scalable GNNs for various graph applications.

CHAPTER 1

INTRODUCTION

1.1 Gaussian Processes

Machine learning is a branch of study that focus on using data and algorithms that enable machines to learn from data, extract knowledge and patterns from them and make predictions for new examples. Over the years, machine learning has become a powerful tool for solving complex problems in various domains, such as object recognition, representation learning and recommendation systems. With limited number of training examples and computational resources, the key challenge in machine learning is how to dealing with noisy, incomplete or heterogeneous data, while ensuring the robustness, efficiency and interpretability of the learned models.

Gaussian processes (GPs) are a powerful and attractive class of machine learning models, enabling us to make predictions about our data by incorporating prior knowledge. They are widely used in machine learning, uncertainty quantification, and global optimization [Rasmussen and Williams, 2006], because of their simplicity and flexibility as building blocks of more complex Bayesian models. For a given set of training points, there are potentially infinitely many functions that fitting the data. Gaussian processes assign a probability to each of these functions, with the mean of this probability distribution representing the most probable characterization of the data. Additionally, the probabilistic view of GPs allows us to estimate the confidence of the prediction result.

We give a brief introduction of the mathematical formula for Gaussian processes. Consider a machine learning scheme where we are interested in to fitting a function to the data. Our task is to model the underlying distribution from a dataset of N examples $X = \{x_i \in \mathbb{R}^d : i = 1, \cdots, N\}$ and their target labels $Y = \{y_i \in \mathbb{R} : i = 1, \cdots, N\}$. The label is either a real number (regression) or a categorical value (classification). The

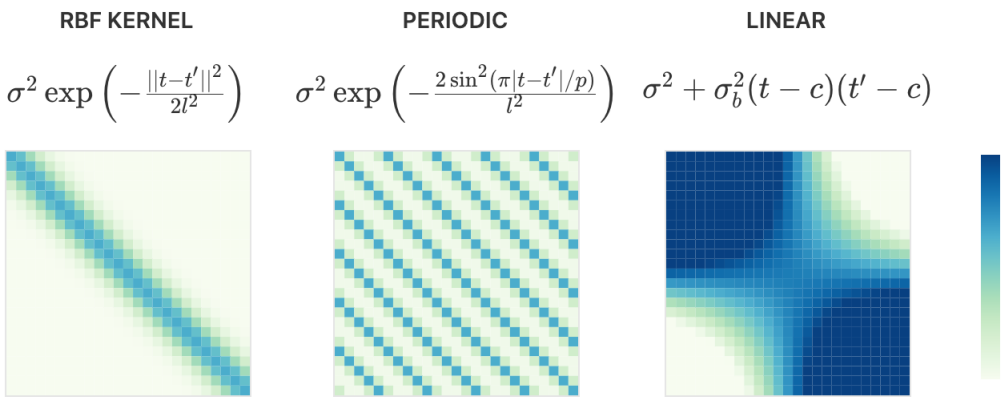
Gaussian distribution is characterized by a mean function μ and a covariance function k, with the following probability distribution:

$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x')), \quad \mu : \mathbb{R}^d \to \mathbb{R}, k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}. \tag{1.1}$$

We often assume that $\mu = 0$ to simplify the equations for our predictions: If $\mu \neq 0$, we can just center the data in preprocessing step and add back the resulting values of μ in the prediction. The covariance function k, also known as the *kernel*, determines the characteristics of the distribution and the statistical model $f(x)$ we want to predict: The function $f(x)$ follows a Gaussian distribution determined by its covariance matrix K, whose entries are generated by evaluating the kernel for each pairwise combination of all points:

$$Y = f(X) \sim \mathcal{N}(0, K), \quad K_{ij} = k(x_i, x_j). \tag{1.2}$$

The kernel describes the similarity between values of our function $f(x)$. Kernels are widely used in machine learning, particularly in support vector machines (SVMs), due to their ability to measure similarity beyond the standard Euclidean distance (L^2-distance). There are numerous kernels that can describe different classes of functions, which can be used to model the desired shape of the function. The following figure illustrates examples of some common kernels for Gaussian processes.



For each kernel, the covariance matrix is created from ($N = 25$) linearly spaced values ranging from $[-5, 5]$. Each entry in the matrix represents the covariance between points in the range of $[0, 1]$. Variance $\sigma = 0.8$, Length $l = 0.8$, Periodicity $p = 0.5$, Variance $\sigma_b = 0.3$, Offset $c = 0$.

Figure 1.1: Various kernels used in Gaussian processes. Taken from [Görtler et al., 2019].

We now shift our focus back to the original task of prediction. In supervised learning scheme, the goal of Gaussian processes is to learn the underlying distribution from *training* examples and generalize to unseen *test* examples and predict their labels correctly. We denote the training data as X_1 the test data as X_2. In Gaussian processes, each point is treated as a random variable, and the corresponding joint distribution of X_1 and X_2 is a N-dimensional Gaussian distribution $\mathcal{N}(0, K)$. In Bayesian inference, this is called the *prior* distribution. If no training examples have been observed, this distribution is symmetric around $\mu = 0$, according to our original assumption.

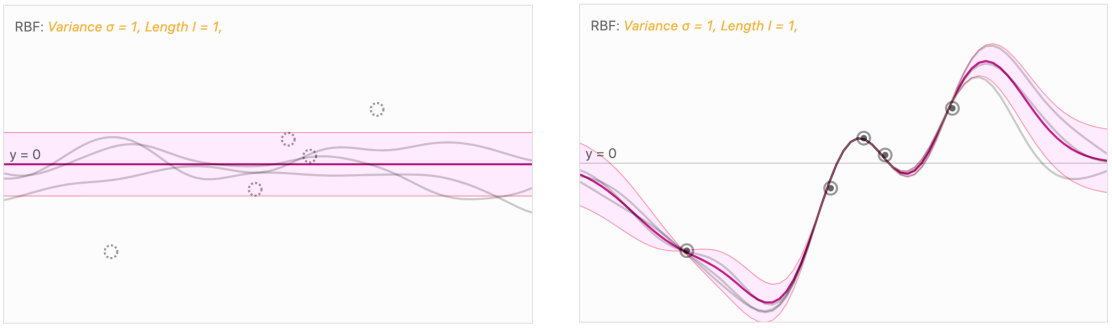
Once provided with the training data, we can incorporate this additional information into our model, yielding the posterior distribution $X_2|X_1$. We form the joint distribution between the test points and the training points as

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = X \sim \mathcal{N}(0, K) = \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}\right). \tag{1.3}$$

The Gaussian conditioning determines the *posterior* distribution as

$$X_2|X_1 \sim \mathcal{N}(K_{21}K_{11}^{-1}X_1, K_{22} - K_{21}K_{11}^{-1}K_{12}). \tag{1.4}$$

The result is also a multivariate Gaussian distribution. Unlike the prior distribution, the result of conditioning the joint distribution of test and training data will most likely have a non-zero mean and smaller covariance. The mean is used for prediction and the covariance serves as an uncertainty measure. The closed-form posterior allows for exact Bayesian inference, resulting in the great attractiveness and wide usage of GPs.



(Left) The prior distribution of a Gaussian process with a RBF kernel. (Right) The situation after datapoints have been observed. The mean prediction is shown as the pink line and three samples are shown as gray lines. In both plots the shaded region denotes twice the standard deviation at each input value x.

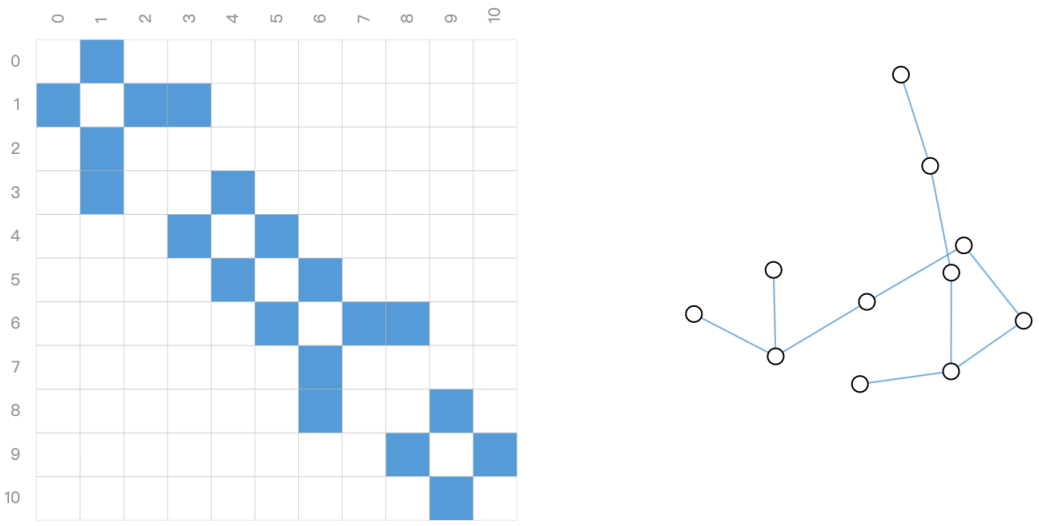
Figure 1.2: Prior vs. Posterior Distribution. Taken from [Görtler et al., 2019].

An example of GP prior and posterior distributions is illustrated in Figure 1.2, helping to develop a visual intuition of the flexibility of GPs. For the prior distribution, no training points have been observed, and accordingly the mean prediction remains at 0 with the same standard deviation for each test point. For the posterior distribution, the Gaussian conditioning precisely pass through each training point. The intuition behind is that if a test point lies on the training point, the prediction value of our function should be the same as the training point. The uncertainty of the prediction is smaller in regions closer to the

training point, suggesting that influence of observing a training point is limited locally.

1.2 Graph Learning

In this thesis, we focus on a particular type of data that is important in many applications: *Graph*-structured data. It is not uncommon to observe objects connected to other entities. The set of objects, and the connections between them, are naturally expressed as a graph. We first establish what a graph is.



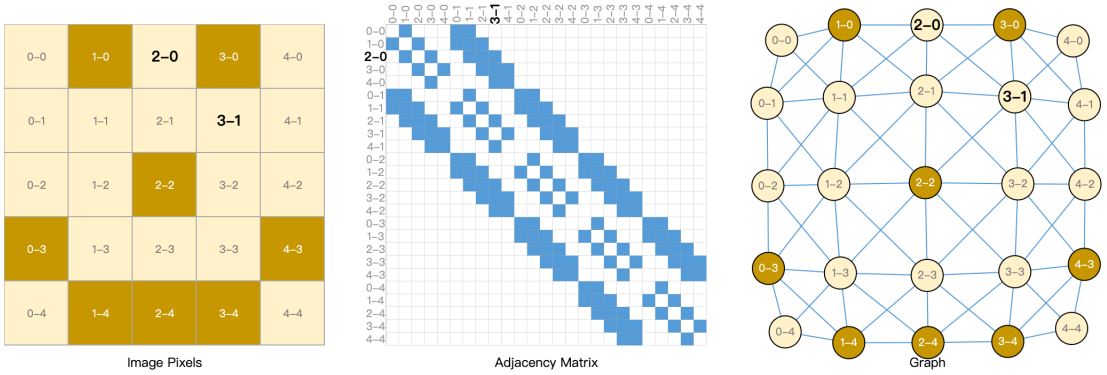
(Left) The *adjacency matrix* is a usual way of visualizing the connectivity of a graph, whose entries are indicators showing if two nodes share an edge. (Right) Graphs are mathematical structures consisting of a set of nodes, or vertices, and a set of edges that connect pairs of nodes. The edges can be directed, where an edge has a source node and a destination node. They can also be undirected, where there is no notion of source or destination nodes, and information flows both directions.

Figure 1.3: Definition of Graphs. Taken from [Sanchez et al., 2021].

Graphs are prominent in real world. For example, modern people are probably familiar with social networks, a type of graph data. We further illustrate several important types of data that can be modeled as graphs.

Images as Graphs. We typically represent an image as a rectangular grid of image pixels. Another way to think of images is a graphs with a regular structure, where each pixel

as a node connected via edges to adjacent pixels (neighbors). The RGB value information at each pixel is stored in a 3-dimensional attribute of the node representing the pixel. The graph representation also helps to build an intuition on how to generalize convolutions over images to convolutions over graphs, which we will discuss in the later section.



(Left) A simple example of a 5×5 image in a rectangular grid. (Right) The same image as a graph, where each node representing one of the 25 pixels in this case, and a nodes is connected via an edge to all its adjacent pixels. (Center) The 25×25 adjacency matrix of the graph. Note that each of these three representations are different views of the same data.

Figure 1.4: Graph representation of a smiley face image. Taken from [Sanchez et al., 2021].

Text as Graphs. Text can be viewed as a sequence of words (or tokens). We can associate a unique index with each word, thus creating a simple directed graph: each word is a node connected via an edge to the node that follows it. This graph representation is often used in Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models. Other models, such as Transformers, can be comprehended as constructing a fully connected graph between words and learn the relationship between them.

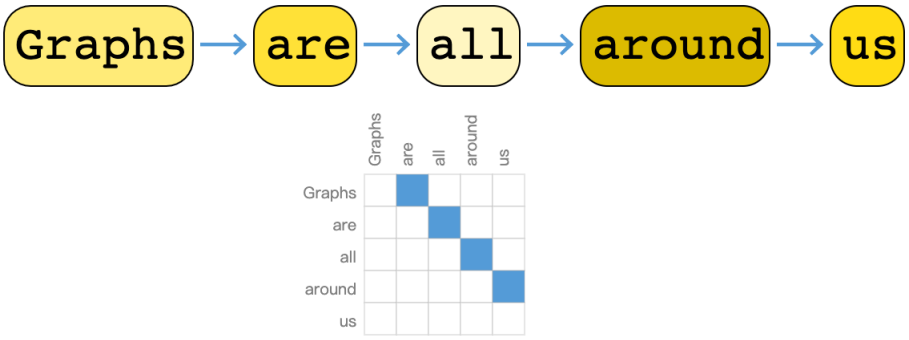
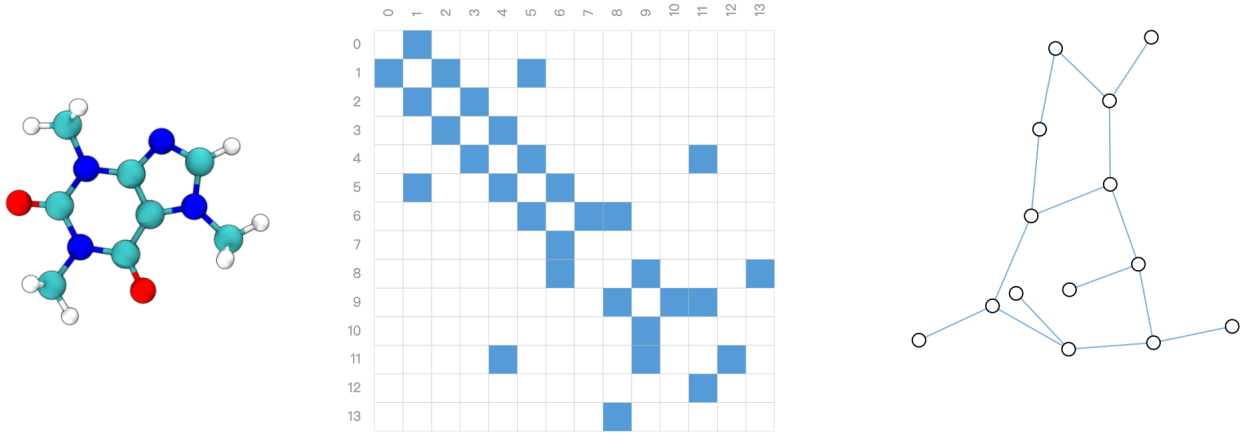


Figure 1.5: Graph representation of a sequence of tokens. Taken from [Sanchez et al., 2021].

Molecules as Graphs. Molecules, the fundamental units of matter, consist of atoms and electrons arranged in 3D space. When atoms maintain a stable distance, they form covalent bonds. These bonds vary in length depending on whether they are single, double, or triple bonds. Representing molecules as graphs is both convenient and common, with atoms as nodes and covalent bonds as edges. Below is an example of a molecule and its corresponding graph representation.

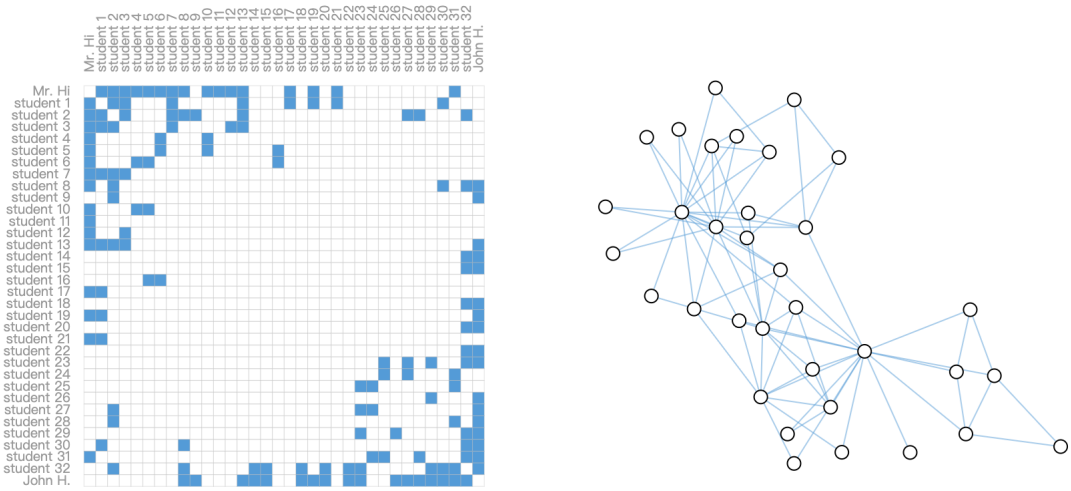


(Left) 3D representation of the molecule. (Center) Adjacency matrix of the molecule. (Right) Graph representation of the molecule.

Figure 1.6: Graph representation of the Caffeine molecule. Taken from [Sanchez et al., 2021].

Social Networks as Graphs. Social networks analyze patterns in the collective be-

havior of individuals, institutions, and organizations. We can represent these networks as graphs by modeling people as nodes and their interactions or relationships as edges.

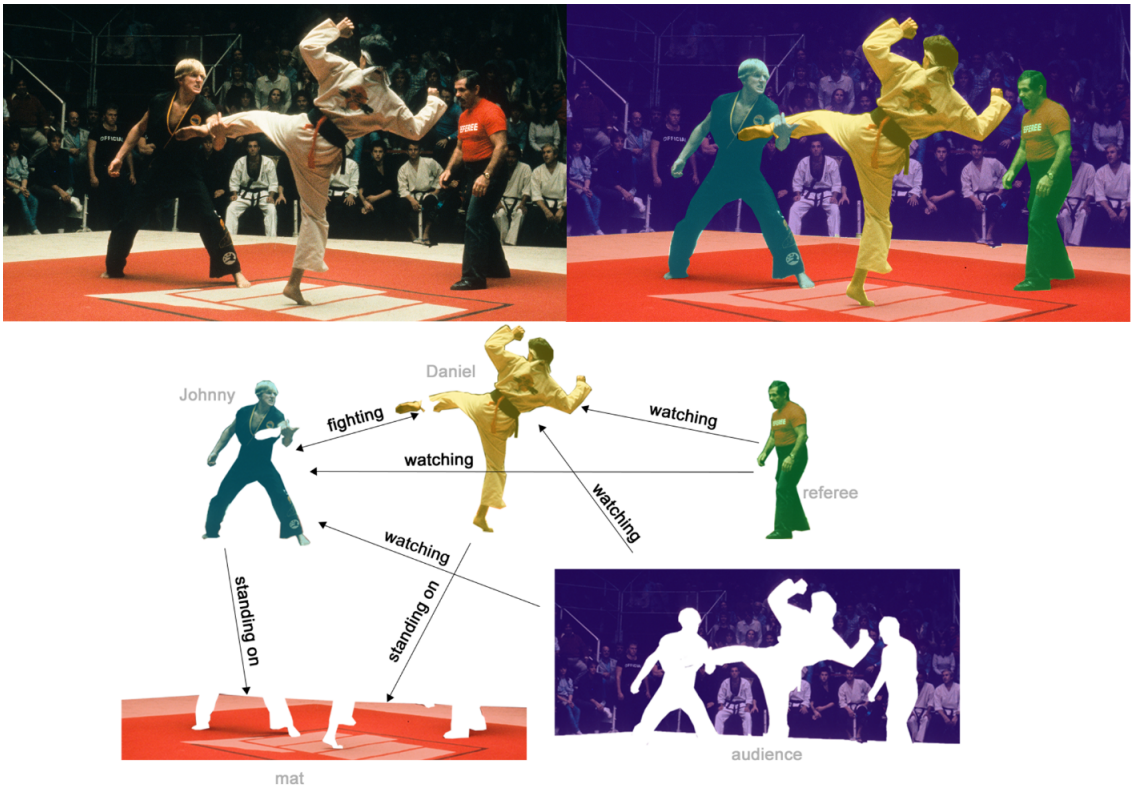


(Left) Adjacency matrix of the social network. (Right) Graph representation of the social network.

Figure 1.7: Graph representation of the Zach's karate club social network [Zachary, 1977]. Taken from [Sanchez et al., 2021].

Citation Networks as Graphs. In academic publishing, scientists frequently cite work from other scientists. These citation networks can be visualized as graphs, where each paper is a node and each citation is a directed edge connecting two nodes. Additional information, such as a word embedding of the abstract, can be included in each node to enrich the graph.

Dataflow as Graphs. In computer vision, labeling objects in visual scenes can be represented as graphs, with objects as nodes and their relationships as edges. Similarly, machine learning models, programming code, and mathematical equations can be expressed as graphs, where variables are nodes and operations are edges connecting these variables.



(Upper) The original image is segmented into five entities: each of the fighters, the referee, the audience and the mat. (Lower) The relationships between these entities represented in a graph.

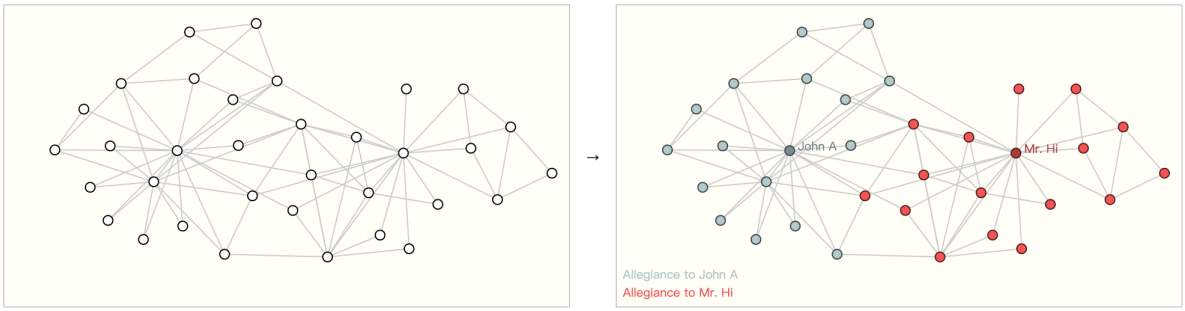
Figure 1.8: Graph representation of dataflow. Taken from [Sanchez et al., 2021].

1.3 Graph Neural Networks

In many real-world scenarios, labeled data is often limited or expensive, while unlabeled data is abundant and easily accessible. This thesis delves into the *semi-supervised learning* paradigm, where only a small amount of labeled data alongside a large amount of unlabeled data can be used together for training [Zhu, 2008]. In recent years, representation learning on graphs has emerged as a promising approach for this problem, when the labeled and unlabeled data are connected by a graph. For example, consider a vast social network with only a few users providing personal information, while we want to create personalized advertising effectively and ethically for everyone. In such situations, the graph structures

play a key role allowing us to leverage all available data, both labeled and unlabeled, to train a model with generalization capability.

In the thesis we focuses on *node-level tasks*, which are concerned with predicting the identity or role of each node within a graph. Fig 1.9 below demonstrates a classic example of node-level prediction problem on Zach's karate club dataset, as we saw in Fig 1.7. The prediction task is to classify each individual of the social network based on whether this member shows allegiance to either the instructor or the administrator after a political rift. Notably, the distance between a node and either the instructor or administrator correlates with this label.



(Left) Initial graph with unlabled nodes. (Right) Graph with each node classified based on the alliance. The dataset depicts a social network of individuals, with nodes represent individual karate practitioners and edges denote interactions outside karate. Following a feud between instructor (Mr. Hi) and administrator (John A.), the karate club splits. The task is to predict which one a member aligns with after the split.

Figure 1.9: Node-level prediction problem of the Zach's karate club social network [Zachary, 1977]. Taken from [Sanchez et al., 2021].

A significant challenge in graph-based learning is representing graphs in a manner compatible with machine learning. Traditional models typically require regular, grid-like input formats, but graphs are inherently lacking in fixed shapes or sizes, and vary widely in topology and complexity. This irregularity complicates the application of standard machine learning methods, as shown in 1.10.

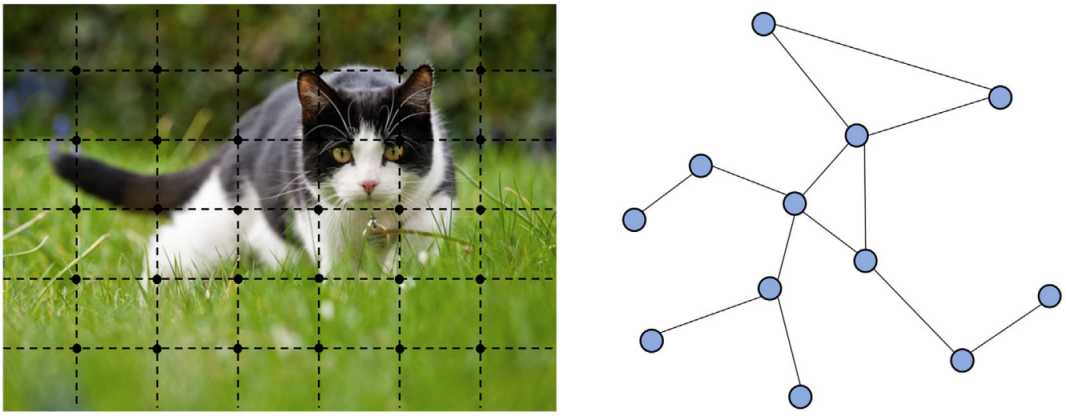
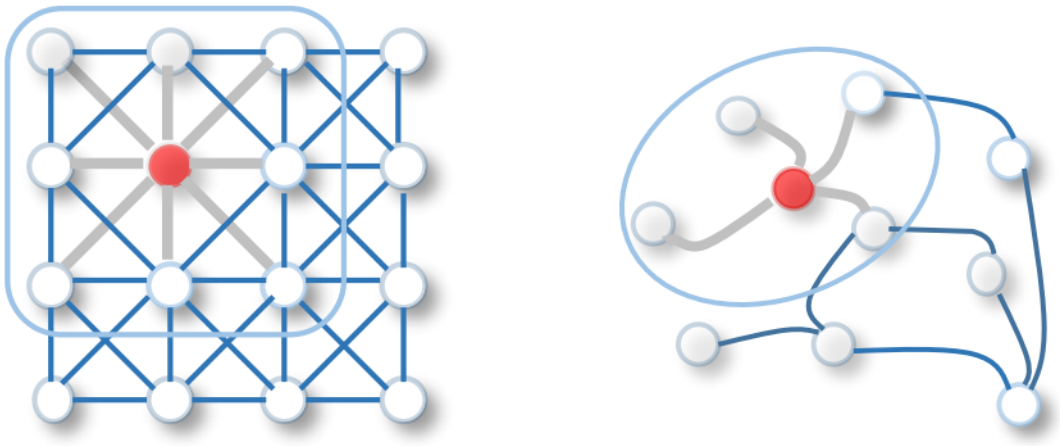


Figure 1.10: Left: image in Euclidean space. Right: graph in non-Euclidean space. Taken from [Zhou et al., 2020] Fig 1.

Graph Neural Networks (GNNs) have marked a groundbreaking milestone in machine learning on graphs [Zhou et al., 2020, Wu et al., 2021b], particularly in the context of semi-supervised learning. They are capable of learning meaningful representations of nodes by capturing both local and global information within the graph. This capability makes them highly effective for various downstream tasks, such as clustering, prediction and generative modelling. A common approach in many GNNs is the message-passing scheme [Gilmer et al., 2017], where information from neighboring nodes is aggregated to update the representation of a target node. Some notable examples include the Graph Convolutional Network (GCN) [Kipf and Welling, 2017], the Graph Attention Network (GAT) [Veličković et al., 2018], and the Graph Isomorphism Network (GIN) [Xu et al., 2019]. It is found that the performance of GNNs tends to degenerate as the networks become deeper. This issue is often addressed by incorporating techniques like residual or skip connections, as done by GraphSAGE [Hamilton et al., 2017], Jumping Knowledge [Xu et al., 2018], and GCNII [Chen et al., 2020].

GNNs operate on the principle of propagating and aggregating information along the edges of a graph. This concept can be viewed as a generalization of 2D convolution that takes the weighted average of a node's neighborhood information, as illustrated in Fig 1.11.



(Left) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size. (Right) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Figure 1.11: 2D Convolution vs. Graph Convolution. Taken from [Wu et al., 2021b] Fig 1.

The competitive performance of GNNs is often attributed to a proper choice of inductive bias, which helps models generalize from training data. However, there is a trade-off: a strong inductive bias might restrict the model's flexibility, limiting its ability to capture all patterns in the data and its generalization power, whereas a weak inductive bias could lead to overfitting, when the model becomes too complex and fits irrelevant details in the training data. Balancing these trade-offs is crucial for effective machine learning. The inductive bias of GNN models inspire us to design GP models under limited observations, by building the graph structure into the covariance kernel.

Another significant challenge in graph learning is designing efficient computational procedures [Daigavane et al., 2021]. From the example above, nodes in graphs typically connect to only a few others, and the number of edges per node can be highly variable, which results in sparse adjacency matrices. These sparse matrices, while space-efficient, can complicate

learning from the graph's global structure and the long-range dependencies. Benchmark graph datasets used in GNN research often contain thousands or even millions of labeled nodes [Hu et al., 2020b]. In a semi-supervised setting, this challenge is exacerbated, as the adjacency matrix needs to be recursively usde in evaluation of the graph convolution operation. Thus, designing efficient computational procedures for predictions (regression and classification) while ensuring scalability concerning the number of training data is essential.

1.4 Main Idea

In this thesis, we propose a new family of Gaussian Processes (GPs) kernels that are derived from the limit of graph neural networks (GNNs) when their layers are infinitely wide. We call these kernels Graph Neural Gaussian Processes (GNNGP) kernels. We show that these kernels can capture rich graph information from graph-structured data, and can be used for various semi-supervised learning tasks on graphs. We also propose efficient and scalable methods for computing these kernels and their derivatives, and analyze their properties and their behavior in depth. We demonstrate the effectiveness and the efficiency of these kernels on several graph datasets, and compare them with the state-of-the-art GNNs and GPs models. We also study the evolution of these kernels as we train the model on data, and introduce the Graph Neural Tangent Kernel (GNTK), which is the kernel obtained at the end of the training process. We show that the GNTK has similar properties and performance as the GNNGP kernels, and can be computed efficiently using our proposed methods. Finally, we perform a comprehensive performance analysis of these kernels, and investigate how they scale with respect to different factors, such as model size, dataset size, and training cost. We derive and empirically verify the scaling laws of these kernels, and provide useful guidance for designing and training scalable GNNs for various graph applications.

The rest of the thesis is organized as follows: Chapter 2 introduces the GNNGP kernels and their computation methods. Chapter 3 extends the GNNGP kernels to different archi-

tectures and combinations of GNNs. Chapter 4 introduces the GNTK and its computation methods. Chapter 5 presents the experimental results and the performance analysis of these kernels. Chapter 6 concludes the thesis and discusses the future work.

CHAPTER 2

GCNGP KERNELS

2.1 Introduction

An intimate relationship between neural networks and GPs is known: a neural network with fully connected layers, equipped with a prior probability distribution on the weights and biases, converges to a GP when each of its layers is infinitely wide [Lee et al., 2018, de G. Matthews et al., 2018]. Such a result is owing to the central limit theorem [Neal, 1994, Williams, 1996] and the GP covariance can be recursively computed if the weights and biases in each layer are iid Gaussian. Similar results for other architectures, such as convolution layers and residual connections, were subsequently established in the literature [Novak et al., 2019, Garriga-Alonso et al., 2019].

One focus of this work is to establish a similar relationship between GNNs and the limiting GPs. We will derive the covariance kernel that incorporates the graph inductive bias as GNNs do. We start with one of the most widely studied GNNs, the graph convolutional network (GCN) [Kipf and Welling, 2017], and analyze the kernel universality as well as the limiting behavior when the depth also tends to infinity.

In this section, we focus on a specific family of GPs, which we call GCNGP kernels, that are derived from the limit of GCNs. We first review the definition and the computation of GCNs, and then show how to obtain the corresponding NNGPs. We also analyze the properties of these NNGPs, such as their universality and their depth limit. We then present some experimental results on different graph datasets, and compare the performance and the efficiency of GCNGP kernels with those of GCNs.

Meanwhile, we design efficient computational procedures for posterior inference (i.e., regression and classification). GPs are notoriously difficult to scale because of the cubic complexity with respect to the number of training data. Benchmark graph datasets used

by the GNN literature may contain thousands or even millions of labeled nodes [Hu et al., 2020b]. The semi-supervised setting worsens the scenario, as the covariance matrix needs to be (recursively) evaluated in full because of the graph convolution operation. We propose a Nyström-like scheme to perform low-rank approximations and apply the approximation recursively on each layer, to yield a low-rank kernel matrix. Such a matrix can be computed scalably. We demonstrate through numerical experiments that the GP posterior inference is much faster than training a GNN and subsequently performing predictions on the test set.

2.2 Kernels from Graph Convolutional Networks

We start with a few notations used throughout this paper. Let an undirected graph be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes and $M = |\mathcal{E}|$ edges. For notational simplicity, we use $A \in \mathbb{R}^{N \times N}$ to denote the original graph adjacency matrix or any modified/normalized version of it. Using d_l to denote the width of the l-th layer, the layer architecture of GCN reads

$$X^{(l)} = \phi\left(AX^{(l-1)}W^{(l)} + b^{(l)}\right), \tag{2.1}$$

where $X^{(l-1)} \in \mathbb{R}^{N \times d_{l-1}}$ and $X^{(l)} \in \mathbb{R}^{N \times d_l}$ are layer inputs and outputs, respectively; $W^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ and $b^{(l)} \in \mathbb{R}^{1 \times d_l}$ are the weights and the biases, respectively; and ϕ is the ReLU activation function. The graph convolutional operator A is a symmetric normalization of the graph adjacency matrix with self-loops added [Kipf and Welling, 2017].

For ease of exposition, it will be useful to rewrite the matrix notation (2.1) in element-sums and products. To this end, for a node x, let $z_i^{(l)}(x)$ and $x_i^{(l)}(x)$ denote the pre- and post-activation value at the i-th coordinate in the l-th layer, respectively. Particularly, in an L-layer GCN, $x^{(0)}(x)$ is the input feature vector and $z^{(L)}(x)$ is the output vector. The layer

architecture of GCN reads

$$y_i^{(l)}(x) = \sum_{j=1}^{d_{l-1}} W_{ji}^{(l)} x_j^{(l-1)}(x), \tag{2.2}$$

$$z_i^{(l)}(x) = b_i^{(l)} + \sum_{v \in \mathcal{V}} A_{xv} y_i^{(l)}(v), \tag{2.3}$$

$$x_i^{(l)}(x) = \phi(z_i^{(l)}(x)). \tag{2.4}$$

The following theorem states that when the weights and biases in each layer are iid zero-mean Gaussians, in the limit on the layer width, the GCN output $z^{(L)}(x)$ is a multi-output Gaussian Process over the index x.

Theorem 1. *Assume $d_1, \cdots, d_{L-1}$ to be infinite in succession and let the bias and weight terms be independent with distributions*

$$b_i^{(l)} \sim \mathcal{N}(0, \sigma_b^2), \quad W_{ij}^{(l)} \sim \mathcal{N}(0, \sigma_w^2/d_{l-1}), \quad l = 1, \cdots, L. \tag{2.5}$$

Then, for each i, the collection $\{z_i^{(l)}(x)\} \in \mathbb{R}^N$ over all graph nodes x follows the Gaussian Process $\mathcal{GP}(0, K^{(l)})$, where the covariance matrix $K^{(l)}$ can be computed recursively by

$$C^{(l)} = \mathbb{E}_{z_i^{(l)} \sim \mathcal{N}(0, K^{(l)})}[\phi(z_i^{(l)})\phi(z_i^{(l)})^T], \qquad l = 1, \cdots, L, \tag{2.6}$$

$$K^{(l+1)} = \sigma_b^2 1_{N \times N} + \sigma_w^2 A C^{(l)} A^T, \qquad l = 0, \cdots, L-1. \tag{2.7}$$

with $C^{(0)}$ defined as

$$C^{(0)} = \frac{X^{(0)}(X^{(0)})^T}{d_0}. \tag{2.8}$$

All proofs of this paper are given in Appendix A. Note that different from a usual GP, which is a random function defined over a connected region of the Euclidean space, here

$z^{(L)}$ is defined over a discrete set of graph nodes. In the usual use of a graph in machine learning, this set is finite, such that the function distribution degenerates to a multivariate distribution. In semi-supervised learning, the dimension of the distribution, N, is fixed when one conducts transductive learning; but it will vary in the inductive setting because the graph will have new nodes and edges. One special care of a graph-based GP over a usual GP is that the covariance matrix will need to be recomputed from scratch whenever the graph alters.

Theorem 1 uses the same construction $C^{(0)}$ as traditional literature [Lee et al., 2018], the inner product that gives the covariance between two input nodes. Meanwhile, nothing prevents us using any positive-definite kernel alternatively. For example, we could use the squared exponential kernel [1]

$$C^{(0)}(x, x') = \exp\left(-\frac{1}{2}\sum_{j=1}^{d_0}\left(\frac{x_j - x'_j}{\ell_j}\right)^2\right). \tag{2.9}$$

Such flexibility in essence performs an implicit feature transformation as preprocessing.

2.3 Computation of Kernels

2.3.1 Computation in Kernel Matrices

Before deriving the approximation recursion, we note that for the ReLU activation ϕ, $C^{(l)}$ in (2.6) is the arc-cosine kernel with a closed-form expression:

$$C^{(l)}_{xx'} = \frac{1}{2\pi}\sqrt{K^{(l)}_{xx}K^{(l)}_{x'x'}}\left(\sin\theta^{(l)}_{xx'} + (\pi - \theta^{(l)}_{xx'})\cos\theta^{(l)}_{xx'}\right), \tag{2.10}$$

1. Here, we abuse the notation and use x in place of $x^{(0)}(x)$ in the inner product.

where

$$\theta_{xx'}^{(l)} = \arccos\left(\frac{K_{xx'}^{(l)}}{\sqrt{K_{xx}^{(l)} K_{x'x'}^{(l)}}}\right). \tag{2.11}$$

Hence, the main idea is that starting with recursive relationship in (2.6). Starting with $K^{(l)}$, we compute the closed-form expression of $C^{(l)}$ by using (2.10), and then obtain an approximation of $K^{(l+1)}$, and repeat. The procedure gives the following naive computation algorithm:

Algorithm 1 Naive computation

Require: $C^{(0)} \geq 0$

1: $K^{(0)} \leftarrow \sigma_b^2 1_{N\times N} + \sigma_w^2 A C^{(0)} A^T$
2: **for** $l = 0, \cdots, L-1$ **do**
3: **for** $x, x' \in V$ **do**
4: $C^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,K^{(l)})}[\phi(z(x))\phi(z(x'))]$
5: **end for**
6: $K^{(l+1)} \leftarrow \sigma_b^2 1_{N\times N} + \sigma_w^2 A C^{(l)} A^T$
7: **end for**

with time complexity $O(LMN)$ and space complexity $O(N^2)$. For larger datasets, the computation is the main computational bottleneck for GP posterior inference.

To reduce the computational costs, we resort to a low-rank approximation of $C^{(l)}$, from which we easily see that $K^{(l+1)}$ is also low-rank using (2.6). Starting with C, we may consider reducing the computation cost by using its by using the Nyström approximation [Drineas and Mahoney, 2005]:

$$C^{(l)} \approx \hat{C}^{(l)} = \begin{pmatrix} C_{aa}^{(l)} & C_{:a}^{(l)^T} \\ C_{:a}^{(l)} & C_{:a}^{(l)}(C_{aa}^{(l)})^{-1} C_{:a}^{(l)^T} \end{pmatrix}, \tag{2.12}$$

where subscript a denotes a set of N_a landmark points and : denotes all remaining rows/columns. Such an approximation requires only computation of terms $C_{aa}^{(l)}$ and $C_{:a}^{(l)}$, effectively a $N \times N_a$ submatrix. We may rewrite the formula in the Cholesky style as

$$C^{(l)} \approx \hat{C}^{(l)} = \mathrm{Chol}(C^{(l)})\,\mathrm{Chol}(C^{(l)})^T, \quad \mathrm{Chol}(C^{(l)}) := \begin{pmatrix} (C_{aa}^{(l)})^{1/2} \\ C_{:a}^{(l)}(C_{aa}^{(l)})^{-1/2} \end{pmatrix}. \tag{2.13}$$

Proceed with induction. Then, (2.6) leads to a Cholesky style approximation of $K^{(l+1)}$, denoted by $Q = \mathrm{Chol}(K) \in \mathbb{R}^{N \times (N_a+1)}$:

$$K^{(l+1)} \approx \hat{K}^{(l+1)} := Q^{(l+1)}{Q^{(l+1)}}^T, \quad Q^{(l+1)} = \begin{pmatrix} \sigma_w A\,\mathrm{Chol}(C^{(l)}) & \sigma_b 1_{N\times 1} \end{pmatrix}. \tag{2.14}$$

Clearly,

$${Q^{(l+1)}}{Q^{(l+1)}}^T = \sigma_b^2 1_{N\times N} + \sigma_w^2 A \hat{C}^{(l)} A^T, \tag{2.15}$$

completing the induction. The Nyström computation is summarized in Algorithm 2:

Algorithm 2 Nyström computation

Require: $C^{(0)} \geq 0$

$Q^{(0)} \leftarrow \begin{pmatrix} \sigma_w A\,\mathrm{Chol}(C^{(0)}) & \sigma_b 1_{N\times 1} \end{pmatrix}$

for $l = 0, \cdots, L-1$ **do**

  for $x \in V, x' \in V_a$ **do**

    $C^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,Q^{(l)}{Q^{(l)}}^T)}[\phi(z(x))\phi(z(x'))]$

  end for

  $Q^{(l+1)} \leftarrow \begin{pmatrix} \sigma_w A\,\mathrm{Chol}(C^{(l)}) & \sigma_b 1_{N\times 1} \end{pmatrix}$

end for

Overall, the time complexity is $O(LMN_a + LNN_a^2)$ and space complexity is $O(NN_a)$.

2.3.2 Scalable Computation in Posterior Inference

We start the exposition with the mean prediction. We compute the posterior mean

$$\hat{y}_c = \hat{K}_{cb}^{(L)}(\hat{K}_{bb}^{(L)} + \epsilon I)^{-1} y_b, \tag{2.16}$$

where the subscripts b and c denote the training set and the prediction set, respectively. Let there be N_b training nodes and N_c prediction nodes. The parameter ϵ, called the *nugget*, is the noise variance of the training data.

Using the Nyström approximation summarized in Algorithm 2, $K^{(L)}$ is approximated by a rank-(N_a+1) matrix $\hat{K}^{(L)} = Q^{(L)}Q^{(L)^T}$. Once it is formed, the $(N_b+N_c)\times N_b$ submatrix of $K^{(L)}$ for the task can be approximated by

$$\hat{K}_{cb}^{(L)} = Q_{ca}^{(L)}Q_{ba}^{(L)^T}, \quad \hat{K}_{bb}^{(L)} = Q_{ba}^{(L)}Q_{ba}^{(L)^T}. \tag{2.17}$$

And using the push-through identity, the posterior mean is computed as

$$\hat{y}_c = Q_{ca}^{(L)}Q_{ba}^{(L)^T}\left(Q_{ba}^{(L)}Q_{ba}^{(L)^T} + \epsilon I\right)^{-1} y_b = Q_{ca}^{(L)}\left(Q_{ba}^{(L)^T}Q_{ba}^{(L)} + \epsilon I\right)^{-1} Q_{ba}^{(L)^T} y_b, \tag{2.18}$$

where note that the matrix to be inverted has size $(N_a+1)\times(N_a+1)$, which is assumed to be significantly smaller than $N_b \times N_b$. Similarly, the posterior variance is

$$\hat{V}_{cc} = \hat{K}_{cc}^{(L)} - \hat{K}_{cb}^{(L)}\left(\hat{K}_{bb}^{(L)} + \epsilon I\right)^{-1}\hat{K}_{bc}^{(L)} \tag{2.19}$$

$$= Q_{ca}^{(L)}Q_{ca}^{(L)^T} - Q_{ca}^{(L)}Q_{ba}^{(L)^T}\left(Q_{ba}^{(L)}Q_{ba}^{(L)^T} + \epsilon I\right)^{-1} Q_{ba}^{(L)}Q_{ca}^{(L)^T} \tag{2.20}$$

$$= \epsilon Q_{ca}^{(L)}\left(Q_{ba}^{(L)^T}Q_{ba}^{(L)} + \epsilon I\right)^{-1} Q_{ca}^{(L)^T}. \tag{2.21}$$

The computational costs of $Q^{(L)}$ and the posterior inference from (2.18) to (2.19) are summarized in Table 2.1.

Table 2.1: Computational costs for GCNGP.

M: number of edges; N: number of nodes; $N_a/N_b/N_c$: number of landmark / training / prediction nodes; L: number of layers. Assume $N_a \ll N_b, N_c$. For posterior variance, assume only the diagonal is needed.

	Time $O(\cdot)$	Storage $O(\cdot)$
Kernel computation	$LMN_a + LNN_a^2$	NN_a
Posterior mean (2.18)	$N_cN_a + N_bN_a^2$	$(N_b + N_c)N_a$
Posterior variance (2.19)	$N_cN_a + N_bN_a^2$	$(N_b + N_c)N_a$

2.4 Properties

2.4.1 Universality

A covariance kernel is positive definite; hence, the Moore-Aronszajn theorem [Aronszajn, 1950] suggests that it defines a unique Hilbert space for which it is a reproducing kernel. If this space is dense, then the kernel is called *universal*. One can verify universality by checking if the kernel matrix is positive definite for any set of distinct points. [2]

For the case of graphs, it suffices to verify if the covariance matrix for all nodes is positive definite. We note (again) that for the ReLU activation ϕ, $C^{(l)}$ in (2.6) is the arc-cosine kernel with a closed-form expression as a function of the angle between x and x', hence named the *arc-cosine* kernel [Cho and Saul, 2009].

It is also known that the expectation in (2.6) is proportional to the arc-cosine kernel up to a factor $\sqrt{K^{(l)}(x,x)K^{(l)}(x',x')}$ [Lee et al., 2018]. Therefore, we iteratively work on the post-activation covariance and the pre-activation covariance (2.6) and show that the covariance kernel resulting from the limiting GCN is universal, for any GCN with two or more layers.

2. Note the conventional confusion in terminology between functions and matrices: a kernel function is positive definite (resp. strictly positive definite) if the corresponding kernel matrix is positive semi-definite (resp. positive definite) for any collection of distinct points.

Theorem 2. *Assume A is invertible and non-negative, and $X^{(0)}$ does not contain two linearly dependent rows. Then, $K^{(l)}$ is positive definite for all $l \geq 2$.*

2.4.2 Limit in the Depth

The depth of a neural network exhibits interesting behaviors. Deep learning tends to favor deep networks because of their empirically outstanding performance, exemplified by generations of convolutional networks for the ImageNet classification [Krizhevsky et al., 2012, Wortsman et al., 2022]; while graph neural networks are instrumental to be shallow because of the over-smoothing and over-squashing properties [Li et al., 2018, Topping et al., 2022].

For multi-layer perceptrons (networks with fully connected layers), several previous works have noted that the recurrence relation of the covariance kernel across layers leads to convergence to a fixed-point kernel, when the depth $L \to \infty$ [Lee et al., 2018]. We elaborate this limit as follows:

Theorem 3. *(Limit for NNGP)*
Let $\delta = \frac{1}{2}\sigma_w^2$. The following results hold as $l \to \infty$.

1. *When $\sigma_b^2 = 0$, $\rho_{\min}(K^{(l)}) \nearrow 1$, where $\rho_{\min}$ denotes the minimum correlation between any two nodes x and x'.*
2. *When $\delta < 1$, the $K^{(l)} \to q1_{N\times N}$ where $q := \frac{\sigma_b^2}{1-\sigma_w^2/2}$ linearly.*
3. *When $\delta > 1$, the scaled $\kappa^{(l)} := K^{(l)}/\delta^l \to vv^T$ where $v_x := \sqrt{\frac{\sigma_b^2}{\sigma_w^2/2-1} + K^{(0)}(x,x)}$.*

The NNGP can be seen as a special case of GCNGP with adjacency matrix $A = I$. We offer the parallel analysis for GCNGP with *primitive*, i.e. irreducible aperiodic non-negative matrix A.

Theorem 4. *(Limit for GCNGP)*
Assume A is a symmetric primitive matrix, with Perron-Frobenius eigenvalue $r > 0$ and corresponding eigenvector v. Let $\delta = \frac{1}{2}\sigma_w^2 r^2$. The following results hold as $l \to \infty$.

1. *When* $\sigma_b^2 = 0$, $\rho_{\min}(K^{(l)}) \nearrow 1$.

2. *When* $\delta < 1$, $K^{(l)}$ *converges linearly with a rate of convergence of* $\delta^{1/2}$.

3. *When* $\delta > 1$, *the scaled* $\kappa^{(l)} := K^{(l)}/\delta^l \to cvv^T$ *for some constant* c.

A few remarks follow. The first case implies that the correlation matrix converges monotonously to a matrix of all ones. As a consequence, up to some scaling c_l that may depend on l, the scaled covariance matrix $K^{(l)}/c_l$ converges to a rank-1 matrix. The third case shares a similar result, with the scaling and limit explicitly spelled out. The second case is challenging to analyze, and we note convergence of $K^{(l)}$ to a unique fixed point, dependent only on A, σ_b^2 and σ_w^2.

2.5 Experiments

In this section, we conduct a comprehensive set of experiments to evaluate the performance of the GP kernels derived by taking limits on the layer width of GCN and other GNNs. We demonstrate that these GPs are comparable with GNNs in prediction performance, while being significantly faster to compute. We also show that the low-rank version scales favorably, suitable for practical use.

2.5.1 Datasets

The experiments are conducted on several benchmark datasets of varying sizes, covering both classification and regression. They include

1. predicting the topic of scientific papers organized in a citation network (Cora, Citeseer, PubMed and ArXiv);

2. predicting the community of online posts based on user comments (Reddit);

3. predicting the average daily traffic of Wikipedia pages using hyperlinks among them (Chameleon, Squirrel, and Crocodile).

The datasets Cora/Citeseer/PubMed/Reddit are downloaded from the PyTorch Geometric library [Fey and Lenssen, 2019] and used as is. The dataset ArXiv comes from the Open Graph Benchmark [Hu et al., 2020b]. The datasets Chameleon/Squirrel/Crocodile come from MUSAE [Rozemberczki et al., 2021]. A summary of the datasets is given in Table 2.2, and additional preprocessing details are given in Appendix B.

Table 2.2: Dataset Statistics.

The edge column lists the number of directed edges.

Dataset	Task	Nodes	Edges	Features	Train/Val/Test Ratio
Cora	Classification	2,708	5,429	1,433	0.05/0.18/0.37
Citeseer	Classification	3,327	4,732	3,703	0.04/0.15/0.30
PubMed	Classification	19,717	44,338	500	0.003/0.025/0.051
ArXiv	Classification	169,343	1,166,243	128	0.54/0.18/0.28
Reddit	Classification	232,965	11,606,919	602	0.66/0.10/0.24
Chameleon	Regression	2,277	36,101	3,132	0.48/0.32/0.20
Squirrel	Regression	5,201	217,073	3,148	0.48/0.32/0.20
Crocodile	Regression	11,631	180,020	13,183	0.48/0.32/0.20

2.5.2 Methodology

We first conduct the semi-supervised learning tasks on all datasets by using GCN and GPs with different kernels. These kernels include the one equivalent to the limiting GCN (GCNGP), a usual squared-exponential kernel (RBF), and the GGP kernel proposed by [Ng et al., 2018]. [3] Each of these kernels has a low-rank version (suffixed with -X). RBF-X and

3. We apply only the kernel but not the likelihood nor the variational inference used in [Ng et al., 2018], for reasons given in Appendix B.

GGP-X [4] use the Nyström approximation, consistent with GCNGP-X.

For all datasets except Reddit, we train GCN using the Adam optimizer in full batch for 100 epochs. Reddit is too large for GPU computation and hence we conduct mini-batch training. Hyperparameter tuning is done on the validation set. For additional hyperparameter details, see Appendix B.

GPs are by nature suitable for regression. For classification tasks, we use the one-hot representation of labels to set up a multi-output regression. Then, we take the coordinate with the largest output as the class prediction. Such an ad hoc treatment is widely used in the literature, as other more principled approaches (such as using the Laplace approximation on the non-Gaussian posterior) are too time-consuming for large datasets, meanwhile producing no noticeable gain in accuracy.

2.5.3 Results

Table 2.3 summarizes the accuracy for classification and the coefficient of determination, R^2, for regression. Whenever randomness is involved, the performance is reported as an average over ten runs. The results of the two tasks show different patterns. For classification, GCNGP(-X) is sligtly better than GCN and GGP(-X), while RBF(-X) is significantly worse than all others; moreover, the low-rank version is outperformed by using the full kernel matrix. On the other hand, for regression, GCNGP(-X) significantly outperforms GCN, RBF(-X), and GGP(-X); and the low-rank version becomes better. The less competitive performance of RBF(-X) is expected, as it does not leverage the graph inductive bias. It is attractive that GCNGP(-X) is competitive with GCN.

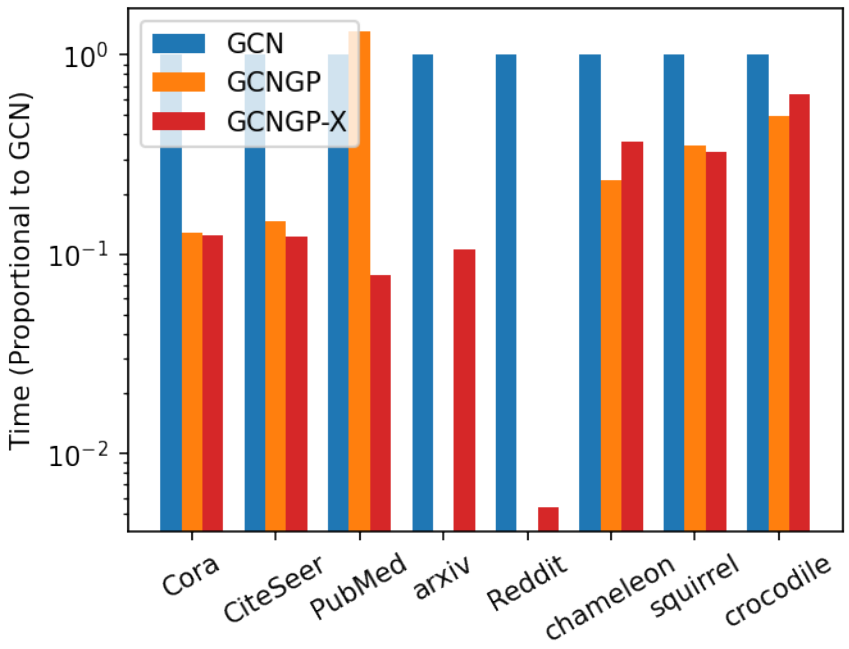
4. GGP-X in our notation is the Nyström approximation of the GGP kernel, different from a technique under the same name in [Ng et al., 2018], which uses additionally the validation set to compute the prediction loss.

Table 2.3: Performance of GCNGP in comparison with GCN and typical GP kernels.

The Micro-F1 score is reported for classification tasks and R^2 is reported for regression tasks.

Dataset	GCN	GCNGP	GCNGP-X	RBF	RBF-X	GGP	GGP-X
Cora	0.8183±0.0055	**0.8280**	0.7980	0.5860	0.5850	0.7850	0.7410
Citeseer	0.6941±0.0079	**0.7090**	0.7080	0.6120	0.6090	0.7060	0.6470
PubMed	0.7649±0.0058	**0.7960**	0.7810	0.7360	0.7340	0.7820	0.7380
ArXiv	0.6990±0.0014	OOM	**0.7011±0.0011**	OOM	0.5382	OOM	0.6527
Reddit	0.9330±0.0006	OOM	**0.9465±0.0003**	OOM	0.5920	OOM	0.9058
Chameleon	0.5690±0.0376	0.6720	**0.6852**	0.5554	0.5613	0.5280	0.5311
Squirrel	0.4243±0.0393	0.4926	**0.4998**	0.3187	0.3185	0.2440	0.2251
Crocodile	0.6976±0.0323	0.8002	**0.8013**	0.6643	0.6710	0.6952	0.6810

We compare the running time of the methods covered by Table 2.3. Different from usual neural networks, the training and inference of GNNs do not decouple in full-batch training. Moreover, there is not a universally agreed split between the training and the inference steps in GPs. Hence, we compare the total time for each method.



For each dataset, the computation times are normalized against that of GCN.

Figure 2.1: Timing comparison.

Figure 2.1 plots the timing results, normalized against the GCN time for ease of comparison. It suggests that GCNGP(-X) is generally faster than GCN. Note that the vertical axis is in the logarithmic scale. Hence, for some of the datasets, the speedup is even one to two orders of magnitude.

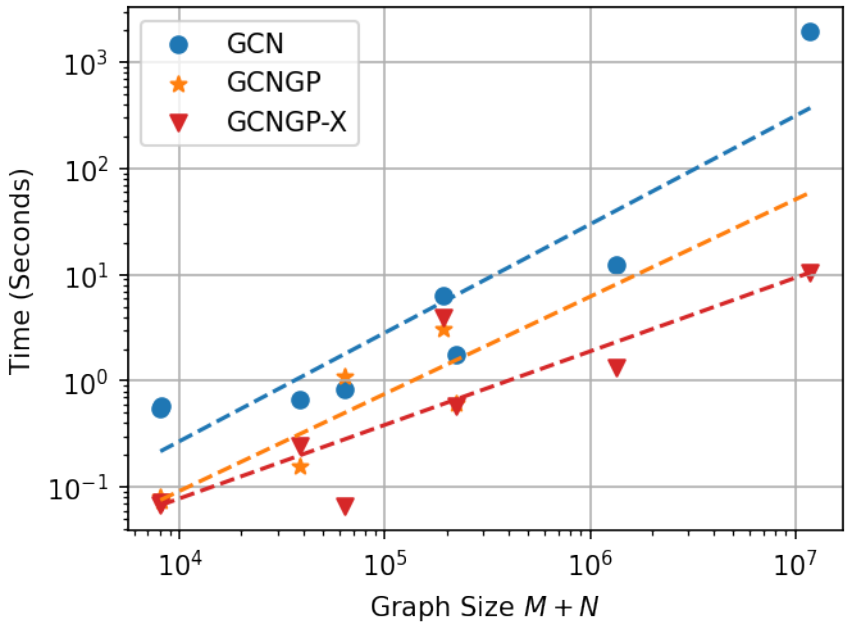


Figure 2.2: Scaling of the running time.

For graphs, especially under the semi-supervised learning setting, the computational cost of a GP is much more complex than that of a usual one (which can be simply described as cubic in the training set size). One sees in Table 2.1 the many factors that determine the cost of our graph-based low-rank kernels. To explore the practicality of the proposed method, we use the timings gathered for Figure 2.1 to obtain an empirical scaling with respect to the graph size, $M + N$.

Figure 2.2 fits the running times with respect to the graph size $M + N$, plotted in the log-log scale, by using a straight line. We see that for neither GCN nor GCNGP(-X), the actual running time closely follows a polynomial complexity. However, interestingly, the least-squares fittings all lead to a slope approximately 1, which agrees with a linear cost. Theoretically, only GCNGP-X and GCN are approximately linear with respect to $M + N$, while GCNGP is cubic.

Chapter 2, in part, is a reprint of the material as it appears in [Niu et al., 2023].

CHAPTER 3

GNNGP KERNELS: EXTENSIONS FROM GCNGP

3.1 Introduction

Graph Neural Networks (GNNs) have emerged as a powerful paradigm for learning on graph-structured data, enabling a wide range of applications from social network analysis to molecular biology [Zhou et al., 2020, Wu et al., 2021b]. While the Graph Convolutional Network (GCN) architecture has been a significant milestone in this domain, researchers have continued to explore various architectural innovations to overcome its limitations and improve performance. In this section, we delve into three such notable variations: GCN with skip/residual connections (GCNII), GraphSAGE, and Graph Attention Networks (GAT).

GCNII: Extending GCN with Skip Connections. The GCNII architecture introduces skip connections, also known as residual connections, to the standard GCN framework [Chen et al., 2020]. This enhancement addresses the over-smoothing problem that often plagues deep GCN models, where repeated graph convolution operations lead to indistinguishable node representations. By incorporating initial residual and identity mapping, GCNII allows for the construction of deeper models without the loss of representational power, thereby preserving the discriminative node features across layers. GCNII shows its effectiveness in semi-supervised tasks across various datasets in accuracy and robustness.

GraphSAGE: Inductive Learning on Large Graphs. GraphSAGE stands out as an inductive framework that generates node embeddings by sampling and aggregating features from a node's local neighborhood [Hamilton et al., 2017]. Unlike transductive methods that require all nodes to be present during training, GraphSAGE can generalize to unseen nodes, making it particularly suitable for dynamic graphs. By learning a function that aggregates local neighborhood information, GraphSAGE can efficiently produce embeddings for new nodes, facilitating scalable and generalizable graph learning.

GAT: Leveraging Attention Mechanisms in GNNs. Graph Attention Networks (GAT) leverage masked self-attentional layers to weigh the importance of neighboring nodes differently, allowing for a more nuanced aggregation of neighborhood information [Veličković et al., 2018]. This approach enables nodes to attend over their neighborhoods' features selectively, implicitly assigning different weights to different nodes in a neighborhood. GATs have demonstrated state-of-the-art results across various graph benchmarks, both transductive and inductive, showcasing their ability to adaptively learn from the graph structure.

Each of these architectural variations represent significant strides in the evolution of GNNs, introducing novel mechanisms to capture and leverage graph-structured data more effectively. By exploring the unique properties and applications of these advanced architecture variations, we gain deeper insights into the design and optimization of GNNs for a multitude of graph-based learning tasks.

3.2 Composing Graph Neural Network-Inspired Kernels

In Chapter 2, we show the equivalence between GCNs and GCNGPs in the limit of infinite width in Theorem 1, which suggests the covariance matrix of the limiting GP can be computed in a composable manner. Moreover, we also derived the low-rank approximation of the covariance matrix in Algorithm 2, indicating they can be similarly composed. Altogether, such a nice property allows one to easily derive the corresponding covariance matrix and its approximation for a new GNN architecture, like writing a program and obtaining a transformation of it automatically through operator overloading [Novak et al., 2020]: the covariance matrix is a transformation of the GNN and the composition of the former is in exactly the same manner and order as that of the latter. We call the covariance matrices *programmable*.

For example, we write a GCN layer as $X \leftarrow A\phi(X)W + b$, where for notational simplicity, X denotes pre-activation rather than post-activation as in the earlier sections. The activation

ϕ on X results in a transformation of the kernel matrix K into $g(K)$, defined as:

$$C = g(K) := \mathbb{E}_{z\sim\mathcal{N}(0,K)}[\phi(z)\phi(z)^T], \tag{3.1}$$

due to (2.6). Moreover, if K admits a low-rank approximation QQ^T where

$$Q = \text{Chol}(K) := K_{:a}K_{aa}^{-1/2}. \tag{3.2}$$

then $g(K)$ admits a low-rank approximation $\text{Chol}(g(QQ^T))$.

The next operation, graph convolution, multiplies A to the left of the post-activation. Correspondingly, the covariance matrix K is transformed to AKA^T and the low-rank approximation factor Q is transformed to AQ. Then, the operation, multiplying the weight matrix W to the right, will transform K to $\sigma_w^2 K$ and Q to $\sigma_w Q$. Finally, adding the bias b will transform K to $K + \sigma_b^2 1_{N\times N}$ and Q to $\begin{pmatrix} Q & \sigma_b 1_{N\times 1}\end{pmatrix}$. Altogether, we have obtained the following updates per layer:

$$\text{GCN}: \quad X \leftarrow A\phi(X)W + b \tag{3.3}$$

$$K \leftarrow \sigma_w^2 A g(K) A^T + \sigma_b^2 1_{N\times N} \tag{3.4}$$

$$Q \leftarrow \begin{pmatrix} \sigma_w A \,\text{Chol}(g(QQ^T)) & \sigma_b 1_{N\times 1}\end{pmatrix}. \tag{3.5}$$

One may verify the K update against Theorem 1 and the Q update against Algorithm 2. Both updates can be automatically derived based on the update of X.

The programmable property arises from the correspondence between neural network building blocks and kernel operations, allowing for the design of kernels that reflect the inductive biases of their GNN counterparts. To elucidate this concept, we present Table 3.1 that maps common neural network components to their analogous kernel operations.

Table 3.1: Neural network building blocks, kernel operations, and the low-rank counterpart.

Building block	Neural network	Kernel operation	Nyström operation
Input	$X \leftarrow X^{(0)}$	$K \leftarrow C^{(0)}$	$Q \leftarrow \mathrm{Chol}(C^{(0)})$
Bias term	$X \leftarrow X + b$	$K \leftarrow K + \sigma_b^2 1_{N\times N}$	$Q \leftarrow \begin{pmatrix} Q & \sigma_b 1_{N\times 1} \end{pmatrix}$
Weight term	$X \leftarrow XW$	$K \leftarrow \sigma_w^2 K$	$Q \leftarrow \sigma_w Q$
Graph convolution	$X \leftarrow AX$	$K \leftarrow AKA^T$	$Q \leftarrow AQ$
Activation	$X \leftarrow \phi(X)$	$K \leftarrow g(K)$	$Q \leftarrow \mathrm{Chol}(g(QQ^T))$
Independent sum	$X \leftarrow X_1 + X_2$	$K \leftarrow K_1 + K_2$	$Q \leftarrow \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$
Skip connection	$X \leftarrow \beta_1 X + \beta_2 XW$	$K \leftarrow (\beta_1^2 + \beta_2^2\sigma_w^2)K$	$Q \leftarrow \sqrt{\beta_1^2 + \beta_2^2\sigma_w^2}Q$
Mean aggregation	$X \leftarrow XW_1 + AXW_2$	$K \leftarrow \sigma_{w_1}^2 K + \sigma_{w_2}^2 AKA^T$	$Q \leftarrow \begin{pmatrix} \sigma_{w_1} Q & \sigma_{w_2} AQ \end{pmatrix}$
Self attention	$X \leftarrow \zeta(d^{-1}XX^T)X$	$K \leftarrow \zeta(K)K\zeta(K)^T$	$Q \leftarrow \zeta(QQ^T)Q$

Using this framework, we can understand the construction of GCNGP and GINGP kernels. For another example of the composability, we consider the popular GIN layer [Xu et al., 2019], which we assume uses a 2-layer MLP after the neighborhood aggregation:

$$\text{GIN}: \quad X \leftarrow A\phi(\phi(X)W + b)W' + b' \tag{3.6}$$

$$K \leftarrow \sigma_w^2 A g(K')A^T + \sigma_b^2 1_{N\times N}, \qquad K' = \sigma_w^2 g(K) + \sigma_b^2 1_{N\times N} \tag{3.7}$$

$$Q \leftarrow \begin{pmatrix} \sigma_w A \,\mathrm{Chol}(g(Q'(Q')^T)) & \sigma_b 1_{N\times 1} \end{pmatrix}, \quad Q' = \begin{pmatrix} \sigma_w \,\mathrm{Chol}(g(QQ^T)) & \sigma_b 1_{N\times 1} \end{pmatrix}. \tag{3.8}$$

Building on the established concepts, we extend the idea to three novel variations of GNNs: GCNII, GraphSAGE, and GAT in the following section. By translating these architectural innovations into kernel operations, we construct GNNGP kernels that retain the expressive power of their GNN inspirations while also benefit from the computational efficiency and probabilistic nature of GPs. This approach paves the way for designing and

training scalable GNNs that can effectively tackle a wide range of graph-based learning tasks.

3.3 Variations in Architectures

3.3.1 Skip Connections

The GCNII architecture extends the standard GCN by incorporating initial residual connections and identity mapping to alleviate the over-smoothing problem [Chen et al., 2020]. The layer-wise propagation rule for GCNII is given by:

$$X^{(l)} = \Big((1-\alpha)A\phi(X^{(l-1)}) + \alpha X^{(0)}\Big)\left((1-\beta)I + \beta W^{(l)}\right), \tag{3.9}$$

where $X^{(l)}$ is the pre-activation values at the l-th layer; A is the graph convolutional operator [Kipf and Welling, 2017]; α and β are hyperparameters controlling the amount of initial residual and identity mapping; I is the identity matrix; $W^{(l)}$ is the layer-specific trainable weight matrix; and ϕ is the ReLU activation function.

This formula incorporates the initial node features $X^{(0)}$ and the transformed node features $\phi(X^{(l-1)})$ from the previous layer, combined through the weights $W^{(l)}$. This helps maintain distinct node representations across layers, enhancing the model's depth without losing its discriminative power. When these skip/residual connections are between two layers separated by a fully connected layer, we conclude that the two inputs are independent and we write the updates of K and Q as in the last row of the table. For example, here is the composition

for the GCNII layer [Chen et al., 2020], where a skip connection with $X^{(0)}$ occurs:

$$\text{GCNII}: \quad X \leftarrow \left((1-\alpha)A\phi(X) + \alpha X^{(0)}\right)((1-\beta)I + \beta W) \tag{3.10}$$

$$K \leftarrow \left((1-\alpha)^2 Ag(K)A^T + \alpha^2 C^{(0)}\right)((1-\beta)^2 + \beta^2\sigma_w^2) \tag{3.11}$$

$$Q \leftarrow \left((1-\alpha)A\,\text{Chol}(g(QQ^T)) \quad \alpha Q^{(0)}\right)\sqrt{(1-\beta)^2 + \beta^2\sigma_w^2}. \tag{3.12}$$

We leave the proof of the recursive formula (3.10) in Appendix A.

3.3.2 GraphSAGE

GraphSAGE generates embeddings by sampling and aggregating features from a node's local neighborhood [Hamilton et al., 2017]. For simplicity, we focus on the mean-aggregation version of GraphSAGE, whose update rule for a node u in GraphSAGE is:

$$X^{(l)}(u) = W_1^{(l)}\phi(X^{(l-1)}(u)) + W_2^{(l)} \cdot \text{mean}_{v\in\mathcal{N}(u)}\phi(X^{(l-1)}(v)). \tag{3.13}$$

As before, X denotes pre-activation rather post activation and $X^{(l)}(u)$ denotes the feature vector of node u at layer l. A GraphSAGE layer has two weight matrices, $W_1^{(l)}$ and $W_2^{(l)}$, and hence we use two variance parameters σ_{w_1} and σ_{w_2} for them, respectively. $\mathcal{N}(u)$ denotes the neighborhood of u, and the mean is computed on all nodes in the neighborhood.

This method combines the node's own features, weighted by $W_1^{(l)}$, with the mean of its neighbors' features, weighted by $W_2^{(l)}$ and effectively the random walk matrix A whose entries have a row sum of 1. The two resulting embeddings are also independent, and the

updates are:

$$\text{GraphSAGE}: \quad X \leftarrow \phi(X)W_1 + A\phi(X)W_2 \tag{3.14}$$

$$K \leftarrow \sigma_{w_1}^2 g(K) + \sigma_{w_2}^2 A g(K) A^T \tag{3.15}$$

$$Q \leftarrow \begin{pmatrix} \sigma_{w_1} \operatorname{Chol}(g(QQ^T)) & \sigma_{w_2} A \operatorname{Chol}(g(QQ^T)) \end{pmatrix}. \tag{3.16}$$

We leave the proof of the recursive formula (3.21) in Appendix A.

3.3.3 Attention Layers

The GAT architecture utilizes masked self-attention layers to weigh the importance of each neighboring node's feature [Veličković et al., 2018]. The attention mechanism in GAT is defined as:

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{w \in \mathcal{N}(u)} \exp(e_{uw})}, \quad e_{uv} = \operatorname{Att}(x_u W, x_v W), \tag{3.17}$$

and

$$z_u = \sum_{v \in \mathcal{N}(u)} \alpha_{uv} x_v W, \tag{3.18}$$

where z_u and x_u are the pre- and post-activation feature vector of node u, respectively; W is the attention weight shared by queries, keys and values, following the convention of the attention mechanism [Hron et al., 2020]; $\mathcal{N}(u)$ is the set of neighbors of u; α_{uv} is the attention coefficient between nodes u and v, being a row-wise masked softmax of the e_{uv} matrix, which we denote by operator $\zeta : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$:

$$[\zeta(K)]_{uv} := \frac{\exp(K_{uv})}{\sum_{w \in \mathcal{N}(u)} \exp(K_{uw})}. \tag{3.19}$$

The self-attention Att : $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ determines how much attention node u should pay on node v during the aggregation, and enables the model to assign different importance to different neighbors. We choose Att to be the d^{-1} scaled inner products as proposed in [Yang, 2019], and rewrite the attention coefficients in matrices as:

$$e_{uv} = d^{-1} x_u W W^T x_v^T, \quad E = d^{-1} X W W^T X^T. \tag{3.20}$$

Finally, the updates for each GAT layer are:

$$\text{GAT}: \quad X \leftarrow \zeta(d^{-1} X W W^T X^T) X W \tag{3.21}$$

$$K \leftarrow \sigma_w^2 \zeta(\sigma_w^2 K) K \zeta(\sigma_w^2 K)^T \tag{3.22}$$

$$Q \leftarrow \sigma_w \zeta(\sigma_w^2 Q Q^T) Q. \tag{3.23}$$

We leave the proof of the recursive formula (3.21) in Appendix A.

3.4 Properties of Composed Kernels

The programmable nature of GNNGPs allows us to extend the property observed in GCNs to more complex architectures. We demonstrate this with the GCN2GP and GraphSAGE-GP model.

Theorem 5. *(Limit for GCN2GP)*
Assume A is a symmetric primitive matrix, with Perron-Frobenius eigenvalue $r > 0$ and corresponding eigenvector v. Let $\delta = \frac{1}{2}((1-\beta)^2 + \beta^2 \sigma_w^2)(1-\alpha)^2 r^2$. The following results hold as $l \to \infty$.

1. *When $\alpha = 0$, $\rho_{\min}(K^{(l)}) \nearrow 1$.*
2. *When $\delta < 1$, $K^{(l)}$ converges linearly with a rate of convergence of $\delta^{1/2}$.*

3. *When $\delta > 1$, the scaled $\kappa^{(l)} := K^{(l)}/\delta^l \to cvv^T$ for some constant c.*

Theorem 6. *(Limit for GraphSAGE-GP)*

Assume A is a symmetric primitive matrix, with Perron-Frobenius eigenvalue $r > 0$ and corresponding eigenvector v. Let $\delta = \frac{1}{2}(\sigma_{w_1}^2 + \sigma_{w_2}^2 r^2)$. The following results hold as $l \to \infty$.

1. *The $\rho_{\min}(K^{(l)}) \nearrow 1$.*

2. *When $\delta < 1$, $K^{(l)}$ converges linearly with a rate of convergence of $\delta^{1/2}$.*

3. *When $\delta > 1$, the scaled $\kappa^{(l)} := K^{(l)}/\delta^l \to cvv^T$ for some constant c.*

A few remarks follow. The first case suggests that, without initial mappings, skip connections can only partially address the degeneration in kernels observed in deep GCNGPs. While they may help mitigate the issue, the node features will still converge towards collinearity and eventually become indistinguishable. The second and third case illustrate extending the contraction property the from GCNGP to GCN2GP and GraphSAGE-GP. The results are consistent: $K^{(l)}$ or its scaled version converge to a unique fixed point, aligning with the observation that deep GNNs converge to linear models [Hron et al., 2020].

The result demonstrates the inherent problem with stability and predictability of these deep architectures. Understanding these behaviors, and adjust to appropriate mechanisms such as initial residual and identity mappings, is crucial for designing GNN or GNNGP architectures that can effectively leverage depth while maintaining robustness and prediction power. Balancing the benefits of depth with the need to prevent over-smoothing remains a key challenge in the development of GNN models.

3.5 Experiments

We conduct experiments with several popularly used GNN architectures (GCNII, GIN, and GraphSAGE) and GPs with the corresponding kernels. We test with the three largest

datasets: PubMed, ArXiv, and Reddit, for the latter two of which a low-rank version of the GPs is used for computational feasibility.

Table 3.2 summarizes the results. The observations on other GNNs extend similarly those on the GCN. In particular, on PubMed the GPs noticeably improve over the corresponding GNNs, while on ArXiv and Reddit the two families perform rather similarly. Noticeably, the best architecture is also often shared between GNN and the GP counterpart. An exception is ArXiv, on which GCN and GCNII results are close. It may improve with an extensive hyperparameter tuning.

Table 3.2: Performance of GNNGP in comparison with corresponding GNNs.

The architecture with highest Micro-F1 score is highlighted in boldface.

Dataset		GCN	GCNII	GIN	GraphSAGE
PubMed	GNN	**0.7649±0.0058**	0.7558±0.0096	0.7406±0.0112	0.7535±0.0047
	GNNGP	**0.7960**	0.7840	0.7690	0.7900
ArXiv	GNN	0.6989±0.0016	**0.7008±0.0021**	0.6340±0.0056	0.6984±0.0021
	GNNGP-X	**0.7011±0.0011**	0.6955±0.0011	0.6652±0.0012	0.6962±0.0007
Reddit	GNN	0.9330±0.0006	0.9482±0.0007	0.9398±0.0016	**0.9628±0.0007**
	GNNGP-X	0.9465±0.0003	0.9500±0.0003	0.9428±0.0005	**0.9539±0.0003**

Chapter 3, in part, is a reprint of the material as it appears in [Niu et al., 2023].

CHAPTER 4

GRAPH NEURAL TANGENT KERNELS: EVOLUTION OF GNNGP

4.1 Introduction

While the Neural Network Gaussian Process (NNGP) framework provides a theoretical understanding of randomly initialized NNs in the infinite width limit, the Neural Tangent Kernel (NTK) represents a generalization of the Neural Network Gaussian Process (NNGP) by incorporating the dynamics of learning over time [Jacot et al., 2018, Lee et al., 2019]. The NNGP provides a static view of the network's function distribution before training. In contrast, NTK extends this perspective by describing how the function evolves as the network is trained using gradient descent.

The NTK was introduced in [Jacot et al., 2018] to study the convergence and generalization properties of fully connected neural networks. Their work laid the foundation for understanding the duality between training wide neural networks and kernel methods. This duality has profound implications, as it simplifies the training dynamics and predictions of wide neural networks into closed-form equations. Subsequent research has extended the NTK framework to various neural network architectures beyond fully connected layers, demonstrating its versatility and robustness.

For instance, [Du et al., 2019] introduced the concept of GNTKs, which correspond to infinitely wide multi-layer GNNs trained by gradient descent. [Krishnagopal and Ruiz, 2023] investigated the training dynamics of large-graph GNNs using GNTKs and graphons, revealing how the GNTK evolves as the graph size increases. Nonetheless, these prior work predominantly focus on graph classification tasks, leaving a gap in research on node classification and regression tasks. Here, we propose to adapt the idea and framework of GNTK, as introduced by [Du et al., 2019], to node-level learning tasks, and gaining a deeper under-

standing of the learning dynamics in graph-based models. This adaptation aims to explore the potential of GNTKs in predicting individual node attributes and behaviors within a graph, expanding their applicability beyond graph-level predictions.

The NTK effectively describes the network's learning trajectory, providing a time-dependent kernel that reflects the influence of gradient descent optimization on the function space [Avidan et al., 2023]. At the initialization of a neural network, the function distribution is described by the NNGP. As training commences, the NTK captures the changes in the network's output concerning infinitesimal changes in its parameters. This dynamic kernel thus serves as a powerful tool for understanding both the static and dynamic aspects of neural network behavior, offering valuable insights into the optimization and generalization properties of deep learning models.

This transition from GNNGP to GNTK over the course of training illustrates the generalization capability of the GNNGP framework. We have represented the GNNGP family and are now poised to introduce the Graph Neural Tangent Kernel (GNTK). The GNTK not only captures the initial function distribution, akin to GNNGP, but also the network's evolution during training, providing a comprehensive view of the learning process in neural networks with infinite width. This comprehensive understanding is pivotal for advancing the theoretical and practical applications of GNNs in complex graph-based learning tasks.

4.2 Evolution of Graph Convolutional Networks

In this section, we once again represent our idea using GCN, extending the connection between the training process of infinitely wide neural networks and Gaussian Processes, and investigate the Graph Convolutional Network Neural Tangent Kernel (GCNNTK) as evolution of the GCNGP.

We briefly review the GNTK parameterization of GCN as follows:

$$y_i^{(l)}(x) = \frac{\sigma_w}{\sqrt{d_{l-1}}} \sum_{j=1}^{d_{l-1}} W_{ji}^{(l)} x_j^{(l-1)}(x), \tag{4.1}$$

$$z_i^{(l)}(x) = \sigma_b b_i^{(l)} + \sum_{v \in \mathcal{V}} A_{xv} y_i^{(l)}(v), \tag{4.2}$$

$$x_i^{(l)}(x) = \phi(z_i^{(l)}(x)). \tag{4.3}$$

Note we apply an explicit rescale weight $d_{l-1}^{-1/2}$ in the formula to avoid divergence with infinite-width networks. The constant scalar σ_b and σ_w controls the bias term and weight terms. Collect all $P = \sum_{l=1}^{L}(d_{l-1}+1)d_l$ neural network parameters $W_{ij}^{(l)}, b_i^{(l)}$ into a vector $\theta \in \mathbb{R}^P$.

During the training process, we aim to minimize the loss function $\mathcal{L}$ using gradient descent on the parameter θ, as described by

$$\frac{d\theta(t)}{dt} = -\eta \nabla_\theta \mathcal{L} = -\eta (\nabla_\theta f)^T (\nabla_f \mathcal{L}), \tag{4.4}$$

where $\eta > 0$ is the learning rate, $f = f(X^{(0)}; \theta(t)) \in \mathbb{R}^{N \times d_L}$ is the neural network output at time t, and the Jacobian $\nabla_\theta f \in \mathbb{R}^{N \times P}, \nabla_f \mathcal{L} \in \mathbb{R}^{d_L}$. Then according to the chain rule,

$$\frac{df(t)}{dt} = (\nabla_\theta f)\frac{d\theta(t)}{dt} = -\eta[(\nabla_\theta f)(\nabla_\theta f)^T](\nabla_f \mathcal{L}). \tag{4.5}$$

The Neural Tangent Kernel (NTK), is defined in the above formula as

$$\Theta = \Theta(\theta(t)) = (\nabla_\theta f)(\nabla_\theta f)^T \in \mathbb{R}^{N \times N}. \tag{4.6}$$

The following theorem states that during the training process, in the limit on the layer width, the NTK for the GCN output $z^{(L)}(x)$ remains constant.

Theorem 7. *Assume $d_1, \cdots, d_{L-1}$ to be infinite in succession and let the bias and weight terms be independent with distributions*

$$b_i^{(l)} \sim \mathcal{N}(0,1), \quad W_{ij}^{(l)} \sim \mathcal{N}(0,1), \quad l = 1, \cdots, L. \tag{4.7}$$

Then, for each i, the collection $\{z_i^{(l)}(x)\} \in \mathbb{R}^N$ over all graph nodes x have a constant NTK $\Theta^{(l)}$, which can be computed recursively by

$$C^{(l)} = \mathbb{E}_{z_i^{(l)} \sim \mathcal{N}(0, K^{(l)})}[\phi(z_i^{(l)})\phi(z_i^{(l)})^T], \qquad l = 1, \cdots, L, \tag{4.8}$$

$$\dot{C}^{(l)} = \mathbb{E}_{z_i^{(l)} \sim \mathcal{N}(0, K^{(l)})}[\dot{\phi}(z_i^{(l)})\dot{\phi}(z_i^{(l)})^T], \qquad l = 1, \cdots, L, \tag{4.9}$$

$$K^{(l+1)} = \sigma_b^2 1_{N\times N} + \sigma_w^2 A C^{(l)} A^T, \qquad l = 0, \cdots, L-1, \tag{4.10}$$

$$\Theta^{(l+1)} = \sigma_b^2 1_{N\times N} + \sigma_w^2 A(\Theta^{(l)} \odot \dot{C}^{(l)} + C^{(l)})A^T, \qquad l = 0, \cdots, L-1. \tag{4.11}$$

with $C^{(0)}$ is defined the same as Theorem 1 as follows:

$$C^{(0)} = \frac{X^{(0)}(X^{(0)})^T}{d_0}. \tag{4.12}$$

To illustrate the evolution process during training, we present a visual representation in Figure 4.1 with a minimal artificial example: Consider a dataset containing one training data point and one test data point connected with each other. We represent the neural network's output with a single blue point, where the x-axis corresponds to the training data and the y-axis corresponds to the test data. To capture the variability of neural network outputs, we perform 100 random initializations of the neural network and record the output at four distinct times: $t = 0, 10, 20, 100$. These times are displayed in four separate subplots. For each time t, we compare the neural network's output with the theoretical Gaussian Process. The theoretical Gaussian Process is depicted with its mean $\mu(t)$ represented by a red point and the 2σ confidence ellipse for that GP with black dashed line, which encapsulates the

variability of the Gaussian Process at each time point.

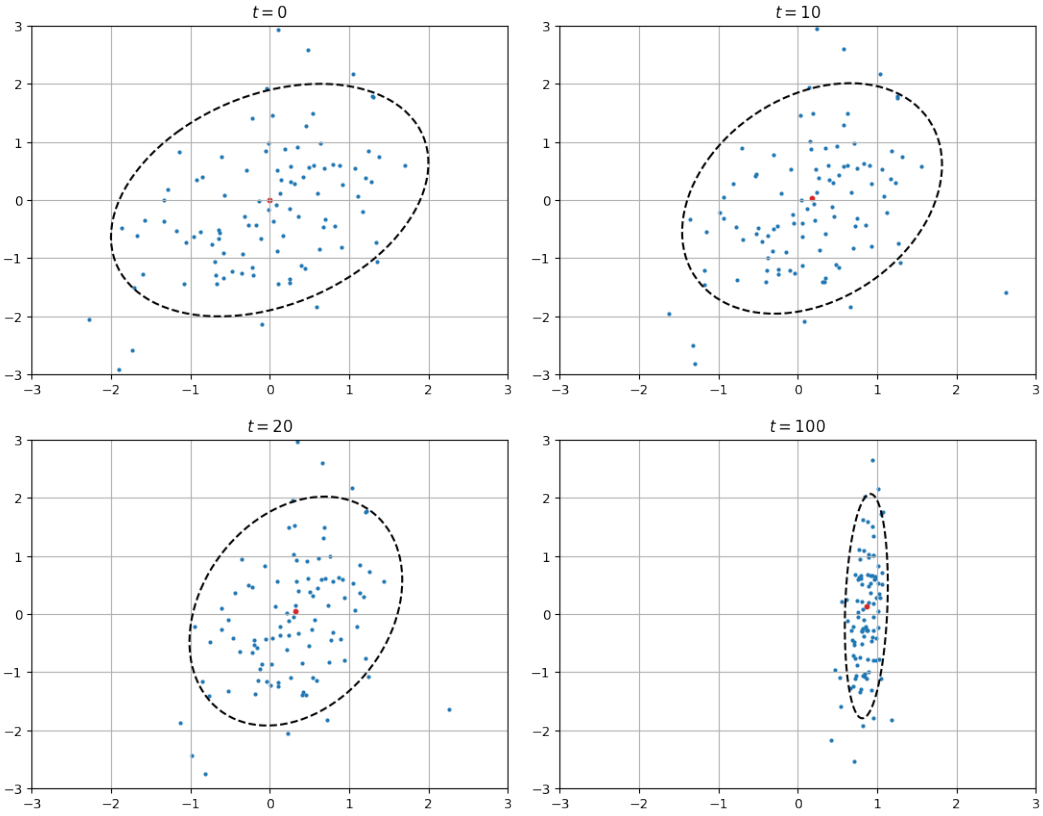


Figure 4.1: Evolution of GNNGP on minimal artificial dataset.

The evolution of the Gaussian Process during training reveals how the network's predictions become more refined and accurate as the training progresses. At the initial time point $t = 0$, the Gaussian Process reflects the prior distribution, which is broad and uncertain. As training advances to $t = 10$ and $t = 20$, the Gaussian Process begins to converge, with the confidence ellipse becoming narrower and the mean shifting closer to the true value. By $t = 100$, the Gaussian Process has further evolved, providing a more precise and confident prediction.

This progression highlights the dynamic nature of the Gaussian Process under the influence of gradient descent optimization. The GCNNTK captures this evolving behavior and extends the GCNGP by incorporating the temporal dynamics of training, providing a

richer and more detailed understanding of the learning process. The GCNNTK framework offers a comprehensive view of how the network's function and uncertainty change over time, which is crucial for designing and optimizing graph neural networks and leverage the power of Gaussian Processes.

4.3 Computation of GNTKs

4.3.1 Computation in Kernel Matrices

The recursive relationship in (4.11) gives the following naive computation algorithm:

Algorithm 3 Naive computation

Require: $C^{(0)} \geq 0$

$K^{(0)} \leftarrow \sigma_b^2 1_{N\times N} + \sigma_w^2 A C^{(0)} A^T$

$\Theta^{(0)} \leftarrow K^{(0)}$

for $l = 0, \cdots, L-1$ **do**

  for $x, x' \in V$ **do**

    $C^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,K^{(l)})}[\phi(z(x))\phi(z(x'))]$

    $\dot{C}^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,K^{(l)})}[\dot{\phi}(z(x))\dot{\phi}(z(x'))]$

  end for

  $K^{(l+1)} \leftarrow \sigma_b^2 1_{N\times N} + \sigma_w^2 A C^{(l)} A^T$

  $\Theta^{(l+1)} \leftarrow \sigma_b^2 1_{N\times N} + \sigma_w^2 A(\Theta^{(l)} \odot \dot{C}^{(l)} + C^{(l)})A^T$

end for

with time complexity $O(LMN)$ and space complexity $O(N^2)$.

We use the Nyström approximation introduced in GNN-GP.

Algorithm 4 Nyström computation

Require: $C^{(0)} \geq 0$

$Q^{(0)} \leftarrow \begin{pmatrix} \sigma_w A \operatorname{Chol}(C^{(0)}) & \sigma_b 1_{N\times 1} \end{pmatrix}$

$R^{(0)} \leftarrow Q^{(0)}$

for $l = 0, \cdots, L-1$ **do**

  for $x \in V, x' \in V_a$ **do**

    $C^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,Q^{(l)}Q^{(l)T})}[\phi(z(x))\phi(z(x'))]$

    $\dot{C}^{(l)}(x, x') \leftarrow E_{z\sim\mathcal{N}(0,Q^{(l)}Q^{(l)T})}[\dot{\phi}(z(x))\dot{\phi}(z(x'))]$

  end for

  $Q^{(l+1)} \leftarrow \begin{pmatrix} \sigma_w A \operatorname{Chol}(C^{(l)}) & \sigma_b 1_{N\times 1} \end{pmatrix}$

  $R^{(l+1)} \leftarrow \begin{pmatrix} \sigma_w A \operatorname{Chol}((R^{(l)} R_{a\cdot}^{(l)T}) \odot \dot{C}^{(l)} + C^{(l)}) & \sigma_b 1_{N\times 1} \end{pmatrix}$

end for

Overall, the time complexity is $O(LMN_a + LNN_a^2)$ and space complexity is $O(NN_a)$.

4.3.2 Scalable Computation in Posterior Inference

Assume there are N data points, denoted by input $X \in \mathbb{R}^{N\times d_0}$ and target $y \in \mathbb{R}^N$, and we use subscripts b and c the same meaning as Section 2.3.2. Consider the choice of $\mathcal{L}$ as the empirical l_2 loss, used in traditional literature [Jacot et al., 2018]:

$$\mathcal{L} = \mathcal{L}(t) = \|f(X_b; \theta(t)) - y_b\|_2^2. \tag{4.13}$$

Note that the empirical loss is computed on training points only. According to the gradient update rule (4.5), we know that the output of an infinite width network is as follows:

$$\frac{d}{dt}\begin{pmatrix} f_b(t) \\ f_c(t) \end{pmatrix} = -\eta\Theta^{(L)}(\nabla_f \mathcal{L}) = -\eta \begin{pmatrix} \Theta_{bb}^{(L)} & \Theta_{bc}^{(L)} \\ \Theta_{cb}^{(L)} & \Theta_{cc}^{(L)} \end{pmatrix} \begin{pmatrix} f_b(t) - y_b \\ 0 \end{pmatrix}. \tag{4.14}$$

Solving the above simple linear ODE gives the following closed-form solution:

$$f_b(t) = y_b - \exp\left(-\eta t \Theta_{bb}^{(L)}\right)(y_b - f_b(0)), \tag{4.15}$$

$$f_c(t) = f_c(0) + \Theta_{cb}^{(L)}\left(\Theta_{bb}^{(L)}\right)^{-1}\left(I - \exp\left(-\eta t \Theta_{bb}^{(L)}\right)\right)(y_b - f_b(0)), \tag{4.16}$$

where $f_b(0)$ and $f_c(0)$ are zero-mean Gaussian processes according to Theorem 1. Thus the estimated posterior mean with nugget ϵ is

$$\hat{y}_c(t) = \hat{\Theta}_{cb}^{(L)}\left(\hat{\Theta}_{bb}^{(L)} + \epsilon I\right)^{-1}\left(I - \exp\left(-\eta t \hat{\Theta}_{bb}^{(L)}\right)\right) y_b. \tag{4.17}$$

With Nyström approximation in Algorithm 4, we have

$$\hat{\Theta}_{cb}^{(L)} = R_{ca}^{(L)} {R_{ba}^{(L)}}^T, \quad \hat{\Theta}_{bb}^{(L)} = R_{ba}^{(L)} {R_{ba}^{(L)}}^T. \tag{4.18}$$

And using the push-through identity,

$$\hat{y}_c(t) = R_{ca}^{(L)} {R_{ba}^{(L)}}^T \left(R_{ba}^{(L)} {R_{ba}^{(L)}}^T + \epsilon I\right)^{-1}\left(I - \exp\left(-\eta t R_{ba}^{(L)} {R_{ba}^{(L)}}^T\right)\right) y_b \tag{4.19}$$

$$= R_{ca}^{(L)}\left({R_{ba}^{(L)}}^T R_{ba}^{(L)} + \epsilon I\right)^{-1} {R_{ba}^{(L)}}^T \left(I - \exp\left(-\eta t R_{ba}^{(L)} {R_{ba}^{(L)}}^T\right)\right) y_b \tag{4.20}$$

$$= R_{ca}^{(L)}\left({R_{ba}^{(L)}}^T R_{ba}^{(L)} + \epsilon I\right)^{-1}\left(I - \exp\left(-\eta t {R_{ba}^{(L)}}^T R_{ba}^{(L)}\right)\right) {R_{ba}^{(L)}}^T y_b. \tag{4.21}$$

where the matrix inversion and exponentiation are done on a size $(N_a+1)\times(N_a+1)$ matrix, significantly smaller than the whole matrix of size $N_b \times N_b$.

Similarly, the posterior variance is

$$\hat{V}_{cc}(t) = \hat{\Theta}_{cc}^{(L)} - \hat{\Theta}_{cb}^{(L)}\left(\hat{\Theta}_{bb}^{(L)} + \epsilon I\right)^{-1}\hat{\Theta}_{bc}^{(L)} \tag{4.22}$$

$$= \epsilon R_{ca}^{(L)}\left({R_{ba}^{(L)}}^T R_{ba}^{(L)} + \epsilon I\right)^{-1}\left(I - \exp\left(-\eta t {R_{ba}^{(L)}}^T R_{ba}^{(L)}\right)\right)^2 {R_{ca}^{(L)}}^T. \tag{4.23}$$

The computational costs of GCNNTK and the posterior inference from (4.19) to (4.22) are summarized in Table 4.1. The complexity differ only by a constant factor from the computational cost of GCNGP in Table 2.1. Note that the GNTK cannot be computed without the GNNGP kernel matrix, which adds to the computation complexity.

Table 4.1: Computational costs for GCNNTK.

M: number of edges; N: number of nodes; $N_a/N_b/N_c$: number of landmark / training / prediction nodes; L: number of layers. Assume $N_a \ll N_b, N_c$. For posterior variance, assume only the diagonal is needed.

	Time $O(\cdot)$	Storage $O(\cdot)$
Kernel computation	$LMN_a + LNN_a^2$	NN_a
Posterior mean (4.19)	$N_cN_a + N_bN_a^2$	$(N_b + N_c)N_a$
Posterior variance (4.22)	$N_cN_a + N_bN_a^2$	$(N_b + N_c)N_a$

Similar to the programmable nature of GNNGP kernels, there exist correspondences between neural network building blocks, NTK operations, and their low-rank counterparts. This allows us to extend the GNTK framework to a wider range of GNN architectures, enhancing its flexibility and applicability across various graph-based models.

Table 4.2: Neural network building blocks, NTK operations, and the low-rank counterpart.

Building block	Neural network	NTK operation	Nyström operation
Input	$X \leftarrow X^{(0)}$	$\Theta \leftarrow 0$	$R \leftarrow 0$
Bias term	$X \leftarrow X + \sigma_b b$	$\Theta \leftarrow \Theta + \sigma_b^2 1_{N\times N}$	$R \leftarrow \begin{pmatrix} R & \sigma_b 1_{N\times 1} \end{pmatrix}$
Weight term	$X \leftarrow \frac{\sigma_w}{\sqrt{d}} XW$	$\Theta \leftarrow \sigma_w^2 \Theta + \sigma_w^2 K$	$R \leftarrow \begin{pmatrix} \sigma_w R & \sigma_w Q \end{pmatrix}$
Graph convolution	$X \leftarrow AX$	$\Theta \leftarrow A\Theta A^T$	$R \leftarrow AR$
Activation	$X \leftarrow \phi(X)$	$\Theta \leftarrow \Theta \odot \dot{K}$	$R \leftarrow \mathrm{Chol}((RR^T) \odot (Q\dot{Q}^T))$
Independent sum	$X \leftarrow X_1 + X_2$	$\Theta \leftarrow \Theta_1 + \Theta_2$	$R \leftarrow \begin{pmatrix} R_1 & R_2 \end{pmatrix}$
Skip connection	$X \leftarrow \beta_1 X + \beta_2 XW$	$\Theta \leftarrow (\beta_1^2 + \beta_2^2\sigma_w^2)\Theta + \beta_2^2\sigma_w^2 K$	$R \leftarrow \begin{pmatrix} \sqrt{\beta_1^2 + \beta_2^2\sigma_w^2} R & \beta_2\sigma_w Q \end{pmatrix}$
Mean aggregation	$X \leftarrow XW_1 + AXW_2$	$\Theta \leftarrow \sigma_{w_1}^2(\Theta + K) + \sigma_{w_2}^2 A(\Theta + K)A^T$	$R \leftarrow \begin{pmatrix} \sigma_{w_1} R & \sigma_{w_2} AR & \sigma_{w_1} Q & \sigma_{w_2} AQ \end{pmatrix}$
Self attention	$X \leftarrow \zeta(d^{-1}XX^T)X$	$\Theta \leftarrow \zeta(K)\Theta\zeta(K)^T + 2K$	$R \leftarrow \begin{pmatrix} \zeta(QQ^T)R & \sqrt{2}Q \end{pmatrix}$

4.4 Experiments

We continue to evaluate the performance of GCNNTK using semi-supervised learning tasks on six smaller datasets introduced in Table 2.2, along with GCN and GCNGP. GCNNTK represents the temporal limit of GCNGP, and we include a low-rank version (suffixed with -X) for the two methods.

Table 4.3 summarizes the performance on classification and regression results. GCN performance and the two low-rank methods on large datasets are averaged over ten runs, whereas other results involve no randomness. These results share a similar pattern to Table 2.3: For classification tasks, GCNNTK(-X) generally surpasses GCN while slightly better than GCNGP, with the full kernel matrix outperforming the low-rank version. In contrast, for regression tasks, GCNNTK(-X) shows superior results compared to GCNGP(-X) and GCN, with the low-rank version leading. These findings align with theoretical expectations, as GCNNTK is the temporal limit of GCNGP.

Table 4.3: Performance of GCNNTK in comparison with GCN and GCNGP.

The Micro-F1 score is reported for classification tasks and R^2 is reported for regression tasks.

Dataset	GCN	GCNGP	GCNNTK	GCNGP-X	GCNNTK-X
Cora	$0.8183_{\pm 0.0055}$	**0.8280**	0.8250	0.7980	0.7950
CiteSeer	$0.6941_{\pm 0.0079}$	0.7090	**0.7180**	0.7080	0.7030
PubMed	$0.7649_{\pm 0.0058}$	**0.7960**	0.7950	0.7810	0.7810
ArXiv	$0.6990_{\pm 0.0014}$	OOM	OOM	$\mathbf{0.7034_{\pm 0.0028}}$	$0.7011_{\pm 0.0011}$
Reddit	$0.9330_{\pm 0.0006}$	OOM	OOM	$\mathbf{0.9471_{\pm 0.0006}}$	$0.9465_{\pm 0.0003}$
Chameleon	$0.5690_{\pm 0.0376}$	0.6720	0.6802	0.6852	**0.6928**
Squirrel	$0.4243_{\pm 0.0393}$	0.4926	0.5007	0.4998	**0.5084**
Crocodile	$0.6976_{\pm 0.0323}$	0.8002	0.8036	0.8012	**0.8058**

For graph learning tasks, the trade-offs between accuracy and computational efficiency

are crucial. The timing results of these tasks are shown in Table 4.4, whereas each time is normalized against the GCN time. Table 4.4 shows the timing results, with each time normalized against the GCN time. GCNNTK(-X) is generally faster than GCN, with the exception of PubMed dataset. The GCNNTK(-X) shows competitive performance on certain tasks, yet have higher computation times compared to GCNGP(-X) methods due to more complex kernel matrix computations. These insights are important for selecting the appropriate method for specific graph-based learning tasks.

Table 4.4: Timing comparison of GCNNTK.

For each dataset, the computation times are normalized against that of GCN.

Dataset	GCN	GCNGP	GCNNTK	GCNGP-X	GCNNTK-X
Cora	1.0000	0.1360	0.1470	0.1217	0.1326
CiteSeer	1.0000	0.1491	0.1571	0.1210	0.1323
PubMed	1.0000	0.8532	1.9420	0.0809	0.0876
ArXiv	1.0000	OOM	OOM	0.1062	0.5412
Reddit	1.0000	OOM	OOM	0.0054	0.0102
Chameleon	1.0000	0.2204	0.2540	0.3884	0.4010
Squirrel	1.0000	0.2693	0.4480	0.3136	0.3439
Crocodile	1.0000	0.4132	0.5796	0.6153	0.6578

CHAPTER 5

PERFORMANCE ANALYSIS AND SCALING LAWS

5.1 Performance Analysis

The relationship between GNNs and GNNGPs allows us to examine the performance of GNNs by considering their GNNGPs. By analyzing GNNGPs, we can gain insights into the theoretical limits and properties of GNNs, providing a powerful analytical framework to study their performance in the asymptotic regime.

5.1.1 Depth

One of the critical challenges in the performance of Graph Neural Networks (GNNs) is the phenomenon of oversmoothing. As the depth of the network increases, node features tend to converge towards similarity, making them indistinguishable. This effect is particularly pronounced in deeper networks, where the repeated application of the graph convolution operation causes the node representations to become overly smooth, leading to a loss of valuable information and a subsequent deterioration in performance. The oversmoothing issue is a significant bottleneck in the scalability of GCNs, as it limits the depth to which these networks can be trained effectively without compromising their discriminative power.

Recent studies have proposed various methods to mitigate oversmoothing, such as introducing skip connections and modifying the aggregation functions, such as in GCNII. A natural question asks if the corresponding GP kernels behave similarly.

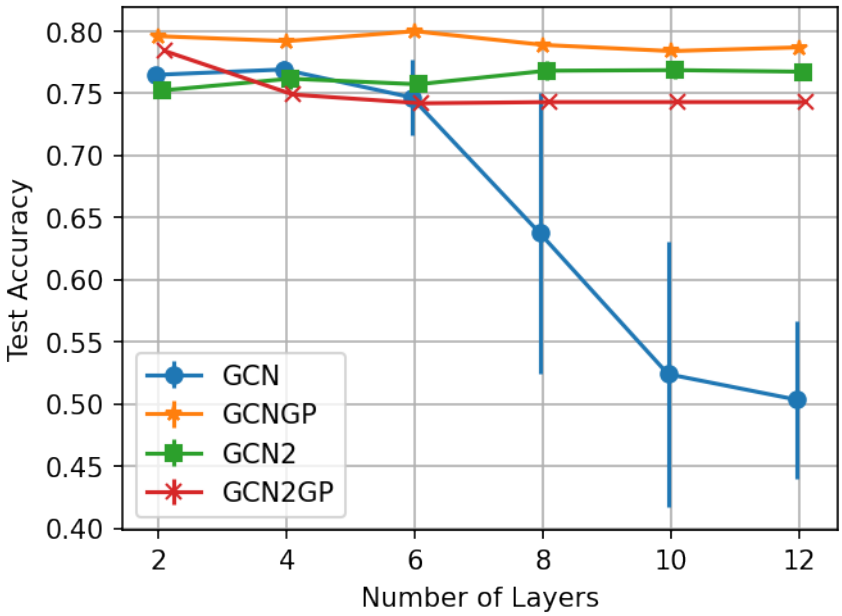


Figure 5.1: Performance of GNNs and GNNGPs as depth L increases. Dataset: Pubmed.

Figure 5.1 shows that the trends of GCN and GCNII are indeed as expected. Interestingly, their GP counterparts both remain stable for depth L as large as 12. Our depth analysis (Theorem 4) suggests that in the limit, the GPs may perform less well because the kernel matrix may degenerate to rank 1. This empirical result indicates that the drop in performance may have not started yet.

5.1.2 Compute

The trade-off between accuracy and computational time is a fundamental consideration in the design of GNNGP models. Higher accuracy often requires more complex models and longer training times, which can be prohibitive in terms of computational resources. This trade-off is particularly relevant in large-scale applications where the efficiency of the model is as critical as its performance. Researchers are actively exploring methods to achieve a balance where the model maintains high accuracy while being computationally feasible.

It is crucial to develop an empirical understanding of the accuracy-time trade-off it in-

curs.In the GNNGP model, the number of landmark nodes, N_a, controls the approximation quality of the low-rank kernels and hence the prediciton accuracy. On the other hand, the computational costs summarized in Table 2.1 indicate a dependency on N_a as high as the third power, highlighting the need for a comprehensive analysis of the trade-off between accuracy and time in GNNGP models.

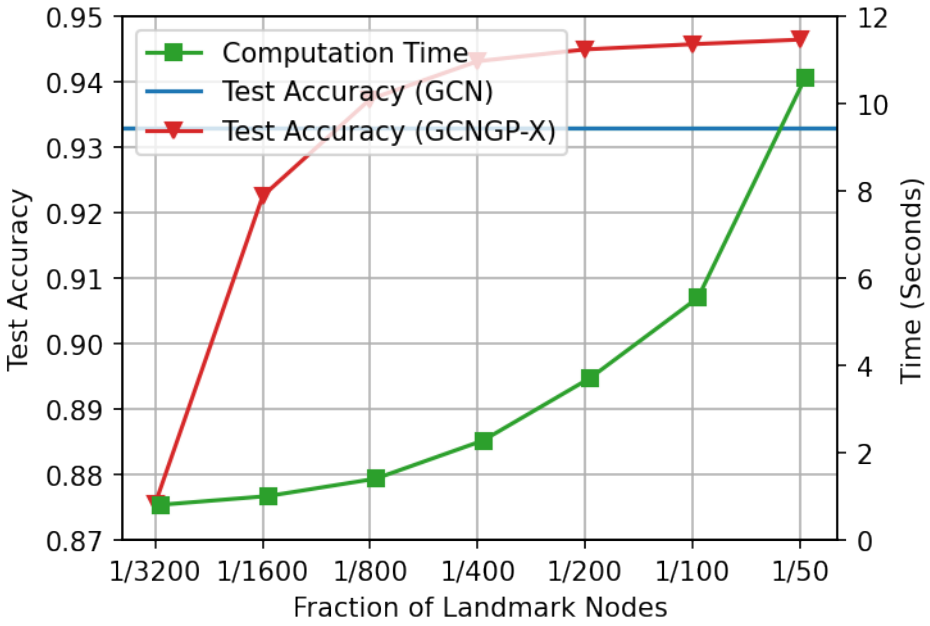


Figure 5.2: Performance of GCNGP-X as number of landmark nodes increases. Dataset: Reddit.

Figure 5.2 clearly shows that as N_a becomes larger, the running time increase is not linear, while the increase of accuracy diminishes as the landmark set approaches the training set. It is remarkable that using only 1/800 of the training set as landmark nodes already achieves an accuracy surpassing that of GCN, by using time that is only a tiny fraction of the time otherwise needed to gain an additional 1% increase in the accuracy.

5.2 Scaling Laws

Another approach to understanding and enhancing the scalability of GNNs is through the study of scaling laws. For various models and datasets, neural network performance has been empirically observed to follow a power-law relationship with respect to factors such as model size, dataset size, and training cost. These studies also offer simple yet valuable predictions for training strategies, optimal computation allocations, and designing large models.

Scaling laws focus on the question: How to trade-off model size and training set size to achieve optimal performance, under the constraint of total compute during training? We aim to answer the question for GNNs. To achieve this, we start by training a range of models varying both model size and dataset size. By systematically varying these factors, we identify empirical estimators of scaling laws that govern GNN performance. We then use the resulting training efficient frontier to fit a compute-optimal model and compare it with its non-optimal counterpart.

Specifically, we employ a self-supervised learning framework, which has proven effective for learning informative and transferable representations of graph data without relying on manual labels. By incorporating self-supervised learning into our scaling law analysis, we aim to reveal key insights into the trade-offs between model complexity and dataset size, as well as the impact of training cost on the graph learning performance of GNNs. Our goal is to discover new strategies for designing and training GNNs that can effectively handle large-scale graph data, paving the way for future advancements in graph-based learning.

5.2.1 Introduction

We review the related work on graph self-supervised learning (SSL) and scaling laws.

While GNNs have garnered significant attention in the field of graph representation learning, most of the works have focused on supervised or semi-supervised learning settings, where the model is directly trained by specific downstream tasks with labeled data. This may limit

its application to real-world scenarios in which labels are often limited, expensive, and even inaccessible due to requirement of extensive domain knowledge. Additionally, these methods are prone to suffer from problems of over-fitting and poor generalization when training data is limited. Furthermore, these models are vulnerable to adversarial attacks targeting the labels, which compromises their robustness [You et al., 2021, Liu et al., 2022].

SSL is a powerful learning paradigm that alleviates the reliance on labeled data and human supervision. As "the key to human-level intelligence" by Turing Award winners Yoshua Bengio and Yann LeCun, SSL has been successfully applied to natural language processing and computer vision, where it has shown to learn inherent structure of data without human intervention. Compared with image and language sequence data, applying SSL to graph data is challenging as graph data can capture rich structural and semantic information that is not easily accessible by other data modalities.

A variety of Graph SSL frameworks have been introduced in the last few years. DGI [Veličković et al., 2018] is a pioneering Graph SSL work that maximizes mutual information between local patch representations and high-level summaries of graphs, capturing meaningful embeddings within the graph by contrasting these representations. Building upon the principles of DGI, InfoGraph [Sun et al., 2020] extends the principles to contrast between the graph-level representation and the representations of substructures of different scales, including nodes, edges, and entire graphs; GMI [Peng et al., 2020] jointly considers edge-level reconstruction and node-level contrast; GraphCL [You et al., 2021] proposes contrasting various data augmentations to the input graphs, such as node dropping, edge perturbation, attribute masking, and subgraph sampling. These methods have demonstrated superior performance on various graph-related tasks.

Scaling laws describe empirical relationships between model size, dataset size, training cost, and test performance of a family of neural networks. Scaling laws have been studied for various domains, particularly in natural language processing tasks, such as machine

translation [Kolachina et al., 2012], speech [Hestness et al., 2017] and language modeling [Kaplan et al., 2020, Hoffmann et al., 2022]. These studies reveal interesting insights into the trade-offs and limits of neural architectures and learning algorithms, and propose formulas to estimate the optimal model size or dataset size for a given task and budget. For example, [Hoffmann et al., 2022] show that for compute-optimal training, the model size and dataset size should be scaled in equal proportions to avoid under-training for large language models.

Scaling laws offer guidance for designing and training scalable neural networks across various applications. However, the scaling laws of GNNs have not been well-studied, due to the unique challenges and opportunities by the nature of graph data. Therefore, we aim to extend the idea of scaling laws to GNNs, investigating how these factors influence GNN performance, and how to choose the optimal model size to train for a given compute budget.

5.2.2 Optimal computation allocation

We use DGI [Veličković et al., 2018] as our self-supervised learning framework. DGI applies a row-wise shuffle $\mathcal{T}$ on the node features of graph $\mathcal{G}$ to obtain an augmented graph $\tilde{\mathcal{G}} = \mathcal{T}(\mathcal{G})$, and passes the two graphs through a GCN f to obtain node embedding

$$H = f(\mathcal{G}), \quad \tilde{H} = f(\tilde{\mathcal{G}}), \tag{5.1}$$

and a graph-level summary $s \in \mathbb{R}^d$ as the mean node feature of $\tilde{H}$. The mutual information between local features and the graph summary is estimated by a bilinear discriminator $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. The learning objective is defined as

$$L = \sum_{h_i \in H} \log \mathcal{D}(h_i, s) + \sum_{\tilde{h}_i \in \tilde{H}} \log \left(1 - \mathcal{D}(\tilde{h}_i, s)\right), \tag{5.2}$$

where h_i and $\tilde{h}_i$ are the node embedding of node i in H and $\tilde{H}$, respectively. The DGI model is trained by maximizing the score for the positive pair (node features and graph summary from the same graph) and minimizing the score for the negative pair (node features and graph summary from different graphs).

We assume a power-law relationship between the DGI loss L w.r.t model size N (number of parameters of DGI model) and dataset size D (number of augmented graph nodes), as done in [Kaplan et al., 2020]. Revisit the question behind scaling laws: How to trade-off model size and training set size to achieve optimal performance, under the constraint of total compute during training? The amount of compute C is a deterministic function $C = \text{FLOPs}(N, D)$ of model size N and dataset size D. With the notation, the question is equivalent to minimizing L under the constraint of C:

$$\min_{N,D} L = L(N, D), \quad s.t.\ C = \text{FLOPs}(N, D). \tag{5.3}$$

For the number of computes C, we have the estimate $C = \text{FLOPs}(N, D) \approx 6ND$, with detailed counting given in Appendix B. We model the DGI loss L with a parametric function as [Hoffmann et al., 2022], following a classical risk decomposition:

$$\hat{L}(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \tag{5.4}$$

whose first term represents the irreducible loss for an ideal model with infinite training time, second term represents the difference between a neural network with N parameters and the ideal model, and third term represents the difference between the model trained using finite optimization steps and negative samples D and the same model trained to convergence.

Finally, minimizing the parametric loss function

$$\min \hat{L} = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \quad s.t.\ C = 6ND, \tag{5.5}$$

gives the optimal allocation

$$N \propto C_{\min}^{\beta/(\alpha+\beta)}, \quad D \propto C_{\min}^{\alpha/(\alpha+\beta)}, \tag{5.6}$$

and the optimal solution $C_{\min}$

$$\hat{L} = E + \frac{F}{C_{\min}^{\gamma}}, \quad \gamma = \frac{\alpha\beta}{\alpha+\beta}, \tag{5.7}$$

with coefficients given in [Hoffmann et al., 2022]. Therefore, with the power-law relationship, the fitted function $\hat{L}$ also has a power-law form w.r.t $C_{\min}$. When compute budget increases, the optimal model size N and dataset size D should increase in $\frac{\beta}{\alpha+\beta} : \frac{\alpha}{\alpha+\beta}$ proportions on log scale.

5.2.3 Empirical Results

We train a range of DGI models on the ArXiv dataset from the Open Graph Benchmark [Hu et al., 2020b], varying both model size N (ranging 2.8k from to 33k) and the number of samples D (ranging from 85m to 1.7b). Each of the experiment gives us a continuous curve of DGI loss w.r.t FLOPs. We drop the first 5% of epochs and collect the epochs with improvements in the SSL loss. We estimate parameters (A, B, E, α, β) by minimizing the log loss between the predicted and observed values:

$$\min_{A,B,E,\alpha,\beta} \sum_{\text{runs } j} (\log L_j - \log \hat{L}(N_j, D_j))^2, \tag{5.8}$$

using Adam stochastic optimization [Kingma, 2014].

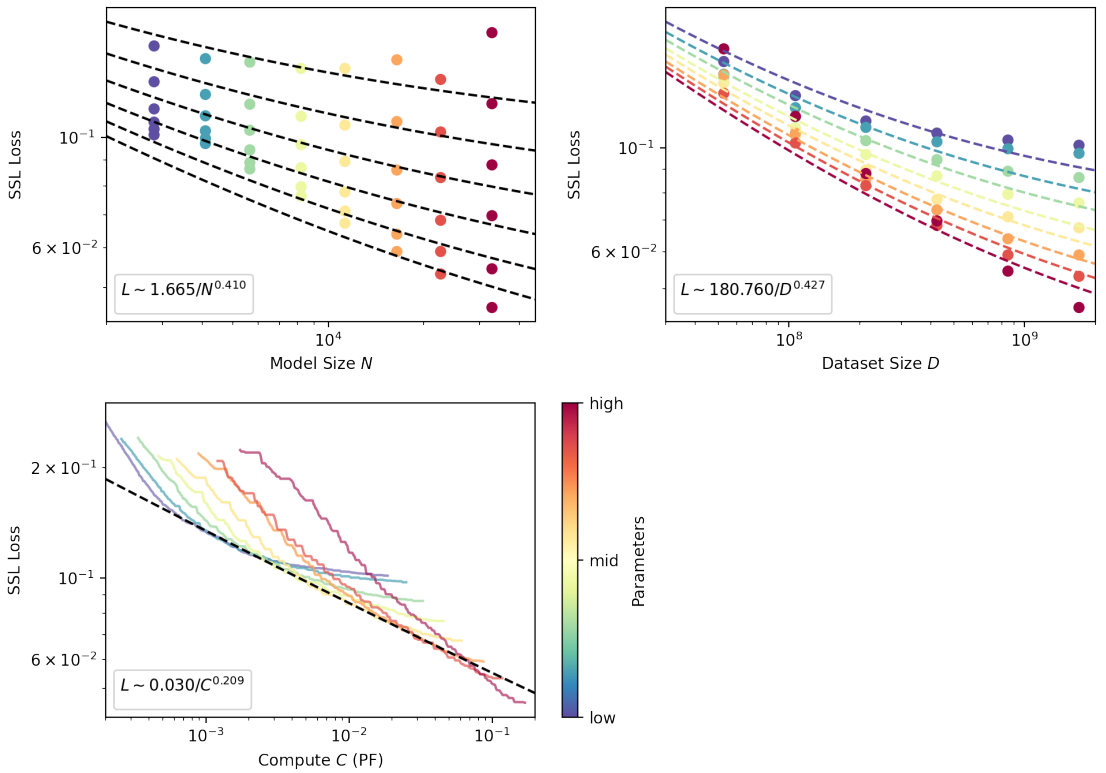


Figure 5.3: SSL loss scales as a empirical power function of the model size (upper left), training sample size (upper right), and amount of compute used for training (lower).

We summarize the scaling laws in Figure 5.3 as follows:

1. For models with a limited number of parameters, trained until convergence:

$$L(N) \sim \frac{A}{N^{\alpha}}, \quad A \sim 1.665, \alpha \sim 0.410. \tag{5.9}$$

2. For models with sufficiently large number of parameters, trained with limited epochs:

$$L(D) \sim \frac{B}{D^{\beta}}, \quad B \sim 180.760, \beta \sim 0.427. \tag{5.10}$$

3. Training an model with optimal allocation of a fixed compute budget (optimally sized

model and optimal number of steps):

$$L(C) \sim \frac{F}{C_{\min}^{\gamma}}, \quad F \sim 0.030, \gamma \sim 0.209. \tag{5.11}$$

The empirical results suggest that as the compute budget increases, both model size and the amount of training data should be increased in approximately equal proportions, in accordance with the observation in [Hoffmann et al., 2022]. It is also noticeable in the negative curvature in the efficient frontier, indicating the empirical results put more weight to points with significantly higher or lower FLOPs due to the square loss. The findings emphasize the importance of balancing model capacity and training data volume to achieve optimal performance.

5.3 Experiments

5.3.1 Datasets

We evaluate our method on five recent real-world datasets specifically chosen for their suitability in rigorously evaluating semi-supervised node classification methods [Mernyei and Cangea, 2020, Shchur et al., 2018]. Detailed statistics of these datasets are provided in Table 5.1.

1. WikiCS: A reference network from Wikipedia, where nodes represent articles about computer science and edges are hyperlinks. Node features are the average of pretrained GloVe word embeddings of words in each article. Nodes are labeled with ten classes representing branches of the field.

2. Amazon-Computers and Amazon-Photo: Networks of co-purchase relationships from Amazon, where nodes are goods and edges indicate items frequently bought together.

Nodes have sparse bag-of-words feature encodings of product reviews and are labeled with subcategories of Amazon goods.

3. Coauthor-CS and Coauthor-Physics: Academic networks from the Microsoft Academic Graph (KDD Cup 2016). Nodes represent authors, and edges indicate co-authorship. Nodes have sparse bag-of-words features based on paper keywords, labeled by the author's most active research field.

Table 5.1: Dataset Statistics.

The edge column lists the number of directed edges.

Dataset	Nodes	Edges	Features	Classes
WikiCS	11,701	216,123	300	10
Am. Comp.	13,752	245,861	767	10
Am. Photo	7,650	119,081	745	8
Co. CS	18,333	81,894	6,805	15
Co. Phy.	34,493	247,962	8,415	5

5.3.2 Methodology

We evaluate the performance of the optimized DGI and demonstrate that proportionately scaling the model size and dataset size can still outperform or match prior methods while utilizing limited computational resources. Following the evaluation scheme introduced in [Veličković et al., 2018], the optimized DGI model is initially trained in an unsupervised manner, and the resulting embeddings are used to train a simple classifier on the downstream task of node classification. We use a 2-layer GCN model as our graph encoder in pre-training, and a fully connected layer as the downstream model.

We primarily compare the performance of the optimized DGI against the original DGI and other self-supervised graph methods from published results [Zhu et al., 2021, Peng

et al., 2020]. To directly compare our proposed method with supervised counterparts, we also report the performance of GCN [Kipf and Welling, 2017], trained directly in end-to-end fashion. This allows us to measure the quality of the inductive biases present in the pre-trained model. For these baselines, the models, architectures, and graph-augmentation settings follow those used in prior works.

5.3.3 Results

The results are shown in Table 5.2. Overall, our proposed model demonstrates strong performance across all five datasets. The superior results of the optimized DGI compared to the original DGI highlight the benefits of our optimal computation allocation strategy. Comparing the DGI methods with the supervised GCN baseline, we observe that our optimized DGI method consistently matches or surpasses the performance of supervised learning. This further validates the importance of proportionately scaling model size and dataset size to achieve high performance with limited computational resources.

Table 5.2: Performance of optimized DGI on node classification.

Dataset	Supervised GCN	Optimized DGI	DGI	GMI
WikiCS	$0.7719_{\pm 0.0012}$	$0.7628_{\pm 0.0022}$	$0.7535_{\pm 0.0014}$	$0.7485_{\pm 0.0008}$
Am. Comp.	$0.8651_{\pm 0.0054}$	$0.8525_{\pm 0.0009}$	$0.8395_{\pm 0.0047}$	$0.8221_{\pm 0.0031}$
Am. Photo	$0.9242_{\pm 0.0022}$	$0.9150_{\pm 0.0004}$	$0.9161_{\pm 0.0022}$	$0.9068_{\pm 0.0017}$
Co. CS	$0.9303_{\pm 0.0031}$	$0.9320_{\pm 0.0008}$	$0.9215_{\pm 0.0063}$	OOM
Co. Phy.	$0.9565_{\pm 0.0016}$	$0.9530_{\pm 0.0004}$	$0.9451_{\pm 0.0052}$	OOM

CHAPTER 6

CONCLUSION

6.1 Contributions

This dissertation has made several significant contributions to the field of graph neural networks (GNNs).

Firstly, we have established a comprehensive framework for understanding the relationship between GNN architectures and their corresponding Gaussian Process (GP) equivalents (GNNGPs). We have extended this framework to include a variety of GNN architectures, such as GCNII, GraphSAGE, and GAT, demonstrating how these architectures can be translated into programmable covariance matrices within the GP paradigm.

Secondly, we have proposed efficient computational methods for the scalable computation of these kernels, addressing the challenges of applying GPs to large-scale graph data. Our methods have shown a significant reduction in computational complexity from cubic to near-linear, making it feasible to apply GPs to graphs with thousands or even millions of nodes.

Thirdly, we further consider the evolution of GNNGP kernels as we train the model on data and study how the kernel values change over time. We propose the Graph Neural Tangent Kernel (GNTK), which represents the kernel values obtained at the end of the training process. We demonstrate that the GNTK shares similar properties and performance characteristics with the GNNGP kernels, and it can be computed efficiently using our proposed methods.

Finally, we have conducted a thorough performance analysis of these kernels, exploring the trade-offs between model depth and oversmoothing, as well as between accuracy and computational time. Our findings provide valuable insights into the design of deep GNNs and highlight the potential of GNNGP models to achieve state-of-the-art performance on various graph datasets.

The contributions of this dissertation not only advance the theoretical understanding of GNNs and their GP counterparts but also provide practical tools and insights that can be leveraged by practitioners in the field.

6.2 Future Work

Looking ahead, there are several promising directions for future research. One area involves further exploration of the depth and width of GNNs to mitigate the oversmoothing problem without compromising the model's expressiveness. Another area of interest is the development of more sophisticated approximation techniques that can improve the efficiency of GNNGP models while maintaining high accuracy.

Additionally, the integration of GNNGP and GNTK frameworks presents an exciting avenue for research, offering the potential to better understand the learning dynamics of GNNs during training. Finally, the empirical scaling laws derived in this work lay the groundwork for future studies to optimize the computational resources required for training GNNs, which could lead to more efficient and powerful graph learning algorithms.

APPENDIX A

PROOFS AND ADDITIONAL THEOREMS

A.1 Proof of Theorem 1

For the purpose of clarity we add an intermediate variable $Y^{(l)}$ to the GCN as follows:

$$\begin{array}{lccccccccc} input = & X^{(0)} & & X^{(1)} & & \cdots & & X^{(L-1)} & & \\ & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & & \\ & Y^{(1)} & \phi & Y^{(2)} & \phi & \cdots & \phi & Y^{(L)} & & \\ & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & & \\ & Z^{(1)} & & Z^{(2)} & & \cdots & & Z^{(L)} & = output. & \end{array} \tag{A.1}$$

In matrix form:

$$\underset{N\times d_{l+1}}{Y^{(l+1)}} = \underset{N\times d_l}{X^{(l)}}\ \underset{d_l\times d_{l+1}}{W^{(l+1)}}, \tag{A.2}$$

$$\underset{N\times d_{l+1}}{Z^{(l+1)}} = \underset{N\times 1}{1}\ \underset{1\times d_{l+1}}{b^{(l+1)}} + \underset{N\times N}{A}\ \underset{N\times d_{l+1}}{Y^{(l+1)}}, \tag{A.3}$$

$$\underset{N\times d_l}{X^{(l)}} = \phi(\underset{N\times d_l}{Z^{(l)}}). \tag{A.4}$$

For $l = 1, \cdots, L$ and $i = 1, \cdots, d_l$, denote the i-th column of $Z^{(l)}$ by $z_i^{(l)} \in \mathbb{R}^N$. For a node $v \in V$, let $z_i^{(l)}(v)$ denote the element w.r.t node v. Similarly define $x_i^{(l)}(v)$ and $y_i^{(l)}(v)$, then specially $x^{(0)}(v)$ is the input of node v. Then we may rewrite the formula as

$$y_i^{(l+1)}(x) = \sum_{j=1}^{d_l} x_j^{(l)}(x) w_{ji}^{(l+1)}, \tag{A.5}$$

$$z_i^{(l+1)}(x) = b_i^{(l+1)} + \sum_{v\in V} A_{xv} y_i^{(l+1)}(v), \tag{A.6}$$

$$x_i^{(l)}(x) = \phi(z_i^{(l)}(x)). \tag{A.7}$$

We use mathematical induction. Assume each $z_i^{(l)}$ is i.i.d, then $x_i^{(l)}$ is also i.i.d after activation ϕ. Thus, $y_i^{(l+1)}(x)$ are infinite sum of i.i.d terms and thus normal, and by applying multivariate CLT $y_i^{(l+1)} \in \mathbb{R}^N$ is multivariate normal. Also for different i, the $y_i^{(l+1)}$ are identically distributed.

Moreover, terms like $y_i^{(l+1)}(x), y_{i'}^{(l+1)}(x')$ with $i \neq i'$ are jointly Gaussian, with zero covariance:

$$Cov(y_i^{(l+1)}(x), y_{i'}^{(l+1)}(x')) = \sum_{j=1}^{d_l}\sum_{j'=1}^{d_l} Cov\left(x_j^{(l)}(x)w_{ji}^{(l+1)}, x_{j'}^{(l)}(x')w_{j'i'}^{(l+1)}\right) = 0. \tag{A.8}$$

Thus, they are independent, despite they may share the same $X^{(l)}$ terms. In conclusion, each column of $Y^{(l+1)}$ is an i.i.d Gaussian Process. And hence, each column of $Z^{(l+1)}$ is also an i.i.d Gaussian Process $\mathcal{GP}(0, K^{(l+1)})$.

The covariance $K^{(l)}$ can be computed recursively. Similar to the case of a fully connected network, the covariance of $Y^{(l+1)}$ is

$$Cov(y_i^{(l+1)}(x), y_i^{(l+1)}(x')) = \sum_{j=1}^{d_l}\sum_{j'=1}^{d_l} Cov\left(x_j^{(l)}(x)w_{ji}^{(l+1)}, x_{j'}^{(l)}(x')w_{j'i}^{(l+1)}\right) \tag{A.9}$$

$$= \frac{\sigma_w^2}{d_l}\sum_{j=1}^{d_l}\sum_{j'=1}^{d_l} \delta_{jj'} E[x_j^{(l)}(x)x_{j'}^{(l)}(x')] \tag{A.10}$$

$$= \sigma_w^2 E[x_j^{(l)}(x)x_j^{(l)}(x')], \tag{A.11}$$

and in matrix form:

$$Cov(Y_i^{(l+1)}) = \sigma_w^2 E[X_j^{(l)}(X_j^{(l)})^T] := \sigma_w^2 C^{(l)}. \tag{A.12}$$

Since $Z_i^{(l+1)} = 1b_i^{(l+1)} + AY_i^{(l+1)}$, We know

$$K^{(l+1)} = \sigma_b^2 1_{N\times N} + ACov(Y_i^{(l+1)})A^T \tag{A.13}$$

$$= \sigma_b^2 1_{N\times N} + \sigma_w^2 AC^{(l)}A^T. \tag{A.14}$$

From the theorem $Z_i^{(l)} \sim \mathcal{GP}(0, K^{(l)})$, so

$$C^{(l)} = E[X_i^{(l)}(X_i^{(l)})^T] = E_{z\sim\mathcal{N}(0,K^{(l)})}[\phi(z)\phi(z)^T]. \tag{A.15}$$

Specially, when we focus on a particular element $C^{(l)}(x, x')$ the expectation can be computed by integration over the marginal distribution of $z \sim \mathcal{N}(0, K^{(l)})$:

$$\begin{pmatrix} z(x) \\ z(x') \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K^{(l)}(x,x) & K^{(l)}(x,x') \\ K^{(l)}(x',x) & K^{(l)}(x',x') \end{pmatrix}\right). \tag{A.16}$$

A.2 Proof of Theorem 2

We first establish the following lemma that states that the kernel is universal over a half-space.

Lemma 1. *The arc-cosine kernel is universal on the upper-hemisphere*

$$S = \left\{x \in \mathbb{R}^d : \|x\|_2 = 1, x_1 > 0\right\}, \tag{A.17}$$

for all $d \geq 2$.

Proof. We know that $\phi(w \cdot x)$ is the kernel representation of the arc-cosine kernel. It suffices to show for different $x_1, \cdots, x_m \in S$, functions

$$\varphi_{x_i}(w) = \phi(w \cdot x_i), \quad i = 1, \cdots, m, \tag{A.18}$$

are linearly independent.

We use proof by contradiction: Assume there exists $c_1, \cdots, c_m$ nonzero, such that

$$\sum_{i=1}^{m} c_i \phi(w \cdot x_i) = \sum_{i=1}^{m} c_i \varphi_{x_i}(w) \equiv 0. \tag{A.19}$$

Without loss of generality, further assume m is the smallest in all that satisfies (A.19).

Let $e_1 = (1, 0, \cdots, 0)$, then $x \cdot e_1 > 0, \forall x \in S$. Assume $x_m \cdot e_1 = \min_{1 \le i \le m} x_i \cdot e_1 > 0$. Then for $1 \le i \le m-1$,

$$\frac{x_i \cdot x_m}{x_i \cdot e_1} < \frac{1}{x_m \cdot e_1} = \frac{x_m \cdot x_m}{x_m \cdot e_1}. \tag{A.20}$$

So there exists t such that

$$\frac{x_i \cdot x_m}{x_i \cdot e_1} < t < \frac{x_m \cdot x_m}{x_m \cdot e_1}, \quad 1 \le i \le m-1. \tag{A.21}$$

Let $w = x_m - te_1$, then

$$w \cdot x_m > 0, w \cdot x_i < 0, \quad 1 \le i \le m-1. \tag{A.22}$$

Using (A.19) we know $c_m(w \cdot x_m) = 0$, so $c_m = 0$. Thus, $c_1, \cdots, c_{m-1}$ and $\varphi_{x_1}, \cdots, \varphi_{x_{m-1}}$ also satisfies (A.19) with a smaller number m, which forms a contradiction. □

Now with Lemma 1, we come back to the proof of Theorem 2.

Proof. By our assumption, $C^{(0)}$ does not contain two parallel rows. Using the recursive relation (2.6), we know $K^{(0)}$ is element-wise non-negative, hence $K^{(1)}$ is element-wise positive. Assume the Cholesky decomposition $K^{(1)} = LL^T$ and denote $L = (l_1, \cdots, l_n)^T$ and $r_i = \|l_i\|_2 > 0, x_i = \frac{l_i}{r_i}$. Then $x_1 = \pm e_1$, and without loss of generality, we assume $x_1 = e_1$. Then

for each x_i, we know

$$x_i \cdot e_1 = \frac{l_i \cdot l_1}{r_i r_1} = \frac{K^{(1)}(i,1)}{r_i r_1} > 0, \tag{A.23}$$

and each $\|x_i\|_2 = 1$, therefore $x_i \in S$ defined in Lemma 1.

By our assumption $K^{(1)}$ also does not contain two parallel rows. Thus, l_i are pairwise non-parallel, and $x_i \in S$ are pairwise distinct. Using Lemma 1, we know $C^{(2)} = g(K^{(1)})$ is positive definite, hence $K^{(2)}$ is positive definite. □

A.3 Proof of Theorem 3

For a covariance matrix K, denote its standard deviance and correlation by

$$\sigma_K(x) = \sqrt{K(x,x)}, \tag{A.24}$$

$$\rho_K(x,x') = \frac{K(x,x')}{\sqrt{K(x,x)K(x',x')}}. \tag{A.25}$$

From the closed-form formula (2.10) for the arc-cosine kernel, we have

$$\rho_{C^{(l)}}(x,x') = \frac{1}{\pi}\left(\sin\theta^{(l)}_{x,x'} + (\pi - \theta^{(l)}_{x,x'})\cos\theta^{(l)}_{x,x'}\right), \tag{A.26}$$

$$\rho_{K^{(l)}}(x,x') = \cos\theta^{(l)}_{x,x'}. \tag{A.27}$$

We define the correlation mapping from $\rho_{K^{(l)}}$ to $\rho_{C^{(l)}}$:

$$f : \cos\theta \mapsto \frac{1}{\pi}\left(\sin\theta + (\pi - \theta)\cos\theta\right), \quad \theta \in [0,\pi]. \tag{A.28}$$

We first establish a few properties of f. A pictorial illustration of f is given in Figure A.1.

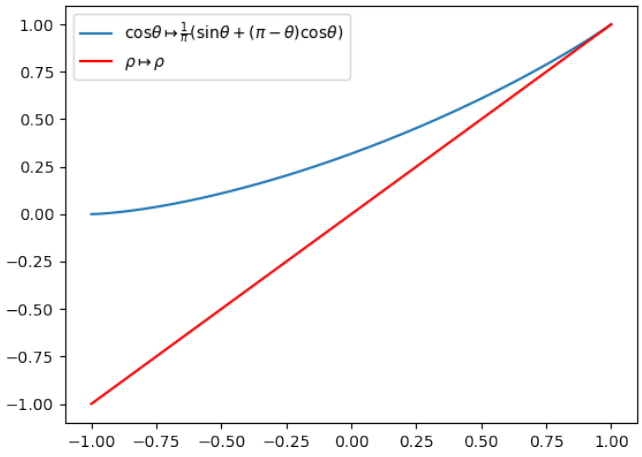


Figure A.1: Correlation mapping f.

Lemma 2. *The following facts hold.*

1. *f is increasing and f is a contraction mapping on $[-1, 1]$.*

2. *$f(\rho) \geq \rho$ and the equation holds only when $\rho = 1$.*

Proof.

Denote $\rho = \cos\theta \in [-1, 1]$. Then,

$$f'(\rho) = \frac{\partial f(\rho)}{\partial \theta}\frac{\partial \theta}{\partial \rho} = \frac{-1}{\pi}(\pi - \theta)\sin\theta \cdot \frac{-1}{\sin\theta} = \frac{\pi - \theta}{\pi} \in [0, 1], \tag{A.29}$$

so f is increasing and f is a contraction mapping on $[-1, 1]$.

We know that $f(1) = 1$. Using the contraction mapping we know $f(\rho) \geq \rho$ for $\rho \in [-1, 1]$, and the equation holds only when $\rho = 1$. □

Now, we are ready to prove Theorem 3.

Proof of Theorem 3, part 1. From Lemma 2 for the correlation mapping f, we have

$$\rho_{K^{(l+1)}}(x, x') = f(\rho_{C^{(l)}}(x, x')) \geq \rho_{K^{(l)}}(x, x'). \tag{A.30}$$

Taking the minimum over all pairs of x and x', we have

$$\rho_{\min}(K^{(l+1)}) = f(\rho_{\min}(C^{(l)})) \geq \rho_{\min}(K^{(l)}). \tag{A.31}$$

so $\rho_{\min}(K^{(l)}) \nearrow$ some $\rho \in [-1, 1]$ with $\rho = f(\rho)$, hence $\rho = 1$. □

Proof of Theorem 3, part 2. Start with diagonal elements. We have

$$K^{(l+1)}(x, x) = \sigma_b^2 + \frac{\sigma_w^2}{2} K^{(l)}(x, x). \tag{A.32}$$

so $K^{(l)}(x, x) \to q = \frac{\sigma_b^2}{1-\sigma_w^2/2}$ linearly.

For the off-diagonal elements, we have

$$K^{(l+1)}(x, x') = \sigma_b^2 + \frac{\sigma_w^2}{2} \sigma_{K^{(l)}}(x) \sigma_{K^{(l)}}(x') f\left(\rho_{K^{(l)}}(x, x')\right) \tag{A.33}$$

$$\geq \sigma_b^2 + \frac{\sigma_w^2}{2} \sigma_{K^{(l)}}(x) \sigma_{K^{(l)}}(x') \rho_{K^{(l)}}(x, x') \tag{A.34}$$

$$= \sigma_b^2 + \frac{\sigma_w^2}{2} K^{(l)}(x, x'), \tag{A.35}$$

so

$$\liminf_l K^{(l)}(x, x') \geq \sigma_b^2 + \frac{\sigma_w^2}{2} \liminf_l K^{(l)}(x, x') \implies \liminf_l K^{(l)}(x, x') \geq q. \tag{A.36}$$

Also

$$\limsup_l K^{(l)}(x, x') \leq \limsup_l \sqrt{K^{(l)}(x, x) K^{(l)}(x', x')} = q. \tag{A.37}$$

so $K^{(l)}(x,x') \to q$ linearly. In conclusion, $K^{(l)} \to q1_{N\times N}$ linearly. $\square$

Proof of Theorem 3, part 3. Again start with diagonal elements. When $\sigma_w^2 > 2$, we have

$$\kappa^{(l+1)}(x,x) = \frac{\sigma_b^2}{(\sigma_w^2/2)^{l+1}} + \kappa^{(l)}(x,x), \tag{A.38}$$

so

$$\lim_{l\to\infty} \kappa^{(l)}(x,x) = \sum_{l=0}^{\infty} \frac{\sigma_b^2}{(\sigma_w^2/2)^{l+1}} + \kappa^{(0)}(x,x) = \frac{\sigma_b^2}{\sigma_w^2/2-1} + K^{(0)}(x,x) = v_x^2. \tag{A.39}$$

For the off-diagonal elements, we have

$$\kappa^{(l+1)}(x,x') = \frac{\sigma_b^2}{c_{l+1}} + \sigma_{\kappa^{(l)}}(x)\sigma_{\kappa^{(l)}}(x') f\left(\rho_{\kappa^{(l)}}(x,x')\right) \tag{A.40}$$

$$\geq \sigma_{\kappa^{(l)}}(x)\sigma_{\kappa^{(l)}}(x')\rho_{\kappa^{(l)}}(x,x') \tag{A.41}$$

$$= \kappa^{(l)}(x,x'), \tag{A.42}$$

so $\kappa^{(l)}(x,x')$ has a limit c. Taking the limits on both side, we have

$$c = v_x v_{x'} \cdot f\left(c/(v_x v_{x'})\right), \tag{A.43}$$

so $c = v_x v_{x'}$. In conclusion, $\kappa^{(l)} \to vv^T$. $\square$

A.4 Proof of Theorem 4

After the correlation mapping f in Lemma 2, we further consider the covariance mapping $g : K^{(l)} \mapsto C^{(l)}$ defined in (2.6). We establish a few properties of g below.

Lemma 3. *For positive semi-definite matrices $B, C \in \mathbb{R}^{N\times N}$, define*

$$\operatorname{dist}(B,C) = \min_{YY^T=B, ZZ^T=C} \|Y-Z\|_F. \tag{A.44}$$

Then

1. $\operatorname{dist}(\cdot,\cdot)$ *is a valid distance function on the set of positive semi-definite matrices, with*

$$\operatorname{dist}(B,C)^2 = \operatorname{tr}(B) + \operatorname{tr}(C) - 2\|B^{1/2}C^{1/2}\|_*, \tag{A.45}$$

   where $(\cdot)^{1/2}$ is the (symmetric) square root, and $\|\cdot\|_*$ is the nuclear norm.

2. $\operatorname{dist}(g(B), g(C)) \le \sqrt{\frac{1}{2}} \cdot \operatorname{dist}(B,C)$.

3. $\operatorname{dist}(B+B_1, C+C_1) \le \sqrt{\operatorname{dist}(B,C)^2 + \operatorname{dist}(B_1,C_1)^2}$.

4. $\operatorname{dist}\left(ABA^T, ACA^T\right) \le \|A\|_2 \cdot \operatorname{dist}(B,C)$.

Proof. 1. Rewrite the definition (A.44) as

$$\operatorname{dist}(B,C) = \min_{YY^T=B} \min_{ZZ^T=C} \|Y-Z\|_F. \tag{A.46}$$

For a given Y_0 with $Y_0Y_0^T = B$, the sub-problem

$$\min_{ZZ^T=C} \|Y_0 - Z\|_F, \tag{A.47}$$

is a orthogonal Procrustes problem. Choose an arbitrary Z with $ZZ^T = C$, and use singular value decomposition $Y_0^T Z = U\Sigma V^T$, the solution is

$$Z_0 = ZVU^T. \tag{A.48}$$

Note that $Y_0^T Z_0 = U\Sigma U^T$ is positive semi-definite, and the minimum

$$\min_{ZZ^T=C} \|Y_0 - Z\|_F^2 = \|Y_0 - Z_0\|_F^2 \tag{A.49}$$

$$= \mathrm{tr}\left(Y_0 Y_0^T\right) + \mathrm{tr}\left(Z_0 Z_0^T\right) - 2 \cdot \mathrm{tr}\left(Y_0^T Z_0\right) \tag{A.50}$$

$$= \mathrm{tr}\,(B) + \mathrm{tr}\,(C) - 2\|Y_0^T Z_0\|_* \tag{A.51}$$

$$= \mathrm{tr}\,(B) + \mathrm{tr}\,(C) - 2\|B^{1/2} C^{1/2}\|_*, \tag{A.52}$$

is invariant w.r.t the choice of Y_0. Therefore,

$$\mathrm{dist}\,(B, C) = \|Y_0 - Z_0\|_F. \tag{A.53}$$

Now for a positive semi-definite A, there exists X_0 such that $X_0 X_0^T = A$, and

$$\mathrm{dist}\,(A, B) = \|X_0 - Y_0\|_F. \tag{A.54}$$

Therefore

$$\mathrm{dist}\,(A, B) + \mathrm{dist}\,(B, C) = \|X_0 - Y_0\|_F + \|Y_0 - Z_0\|_F \geq \|X_0 - Z_0\|_F \geq \mathrm{dist}\,(A, C)\,, \tag{A.55}$$

so $\mathrm{dist}\,(\cdot, \cdot)$ satisfies the triangle inequality. Easy to see it also satisfies positivity and symmetry, so we conclude that $\mathrm{dist}\,(\cdot, \cdot)$ is a valid distance function.

2. Assume $\mathrm{dist}\,(B, C) = \|Y_0 - Z_0\|_F$. Denote

$$Y_0^T = (y_1, \cdots, y_N), Z_0^T = (z_1, \cdots, z_N), \tag{A.56}$$

then

$$\text{dist}\,(B,C)^2 = \|Y_0 - Z_0\|_F^2 = \sum_{i=1}^{N} \|y_i - z_i\|_2^2 = \sum_{i=1}^{N} [\langle y_i, y_i\rangle + \langle z_i, z_i\rangle - 2\langle y_i, z_i\rangle], \tag{A.57}$$

consists of only inner products. Now, using kernel representation of g:

$$\varphi_{y_i}(w) = \phi(w \cdot y_i), \quad \varphi_{z_i}(w) = \phi(w \cdot z_i), \tag{A.58}$$

we have a similar expression of inner products:

$$\text{dist}\,(g(B), g(C))^2 \leq \sum_{i=1}^{N} \|\varphi_{y_i} - \varphi_{z_i}\|_2^2 = \sum_{i=1}^{N} [\langle \varphi_{y_i}, \varphi_{y_i}\rangle + \langle \varphi_{z_i}, \varphi_{z_i}\rangle - 2\langle \varphi_{y_i}, \varphi_{z_i}\rangle]. \tag{A.59}$$

From Lemma 2 for the correlation mapping f, we have

$$\langle \varphi_{y_i}, \varphi_{y_i}\rangle = \frac{1}{2}\langle y_i, y_i\rangle, \tag{A.60}$$

$$\langle \varphi_{z_i}, \varphi_{z_i}\rangle = \frac{1}{2}\langle z_i, z_i\rangle, \tag{A.61}$$

$$\langle \varphi_{y_i}, \varphi_{z_i}\rangle = \frac{1}{2}\sqrt{\langle y_i, y_i\rangle\langle z_i, z_i\rangle} f(\rho_{y_i,z_i}) \geq \frac{1}{2}\sqrt{\langle y_i, y_i\rangle\langle z_i, z_i\rangle}\rho_{y_i,z_i} = \frac{1}{2}\langle y_i, z_i\rangle. \tag{A.62}$$

Combining the results, we have

$$\text{dist}\,(g(B), g(C))^2 \leq \sum_{i=1}^{N} [\langle \varphi_{y_i}, \varphi_{y_i}\rangle - 2\langle \varphi_{y_i}, \varphi_{z_i}\rangle + \langle \varphi_{z_i}, \varphi_{z_i}\rangle] \tag{A.63}$$

$$\leq \frac{1}{2}\sum_{i=1}^{N} [\langle y_i, y_i\rangle - 2\langle y_i, z_i\rangle + \langle z_i, z_i\rangle] \tag{A.64}$$

$$= \frac{1}{2}\text{dist}\,(B,C)^2\,. \tag{A.65}$$

3. Note that, in the definition (A.44)

$$\operatorname{dist}(B, C) = \min_{YY^T=B, ZZ^T=C} \|Y - Z\|_F, \tag{A.66}$$

Y and Z are not required to be a $N \times N$ square matrix, but rather any matrix of shape $N \times M$ and the definition remains the same.

Assume $\operatorname{dist}(B, C) = \|Y_0 - Z_0\|_F$ and $\operatorname{dist}(B_1, C_1) = \|Y_1 - Z_1\|_F$. Then

$$\begin{aligned}
\operatorname{dist}(B + B_1, C + C_1) &\leq \| \begin{pmatrix} Y_0 & Y_1 \end{pmatrix} - \begin{pmatrix} Z_0 & Z_1 \end{pmatrix} \|_F && \text{(A.67)} \\
&= \sqrt{\|Y_0 - Z_0\|_F^2 + \|Y_1 - Z_1\|_F^2} && \text{(A.68)} \\
&= \sqrt{\operatorname{dist}(B, C)^2 + \operatorname{dist}(B_1, C_1)^2}. && \text{(A.69)}
\end{aligned}$$

4. Assume $\operatorname{dist}(B, C) = \|Y_0 - Z_0\|_F$. Then

$$\begin{aligned}
\operatorname{dist}\left(ABA^T, ACA^T\right) &\leq \|AY_0 - AZ_0\|_F && \text{(A.70)} \\
&\leq \|A\|_2 \|Y_0 - Z_0\|_F && \text{(A.71)} \\
&= \|A\|_2 \cdot \operatorname{dist}(B, C). && \text{(A.72)}
\end{aligned}$$

□

Now, we are ready to prove Theorem 4.

Proof of Theorem 4, part 1. When $\sigma_b^2 = 0$, the recursion (2.6) can be simplified as

$$K^{(l+1)} = \sigma_w^2 A C^{(l)} A^T, \quad C^{(l)} = g(K^{(l)}). \tag{A.73}$$

To simplify notation, we temporarily write $C = C^{(l)}$. By the definition of $\rho_{\min}(C)$,

$$C(x, x') \geq \rho_{\min}(C) \cdot \sigma_C(x)\sigma_C(x'), \quad \forall x, x'. \tag{A.74}$$

Therefore

$$K^{(l+1)}(x, x') = \sigma_w^2 \sum_{v,v'} A_{xv} A_{x'v'} C(v, v') \tag{A.75}$$

$$\geq \rho_{\min}(C) \cdot \sigma_w^2 \sum_{v,v'} A_{xv} A_{x'v'} \sigma_C(v)\sigma_C(v') \tag{A.76}$$

$$= \rho_{\min}(C) \cdot \sigma_w^2 \left(\sum_v A_{xv} \sigma_C(v) \right) \left(\sum_{v'} A_{x'v'} \sigma_C(v') \right). \tag{A.77}$$

Also

$$K^{(l+1)}(x, x) = \sigma_w^2 \sum_{v,v'} A_{xv} A_{xv'} C(v, v') \tag{A.78}$$

$$\leq \sigma_w^2 \sum_{v,v'} A_{xv} A_{xv'} \sigma_C(v)\sigma_C(v') \tag{A.79}$$

$$= \sigma_w^2 \left(\sum_v A_{xv} \sigma_C(v) \right)^2, \tag{A.80}$$

and similarly

$$K^{(l+1)}(x', x') \leq \sigma_w^2 \left(\sum_{v'} A_{x'v'} \sigma_C(v') \right)^2. \tag{A.81}$$

Combining the results, we obtain

$$\rho(K^{(l+1)})(x, x') = \frac{K^{(l+1)}(x, x')}{\sqrt{K^{(l+1)}(x, x) K^{(l+1)}(x', x')}} \geq \rho_{\min}(C^{(l)}), \quad \forall x, x'. \tag{A.82}$$

Hence

$$\rho_{\min}(K^{(l+1)}) \geq \rho_{\min}(C^{(l)}). \tag{A.83}$$

From Lemma 2 for the correlation mapping f, we have

$$\rho_{\min}(C^{(l)}) = f(\rho_{\min}(K^{(l)})), \tag{A.84}$$

and by the properties of f,

$$\rho_{\min}(K^{(l+1)}) \geq f(\rho_{\min}(C^{(l)})) \geq \rho_{\min}(K^{(l)}), \tag{A.85}$$

so $\rho_{\min}(K^{(l)}) \nearrow$ some $\rho \in [-1, 1]$ with $\rho = f(\rho)$, hence $\rho = 1$. □

Proof of Theorem 4, part 2. Consider mapping $h : K^{(l)} \mapsto K^{(l+1)}$ at each layer:

$$K^{(l+1)} = \sigma_b^2 1_{N\times N} + \sigma_w^2 A g(K^{(l)}) A^T := h(K^{(l)}). \tag{A.86}$$

Note that

$$\begin{aligned}
\operatorname{dist}\left(h(K^{(l)}), h(\tilde{K}^{(l)})\right) &= \operatorname{dist}\left(\sigma_b^2 + \sigma_w^2 A g(K^{(l)}) A^T, \sigma_b^2 + \sigma_w^2 A g(\tilde{K}^{(l)}) A^T\right) && \text{(A.87)}\\
&\leq \operatorname{dist}\left(\sigma_w^2 A g(K^{(l)}) A^T, \sigma_w^2 A g(\tilde{K}^{(l)}) A^T\right) && \text{(A.88)}\\
&\leq \sqrt{\sigma_w^2 r^2} \cdot \operatorname{dist}\left(g(K^{(l)}), g(\tilde{K}^{(l)})\right) && \text{(A.89)}\\
&\leq \sqrt{\frac{1}{2}\sigma_w^2 r^2} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right) && \text{(A.90)}\\
&= \delta^{1/2} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right), && \text{(A.91)}
\end{aligned}$$

so h is a contraction mapping. Using the Banach fixed-point theorem, we know h has a

unique fixed point K, such that

$$K = \sigma_b^2 1_{N\times N} + \sigma_w^2 Ag(K)A^T = h(K). \tag{A.92}$$

So K is dependent only on A, σ_b^2 and σ_w^2. Let $\tilde{K}^{(l)} = K$, we have

$$\text{dist}\left(K^{(l+1)}, K\right) \leq \delta^{1/2} \cdot \text{dist}\left(K^{(l)}, K\right), \tag{A.93}$$

so $K^{(l)}$ converges to K linearly with a rate of convergence of $\delta^{1/2}$. □

Proof of Theorem 4, part 3. We have recursive relationship between $\kappa^{(l)}$ and $\kappa^{(l+1)}$:

$$\kappa^{(l+1)} = \frac{1}{\delta^{l+1}}(\sigma_b^2 1_{N\times N} + \sigma_w^2 Ag(K^{(l)})A^T) = \frac{\sigma_b^2}{\delta^{l+1}} 1_{N\times N} + \frac{2}{r^2} Ag(\kappa^{(l)})A^T. \tag{A.94}$$

Using Lemma 3 for the covariance mapping g,

$$\text{dist}\left(\kappa^{(l+1)}, 0\right) \leq \text{dist}\left(\frac{\sigma_b^2}{\delta^{l+1}} 1_{N\times N}, 0\right) + \text{dist}\left(\frac{2}{r^2} Ag(\kappa^{(l)})A^T, 0\right) \tag{A.95}$$

$$\leq \sqrt{\frac{N\sigma_b^2}{\delta^{l+1}}} + \text{dist}\left(2g(\kappa^{(l)}), 0\right) \tag{A.96}$$

$$\leq \sqrt{\frac{N\sigma_b^2}{\delta^{l+1}}} + \text{dist}\left(\kappa^{(l)}, 0\right), \tag{A.97}$$

so $\kappa^{(l)}$ is bounded.

Using the Perron-Frobenius theorem, we know $v \succ 0$. Then

$$v^T \kappa^{(l+1)} v = \frac{\sigma_b^2}{c_{l+1}} v^T 1_{N\times N} v + \frac{2}{r^2} v^T Ag(\kappa^{(l)})A^T v \tag{A.98}$$

$$\geq 2v^T g(\kappa^{(l)}) v \tag{A.99}$$

$$\geq v^T \kappa^{(l)} v, \tag{A.100}$$

so $v^T \kappa^{(l)} v$ has a limit $c > 0$.

Also from the Perron-Frobenius theorem, r is a simple eigenvalue, and there is no other eigenvalues with absolute value $\geq r$. Let $P = I - \text{Proj}_v = I - vv^T$, then

$$PA = AP, \quad \|PA\|_2 = r' < r. \tag{A.101}$$

then

$$\begin{aligned}
\text{dist}\left(Pg(\kappa^{(l)})P^T, 0\right)^2 &= \text{tr}\left(Pg(\kappa^{(l)})P^T\right) && \text{(A.102)}\\
&= \text{tr}\left(g(\kappa^{(l)})\right) - v^T g(\kappa^{(l)}) v && \text{(A.103)}\\
&\leq \frac{1}{2}\text{tr}\left(\kappa^{(l)}\right) - \frac{1}{2} v^T \kappa^{(l)} v && \text{(A.104)}\\
&= \text{tr}\left(\frac{1}{2} P\kappa^{(l)} P^T\right) && \text{(A.105)}\\
&= \text{dist}\left(\frac{1}{2} P\kappa^{(l)} P^T, 0\right)^2. && \text{(A.106)}
\end{aligned}$$

Using Lemma 3 for the covariance mapping g,

$$\begin{aligned}
\text{dist}\left(P\kappa^{(l+1)}P^T, 0\right) &\leq \text{dist}\left(\frac{\sigma_b^2}{\delta^{l+1}} P 1_{N\times N} P^T, 0\right) + \text{dist}\left(\frac{2}{r^2} PAg(\kappa^{(l)})A^T P^T, 0\right) && \text{(A.107)}\\
&\leq \sqrt{\frac{N\sigma_b^2}{\delta^{l+1}}} + \text{dist}\left(\frac{2}{r^2} PAPg(\kappa^{(l)})P^T A^T P^T, 0\right) && \text{(A.108)}\\
&\leq \sqrt{\frac{N\sigma_b^2}{\delta^{l+1}}} + \frac{r'}{r}\text{dist}\left(2Pg(\kappa^{(l)})P^T, 0\right) && \text{(A.109)}\\
&\leq \sqrt{\frac{N\sigma_b^2}{\delta^{l+1}}} + \frac{r'}{r}\text{dist}\left(P\kappa^{(l)}P^T, 0\right), && \text{(A.110)}
\end{aligned}$$

so $P\kappa^{(l)}P^T \to 0$.

Combining the results, we conclude that $\kappa^{(l)} \to cvv^T$. □

A.5 Proof of Section 3.3

For initial residual and identity mapping used in GCNII (3.10) and GraphSAGE (3.14), the key is the independent sum in Table 3.1. This building block is applicable here, as the independence is guaranteed by the following lemma:

Lemma 4. *Assume the input X passed through a fully connected layer $X \leftarrow XW$, where the weight term W has width $d \to \infty$ and elementwise i.i.d $\sim \mathcal{N}(0, \frac{\sigma_w^2}{d})$. Then X and XW are elementwise independent; Further assume input X passed through two independent fully connected layers, resulting in two outputs XW_1 and XW_2, then XW_1 and XW_2 are elementwise independent.*

Proof. For a finite dimensional distribution of X and XW, the number of shared weights in the form of w_{ij} are also finite. With $d \to \infty$, all these terms has a limit of 0, meaning the input X and output XW are independent.
For a finite dimensional distribution of XW_1 and XW_2, the number of shared weights in the form of x_{ij} are also finite. With $d_1, d_2 \to \infty$, they are jointly zero-mean normal distributed because of multivariate CLT, and their products are sum of random variables symmetrically distributed around 0. Thus these entries are uncorrelated and thus independent. □

For self-attention layers in GAT (3.21), the proof is directly derived from Theorem 1:

Proof. Let

$$Y = XW, \quad Z = \zeta(E)Y. \tag{A.111}$$

Using Theorem 1, each column of Y is an i.i.d Gaussian Process $\mathcal{GP}(0, \sigma_w^2 C)$. Hence

$$E = d^{-1}XWW^TX = d^{-1}\sum_{i=1}^{d} Y_i(Y_i)^T \to \sigma_w^2 C. \tag{A.112}$$

Therefore, each column of Z is an i.i.d Gaussian Process:

$$Z_i \sim \mathcal{GP}\left(0, \sigma_w^2 \zeta(\sigma_w^2 C) C \zeta(\sigma_w^2 C)^T\right). \tag{A.113}$$

□

A.6 Proof of Theorem 5

Proof of Theorem 5, part 1. When $\alpha = 0$, the recursion (3.10) can be simplified as

$$K^{(l+1)} = ((1-\beta)^2 + \beta^2\sigma_w^2) A g(K^{(l)}) A^T, \tag{A.114}$$

which is same up to a constant as Theorem 4, part 1. The rest is the same. □

Proof of Theorem 5, part 2. Consider mapping $h : K^{(l)} \mapsto K^{(l+1)}$ at each layer:

$$K^{(l+1)} = \left((1-\alpha)^2 A g(K^{(l)}) A^T + \alpha^2 C^{(0)}\right)((1-\beta)^2 + \beta^2\sigma_w^2) := h(K^{(l)}). \tag{A.115}$$

Note that

$$\begin{aligned}
&\operatorname{dist}\left(h(K^{(l)}), h(\tilde{K}^{(l)})\right) && \text{(A.116)}\\
&= \sqrt{(1-\beta)^2 + \beta^2\sigma_w^2} \cdot \operatorname{dist}\left((1-\alpha)^2 A g(K^{(l)}) A^T + \alpha^2 C^{(0)}, (1-\alpha)^2 A g(\tilde{K}^{(l)}) A^T + \alpha^2 C^{(0)}\right) && \text{(A.117)}\\
&\le \sqrt{(1-\beta)^2 + \beta^2\sigma_w^2} \cdot \operatorname{dist}\left((1-\alpha)^2 A g(K^{(l)}) A^T, (1-\alpha)^2 A g(\tilde{K}^{(l)}) A^T\right) && \text{(A.118)}\\
&\le \sqrt{((1-\beta)^2 + \beta^2\sigma_w^2)(1-\alpha)^2 r^2} \cdot \operatorname{dist}\left(g(K^{(l)}), g(\tilde{K}^{(l)})\right) && \text{(A.119)}\\
&\le \sqrt{\frac{1}{2}((1-\beta)^2 + \beta^2\sigma_w^2)(1-\alpha)^2 r^2} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right) && \text{(A.120)}\\
&= \delta^{1/2} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right), && \text{(A.121)}
\end{aligned}$$

so h is a contraction mapping, and has a unique fixed point K. Let $\tilde{K}^{(l)} = K$, we have

$$\text{dist}\left(K^{(l+1)}, K\right) \leq \delta^{1/2} \cdot \text{dist}\left(K^{(l)}, K\right), \tag{A.122}$$

so $K^{(l)}$ converges to K linearly with a rate of convergence of $\delta^{1/2}$. □

Proof of Theorem 5, part 3. We have recursive relationship between $\kappa^{(l)}$ and $\kappa^{(l+1)}$:

$$\kappa^{(l+1)} = \frac{(1-\beta)^2 + \beta^2\sigma_w^2}{\delta^{l+1}}\left((1-\alpha)^2 Ag(K^{(l)})A^T + \alpha^2 C^{(0)}\right) \tag{A.123}$$

$$= \frac{((1-\beta)^2 + \beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}}C^{(0)} + \frac{2}{r^2}Ag(\kappa^{(l)})A^T. \tag{A.124}$$

Using Lemma 3 for the covariance mapping g,

$$\text{dist}\left(\kappa^{(l+1)}, 0\right) \leq \text{dist}\left(\frac{((1-\beta)^2 + \beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}}C^{(0)}, 0\right) + \text{dist}\left(\frac{2}{r^2}Ag(\kappa^{(l)})A^T, 0\right) \tag{A.125}$$

$$\leq \sqrt{\frac{((1-\beta)^2 + \beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}}\text{tr}\left(C^{(0)}\right)} + \text{dist}\left(2g(\kappa^{(l)}), 0\right) \tag{A.126}$$

$$\leq \sqrt{\frac{((1-\beta)^2 + \beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}}\text{tr}\left(C^{(0)}\right)} + \text{dist}\left(\kappa^{(l)}, 0\right), \tag{A.127}$$

so $\kappa^{(l)}$ is bounded.

Using the Perron-Frobenius theorem, we know $v \succ 0$. Then

$$v^T\kappa^{(l+1)}v = \frac{((1-\beta)^2 + \beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}}v^TC^{(0)}v + \frac{2}{r^2}v^TAg(\kappa^{(l)})A^Tv \tag{A.128}$$

$$\geq 2v^Tg(\kappa^{(l)})v \tag{A.129}$$

$$\geq v^T\kappa^{(l)}v, \tag{A.130}$$

so $v^T\kappa^{(l)}v$ has a limit $c > 0$.

Also from the Perron-Frobenius theorem, r is a simple eigenvalue, and there is no other

eigenvalues with absolute value $\geq r$. Let $P = I - \text{Proj}_v = I - vv^T$, then

$$PA = AP, \quad \|PA\|_2 = r' < r. \tag{A.131}$$

Same as Theorem 4, part 3, we have

$$\text{dist}\left(P g(\kappa^{(l)}) P^T, 0\right) \leq \text{dist}\left(\frac{1}{2} P \kappa^{(l)} P^T, 0\right). \tag{A.132}$$

Using Lemma 3 for the covariance mapping g,

$$\begin{aligned}
&\text{dist}\left(P\kappa^{(l+1)}P^T, 0\right) && \text{(A.133)}\\
&\leq \text{dist}\left(\frac{((1-\beta)^2+\beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}} P C^{(0)} P^T, 0\right) + \text{dist}\left(\frac{2}{r^2} PAg(\kappa^{(l)})A^T P^T, 0\right) && \text{(A.134)}\\
&\leq \sqrt{\frac{((1-\beta)^2+\beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}} \text{tr}\left(C^{(0)}\right)} + \text{dist}\left(\frac{2}{r^2} PAPg(\kappa^{(l)})P^T A^T P^T, 0\right) && \text{(A.135)}\\
&\leq \sqrt{\frac{((1-\beta)^2+\beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}} \text{tr}\left(C^{(0)}\right)} + \frac{r'}{r}\text{dist}\left(2Pg(\kappa^{(l)})P^T, 0\right) && \text{(A.136)}\\
&\leq \sqrt{\frac{((1-\beta)^2+\beta^2\sigma_w^2)\alpha^2}{\delta^{l+1}} \text{tr}\left(C^{(0)}\right)} + \frac{r'}{r}\text{dist}\left(P\kappa^{(l)}P^T, 0\right), && \text{(A.137)}
\end{aligned}$$

so $P\kappa^{(l)}P^T \to 0$.

Combining the results, we conclude that $\kappa^{(l)} \to cvv^T$. □

A.7 Proof of Theorem 6

Proof of Theorem 6, part 1. Note that, in the proof of Theorem 4, part 1, the result

$$\rho_{\min}(ACA^T) \geq \rho_{\min}(C), \tag{A.138}$$

A is not required to be a $N \times N$ square matrix, but rather any non-negative matrix of shape $M \times N$ and the result remains the same. Consider

$$\mathcal{A} = \begin{pmatrix} \sigma_{w_1} I & \sigma_{w_2} A \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C^{(l)} & C^{(l)} \\ C^{(l)} & C^{(l)} \end{pmatrix}, \tag{A.139}$$

then in the recursion (3.14),

$$\rho_{\min}(K^{(l+1)}) = \rho_{\min}(\mathcal{A}\mathcal{C}\mathcal{A}^T) \geq \rho_{\min}(\mathcal{C}) = \rho_{\min}(C^{(l)}), \tag{A.140}$$

which is same as Theorem 4, part 1. The rest is the same. □

Proof of Theorem 6, part 2. Consider mapping $h : K^{(l)} \mapsto K^{(l+1)}$ at each layer:

$$K^{(l+1)} = \sigma_{w_1}^2 g(K^{(l)}) + \sigma_{w_2}^2 A g(K^{(l)}) A^T := h(K^{(l)}). \tag{A.141}$$

Note that

$$\begin{aligned}
&\operatorname{dist}\left(h(K^{(l)}), h(\tilde{K}^{(l)})\right) && \text{(A.142)}\\
&= \operatorname{dist}\left(\sigma_{w_1}^2 g(K^{(l)}) + \sigma_{w_2}^2 A g(K^{(l)}) A^T, \sigma_{w_1}^2 g(\tilde{K}^{(l)}) + \sigma_{w_2}^2 A g(\tilde{K}^{(l)}) A^T\right) && \text{(A.143)}\\
&\leq \sqrt{\operatorname{dist}\left(\sigma_{w_1}^2 g(K^{(l)}), \sigma_{w_1}^2 g(\tilde{K}^{(l)})\right)^2 + \operatorname{dist}\left(\sigma_{w_2}^2 A g(K^{(l)}) A^T, \sigma_{w_2}^2 A g(\tilde{K}^{(l)}) A^T\right)^2} && \text{(A.144)}\\
&\leq \sqrt{\sigma_{w_1}^2 + \sigma_{w_2}^2 r^2} \cdot \operatorname{dist}\left(g(K^{(l)}), g(\tilde{K}^{(l)})\right) && \text{(A.145)}\\
&\leq \sqrt{\frac{1}{2}(\sigma_{w_1}^2 + \sigma_{w_2}^2 r^2)} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right) && \text{(A.146)}\\
&= \delta^{1/2} \cdot \operatorname{dist}\left(K^{(l)}, \tilde{K}^{(l)}\right), && \text{(A.147)}
\end{aligned}$$

so h is a contraction mapping, and has a unique fixed point K. Let $\tilde{K}^{(l)} = K$, we have

$$\text{dist}\left(K^{(l+1)}, K\right) \leq \delta^{1/2} \cdot \text{dist}\left(K^{(l)}, K\right), \tag{A.148}$$

so $K^{(l)}$ converges to K linearly with a rate of convergence of $\delta^{1/2}$. □

Proof of Theorem 6, part 3. We have recursive relationship between $\kappa^{(l)}$ and $\kappa^{(l+1)}$:

$$\kappa^{(l+1)} = \frac{1}{\delta^{l+1}}(\sigma_{w_1}^2 g(K^{(l)}) + \sigma_{w_2}^2 A g(K^{(l)}) A^T) = \frac{\sigma_{w_1}^2}{\delta} g(\kappa^{(l)}) + \frac{\sigma_{w_2}^2}{\delta} A g(\kappa^{(l)}) A^T. \tag{A.149}$$

Using Lemma 3 for the covariance mapping g,

$$\begin{aligned}
\text{dist}\left(\kappa^{(l+1)}, 0\right)^2 &\leq \text{dist}\left(\frac{\sigma_{w_1}^2}{\delta} g(\kappa^{(l)}), 0\right)^2 + \text{dist}\left(\frac{\sigma_{w_2}^2}{\delta} A g(\kappa^{(l)}) A^T, 0\right)^2 && \text{(A.150)}\\
&\leq \frac{\sigma_{w_1}^2}{2\delta} \text{dist}\left(2g(\kappa^{(l)}), 0\right)^2 + \frac{\sigma_{w_2}^2 r^2}{2\delta} \text{dist}\left(2g(\kappa^{(l)}), 0\right)^2 && \text{(A.151)}\\
&= \text{dist}\left(2g(\kappa^{(l)}), 0\right)^2 && \text{(A.152)}\\
&\leq \text{dist}\left(\kappa^{(l)}, 0\right)^2, && \text{(A.153)}
\end{aligned}$$

so $\kappa^{(l)}$ is bounded.

Using the Perron-Frobenius theorem, we know $v \succ 0$. Then

$$\begin{aligned}
v^T \kappa^{(l+1)} v &= \frac{\sigma_{w_1}^2}{\delta} v^T g(\kappa^{(l)}) v + \frac{\sigma_{w_2}^2}{\delta} v^T A g(\kappa^{(l)}) A^T v && \text{(A.154)}\\
&= 2 v^T g(\kappa^{(l)}) v && \text{(A.155)}\\
&\geq v^T \kappa^{(l)} v, && \text{(A.156)}
\end{aligned}$$

so $v^T \kappa^{(l)} v$ has a limit $c > 0$.

Also from the Perron-Frobenius theorem, r is a simple eigenvalue, and there is no other

eigenvalues with absolute value $\geq r$. Let $P = I - \text{Proj}_v = I - vv^T$, then

$$PA = AP, \quad \|PA\|_2 = r' < r. \tag{A.157}$$

Same as Theorem 4, part 3, we have

$$\text{dist}\left(Pg(\kappa^{(l)})P^T, 0\right) \leq \text{dist}\left(\frac{1}{2}P\kappa^{(l)}P^T, 0\right). \tag{A.158}$$

Using Lemma 3 for the covariance mapping g,

$$\begin{aligned}
&\text{dist}\left(P\kappa^{(l+1)}P^T, 0\right)^2 && \text{(A.159)}\\
&\leq \text{dist}\left(\frac{\sigma_{w_1}^2}{\delta}Pg(\kappa^{(l)})P^T, 0\right)^2 + \text{dist}\left(\frac{\sigma_{w_2}^2}{\delta}PAg(\kappa^{(l)})A^TP^T, 0\right)^2 && \text{(A.160)}\\
&= \frac{\sigma_{w_1}^2}{2\delta}\text{dist}\left(2Pg(\kappa^{(l)})P^T, 0\right)^2 + \frac{\sigma_{w_2}^2}{2\delta}\text{dist}\left(2PAPg(\kappa^{(l)})P^TA^TP^T, 0\right)^2 && \text{(A.161)}\\
&\leq \frac{\sigma_{w_1}^2}{2\delta}\text{dist}\left(2Pg(\kappa^{(l)})P^T, 0\right)^2 + \frac{\sigma_{w_2}^2 r'^2}{2\delta}\text{dist}\left(2Pg(\kappa^{(l)})P^T, 0\right)^2 && \text{(A.162)}\\
&= \frac{\sigma_{w_1}^2 + \sigma_{w_2}^2 r'^2}{\sigma_{w_1}^2 + \sigma_{w_2}^2 r^2}\text{dist}\left(2P\kappa^{(l)}P^T, 0\right)^2 && \text{(A.163)}\\
&\leq \frac{\sigma_{w_1}^2 + \sigma_{w_2}^2 r'^2}{\sigma_{w_1}^2 + \sigma_{w_2}^2 r^2}\text{dist}\left(P\kappa^{(l)}P^T, 0\right)^2, && \text{(A.164)}
\end{aligned}$$

so $P\kappa^{(l)}P^T \to 0$.

Combining the results, we conclude that $\kappa^{(l)} \to cvv^T$. □

A.8 Proof of Theorem 7

We use mathematical induction.

Start with $L = 1$, when there is no nonlinearity function and the input is only processed by

an affine transformation:

$$f(X^{(0)};\theta) = Z^{(1)} = \frac{\sigma_w}{\sqrt{d_0}} A X^{(0)} W^{(1)} + \sigma_b 1_{N\times 1} b^{(1)}. \tag{A.165}$$

Since the weights and biases are initialized i.i.d., all the output nodes of this network $Z_i^{(1)}(x)$ are also i.i.d. Given different inputs, the i-th network outputs $Z_i^{(1)}(x)$ follow a Gaussian Process $\mathcal{GP}(0, K^{(1)})$, according to Theorem 1.

Now, we have

$$\Theta^{(1)}(\theta) = \frac{\partial Z^{(1)}}{\partial W^{(1)}}\Big(\frac{\partial Z^{(1)}}{\partial W^{(1)}}\Big)^T + \frac{\partial Z^{(1)}}{\partial b^{(1)}}\Big(\frac{\partial Z^{(1)}}{\partial b^{(1)}}\Big)^T \tag{A.166}$$

$$= \frac{\sigma_w^2}{d_0} A X^{(0)} (X^{(0)})^T A^T + \sigma_b^2 1_{N\times 1} 1_{N\times 1}^T \tag{A.167}$$

$$= \sigma_w^2 A C^{(0)} A^T + \sigma_b^2 1_{N\times N} \tag{A.168}$$

$$= \Theta^{(1)}, \tag{A.169}$$

which has no dependency on θ.

Using induction, we first assume the claim is true for a l-layer network with $\tilde{P}$ parameters in total:

$$\tilde{\theta} = (W^{(1)}, \ldots, W^{(l)}, b^{(1)}, \ldots, b^{(l)}) \in \mathbb{R}^{\tilde{P}}. \tag{A.170}$$

We need to prove the claim is also true for $L = l+1$. The output function of this $(l+1)$-layer network is:

$$f(X^{(0)};\theta) = Z^{(l+1)} = \frac{\sigma_w}{\sqrt{d_l}} A \phi(Z^{(l)}) W^{(l+1)} + \sigma_b 1_{N\times 1} b^{(l+1)}. \tag{A.171}$$

Using Theorem 1 for $L = l$, we know all the output nodes of this network $Z_i^{(l)}(x)$ are also i.i.d. and the i-th network outputs $Z_i^{(l)}(x)$ follow a Gaussian distribution $\mathcal{GP}(0, K^{(l)})$. Note

that $K^{(l)}$ has no dependency on θ.

Also from our induction assumption for a l-layer network, the $Z^{(l)}$ has a NTK converging to a deterministic limit $\Theta^{(l)}$ when $d_1, \ldots, d_{l-1} \to \infty$:

$$\Theta^{(l)}(\tilde{\theta}) = \frac{\partial Z^{(l)}}{\partial \tilde{\theta}} \Big(\frac{\partial Z^{(l)}}{\partial \tilde{\theta}} \Big)^T \to \Theta^{(l)}. \tag{A.172}$$

Next let's check the case $L = l + 1$. Compared to a l-layer network, a $(l+1)$-layer network has additional weight matrix $W^{(l+1)}$ and bias $b^{(l+1)}$. Thus the total parameters $\theta = (W^{(l+1)}, b^{(l+1)}, \tilde{\theta})$. We can compute the derivative with respect to three different sets of parameters:

$$\Theta^{(l+1)}(\theta) = \underbrace{\frac{\partial Z^{(l+1)}}{\partial W^{(l+1)}} \Big(\frac{\partial Z^{(l+1)}}{\partial W^{(l+1)}} \Big)^T}_{(A)} + \underbrace{\frac{\partial Z^{(l+1)}}{\partial b^{(l+1)}} \Big(\frac{\partial Z^{(l+1)}}{\partial b^{(l+1)}} \Big)^T}_{(B)} + \underbrace{\sum_{p=1}^{\tilde{P}} \frac{\partial Z^{(l+1)}}{\partial \tilde{\theta}_p} \Big(\frac{\partial Z^{(l+1)}}{\partial \tilde{\theta}_p} \Big)^T}_{(C)}. \tag{A.173}$$

Term (A):

$$(A) = \frac{\sigma_w^2}{d_l} A \phi\big(Z^{(l)}\big) \phi\big(Z^{(l)}\big)^T A^T \tag{A.174}$$

$$= \frac{\sigma_w^2}{d_l} A \left(\sum_{i=1}^{d_l} \phi\big(Z_i^{(l)}\big) \phi\big(Z_i^{(l)}\big)^T \right) A^T \tag{A.175}$$

$$\to \sigma_w^2 A \left(\mathbb{E}_{z_i^{(l)} \sim \mathcal{N}(0, K^{(l)})} [\phi(z_i^{(l)}) \phi(z_i^{(l)})^T] \right) A^T \tag{A.176}$$

$$= \sigma_w^2 A C^{(l)} A^T. \tag{A.177}$$

Term (B):

$$(B) = \sigma_b^2 1_{N \times 1} 1_{N \times 1}^T = \sigma_b^2 1_{N \times N}. \tag{A.178}$$

Term (C):

$$\frac{\partial Z^{(l+1)}}{\partial \tilde{\theta}_p} = \frac{\sigma_w}{\sqrt{d_l}} A \frac{\partial \phi(Z^{(l)})}{\partial \tilde{\theta}_p} W^{(l+1)} = \frac{\sigma_w}{\sqrt{d_l}} A \Big(\dot{\phi}(Z^{(l)}) \odot \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big) W^{(l+1)}, \tag{A.179}$$

and

$$(C) = \frac{\sigma_w^2}{d_l} \sum_{p=1}^{\tilde{P}} A \Big(\dot{\phi}(Z^{(l)}) \odot \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big) W^{(l+1)} W^{(l+1)^T} \Big(\dot{\phi}(Z^{(l)}) \odot \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big)^T A^T \tag{A.180}$$

$$\to \frac{\sigma_w^2}{d_l} \sum_{p=1}^{\tilde{P}} A \Big(\dot{\phi}(Z^{(l)}) \odot \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big) \Big(\dot{\phi}(Z^{(l)}) \odot \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big)^T A^T \tag{A.181}$$

$$= \frac{\sigma_w^2}{d_l} \sum_{p=1}^{\tilde{P}} A \left(\Big(\dot{\phi}(Z^{(l)}) \dot{\phi}(Z^{(l)})^T \Big) \odot \left(\frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big(\frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big)^T \right) \right) A^T \tag{A.182}$$

$$= \frac{\sigma_w^2}{d_l} A \left(\Big(\dot{\phi}(Z^{(l)}) \dot{\phi}(Z^{(l)})^T \Big) \odot \left(\sum_{p=1}^{\tilde{P}} \frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big(\frac{\partial Z^{(l)}}{\partial \tilde{\theta}_p} \Big)^T \right) \right) A^T \tag{A.183}$$

$$= \frac{\sigma_w^2}{d_l} A \Big(\Big(\dot{\phi}(Z^{(l)}) \dot{\phi}(Z^{(l)})^T \Big) \odot \Theta^{(l)} \Big) A^T \tag{A.184}$$

$$= \frac{\sigma_w^2}{d_l} A \left(\left(\sum_{i=1}^{d_l} \dot{\phi}(Z_i^{(l)}) \dot{\phi}(Z_i^{(l)})^T \right) \odot \Theta^{(l)} \right) A^T \tag{A.185}$$

$$\to \sigma_w^2 A \left(\Big(\mathbb{E}_{z_i^{(l)} \sim \mathcal{N}(0, K^{(l)})} [\dot{\phi}(z_i^{(l)}) \dot{\phi}(z_i^{(l)})^T] \Big) \odot \Theta^{(l)} \right) A^T \tag{A.186}$$

$$= \sigma_w^2 A \Big(\dot{C}^{(l)} \odot \Theta^{(l)} \Big) A^T. \tag{A.187}$$

Altogether, the NTK for this $(l+1)$-layer network

$$\Theta^{(l+1)}(\theta) \to \sigma_w^2 A C^{(l)} A^T + \sigma_b^2 1_{N \times N} + \sigma_w^2 A (\dot{C}^{(l)} \odot \Theta^{(l)}) A^T = \Theta^{(l+1)}. \tag{A.188}$$

APPENDIX B

EXPERIMENT DETAILS

B.1 Datasets

The predefined training/validation/test splits for datasets Cora / Citeseer / PubMed / Reddit / WikiCS come from the PyTorch Geometric library [Fey and Lenssen, 2019]. The splits for dataset ArXiv comes from the Open Graph Benchmark [Hu et al., 2020b]. The splits of Chameleon / Squirrel come from Geom-GCN [Pei et al., 2020], in accordance with the PyTorch Geometric library. The split for Crocodile is not available, so we conduct a random split with the same 0.48/0.32/0.20 proportion as that used for Chameleon and Squirrel [Rozemberczki et al., 2021]. The splits for Amazon and Coauthor dataset come from GCA [Zhu et al., 2021].

For each graph, we treat the edges as undirected and construct a binary, symmetric adjacent matrix $\underline{A}$. Then, we apply the normalization proposed by Kipf and Welling [2017] to define A used in (2.1). Specifically, $A = (I_N + D)^{-1/2}(I + \underline{A})(I_N + D)^{-1/2}$, where $D = \text{diag}\{\sum_j \underline{A}_{ij}\}$. For GraphSAGE and GGP, we instead use the row-normalized version $A = (I_N + D)^{-1}(I_N + \underline{A})$. No feature preprocessing is done except running the GPs on ArXiv, for which we apply a centering on the input features.

B.2 Environment

All experiments are conducted on a Nvidia Quadro GV100 GPU with 32GB of HBM2 memory. The code is written in Python 3.10.4 as distributed with Ubuntu 22.04 LTS. We use PyTorch 1.11.0 and PyTorch Geometric 2.1.0 with CUDA 11.3.

For all datasets except Reddit, we train GCN using the Adam optimizer in full batch for 100 epochs. Reddit is too large for GPU computation and hence we conduct mini-batch training (with a batch size 10240) for 10 epochs.

B.3 Hyperparameters

Table B.1: Hyperparameters For Table 2.3 and 4.3.

classification	GCNGP	$L = 2, \sigma_b = 0.0, \sigma_w = 1.0$.
	GCN	hidden dimension: 256, dropout: 0.5, lr: 0.01.
regression	GCNGP	$L = 2, \sigma_b = \sqrt{0.1}, \sigma_w = 1.0$.
	GCN	hidden dimension: 256, dropout: 0.5, lr: $\sqrt{0.1}$.

Table B.2: Hyperparameters for Table 3.2.

GCN	$L = 2$.
GCNII	$L = 2, \alpha = 0.1, \lambda = 0.5$ and $\beta_l = \log(\frac{\lambda}{l} + 1)$.
GIN	$L = 2, \epsilon = 0.0$.
GraphSAGE	$L = 2, \sigma_{w_1} = \sqrt{0.1}$ (Reddit) and $\sigma_{w_1} = 0.0$ (others), $\sigma_{w_2} = 1.0$.

B.4 Kernels

We choose $C^{(0)}$ to be the inner product kernel for GCNGPs and GNNGPs. The exception is when working with the low-rank versions, we apply PCA on the input features to ensure the number of features is smaller than the number of landmark nodes.

For low-rank kernels (suffixed with -X), the landmark points are chosen to be the training set for small datasets, while for large datasets (ArXiv and Reddit) the landmark points are a random 1/50 of the training set.

In posterior inference, the nugget ϵ is chosen using a grid search over $[10^{-3}, 10^{1}]$.

For the RBF kernel,

$$K(x, x') = \exp(-\gamma \|x - x'\|_2^2), \tag{B.1}$$

where γ is chosen using a grid search over $[10^{-2}, 10^{2}]$.

For the GGP kernel,

$$K = AK_0A^T, \quad K_0(x, x') = (x^T x' + c)^d, \tag{B.2}$$

where $c = 5.0, d = 3.0$ following the default of GGP [Ng et al., 2018]. Note that we use only the kernel but not the robust-max likelihood nor the ELBO training proposed by [Ng et al., 2018], because their code written in Python 2 is outdated and Pytorch officially dropping Python 2 support.

B.5 FLOPs computation

We summarize the parameter and FLOPs counts in a full-batch forward pass as follows. The counts include all layers of the GCN graph encoder of our DGI model. We use a factor of 2 for the multiply accumulate cost. For large models, the parameter and FLOP contribution of bilinear discriminator is negligible.

Table B.3: Parameter and FLOPs Counts

Neural network	Parameters	FLOPs
Graph convolution	0	$2d_{\text{input}}n_{\text{edge}}$
Weight term	$d_{\text{output}}d_{\text{input}}$	$2d_{\text{output}}d_{\text{input}}n_{\text{node}}$
Bias term	d_{output}	$d_{\text{output}}n_{\text{node}}$
Total (approximate)	$d_{\text{output}}d_{\text{input}}$	$2d_{\text{output}}d_{\text{input}}n_{\text{node}}$

The total amount of FLOPs used during forward step can be estimated as $C = 2ND$, where $N = d_{\text{output}}d_{\text{input}}$ is the total parameter count of each layer and $D = n_{\text{node}}$ for full-batch training. The backwards step is approximately twice the compute as the forwards step, so the overall compute can be estimated as $C = 6ND$ as [Kaplan et al., 2020].

APPENDIX C

CODE

Code for [Niu et al., 2023] is available at `https://github.com/niuzehao/gnn-gp`.

REFERENCES

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

Yehonatan Avidan, Qianyi Li, and Haim Sompolinsky. Connecting NTK and NNGP: A Unified Theoretical Framework for Neural Network Learning Dynamics in the Kernel Regime, 2023.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining Neural Scaling Laws, 2021.

Jie Chen and Michael L. Stein. Linear-cost covariance functions for Gaussian random fields. *Journal of the American Statistical Association*, 2021.

Jie Chen, Haim Avron, and Vikas Sindhwani. Hierarchically compositional kernels for scalable nonparametric learning. *Journal of Machine Learning Research*, 18(66):1–42, 2017.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *The International Conference on Machine Learning*, 2020.

Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In *The Neural Information Processing Systems*, 2009.

Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs. *Distill*, 2021. doi:10.23915/distill.00032. https://distill.pub/2021/understanding-gnns.

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *The International Conference on Learning Representations*, 2018.

Petros Drineas and Michael W. Mahoney. On the nystrom method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6(72): 2153–2175, 2005.

Simon S. Du, Kangcheng Hou, Barnabás Póczos, Ruslan Salakhutdinov, Ruosong Wang, and Keyulu Xu. Graph Neural Tangent Kernel: Fusing Graph Neural Networks with Graph Kernels, 2019.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *The International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow Gaussian processes. In *The International Conference on Learning Representations*, 2019.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *The International Conference on Learning Representations*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *The International Conference on Machine Learning*, 2017.

Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. A visual exploration of gaussian processes. *Distill*, 2019. doi:10.23915/distill.00017. https://distill.pub/2019/visual-exploration-gaussian-processes.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *The Neural Information Processing Systems*, 2017.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models, 2022.

Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: NNGP and NTK for deep attention networks, 2020.

Jilin Hu, Jianbing Shen, Bin Yang, and Ling Shao. Infinitely wide graph convolutional networks: Semi-supervised learning via Gaussian processes. Preprint arXiv:2002.12168, 2020a.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *The Neural Information Processing Systems*, 2020b.

Wei Huang, Yayong Li, Weitao Du, Richard Xu, Jie Yin, Ling Chen, and Miao Zhang. Towards Deepening Graph Neural Networks: A GNTK-based Optimization Perspective. In *International Conference on Learning Representations*, 2022.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, 2020.

Matthias Katzfuss. A multi-resolution approximation for massive spatial datasets. *Journal of the American Statistical Association*, 112(517):201–214, 2017.

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

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *The International Conference on Learning Representations*, 2017.

Prasanth Kolachina, Nicola Cancedda, Marc Dymetman, and Sriram Venkatapathy. Prediction of learning curves in machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22–30, 2012.

Sanjukta Krishnagopal and Luana Ruiz. Graph Neural Tangent Kernel: Convergence on Large Graphs, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *The Neural Information Processing Systems*, 2012.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. In *The International Conference on Learning Representations*, 2018.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *The Neural Information Processing Systems*, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *The Association for the Advancement of Artificial Intelligence*, 2018.

Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. Graph Self-Supervised Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2022. doi:10.1109/TKDE.2022.3172903.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Radford M. Neal. Priors for infinite networks. Technical Report CRG-TR-94-1, University of Toronto, 1994.

Yin Cheng Ng, Nicolo Colombo, and Ricardo Silva. Bayesian semi-supervised learning with graph Gaussian processes. In *The Neural Information Processing Systems*, 2018.

Zehao Niu, Mihai Anitescu, and Jie Chen. Graph neural network-inspired kernels for Gaussian processes in semi-supervised learning. In *The International Conference on Learning Representations*, 2023.

Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *The International Conference on Learning Representations*, 2019.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in Python. In *The International Conference on Learning Representations*, 2020.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric graph convolutional networks. In *The International Conference on Learning Representations*, 2020.

Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, pages 259–270, 2020.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *The Neural Information Processing Systems*, 2007.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2), 2021.

Benjamin Sanchez, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. doi:10.23915/distill.00033. https://distill.pub/2021/gnn-intro.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization, 2020.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *The International Conference on Learning Representations*, 2022.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *The International Conference on Learning Representations*, 2018.

Christopher Williams. Computing with infinite networks. In *The Neural Information Processing Systems*, 1996.

Andrew Gordon Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *The International Conference on Machine Learning*, 2015.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *The International Conference on Machine Learning*, 2022.

Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, and Stan Z. Li. Self-supervised Learning on Graphs: Contrastive, Generative,or Predictive, 2021a.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021b.

Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-Supervised Learning of Graph Neural Networks: A Unified Review, 2022.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *The International Conference on Machine Learning*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *The International Conference on Learning Representations*, 2019.

Greg Yang. Tensor programs I: Wide feedforward or recurrent neural networks of any architecture are Gaussian processes. In *The Neural Information Processing Systems*, 2019.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations, 2021.

Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34:249–270, 2022.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report TR1530, University of Wisconsin-Madison, 2008.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the web conference 2021*, pages 2069–2080, 2021.