Graph Neural Tangent Kernel:
Fusing Graph Neural Networks with Graph Kernels

Simon S. Du[*]   Kangcheng Hou[†]   Barnabás Póczos[‡]   Ruslan Salakhutdinov[§]

Ruosong Wang[¶]   Keyulu Xu[||]

Abstract

While graph kernels (GKs) are easy to train and enjoy provable theoretical guarantees, their practical performances are limited by their expressive power, as the kernel function often depends on hand-crafted combinatorial features of graphs. Compared to graph kernels, graph neural networks (GNNs) usually achieve better practical performance, as GNNs use multi-layer architectures and non-linear activation functions to extract high-order information of graphs as features. However, due to the large number of hyper-parameters and the non-convex nature of the training procedure, GNNs are harder to train. Theoretical guarantees of GNNs are also not well-understood. Furthermore, the expressive power of GNNs scales with the number of parameters, and thus it is hard to exploit the full power of GNNs when computing resources are limited. The current paper presents a new class of graph kernels, Graph Neural Tangent Kernels (GNTKs), which correspond to *infinitely wide* multi-layer GNNs trained by gradient descent. GNTKs enjoy the full expressive power of GNNs and inherit advantages of GKs. Theoretically, we show GNTKs provably learn a class of smooth functions on graphs. Empirically, we test GNTKs on graph classification datasets and show they achieve strong performance.

1 Introduction

Learning on graph-structured data such as social networks and biological networks requires one to design methods that effectively exploit the structure of graphs. Graph Kernels (GKs) and Graph Neural Networks (GNNs) are two major classes of methods for learning on graph-structured data. GKs, explicitly or implicitly, build feature vectors based on combinatorial properties of input graphs. Popular choices of GKs include Weisfeiler-Lehman subtree kernel [Shervashidze et al., 2011], graphlet kernel [Shervashidze et al., 2009] and random walk kernel [Vishwanathan et al., 2010, Gärtner et al., 2003]. GKs inherit all benefits of kernel methods. GKs are easy to train, since the corresponding optimization problem is convex. Moreover, the kernel function often has explicit expressions, and thus we can analyze their theoretical guarantees using tools in learning theory. The downside of GKs, however, is that hand-crafted features may not be powerful enough to capture high-order information that involves complex interaction between nodes, which could lead to worse practical performance.

---

[*]Carnegie Mellon University. Email: ssdu@cs.cmu.edu

[†]Zhejiang University. Email: kangchenghou@gmail.com

[‡]Carnegie Mellon University. Email: bapoczos@cs.cmu.edu

[§]Carnegie Mellon University. Email:rsalakhu@cs.cmu.edu.

[¶]Carnegie Mellon University. Email: ruosongw@andrew.cmu.edu

[||]Massachusetts Institute of Technology. Email: keyulu@mit.edu.

GNNs, on the other hand, do not require explicitly hand-crafted feature maps. Similar to convolutional neural networks (CNNs) which are widely applied in computer vision, GNNs use multi-layer structures and convolutional operations to aggregate local information of nodes, together with non-linear activation functions to extract features from graphs. Various architectures have been proposed [Xu et al., 2019, 2018]. GNNs extract higher-order information of graphs, which lead to more powerful features compared to hand-crafted combinatorial features used by GKs. As a result, GNNs have achieved state-of-the-art performance on a large number of tasks on graph-structured data. Nevertheless, there are also disadvantages of using GNNs. The objective function of GNNs is highly non-convex, and thus it requires careful hyper-parameter tuning to stabilize the training procedure. Meanwhile, due to the non-convex nature of the training procedure, it is also hard to analyze the learned GNNs directly. For example, one may ask whether GNNs can provably learn certain class of functions. This question seems hard to answer given our limited theoretical understanding of GNNs. Another disadvantage of GNNs is that the expressive power of GNNs scales with the number of parameters. Thus, it is hard to learn a powerful GNN when computing resources are limited. Can we build a model that enjoys the best of both worlds, i.e., a model that extracts powerful features as GNNs and is easy to train and analyze like GKs?

In this paper, we give an affirmative answer to this question. Inspired by recent connections between kernel methods and over-parameterized neural networks [Arora et al., 2019b,a, Du et al., 2019, 2018, Jacot et al., 2018], we propose a class of new graph kernels, Graph Neural Tangent Kernels (GNTKs). GNTKs are equivalent to *infinitely wide* GNNs trained by gradient descent, where the word "tangent" corresponds to the training algorithm — gradient descent. While GNTKs are induced by infinitely wide GNNs, the prediction of GNTKs depends only on pairwise kernel values between graphs, for which we give an analytic formula to calculate efficiently. Therefore, GNTKs enjoy the full expressive power of GNNs, while inheriting benefits of GKs.

Our Contributions. First, inspired by recent connections between over-parameterized neural networks and kernel methods Jacot et al. [2018], Arora et al. [2019a], we present a general recipe which translates a GNN architecture to its corresponding GNTK. This recipe works for a wide range of GNNs, including graph convolutional neural network (GCN) [Kipf and Welling, 2016], graph isomorphism neural network (GIN) [Xu et al., 2019], and GNN with jumping knowledge [Xu et al., 2018]. Second, we conduct a theoretical analysis of GNTKs. Using the technique developed in Arora et al. [2019b], we show for a broad range of smooth functions over graphs, a certain GNTK can learn them with polynomial number of samples. To our knowledge, this is the first sample complexity analysis in the GK and GNN literature. Finally, we validate the performance of GNTKs on 7 standard benchmark graph classification datasets. On four of them, we find GNTK outperforms all baseline methods and achieves state-of-the-art performance. In particular, GNKs achieve 83.6% accuracy on COLLAB dataset and 67.9% accuracy on PTC dataset, compared to the best of baselines, 81.0% and 64.6% respectively.

This paper is organized as follow. In Section 2, we provide necessary background and review operations in GNNs that we will use to derive GNTKs. In Section 3, we present our general recipe that translates a GNN to its corresponding GNTK. In Section 4, we give our theoretical analysis of GNTKs. In Section 5, we compare GNTK with state-of-the-art methods on graph classification datasets. We defer technical proofs to the supplementary material.

2 Preliminaries

We begin by summarizing the most common models for learning with graphs and, along the way, introducing our notation. Let $G = (V, E)$ be a graph with node features $\boldsymbol{h}_v \in \mathbb{R}^d$ for each $v \in V$. We denote the neighborhood of node v by $\mathcal{N}(v)$. In this paper, we consider the graph classification task, where, given a set of graphs $\{G_1, ..., G_n\} \subseteq \mathcal{G}$ and their labels $\{y_1, ..., y_n\} \subseteq \mathcal{Y}$, our goal is to learn to predict labels of unseen graphs.

Graph Neural Network. GNN is a powerful framework for graph representation learning. Modern GNNs generally follow a neighborhood aggregation scheme Xu et al. [2019], Gilmer et al. [2017], Xu et al. [2018], where the representation $\boldsymbol{h}_v^{(\ell)}$ of each node v (in layer ℓ) is recursively updated by aggregating and transforming the representations of its neighbors. After iterations of aggregation, the representation of an entire graph is then obtained through pooling, e.g., by summing the representations of all nodes in the graph. Many GNNs, with different aggregation and graph readout functions, have been proposed under the neighborhood aggregation framework Xu et al. [2019], Scarselli et al. [2009], Li et al. [2016], Kearnes et al. [2016], Ying et al. [2018], Velickovic et al. [2018], Hamilton et al. [2017], Xu et al. [2018], Duvenaud et al. [2015], Kipf and Welling [2016], Defferrard et al. [2016], Santoro et al. [2018, 2017], Battaglia et al. [2016].

Next, we formalize the GNN framework. We refer to the neighbor aggregation process as a BLOCK operation, and to graph-level pooling to as a READOUT operation.

BLOCK Operation. A BLOCK operation aggregates features over a neighborhood $\mathcal{N}(u) \cup \{u\}$ via, e.g., summation, and transforms the aggregated features with non-linearity, e.g. multi-layer perceptron (MLP) or a fully-connected layer followed by ReLU. We denote the number of fully-connected layers in each BLOCK operation, i.e., the number of hidden layers of an MLP, by R.

When $R = 1$, the BLOCK operation can be formulated as

$$\text{BLOCK}^{(\ell)}(u) = \sqrt{\frac{c_\sigma}{m}} \cdot \sigma \left(\boldsymbol{W}_\ell \cdot c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v^{(\ell-1)} \right). \tag{1}$$

Here, $\boldsymbol{W}_\ell$ are learnable weights, initialized as Gaussian random variables. σ is an activation function like ReLU. m is the output dimension of $\boldsymbol{W}_\ell$. We set the scaling factor c_σ to 2, following the initialization scheme in He et al. [2015]. c_u is a scaling factor for neighbor aggregation. Different GNNs often have different choices for c_u. In Graph Convolution Network (GCN) [Kipf and Welling, 2016], $c_u = \frac{1}{|\mathcal{N}(u)|+1}$, and in Graph Isomorphism Network (GIN) [Xu et al., 2019], $c_u = 1$, which correspond to averaging and summing over neighbor features, respectively.

When the number of fully-connected layers $R = 2$, the BLOCK operation can be written as

$$\text{BLOCK}^{(\ell)}(u) = \sqrt{\frac{c_\sigma}{m}} \sigma \left(\boldsymbol{W}_{\ell,2} \sqrt{\frac{c_\sigma}{m}} \cdot \sigma \left(\boldsymbol{W}_{\ell,1} \cdot c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v^{(\ell-1)} \right) \right), \tag{2}$$

where $\boldsymbol{W}_{\ell,1}$ and $\boldsymbol{W}_{\ell,2}$ are learnable weights. BLOCK operations can be defined similarly for $R > 2$.

READOUT Operation. To get the representation of an entire graph $\boldsymbol{h}_G$ after L steps of aggregation, we take the summation over all node features, i.e.,

$$\boldsymbol{h}_G = \text{READOUT}\left(\left\{\boldsymbol{h}_u^{(L)}, u \in V\right\}\right) = \sum_{u \in V} \boldsymbol{h}_u^{(L)}.$$

There are more sophisticated READOUT operations than a simple summation Xu et al. [2018], Zhang et al. [2018a], Ying et al. [2018]. Jumping Knowledge Network (JK-Net) Xu et al. [2018] considers graph structures of different granularity, and aggregates graph features across all layers as

$$\boldsymbol{h}_G = \text{READOUT}^{\text{JK}}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in V, \ell \in [L]\right\}\right) = \sum_{u \in V}\left[\boldsymbol{h}_u^{(0)}; \ldots; \boldsymbol{h}_u^{(L)}\right]. \tag{3}$$

Building GNNs using BLOCK and READOUT. Most modern GNNs are constructed using the BLOCK operation and the READOUT operation Xu et al. [2019]. We denote the number of BLOCK operations (aggregation steps) in a GNN by L. For each $\ell \in [L]$ and $u \in V$, we define $\boldsymbol{h}_u^{(\ell)} = \text{BLOCK}^{(\ell)}(u)$. The graph-level feature is then $\boldsymbol{h}_G = \text{READOUT}\left(\left\{\boldsymbol{h}_u^{(L)}, u \in V\right\}\right)$ or $\boldsymbol{h}_G = \text{READOUT}^{\text{JK}}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in V, \ell \in [L]\right\}\right)$, depending on whether jumping knowledge is applied or not.

3 GNTK Formulas

In this section we present our general recipe which translates a GNN architecture to its corresponding GNTK. We first provide some intuitions on neural tangent kernels (NTKs). We refer readers to Jacot et al. [2018], Arora et al. [2019a] for more comprehensive descriptions.

3.1 Intuition of the Formulas

Consider a general neural network $f(\theta, x) \in \mathbb{R}$ where $\theta \in \mathbb{R}^m$ is all the parameters in the network and x is the input. Given a training dataset $\{(x_i, y_i)_{i=1}^n\}$, consider training the neural network by minimizing the squared loss over training data

$$\ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(\theta, x_i) - y_i)^2.$$

Suppose we minimize the squared loss $\ell(\theta)$ by gradient descent with infinitesimally small learning rate, i.e., $\frac{d\theta(t)}{dt} = -\nabla\ell(\theta(t))$. Let $u(t) = (f(\theta, x_i))_{i=1}^n$ be the network outputs. $u(t)$ follows the evolution

$$\frac{du}{dt} = -\boldsymbol{H}(t)(u(t) - y),$$

where

$$\boldsymbol{H}(t)_{ij} = \left\langle\frac{\partial f(\theta(t), x_i)}{\partial \theta}, \frac{\partial f(\theta(t), x_j)}{\partial \theta}\right\rangle \text{ for } (i, j) \in [n] \times [n].$$

Recent advances in optimization of neural networks have shown, for sufficiently over-parameterized neural networks, the matrix $\boldsymbol{H}(t)$ keeps almost unchanged during the training process Arora et al. [2019b,a], Du et al. [2019, 2018], Jacot et al. [2018], in which case the training dynamics is identical

4

to that of kernel regression. Moreover, under a random initialization of parameters, the random matrix $\boldsymbol{H}(0)$ converges in probability to a certain deterministic kernel matrix, which is called Neural Tangent Kernel (NTK) Jacot et al. [2018] and corresponds to infinitely wide neural networks. See Figure 4 in the supplementary material for an illustration.

Explicit formulas for NTKs of fully-connected neural networks have been given in Jacot et al. [2018]. Recently, explicit formulas for NTKs of convolutional neural networks are given in Arora et al. [2019a]. The goal of this section is to give an explicit formula for NTKs that correspond to GNNs defined in Section 2. Our general strategy is inspired by Arora et al. [2019a]. Let $f(\theta, G) \in \mathbb{R}$ be the output of the corresponding GNN under parameters θ and input graph G, for two given graphs G and G', to calculate the corresponding GNTK value, we need to calculate the expected value of

$$\left\langle \frac{\partial f(\theta, G)}{\partial \theta}, \frac{\partial f(\theta, G')}{\partial \theta} \right\rangle$$

in the limit that $m \to \infty$ and θ are all Gaussian random variables, which can be viewed as a Gaussian process. For each layer in the GNN, we use $\boldsymbol{\Sigma}$ to denote the covariance matrix of outputs of that layer, and $\dot{\boldsymbol{\Sigma}}$ to denote the covariance matrix corresponds to the derivative of that layer. Due to the multi-layer structure of GNNs, these covariance matrices can be naturally calculated via dynamic programming.

3.2 Formulas for Calculating GNTKs

Given two graphs $G = (V, E), G' = (V', E')$ with $|V| = n$, $|V'| = n'$ and a GNN with L BLOCK operations and R fully-connected layers with ReLU activation in each BLOCK operation. We give the GNTK formula of pairwise kernel value $\Theta(G, G') \in \mathbb{R}$ induced by this GNN.

We first define the covariance matrix between input features of two input graphs G, G', which we use $\boldsymbol{\Sigma}^{(0)}(G, G') \in \mathbb{R}^{n \times n'}$ to denote. For two nodes $u \in V$ and $u' \in V'$, $\left[\boldsymbol{\Sigma}^{(0)}(G, G')\right]_{uu'}$ is defined to be $\boldsymbol{h}_u^\top \boldsymbol{h}_{u'}$, where $\boldsymbol{h}_u$ and $\boldsymbol{h}_{u'}$ are the input features of $u \in V$ and $u' \in V'$.

BLOCK Operation. A BLOCK operation in GNTK calculates a covariance matrix $\boldsymbol{\Sigma}_{(R)}^{(\ell)}(G, G') \in \mathbb{R}^{n \times n'}$ using $\boldsymbol{\Sigma}_{(R)}^{(\ell-1)}(G, G') \in \mathbb{R}^{n \times n'}$, and calculates intermediate kernel values $\boldsymbol{\Theta}_{(r)}^{(\ell)}(G, G') \in \mathbb{R}^{n \times n'}$, which will be later used to compute the final output.

More specifically, we first perform a neighborhood aggregation operation

$$\left[\boldsymbol{\Sigma}_{(0)}^{(\ell)}(G, G')\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \left[\boldsymbol{\Sigma}_{(R)}^{(\ell-1)}(G, G')\right]_{vv'},$$

$$\left[\boldsymbol{\Theta}_{(0)}^{(\ell)}(G, G')\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \left[\boldsymbol{\Theta}_{(R)}^{(\ell-1)}(G, G')\right]_{vv'}.$$

Here we define $\boldsymbol{\Sigma}_{(R)}^{(0)}(G, G')$ and $\boldsymbol{\Theta}_{(R)}^{(0)}(G, G')$ as $\boldsymbol{\Sigma}^{(0)}(G, G')$, for notational convenience. Next we perform R transformations that correspond to the R fully-connected layers with ReLU activation. Here $\sigma(z) = \max\{0, z\}$ is the ReLU activation function. We denote $\dot{\sigma}(z) = \mathbb{1}[z \geq 0]$ to be the derivative of the ReLU activation function.

For each $r \in [R]$, we define

- For $u \in V, u' \in V'$,

$$\left[\boldsymbol{A}_{(r)}^{(\ell)}\left(G, G'\right) \right]_{uu'} = \begin{pmatrix} \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G, G)\right]_{u,u} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G, G')\right]_{uu'} \\ \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G', G)\right]_{uu'} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G', G')\right]_{u'u'} \end{pmatrix} \in \mathbb{R}^{2 \times 2}.$$

- For $u \in V, u' \in V'$,

$$\left[\boldsymbol{\Sigma}_{(r)}^{(\ell)}(G, G')\right]_{uu'} = c_\sigma \mathbb{E}_{(a,b) \sim \mathcal{N}\left(\boldsymbol{0}, \left[\boldsymbol{A}_{(r)}^{(\ell)}(G,G')\right]_{uu'}\right)} \left[\sigma\left(a\right)\sigma\left(b\right)\right],$$

$$\left[\dot{\boldsymbol{\Sigma}}_{(r)}^{(\ell)}\left(G, G'\right)\right]_{uu'} = c_\sigma \mathbb{E}_{(a,b) \sim \mathcal{N}\left(\boldsymbol{0}, \left[\boldsymbol{A}_{(r)}^{(\ell)}(G,G')\right]_{uu'}\right)} \left[\dot{\sigma}(a)\dot{\sigma}(b)\right].$$

- For $u \in V, u' \in V'$,

$$\left[\boldsymbol{\Theta}_{(r)}^{(\ell)}(G, G')\right]_{uu'} = \left[\boldsymbol{\Theta}_{(r-1)}^{(\ell)}(G, G')\right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(r)}^{(\ell)}\left(G, G'\right)\right]_{uu'} + \left[\boldsymbol{\Sigma}_{(r)}^{(\ell)}\left(G, G'\right)\right]_{uu'}.$$

Note in the above we have shown how to calculate $\boldsymbol{\Theta}_{(R)}^{(\ell)}(G, G')$ for each $\ell \in \{0, 1, \ldots, L\}$. These intermediate outputs will be used to calculate the final output of the corresponding GNTK.

READOUT Operation. Given these intermediate outputs, we can now calculate the final output of GNTK using the following formula.

$$\Theta(G, G') = \begin{cases} \sum_{u \in V, u' \in V'} \left[\boldsymbol{\Theta}_{(R)}^{(\ell)}\left(G, G'\right)\right]_{uu'} & \text{without jumping knowledge} \\ \sum_{u \in V, u' \in V'} \left[\sum_{\ell=0}^{L} \boldsymbol{\Theta}_{(R)}^{(\ell)}(G, G')\right]_{uu'} & \text{with jumping knowledge} \end{cases}.$$

To better illustrate our general recipe, in Figure 1 we give a concrete example in which we translate a GNN with $L = 2$ BLOCK operations, $R = 1$ fully-connection layer in each BLOCK operation, and jumping knowledge, to its corresponding GNTK.

4 Theoretical Analysis of GNTK

In this section, we analyze the generalization ability of a GNTK that corresponds to a simple GNN. We consider the standard supervised learning setup. We are given n training data $\{(G_i, y_i)\}_{i=1}^n$ drawn i.i.d. from the underlying distribution $\mathcal{D}$, where G_i is the i-th input graph and y_i is its label. Consider a GNN with a single BLOCK operation, followed by the READOUT operation (without jumping knowledge). Here we set $c_u = \left(\left\|\sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v\right\|_2\right)^{-1}$. We use $\boldsymbol{\Theta} \in \mathbb{R}^{n \times n}$ to denote the kernel matrix, where $\boldsymbol{\Theta}_{ij} = \Theta(G_i, G_j)$. Here $\Theta(G, G')$ is the kernel function that corresponds to the simple GNN. See Section 3 for the formulas for calculating $\Theta(G, G')$. Throughout the discussion, we assume that the kernel matrix $\boldsymbol{\Theta} \in \mathbb{R}^{n \times n}$ is invertible.

For a testing point G_{te}, the prediction of kernel regression using GNTK on this testing point is

$$f_{ker}(G_{te}) = [\Theta(G_{te}, G_1), \Theta(G_{te}, G_1), \ldots, \Theta(G_{te}, G_n)]^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}.$$

The following result is a standard result for kernel regression proved using Rademacher complexity. For a proof, see Bartlett and Mendelson [2002].
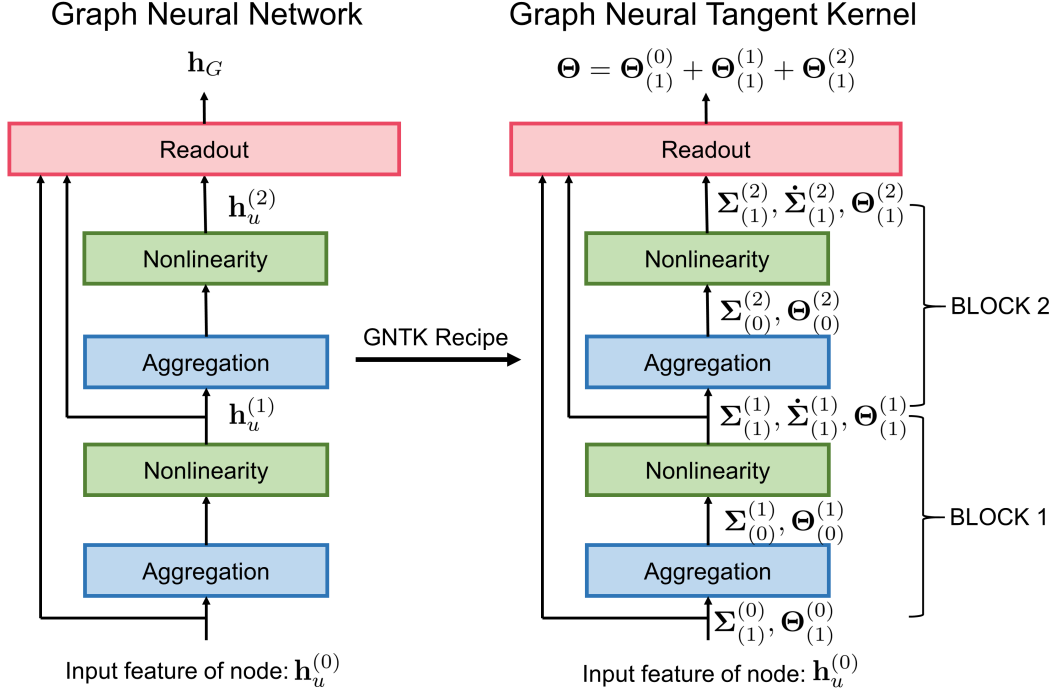
Figure 1: **Illustration of our recipe that translates a GNN to a GNTK.** For a GNN with $L = 2$ BLOCK operations, $R = 1$ fully-connected layer in each BLOCK operation, and jumping knowledge, the corresponding GNTK is calculated as follow. For two graphs G and G', we first calculate $\left[\boldsymbol{\Theta}^{(0)}_{(1)}(G, G')\right]_{uu'} = \left[\boldsymbol{\Sigma}^{(0)}_{(1)}(G, G')\right]_{uu'} = [\boldsymbol{\Sigma}^{(0)}(G, G')]_{uu'} = \boldsymbol{h}_u^\top \boldsymbol{h}_{u'}$. We follow the kernel formulas in Section 3 to calculate $\boldsymbol{\Sigma}^{(\ell)}_{(0)}, \boldsymbol{\Theta}^{(\ell)}_{(0)}$ using $\boldsymbol{\Sigma}^{(\ell-1)}_{(R)}, \boldsymbol{\Theta}^{(\ell-1)}_{(R)}$ (Aggregation) and calculate $\boldsymbol{\Sigma}^{(\ell)}_{(r)}, \dot{\boldsymbol{\Sigma}}^{(\ell)}_{(r)}, \boldsymbol{\Theta}^{(\ell)}_{(r)}$ using $\boldsymbol{\Sigma}^{(\ell)}_{(r-1)}, \boldsymbol{\Theta}^{(\ell)}_{(r-1)}$ (Nonlinearity). The final output is $\Theta(G, G') = \sum_{u \in V, u' \in V'} \left[\sum_{\ell=0}^{L} \boldsymbol{\Theta}^{(\ell)}_{(R)}(G, G')\right]_{uu'}$.

7

Theorem 4.1 (Bartlett and Mendelson [2002]). *Given n training data $\{(G_i, y_i)\}_{i=1}^n$ drawn i.i.d. from the underlying distribution $\mathcal{D}$. Consider any loss function $\ell : \mathbb{R} \times \mathbb{R} \to [0, 1]$ that is 1-Lipschitz in the first argument such that $\ell(y, y) = 0$. With probability at least $1 - \delta$, the population loss of the GNTK predictor can be upper bounded by*

$$L_{\mathcal{D}}(f_{ker}) = \mathbb{E}_{(G,y)\sim\mathcal{D}} [\ell(f_{ker}(G), y)] \leq O\left(\frac{\sqrt{\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y} \cdot \operatorname{tr}(\boldsymbol{\Theta})}}{n} + \sqrt{\frac{\log(1/\delta)}{n}}\right).$$

Note that this theorem presents a data-dependent generalization bound which is related to the kernel matrix $\boldsymbol{\Theta} \in \mathbb{R}^{n \times n}$ and the labels $\{y_i\}_{i=1}^n$. Using this theorem, if we can bound $\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}$ and $\operatorname{tr}(\boldsymbol{\Theta})$, then we can obtain a concrete sample complexity bound. We instantiate this idea to study the class of graph labeling functions that can be efficiently learned by GNTKs.

The following two theorems guarantee that if labels are generated as described in (4), then the GNTK that corresponds to the simple GNN described above can learn this function with polynomial number of samples. We first give an upper bound on $\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}$.

Theorem 4.2. *For each $i \in [n]$, if the labels $\{y_i\}_{i=1}^n$ satisfy*

$$y_i = \alpha_1 \sum_{u \in V} \left(\overline{\boldsymbol{h}}_u^\top \boldsymbol{\beta}_1\right) + \sum_{l=1}^\infty \alpha_{2l} \sum_{u \in V} \left(\overline{\boldsymbol{h}}_u^\top \boldsymbol{\beta}_{2l}\right)^{2l}, \tag{4}$$

where $\overline{\boldsymbol{h}}_u = c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v$, $\alpha_1, \alpha_2, \alpha_4, \ldots \in \mathbb{R}$, $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_4, \ldots \in \mathbb{R}^d$, and $G_i = (V, E)$, then we have

$$\sqrt{\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}} \leq 2|\alpha_1| \cdot \|\boldsymbol{\beta}_1\|_2 + \sum_{l=1}^\infty \sqrt{2\pi}(2l-1)|\alpha_{2l}| \cdot \|\boldsymbol{\beta}_{2l}\|_2^{2l}.$$

The following theorem gives an upper bound on $\operatorname{tr}(\boldsymbol{\Theta})$.

Theorem 4.3. *If for all graphs $G_i = (V_i, E_i)$ in the training set, $|V_i|$ is upper bounded by $\overline{V}$, then $\operatorname{tr}(\boldsymbol{\Theta}) \leq O(n\overline{V}^2)$. Here n is the number of training samples.*

Combining Theorem 4.2 and Theorem 4.3 with Theorem 4.1, we know if

$$2|\alpha_1| \cdot \|\boldsymbol{\beta}_1\|_2 + \sum_{l=1}^\infty \sqrt{2\pi}(2l-1)|\alpha_{2l}| \cdot \|\boldsymbol{\beta}_{2l}\|_2^{2l}$$

is bounded, and $|V_i|$ is bounded for all graphs $G_i = (V_i, E_i)$ in the training set, then the GNTK that corresponds to the simple GNN described above can learn functions of forms in (4), with polynomial number of samples. To our knowledge, this is the first sample complexity analysis in the GK and GNN literature.

5 Experiments

In this section, we demonstrate the effectiveness of GNTKs using experiments on graph classification tasks. For ablation study, we investigate how the performance varies with the architecture of the corresponding GNN. Following common practices of evaluating performance of graph classification models Yanardag and Vishwanathan [2015], we perform 10-fold cross validation and report the mean and standard deviation of validation accuracies. More details about the experiment setup can be found in Section B of the supplementary material.

	Method	COLLAB	IMDB-B	IMDB-M	PTC	NCI1	MUTAG	PROTEINS
GNN	GCN	79.0 ± 1.8	74.0 ± 3.4	51.9 ± 3.8	64.2 ± 4.3	80.2 ± 2.0	85.6 ± 5.8	76.0 ± 3.2
	GraphSAGE	–	72.3 ± 5.3	50.9 ± 2.2	63.9 ± 7.7	77.7 ± 1.5	85.1 ± 7.6	75.9 ± 3.2
	PatchySAN	72.6 ± 2.2	71.0 ± 2.2	45.2 ± 2.8	60.0 ± 4.8	78.6 ± 1.9	$\mathbf{92.6 \pm 4.2}$	75.9 ± 2.8
	DGCNN	73.7	70.0	47.8	58.6	74.4	85.8	75.5
	GIN	80.2 ± 1.9	75.1 ± 5.1	52.3 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	89.4 ± 5.6	$\mathbf{76.2 \pm 2.8}$
GK	WL subtree	78.9 ± 1.9	73.8 ± 3.9	50.9 ± 3.8	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$	90.4 ± 5.7	75.0 ± 3.1
	AWL	73.9 ± 1.9	74.5 ± 5.9	51.5 ± 3.6	–	–	87.9 ± 9.8	–
	RetGK	81.0 ± 0.3	71.9 ± 1.0	47.7 ± 0.3	62.5 ± 1.6	84.5 ± 0.2	90.3 ± 1.1	75.8 ± 0.6
	GNTK	$\mathbf{83.6 \pm 1.0}$	$\mathbf{76.9 \pm 3.6}$	$\mathbf{52.8 \pm 4.6}$	$\mathbf{67.9 \pm 6.9}$	84.2 ± 1.5	90.0 ± 8.5	75.6 ± 4.2

Table 1: **Classification results (in %) for graph classification datasets.** GNN: graph neural network based methods. GK: graph kernel based methods. GNTK: fusion of GNN and GK.

Datasets. The benchmark datasets include four bioinformatics datasets MUTAG, PTC, NCI1, PROTEINS and three social network datasets COLLAB, IMDB-BINARY, IMDB-MULTI. For each graph, we transform the categorical input features to one-hot encoding representations. For datasets where the graphs have no node features, i.e. only graph structure matters, we use degrees as input node features.

5.1 Results

We compare GNTK with various state-of-the-art graph classification algorithms: (1) the WL subtree kernel Shervashidze et al. [2011]; (2) state-of-the-art deep learning architectures, including Graph Convolutional Network (GCN) Kipf and Welling [2016], GraphSAGE Hamilton et al. [2017], Graph Isomorphism Network(GIN) Xu et al. [2019], PATCHY-SANNiepert et al. [2016] and Deep Graph CNN (DGCNN) Zhang et al. [2018a]; (3) Graph kernels based on random walks, i.e., Anonymous Walk Embeddings Ivanov and Burnaev [2018] and RetGK Zhang et al. [2018b]. For deep learning methods and random walk graph kernels, we report the accuracies reported in the original papers. The experiment setup is deferred to Section B.

The graph classification results are shown in Table 1. The best results are highlighted as bold. Our proposed GNTKs are powerful and achieve state-of-the-art classification accuracy on most datasets. In four of them, we find GNTKs outperform all baseline methods. In particular, GNTKs achieve 83.6% accuracy on COLLAB dataset and 67.9% accuracy on PTC dataset, compared to the best of baselines, 81.0% and 64.6% respectively. Notably, GNTKs give the best performance on all social network datasets.

5.2 Relation between GNTK Performance and the Corresponding GNN

We conduct ablation study to investigate how the performance of GNTK varies as we change the architecture of the corresponding GNN. We select two representative datasets, one social network dataset IMDBBINARY, and another bioinformatics dataset NCI1. For IMDBBINARY, we vary the number of BLOCK operations in $\{2, 3, 4, 5, 6\}$. For NCI1, we vary the number of BLOCK operations in $\{8, 10, 12, 14, 16\}$. For both datasets, we vary the number of MLP layers in $\{1, 2, 3\}$.
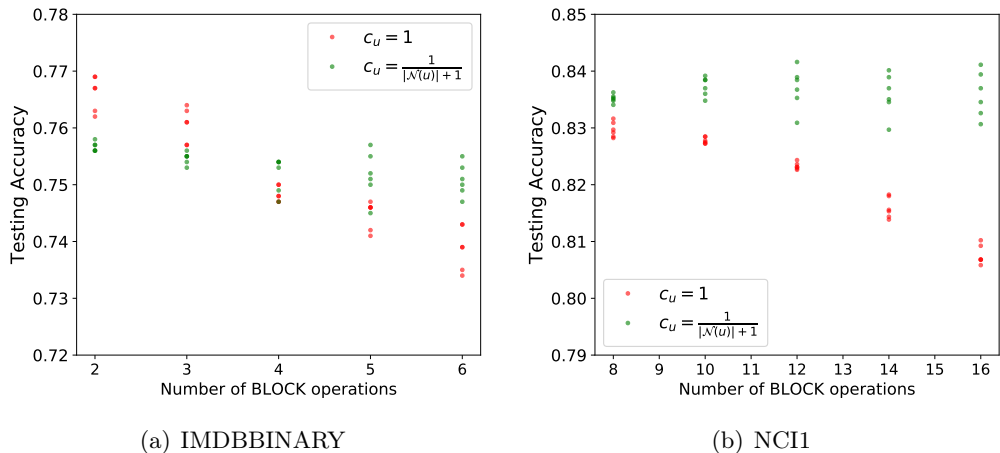
(a) IMDBBINARY	(b) NCI1

Figure 2: **Effects of number of BLOCK operations and the scaling factor c_u on the performance of GNTK.** Each dot represents the performance of a particular GNTK architecture. We divide different architectures into different groups by number of BLOCK operations. We color these GNTK architecture points by the scaling factor c_u. We observe the test accuracy is correlated with the dataset and the architecture.

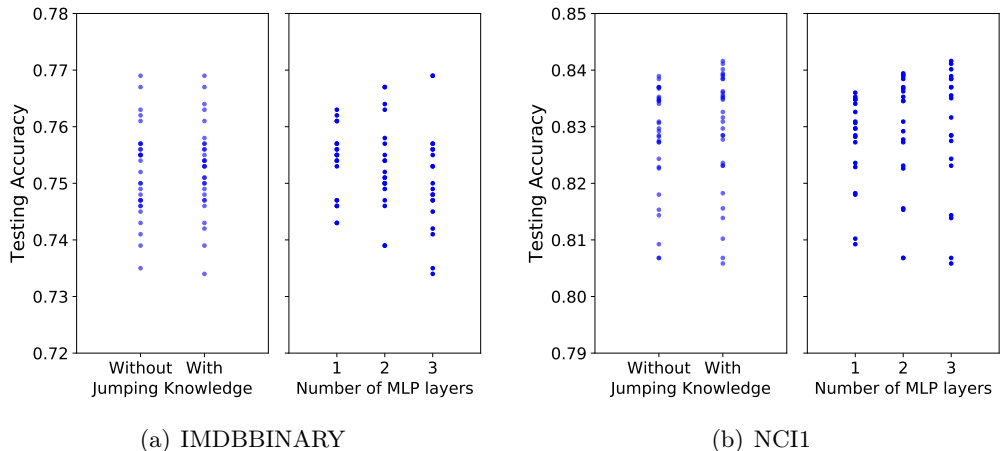|	
(a) IMDBBINARY	(b) NCI1

Figure 3: **Effects of jumping knowledge and number of MLP layers on the performance of GNTK.** Each dot represents the test performance of a GNTK architecture. We divide all GNTK architectures into different groups, according to whether jumping knowledge is applied, or number of MLP layers.

Effects of Number of BLOCK Operations and the Scaling Factor c_u. We investigate how the performance of GNTKs is correlated with number of BLOCK operations and the scaling factor c_u. First, on the bioinformatics dataset (NCI), we observe that GNTKs with more layers perform better. This is perhaps because, for molecules and bio graphs, more global structural information is helpful, as they provide important information about the chemical/bio entity. On such graphs, GNTKs are particularly effective because GNTKs can easily scale to many layers, whereas the number of layers in GNNs may be restricted by computing resources.

Moreover, the performance of GNTK is correlated with that of the corresponding GNN. For

10

example, in social networks, GNTKs with sum aggregation $c_u = 1$ work better than average aggregation $c_u = \frac{1}{|\mathcal{N}(u)|+1}$. The similar pattern holds in GNNs, because sum aggregation learns more graph structure information than average aggregation Xu et al. [2019]. This suggests GNTK can indeed inherit the properties and advantages of the corresponding GNN, while also gaining the benefits of graph kernels.

Effects of Jumping Knowledge and Number of MLP Layers In the GNN literature, jumping knowledge network is expected to improve performance Xu et al. [2018]. In Figure 3, we observe that a similar trend holds for GNTK. The performance of GNTK is improved on both NCI and IMDB datasets when jumping knowledge is applied. Moreover, increasing the number of MLP layers can increase the performance by $\sim 0.8\%$. These empirical findings further confirm that GNTKs can inherit the benefits of GNNs, since improvements on GNN architectures are reflected in the improvements GNTKs.

Thus, we conclude that GNTKs are attractive for graph representation learning because they can combine the advantages of both GNNs and GKs.

6   Acknowledgment

References

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019a.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019b.

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

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1273–1272, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Sergey Ivanov and Evgeniy Burnaev. Anonymous walk embeddings. In *ICML*, 2018.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8): 595–608, 2016.

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

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

Adam Santoro, Felix Hill, David Barrett, Ari Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, pages 4477–4486, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

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

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5449–5458, 2018.

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. RetGK: Graph kernels based on return probabilities of random walks. In *NeurIPS*, 2018b.
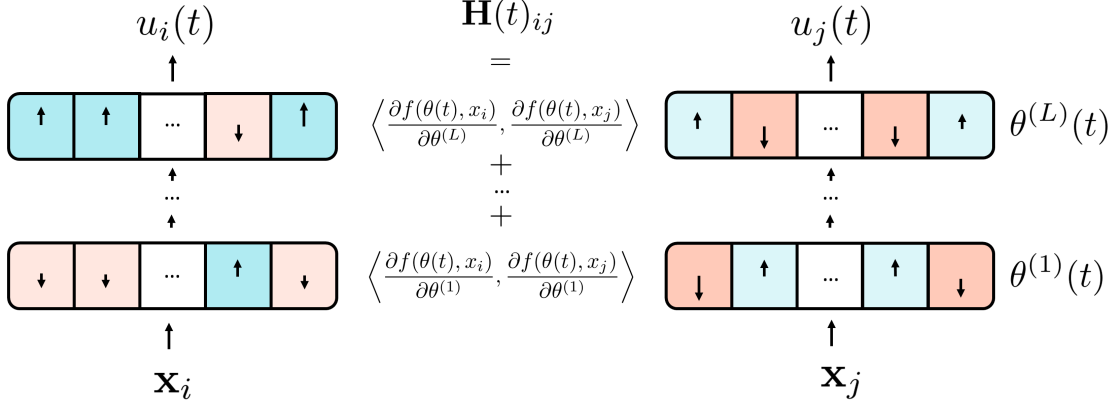
Figure 4: Illustration of NTK theory. Consider a general neural network with L layers $\theta^{(1)}, \ldots, \theta^{(L)}$, given input x_i and x_j, the neural network will output $f(\theta(t), x_i), f(\theta(t), x_j)$. When trained by gradient descent, evolution of $u(t)$ follows $\frac{du}{dt} = -\boldsymbol{H}(t)(u(t) - y)$. One can show that when the number of parameters in the neural network is large enough, and parameters of the neural network are initialized as Gaussian variables, $\boldsymbol{H}(t) \approx \boldsymbol{H}(0)$ and can be calculated analytically.

A Missing Proofs

A.1 Proof of Theorem 4.2

Proof. By Section 3, for two graph G and G', the GNTK kernel function that corresponds to the simple GNN can be described as

$$\Theta(G, G') = \sum_{u \in V, u' \in V'} \left(\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(1)}^{(1)}(G, G') \right]_{uu'} + \left[\boldsymbol{\Sigma}_{(1)}^{(1)}(G, G') \right]_{uu'} \right).$$

Here, we have

$$\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} = c_u c_{u'} \left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v \right)^{\top} \left(\sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \boldsymbol{h}_{v'} \right) = \overline{\boldsymbol{h}}_u^{\top} \overline{\boldsymbol{h}}_{u'}.$$

Recall that

$$\left[\boldsymbol{\Sigma}_{(r)}^{(\ell)}(G, G') \right]_{uu'} = c_\sigma \mathbb{E}_{(a,b) \sim \mathcal{N}\left(\mathbf{0}, \left[\boldsymbol{A}_{(r)}^{(\ell)}(G, G') \right]_{uu'} \right)} \left[\sigma(a) \sigma(b) \right],$$

$$\left[\dot{\boldsymbol{\Sigma}}_{(r)}^{(\ell)}(G, G') \right]_{uu'} = c_\sigma \mathbb{E}_{(a,b) \sim \mathcal{N}\left(\mathbf{0}, \left[\boldsymbol{A}_{(r)}^{(\ell)}(G, G') \right]_{uu'} \right)} \left[\dot{\sigma}(a) \dot{\sigma}(b) \right]$$

and

$$\left[\boldsymbol{A}_{(r)}^{(\ell)}(G, G') \right]_{uu'} = \begin{pmatrix} \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G, G) \right]_{u,u} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G, G') \right]_{uu'} \\ \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G', G) \right]_{uu'} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G', G') \right]_{u'u'} \end{pmatrix} \in \mathbb{R}^{2 \times 2}.$$

14

Since $\sigma(z) = \max\{0, z\}$ is the ReLU activation function, and $\dot{\sigma}(z) = \mathbb{1}[z \geq 0]$ is the derivative of the ReLU activation function, and $\|\overline{\boldsymbol{h}}_u\|_2 = 1$ for all nodes u, by calculation, we have

$$\left[\dot{\boldsymbol{\Sigma}}_{(1)}^{(1)}(G, G')\right]_{uu'} = \frac{\pi - \arccos\left(\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'}\right)}{2\pi},$$

$$\left[\boldsymbol{\Sigma}_{(1)}^{(1)}(G, G')\right]_{uu'} = \frac{\pi - \arccos\left(\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'}\right) + \sqrt{1 - \left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'}^2}}{2\pi}.$$

Since

$$\arcsin(x) = \sum_{l=0}^{\infty} \frac{(2l-1)!!}{(2l)!!} \cdot \frac{x^{2l+1}}{2l+1},$$

we have

$$\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(1)}^{(1)}(G, G')\right]_{uu'} = \frac{1}{4}\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} + \frac{1}{2\pi}\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} \arcsin\left(\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'}\right)$$

$$= \frac{1}{4}\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} + \frac{1}{2\pi}\sum_{l=1}^{\infty} \frac{(2l-3)!!}{(2l-2)!! \cdot (2l-1)} \cdot \left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'}^{2l}$$

$$= \frac{1}{4}\overline{\boldsymbol{h}}_u^\top \overline{\boldsymbol{h}}_{u'} + \frac{1}{2\pi}\sum_{l=1}^{\infty} \frac{(2l-3)!!}{(2l-2)!! \cdot (2l-1)} \cdot \left(\overline{\boldsymbol{h}}_u^\top \overline{\boldsymbol{h}}_{u'}\right)^{2l}.$$

Let $\boldsymbol{\Phi}^{(2l)}(\cdot)$ be the feature map of the polynomial kernel of degree $2l$, i.e.,

$$k^{(2l)}(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{x}^\top \boldsymbol{y}\right)^{2l} = \boldsymbol{\Phi}^{(2l)}(\boldsymbol{x})^\top \boldsymbol{\Phi}^{(2l)}(\boldsymbol{y}).$$

We have

$$\left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(1)}^{(1)}(G, G')\right]_{uu'}$$

$$= \frac{1}{4}\overline{\boldsymbol{h}}_u^\top \overline{\boldsymbol{h}}_{u'} + \frac{1}{2\pi}\sum_{l=1}^{\infty} \frac{(2l-3)!!}{(2l-2)!! \cdot (2l-1)} \cdot \left(\boldsymbol{\Phi}^{(2l)}(\overline{\boldsymbol{h}}_u)\right)^\top \boldsymbol{\Phi}^{(2l)}(\overline{\boldsymbol{h}}_{u'}).$$

Let

$$\Theta_1(G, G') = \sum_{u \in V, u' \in V'} \left[\boldsymbol{\Sigma}_{(0)}^{(1)}(G, G')\right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(1)}^{(1)}(G, G')\right]_{uu'},$$

we have

$$\Theta_1(G, G') = \frac{1}{4}\left(\sum_{u \in V} \overline{\boldsymbol{h}}_u\right)^\top \left(\sum_{u' \in V'} \overline{\boldsymbol{h}}_{u'}\right) + \frac{1}{2\pi}\sum_{l=1}^{\infty} \frac{(2l-3)!!}{(2l-2)!! \cdot (2l-1)} \cdot \left(\sum_{u \in V} \boldsymbol{\Phi}^{(2l)}(\overline{\boldsymbol{h}}_u)\right)^\top \left(\sum_{u' \in V'} \boldsymbol{\Phi}^{(2l)}(\overline{\boldsymbol{h}}_{u'})\right).$$

Since $\boldsymbol{\Theta} = \boldsymbol{\Theta}_1 + \boldsymbol{\Theta}_2$ where $\boldsymbol{\Theta}_2$ is a kernel matrix (and thus positive semi-definite), for any $y \in \mathbb{R}^n$, we have

$$\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y} \leq \boldsymbol{y}^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}.$$

Recall that

$$y_i = \alpha_1 \sum_{u \in V} \left(\overline{\boldsymbol{h}}_u^\top \boldsymbol{\beta}_1 \right) + \sum_{l=1}^\infty \alpha_{2l} \sum_{u \in V} \left(\overline{\boldsymbol{h}}_u^\top \boldsymbol{\beta}_{2l} \right)^{2l}.$$

We rewrite

$$y_i = y_i^{(0)} + \sum_{l=1}^\infty y_i^{(2l)},$$

where

$$y_i^{(0)} = \alpha_1 \left(\sum_{u \in V} \overline{\boldsymbol{h}}_u \right)^\top \boldsymbol{\beta}_1,$$

and for each $l \geq 1$,

$$y_i^{(2l)} = \alpha_{2l} \sum_{u \in V} \left(\overline{\boldsymbol{h}}_u^\top \boldsymbol{\beta}_{2l} \right)^{2l} = \alpha_{2l} \sum_{u \in V} \left(\boldsymbol{\Phi}^{2l} \left(\overline{\boldsymbol{h}}_u \right) \right)^\top \boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right) = \alpha_{2l} \left(\sum_{u \in V} \boldsymbol{\Phi}^{2l} \left(\overline{\boldsymbol{h}}_u \right) \right)^\top \boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right).$$

We have

$$\boldsymbol{y} = \boldsymbol{y}^{(0)} + \sum_{l=1}^\infty \boldsymbol{y}^{(2l)}.$$

Thus,

$$\sqrt{\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}} \leq \sqrt{\boldsymbol{y}^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}} \leq \sqrt{\left(\boldsymbol{y}^{(0)} \right)^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}^{(0)}} + \sum_{l=1}^\infty \sqrt{\left(\boldsymbol{y}^{(2l)} \right)^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}^{(2l)}}.$$

When $l = 0$, we have

$$\sqrt{\left(\boldsymbol{y}^0 \right)^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}^0} \leq 2 |\alpha_1| \|\boldsymbol{\beta}_1\|_2.$$

When $l \geq 1$, we have

$$\sqrt{\left(\boldsymbol{y}^{2l} \right)^\top \boldsymbol{\Theta}_1^{-1} \boldsymbol{y}^{2l}} \leq \sqrt{2\pi} (2l - 1) |\alpha_{2l}| \left\| \boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right) \right\|_2.$$

Notice that

$$\left\| \boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right) \right\|_2^2 = \left(\boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right) \right)^\top \boldsymbol{\Phi}^{2l} \left(\boldsymbol{\beta}_{2l} \right) = \|\boldsymbol{\beta}_{2l}\|_2^{4l}.$$

Thus,

$$\sqrt{\boldsymbol{y}^\top \boldsymbol{\Theta}^{-1} \boldsymbol{y}} \leq 2 |\alpha_1| \|\boldsymbol{\beta}_1\|_2 + \sum_{l=1}^\infty \sqrt{2\pi} (2l - 1) |\alpha_{2l}| \|\boldsymbol{\beta}_{2l}\|_2^{2l}.$$

$\square$

A.2 Proof of Theorem 4.3

Proof. Recall that

$$\Theta(G, G') = \sum_{u \in V, u' \in V'} \left(\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \left[\dot{\mathbf{\Sigma}}_{(1)}^{(1)}(G, G') \right]_{uu'} + \left[\mathbf{\Sigma}_{(1)}^{(1)}(G, G') \right]_{uu'} \right),$$

where

$$\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} = c_u c_{u'} \left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v \right)^{\top} \left(\sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \boldsymbol{h}_{v'} \right) = \overline{\boldsymbol{h}}_u^{\top} \overline{\boldsymbol{h}}_{u'}$$

and

$$\left[\dot{\mathbf{\Sigma}}_{(1)}^{(1)}(G, G') \right]_{uu'} = \frac{\pi - \arccos \left(\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \right)}{2\pi},$$

$$\left[\mathbf{\Sigma}_{(1)}^{(1)}(G, G') \right]_{uu'} = \frac{\pi - \arccos \left(\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \right) + \sqrt{1 - \left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'}^2}}{2\pi}.$$

Since for each node u, $\overline{\boldsymbol{h}}_u = c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v$, and $c_u = \left(\left\| \sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{h}_v \right\|_2 \right)^{-1}$, we have $\|\overline{\boldsymbol{h}}_u\|_2 = 1$. Moreover,

$$\left[\dot{\mathbf{\Sigma}}_{(1)}^{(1)}(G, G') \right]_{uu'} = \frac{\pi - \arccos \left(\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \right)}{2\pi} \leq 1/2$$

and

$$\left[\mathbf{\Sigma}_{(1)}^{(1)}(G, G') \right]_{uu'} = \frac{\pi - \arccos \left(\left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'} \right) + \sqrt{1 - \left[\mathbf{\Sigma}_{(0)}^{(1)}(G, G') \right]_{uu'}^2}}{2\pi} \leq \frac{1 + \pi}{2\pi} \leq 1,$$

we have

$$\Theta(G, G') \leq 2|V||V'|.$$

Thus,

$$\operatorname{tr}(\mathbf{\Theta}) \leq 2n\overline{V}^2.$$

$\square$

B Experiment Setup

For GNTKs, we tune the following hyperparameters.
1. The number of BLOCK operations. We search from candidate values $\{1, 2, 3, 4\}$.
2. The number of fully-connected layers in each BLOCK operation. We search from candidate values $\{1, 2, \ldots, 16\}$.
3. The parameter c_u. We search from candidate values $\left\{ 1, \frac{1}{|\mathcal{N}(u)|+1} \right\}$.

To utilize the GNTKs we compute, we test with kernel regression and C-SVM as the final classifier. The regularization parameter C in C-SVM is determined using grid search from 60 values evenly chosen from $[10^{-2}, 10^3]$, in log scale.