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

Simon S. Du
Institute for Advanced Study
ssdu@ias.edu

Kangcheng Hou
Zhejiang University
kangchenghou@gmail.com

Barnabás Póczos
Carnegie Mellon University
bapoczos@cs.cmu.edu

Ruslan Salakhutdinov
Carnegie Mellon University
rsalakhu@cs.cmu.edu

Ruosong Wang
Carnegie Mellon University.
ruosongw@andrew.cmu.edu

Keyulu Xu
Massachusetts Institute of Technology
keyulu@mit.edu

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, Karrer et al., 2015]. 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 than GNNs.

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., 2019a, 2019b]. 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, Du et al., 2019, 2018, Jacot et al., 2018, Yang, 2019], 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, Yang, 2019], 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 isomorphism network (GIN) [Xu et al., 2019a], graph convolutional network (GCN) [Kipf and Welling, 2016], 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. Moreover, in our experiments, we also observe that GNTK is more computationally efficient than its GNN counterpart.

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 $h_v \in \mathbb{R}^d$ for each $v \in V$. We denote the neighborhood of node $v$ by $N(v)$. 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., 2019a, Gilmer et al., 2017].
Xu et al. [2018], where the representation \( h_u^{(\ell)} \) of each node \( v \) (in layer \( \ell \)) 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. [2019a, 2018], Scarselli et al. [2009], Li et al. [2016], Kearnes et al. [2016], Ying et al. [2018], Velickovic et al. [2018], Hamilton et al. [2017], 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

\[
\text{BLOCK}^{(\ell)}(u) = \frac{c_\sigma}{m} \cdot \sigma \left( W_{\ell} \cdot c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} h_v^{(\ell-1)} \right),
\]

Here, \( W_{\ell} \) are learnable weights, initialized as Gaussian random variables, \( \sigma \) is an activation function like ReLU, \( m \) is the output dimension of \( W_{\ell} \). We set the scaling factor \( c_\sigma \) 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. 2019a], \( c_u = 1 \), 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

\[
\text{BLOCK}^{(\ell)}(u) = \frac{c_\sigma}{m} \left( W_{\ell,2} \frac{c_\sigma}{m} \cdot \sigma \left( W_{\ell,1} \cdot c_u \sum_{v \in \mathcal{N}(u) \cup \{u\}} h_v^{(\ell-1)} \right) \right),
\]

where \( W_{\ell,1} \) and \( W_{\ell,2} \) are learnable weights. Notice that here we first aggregate features over neighborhood \( \mathcal{N}(u) \cup \{u\} \) and then transforms the aggregated features with an MLP with \( R = 2 \) hidden layers. BLOCK operations can be defined similarly for \( R > 2 \). Notice that the BLOCK operation we defined above is also known as the graph (spatial) convolutional layer in the GNN literature.

**READOUT Operation.** To get the representation of an entire graph \( h_G \) after \( L \) steps of aggregation, we take the summation over all node features, i.e.,

\[
h_G = \text{READOUT} \left( \left\{ h_u^{(L)}, u \in V \right\} \right) = \sum_{u \in V} h_u^{(L)}.
\]

There are more sophisticated READOUT operations than a simple summation Xu et al. [2018], Zhang et al. [2018], 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

\[
h_G = \text{READOUT}^{\text{JK}} \left( \left\{ h_u^{(\ell), u \in V, \ell \in [L]} \right\} \right) = \sum_{u \in V} \left[ h_u^{(0)}; \ldots ; h_u^{(L)} \right].
\]

**Building GNNs using BLOCK and READOUT.** Most modern GNNs are constructed using the BLOCK operation and the READOUT operation Xu et al. [2019a]. 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 \( h_u^{(\ell)} = \text{BLOCK}^{(\ell)}(u) \). The graph-level feature is then \( h_G = \text{READOUT} \left( \left\{ h_u^{(L)}, u \in V \right\} \right) \) or \( h_G = \text{READOUT}^{\text{JK}} \left( \left\{ h_u^{(\ell)}, u \in V, \ell \in [L] \right\} \right) \), depending on whether jumping knowledge (JK) 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 \( u(t) = (f(\theta(t), x_i))_{i=1}^n \) be the network outputs. \( u(t) \) follows the evolution

\[
\frac{du}{dt} = -H(t)(u(t) - y),
\]

where

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

Recent advances in optimization of neural networks have shown, for sufficiently over-parameterized neural networks, the matrix \( H(t) \) keeps almost unchanged during the training process. Arora et al. [2019a], Ding et al. [2019, 2018], Jacot et al. [2018], in which case the training dynamics is identical to that of kernel regression. Moreover, under a random initialization of parameters, the random matrix \( H(0) \) 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 \( \theta \) 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 \( \theta \) are all Gaussian random variables, which can be viewed as a Gaussian process. For each layer in the GNN, we use \( \Sigma \) to denote the covariance matrix of outputs of that layer, and \( \Delta \Sigma \) 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 \( \Sigma^{(0)}(G, G') \in \mathbb{R}^{n \times n'} \) to denote. For two nodes \( u \in V \) and \( u' \in V' \), \( [\Sigma^{(0)}(G, G')]_{uu'} \) is defined to be \( h_u^\top h_{u'} \), where \( h_u \) and \( h_{u'} \) are the input features of \( u \in V \) and \( u' \in V' \).
For we give a concrete example in which we translate 1 derivative of the ReLU activation function. Here we define \( \Sigma \) and \( \Theta \), which will be later used to compute the final output.

More specifically, we first perform a neighborhood aggregation operation

\[
\left[ \Sigma^{(0)}(G, G') \right]_{uv'} = c_{uv} \sum_{v \in \mathcal{N}(u) \cup \{u\}, v' \in \mathcal{N}(v) \cup \{v\}} \left[ \Sigma^{(0)}(G, G') \right]_{uv'} ,
\]

\[
\left[ \Theta^{(0)}(G, G') \right]_{uv'} = c_{uv} \sum_{v \in \mathcal{N}(u) \cup \{u\}, v' \in \mathcal{N}(v) \cup \{v\}} \left[ \Theta^{(0)}(G, G') \right]_{uv'} .
\]

Here we define \( \Sigma^{(0)}(G, G') \) and \( \Theta^{(0)}(G, G') \) as \( \Sigma^{(0)}(G, G') \), 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 \( \sigma(z) = 1[z \geq 0] \) to be the derivative of the ReLU activation function.

For each \( r \in [R] \), we define

- For \( u \in V, u' \in V' \),
  
  \[
  \left[ \mathbf{A}^{(r)}(G, G') \right]_{uu'} = \left[ \begin{array}{c}
  \left[ \Sigma^{(r-1)}(G, G') \right]_{u,u'} \\
  \left[ \Sigma^{(r-1)}(G, G') \right]_{u,u'} \\
  \end{array} \right] \in \mathbb{R}^{2 \times 2}.
  \]

- For \( u \in V, u' \in V' \),
  
  \[
  \left[ \Sigma^{(r)}(G, G') \right]_{uu'} = c_{uv} \mathbb{E}_{(u,v) \sim \mathcal{N}} \left[ A^{(r)}(G, G') \right]_{uu'} ,
  \]

- For \( u \in V, u' \in V' \),
  
  \[
  \left[ \Theta^{(r)}(G, G') \right]_{uu'} = \left[ \begin{array}{c}
  \left[ \Sigma^{(r)}(G, G') \right]_{u,u'} \\
  \left[ \Sigma^{(r)}(G, G') \right]_{u,u'} \\
  \end{array} \right] + \left[ \Sigma^{(r)}(G, G') \right]_{uu'} .
  \]

Note in the above we have shown how to calculate \( \Theta^{(r)}(G, G') \) for each \( r \in [0, 1, \ldots, L] \). 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:

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

To better illustrate our general recipe, in Figure 11 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 

\sum_{v \in \mathcal{N}(u) \cup \{u\}} h_v \right\|_2 \right)^{-1} \). We use \( \Theta \in \mathbb{R}^{n \times n} \) to denote the kernel matrix, where \( \Theta_{ij} = \Theta(G_i, G_j) \). Here \( \Theta(G, G') \) is the kernel function that corresponds
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 $\Theta^{(0)}(0,1)_{uu'} = \Sigma^{(0)}(G,G')_{uu'} = h^\top u h_{u'}$. We follow the kernel formulas in Section 3 to calculate $\Sigma^{(\ell)}_{(R)}(0), \Theta^{(\ell)}_{(R)}(0)$ using $\Sigma^{(\ell-1)}_{(R)}, \Theta^{(\ell-1)}_{(R)}$ (Aggregation) and calculate $\Sigma^{(\ell)}_{(r)}, \Theta^{(\ell)}_{(r)}$ using $\Sigma^{(\ell-1)}_{(r)}, \Theta^{(\ell-1)}_{(r)}$ (Nonlinearity). The final output is $\Theta(G,G') = \sum_{u \in V, u' \in V'} \left[ \sum_{\ell=0}^{L} \Theta^{(\ell)}_{(R)}(G,G') \right]_{uu'}$.

The following result is a standard result for kernel regression proved using Rademacher complexity. For a proof, see Bartlett and Mendelson [2002].

**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 $D$. 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

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

Note that this theorem presents a data-dependent generalization bound which is related to the kernel matrix $\Theta \in \mathbb{R}^{n \times n}$ and the labels $\{y_i\}_{i=1}^n$. Using this theorem, if we can bound $y^\top \Theta^{-1} y$ and $\text{tr} (\Theta)$, 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 (3), 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 $y^\top \Theta^{-1} y$.
Theorem 4.2. For each \( i \in [n] \), if the labels \( \{y_i\}_{i=1}^n \) satisfy
\[
y_i = \alpha_1 \sum_{u \in V} \left( \tilde{h}_u \beta_1 \right) + \frac{\sum_{i=1}^{\infty} \alpha_{2l} \sum_{u \in V} \left( \tilde{h}_u \beta_{2l} \right)^{2l}}{\Theta} ,
\]
where \( \tilde{h}_u = c_u \sum_{v \in N(u) \cup \{u\}} h_v, \alpha_1, \alpha_2, \alpha_4, \ldots \in \mathbb{R}, \beta_1, \beta_2, \beta_4, \ldots \in \mathbb{R}^d \), and \( G_i = (V, E) \), then we have
\[
\sqrt{y^T \Theta^{-1} y} \leq 2|\alpha_1| \cdot ||\beta_1||_2 + \sum_{l=1}^{\infty} \sqrt{2\pi(2l - 1)} |\alpha_{2l}| \cdot ||\beta_{2l}||_2^{2l}.
\]

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

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

Combining Theorem 4.2 and Theorem 4.3 with Theorem 4.1, we know if
\[
2|\alpha_1| \cdot ||\beta_1||_2 + \sum_{l=1}^{\infty} \sqrt{2\pi(2l - 1)} |\alpha_{2l}| \cdot ||\beta_{2l}||_2^{2l}
\]
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 (3), 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 3 of the supplementary material.

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. [2019a], PATCHY-SAN Niepert 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 3.

The graph classification results are shown in Table 2. 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. Moreover, in our experiments, we also observe that with the same architecture, GNTK is more computational efficient that its GNN counterpart. On IMDB-B dataset, running GIN with the default setup (official implementation of Xu et al. [2019a]) takes 19 minutes on a TITAN X GPU and running GNTK only takes 2 minutes.
### Table 1: Classification results (in %) for graph classification datasets.

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

GNN: graph neural network based methods. GK: graph kernel based methods. GNTK: fusion of GNN and GK.

### 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.

#### 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 $\{1, 2, 3\}$.

**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 example, in social networks, GNTKs with sum aggregation $c_u = 1$ work better than average aggregation $c_u = \frac{1}{|N(u)|+1}$. The similar pattern holds in GNNs, because sum aggregation learns more graph structure information than average aggregation [Xu et al. 2019a]. This suggests GNTK can indeed inherit the properties and advantages of the corresponding GNN, while also gaining the benefits of graph kernels.
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 Jumping Knowledge and Number of MLP Layers** In the GNN literature, jumping knowledge network (JK) is expected to improve performance [Xu et al. 2018, Fey 2019]. 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.

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

**Acknowledgments**

S. S. Du and B. Póczos acknowledge support from AFRL grant FA8750-17-2-0212 and DARPA D17AP0000. R. Salakhutdinov and R. Wang are supported in part by NSF IIS-1763562, Office of Naval Research grant N000141812861, and Nvidia NVAIL award. K. Xu is supported by NSF CAREER award 1553284 and a Chevron-MIT Energy Fellowship. This work was performed while S. S. Du was a Ph.D. student at Carnegie Mellon University and K. Hou was visiting Carnegie Mellon University.
References


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.