Fast Graph Neural Tangent Kernel via Kronecker Sketching

Shunhua Jiang[1], **Yunze Man**[2], **Zhao Song**[3], **Zheng Yu**[4], **Danyang Zhuo**[5]
[1]Columbia University
[2]Carnegie Mellon University
[3]Adobe Research
[4]Princeton University
[5]Duke University
sj3005@columbia.edu, yman@cs.cmu.edu, zsong@adobe.com, zhengy@princeton.edu, danyang@cs.duke.edu

Abstract

Many deep learning tasks have to deal with graphs (e.g., protein structures, social networks, source code abstract syntax trees). Due to the importance of these tasks, people turned to Graph Neural Networks (GNNs) as the de facto method for learning on graphs. GNNs have become widely applied due to their convincing performance. Unfortunately, one major barrier to using GNNs is that GNNs require substantial time and resources to train. Recently, a new method for learning on graph data is *Graph Neural Tangent Kernel (GNTK)* (Du et al. 2019a). GNTK is an application of Neural Tangent Kernel (NTK) (Jacot, Gabriel, and Hongler 2018) (a kernel method) on graph data, and solving NTK regression is equivalent to using gradient descent to train an infinite-wide neural network. The key benefit of using GNTK is that, similar to any kernel method, GNTK's parameters can be solved directly in a single step. This can avoid time-consuming gradient descent. Meanwhile, sketching has become increasingly used in speeding up various optimization problems, including solving kernel regression. Given a kernel matrix of n graphs, using sketching in solving kernel regression can reduce the running time to $o(n^3)$. But unfortunately such methods usually requires extensive knowledge about the kernel matrix beforehand, while in the case of GNTK we find that the construction of the kernel matrix is already $O(n^2N^4)$, assuming each graph has N nodes. The kernel matrix construction time can be a major performance bottleneck when the size of graphs N increases. A natural question to ask is thus whether we can speed up the kernel matrix construction to improve GNTK regression's end-to-end running time. This paper provides the first algorithm to construct the kernel matrix in $o(n^2N^3)$ running time.

Introduction

Graph Neural Networks (GNNs) have quickly become a popular method for machine learning on graph data. GNNs have delivered ground-breaking results in many important areas of AI, including social networking (Yang et al. 2020a), bioinformatics (Zitnik and Leskovec 2017; Yue et al. 2020), recommendation systems (Ying et al. 2018), and autonomous driving (Weng et al. 2020a; Yang et al. 2020b). Given the importance of GNNs, how to train GNNs efficiently has become one of the most important problems in the AI community. However, efficient GNN training is challenging due to the relentless growth in the model complexity and dataset sizes, both in terms of the number of graphs in a dataset and the sizes of the graphs.

Recently, a new direction for fast GNN training is to use Graph Neural Tangent Kernel (GNTK). (Du et al. 2019a) has shown that GNTK can achieve similar accuracy as GNNs on many important learning tasks. At that same time, GNTK regression is significantly faster than iterative stochastic gradient descent optimization because solving the parameters in GNTK is just a single-step kernel regression process. Further, GNTK can scale with GNN model sizes because the regression running time grows only linearly with the complexity of GNN models.

Meanwhile, sketching has been increasingly used in optimization problems (Clarkson and Woodruff 2013; Nelson and Nguyên 2013; Meng and Mahoney 2013; Boutsidis and Woodruff 2014; Song, Woodruff, and Zhong 2017; Andoni et al. 2018; Jiang et al. 2021; Dong, Lee, and Ye 2021), including linear regression, kernel regression and linear programming. Various approaches have been proposed either to sketch down the dimension of kernel matrices then solve the low-dimensional problem, or to approximate the kernel matrices by randomly constructing low-dimensional feature vectors then solve using random feature vectors directly.

It is natural to think about whether we can use sketching to improve the running time of GNTK regression. Given n graphs where each graph has N nodes, the kernel matrix has a size of $n \times n$. Given the kernel matrix is already constructed, it is well-known that using sketching can reduce the regression time from $O(n^3)$ to $o(n^3)$ under certain conditions. However, in the context of GNTK, the construction of the kernel matrix need the running time of $O(n^2N^4)$ to begin with. This is because for each pair of graphs, we need to compute the Kronecker product of those two graphs. There are n^2 pairs in total and each pair requires N^4 time computation. The end-to-end GNTK regression time is sum of the time for kernel matrix construction time and the kernel regression time. This means, the kernel matrix construction can be a major barrier for adopting sketching to speed up the end-to-end GNTK regression time, especially when we need to use GNTK for large graphs (large N).

This raises an important question:

Can we speed up the kernel matrix construction time?

This question is worthwhile due to two reasons. First, cal-

culating the Kronecker product for each pair of graphs to construct the kernel matrix is a significant running time bottleneck for GNTK and is a major barrier for adopting sketching to speed up GNTK. Second, unlike kernel matrix construction for traditional neural networks in NTK, the construction of the kernel matrix is a complex iterative process, and it is unclear how to use sketching to reduce its complexity.

We accelerate the GNTK constructions in two steps. 1) Instead of calculating the time consuming Kronecker product for each pair of graphs directly, we consider the multiplication of a Kronecker product with a vector as a whole and accelerate it by decoupling it into two matrix multiplications of smaller dimensions. This allows us to achieve kernel matrix construction time of $O(n^2N^3)$; 2) Further, we propose a new iterative sketching procedure to reduce the running time of the matrix multiplications in the construction while providing the guarantee that the introduced error will not destruct the generalization ability of GNTK. This allows us to further improve the kernel matrix construction time to $o(n^2N^3)$.

To summarize, this paper has the following results and contributions:

- We identify that the kernel matrix construction is the major running time bottleneck for GNTK regression in large graph tasks.
- We improve the kernel construction time in two steps: 1) We decouple the Kronecker-vector product in GNTK construction into matrix multiplications; and 2) We design an iterative sketching algorithm to further reduce the matrix multiplication time in calculating GNTK while maintaining its generalization guarantee. Overall, we are able to reduce the running time of constructing GNTK from $O(n^2N^4)$ to $o(n^2N^3)$.
- We rigorously quantify the impact of the error resulted from proposed sketching and show under certain assumptions that we maintain the generalization ability of the original GNTK.

Related Work

Sketching Sketching has many applications in numerical linear algebra, such as linear regression, low-rank approximation (Clarkson and Woodruff 2013; Nelson and Nguyên 2013; Meng and Mahoney 2013; Boutsidis and Woodruff 2014; Razenshteyn, Song, and Woodruff 2016; Song, Woodruff, and Zhong 2017; Andoni et al. 2018), distributed problems (Woodruff and Zhong 2016; Boutsidis, Woodruff, and Zhong 2016), federated learning (Song, Yu, and Zhang 2021), reinforcement learning (Wang et al. 2020), tensor decomposition (Song, Woodruff, and Zhong 2019), polynomial kernels (Song et al. 2021), kronecker regression (Diao et al. 2018, 2019), cutting plane method (Jiang et al. 2020), generative adversarial networks (Xiao, Zhong, and Zheng 2018) and linear programming (Lee, Song, and Zhang 2019; Ye 2020; Jiang et al. 2021; Song and Yu 2021; Dong, Lee, and Ye 2021).

Graph neural network (GNN) Graph neural networks are machine learning methods that can operate on graph data (Zhang, Cui, and Zhu 2020; Wu et al. 2020b; Chami et al. 2020; Zhou et al. 2020), which has applications in various areas including social science (Wu et al. 2020a), natural science (Sanchez-Gonzalez et al. 2018; Fout 2017), knowledge graphs (Hamaguchi et al. 2017), 3D perception (Wang et al. 2019; Sarlin et al. 2020), autonomous driving (Shi and Rajkumar 2020; Weng et al. 2020b), and many other research areas (Dai et al. 2017). Due to its convincing performance, GNN has become a widely applied graph learning method recently.

Neural tangent kernel (NTK) The theory of NTK has been proposed to interpret the learnablity of neural networks. Given neural network $f : \mathcal{W} \times \mathcal{D} \longrightarrow \mathbb{R}$ with parameter $W \in \mathcal{W}$ and input data $x \in \mathcal{D}$, the neural tangent kernel between data x, y is defined to be (Jacot, Gabriel, and Hongler 2018)

$$\mathsf{K}_{\mathrm{ntk}}(x, y) := \mathop{\mathbb{E}}_{W \sim \mathcal{N}} \left[\left\langle \frac{\partial f(W, x)}{\partial W}, \frac{\partial f(W, y)}{\partial W} \right\rangle \right].$$

Here expectation $\mathbb{E}$ is over random Gaussian initialization. It has been shown under the assumption of the NTK matrix being positive-definite (Du et al. 2019b; Arora et al. 2019a,b; Song and Yang 2019; Lee et al. 2020; Brand et al. 2021; Song, Yang, and Zhang 2021; Song, Zhang, and Zhang 2021)) or separability of training data points (Li and Liang 2018; Allen-Zhu, Li, and Song 2019b; Song, Yang, and Zhang 2021; Allen-Zhu, Li, and Song 2019a)), training (regularized) neural network is equivalent to solving the neural tangent kernel (ridge) regression as long as the neural network is polynomial sufficiently wide. In the case of graph neural network, we study the corresponding graph neural tangent kernel (Du et al. 2019a).

Notations. Let n be an integer, we define $[n] := \{1, 2, \cdots, n\}$. For a full rank square matrix A, we use A^{-1} to denote its true inverse. We define the big O notation such that $f(n) = O(g(n))$ means there exists $n_0 \in \mathbb{N}_+$ and $M \in \mathbb{R}$ such that $f(n) \leq M \cdot g(n)$ for all $n \geq n_0$. For a matrix A, we use $\|A\|$ or $\|A\|_2$ to denote its spectral norm. Let $\|A\|_F$ denote its Frobenius norm. Let $A^\top$ denote the transpose of A. For a matrix A and a vector x, we define $\|x\|_A := \sqrt{x^\top A x}$. We use ϕ to denote the ReLU activation function, i.e. $\phi(z) = \max\{z, 0\}$.

Preliminaries

In this work, we consider a vanilla graph neural network (GNN) model consisting of three operations: AGGREGATE, COMBINE and READOUT. In each level of GNN, we have one AGGREGATE operation followed by a COMBINE operation, which consists of R fully-connected layers. At the end of the L level, the GNN outputs using a READOUT operation, which can be viewed as a pooling operation.

Specifically, let $G = (U, E)$ be a graph with node set U and edge set E. Assume it contains $|U| = N$ nodes. Each node $u \in U$ is given a feature vector $h_u \in \mathbb{R}^d$. We formally define the graph neural network $f_{\mathrm{gnn}}(G)$ as follows. We use the notation $h_u^{(l,r)}$ to denote the intermediate output at level $l \in [L]$ of the GNN and layer $r \in [R]$ of the COMBINE operation.

To start with, we set the initial vector $h_u^{(0,R)} = h_u \in \mathbb{R}^d$, $\forall u \in U$. For the first L layer, we have the following AGGREGATE and COMBINE operations.

AGGREGATE operation. There are in total L AGGREGATE operations. For any $l \in [L]$, the AGGREGATE operation aggregates the information from last level as follows:

$$h_u^{(l,0)} := c_u \cdot \sum_{a \in \mathcal{N}(u) \cup \{u\}} h_a^{(l-1,R)}.$$

Note that the vectors $h_u^{(l,0)} \in \mathbb{R}^m$ for all $l \in [2:L]$, and the only special case is $h_u^{(1,0)} \in \mathbb{R}^d$. $c_u \in \mathbb{R}$ is a scaling parameter, which controls weight of different nodes during neighborhood aggregation.

COMBINE operation. The COMBINE operation has R fully-connected layers with ReLU activation: $\forall r \in [R]$,

$$h_u^{(l,r)} := (c_\phi/m)^{1/2} \cdot \phi(W^{(l,r)} \cdot h_u^{(l,r-1)}) \in \mathbb{R}^m.$$

The parameters $W^{(l,r)} \in \mathbb{R}^{m \times m}$ for all $(l,r) \in [L] \times [R] \backslash \{(1,1)\}$, and the only special case is $W^{(1,1)} \in \mathbb{R}^{m \times d}$. $c_\phi \in \mathbb{R}$ is a scaling parameter, which is set to be 2, following the initialization scheme used in (Du et al. 2019a; He et al. 2015).

After the first L layer, we have the final READOUT operation before output.

READOUT operation. We consider two different kinds of READOUT operation.

1) In the simplest READOUT operation, the final output of the GNN on graph G is

$$f_{\text{gnn}}(G) := \sum_{u \in U} h_u^{(L,R)} \in \mathbb{R}^m.$$

2) Using the READOUT operation with jumping knowledge as in (Xu et al. 2018), the final output of the GNN on graph G is

$$f_{\text{gnn}}(G) := \sum_{u \in U} [h_u^{(0,R)}, h_u^{(1,R)}, \cdots, h_u^{(L,R)}] \in \mathbb{R}^{m \times (L+1)}.$$

When the context is clear, we also write $f_{\text{gnn}}(G)$ as $f_{\text{gnn}}(W, G)$, where W denotes all the parameters: $W = (W^{(1,1)}, \cdots, W^{(L,R)})$.

We also introduce the following current matrix multiplication time and the notations for Kronecker product and vectorization for completeness.

Fast matrix multiplication. We use the notation $\mathcal{T}_{\text{mat}}(n,d,m)$ to denote the time of multiplying an $n \times d$ matrix with another $d \times m$ matrix. Let ω denote the exponent of matrix multiplication, i.e., $\mathcal{T}_{\text{mat}}(n,n,n) = n^\omega$. The first result shows $\omega < 3$ is (Strassen 1969). The current best exponent is $\omega \approx 2.373$, due to (Williams 2012; Le Gall 2014). The common belief is $\omega \approx 2$ in the computational complexity community (Cohn et al. 2005; Williams 2012; Jiang et al. 2021). The following fact is well-known in the fast matrix multiplication literature (Coppersmith 1982; Strassen 1991; Bürgisser, Clausen, and Shokrollahi 1997) : $\mathcal{T}_{\text{mat}}(a,b,c) = O(\mathcal{T}_{\text{mat}}(a,c,b)) = O(\mathcal{T}_{\text{mat}}(c,a,b))$ for any positive integers a, b, c.

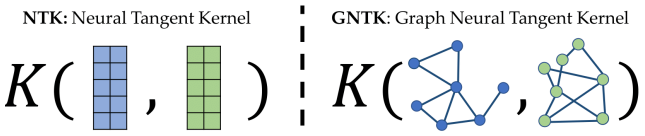

Figure 1: Illustration of NTK (left) and GNTK (right). Here $\mathsf{K}(\cdot,\cdot)$ denotes the kernel function.

Kronecker product and vectorization. Given two matrices $A \in \mathbb{R}^{n_1 \times d_1}$ and $B \in \mathbb{R}^{n_2 \times d_2}$. We use $\otimes$ to denote the Kronecker product, i.e., for $C = A \otimes B \in \mathbb{R}^{n_1 n_2 \times d_1 d_2}$, the $(i_1 + (i_2 - 1) \cdot n_1, j_1 + (j_2 - 1) \cdot d_1)$-th entry of C is $A_{i_1,j_1} B_{i_2,j_2}$, $\forall i_1 \in [n_1], i_2 \in [n_2], j_1 \in [d_1], j_2 \in [d_2]$. For any given matrix $H \in \mathbb{R}^{d_1 \times d_2}$, we use $h = \text{vec}(H) \in \mathbb{R}^{d_1 d_2}$ to denote the vector such that $h_{j_1 + (j_2-1) \cdot d_1} = H_{j_1,j_2}$, $\forall j_1 \in [d_1], j_2 \in [d_2]$.

Graph Neural Tangent Kernel Revisited

In this section, we revisit the graph neural tangent kernel (GNTK) proposed in (Du et al. 2019a). A simplified illustration is shown in Fig. 1. Following the setting discussed in previous section, let $G = (U, E)$ and $H = (V, F)$ be two graphs with $|U| = N$ and $|V| = N'$. We use A_G and A_H to denote the adjacency matrix of G and H. We give the recursive formula to compute the kernel value $\mathsf{K}_{\text{gntk}}(G,H) \in \mathbb{R}$ induced by this GNN, which is defined as

$$\mathsf{K}_{\text{gntk}}(G,H) := \mathop{\mathbb{E}}_{W \sim \mathcal{N}(0,I)} \Big[\Big\langle \frac{\partial f_{\text{gnn}}(W,G)}{\partial W}, \frac{\partial f_{\text{gnn}}(W,H)}{\partial W} \Big\rangle \Big],$$

where $\mathcal{N}(0,I)$ is a multivariate Gaussian distribution.

Recall that the GNN uses scaling factors c_u for each node $u \in G$. We define C_G to be the diagonal matrix such that $(C_G)_u = c_u$ for any $u \in U$. Similarly we define C_H. For each level of AGGREGATE and COMBINE operations $\ell \in [0:L]$ and each level of fully-connected layers inside a COMBINE operation $r \in [0:R]$, we recursively define the intermediate matrices $\Sigma^{(\ell,r)}(G,H) \in \mathbb{R}^{N \times N'}$ and $K^{(\ell,r)}(G,H) \in \mathbb{R}^{N \times N'}$.

Initially we define $\Sigma^{(0,R)}(G,H)$, $K^{(0,R)}(G,H) \in \mathbb{R}^{N \times N'}$ as follows: $\forall u \in U, v \in V$,

$$[\Sigma^{(0,R)}(G,H)]_{u,v} := \langle h_u, h_v \rangle,$$
$$[K^{(0,R)}(G,H)]_{u,v} := \langle h_u, h_v \rangle.$$

Here $h_u, h_v \in \mathbb{R}^d$ denote the input features of u and v. Next we recursively define $\Sigma^{(\ell,r)}(G,H)$ and $K^{(\ell,r)}(G,H)$ for $l \in [L]$ and $r \in [R]$ by interpreting the AGGREGATE and COMBINE operation.

Exact AGGREGATE operation. The AGGREGATE operation gives the following formula:

$$\begin{aligned} &[\Sigma^{(\ell,0)}(G,H)]_{u,v} \\ :=\ & c_u c_v \sum_{a \in \mathcal{N}(u) \cup \{u\}} \sum_{b \in \mathcal{N}(v) \cup \{v\}} [\Sigma^{(\ell-1,R)}(G,H)]_{a,b}, \\ &[K^{(\ell,0)}(G,H)]_{u,v} \\ :=\ & c_u c_v \sum_{a \in \mathcal{N}(u) \cup \{u\}} \sum_{b \in \mathcal{N}(v) \cup \{v\}} [K^{(\ell-1,R)}(G,H)]_{a,b}. \end{aligned} \tag{1}$$

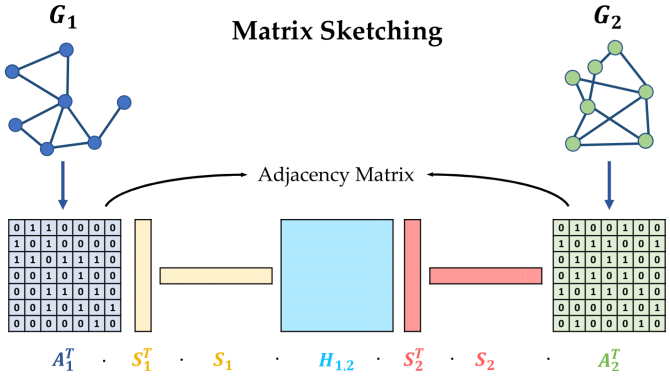

Figure 2: Illustration of our proposed sketching mechanism. Graphs G_1, G_2 are represented as adjacent matrices A_1, A_2. We apply sketching matrices to approximately calculate $A_1^\top H_{1,2} A_2^\top$.

Exact COMBINE operation. The COMBINE operation has R fully-connected layers with ReLU activation $\phi(z) = \max\{0, z\}$. We use $\dot{\phi}(z) = \mathbf{1}[z \geq 0]$ to denote be the derivative of ϕ.

For each $r \in [R]$, for each $u \in U$ and $v \in V$, we define a covariance matrix

$$[A^{(\ell,r)}(G,H)]_{u,v} := \begin{pmatrix} [\Sigma^{(\ell,r-1)}(G,G)]_{u,u} & [\Sigma^{(\ell,r-1)}(G,H)]_{u,v} \\ [\Sigma^{(\ell,r-1)}(H,G)]_{u,v} & [\Sigma^{(\ell,r-1)}(H,H)]_{v,v} \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

Then we recursively define $[\Sigma^{(\ell,r)}(G,H)]_{u,v}$ and $[K^{(\ell,r)}(G,H)]_{u,v}$ as follows:

$$[\Sigma^{(\ell,r)}(G,H)]_{u,v} := c_\phi \cdot \mathop{\mathbb{E}}_{(a,b)\sim\mathcal{N}(0,[A^{(\ell,r)}(G,H)]_{u,v})} [\phi(a)\phi(b)],$$

$$[\dot{\Sigma}^{(\ell,r)}(G,H)]_{u,v} := c_\phi \cdot \mathop{\mathbb{E}}_{(a,b)\sim\mathcal{N}(0,[A^{(\ell,r)}(G,H)]_{u,v})} [\dot{\phi}(a)\dot{\phi}(b)],$$

$$[K^{(\ell,r)}(G,H)]_{u,v} := [K^{(\ell,r-1)}(G,H)]_{u,v} \cdot [\dot{\Sigma}^{(\ell,r)}(G,H)]_{u,v} + [\Sigma^{(\ell,r)}(G,H)]_{u,v}.$$

The intermediate results will be used to calculate the final output of the corresponding GNTK.

Exact READOUT operation. As the final step, we compute $\mathsf{K}_{\mathrm{gntk}}(G,H) \in \mathbb{R}$ using the intermediate matrices. This step corresponds to the READOUT operation.

If we do not use jumping knowledge,

$$\mathsf{K}_{\mathrm{gntk}}(G,H) = \sum_{u\in U, v\in V} [K^{(L,R)}(G,H)]_{u,v}.$$

If we use jumping knowledge,

$$\mathsf{K}_{\mathrm{gntk}}(G,H) = \sum_{u\in U, v\in V} \sum_{l=0}^{L} [K^{(l,R)}(G,H)]_{u,v}.$$

We briefly review the running time in previous work.

Theorem 1 (Running time of (Du et al. 2019a), simplified)**.** *Consider a GNN with L AGGREGATE operations and L COMBINE operations, and each COMBINE operation has R fully-connected layers. We compute the kernel matrix using n graphs $\{G_i = (V_i, E_i)\}_{i=1}^n$ with $|V_i| \leq N$. Let $d \in \mathbb{N}_+$ be the dimension of the feature vectors. The total running time is*

$$O(n^2) \cdot (\mathcal{T}_{\mathrm{mat}}(N,N,d) + L \cdot N^4 + LR \cdot N^2).$$

When using GNN, we usually use constant number of operations and fully-connected layers, i.e., $L = O(1), R = O(1)$, and we have $d = o(N)$, while the size of the graphs can grow arbitrarily large. Thus it is easy to see that the dominating term in the above running time is $O(n^2N^4)$.

Approximate GNTK via Iterative Sketching

Now we consider the running time of solving GNTK regression. Despite rich related work to accelerate kernel regression by constructing random feature vector using sampling and sketching methods, they all require sufficient knowledge of the kernel matrix itself. While Theorem 1 shows the running complexity of constructing GNTK can be as large as $O(n^2N^4)$, which dominates the overall computation complexity especially in the case of relative large graphs $n = o(N^2)$. Therefore, our main task focus on a fast construction of the GNTK by randomization techniques, while ensuring the resulting GNTK regression give same generalization guarantee.

To compute an approximate version of the kernel value $\widetilde{K}(G,H) \in \mathbb{R}$ such that $\widetilde{\mathsf{K}}_{\mathrm{gntk}}(G,H) \approx \mathsf{K}_{\mathrm{gntk}}(G,H)$, we will use the notation $\widetilde{\Sigma}^{(\ell,r)}(G,H) \in \mathbb{R}^{N\times N'}$ and $\widetilde{K}^{(\ell,r)}(G,H) \in \mathbb{R}^{N\times N'}$ to denote the approximated intermediate matrices for each $\ell \in [0:L]$ and $r \in [0:R]$. We observe that the computation bottleneck is to conduct the AGGREGATE operation (1), which takes $O(N^4)$ for obtaining $\Sigma^{(\ell,0)}(G,H)$ and $K^{(\ell,0)}(G,H)$. Hence we apply random sketching matrices $S_G \in \mathbb{R}^{b\times N}$ and $S_H \in \mathbb{R}^{b'\times N'}$ iteratively to accelerate the computation. Our proposed sketching mechanism is shown in Fig. 2.

Approximate AGGREGATE operation. We note that Eq. (1) can be equivalently rewrite in following forms us-

ing Kronecker product:

$$\begin{aligned}\mathrm{vec}(\Sigma^{(\ell,0)}(G,H)) &:= ((C_G A_G)\otimes(C_H A_H)) \\ &\quad \cdot \mathrm{vec}(\Sigma^{(\ell-1,R)}(G,H)), \\ \mathrm{vec}(K^{(\ell,0)}(G,H)) &:= ((C_G A_G)\otimes(C_H A_H)) \\ &\quad \cdot \mathrm{vec}(K^{(\ell-1,R)}(G,H)).\end{aligned} \tag{2}$$

Now we make the following key observation about kronecker product and vectorization.

Fact 2 (Equivalence between two matrix products and Kronecker product then matrix-vector multiplication)**.** *Given matrices $A \in \mathbb{R}^{n_1\times d_1}$, $B \in \mathbb{R}^{n_2\times d_2}$, and $H \in \mathbb{R}^{d_1\times d_2}$, we have* $\mathrm{vec}(AHB^\top) = (A\otimes B)\cdot \mathrm{vec}(H)$.

Above fact implies the intermediate matrices $\Sigma^{(\ell,0)}(G,H)$ and $K^{(\ell,0)}(G,H)$ can be calculated by

$$\begin{aligned}\Sigma^{(\ell,0)}(G,H) &:= C_G A_G \cdot \widetilde{\Sigma}^{(\ell-1,R)}(G,H)\cdot A_H C_H, \\ K^{(\ell,0)}(G,H) &:= C_G A_G \cdot \widetilde{K}^{(\ell-1,R)}(G,H)\cdot A_H C_H.\end{aligned} \tag{3}$$

We emphasize that the above Eq. (3) calculates matrix production instead of Kronecker product, which reduces the running time from $O(N^4)$ to $O(\mathcal{T}_{\mathrm{mat}}(N,N,N))$. This is our first improvement in running time.

Further, we propose to introduce sketching matrices $S_G \in \mathbb{R}^{b\times N}$ and $S_H \in \mathbb{R}^{b'\times N'}$ iteratively into Eq. (3) as follows:

$$\begin{aligned}\widetilde{\Sigma}^{(\ell,0)}(G,H) &:= C_G A_G \cdot (S_G^\top S_G)\cdot \widetilde{\Sigma}^{(\ell-1,R)}(G,H) \\ &\quad \cdot (S_H^\top S_H)\cdot A_H C_H, \\ \widetilde{K}^{(\ell,0)}(G,H) &:= C_G A_G \cdot (S_G^\top S_G)\cdot \widetilde{K}^{(\ell-1,R)}(G,H) \\ &\quad \cdot (S_H^\top S_H)\cdot A_H C_H.\end{aligned} \tag{4}$$

Note that for the special case $S_G^\top S_G = S_H^\top S_H = I$, the Eq. (4) degenerates to the original case. Such randomization ensures the sketched version approximates the exact matrix multiplication as in the following Lemma, which justifies our approach to speed up calculation.

Lemma 3 (Informal, Error bound of adding sketching)**.** *Let $S_i \in \mathbb{R}^{b\times N}$'s denote independent AMS matrices (Alon, Matias, and Szegedy 1999). Then for any given n^2 matrices $H_{1,1},\cdots,H_{n,n} \in \mathbb{R}^{N\times N}$ and n matrices $A_1,\cdots,A_n \in \mathbb{R}^{N\times N}$, we have the following guarantee with high probability: for all $i,j\in[n]$, $A_i^\top S_i^\top S_i H_{i,j} S_j^\top S_j A_j \approx A_i^\top H_{i,j} A_j$.*

Apart from above mentioned AMS sketching matrices, we point out other well-known sketching matrices such as random Gaussian, SRHT (Lu et al. 2013), countsketch (Charikar, Chen, and Farach-Colton 2002), sparse embedding matrices (Nelson and Nguyên 2013) also apply in this case.

In the end, the **COMBINE** and **READOUT** operation are the same as in the exact case, except now we are always working with the approximated intermediate matrices $\widetilde{\Sigma}^{(\ell,0)}(G,H)$ and $\widetilde{K}^{(\ell,0)}(G,H)$.

Running Time Analysis

The main contribution of our paper is to show that we can accelerate the computation of GNTK defined in (Du et al. 2019a), while maintaining a similar generalization bound.

In this section we first present our main running time improvement theorem.

Theorem 4 (Main theorem, running time)**.** *Consider a GNN with L* AGGREGATE *operations and L* COMBINE *operations, and each* COMBINE *operation has R fully-connected layers. We compute the kernel matrix using n graphs $\{G_i = (V_i,E_i)\}_{i=1}^n$ with $|V_i|\le N$. Let $b = o(N)$ be the sketch size. Let $d\in\mathbb{N}_+$ be the dimension of the feature vectors. The total running time is*

$$O(n^2)\cdot(\mathcal{T}_{\mathrm{mat}}(N,N,d) + L\cdot\mathcal{T}_{\mathrm{mat}}(N,N,b) + LR\cdot N^2).$$

Proof Sketch:. Our main improvement focus on the AGGREGATE operations, where we first use the equivalent form (3) to accelerate the exact computation from $O(N^4)$ to $\mathcal{T}_{\mathrm{mat}}(N,N,N)$.

Further, by introducing the iterative sketching (4), with an appropriate ordering of computation, we can avoid the time-consuming step of multiplying two $N\times N$ matrices. Specifically, by denoting $A_i = C_{G_i}A_{G_i}, A_j = C_{G_j}A_{G_j}, H_{i,j} = \widetilde{K}^{(l-1,R)}(G_i,G_j)$, we compute Eq. (4), i.e., $A_i^\top S_i^\top S_i H_{i,j} S_j S_j^\top A_j$ in the following order:

- $A_i^\top S_i^\top$ and $S_j A_j$ both takes $\mathcal{T}_{\mathrm{mat}}(N,N,b)$ time.
- $S_i\cdot H_{i,j}$ takes $\mathcal{T}_{\mathrm{mat}}(b,N,N)$ time.
- $(S_i H_{i,j})\cdot S_j^\top$ takes $\mathcal{T}_{\mathrm{mat}}(b,N,b)$ time.
- $(A_i^\top S_i^\top)\cdot(S_i H_{i,j} S_j)$ takes $\mathcal{T}_{\mathrm{mat}}(N,b,b)$ time.
- $(A_i^\top S_i^\top S_i H_{i,j} S_j)\cdot(S_j A_j)$ takes $\mathcal{T}_{\mathrm{mat}}(N,b,N)$ time.

Thus, we improve the running time from $\mathcal{T}_{\mathrm{mat}}(N,N,N)$ to $\mathcal{T}_{\mathrm{mat}}(N,N,b)$. □

Note that we improve the dominating term from N^4 to $\mathcal{T}_{\mathrm{mat}}(N,N,b)$. We achieve this improvement using two techniques:

1. We accelerate the multiplication of a Kronecker product with a vector by decoupling it into two matrix multiplications of smaller dimensions. In this way we improve the running time from N^4 down to $\mathcal{T}_{\mathrm{mat}}(N,N,N)$.
2. We further accelerate the two matrix multiplications by using two sketching matrices. In this way, we improve the running time from $\mathcal{T}_{\mathrm{mat}}(N,N,N)$ to $\mathcal{T}_{\mathrm{mat}}(N,N,b)$.

Error Analysis

In this section, we prove that the introduced error due to the added sketching in calculating GNTK can be well-bounded, thus we can prove a similar generalization result.

We first list all the notations and assumptions we used before proving the generalization bound.

Definition 5 (Approximate GNTK with n data)**.** *Let $\{(G_i,y_i)\}_{i=1}^n$ be the training data and labels, and $G_i = (V_i,E_i)$ with $|V_i| = N_i$, and we assume $N_i = O(N)$,*

$\forall i \in [n]$. *For each* $i \in [n]$ *and each* $u \in V_i$, *let* $h_u \in \mathbb{R}_+^d$ *be the feature vector for* u, *and we define feature matrix*

$$H_{G_i} := [h_{u_1}, h_{u_2}, \cdots, h_{u_{N_i}}] \in \mathbb{R}_+^{d \times N_i}.$$

We also define $A_{G_i} \in \mathbb{R}^{N_i \times N_i}$ *to be the adjacency matrix of* G_i, *and* $S_{G_i} \in \mathbb{R}^{b_i \times N_i}$ *to be the sketching matrix used for* G_i.

Let $\widetilde{K} \in \mathbb{R}^{n \times n}$ *be the approximate GNTK of a GNN that has one* AGGREGATE *operation followed by one* COMBINE *operation with one fully-connected layer (*$L = 1$ *and* $R = 1$*) and without jumping knowledge. For each* $l \in [L]$, $r \in [R]$, $i, j \in [n]$, *let* $\widetilde{\Sigma}^{(l,r)}(G_i, G_j), \widetilde{K}^{(l,r)}(G_i, G_j) \in \mathbb{R}^{N_i \times N_j}$ *be defined as in Eq.* (4).

We set the scaling parameters used by the GNN for G_i *are* $c_\phi = 2$ *and* $c_u = (\|[H_{G_i} S_{G_i}^\top S_{G_i} A_{G_i}]_{*,u}\|_2)^{-1}$, *for each* $u \in V_i$. *We use* $C_{G_i} \in \mathbb{R}^{N_i \times N_i}$ *to denote the diagonal matrix with* $[C_{G_i}]_{u,u} = c_u$.

We further define two vectors for each $i \in [n]$ *and each* $u \in V_i$:

$$\overline{h}_u := [H_{G_i} A_{G_i} C_{G_i}]_{*,u} \in \mathbb{R}^d, \tag{5}$$

$$\widetilde{h}_u := [H_{G_i} S_{G_i}^\top S_{G_i} A_{G_i} C_{G_i}]_{*,u} \in \mathbb{R}^d. \tag{6}$$

And let $T \in \mathbb{N}_+$ *be a integer. For each* $t \in \mathbb{N}_+$, *we define two matrices* $\overline{H}^{(t)}, \widetilde{H}^{(t)} \in \mathbb{R}^{d \times n}$:

$$\overline{H}^{(t)} := \Big[\sum_{u \in V_1} \Phi^{(t)}(\overline{h}_u), \cdots, \sum_{u \in V_n} \Phi^{(t)}(\overline{h}_u) \Big] \in \mathbb{R}^{d \times n}, \tag{7}$$

$$\widetilde{H}^{(t)} := \Big[\sum_{u \in V_1} \Phi^{(t)}(\widetilde{h}_u), \cdots, \sum_{u \in V_n} \Phi^{(t)}(\widetilde{h}_u) \Big] \in \mathbb{R}^{d \times n}, \tag{8}$$

where we define $\Phi^{(t)}(\cdot)$ *to be the feature map of the polynomial kernel of degree* t *s.t.*

$$\langle x, y \rangle^t = \langle \Phi^{(t)}(x), \Phi^{(t)}(y) \rangle \quad \forall x, y \in \mathbb{R}^d.$$

Assumption 6. *We assume the following properties about the input graphs, its feature vectors, and its labels.*

1. **Labels.** *We assume for all* $i \in [n]$, *the label* $y_i \in \mathbb{R}$ *we want to learn satisfies*

$$y_i = \alpha_1 \sum_{u \in V_i} \langle \overline{h}_u, \beta_1 \rangle + \sum_{l=1}^{T} \alpha_{2l} \sum_{u \in V_i} \langle \overline{h}_u, \beta_{2l} \rangle^{2l}, \tag{9}$$

   Note that $\alpha_1, \alpha_2, \cdots, \alpha_{2T}$ *are scalars,* $\beta_1, \beta_2, \cdots, \beta_{2T}$ *are vectors in* d*-dimensional space.*
2. **Feature vectors and graphs.** *For each* $t \in \{1\} \cup \{2l\}_{l=1}^T$, *we assume we have* $\|(\overline{H}^{(t)})^\top \overline{H}^{(t)}\|_F \le \gamma_t \cdot \|(\overline{H}^{(t)})^\top \overline{H}^{(t)}\|_2$, *where* $\gamma_1, \gamma_2, \gamma_4, \cdots, \gamma_{2T} \in \mathbb{R}$. *We also let* $\gamma = \max_{t \in \{1\} \cup \{2l\}_{l=1}^T} \{\gamma_t\}$. *Note that* $\gamma \ge 1$.
3. **Sketching sizes.** *We assume the sketching sizes* $\{b_i\}_{i=1}^n$ *satisfy that* $\forall i, j \in [n]$,

$$\|A_{G_j} C_{G_j} \mathbf{1}_{[N_j]}\|_2 \|A_{G_i} C_{G_i} \mathbf{1}_{[N_i]}\|_2 \cdot \|H_{G_j}^\top H_{G_i}\|_F$$

$$\lesssim \frac{\min\{\sqrt{b_i}, \sqrt{b_j}\}}{\gamma T \log^3 N} \mathbf{1}_{[N_i]}^\top C_{G_i}^\top A_{G_i}^\top H_{G_i}^\top H_{G_j} A_{G_j} C_{G_j} \mathbf{1}_{[N_j]},$$

   where $\mathbf{1}_{[N_i]} \in \mathbb{R}^{N_i}, \mathbf{1}_{[N_j]} \in \mathbb{R}^{N_j}$ *are the all one vectors of size* N_i *and* N_j.

We now provide the generalization bound of our work. We start with a standard tool.

Theorem 7 ((Bartlett and Mendelson 2002))**.** *Consider* n *training data* $\{(G_i, y_i)\}_{i=1}^n$ *drawn i.i.d. from distribution* $\mathcal{D}$ *and* 1*-Lipschitz loss function* $\ell : \mathbb{R} \times \mathbb{R} \to [0, 1]$ *satisfying* $\ell(y, y) = 0$. *Then with probability at least* $1 - \delta$, *the population loss of the* GNTK *predictor* f_{gntk} *is upper bounded by*

$$L_{\mathcal{D}}(f_{\mathrm{gntk}}) = \mathop{\mathbb{E}}_{(G,y) \sim \mathcal{D}} [\ell(f_{\mathrm{gntk}}(G), y)] \lesssim (\|y\|_{\widetilde{K}^{-1}}^2 \cdot \mathrm{tr}[\widetilde{K}])^{1/2}/n + \sqrt{\log(1/\delta)/n}.$$

Now it remains to bound $\|y\|_{\widetilde{K}^{-1}}$ and $\mathrm{tr}[\widetilde{K}]$. We have the following three technical lemmas to address $\|y\|_{\widetilde{K}^{-1}}$ and $\mathrm{tr}[\widetilde{K}]$. To start with, we give a close-form reformulation of our approximate GNTK.

Lemma 8 (Close-form formula of approximate GNTK)**.** *Following the notations of Definition 5, we can decompose* $\widetilde{K} \in \mathbb{R}^{n \times n}$ *into* $\widetilde{K} = \widetilde{K}_1 + \widetilde{K}_2 \succeq \widetilde{K}_1$, *where* $\widetilde{K}_2 \in \mathbb{R}^{n \times n}$ *is a PSD matrix, and* $\widetilde{K}_1 \in \mathbb{R}^{n \times n}$. $\widetilde{K}_1$ *satisfies the following:*

$$\widetilde{K}_1 = \frac{1}{4} (\widetilde{H}^{(1)})^\top \cdot \widetilde{H}^{(1)} + \frac{1}{2\pi} \sum_{l=1}^{\infty} c_l \cdot (\widetilde{H}^{(2l)})^\top \cdot \widetilde{H}^{(2l)},$$

where $c_l = \frac{(2l-3)!!}{(2l-2)!!(2l-1)}$ *and equivalently for each* $i, j \in [n]$, $\widetilde{K}_1(G_i, G_j) \in \mathbb{R}$ *satisfies the following:*

$$\widetilde{K}_1(G_i, G_j) = \sum_{u \in V_i} \sum_{v \in V_j} \langle \widetilde{h}_u, \widetilde{h}_v \rangle \cdot \frac{1}{2\pi} \big(\pi - \arccos(\langle \widetilde{h}_u, \widetilde{h}_v \rangle)\big),$$

For each $i, j \in [n]$, $\widetilde{K}_2(G_i, G_j) \in \mathbb{R}$ *satisfies the following:*

$$\widetilde{K}_2(G_i, G_j) = \sum_{u \in V_i} \sum_{v \in V_j} \langle \widetilde{h}_u, \widetilde{h}_v \rangle \cdot \frac{1}{2\pi} \Big(\pi - \arccos(\langle \widetilde{h}_u, \widetilde{h}_v \rangle) + \sqrt{1 - \langle \widetilde{h}_u, \widetilde{h}_v \rangle^2}\Big).$$

Based upon the above characterization, we are ready to bound $y^\top \widetilde{K}^{-1} y$, as a generalization of Theorem 4.2 of (Du et al. 2019a).

Lemma 9 (Bound on $y^\top \widetilde{K}^{-1} y$)**.** *Following the notations of Definition 5 and under the assumptions of Assumption 6, we have*

$$\|y\|_{\widetilde{K}^{-1}} \le 4 \cdot |\alpha_1| \|\beta_1\|_2 + \sum_{l=1}^{T} 4\sqrt{\pi}(2l-1) \cdot |\alpha_{2l}| \|\beta_{2l}\|_2.$$

We provide a high-level proof sketch here. We first compute all the variables in the approximate GNTK formula to get a close-form formula of $\widetilde{K}$. Then combining with the

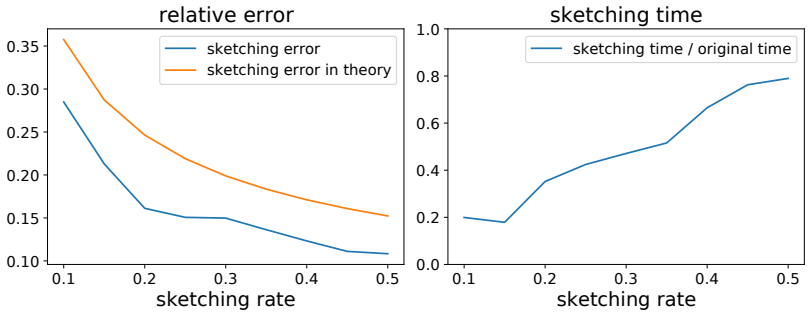

Figure 3: Comparison between theoretical and experimental sketching errors (left) and sketching time (right) under different sketching rates.

assumption on the labels y, we show that $\|y\|_{\widetilde{K}_1^{-1}}$ is upper bounded by

$$\|y\|_{\widetilde{K}_1^{-1}} \leq (4\alpha^2 \cdot \beta^\top \overline{H} \cdot (\widetilde{H}^\top \widetilde{H})^{-1} \cdot \overline{H}^\top \beta)^{1/2}, \quad (10)$$

where $\overline{H}, \widetilde{H} \in \mathbb{R}^{d\times n}$ are two matrices such that $\forall i,j \in [n]$, $[\overline{H}^\top \overline{H}]_{i,j} = \mathbf{1}_{N_i}^\top C_{G_i}^\top A_{G_i}^\top H_{G_i}^\top \cdot H_{G_j} A_{G_j} C_{G_j} \mathbf{1}_{N_j}$, and $[\widetilde{H}^\top \widetilde{H}]_{i,j} = \mathbf{1}_{N_i}^\top C_{G_i}^\top A_{G_i}^\top (S_{G_i}^\top S_{G_i}) H_{G_i}^\top \cdot H_{G_j} (S_{G_j}^\top S_{G_j}) A_{G_j} C_{G_j} \mathbf{1}_{N_j}$. Note by Lemma 3, we can show that the sketched version $\widetilde{H}^\top \widetilde{H}$ is a PSD approximation of $\overline{H}^\top \overline{H}$ in the sense of $(1 - \frac{1}{10})\overline{H}^\top \overline{H} \preceq \widetilde{H}^\top \widetilde{H} \preceq (1 + \frac{1}{10})\overline{H}^\top \overline{H}$. Plugging into Eq. 10 we can complete the proof.

Lastly, we give a bound on the trace of $\widetilde{K}$. We defer the proof to Appendix.

Lemma 10 (Bound on trace of $\widetilde{K}$)**.** *Following the notations of Definition 5 and under the assumptions of Assumption 6, we have* $\mathrm{tr}[\widetilde{K}] \leq 2nN^2$.

Combining Lemma 9 and Lemma 10 with Theorem 7, we conclude with the following main generalization theorem:

Theorem 11 (Main generalization theorem)**.** *Let $c \in (0,1)$ denote a fixed constant. Following the notations of Definition 5, and under the assumptions of Assumption 6, if we further have the conditions that*

$$4 \cdot \alpha_1 \|\beta_1\|_2 + \sum_{l=1}^{T} 4\sqrt{\pi}(2l-1) \cdot \alpha_{2l} \|\beta_{2l}\|_2 = o(n)$$

and $N = o(\sqrt{n})$, then we can upper bound the generalization error of the approximate GNTK by

$$L_{\mathcal{D}}(f_{\mathrm{gntk}}) = \mathop{\mathbb{E}}_{(G,y)\sim\mathcal{D}}[\ell(f_{\mathrm{gntk}}(G), y)] \lesssim O(1/n^c).$$

Above theorem shows that the approximate GNTK corresponds to the vanilla GNN described above is able to, with polynomial number of samples, learn functions of forms in (9). Such a guarantee is similar to the result for exact GNTK (Arora et al. 2019a), indicating our proposed sketching does not influence the generalization ability of GNTK.

We conduct experiments to validate that the error introduced by matrix sketching is strictly bounded. Following Lemma 3 , we validate the error difference between matrix multiplication with and without the sketching method. Specifically, we randomly generate $n \times n$ matrices A, G and H. And matrix multiplication without sketching is calculated by $M = G^\top AH$. For the sketching method, we randomly generate two AMS matrices R and S with size $\gamma n \times n$ where γ is the sketching ratio. And matrix multiplication with sketching is calculated by $M_{\text{sketch}} = G^\top R^\top RAS^\top SH$. The experimental error matrix is calculated by $|M - M_{\text{sketch}}|$, and the theoretical error matrix is calculated by the RHS of Lemma 3. We divide both errors by the original matrix M to show the relative error. And we show the final mean error by taking the average over all entries of the error matrices.

Fig. 3 shows the result. We set $n = 500$, and run experiments under different sketching rates from 0.1 to 0.9. We run each sketching rate for 100 times and calculate the mean error. We also show the matrix multiplication time with/without sketching. Experiments show that our sketching error is always lower than the theoretical bound. When the sketching rate is high, the error decreases and the running time increases because the dimension of the matrix is larger. This experiment validates our Lemma 3, showing that our matrix sketching method has a strictly bounded error.

Conclusion

Graph Neural Networks (GNNs) have recently become the most important method for machine learning on graph data (e.g., protein structures, code AST, social networks), but training GNNs efficiently is a major challenge. An alternative method is Graph Neural Tangent Kernel (GNTK). GNTK's parameters are solved directly in a single step. This avoids time-consuming gradient descent. GNTK has thus become the state-of-the-art method to achieve high training speed without compromising accuracy. In this paper, we accelerate the construction of GNTK by two steps: 1) accelerate the multiplication of a Kronecker product with a vector by decoupling it into two matrix multiplications of smaller dimensions; 2) we introduce sketching matrices iteratively to further accelerate the multiplication between two matrices. Our techniques speed up generating kernel matrices for GNTK and thus improve the end-to-end running time for GNTK regression.

References

Allen-Zhu, Z.; Li, Y.; and Song, Z. 2019a. A convergence theory for deep learning via over-parameterization. In *ICML*.

Allen-Zhu, Z.; Li, Y.; and Song, Z. 2019b. On the convergence rate of training recurrent neural networks. In *NeurIPS*.

Alon, N.; Matias, Y.; and Szegedy, M. 1999. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1): 137–147.

Andoni, A.; Lin, C.; Sheng, Y.; Zhong, P.; and Zhong, R. 2018. Subspace embedding and linear regression with Orlicz norm. In *ICML*, 224–233.

Arora, S.; Du, S.; Hu, W.; Li, Z.; and Wang, R. 2019a. Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. In *ICML*, 322–332.

Arora, S.; Du, S. S.; Hu, W.; Li, Z.; Salakhutdinov, R.; and Wang, R. 2019b. On exact computation with an infinitely wide neural net. In *NeurIPS*.

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

Boutsidis, C.; and Woodruff, D. P. 2014. Optimal CUR matrix decompositions. In *STOC*, 353–362. ACM, https://arxiv.org/pdf/1405.7910.

Boutsidis, C.; Woodruff, D. P.; and Zhong, P. 2016. Optimal principal component analysis in distributed and streaming models. In *STOC*, 236–249.

Brand, J. v. d.; Peng, B.; Song, Z.; and Weinstein, O. 2021. Training (Overparametrized) Neural Networks in Near-Linear Time. In *ITCS*.

Bürgisser, P.; Clausen, M.; and Shokrollahi, M. A. 1997. *Algebraic complexity theory*, volume 315. Springer Science & Business Media.

Chami, I.; Abu-El-Haija, S.; Perozzi, B.; Ré, C.; and Murphy, K. 2020. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*.

Charikar, M.; Chen, K.; and Farach-Colton, M. 2002. Finding frequent items in data streams. In *ICALP*.

Clarkson, K. L.; and Woodruff, D. P. 2013. Low rank approximation and regression in input sparsity time. In *STOC*, 81–90. https://arxiv.org/pdf/1207.6365.

Cohn, H.; Kleinberg, R.; Szegedy, B.; and Umans, C. 2005. Group-theoretic algorithms for matrix multiplication. In *FOCS*.

Coppersmith, D. 1982. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3): 467–471.

Dai, H.; Khalil, E. B.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*.

Diao, H.; Jayaram, R.; Song, Z.; Sun, W.; and Woodruff, D. 2019. Optimal sketching for kronecker product regression and low rank approximation. *NeurIPS*.

Diao, H.; Song, Z.; Sun, W.; and Woodruff, D. 2018. Sketching for kronecker product regression and p-splines. In *International Conference on Artificial Intelligence and Statistics*, 1299–1308. PMLR.

Dong, S.; Lee, Y. T.; and Ye, G. 2021. A Nearly-Linear Time Algorithm for Linear Programs with Small Treewidth: A Multiscale Representation of Robust Central Path. In *STOC*, STOC 2021, 1784–1797.

Du, S. S.; Hou, K.; Salakhutdinov, R. R.; Poczos, B.; Wang, R.; and Xu, K. 2019a. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *NeurIPS*, 5723–5733.

Du, S. S.; Zhai, X.; Poczos, B.; and Singh, A. 2019b. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*. arXiv preprint arXiv:1810.02054.

Fout, A. M. 2017. *Protein interface prediction using graph convolutional networks*. Ph.D. thesis, Colorado State University.

Hamaguchi, T.; Oiwa, H.; Shimbo, M.; and Matsumoto, Y. 2017. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*.

Jacot, A.; Gabriel, F.; and Hongler, C. 2018. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 8571–8580.

Jiang, H.; Lee, Y. T.; Song, Z.; and Wong, S. C.-w. 2020. An Improved Cutting Plane Method for Convex Optimization, Convex-Concave Games and its Applications. In *STOC*.

Jiang, S.; Song, Z.; Weinstein, O.; and Zhang, H. 2021. Faster dynamic matrix inverse for faster lps. In *STOC*.

Le Gall, F. 2014. Powers of tensors and fast matrix multiplication. In *ISSAC*, 296–303. ACM.

Lee, J. D.; Shen, R.; Song, Z.; Wang, M.; and Yu, Z. 2020. Generalized Leverage Score Sampling for Neural Networks. In *NeurIPS*.

Lee, Y. T.; Song, Z.; and Zhang, Q. 2019. Solving Empirical Risk Minimization in the Current Matrix Multiplication Time. In *COLT*. https://arxiv.org/pdf/1905.04447.pdf.

Li, Y.; and Liang, Y. 2018. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*.

Lu, Y.; Dhillon, P.; Foster, D. P.; and Ungar, L. 2013. Faster ridge regression via the subsampled randomized hadamard transform. In *NeurIPS*, 369–377.

Meng, X.; and Mahoney, M. W. 2013. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *STOC*, 91–100.

Nelson, J.; and Nguyên, H. L. 2013. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *FOCS*. IEEE.

Razenshteyn, I.; Song, Z.; and Woodruff, D. P. 2016. Weighted low rank approximations with provable guarantees. In *STOC*.

Sanchez-Gonzalez, A.; Heess, N.; Springenberg, J. T.; Merel, J.; Riedmiller, M.; Hadsell, R.; and Battaglia, P. 2018. Graph networks as learnable physics engines for inference and control. In *ICML*.

Sarlin, P.-E.; DeTone, D.; Malisiewicz, T.; and Rabinovich, A. 2020. SuperGlue: Learning Feature Matching With Graph Neural Networks. In *CVPR*.

Shi, W.; and Rajkumar, R. 2020. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *CVPR*.

Song, Z.; Woodruff, D.; Yu, Z.; and Zhang, L. 2021. Fast sketching of polynomial kernels of polynomial degree. In *ICML*.

Song, Z.; Woodruff, D. P.; and Zhong, P. 2017. Low Rank Approximation with Entrywise ℓ_1-Norm Error. In *STOC*.

Song, Z.; Woodruff, D. P.; and Zhong, P. 2019. Relative Error Tensor Low Rank Approximation. In *SODA*.

Song, Z.; Yang, S.; and Zhang, R. 2021. Does Preprocessing Help Training Over-parameterized Neural Networks? In *NeurIPS*.

Song, Z.; and Yang, X. 2019. Quadratic suffices for over-parametrization via matrix chernoff bound. In *arXiv preprint*. https://arxiv.org/pdf/1906.03593.pdf.

Song, Z.; and Yu, Z. 2021. Oblivious Sketching-based Central Path Method for Solving Linear Programming Problems. In *ICML*.

Song, Z.; Yu, Z.; and Zhang, L. 2021. Iterative Sketching and its Applications to Federated Learning. In *Manuscript*. https://openreview.net/forum?id=U_Jog0t3fAu.

Song, Z.; Zhang, L.; and Zhang, R. 2021. Training Multi-layer Over-parametrized Neural Networks in Subquadratic Time. In *Manuscript*. https://openreview.net/forum?id=OMxLn4t03FG.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4): 354–356.

Strassen, V. 1991. Degeneration and complexity of bilinear maps: some asymptotic spectra. *J. reine angew. Math*, 413: 127–180.

Wang, R.; Zhong, P.; Du, S. S.; Salakhutdinov, R. R.; and Yang, L. F. 2020. Planning with General Objective Functions: Going Beyond Total Rewards. In *NeurIPS*.

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5): 1–12.

Weng, X.; Wang, Y.; Man, Y.; and Kitani, K. M. 2020a. GNN3DMOT: Graph Neural Network for 3D Multi-Object Tracking With 2D-3D Multi-Feature Learning. In *CVPR*, 6499–6508.

Weng, X.; Wang, Y.; Man, Y.; and Kitani, K. M. 2020b. Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning. In *CVPR*.

Williams, V. V. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, 887–898. ACM.

Woodruff, D. P.; and Zhong, P. 2016. Distributed low rank approximation of implicit functions of a matrix. In *ICDE*, 847–858. IEEE.

Wu, Y.; Lian, D.; Xu, Y.; Wu, L.; and Chen, E. 2020a. Graph convolutional networks with markov random field reasoning for social spammer detection. In *AAAI*, volume 34, 1054–1061.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020b. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Xiao, C.; Zhong, P.; and Zheng, C. 2018. BourGAN: generative networks with metric embeddings. In *NeurIPS*, 2275–2286.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

Yang, C.; Zhang, J.; Wang, H.; Li, S.; Kim, M.; Walker, M.; Xiao, Y.; and Han, J. 2020a. Relation learning on social networks with multi-modal graph edge variational autoencoders. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 699–707.

Yang, Z.; Sun, Y.; Liu, S.; and Jia, J. 2020b. 3dssd: Point-based 3d single stage object detector. In *CVPR*, 11040–11048.

Ye, G. 2020. Fast Algorithm for Solving Structured Convex Programs. *The University of Washington, Undergraduate Thesis*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.

Yue, X.; Wang, Z.; Huang, J.; Parthasarathy, S.; Moosavinasab, S.; Huang, Y.; Lin, S. M.; Zhang, W.; Zhang, P.; and Sun, H. 2020. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4): 1241–1251.

Zhang, Z.; Cui, P.; and Zhu, W. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.

Zitnik, M.; and Leskovec, J. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14): i190–i198.