LABEL PROPAGATION ACROSS GRAPHS: NODE CLASSIFICATION USING GRAPH NEURAL TANGENT KERNELS

Artun Bayer, Arindam Chowdhury, and Santiago Segarra

Electrical and Computer Engineering, Rice University, USA

ABSTRACT

Graph neural networks (GNNs) have achieved superior performance on node classification tasks in the last few years. Commonly, this is framed in a transductive semi-supervised learning setup wherein the entire graph – including the target nodes to be labeled – is available for training. Driven in part by scalability, recent works have focused on the inductive case where only the labeled portion of a graph is available for training. In this context, our current work considers a challenging inductive setting where a set of labeled graphs are available for training while the unlabeled target graph is completely separate, i.e., there are no connections between labeled and unlabeled nodes. Under the implicit assumption that the testing and training graphs come from similar distributions, our goal is to develop a labeling function that generalizes to unobserved connectivity structures. To that end, we employ a graph neural tangent kernel (GNTK) that corresponds to infinitely wide GNNs to find correspondences between nodes in different graphs based on both the topology and the node features. We augment the capabilities of the GNTK with residual connections and empirically illustrate its performance gains on standard benchmarks.

Index Terms— Node classification, graph representation learning, graph neural network, neural tangent kernel.

1. INTRODUCTION

In this era of information revolution, data processing systems and methods are tasked with generating suitable representations of enormous amounts of data obtained from multiple domains and modalities. In addition to the individual attributes and features, it is crucial to extract essential relationships among multiple data points. Graphs have become ubiquitous in modern data processing and algorithms on account of the versatility that they offer in representing relational structures [1, 2]. The availability of vast quantities of network data have also paved the way for application of advanced signal processing [3, 4] and machine learning tools [5, 6] for graph representation learning.

Among these generalizations, the most prominent examples are graph neural networks (GNNs). A variety of GNN architectures [7–9], both centralized and distributed [10, 11] have been successfully applied to solve challenging problems in multiple fields. While these architectures vary in structure and scope, their core operating principle involves combining the information stored in connected nodes to generate useful node-level and graph-level embeddings [8] that can be leveraged for multiple downstream tasks like classification, regression, and clustering [12]. In particular, node classification [13, 14] is a well-studied application of node representation learning.

Gradient descent (GD) [15] is, arguably, the most popular learning algorithm for these connectionist methods. Its basic operation involves taking a sequence of gradient steps towards a desired optimum. However, tuning its operating conditions can be quite tricky [16]. Current research has shown that under certain limiting conditions on the neural architectures, GD resembles kernel regression with a specialized deterministic kernel, namely, the Neural Tangent Kernel (NTK) [17]. Its utility lies in eliminating the necessity of explicit gradient updates for training. Closed-form expressions to determine the NTK corresponding to multi-layer perceptrons (MLPs) [17] and convolutional neural networks (CNNs) [18] have been determined. More recently, these methods have been extended to GNNs facilitating their fusion with the more classical graph kernel approach by deriving a specialized kernel, called Graph Neural Tangent Kernel (GNTK) [19], from an infinitely wide GNN.

The GNTK has been primarily applied to determine similarities between graphs, which are then used for graph classification tasks [19]. In this work, we focus on a different capability of the GNTK and leverage it to find similarities among nodes in a graph, which can be used for node classification. Specifically, our approach caters to a purely inductive setting [8] wherein the node labels of a previously-unseen and completely-unlabeled target graph are to be predicted based on a set of training graphs with fully-labeled nodes. This is more challenging than the typical transductive setting wherein the target graph is available during training, albeit without any labels [7]. A formulation of this form is useful for ego-networks [20] and wireless networks [21] among several other scenarios, where the aim is to extract representations of a new instance based on a labeled set of identically sampled instances. Further, inspired by the general observation that residual connections tend to improve overall performance of GNNs [22], we have established a framework to extract GNTKs corresponding to GNNs with residual connections. Our empirical analysis suggests that there is a clear performance gain due to the proposed augmentation.

Contributions. The contributions of this paper are threefold:
1) We provide a closed-form recurrent expression for the computation of a GNTK associated with a GNN with residual connections.
2) We incorporate the derived GNTK into a node classification pipeline to solve the inductive problem of estimating the labels of a completely unlabeled graph.
3) Through numerical experiments, we illustrate the benefit of incorporating residual connections into the GNTK and compare the performance of the proposed node classification pipeline against GNN baselines.

2. PRELIMINARIES AND PROBLEM FORMULATION

In Section 2.1 we introduce the concepts of graphs and GNNs and in Section 2.2 we present NTKs. We provide a precise formulation of our problem of interest in Section 2.3.

Emails: {ab116, ac131, segarra}@rice.edu.

978-1-6654-0540-9/22/$31.00 ©2022 IEEE

2.1. Graphs and graph neural networks

Let $G = (V, E)$ be an undirected and unweighted graph where V is the set of nodes with cardinality $|V| = n$, and E is the set of edges such that the unordered pair $(i, j) \in E$ only if there exists an edge between nodes i and j. We adopt the convention where the neighborhood $N(u)$ of a node $u \in V$ includes itself and all the nodes connected to it , i.e., $N(u) = \{u\} \cup \{v \in V | (u, v) \in E\}$. Moreover, we consider the case where nodes in V have associated features and labels. Node features $\mathbf{x}_u \in \mathbb{R}^d$ represent node descriptors that can encode information that goes beyond the graph structure. Also, every node u may be associated with a label or class $y_u \in \mathcal{C}$, where $\mathcal{C}$ is a discrete set of labels. We denote a (featured)[1] labeled graph as the tuple $(G, \mathbf{X}, \mathbf{y})$, where the rows of $\mathbf{X} \in \mathbb{R}^{n \times d}$ correspond to the features of every node and $\mathbf{y} \in \mathcal{C}^n$ collects the node labels. Analogously, a tuple $(G, \mathbf{X})$ represents an unlabeled graph.

GNNs are a general class of neural architectures that leverage the underlying connectivity structure of a graph to learn suitable node-level and graph-level representations for various downstream tasks [5]. A generic GNN architecture $\Phi_G : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times o}$ is a learnable parametric transformation of the feature space of a graph G through a sequence of layers. We denote the output of a GNN as $\Phi_G(\mathbf{X}; \mathbf{W})$, where $\mathbf{W}$ contains the parameters of the network. Each layer of a GNN is composed of two basic operations: *neighborhood aggregation* followed by *feature transformation*. While a multitude of architectures [7–9, 23] has been proposed, each offering distinct structures to one or more of these operations, the core principles for an arbitrary node u at any layer l of an L-layered GNN can be formalized as

$$\mathbf{z}_u^{(l+1)} = \sigma\left(\frac{1}{\sqrt{d_l}} c_u \mathbf{W}^{(l)} \sum_{v \in N(u)} \mathbf{z}_v^{(l)}\right), \tag{1}$$

where $\mathbf{z}_u^{(0)} = \mathbf{x}_u$, $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l+1} \times d_l}$ contains trainable weights such that $\mathbf{W} = \{\mathbf{W}^{(0)}, \ldots, \mathbf{W}^{(L-1)}\}$, and c_u is a normalizing factor for neighborhood density. We adopt the common choice of $c_u = 1/|N(u)|$. Through these layered operations, the final output $\mathbf{z}_u^{(L)} \in \mathbb{R}^o$ fuses the local neighborhood information with the attributes of a given node u to generate rich node-level representations.

2.2. Neural tangent kernels

Consider a general neural network $f(\mathbf{x}; \boldsymbol{\omega})$ where $\mathbf{x}$ is the input and $\boldsymbol{\omega}$ is a vector of weights, which have been trained by minimizing a squared loss. We can then approximate the learned function through a first-order Taylor expansion on the weights as

$$f(\mathbf{x}; \boldsymbol{\omega}) \approx f(\mathbf{x}; \boldsymbol{\omega}_0) + \nabla_{\boldsymbol{\omega}} f(\mathbf{x}; \boldsymbol{\omega}_0)^\top (\boldsymbol{\omega} - \boldsymbol{\omega}_0), \tag{2}$$

where $\boldsymbol{\omega}_0$ denotes the weights at initialization. In general, the approximation in (2) would be extremely crude since the learned weights $\boldsymbol{\omega}$ would differ significantly from $\boldsymbol{\omega}_0$. However, it has been observed in practice and theoretically shown that for severely *over-parametrized* neural networks, the parameters $\boldsymbol{\omega}$ only barely change during training [17]. In such a setting, the approximation in (2) is quite accurate and the neural network is close to a linear function of the weights $\boldsymbol{\omega}$. Hence, minimizing a squared loss reduces to just solving a linear regression. Notice, however, that the approximation in (2) is still non-linear in the input $\mathbf{x}$. In fact, it is linear on a very specific feature transformation given by $\eta(\mathbf{x}) = \nabla_{\boldsymbol{\omega}} f(\mathbf{x}; \boldsymbol{\omega}_0)$. This feature transformation naturally induces a kernel on the input, which is termed the NTK [17]. To be more precise, the NTK entry between two inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by $\nabla_{\boldsymbol{\omega}} f(\mathbf{x}_i; \boldsymbol{\omega}_0)^\top \nabla_{\boldsymbol{\omega}} f(\mathbf{x}_j; \boldsymbol{\omega}_0)$. Reformulating a neural architecture as an NTK generates closed-form expressions to compute its output without performing explicit gradient updates on its parameters. More recently, this idea was extended to GNNs to study their performance in the over-parametrized (infinite-width) limit [19, 24, 25].

2.3. Problem formulation

Our goal is to learn from a set of fully-labeled graphs in order to estimate the labels of a different completely unlabeled graph. Formally, we state our problem as follows.

Problem 1 *Given a set of m labeled graphs $\{(G_i, \mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^m$, provide an estimate $\hat{\mathbf{y}}_0$ of the labels of a given (unlabeled) graph $(G_0, \mathbf{X}_0)$.*

The above problem is of practical relevance when the labeled and unlabeled graphs are expected to share some common traits. For example, we can have a series of social networks with features associated with each person (such as age, nationality, or salary) along with a label of interest such as whether they bought a given product or not. Then, our unlabeled social network $(G_0, \mathbf{X}_0)$ can represent a new untapped market and our goal is to estimate the individuals that might be interested in the product. As another example, consider the case where the labeled graphs represent brain networks from several individuals and the labels indicate whether the brain regions were significantly involved in performing a given task. Then, given a new individual, we want to predict which brain regions will become particularly active when the task is performed.

Both mentioned examples highlight the inductive nature of the problem at hand. Unlike the (more common) problem of semi-supervised learning where we are given a partially labeled graph and we want to propagate those labels to the rest of the graph, in Problem 1 we have no connections between labeled and unlabeled nodes. Instead, our task is to extract a fundamental way to relate the structure and features of a graph with its labels, and then apply these learned relations on a new graph $(G_0, \mathbf{X}_0)$ that is unavailable during training.

At a high level, we tackle Problem 1 by defining a *similarity measure between nodes that potentially belong to different graphs*. Having access to this similarity, we can rely on a kernel-based method to train a node classifier. Then, given a new (unlabeled) graph, we can compute the similarities between its nodes and those in our training set, and apply the trained classifier to estimate its labels. For the essential step of defining the aforementioned similarity, we rely on infinitely-wide GNNs with residual connections, as explained in Section 3.

3. NODE CLASSIFICATION USING GRAPH NEURAL TANGENT KERNELS

One could define a similarity measure between nodes based entirely on their features by, e.g., applying a decreasing function to any metric in the feature space. In this way, two nodes that have feature vectors lying close to each other in feature space will have a high similarity value. An immediate drawback of this approach is that it completely ignores the topological information of the graphs. Alternatively, we could also define an entirely topological notion of similarity by, e.g., comparing the centrality values of the nodes. In

[1] By default, all considered graphs will have features.

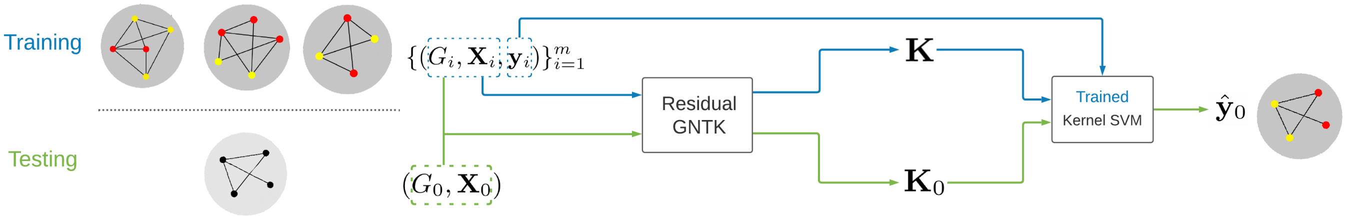


Fig. 1. Scheme of our proposed pipeline. Following Proposition 1 we compute the GNTK with residual connections $\mathbf{K}$ between all the nodes in the training set. We use $\mathbf{K}$ along with the given labels to train a kernel SVM classifier. Given a new unlabeled graph $(G_0, \mathbf{X}_0)$ we first compute the GNTK entries between the nodes in the new graph and those in all the training graphs to obtain $\mathbf{K}_0$. Finally, we rely on our kernel SVM classifier to estimate the labels $\hat{\mathbf{y}}_0$.

this way, two central nodes, even if belonging to different graphs, will be deemed similar to each other.

With the objective of finding a good solution to Problem 1, we rely on GNTKs to define a similarity (kernel) between nodes that combines both sources of information (features and topology).

3.1. GNTKs with residual connections

The incorporation of residual or skip connections [26] in neural architectures is a general method for creating additional routes for forward propagation of hidden features and backward propagation of gradients. The main advantages of this technique include reduction in model complexity by conferring the model the flexibility needed to skip unnecessary layers and allowance for improved gradient flow to lower layers [27]. In addition to other classes of connectionist architectures, residual connections have also been shown to significantly improve convergence behavior of GNNs through implicit acceleration [22]. In this context, it is of interest to establish the expression of the GNTK that corresponds to an infinitely-wide GNN *with residual connections*.

To be more precise, in a generic GNN architecture with 1-skip connections, we replace the layer update in (1) with

$$\mathbf{z}_u^{(l+1)} = \sigma\left(\frac{1}{\sqrt{d_l}} c_u \mathbf{W}_1^{(l)} \sum_{v \in N(u)} \mathbf{z}_v^{(l)} + \frac{1}{\sqrt{d_l}} \mathbf{W}_2^{(l)} \mathbf{z}_u^{(l)}\right), \tag{3}$$

where we have now added a second term representing the residual connection with its own set of learnable parameters $\mathbf{W}_2^{(l)} \in \mathbb{R}^{d_{l+1} \times d_l}$.

As explained in Section 2.2, our goal is to extract the kernel induced by a GNN's severely over-parametrized layers with the form in (3). In essence, given any two graphs $G = (V, E)$ and $G' = (V', E')$ with n and n' nodes, respectively, we seek to recover a similarity matrix $\mathbf{\Theta} \in \mathbb{R}^{n \times n'}$ such that $[\mathbf{\Theta}]_{uu'}$ captures the similarity between nodes $u \in V$ and $u' \in V'$. Following the now-established procedure for the computation of NTKs [17, 18], we provide a recurrent formula for the computation of $\mathbf{\Theta}$, where we propagate the similarities through the L layers of our infinitely-wide GNN. We denote by $\mathbf{\Theta}^{(l)}$ the similarity matrix after l layers of our GNN and define $\mathbf{\Theta} = \sum_{l=1}^{L} \mathbf{\Theta}^{(l)}$. This modality where the similarities are aggregated across layers is sometimes referred to as the jumping knowledge variant [19]. Alternatively, one can define $\mathbf{\Theta} = \mathbf{\Theta}^{(L)}$. However, due to its better empirical performance, we adopt the jumping knowledge modality. Moreover, the computation of $\mathbf{\Theta}^{(l)}$ relies on another series of matrices $\mathbf{\Sigma}^{(l)}$ that are also recursively updated. In their entries $\left[\mathbf{\Sigma}^{(l)}\right]_{uu'}$, these matrices encode the covariance between Gaussian processes $\Psi_l(u)$ and $\Psi_l(u')$, which are instrumental for the derivation of the GNTK and arise when considering the infinitely wide setting, i.e., $d_1, \ldots, d_{L-1} \to \infty$. Having introduced this notation, the following result holds.

Proposition 1 *Given two graphs G and G', the GNTK $\mathbf{\Theta} = \sum_{l=1}^{L} \mathbf{\Theta}^{(l)}$ of the residual GNN with layers as described in* (3) *is given (elementwise) by the following recursive procedure*

$$\begin{aligned}\left[\mathbf{\Theta}^{(l+1)}\right]_{uu'} = \left[\mathbf{\Sigma}^{(l+1)}\right]_{uu'} + \left[\mathbf{\Theta}^{(l)}\right]_{uu'} \overline{\Psi}_l(u, u'; \dot{\sigma}) \\ + c_u c_{u'} \sum_{v \in N(u)} \sum_{v' \in N(u')} \left(\left[\mathbf{\Theta}^{(l)}\right]_{vv'} \overline{\Psi}_l(v, v'; \dot{\sigma})\right),\end{aligned} \tag{4}$$

where

$$\overline{\Psi}_l(u, u'; \dot{\sigma}) = \mathbb{E}_{\Psi_l \sim \mathcal{N}(0, \mathbf{\Sigma}^{(l)})}\left[\dot{\sigma}(\Psi_l(u))\dot{\sigma}(\Psi_l(u'))\right], \tag{5}$$

$$\left[\mathbf{\Theta}^{(1)}\right]_{uu'} = \left[\mathbf{\Sigma}^{(1)}\right]_{uu'}, \tag{6}$$

$$\left[\mathbf{\Sigma}^{(1)}\right]_{uu'} = \frac{1}{d}\mathbf{x}_u^\top \mathbf{x}_{u'} + \frac{1}{d} c_u c_{u'} \sum_{\substack{v \in N(u) \\ v' \in N(u')}} \mathbf{x}_v^\top \mathbf{x}_{v'}, \tag{7}$$

$$\left[\mathbf{\Sigma}^{(l+1)}\right]_{uu'} = \overline{\Psi}_l(u, u'; \sigma) + c_u c_{u'} \sum_{\substack{v \in N(u) \\ v' \in N(u')}} \overline{\Psi}_l(v, v'; \sigma), \tag{8}$$

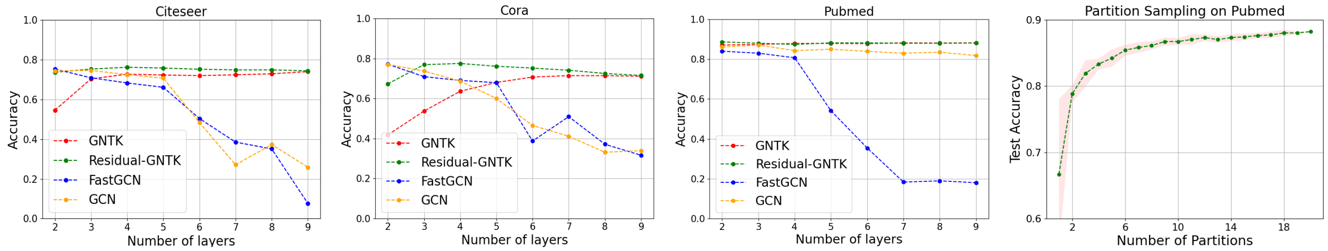
and $\dot{\sigma}$ denotes the derivative of σ.

Proof: The above result can be proven by induction similar to the proof derived in [17, Prop. 1] for multi-layer perceptrons (MLPs). Essentially, we can consider the propagation rule in (3) as a combination of an MLP given by $\mathbf{W}_2^{(l)}$ and the aggregation term corresponding to a standard GNN. The key difference between our approach and that of [19] is that by considering the residual layer as an MLP, the expression for the covariance $\left[\mathbf{\Sigma}^{(l+1)}\right]_{uu'}$ in [19] is modified as in (8). Once that relation has been established, we can follow analogous steps to the derivation of $\left[\mathbf{\Theta}^{(l+1)}\right]_{uu'}$ in [17] to obtain the expression in Proposition 1. ■

The key component of the above result is its treatment of the residual connection as an MLP that is added to to the aggregation structure of a GNN. This allows us to establish an explicit formula to capture the layer-wise evolution of the residual GNTK. A node-level similarity structure of this form allows us to learn suitable node representations based on corresponding node features of multiple graphs.

3.2. Label propagation across graphs

Having established a relevant similarity measure between nodes that potentially reside on different graphs (cf. Proposition 1), we now



(a) (b) (c) (d)

Fig. 2. Node classification on citation networks. (a-c) Performance comparison in terms of test accuracy of GCN, FastGCN, vanilla GNTK, and Residual-GNTK on three benchmarked datasets: (a) Citeseer, (b) Cora, and (c) Pubmed. (d) Test accuracy of Residual-GNTK on Pubmed as a function of the number of randomly selected training graphs.

detail our node classification method by subsequently going over its training and testing phases.

Training. Given our training graphs $\{(G_i, \mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^m$ we compute the residual GNTK matrices $\mathbf{\Theta}(G_i, G_j)$ following (4)-(8) for every pair of graphs, where we now explicitly state in the GNTK notation which graphs are involved in its computation. We can then collect these $\mathbf{\Theta}(G_i, G_j)$ into a kernel matrix $\mathbf{K} \in \mathbb{R}^{\sum_{i=1}^m n_i \times \sum_{i=1}^m n_i}$, given as follows

$$\mathbf{K} = \begin{bmatrix} \mathbf{\Theta}(G_1, G_1) & \cdots & \mathbf{\Theta}(G_1, G_m) \\ \vdots & \ddots & \vdots \\ \mathbf{\Theta}(G_m, G_1) & \cdots & \mathbf{\Theta}(G_m, G_m) \end{bmatrix}. \tag{9}$$

Notice that $\mathbf{K}$ stores a similarity measure between every pair of nodes in our training set. Leveraging the fact that we have labels $\{\mathbf{y}_i\}_{i=1}^m$ for all these nodes, we can train any kernel-based classifier on these data such as a kernel SVM or a kernel logistic regression [28]. Due to their widespread adoption, we select kernel SVMs in our implementation. We do not provide a detailed description of how kernel SVMs work since this is not the focus of the paper. However, we direct the interested reader to [29]. The described training procedure is depicted in Fig. 1 using blue arrows.

Testing. When presented with the new unlabeled graph $(G_0, \mathbf{X}_0)$, we can compute the GNTK similarities between every node in G_0 and all the nodes in our training dataset and store them in $\mathbf{K}_0 \in \mathbb{R}^{n_0 \times \sum_{i=1}^m n_i}$ as

$$\mathbf{K}_0 = \begin{bmatrix} \mathbf{\Theta}(G_0, G_1) & \cdots & \mathbf{\Theta}(G_0, G_m) \end{bmatrix}. \tag{10}$$

Due to the nature of kernel methods [30], the information in $\mathbf{K}_0$ is enough to obtain label estimates $\hat{\mathbf{y}}_0$ from our previously trained kernel SVM. This testing pipeline is represented using green arrows in Fig. 1 and we can see its implementation in the next section.

4. NUMERICAL EXPERIMENTS

We empirically demonstrate the performance of our GNTK with residual connections (Residual-GNTK) for node classification on citation networks. For our experiments, we consider a purely inductive setting – test graph is completely isolated from the training graph. A description of the datasets is provided next, followed by a comparison of our model performance with that of GCN [7] and FastGCN [9], in terms of classification accuracy. Finally we discuss the effect of partitioning a large training graph into multiple sub-graphs of amenable sizes and using only a subset of these smaller graphs for training a classifier.

Datasets. We use three benchmarked public citation networks of different sizes: Cora, Citeseer, and Pubmed [7]. While Cora and Citeseer are of amenable size ($<$ 5k nodes), Pubmed has about 19k nodes, thus, we rely on METIS [31] to partition this graph into smaller subgraphs. In particular, we obtain 20 subgraphs of equal size following the data splits in FastGCN [9] with the modification that we remove all the edges connecting the test graphs to the training and validation graphs for each dataset to perfectly emulate an inductive setting.

Performance comparison. We compare the node classification accuracy achieved by Residual-GNTK with that of vanilla GNTK and two existing solution models, FastGCN [9] and GCN [7]. For GNTK and Residual-GNTK, we train a support vector machine classifier (SVM-C) with $\mathbf{K}$ as the precomputed kernel, and test on the completely unseen graph using $\mathbf{K}_0$. The comparisons are shown in Fig 2. While there is a clear dip in performance of GCN and FastGCN for all three datasets as evident in Fig 2(a), 2(b) and 2(c) indicating oversmoothing, the GNTK models maintain a consistent performance even for deeper architectures. Moreover, Residual-GNTK shows a consistent improvement in accuracy over that of vanilla GNTK, clearly translating the effectiveness of residual connections to the kernel formulation.

Scalability. We now explore the possibility of using a small subset of partitions to train Residual-GNTK, especially for large graphs. To this end, we randomly select $m \in \{1, 2, \ldots, 20\}$ partitions of the training graph of Pubmed to train an SVM-C using a Residual-GNTK of 5 layers. We present the mean and standard deviation of 10 trials for each m in Fig 2(d). We can observe a 10-fold decrease in the standard deviation of classification accuracy when 2 partitions are used instead of 1. Furthermore, the steep increase of accuracy as we move from 1 to 10 sub-graphs suggests that it is possible to achieve sufficiently high accuracy even by using just about half of the training graphs, sampled randomly.

5. CONCLUSIONS

We have presented a method to estimate all the labels of an entirely unlabeled graph. The method relies on calculating similarities between the unlabeled nodes and the labeled training nodes belonging to different graphs. The similarities are computed based on a proposed variation of the GNTK, which are in turn fed to a kernel based classifier. Future research avenues include: i) The derivation and implementation of other GNTKs based on variants of the underlying GNN, and ii) The design of scalable and parallelizable pipelines that rely on ensemble classifiers for cases where the number of training graphs m is large.

6. REFERENCES

[1] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, 2013.

[2] H Howie Huang and Hang Liu, “Big data machine learning and graph analytics: Current state and future challenges,” in *IEEE Intl. Conf. on Big Data*, 2014, pp. 16–17.

[3] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst, “Graph signal processing: Overview, challenges, and applications,” *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[4] Gonzalo Mateos, Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro, “Connecting the dots: Identifying network structure via graph signal processing,” *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 16–43, 2019.

[5] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, “A comprehensive survey on graph neural networks,” *IEEE Trans. Neural Netw. and Learn. Sys.*, vol. 32, no. 1, pp. 4–24, 2020.

[6] Ziwei Zhang, Peng Cui, and Wenwu Zhu, “Deep learning on graphs: A survey,” *IEEE Trans. Knowledge and Data Eng.*, 2020.

[7] Thomas N Kipf and Max Welling, “Semi-supervised classification with graph convolutional networks,” in *Intl. Conf. Learn. Repres. (ICLR)*, 2017.

[8] William L Hamilton, Rex Ying, and Jure Leskovec, “Inductive representation learning on large graphs,” in *Adv. Neural Info. Process. Syst. (NeurIPS)*, 2017, pp. 1025–1035.

[9] Jie Chen, Tengfei Ma, and Cao Xiao, “FastGCN: Fast learning with graph convolutional networks via importance sampling,” *arXiv preprint arXiv:1801.10247*, 2018.

[10] Lei Wang, Qiang Yin, Chao Tian, Jianbang Yang, Rong Chen, Wenyuan Yu, Zihang Yao, and Jingren Zhou, “FlexGraph: A flexible and efficient distributed framework for GNN training,” in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 67–82.

[11] Cameron R Wolfe, Jingkang Yang, Arindam Chowdhury, Chen Dun, Artun Bayer, Santiago Segarra, and Anastasios Kyrillidis, “GIST: Distributed training for large-scale graph convolutional networks,” *arXiv preprint arXiv:2102.10424*, 2021.

[12] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo, “Graph representation learning: A survey,” *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[13] Fenyu Hu, Yanqiao Zhu, Shu Wu, Liang Wang, and Tieniu Tan, “Hierarchical graph convolutional networks for semi-supervised node classification,” *arXiv preprint arXiv:1902.06667*, 2019.

[14] Kenta Oono and Taiji Suzuki, “Graph neural networks exponentially lose expressive power for node classification,” *arXiv preprint arXiv:1905.10947*, 2019.

[15] Sebastian Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[16] Léon Bottou, “Stochastic gradient descent tricks,” in *Neural networks: Tricks of the trade*, pp. 421–436. Springer, 2012.

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

[18] 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*, 2019.

[19] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu, “Graph neural tangent kernel: Fusing graph neural networks with graph kernels,” *Adv. Neural Info. Process. Syst. (NeurIPS)*, vol. 32, pp. 5723–5733, 2019.

[20] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec, “Identity-aware graph neural networks,” *arXiv preprint arXiv:2101.10320*, 2021.

[21] Arindam Chowdhury, Gunjan Verma, Chirag Rao, Ananthram Swami, and Santiago Segarra, “Unfolding WMMSE using graph neural networks for efficient power allocation,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 6004–6017, 2021.

[22] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi, “Optimization of graph neural networks: Implicit acceleration by skip connections and more depth,” *arXiv preprint arXiv:2105.04550*, 2021.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[24] Ting Chen, Song Bian, and Yizhou Sun, “Are powerful graph neural nets necessary? a dissection on graph classification,” *arXiv preprint arXiv:1905.04579*, 2019.

[25] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola, “Generalization and representational limits of graph neural networks,” in *Intl. Conf. Mach. Learn. (ICML)*. PMLR, 2020, pp. 3419–3430.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” in *IEEE Conf. Comp. Vision and Pattern Recog. (CVPR)*, 2016, pp. 770–778.

[27] Andreas Veit, Michael J Wilber, and Serge Belongie, “Residual networks behave like ensembles of relatively shallow networks,” *Adv. Neural Info. Process. Syst. (NeurIPS)*, vol. 29, pp. 550–558, 2016.

[28] Ji Zhu and Trevor Hastie, “Support vector machines, kernel logistic regression and boosting,” in *International Workshop on Multiple Classifier Systems*. Springer, 2002, pp. 16–26.

[29] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An Introduction to Statistical Learning*, vol. 112, Springer, 2013.

[30] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola, “Kernel methods in machine learning,” *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[31] George Karypis and Vipin Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. on Scientific Computing*, 1998.