Oversmoothing in Graph Neural Networks

Joseph Foster
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93117
josephfoster@ucsb.edu

Sidharth Kannan
College of Creative Studies
University of California, Santa Barbara
Santa Barbara, CA 93117
skannan@ucsb.edu

Abstract

Depth has been a critical factor in the success of modern neural networks. In the graph learning domain, however, it has been observed that increasing depth leads to significant performance degradations, due to a phenomenon termed "oversmoothing." In this project, we use graph neural tangent kernels (GNTKs) to study the effects of proposed strategies to mitigate oversmoothing. In particular, we study the effects of jumping knowledge, skip connections, and the simple spectral convolution proposed by Zhu and Koniusz [2021]. Finally, we develop an effective field theory of graph convolutional networks (GCNs) at initialization, and use it to demonstrate the cause of oversmoothing in GCNs.

1 Introduction

One of the great successes of modern machine learning is the excellent performance of deep neural networks in a wide range of tasks like computer vision, natural language processing, and speech recognition. These networks often see performance continue to improve as their depth increases to dozens or even hundreds of layers. However, this is not the case in the graph domain. Deep graph neural networks suffer from oversmoothing, in which features become homogeneous across the graph, leading to an inability to distinguish differences between nodes and degraded performance.

Oversmoothing is caused by repeated message passing layers in which node features are mixed with their neighbors. The receptive field of the network grows with depth, allowing it to incorporate messages from further away in the graph, but simultaneously leads to features across the graph looking increasingly homogeneous. This washes out important structural information about the graph, and results in degrading performance.

Many papers have examined the issue of oversmoothing in graphs both theoretically and experimentally. Oono and Suzuki [2021] shows that under certain theoretical conditions, GCNs suffer from information loss in the limit of infinite layers. The authors then use their theoretical results to propose a form of weight normalization to mitigate oversmoothing, which they test experimentally. Cai and Wang [2020] builds on this work and shows that the Dirichlet energy of embeddings will converge to zero as layers increase under a certain broader set of conditions compared to the previous paper, leading to a loss of discriminative power. Sabanayagam et al. [2021] uses graph neural tangent kernels to study the effect of oversmoothing, and proposes using max normalization and skip connections to fix the issue.

Being able to train deeper GNNs without oversmoothing would enable GNNs to capture richer structural information about the graph, and pass messages from far away nodes in order to inform downstream tasks. This has motivated a great deal of research into ways to mitigate oversmoothing. In this report, we examine three proposed strategies: jumping knowledge, skip connections, and the simple spectral convolution. In order to efficiently analyze these experimentally, we make use of

Preprint. Under review.

graph neural tangent kernels, which have been shown by Sabanayagam et al. [2021] to capture an ideal case of the performance of their respective networks.

Finally, based on methods pioneered by Roberts et al. [2022], we explicitly compute a recursion for the distribution of network outputs at initialization as a function of depth in the infinite-width limit, and show that the distribution provably collapses.

2 Background and Related Work

2.1 Graph Neural Networks and Node Classification

In this project, we focus on the node classification problem in the *transductive* setting. Given a single graph G, a set of features $\mathbf{x} \in \mathbb{R}^{n_0}$ on each node, and a label y for some subset of nodes, we wish to predict the labels of the unlabeled nodes.

In particular, GNNs attempt to tackle this problem by repeatedly aggregating and then transforming these feature vectors, using a sequence of learned transformations, to produce a final node embedding which should encode the class information of the node. Each layer of a vanilla GCN is a transformation of the form

$$\mathbf{X}^{(l+1)} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{W}^{(l)} \tag{1}$$

where $\mathbf{D}$ is the degree matrix of the graph, $\mathbf{A}$ is its adjacency matrix (with self loops), $\mathbf{X}^{(k)}$ represents the node embedding in the k-th layer, and $\mathbf{W}$ is learned linear transformation.

2.2 Simple Spectral Graph Convolution

The simple spectral graph convolution is a proposed improvement to the vanilla graph convolutional network, that widens the receptive field of the GNN, while mitigating oversmoothing. The SSGC convolution uses a modified diffusion kernel, of the form

$$\tilde{\mathbf{T}} = \frac{1}{K}\sum_{k=0}^{K}(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^k \tag{2}$$

instead of just using the adjacency matrix, and uses a single layer instead of stacking multiple layers.

2.3 Neural Tangent Kernels and GNTKs

Consider a neural network, $f(x, \theta)$, being trained under gradient descent with an infinitesimally small learning rate. The evolution of the network outputs, $u(t)$, over the course of training is given by

$$\frac{du}{dt} = \mathbf{H}(u(t) - y) \tag{3}$$

where the matrix $\mathbf{H}$ is given by

$$\mathbf{H}_{ij} = \left\langle \frac{df(x_i, \theta(t))}{d\theta}, \frac{df(x_j, \theta(t))}{d\theta} \right\rangle \tag{4}$$

Jacot et al. [2018] show that in the infinite-width limit of a randomly initialized neural network, $\mathbf{H}$ will reduce to a deterministic kernel function, called the Neural Tangent Kernel (NTK).

The NTK is dependent only on the architecture of the network at hand, and the dataset it is trained on, and thus requires no training hyperparameters. Furthermore, in the limit of infinite width, the NTK does not change over the course of training. That is, the NTK at initialization will be the same as the NTK of the fullly trained network.

Du et al. [2019] derive a recursion relation for the NTK corresponding to a graph neural network performing graph classification, and Sabanayagam et al. [2021] extend this to the transductive node classification setting. Following these two works, we compute GNTKs for node classification using the following set of equations:

For each pair $u, u' \in V$

$$\left[\boldsymbol{\Sigma}_{(0)}^{(\ell)}(G)\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}(u)\cup\{u\}} \sum_{v' \in \mathcal{N}(u')\cup\{u'\}} \left[\boldsymbol{\Sigma}_{(R)}^{(\ell-1)}(G)\right]_{vv'} \tag{5}$$

$$\left[\boldsymbol{\Theta}_{(0)}^{(\ell)}(G)\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \left[\boldsymbol{\Theta}_{(R)}^{(\ell-1)}(G)\right]_{vv'} \tag{6}$$

$$\left[\mathbf{A}_{(r)}^{(\ell)}(G)\right]_{uu'} = \begin{pmatrix} \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G)\right]_{u,u} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G)\right]_{uu'} \\ \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G)\right]_{u'u} & \left[\boldsymbol{\Sigma}_{(r-1)}^{(\ell)}(G)\right]_{u'u'} \end{pmatrix} \in \mathbb{R}^{2\times 2} \tag{7}$$

$$\left[\boldsymbol{\Sigma}_{(r)}^{(\ell)}(G)\right]_{uu'} = c_\sigma \mathbb{E}_{(a,b)\sim\mathcal{N}\left(\mathbf{0},\left[\mathbf{A}_{(r)}^{(\ell)}(G)\right]_{uu'}\right)}[\sigma(a)\sigma(b)] \tag{8}$$

$$\left[\dot{\boldsymbol{\Sigma}}_{(r)}^{(\ell)}(G)\right]_{uu'} = c_\sigma \mathbb{E}_{(a,b)\sim\mathcal{N}\left(\mathbf{0},\left[\mathbf{A}_{(r)}^{(\ell)}(G)\right]_{uu'}\right)}[\dot{\sigma}(a)\dot{\sigma}(b)] \tag{9}$$

$$\left[\boldsymbol{\Theta}_{(r)}^{(\ell)}(G)\right]_{uu'} = \left[\boldsymbol{\Theta}_{(r-1)}^{(\ell)}(G)\right]_{uu'} \left[\dot{\boldsymbol{\Sigma}}_{(r)}^{(\ell)}(G)\right]_{uu'} + \left[\boldsymbol{\Sigma}_{(r)}^{(\ell)}(G)\right]_{uu'} \tag{10}$$

2.4 Jumping Knowledge

Xu et al. [2018] introduces jumping knowledge as a way to combat oversmoothing in graph neural networks. In a jumping knowledge graph neural network, the hidden representations from each layer, $\mathbf{X}^{(l)}$, are all aggregated together in the final layer of the network. Multiple aggregation functions can be considered, here we use summation following Du et al. [2019]. The equation for the final representation is then:

$$\mathbf{X}_G = \sum_{l=0}^{L} \mathbf{X}^{(l)} \tag{11}$$

Jumping knowledge prevents oversmoothing by preserving earlier hidden representations. These earlier hidden representations contain information about graph structures of different sizes. Without jumping knowledge, only the final layer representing the largest graph structures is used, and information about smaller graph structures is lost.

Jumping knowledge is closely connected to the simple spectral graph convolution as they both take an aggregation over neighborhood sizes. The simple spectral graph convolution is essentially a linearized and simplified version of a GCN with jumping knowledge.

A GNTK can also be derived for a graph neural network using jumping knowledge. Du et al. [2019] derives such a GNTK for graph level tasks by modifying the readout operation to sum over all intermediate outputs, before aggregating over all node pairs. To adapt this for node level tasks, we simply remove the final aggregation over node pairs, resulting in the following readout operation:

$$\Theta(G) = \sum_{l=0}^{L} \Theta_{(R)}^{(l)}(G) \tag{12}$$

2.5 Skip Connection

Kipf and Welling [2017] proposed skip connections as another method to reduce oversmoothing. This skip connection takes a transformed input and adds it to each hidden layer before applying the diffusion. Reintroducing the input features to each hidden layer in this way ensures that they never get smoothed out, and allows information of small graph structures to be preserved. Following Sabanayagam et al. [2021], we consider a transformed input H_0, which is transformed to have dimensions matching the hidden layer size. Then, the equation for each layer of a Skip-PC (pre-convolution) network is:

$$\mathbf{X}^{(l+1)} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}(\mathbf{X}^{(l)} + H_0)\mathbf{W}^{(l)} \tag{13}$$

Now, Sabanayagam et al. [2021] shows that a GNTK can be derived for networks containing skip connections. We modify equation (5) to add in the skip connection:

For each pair $u, u' \in V$ and $l > 1$ (for $l = 1$ equation (5) is used unchanged)

$$\left[\boldsymbol{\Sigma}_{(0)}^{(\ell)}(G)\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \sum_{v' \in \mathcal{N}(u') \cup \{u'\}} \left(\left[\boldsymbol{\Sigma}_{(R)}^{(\ell-1)}(G)\right]_{vv'} + \left[\boldsymbol{\Sigma}_{(R)}^{(0)}(G)\right]_{vv'}\right) \tag{14}$$

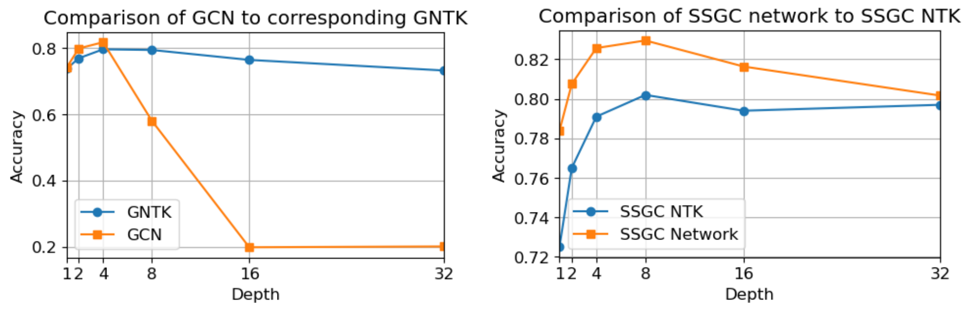

Figure 1: Comparison of NTKs to corresponding finite width networks on Cora

3 Simple Spectral Neural Tangent Kernels

We derive a neural tangent kernel for SSGC networks following the set of equations presented by Du et al. [2019]. The key difference of the SSGC is the modified diffusion kernel, and so the only GNTK equations which need to be modified are those governing neighborhood aggregation. Using the modified diffusion kernel $\tilde{\mathbf{T}}$ from section 2.2, we define $\mathcal{N}_{\tilde{T}}(u)$ to be the neighborhood of u in the diffusion kernel $\tilde{\mathbf{T}}$. We modify equations (5) and (6) as follows:

For each pair $u, u' \in V$

$$\left[\mathbf{\Sigma}^{(\ell)}_{(0)}(G)\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}_{\tilde{T}}(u)} \sum_{v' \in \mathcal{N}_{\tilde{T}}(u')} \tilde{\mathbf{T}}_{vv'} \left[\mathbf{\Sigma}^{(\ell-1)}_{(R)}(G)\right]_{vv'} \tag{15}$$

$$\left[\mathbf{\Theta}^{(\ell)}_{(0)}(G)\right]_{uu'} = c_u c_{u'} \sum_{v \in \mathcal{N}_{\tilde{T}}(u)} \sum_{v' \in \mathcal{N}_{\tilde{T}}(u')} \tilde{\mathbf{T}}_{vv'} \left[\mathbf{\Theta}^{(\ell-1)}_{(R)}(G)\right]_{vv'} \tag{16}$$

With these modifications, the rest of the equations remain the same as for the GNTK.

4 Results

We first compare the GNTK and SSGC NTK to corresponding finite width GCN and SSGC networks, with the goal to determine whether the NTKs suffer from oversmoothing with depth in a similar way to the finite width networks. All models are tested at depths of 1, 2, 4, 8, 16 and 32.

Results of this comparison are shown in Figure 1. The SSGC NTK tracks the performance of the SSGC network fairly closely, with both gradually degrading in performance when depth increases beyond 8. The GNTK however performs much better than the GCN for deeper depths, although it does still degrade. Sabanayagam et al. [2021] proposes that the GNTK captures the "best" performance of its corresponding GCN; perhaps with changes to normalization or training hyperparameters the GCN would perform more similarly. Notably, both the GNTK and GCN do achieve their best performances at the same depth, 4, so while the GNTK may suffer from oversmoothing less, it still captures the overall trend of the GCN.

Next, we perform an experimental comparison of the NTKs on the Cora and Citeseer datasets, using the public splits. We compare the basic GNTK, the GNTK with jumping knowledge, GNTK with skip connections, and SSGC GNTK. In order to determine the impact of oversmoothing on each NTK, we test each of them at depths 1, 2, 4, 8, 16 and 32. Our results for shown in Figure 2.

From these results, we conclude that both SSGC and jumping knowledge are effective in reducing oversmoothing, with improved performance at large depths over the base GNTK. Skip connections however do not seem to help, in fact they cause performance to degrade much more with depth in comparison to the base GNTK. It is unclear why this is, Sabanayagam et al. [2021] also evaluates the performance of GNTKS with this same type of skip connection, however they find that the skip connection improves accuracy at high depths. There are some differences in our experimental setup, Sabanayagam et al. [2021] converts Cora and Citeseer into a binary classification problem, while we are solving the standard multi-class classification problem for the datasets. It could also be some

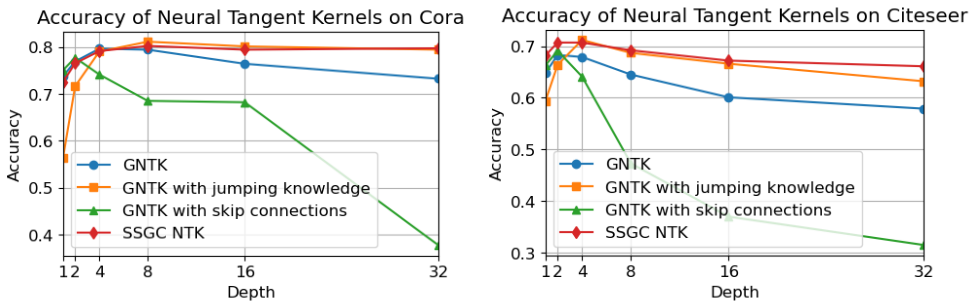

Figure 2: Accuracy of kernel regression relative to depth on Cora and Citeseer

difference in implementation causing this discrepancy, however Sabanayagam et al. [2021] does not have any publicly available code for their work, so we are unable to check this.

5 Effective Field Theory of GCNs

In this section we will extend the formalism presented by Roberts et al. [2022] for multilayer perceptrons, and so we adopt their terminology and notation. We will continue to study the node classification problem in the transductive setting. Given a graph $G = (V, E)$, we will use α to number the node indices, and i to number the neuron indices. $\mathbf{h}^{(l)}$ will be used to denote the result of an aggregation at the l-th layer, $\mathbf{z}^{(l)}$ will be used to denote the preactivations, and $\sigma(\mathbf{z})$ will denote the postactivations.

At initialization, the network parameters are each drawn from i.i.d. Gaussians, with variance for the weights at layer l denoted C_W, and variance for the biases denoted C_b. Finally, n_l will denote the width of layer l. We will attempt to come up with a closed form expression for the *distribution* of network outputs at each layer, conditioned on a dataset $\mathcal{D}$, which we will denote $\mathcal{P}(\mathbf{z}^{(l)}|\mathcal{D})$. The goal is explicitly demonstrate the oversmoothing phenomenon and its relation to the network depth.

The forward iteration equations for a vanilla GCN is given in Eq. 1. As demonstrated in Roberts et al. [2022], the preactivation distribution for a single MLP layer is a mean-zero Gaussian, with covariance

$$G^{(1)}_{\alpha_1\alpha_2} = C_b + \frac{C_W}{n_0}\mathbf{x}^T_{\alpha_1}\mathbf{x}_{\alpha_2} \tag{17}$$

or, in matrix form

$$\mathbf{G}^{(1)} = C_b + \frac{C_W}{n_0}\mathbf{X}^T\mathbf{X} \tag{18}$$

here, we use α_i as our sample or node indices, in accordance with the notation in Roberts et al. [2022]. Upon aggregation, these feature vectors are redistributed according to a sum of Gaussians. In particular, since the feature aggregation occurs before the linear transformation, for a GCN

$$\mathbf{G}^{(1)} = C_b + \frac{C_W}{n_0}\tilde{\mathbf{A}}\mathbf{X}^T\mathbf{X}\tilde{\mathbf{A}}^T \tag{19}$$

where $\tilde{\mathbf{A}}$ is the degree-normalized adjacency matrix of the input graph. We then repeat this process to get the distribution of the second layer.

$$\mathcal{P}(\mathbf{z}^{(2)}|\mathcal{D}) = \frac{1}{Z}\exp\left(-S(z)\right) \tag{20}$$

The action, $S(z)$, is given by

$$S^{(2)}(\mathbf{z}) = \frac{1}{2}\mathbf{z}^{(2)^T}\left(\mathbf{G}^{(2)}\right)^{-1}\mathbf{z}^{(2)} - \frac{1}{8n_1}\mathbf{z}^{(2)^T}\mathbf{z}^{(2)^T}\left(\mathbf{V}^{(2)}\right)^{-1}\mathbf{z}^{(2)}\mathbf{z}^{(2)} \tag{21}$$

This is, as expected, a slightly more complicated distribution than the first layer, as we now have a second (non-Gaussian) term in the distribution. The second term $\mathbf{V}$ is a four-dimensional tensor,

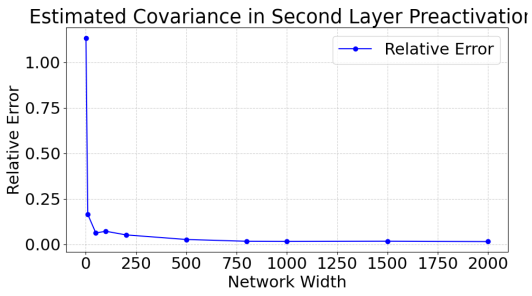

Figure 3: Relative difference between analytically calculated covariance and covariance estimated by sampling hidden width $\times$ 10 randomly initialized GNNs with $C_b = 0$, $C_W = 1$. Error quickly drops to 0, indicating that finite width corrections are minimal at layer 2.

called the four-point vertex, which marks the break from a purely Gaussian distribution. However, this contribution is suppressed by $\frac{1}{n_1}$, and so in this limit, it vanishes.

The covariance is given by

$$\mathbf{G}^{(2)} = C_b^{(2)} + C_W^{(2)} \tilde{\mathbf{A}} \mathbf{E}^{(1)} \tilde{\mathbf{A}}^T \tag{22}$$

where the matrix $\mathbf{E}$ is the two point correlator, or cumulant,

$$\mathbf{E}_{ij}^{(1)} = \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\mathbf{G}^{(1)})} \left[\sigma(u_i)\sigma(v_j)\right] \tag{23}$$

This expectation evaluates to

$$\mathbb{E}_{(u,v)\sim\mathcal{N}(0,\mathbf{G}^{(1)})} \left[\sigma(u_i)\sigma(v_j)\right] = \frac{\mathbf{G}_{ij}^{(1)} \left(\pi - \cos^{-1}\left(\frac{\mathbf{G}_{ij}^{(1)}}{\sqrt{\mathbf{G}_{ii}^{(1)}\mathbf{G}_{jj}^{(1)}}}\right)\right) + \sqrt{1 - \frac{\mathbf{G}_{ij}^{(1)2}}{\mathbf{G}_{ii}^{(1)}\mathbf{G}_{jj}^{(1)}}}}{2\pi \mathbf{G}_{ii}^{(1)}\mathbf{G}_{jj}^{(1)}} \tag{24}$$

We verify this formula in Fig. 3, by computing the infinite width covariance at layer two, and showing that it matches with the covariance of preactivations in randomly sampled GNNs.

This same strategy can be used to compute the distribution in the l-th layer, by iterating using 22, for the $l+1$ and l-th layers, instead of the second and first. Code is provided that checks this formula experimentally.

The key point is that this recursion gives us a basis for saying that the *covariance of the l-th layer preactivation distribution of a GCN at initialization depends on the l-th power of the adjacency matrix*. For the SSGC, we only have one layer, so we stop at the first layer distribution (Eq. 19), using the diffusion kernel (Eq. 2) instead of a normalized adjacency matrix. In Fig. 4, we plot the spectrum of the diffusion kernel and the l-th power of the adjacency matrix as a function of l, and show that while both decay as the network grows deeper, SSGC decays more slowly. The decay rate of the spectrum of the covariance tells us how quickly the network outputs converge to some low dimensional manifold in feature space as we increase the depth, and so we see that the SSGC maintains variability in feature space better than a vanilla GCN.

6 Conclusion

In this report we discuss the problem of oversmoothing in graph neural networks. We use graph neural tangent kernels to show experimentally how jumping knowledge and the simple spectral graph convolution can be used to help solve this problem. We also use effective field theory to show theoretically why oversmoothing occurs, and provide a theoretical basis for why the simple spectral graph convolution suffers less from oversmoothing compared to the GCN.

6.1 Limitations and Future Directions

In this report, we assume that the NTK, which models an infinite width neural network, behaves similarly to finite width networks which see practical use. This assumption rests on the observation

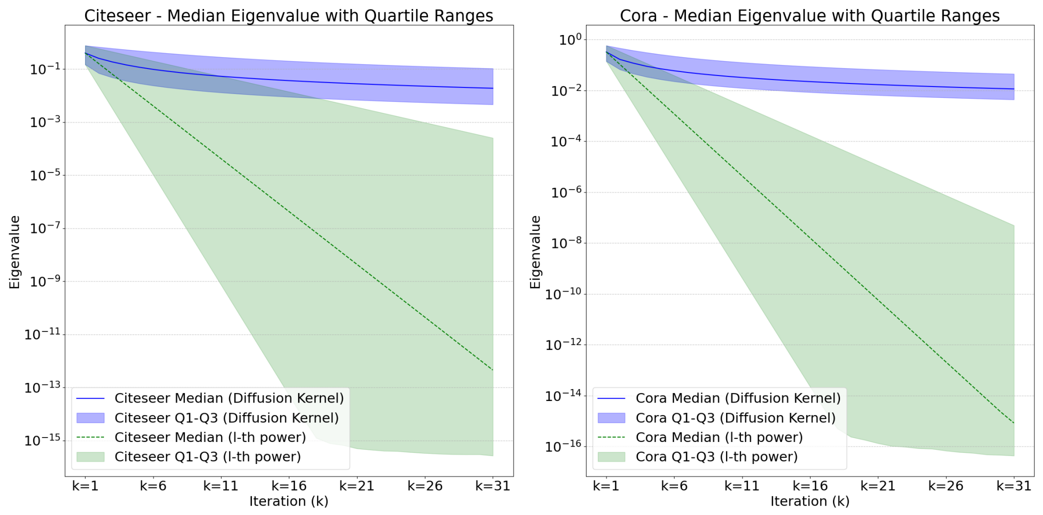

Figure 4: The spectrum of the l-step diffusion kernel and the l-th powers of the adjacency matrix for Cora and Citeseer, plotted on a log scale. The diffusion kernel's spectrum decays more slowly than the adjacency matrix.

that high width networks typically perform well, despite their over-parametrization. Vyas et al. [2022] examines the limitations of NTKs in scaling with large datasets, finding that NTKs do not scale as well as neural networks and suffer from worse performance comparatively on large datasets. It remains an open question whether a similar discrepancy may apply to how GNTKs and GNNs scale with depth. Both our report and Sabanayagam et al. [2021] run some experiments in this direction, however more rigorous and comprehensive experiments or theoretical results are needed to fully characterize when GNTKs can effectively be used to model GNNs. In our effective field theory formalism, we note where the finite width correction modifies the network dynamics; in particular it leads to deviations from Gaussianity in the output distribution at initialization, and induces non-trivial couplings between neurons in the same layer. Actually computing this distribution, however, is challenging, and is left to future work.

Another limitation of our report is that we present various strategies to mitigate oversmoothing, but none are effective in fully solving it. In the infinite depth limit, both jumping knowledge and SSGC networks will end up with high receptive field aggregations drowning out the lower receptive field aggregations which were intended to be preserved. Skip connections do not necessarily suffer from this same issue, however our empirical analysis shows very poor performance at higher depths. It remains an open question how to create a GNN which can improve in performance even as depth grows very large.

Finally, we note that the effective field theory formalism that we develop here applies only to the network at initialization and assumes the infinite width limit. Thus, it does not deal directly with the dynamics of training. Roberts et al. [2022] handle this by extending their formalism for MLPs to developing an effective field theory of the neural tangent kernel, and computing finite width corrections to it. In future work, we can apply similar techniques to the graph learning context, to study how oversmoothing affects and is affected by training.

6.2 Code

All code and instructions to replicate our results are at https://github.com/sidk2/gntk

6.3 Contributions Statement

JF derived the NTK for the simple spectral graph convolution, implemented the NTK for node classification and skip connections. SK implemented the NTK for the simple spectral convolution. JF and SK both ran experiments. SK derived the preactivation distribution for the GCN and SSGC. JF and SK both wrote this report.

References

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International conference on learning representations*, 2021.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification, 2021. URL `https://arxiv.org/abs/1905.10947`.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks, 2020. URL `https://arxiv.org/abs/2006.13318`.

Mahalakshmi Sabanayagam, Pascal Mattia Esser, and Debarghya Ghoshdastidar. New insights into graph convolutional networks using neural tangent kernels. *CoRR*, abs/2110.04060, 2021. URL `https://arxiv.org/abs/2110.04060`.

Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*, volume 46. Cambridge University Press Cambridge, MA, USA, 2022.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018. URL `http://arxiv.org/abs/1806.07572`.

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. *Advances in neural information processing systems*, 32, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018. URL `https://arxiv.org/abs/1806.03536`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL `https://arxiv.org/abs/1609.02907`.

Nikhil Vyas, Yamini Bansal, and Preetum Nakkiran. Limitations of the ntk for understanding generalization in deep learning, 2022. URL `https://arxiv.org/abs/2206.10012`.