Graph Edge Pruning using NNK Graphs

Carlos Hurtado
carloshurtadocomin@gmail.com

Javier Ruiz-Hidalgo
j.ruiz@upc.edu

David Bonet
davidbonetsole@gmail.com

Abstract

Graph Neural Networks (GNNs) show promising results in a great variety of tasks, but their performance tends to suffer, especially as model depth increases, due to difficulties in training such as but not limited to overfitting and training stability. Overfitting weakens the generalization ability of a model. In this paper, we study non-negative kernel edge pruning (NNK-EP), a novel form of regularization based on non-negative kernel (NNK) regression graphs, with which we perform pruning on node neighborhoods in order to alleviate overfitting. We do so by experimenting on Graph Convolutional Networks (GCN), a typical GNN architecture. This decreases the connectivity of the graph data thus acting as a message passing reducer. In our experiments, we fail to show whether there is any benefit to using this approach in solving training difficulties caused by overfitting.

1. Introduction

Graph Convolutional Networks (GCN) [20] as well as other Graph Neural Network (GNN) architectures, have become very popular and widely adopted for learning graph data. In this context, we work with graph representation of data points, where each node in a graph usually has a position and an embedding and is connected with other nodes through weighted edges. This way, at each graph convolutional layer, every node state is updated by summarizing the information of the current neighbor states, in combination with some function with the current node state. This framework is known as *neural message passing* [16], and many other popular GNN architectures [35, 18, 9, 5, 36] build upon it.

The receptive field is a function of the number of GCN layers, such that the deeper a network is, the further in the graph the information of a node will be passed. In contrast to traditional CNNs, GNNs are generally shallow, consisting of no more than 3 or 4 layers, which means that a shorter range of interactions between nodes is considered. This is because there is a decline in performance of GNNs directly correlated with their depth [25, 27, 28, 7].

In this paper, we tackle this issue by changing the connectivity of the graph data by decreasing the number of edges in the graphs. To achieve this we study the node neighbor spaces using non-negative kernel (NNK) regression graphs [31] using the features of the nodes. NNK graphs take advantage of the local geometry to sparsify a K-nearest neighbor (KNN) graph. In this work, we instead of finding an initial set of neighbors, we start from the connectivity of the nodes from the graph data. In Fig. 1 we show a simple schema of the algorithm we propose.

NNK has been shown to deliver good results for semi-supervised learning, image representation [30], and label interpolation and generalization estimation in neural networks [29]. Furthermore, NNK has also been used to understand CNN channel redundancy [3] and propose an early stopping criterion [4]. Moreover, graph properties have been also proposed for the understanding and interpretation of deep neural network performance [17], latent space geometry [21, 22], improve model robustness [23]. Nevertheless, in this work, we explore the usefulness of the NNK algorithm for training Graph Neural Networks. We use NNK to reduce the size of each node's neighborhood in graph data at different stages of a Graph Neural Network. We study whether this decrease in connectivity buffers the drop in performance and makes the network have a harder time overfitting the training data. Additionally, we also investigate if dropping edges with the

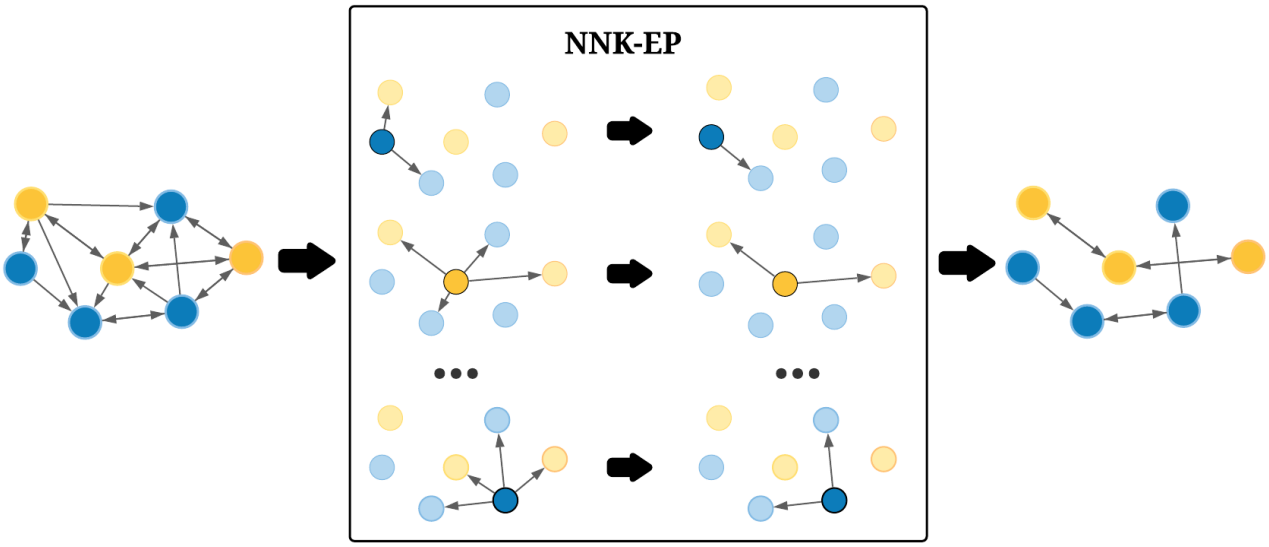

Figure 1. NNK-EP block schema, where the color of the nodes represents similarity in their embeddings, and the arrows represent edges. For a given graph input the NNK-EP algorithm, for each node, constructs the NNK graph from a Gaussian kernel of the node neighborhood (i.e., the nodes adjacent to it). Then, the edges pointing to the nodes that do not belong to the NNK graph are pruned from the original graph.

NNK-EP algorithm provides any advantage to randomly dropping a fraction of the edges [28], NNK-EP being much more costly.

2. Related Work

The problems of overfitting and training instability are conventionally addressed by different normalization layers both from traditional Deep Neural Networks [19, 34, 1] and also specific to GNN architectures [6, 13, 38, 24, 39], since they stabilize the training process and serve as an implicit regularization method. Still, most normalization methods on their own are not enough and often additional types of regularization are necessary for effectively training GNNs. Furthermore, the methods mentioned above consist of normalizing different values involved in the training of a GNN (node features, parameters, or layer inputs), and even sometimes with no concrete theoretical basis and proposed based on promising empirical results [6], do not exploit the geometrical properties of the graph data involved in GNNs, and leave the graph shapes and connectivity unchanged.

In contrast, there exist approaches similar to Dropout [32] such as DropEdge [28] and DropNode [10], where a percentage of the edges and/or nodes are dropped from the underlying graphs. As opposed to Dropout, which alters the neural units of the network architecture, the other methods change the data. These methods, which are effective in improving GNN training stability, work in a purely randomized fashion.

3. Background

3.1. Non-Negative Kernel (NNK) regression graphs

For a given matrix $\boldsymbol{K}$ representing the similarity between N feature vectors x, the NNK algorithm constructs a graph such that each data point or node is connected to the data points most similar to it. This way, the similarity between nodes is denoted by the weight of the edge between them, while the absence of an edge is indicated with a zero weight. The similarity between two data points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is usually defined with a positive definite kernel such as the Gaussian kernel with bandwidth σ (1) or the range normalized cosine kernel (2), but the NNK graph construction framework may potentially be extended to non-standard kernels.

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp\left(-\frac{||\boldsymbol{x_i} - \boldsymbol{x_j}||^2}{2\sigma^2}\right) \tag{1}$$

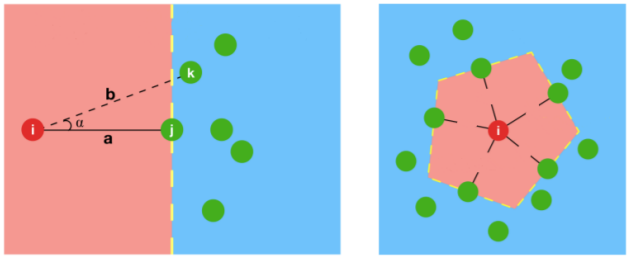

Figure 2. Local geometry (denoted in red) of NNK graph construction for a Gaussian kernel. Left shows the plane associated to an edge. Right shows convex polytope associated with a node. Extracted from [31].

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \frac{1}{2} + \frac{\langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle}{2||\boldsymbol{x_i}||||\boldsymbol{x_j}||} \tag{2}$$

With this approach (NNK), graph construction is viewed as a signal representation problem [31]. In contrast, K-nearest neighbor (KNN) [12] and ε-neighborhood graphs [8], which optimize the edge weights while leaving connectivity unchanged, are very sensitive to the choice of the sparsification parameters K and ε respectively [2].

Even though some form of neighbor finding (such as KNN) is used for initialization, NNK performs a selection similar to the orthogonal step in orthogonal matching pursuits [33] which makes NNK more robust to the choice of sparsity parameters in initialization. Additionally, the resulting graph has a geometric interpretation [31] for the case of the Gaussian kernel 1, such that each edge in an NNK graph corresponds to a hyperplane with normal in the edge direction, as illustrated in Fig. 2. Points beyond each hyperplane are ignored (edge weight zero).

4. Edge Prunning with NNK Graphs (NNK-EP)

NNK aims at constructing an optimal graph, robust to the choice of sparsity parameters and with interesting geometric and theoretical properties. This framework can be extended to the optimization of node adjacency in a graph. Graph datasets (i.e., datasets where each data point is a single graph) often define adjacency as the K-closest nodes to each node [14, 26], thus resulting in graphs where all nodes have the same degree, and information is passed more nodes that might be necessary.

To address this problem, in this work we propose the construction of a local polytope for each node

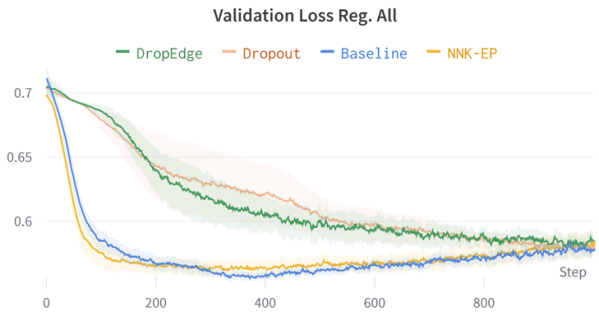

Figure 3. Validation loss resulting from the training of four models: a baseline, and three models with DropEdge, Dropout, or NNK-EP regularization after each of the four GCN layers. While the slope of the curves relative to the Baseline and NNK-EP show a tendency to overfit, the models with Dropout and DropEdge don't show an overfitting pattern.

in order to make the graph data more sparse (NNK-EP). By applying NNK-EP to the graph at different stages in a neural network we allow the information in the message passing step of the following convolutional layer to be passed only to a select set of neighbors. This decrease in connectivity can alleviate oversmoothing and/or overfitting, common issues when training Graph Convolutional Networks. Moreover, studying the size of node neighborhoods at different layers of a network can allow for the development of a better understanding of node neighborhoods in GCNs.

4.1. NNK-EP algorithm

For a graph with N nodes, we use the adjacency and the features of each node to construct N independent graphs. This way, the convex polytope in kernel space is constructed from the adjacency of the graph before the NNK-EP step, which can be the initial adjacency of the graph as defined by the dataset or the resulting adjacency after a previous NNK-EP step. After the NNK optimization, we prune the edges with a zero weight, while the remaining edges and their edge attributes stay untouched. Thus, NNK-EP is strictly an edge removal algorithm. Fig. 1 shows what the NNK-EP algorithm does with a simple example.

While the output for a directed graph is straightforward, there is some nuance regarding undirected graphs. We optimize the neighborhood of each node, then the most intuitive outcome is that each node keeps the edges that are more relevant to itself and there-

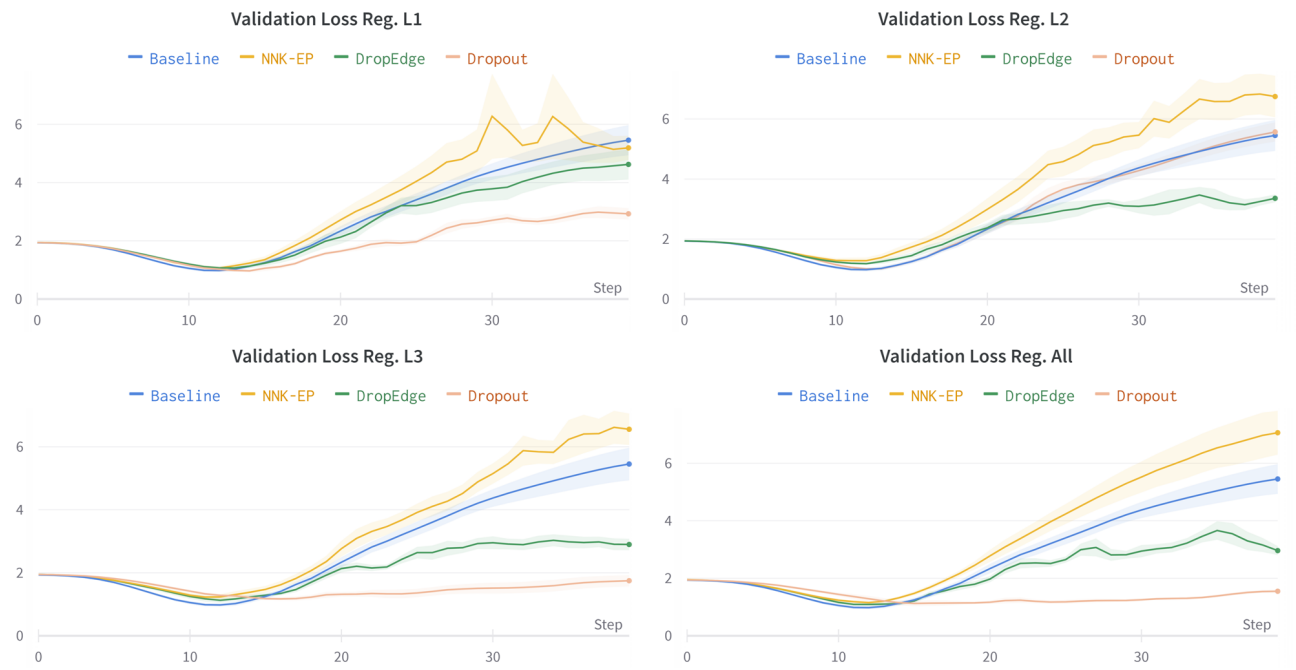

Figure 4. This figure shows the Validation loss for a Baseline model, a model with Dropout with $p = \frac{1}{2}$, DropEdge with $p = \frac{1}{2}$, and a model with NNK-EP. Each line represents the mean of 5 runs, and the area above and below it is its standard error. Each chart shows the results with the regularization happening only after the corresponding layer, the chart on the bottom right having regularization after each layer. DropEdge and Dropout push the validation loss below the baseline, while NNK-EP seems to decrease performance.

fore the graph becomes directed. Furthermore, this directed approach can not result in isolated nodes, since each node will be adjacent to at least one other node. Instead, nodes that pass their information but receive none can appear in the graph. This directed approach is achieved by making all edges in an undirected graph bi-directional, which is the default manner in which *PyTorch Geometric* [15], the library used for the experiments, handles undirected graphs.

5. Experiments

In the following experiments, we use NNK-EP to study the performance of Graph Convolutional Networks under conditions of overfitting while comparing it to other existing regularization methods. Moreover, we show their behavior when used at different levels of a network. In order to assess the validity of our proposed algorithm, we study its behavior on the two main tasks for which GNNs are used. These are graph classification and node classification. We have performed the graph classification experiments on PROTEINS [26, 11], a graph classification dataset. We have also experimented with overfitting on the node classification task on the CORA [37] dataset, which consists of a single graph of 2708 scientific publications classified into one of seven classes.

5.1. Tackling overfitting with NNK-EP

5.1.1 Graph Classification Task

For this task, the tests have been run for 1000 epochs with the Adam optimizer, a learning rate of 10^{-4}, and a batch size of 32. We train over a class-balanced subset of 70% of the data points of the original dataset. We use a model with four GCN layers of sizes 6, 12, 24, 48, followed by four fully connected layers of sizes 24, 12, 4, 2. This is a baseline model, over which we add either a Dropout, DropEdge, or NNK-EP step after each of the four GCN layers.

The results of this experiment are shown in Fig. 3. On the other hand, it can be seen that both the Baseline and the NNK-EP models quickly find low values of the validation loss, but have a hard time improving and instead show an upward trend, a sign of overfit-

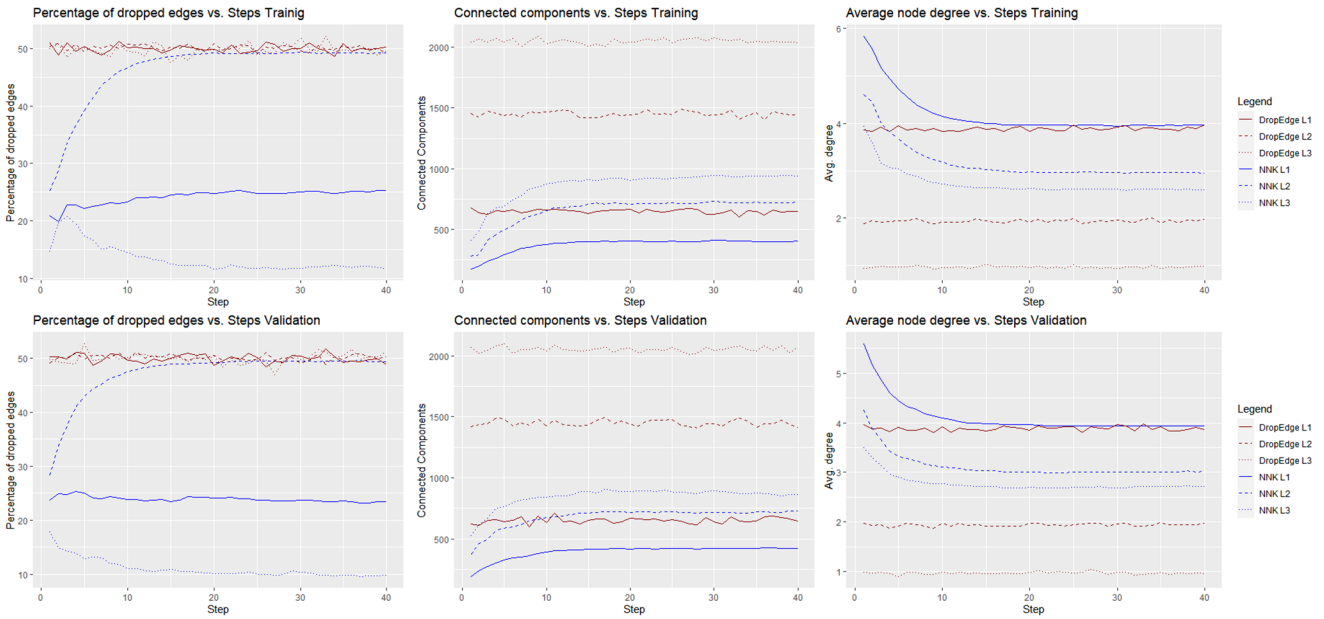

Figure 5. Different properties of the CORA graph as a function of the regularization algorithm used (i.e., DropEdge or NNK) and the training steps performed. The first column shows the percentage of dropped edges from the graph, the second the number of connected components, and the third the average number of neighbors of the nodes.

ting. On the other hand, the models with DropEdge and Dropout regularization steps learn slower, and the slope of the validation loss does not show any sign of overfitting.

5.1.2 Node Classification Task

The experiments run on this dataset span 40 epochs, using the Adam optimizer, a learning rate of 10^{-2}, and a batch size of 64. The baseline model consists of a GNN with three Graph Convolutional Layers [20], all of size 32, followed by a fully connected layer also of size 32 and finally a fully connected layer of size 7, as many as possible classes. In contrast to the models used for graph classification, here there is no pooling since we want all the nodes at the end.

In this section, we try to regularize the overfitting baseline with Dropout with $p = \frac{1}{2}$, DropEdge with $p = \frac{1}{2}$, and finally NNK-EP. This way, we can assess the validity of NNK-EP as a regularizer and also compare it to other regularization methods.

We show our results for this experiment in Fig. 4, where we plot the validation loss resulting from training the four models mentioned above. Each line encodes the mean of five executions, together with the standard error. Each chart shows the validation loss when the regularization step is performed either after the first GCN layer, after the second, the third, or after all the GCN layers. On the one hand, it can be seen that despite not being very effective when set only after the middle layer, Dropout seems to be the most reliable form of regularization. On the other hand, DropEdge also appears to be useful, to tackle overfitting, and its effectiveness seems to increase the later in the network it is performed. Finally, although NNK-EP seems to be slightly useful when performed after the first GCN layer, it causes a decrease in performance in the other examples. Moreover, the bands of the standard error for the NNK-EP line are much wider, which indicates a high training instability that decreases in the other two regularization examples with respect to the baseline.

5.2. Efficacy of NNK-EP

In this section, we explore the differences in how NNK-EP and DropEdge interact with the graph data during training. We train the same model as in section *5.1.2 Node Classification Task*, once with an NNK-EP step after each GCN layer, and another one with a DropEdge with $p = \frac{1}{2}$ step after each GCN layer. We study the percentage of edges dropped, the average degree of the nodes, and the number of connected

components all after each regularization step. We also compare NNK-EP to DropEdge with different probabilities of removing edges from the graph.

5.2.1 Graph geometry and regularization

In the first column of Fig. 5 we show the percentage of edges dropped after each regularization step for both models at each epoch. While the number of edges removed by DropEdge after every layer remains constant, the number removed by NNK-EP changes with training. This behavior change is due to the fact that NNK-EP uses the node attributes, which change as the model learns. We might think that the increase in the number of edges dropped is because as the model learns, the node attributes convey meaningful information that might be affecting the NNK-EP process. We should then expect this not to happen in validation when the model is overfitting, but despite overfitting the training data, this behavior is not observed. The edges dropped after the first layer remain stable at around 20%; the edges dropped after the second layer rapidly increase and stabilize close to 50%; finally, the edges dropped after the third layer decreased to around 10%.

In the second column of Fig. 5 we show the number of connected components in the graph at each layer as a function of the training steps. To obtain the connected components we turn the graph, which was initially directed, into an undirected graph. Before dropping any edges, the graph consists of 78 connected components. It is interesting to observe that after the first layer, even though both NNK-EP and DropEdge remove a similar number of edges, the total of connected components is much smaller after the NNK-EP step.

Finally, in the third column, we show the average degree of the nodes in each layer after the regularization step. As opposed to the number of connected components, in this case, the behavior of NNK-EP and DropEdge is very similar in the first layer.

5.2.2 Regularization intensity

We now test whether the difference in performance between DropEdge and NNK-EP is because NNK-EP removes fewer edges than DropEdge with $p = 0.5$. To do so, we compare a model that uses NNK-EP after each layer with three models using DropEdge at different dropping rates. While NNK-EP, following the percentage of dropped edges in Fig. 5, leaves a total of 33.8% of the edges at the end of the network, DropEdge of $p = \frac{1}{6}, \frac{1}{2}$ and $\frac{2}{4}$, a total $57.9\%, 12.5\%$ and 3.7% of the edges remain respectively. In Fig. 6 we can see that there is no impact on the loss caused by the difference in power of the DropEdge used. Then, the difference in performance is caused not by the regularizing power but by the way the edges are dropped from the graph.

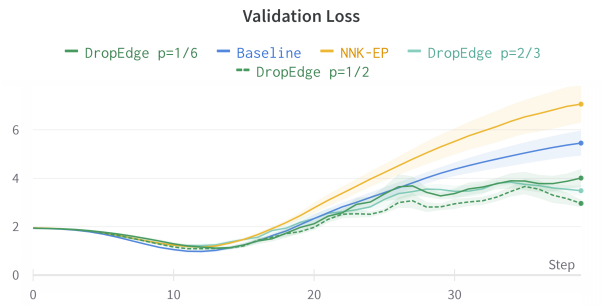

Figure 6. Plot of the validation loss for four different models trained on the CORA dataset for 40 epochs. The plot shows that the performance of a model with regularization is independent of the power of the regularization step.

6. Conclusion

We have studied the performance of NNK-EP as a regularizer for solving graph classification and node classification problems. Furthermore, we have analyzed the impact of both NNK-EP and DropEdge on the graph data during the training and validation process. We have observed that NNK-EP does not work as a regularizer, not because of a lack of strength, but because of the way the edges are dropped. While algorithms such as Dropout or DropEdge add random noise to the data, with NNK-EP, by removing edges from a node's neighborhood, we are decreasing the receptive field of the GCN layers.

Future work should consider a different neighborhood for the NNK-EP process to work on, so instead of using the existing edges in the graph, the algorithm may first perform a KNN step and use the resulting edges, or even apply NNK-EP to a fully connected graph. Additionally, combining a DropEdge layer with an NNK-EP layer might prove to be useful.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. 11 label propagation and quadratic criterion. 2006.

[3] David Bonet, Antonio Ortega, Javier Ruiz-Hidalgo, and Sarath Shekkizhar. Channel redundancy and overlap in convolutional neural networks with channel-wise nnk graphs. *arXiv preprint arXiv:2110.11400*, 2021.

[4] David Bonet, Antonio Ortega, Javier Ruiz-Hidalgo, and Sarath Shekkizhar. Channel-wise early stopping without a validation set via nnk polytope interpolation. *arXiv preprint arXiv:2107.12972*, 2021.

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[6] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR, 2021.

[7] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

[8] Václav Chvátal and PL Hammer. Aggregations of inequalities. *Studies in Integer Programming, Annals of Discrete Mathematics*, 1:145–162, 1977.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[10] Tien Huu Do, Duc Minh Nguyen, Giannis Bekoulis, Adrian Munteanu, and Nikos Deligiannis. Graph convolutional neural networks with node transition probability-based message passing and dropnode regularization. *Expert Systems with Applications*, 174:114711, 2021.

[11] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[12] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

[13] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[14] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[15] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[17] Vincent Gripon, Antonio Ortega, and Benjamin Girault. An inside look at deep neural networks using graph signal processing. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9. IEEE, 2018.

[18] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

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

[21] Carlos Lassance, Myriam Bontonou, Ghouthi Boukli Hacene, Vincent Gripon, Jian Tang, and Antonio Ortega. Deep geometric knowledge distillation with graphs. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8484–8488. IEEE, 2020.

[22] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Laplacian networks: Bounding indicator function smoothness for neural networks robustness. *APSIPA Transactions on Signal and Information Processing*, 10, 2021.

[23] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Representing deep neural networks latent space geometries with graphs. *Algorithms*, 14(2):39, 2021.

[24] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[26] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[27] Kenta Oono and Taiji Suzuki. On asymptotic behaviors of graph cnns from dynamical systems perspective. *CoRR*, abs/1905.10947, 2019.

[28] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[29] Sarath Shekkizhar and Antonio Ortega. Deepnnk: Explaining deep models and their generalization using polytope interpolation. *arXiv preprint arXiv:2007.10505*, 2020.

[30] Sarath Shekkizhar and Antonio Ortega. Efficient graph construction for image representation. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1956–1960. IEEE, 2020.

[31] Sarath Shekkizhar and Antonio Ortega. Graph construction from data by non-negative kernel regression. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3892–3896. IEEE, 2020.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[33] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on information theory*, 53(12):4655–4666, 2007.

[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

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

[36] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[37] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[38] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[39] Kuangqi Zhou, Yanfei Dong, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Effective training strategies for deep graph neural networks. *arXiv e-prints*, pages arXiv–2006, 2020.