OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Normalization Matters for Optimization Performance on Graph Neural Networks

Alan Milligan ALANMIL@CS.UBC.CA
Frederik Kunstner KUNSTNER@CS.UBC.CA
Hamed Shirzad SHIRZAD@CS.UBC.CA
Mark Schmidt[†] SCHMIDTM@CS.UBC.CA
Danica J. Sutherland[†] DSUTH@CS.UBC.CA
University of British Columbia, Vancouver, Canada
Canada CIFAR AI Chair (Amii)[†]

Abstract

We show that feature normalization has a drastic impact on the performance of optimization algorithms in the context of graph neural networks. The standard normalization scheme used throughout the graph neural network literature is not motivated from an optimization perspective, and leads (S)GD to frequently fail. Adam does not fail, but is also negatively impacted by standard normalization methods. We show across multiple datasets and models that better motivated feature normalization closes the gap between Adam and (S)GD, and speeds up optimization for both.

1. Introduction

Data normalization is a standard step in data processing pipelines. Normalizing the features to have mean zero and unit variance is taught in introductory machine learning (e.g. Murphy, 2022, Ch. 10) and widely used in deep learning (e.g. LeCun et al., 2012)). But the increasing complexity of model architectures, data processing schemes, and optimization algorithms often takes the spotlight off these classical tools. Each subcommunity ends up developing its own set of standard tricks and practice, tuned to the specific problems encountered in vision, language, or graph data. While it makes sense that different tools and practices may be needed for different data types, it can also lead to situations where practices that are considered standard or common knowledge in one subcommunity are unknown or ignored in another. This paper highlights one such case. While much of the machine learning community is aware of the benefits of feature normalization, it appears to have been overlooked in standard benchmarks used to evaluate graph neural networks (GNNs). We show that paying attention to normalization has a substantial impact on training performance.

Our main contributions are:

- We highlight that fitting GNNs with gradient descent (GD) can be difficult, even for small problems included in standard GNN benchmarks, while Adam (Kingma and Ba, 2015) performs better.
- We show that this performance gap is due to (a lack of) data normalization. The gap is more pronounced on datasets with bag-of-words features, which lead to data matrices with heavy-tailed singular values. Normalizing *per-feature*, rather than the *per-row* that appears to be the default in GNNs, gives a simple fix that drastically speeds up optimization, regardless of the algorithm used.

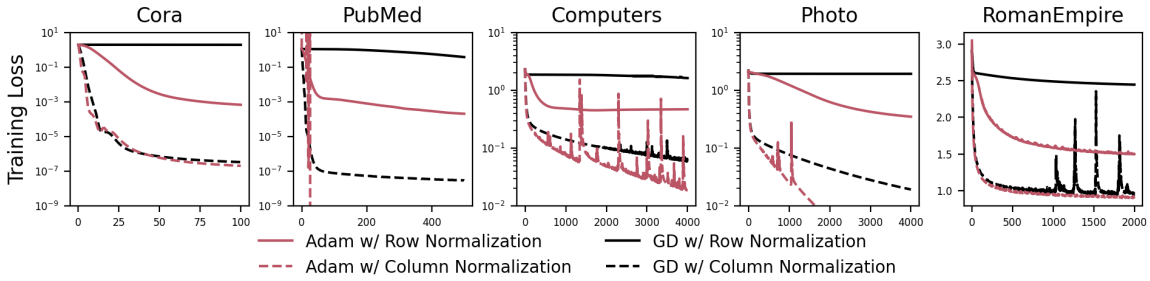


Figure 1: **GD struggles on GNNs with standard row normalization, but optimization becomes much easier with column normalization.** Training a Graph Convolutional Network (GCN) using **SGD** and **Adam**, using row normalization (solid) and column normalization (dashed). Similar results hold across models and datasets, as shown in Appendix B.

Our contributions are summarized in Figure 1, which shows that GD fails to fit the model while other algorithms such as Adam work when using the *per-row* normalization. Normalizing *per-feature* leads to a dramatic performance improvement for both algorithms.

2. Effects of column- and row-normalization

While it is "common knowledge" in some circles that GD can struggle on GNNs, this problem is not well documented in the literature. There are a few statements that GD is considered slow or requires more tuning (Morris et al., 2024), and some empirical comparisons show poor performance for GD on some problems (e.g. You et al. 2020 or Izadi et al. 2020, Fig. 1). To our knowledge, there has not been a systematic investigation of the performance of optimization algorithms on GNNs. We first show that even well-studied problems common in GNN benchmarks can be hard to fit with GD.

We run full-batch gradient descent with momentum $\beta = 0.9$, and tune the step size with a grid search to minimize the training loss at the end of the given budget. We repeat this process with Adam, tuning the step size but leaving the momentum parameters to the default $\beta_1 = 0.9$, $\beta_2 = 0.999$.

We show results using a Graph Convolutional Network (GCN) (Kipf and Welling, 2016) in Figure 1, using the row normalization used in most GNN data preprocessing (solid lines). Strikingly, GD makes almost no progress, across all datasets. Adam, on the other hand, does roughly work. However, when we change to *per-feature* normalization (dashed), both GD *and Adam* train drastically faster, mostly eliminating the performance gap. This effect is due to the features and can be replicated across architectures. We show similar effects with linear models in Figure 2, and Graph Attention (GAT) networks and GraphSAGE in Appendix B.

The per-feature normalization step used above common standardization method to achieve zero mean and unit variance. With `numpy`-like broadcasting rules, assuming $\mathbf{X}$ is an $n \times d$ matrix of n samples with d features each, the operation computes the normalized data matrix $\tilde{\mathbf{X}}$ as

$$\tilde{\mathbf{X}} \leftarrow (\mathbf{X} - \text{mean}(\mathbf{X}, \text{dim} = 0)) \quad \text{\# mean } \textit{returns a vector of } d \textit{ means}$$
$$\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} / \text{std}(\tilde{\mathbf{X}}, \text{dim} = 0) \quad \text{\# std } \textit{returns a vector of } d \textit{ standard deviations}.$$

We call this *column normalization*, as it normalizes each column of features independently.

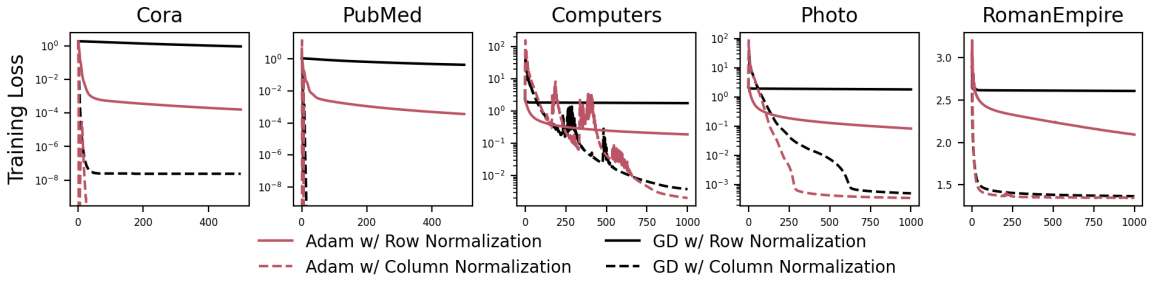


Figure 2: **On a linear model, both Adam and GD are improved by by switching from row normalization to column normalization.** Training a Linear GCN using **SGD** and **Adam**, using row normalization (solid) and column normalization (dashed).

The preprocessing step found in many GNN problems, *row normalization*, differs in two ways. Rather than ensure the data has mean zero and unit standard deviation, it normalizes the data to be non-negative and sum to 1. More importantly, it operates on each row (the features for one node in the graph) independently. That is,

$$\tilde{\mathbf{X}} \leftarrow (\mathbf{X} - \min(\mathbf{X}, \text{dim} = 1)) \quad \text{\# min } \textit{returns a vector of } n \textit{ minima}$$
$$\tilde{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} / \text{sum}(\tilde{\mathbf{X}}, \text{dim} = 1) \quad \text{\# sum } \textit{returns a vector of } n \textit{ sums}.$$

This row normalization operation is implemented in PyTorch Geometric (Fey and Lenssen, 2019) as `NormalizeFeatures`, DGL (Wang et al., 2019, `RowFeatNormalizer`) and Spektral (Grattarola and Alippi, 2021, `NormalizeOne`). This row-normalization may have been inherited from Kipf and Welling (2016) who used it on datasets where node are represented by bag-of-word features on a textual representation of the node.[1] Those features take value in $\{0, 1\}$ to indicate whether a word is present. On this data, subtracting the minimum does nothing, while the sum-to-one normalization changes the representation of a sample from the sum of the word embedings to their average.

Both normalization procedures address orthogonal issues (samples of different scales vs. features of different scales). They can, and depending on the application possibly should, be combined. The issue is that row-normalization alone appears to have become a standard in GNN benchmarks. It appeared in problems used to establish foundational GNN methodology (Kipf and Welling, 2016; Veličković et al., 2018) and continues to be used in modern evaluations (Luo et al., 2024). Others have also called attention to data normalization in GNNs, as Tönshoff et al. (2024) criticized the benchmark Dwivedi et al. (2022) for not normalizing vision data following best practices (i.e., normalizing the channels). But far more attention is devoted to normalization procedure within the network (Ioffe and Szegedy, 2015; Ulyanov et al., 2016; Ba et al., 2016; Zhao and Akoglu, 2020; Cai et al., 2021; Dwivedi et al., 2022) than the normalization of the input data. By ignoring the inputs, we are routinely solving much harder optimization problems than we need to.

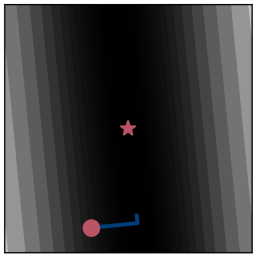
1. Kipf and Welling (2016) motivate this normalization by saying that the use the parameter initialization scheme of Glorot and Bengio (2010) and "accordingly (row-)normalize input feature vectors" (and indeed they use $n \times d$ matrices). We can find no reference to row normalization by Glorot and Bengio; rather, their scheme is explicitly motivated by an assumption that "input features variances are the same," i.e. *column* normalization.

3. Effect of normalization on imbalanced features

To highlight the benefits of column normalization and the potential pitfall of using only row-normalization, consider the used-car dataset below. It has three features, mileage, sale price and year of construction, which span orders of magnitude. Fitting a linear regression on this input data would give give a poorly conditioned quadratic optimization problem, leading to slow optimization.

Raw data X

Mileage (km)	Sale price ($)	Year
100 000	16 999	2008
130 000	14 999	2013
150 000	11 999	2012
200 000	17 999	2014

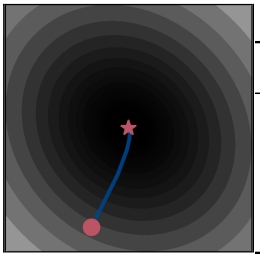


Normalized loss Landscape. 2d-slice of the loss landscape on the first two coordinates for a quadratic $\mathcal{L}(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w}\|^2$. In blue, 100 iterations of GD starting at ●. Minimum at ⋆.

The quadratic optimization problem resulting from a linear regression with this input data has a condition number of $\approx 10^5$. The reason for this large condition number is features of different scales. Column normalization fixes this problem and yields a condition number of ≈ 10. As gradient descent requires $O(\kappa \log(\epsilon))$ iterations to achieve an error of ϵ (in the worst-case), column-normalization gives a speed up in optimization performance of 10^4, see below. Row-normalization does not fixes this imbalance, as shown below, and using it on generic data introduces other problems.

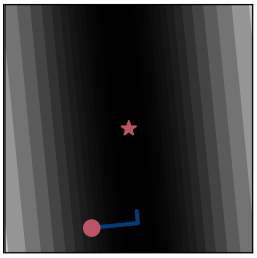
Column-normalized

Mileage	Price	Year
-1.24	0.66	-1.64
-0.41	-0.22	0.54
0.14	-1.53	0.11
1.51	1.09	0.98



Row-normalized

Mileage	Price	Year
0.87	0.13	0
0.91	0.09	0
0.94	0.06	0
0.93	0.08	0



Problem 1: Row-normalization does not normalizing the features. Despite transforming the data (and being called `NormalizeFeatures` in some libraries), row-normalization does not adress the problem of having very different scales across features. The optimization problem is not made any easier, and can be made more difficult. In Appendix C, we show that **row normalization can lead to worse performance than not preprocessing the data at all** across models and datasets.

Problem 2: Destroying information. Column normalization can be implemented as a linear operation, $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{R} + \mathbf{b}$, where $\mathbf{R} : \mathbb{R}^{d \times d}$ and $\mathbf{b} : \mathbb{R}^d$. Thus, linear models (with biases) on the raw data and the normalized data have equivalent expressivity.[2] Row normalization, however, can destroy information. On the example above, the *year* feature becomes exactly 0 and the dataset is now only 2-dimensional. This is not an issue when each samples does not have the same minimum feature, for example if all features are binary as the subtraction of the minimum is a no-op. But on arbitrary dataset, as above, row-normalization can make it harder to fit the data.

2. Generalization, however, may differ due to regularization. This can be explicit, by changing the impact of l2-regularization, or implicit, by changing which minimum-norm solution the algorithm converges to.

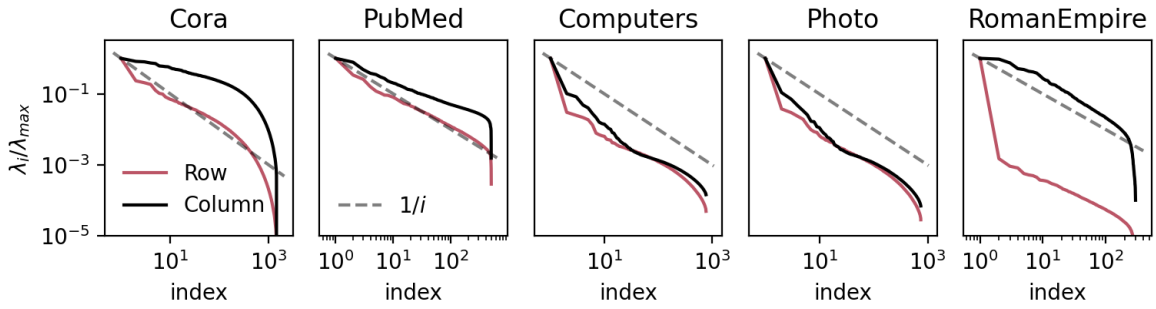


Figure 3: **Column Normalization improves conditioning on a linear model with GNN datasets.** Normalized eigenvalues ($\lambda_i/\lambda_{\max}$) of the Hessian of a linear model with row- and column-normalization. Dashed line includes $1/i$ for scale. The relative eigenvalues are larger when normalizing by column, indicating faster convergence with GD.

4. Features with different scales in GNN benchmark problems

Typical graph benchmark datasets exhibit an imbalance in the features. However, this imbalance does not appear through features of different scales, as in Section 3, but through features of different frequencies. Many datasets use bag-of-words features, binary vectors encoding whether a word is present. Those words have different frequencies (typically roughly a power law), which leads to data matrices $\mathbf{X}^\top\mathbf{X}$ that have a very imbalanced diagonal and imbalanced eigenvalues (if $\mathbf{x}_j$ is binary, $\frac{1}{n}(\mathbf{X}^\top\mathbf{X})_{jj}$ is the frequency of $\mathbf{x}_j = 1$). Row normalization does not address this frequency-based imbalance while column normalization drastically improves performance.

To illustrate this effect, we give in Figure 3 a visualization of the spectrum of the eigenvalues of a linear (one layer, no activation function) GNN using row- and column-normalization. We show the normalized $\lambda_i/\lambda_{\max}$, which are indicative of the convergence speed of gradient descent along the ith eigenvector, as the error contracts by a factor of $(1 - \lambda_i/\lambda_{\max})$ at each step if the step-size if $\alpha = 1/\lambda_{\max}$. Column normalization produces values of $\lambda_i/\lambda_{\max}$ that uniformly higher than row-normalization. While the eigenvalues only provide a direct argument for why normalization helps on linear models (such as in Figure 2), it helps us understand where the difficulty of the problem arises and suggests why the optimization performance improves on the deep models of Figure 1.

5. Discussion, Related Work, and Future Work

Limitations and Future Work. We focused solely on the training dynamics of GNNs with classical models and datasets. We plan to extend this study to consider generalization and more moden models. Our focus is on deterministic optimization, which is common on graph networks as it is difficult to obtain estimates of the stochastic gradient. However, it is possible that column-normalization might increase the variance during stochastic optimization, as rare features are divided by a small standard deviation, increasing their magnitude. We also do not claim that poorly conditioned inputs are the *only* reason Adam consistently outperforms GD on GNNs; there are likely other properties of model architectures and datasets that influence the optimization dynamics of both algorithms. We hope to explore these further in future work.

GNNs and Normalization. Graph Neural Networks such as GCNs (Kipf and Welling, 2016) and GATs (Veličković et al., 2018) have been used in domains with graph structure such as proteins, co-purchase networks, citation networks, and traffic analysis (Hu et al., 2021; Takac and Zabovsky, 2012; Derrow-Pinion et al., 2021). You et al. (2020) found that Adam usually performs better than GD, but indicated this might be a problem with hyperparameter tuning. The difficulties due to data normalization might have lead to the more widespread adoption of Graph Transformers (e.g. Rampášek et al., 2022; Shirzad et al., 2023; Shirzad et al., 2024; Deng et al., 2024), which use batch normalization (Ioffe and Szegedy, 2015) rather than layer normalization (Ba et al., 2016). Batch norm is very similar to column normalization while layer norm is closer to row normalization, and might mitigate some of the issues highlighted here.

Adam versus (S)GD. Duchi et al. (2010) claimed AdaGrad (a predecessor to Adam) has better performance over GD on sparse data, using bag-of-words features as an example. This is consistent with our results, but we highlight that it is likely the imbalanced features caused an ill-conditioning problem, rather than sparsity. Column normalization densifies features, but helps improve conditioning. Other analysis has shown that heavy tail language based *labels* cause (S)GD to fail due to a poorly conditioned Hessian, while Adam is unaffected (Kunstner et al., 2024). Adam has also been shown to follow a trajectory over which the robust condition number (see Figure 3) is smaller than (S)GD (Jiang et al., 2023).

Acknowledgements

This research was supported in part by the Canada CIFAR AI Chairs program, the Natural Sciences and Engineering Research Council of Canada (NSERC) through Discovery Grants RGPIN-2022-03669 and RGPIN-2021-02974, and the NSERC/FRQNT grant ALLRP-57708-2022, as well as the BC DRI Group and the Digital Research Alliance of Canada.

Bibliography

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 (cited on pages 3, 6).

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang (2021). "GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training." *ICML* (cited on page 3).

Chenhui Deng, Zichao Yue, and Zhiru Zhang (2024). "Polynormer: Polynomial-Expressive Graph Transformer in Linear Time." arXiv: 2403.01232 (cited on page 6).

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Veličković (2021). "ETA prediction with Graph Neural Networks in Google Maps." *Conference on Information and Knowledge Management (CIKM)* (cited on page 6).

John C. Duchi, Elad Hazan, and Yoram Singer (2010). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *COLT* (cited on page 6).

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini (2022). "Long range graph benchmark." *NeurIPS* 35, pages 22326–22340 (cited on page 3).

Matthias Fey and Jan E. Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric." *ICLR Workshop on Representation Learning on Graphs and Manifolds* (cited on pages 3, 11).

Xavier Glorot and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." *AISTATS* (cited on page 3).

Daniele Grattarola and Cesare Alippi (2021). "Graph Neural Networks in TensorFlow and Keras with Spektral [Application Notes]." *IEEE Comput. Intell. Mag.* (cited on page 3).

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov (2018). *Learning word vectors for 157 languages*. arXiv: 1802.06893 (cited on page 9).

Will Hamilton, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs." *NeurIPS* 30 (cited on page 10).

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec (2021). *OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs*. arXiv: 2103.09430 (cited on page 6).

Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ICML* (cited on pages 3, 6).

Mohammad Rasool Izadi, Yihao Fang, Robert Stevenson, and Lizhen Lin (2020). "Optimization of Graph Neural Networks with Natural Gradient Descent." *IEEE BigData* (cited on page 2).

Kaiqi Jiang, Dhruv Malik, and Yuanzhi Li (2023). "How Does Adaptive Optimization Impact Local Neural Network Geometry?" *NeurIPS* (cited on page 6).

Diederik P. Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." *ICLR* (cited on page 1).

Thomas N Kipf and Max Welling (2016). "Semi-supervised classification with graph convolutional networks." arXiv: 1609.02907 (cited on pages 2, 3, 6, 10).

Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti (2024). "Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models." *NeurIPS*. arXiv: 2402.19449 (cited on page 6).

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller (2012). "Efficient BackProp." *Neural Networks: Tricks of the Trade - Second Edition*. Edited by Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller. Springer (cited on page 1).

Yuankai Luo, Lei Shi, and Xiao-Ming Wu (2024). "Classic GNNs are Strong Baselines: Reassessing GNNs for Node Classification." *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (cited on page 3).

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel (2015). "Image-based recommendations on styles and substitutes." *SIGIR* (cited on page 9).

Christopher Morris, Fabrizio Frasca, Nadav Dym, Haggai Maron, İsmail İlkan Ceylan, Ron Levie, Derek Lim, Michael M. Bronstein, Martin Grohe, and Stefanie Jegelka (2024). "Position: Future Directions in the Theory of Graph Machine Learning." *ICML* (cited on page 2).

Kevin P. Murphy (2022). *Probabilistic Machine Learning: An introduction*. MIT Press (cited on page 1).

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova (2023). *A critical look at the evaluation of GNNs under heterophily: Are we really making progress?* arXiv: 2302.11640 (cited on page 9).

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini (2022). "Recipe for a general, powerful, scalable graph transformer." *NeurIPS* (cited on page 6).

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann (2018). *Pitfalls of graph neural network evaluation*. arXiv: 1811.05868 (cited on page 9).

Hamed Shirzad, Honghao Lin, Balaji Venkatachalam, Ameya Velingker, David Woodruff, and Danica J Sutherland (2024). "Even Sparser Graph Transformers." *NeurIPS*. arXiv: 2411.16278 (cited on page 6).

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop (2023). "Exphormer: Sparse transformers for graphs." *ICML*. arXiv: 2303.06147 (cited on page 6).

Lubos Takac and Michal Zabovsky (2012). "Data analysis in public social networks." *International Scientific Conference and International Workshop – Present Day Trends of Innovations*. Volume 1. 6 (cited on page 6).

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe (2024). "Where Did the Gap Go? Reassessing the Long-Range Graph Benchmark." *Trans. Mach. Learn. Res.* 2024 (cited on page 3).

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization." *CoRR* abs/1607.08022. arXiv: 1607.08022 (cited on page 3).

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio (2018). "Graph attention networks." *ICLR* (cited on pages 3, 6, 10).

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang (2019). *Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks*. arXiv: 1909.01315 (cited on page 3).

Zhilin Yang, William Cohen, and Ruslan Salakhudinov (2016). "Revisiting semi-supervised learning with graph embeddings." *ICML*. PMLR (cited on page 9).

Jiaxuan You, Zhitao Ying, and Jure Leskovec (2020). "Design space for graph neural networks." *NeurIPS* (cited on pages 2, 6).

Lingxiao Zhao and Leman Akoglu (2020). "PairNorm: Tackling Oversmoothing in GNNs." *International Conference on Learning Representations*. OpenReview.net (cited on page 3).

Appendix A. Models and Datasets

Code is available at https://github.com/alanmilligan/data-normalization-gnn

For each model and dataset, we search over $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ for the learning rate and display the best results. All optimizers use a momentum parameter (β and β_1) of 0.9. Weight decay is not used for any models.

A.1. Datasets

We use the following datasets, with summary statistics provided in Table 1.

Cora & PubMed Both datasets are citation graphs. Each node represents a document, and the node features are constructed from bag-of-words representations of these documents. An edge exists between two nodes if and only if one document cites the other. The task for these datasets is to classify each node into one of the predefined document classes. We use the standard split for training, which consists of twenty samples per class (Yang et al., 2016).

Computers & Photo These datasets are part of the Amazon co-purchase graph (McAuley et al., 2015). Each node represents a product on the Amazon website, and an edge exists between two nodes if the corresponding products were frequently purchased together. Node features are derived from a bag-of-words summary of the reviews for each product. The task is to classify the nodes into different product categories (Shchur et al., 2018). We use a random train/validation/test split with a 60/20/20 ratio for training.

Roman-Empire The dataset is constructed from Wikipedia articles. Each node represents a word, and node embeddings are derived from fastText embeddings (Grave et al., 2018). An edge exists between two nodes if they appear consecutively in the text or are connected in the dependency tree of the sentence. The task is to classify the nodes into their respective grammatical roles within the sentences (Platonov et al., 2023). We use a random train/validation/test split with a 60/20/20 ratio.

Table 1: Dataset statistics. Standard evaluation metric on all these dataset is Accuracy.

Dataset	Number of Nodes	Number of Edges	Node Features Size	Classes
Cora	2,708	10,556	1,433	7
PubMed	19,717	88,648	500	3
Photo	7,487	238,162	745	8
Computers	13,381	491,722	767	10
Roman-Empire	22,662	32,927	300	18

A.2. Models

General GNNs A graph consists of a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, often denoted as $G = (\mathcal{V}, \mathcal{E})$. Nodes typically have associated features represented by a matrix $\mathbf{X} \in \mathbb{R}^{n \times d_V}$, where n is the number of nodes and d_V is the dimension of node features. Edges connect pairs of nodes, and in some cases, they also have corresponding features represented by a matrix $\mathbf{E} \in \mathbb{R}^{m \times d_E}$, where m is the number of edges and d_E is the dimension of edge features. If for each node v we consider $h_v^{(0)} = x_v$, or the initial embeddings are the node features, each layer of a message passing network can be written as

$$h_v^{(\ell)} = \phi\left(h_v^{(\ell-1)}, \bigoplus_{u \in \mathcal{N}(v)} \psi\left(h_v^{(\ell-1)}, h_u^{(\ell-1)}, e_{uv}\right)\right) \tag{1}$$

where $h_v^{(\ell)}$ is the embedding for node v in layer ℓ, ψ is a message function getting the input of the receiver node, sender node, and edge features between them as input. In many variants of the GNNs the input are a subset of these (e.g. only the sender node embeddings). The aggregation function over the neighbors $\bigoplus$ is a simple function such as sum or mean, and ϕ is a node-level update function.

GCN Graph Convolutional Networks (GCN) extend the convolution operation to the graph domain (Kipf and Welling, 2016). A layer of this network can be formulated as:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{\ell}\mathbf{W}\right),$$

where $\tilde{A} = A + I$, A is the adjacency matrix and I is the identity matrix, $\tilde{D}$ is the diagonal degree matrix for adjacency matrix $\tilde{A}$, $\mathbf{W}$ is the learnable weight matrix, and σ is an activation function. The formula can be also written as a message-passing function,

$$h_v^{\ell+1} = \sigma\left(\mathbf{W}^{\top}\sum_{u \in \mathcal{N}(v)\cup\{v\}} \frac{1}{\sqrt{d_v d_u}} h_u^{(\ell)}\right),$$

where $\mathcal{N}(v)$ are the neighbors of v in the graph after adding self-loops and d_v is the degree v.

GAT Graph Attention Networks (GAT) add the attention mechanism to the convolution. Instead of normalizing the neighbors based on their degree, they let the method learn to weight the messages from the neighbors (Veličković et al., 2018). The attention mechanism can be formulated as

$$\alpha_{vu} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_v \| \mathbf{W}h_u\right]\right)\right)}{\sum_{i \in \mathcal{N}(v)\cup\{v\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}h_v \| \mathbf{W}h_i\right]\right)\right)},$$

where $\mathbf{W}$ is a weight matrix shared between the attention mechanism and embedding mappings, $\mathbf{a}$ is a learnable vector and $\|$ is the concatenation operation. The message passing can be formulated as:

$$h_v^{(\ell+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)\cup\{v\}} \alpha_{uv}\mathbf{W}h_u^{(\ell)}\right).$$

GraphSAGE Instead of performing convolution over all neighbors, GraphSAGE samples a fixed number of neighbors for each node at each convolution operation, hoping to improve the ability to handle inductive setups and help generalize to unseen nodes during training (Hamilton et al., 2017).

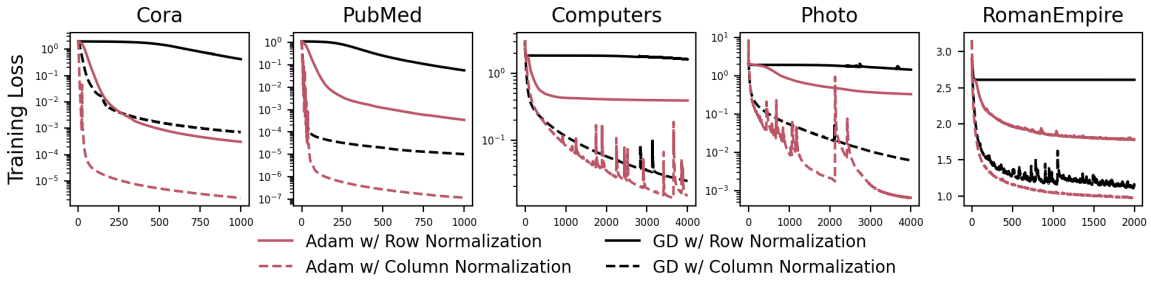
Linear Networks For the linear model, we use one layer of a GCN network, which is equivalent to a logistic regression with a graph convolution over the data with cross entropy loss,

$$\mathbf{Y} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\right),$$

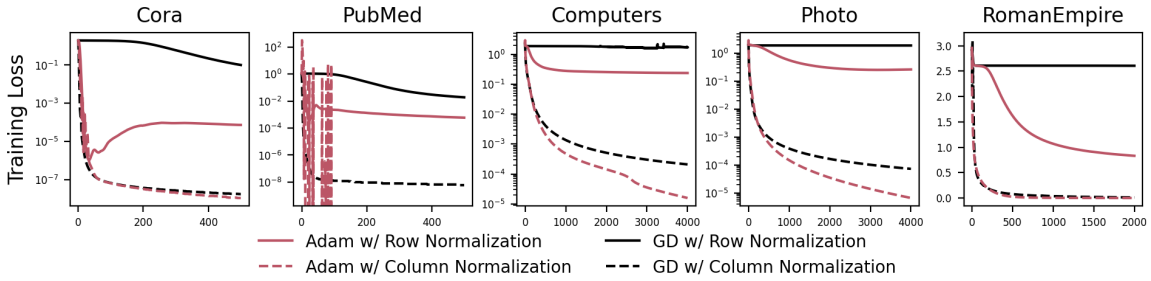
where σ is the softmax function and the graph structure matrices are as described above.

We use the standard PyTorch-Geometric (Fey and Lenssen, 2019) library implementation for GCN, GAT, and GraphSAGE models. In these models we use two layers and a hidden dimension of size 16.

Appendix B. Results with other GNNs



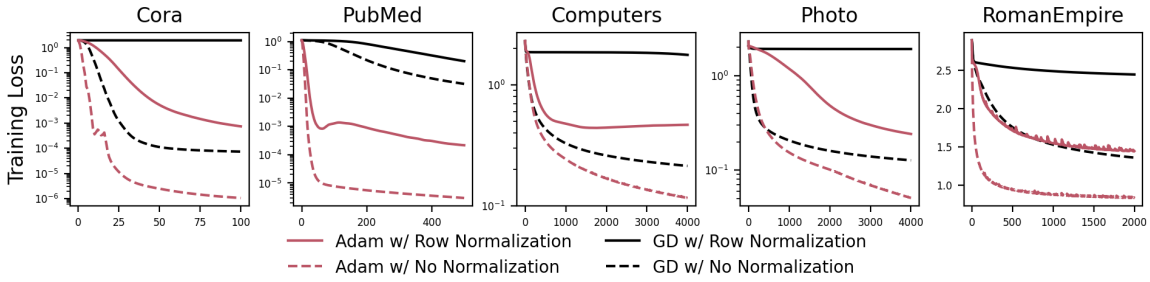
(*a*) Graph Attention Network



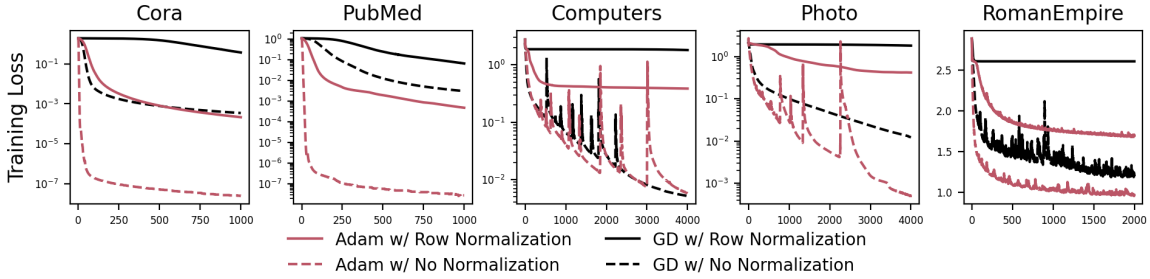
(*b*) GraphSAGE Network

Figure 4: **GD struggles on GNNs with standard row normalization, but optimization becomes much easier with columns normalization.** Training with **GD** and **Adam**, using row normalization (solid) and column normalization (dashed).

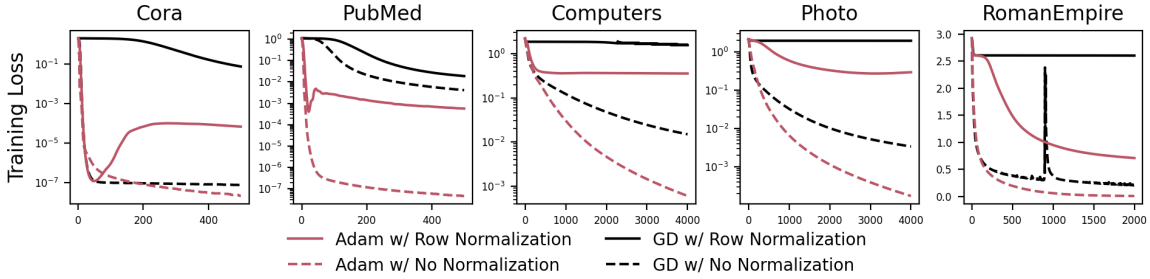
Appendix C. Row Normalization can be worse than no preprocessing



(*a*) Graph Convolutional Network



(*b*) Graph Attention Network



(*c*) GraphSAGE Network

Figure 5: **GD performs worse with standard row-normalization than without any normalization.** Training with **GD** and **Adam**, using row normalization (solid) and no normalization (dashed).