AUGMENTED NORMALIZATION: DIFFERENTIATING A GENERALIZED GEOMETRIC MEDIAN

Tyler King, Wayne Chen & Edward Gu
Department of Computer Science
Cornell University
Ithaca, NY 14850, USA
{ttk22,zc272,elg227}@cornell.edu

ABSTRACT

Graph Neural Networks (GNNs) have continued to be an effective approach to learning on graph-structured data. These networks typically involve stacking numerous layers that each aggregate neighboring node features. For GNNs that consist of numerous layers, normalization has been shown as an effective approach to improve testing performance and stabilize training. Typically, normalization methods subtract the mean value and then divide by the standard deviation, but we believe that subtracting a more noise-resistant test statistic (for example the median) would improve performance by reducing the effect of feature outliers. However, computing and using a median via traditional methods (sorting) results in sparse gradients. As a result, we introduce a new normalization method called AugNorm that subtracts a generalized geometric median which interpolates between the median and mean, and is expressed as the minima of an optimization function. We discuss various implementations of AugNorm, along with methods to compute the generalized geometric median minimum, and finally compute explicit derivatives for backpropagation. We benchmarked our algorithm against two popular normalization methods in GNNs on the MUTAG, PTC, and the OGBN-PROTEINS dataset.

1 INTRODUCTION

Graph Neural Networks (GNNs) have surged in popularity as a tool for learning on graph-structured data, finding relevant applications in a variety of subfields such as social networks Chen et al. (2018), citation networks Kipf & Welling (2017), biochemical networks Zhang et al. (2021), and computer vision Wang et al. (2018); Li et al. (2020); Cheng et al. (2020). These neural networks operate on discrete structures called graphsChen et al. (2022), and must incorporate their structure for learning and inference. This represents a new class of problems for neural networks, which mostly operate on continuous data.

Typically nodes are mapped to some vectorized representation (where this mapping is learned), and representations are then updated by some permutation-invariant aggregating function applied to each node's neighbors' representations Gilmer et al. (2017); Cai et al. (2021). Each layer in the GNN represents a single aggregation and update step. Common GNN tasks include node class prediction, edge prediction, or graph class identification. While there have been rigorous theoretical studies of GNNs that focus on expressive power Keriven & Peyré (2019); Xu et al. (2018), generalization Scarselli et al. (2018); Du et al. (2019), and extrapolation Xu et al. (2019), there is less understanding surrounding optimization of GNN performance, particularly during training Cai et al. (2021).

For deep neural networks, normalization methods (which involve shifting and scaling hidden representations) have been shown to help optimization Ioffe & Szegedy (2015); Ba et al. (2016); Ulyanov et al. (2017); Wu et al. (2019). While certain normalization schemes yield stronger performance in distinct domains, recent literature has shown that InstanceNorm converges faster than other common approaches such as BatchNorm (which normalizes the same feature across an entire batch) and LayerNorm (which normalizes input features across all nodes and their features in a single graph), and

GraphNorm (which augments InstanceNorm by adding a learnable parameter controlling the shift amount) Cai et al. (2021).

However, most normalization schemes solely focus on normalizing around the mean. An alternative is to use the median, which is resistant to data outliers and skewed data S. (2011). Historically, the median (which can be computed by minimizing the sum of absolute value error terms) is non-differentiable and thus gradients for backpropagation are not well defined. Even if we take the weak derivative of these absolute value error terms (which is analogous to Pytorch's backpropagation implementation of the median function) Paszke et al. (2019), we end up with a sparse gradient where only the element that has the value of the median has a gradient (all other variables have a gradient of 0). As a result, this zeros your gradients during backpropagation, resulting in a situation similar to vanishing gradients where model weights are not updated and thus little is learned Hochreiter (1998).

Given the difficulty of differentiating medians, we propose a novel normalization scheme (which we call Augmented Normalization, or AugNorm) that normalizes around a generalization of the geometric median by introducing an exponent denoted as ϕ, which is known to compute the median in the univariate case Minsker (2015); Haldane (1948). We then introduce a simple method to compute the minima of these functions based on Newton's method, along with explicitly deriving gradients for these functions during backpropagation by differentiating the geometric median in an argmin setting Gould et al. (2016). This method allows for fast (since the partial derivatives needed to compute backpropagation were observed during Newton's method and can thus be cached) and precise gradients for any deep learning architecture that utilizes a normalization layer. We further generalize our approach by introducing ϕ as a parameter, increasing the expressivity in some arbitrary neural network.

AugNorm can be easily composed with BatchNorm Ioffe & Szegedy (2015), InstanceNorm Ulyanov et al. (2017), LayerNorm Ba et al. (2016), GraphNorm Cai et al. (2021), and many other common normalization schemes, by simply replacing the mean with the median as a test statistic. We showcase its performance on a variety of graph datasets, benchmarking Graph Convolutional Networks (GCNs) Kipf & Welling (2017) with generic GraphNorm and BatchNorm against its variant utilizing AugNorm. Empirically, we show AugNorm achieves better validation accuracy without extensive hyperparameter fine tuning, indicating that our methods have strong generalizability.

Due to limitations on local compute resources, we are unable to benchmark with hyperparameter sweeps or utilize larger graph datasets. As a result, we focus on small-scale graphs and simple GNN architectures, showcasing that in these instances we observe stronger validation accuracy than current approaches such as GraphNorm.

2 Related Work

GraphNorm Cai et al. (2021) claims that InstanceNorm outperforms LayerNorm and BatchNorm in GNNs, explaining that the shift operation in InstanceNorm serves as a preconditioner of the graph aggregation operation. They also show that batch level statistics are noisy, thus the normalizing value in BatchNorm does not capture meaningful information. GraphNorm also further augments InstanceNorm by adding a parameter α that is multiplied by the feature's mean (across the graph). This is an important addition since the mean could capture valuable global context, which is lost if completely subtracted. In fact, Cai et al. (2021) show that for both completely regular and complete graphs, subtracting the mean results in loosing all feature representations.

Furthermore, GraphNorm's simple GCN and GIN models serve as good baselines for our model architectures, allowing us to test lightweight and simplistic models that have been shown to be extremely effective on a variety of graph datasets. Considering GraphNorm is one of the best normalization algorithms for node-level prediction tasks Cai et al. (2021); Chen et al. (2022), we benchmark our approach against vanilla GraphNorm. Also because of its low computational overhead, we also benchmarked against GraphNorm.

To compute the gradients necessary for backpropagating through our model, we utilize recent work on differentiating argmin functions Gould et al. (2016). This allows us to compute exact derivatives during backpropagation, which could not be done with Pytorch's automatic differentiation Paszke et al. (2019), which zeros too many gradients by ignoring all but 2 or 1 points. Note that using a

hill-climbing method (i.e. Newton's method), and tracking gradients across many iterations with auto-differentiation is too memory intensive.

Similar to our paper, recent work has been done on differentiating the Fréchet mean, where they utilize these derivatives to compute gradients for hyperbolic batch normalization on Riemannian manifolds Lou et al. (2020). Since the Fréchet mean is a generalization of the geometric median Fletcher et al. (2008; 2009), we note that their work to compute gradients aligns closely with our derivations.

3 GENERALIZED GEOMETRIC MEDIAN

We first introduce the concept of a generalized geometric median, which serves as a basis for our normalization scheme. For m points $x_1, x_2, \dots x_m$, where each $x_i \in \mathbb{R}$, the geometric median in the 1-dimensional case (which coincides with the median in $\mathbb{R}$) is defined as

$$\arg\min_{y \in \mathbb{R}} \sum_{i=1}^{m} |x_i - y|$$

We generalize this to the formula

$$\arg\min_{y \in \mathbb{R}} \sum_{i=1}^{m} |x_i - y|^{\phi},$$

where ϕ is some arbitrary parameter. When $\phi = 1$, our minima y coincides with the median, and when $\phi = 2$, y coincides with the mean.

Theorem 3.1. *The landscape of the generalized 1-dimensional geometric median*

$$\sum_{i=1}^{m} |x_i - y|^{\phi}$$

for $y, x_i \in \mathbb{R}$ is convex with respect to y when $\phi \geq 1$. Specifically the second partial derivative of this surface has the form

$$\frac{\partial f(x,y)}{\partial^2 y} = \sum_{i=1}^{m} \frac{(\phi^2 - \phi)|x_i - y|^{\phi}}{(x_i - y)^2}$$

which is always nonnegative.

When $\phi < 1$ there does not exist a closed form equation or algorithm that easily computes y (as there does with the median and mean). However, by theorem 3.1 (proven in A.1), we can utilize gradient descent, Newton's method, or any other hill climbing algorithm to estimate y. We further expand on these processes in section 4.

In our experiments, we restrict ϕ to be in $[1, 2]$. The generalized geometric median landscape becomes non-convex for $\phi < 1$ and also over compensates for outliers for $\phi > 2$ (thus defeating the purpose of using our approach to obtain a noise resistant value).

4 OPTIMIZING GENERALIZED GEOMETRIC MEDIAN

In order to optimize the generalized geometric median, we decided to use Newton's method for efficient calculations. This was due to it being one of the most efficient methods for finding convex minima, which our generalized geometric median was when $\phi > 1$.

While first order Taylor series approximations to a function leads to the local optimization framework of gradient descent, higher order Taylor series approximations yield faster converging algorithms. Despite this, they typically involve more overhead due to the need of inverting the Hessian Goodfellow et al. (2016), resulting in slower algorithms that are prone to getting stuck in local minima.

4.1 NEWTON'S METHOD

In this section we discuss a local optimization scheme based on the second order Taylor series approximation - called Newton's method. Because it is based on the second order approximation Newton's method has natural strengths and weaknesses when compared to gradient descent Watt et al. (2020). In summary we will see that the cumulative effect of these trade-offs is - in general - that Newton's method is especially useful for minimizing convex functions of a moderate number of inputs Kashyap (2022); Watt et al. (2020).

The second order Taylor series approximation of a single input functions $g(w)$ at a particular point v is a quadratic function of the form

$$h(w) = g(v) + \left(\frac{\mathrm{d}}{\mathrm{d}w}g(v)\right)(w-v) + \frac{1}{2}\left(\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)\right)(w-v)^2 \tag{1}$$

which is typically a better approximation of the underlying function near v than is the first order Taylor series approximation on which gradient descent is built upon. Since Newton's method takes advantage of the second derivative, it can be faster to achieve a convex function minima given that the second derivative is known and easy to compute Kashyap (2022). For example, this approach is popularly used in logistic regression Watt et al. (2020). Since we utilize this to minimize a univariate function in AugNorm, the second derivative is extremely fast to compute and to invert, resulting in our decision to utilize this algorithm over alternative optimization algorithms such as gradient descent, tabu search, or simulated annealing.

Newton's method is the local optimization algorithm produced by repeatedly taking steps that are stationary points of the second order Taylor series approximations to a function. Repeatedly iterating in this manner, at the k-th step we move to the stationary point of the quadratic approximation generated at the previous step w^{k-1} which is given as:

$$h(\mathbf{w}) = g(\mathbf{w}^{k-1}) + \nabla g(\mathbf{w}^{k-1})^T(\mathbf{w}-\mathbf{w}^{k-1}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^{k-1})^T\nabla^2 g\left(\mathbf{w}^{k-1}\right)(\mathbf{w}-\mathbf{w}^{k-1}). \tag{2}$$

A stationary point of this quadratic is given - using the first order condition for optimality - as:

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1}\nabla g(\mathbf{w}^{k-1}). \tag{3}$$

Notice for single input functions this formula reduces naturally to:

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})}. \tag{4}$$

In the case of AugNorm, we can represent each iteration of Newton's method as

$$y^k = y^{k-1} - \frac{\sum_{i=1}^m \phi\frac{|x_i-y^{k-1}|^\phi}{x_i-y^{k-1}}}{\sum_{i=1}^m \frac{(\phi^2-\phi)|x_i-y^{k-1}|^\phi}{(x_i-y^{k-1})^2}} \tag{5}$$

where x represents the features to be normalized together and y represents the minima. These derivatives are computed in the appendix in section A.1. While this is somewhat computationally expensive, there are a few nice properties. A n-digit Newton's method approximation converges typically with $\log n$ iterations (empirically, we found around 8 iterations was sufficient to reach machine precision for float16).

Furthermore, when comparing to GraphNorm (the current state-of-the-art algorithm for normalization schemes on graphs for node-level classification) we are able to utilize batching instead of instance normalizing, thus decreasing runtime with parallelization. Additionally, a component of computing the second order derivative for Newton's method is also needed for backpropagation (refer to section A.1), and thus can save computational costs by caching. Specifically, we need to compute f_Y and f_{YY}, the first and second derivative of our generalized argmin function with respect

to our minima y (which represents the numerator and denominator of our updates for y^k, respectively). While these values are approximated for in Newton's method (since the input y for f_Y and f_{YY} is not the exact global minima), they still yield low error. Furthermore, since we need both Gould et al. (2016) to compute $\frac{\partial y}{\partial x_i}$ for each x_i Gould et al. (2016), we can cache f_{YY} from forward propagation and note that f_{XY} is easily computed from f_Y.

5 PARAMETERIZING ϕ

To increase expressivity, we allow the parameter ϕ to be learned for each feature dimension. This leads to algorithm 1.

Algorithm 1 AugNorm algorithm

Input: Input nodes $\mathbf{X} \in \mathbb{R}^{m \times d}$, learnable parameters $\boldsymbol{\gamma} \in \mathbb{R}^d$, $\boldsymbol{\beta} \in \mathbb{R}^d$, $\boldsymbol{\phi} \in \mathbb{R}^d$, $\boldsymbol{\alpha} \in \mathbb{R}^d$
Intermediate outputs: $\boldsymbol{\mu} \in \mathbb{R}^d$, $\boldsymbol{\sigma^2} \in \mathbb{R}^d$, $\hat{\mathbf{x}} \in \mathbb{R}^{m \times d}$
Output: Output nodes $\mathbf{Y} \in \mathbb{R}^{m \times d}$

for each j in $1 \dots d$ **do**
  $\mu_j \leftarrow \arg\min_y \sum_{i=1}^{m} |X_{ij} - y|^{\phi_j}$
  $\sigma_j^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (X_{ij} - \alpha_j \cdot \mu_j)^2$
  for each i in $1 \dots m$ **do**
    $\hat{x}_{ij} \leftarrow (x_{ij} - \alpha_j \cdot \mu_j) \cdot (\sigma_j^2 + \epsilon)^{-1/2}$
    $Y_{ij} \leftarrow \gamma_j \cdot \hat{x}_{ij} + \beta_j$
  end for
end for

This outlines generalizes GraphNorm, but it is trivial to apply our method to BatchNorm or InstanceNorm by modifying the dimension of the input to be normalized and setting $\alpha_j = 1$, respectively.

Of course, it is also possible to fix ϕ value to a constant. We expect $\phi \in (1, 2)$ to be most effective than other potential values of ϕ.

6 BACKPROPAGATION

Training a model using an AugNorm layer requires computing gradients with respect to some chosen loss function for backpropagation. We define gradients similar to the original BatchNorm paper Ioffe & Szegedy (2015) along with their derivations in the appendix. We use the multivariate version of chain rule on computational graphs

$$\frac{\partial \ell}{\partial x_i} = \sum_{j \in N(i)} \left(\frac{\partial x_j}{\partial x_i} \right)^T \cdot \frac{\partial \ell}{\partial x_j}$$

where $\frac{\partial \ell}{\partial x_i}$ is the desired gradient with respect to some node x_i, N is a function for all descendant nodes of x_i, and $\frac{\partial x_j}{\partial x_i}$ is a Jacobian matrix. For AugNorm, our computational graph can be found in figure 1.

The only nontrivial derivation is computing a gradient across the $\arg\min$ operator (all other derivations are derived similarly to BatchNorm and can be found in Appendix A.2). A closed form expression exists and a single variable variant is proved in Gould et al. (2016). In the appendix, we provide a proof for a multi-variable variant and apply it on the above computational graph to derive the gradients in 6.1.

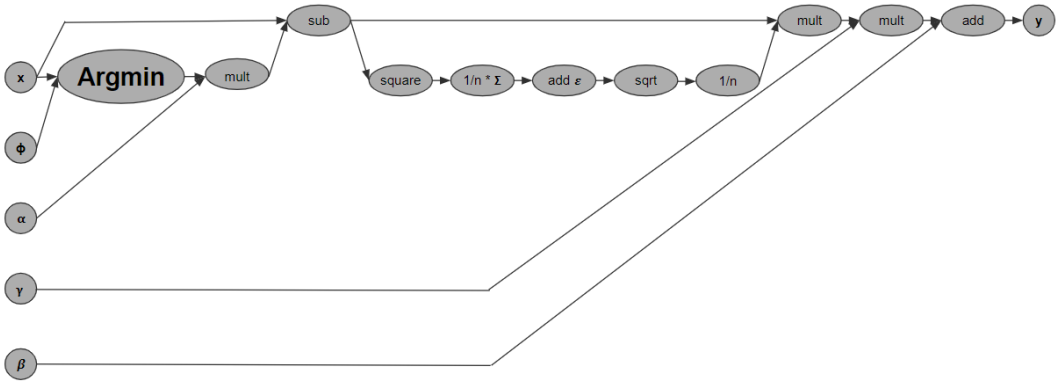

Figure 1: The computational graph of the AugNorm variant of GraphNorm. Note that the argmin step (i.e. when the generalized geometric median is taken) is enlarged to indicate its importance.

Theorem 6.1. *When backpropagating through AugNorm combined with GraphNorm with a variable ϕ on inputs $\mathbf{X} \in \mathbb{R}^{m \times d}$ (m nodes each with d feature dimensions) we get the following derivatives:*

$$\begin{aligned}
\frac{\partial \ell}{\partial \hat{x}_{ij}} &= \frac{\partial \ell}{\partial y_{ij}} \gamma_j \\
\frac{\partial \ell}{\partial \sigma_j^2} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_{ij}} \cdot -\frac{x_{ij} - \alpha_j \mu_j}{2(\sigma_j^2 + \epsilon)^{\frac{3}{2}}} \\
\frac{\partial \ell}{\partial s_{ij}} &= \frac{\partial \ell}{\partial \sigma_j^2} \frac{2}{m}(x_{ij} - \alpha_j \mu_j) + \frac{\partial \ell}{\partial \hat{x}_{ij}} \frac{1}{\sqrt{\sigma_j^2 + \epsilon}} \\
\frac{\partial \ell}{\partial \mu_j} &= -\alpha_j \sum_{i=1}^{m} \frac{\partial \ell}{\partial s_{ij}} \\
\frac{\partial \ell}{\partial x_{ij}} &= \frac{\partial \ell}{\partial s_{ij}} + \frac{\partial \ell}{\partial \mu_j} \cdot \frac{-|x_{ij} - \mu_j|^{\phi - 2}}{\sum_{i=1}^{m} |x_{ij} - \mu_j|^{\phi - 2}} \\
\frac{\partial \ell}{\partial \phi_j} &= \frac{\partial \ell}{\partial \mu_j} \cdot \frac{\sum_{i=1}^{m} \frac{|x_i - \mu_j|^{\phi}(1 + \ln |x_i - \mu_j|)}{x_i - \mu_j}}{(\phi^2 - \phi) \sum_{i=1}^{m} |x_i - \mu_j|^{\phi - 2}} \\
\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_{ij}} \hat{x}_{ij} \\
\frac{\partial \ell}{\partial \beta_j} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_{ij}}
\end{aligned}$$

Theorem 6.2. *When backpropagating through AugNorm combined with AugNorm with a fixed ϕ on inputs $\mathbf{X} \in \mathbb{R}^{m \times d}$ (m nodes each with d feature dimensions), all derivatives stay the same and the derivative with respect to ϕ_j may be ignored.*

For LayerNorm and InstanceNorm, we set $\alpha = 1$ and change the summations to reflect the features to be normalized over.

7 Experiments

Due to computational restrictions, we only trained on a simplistic GCN model that consisted of five GCN layers with a normalization layer after each GCN layer. Note that this experimental setup is identical to the setup used in GraphNorm Cai et al. (2021). As baselines, we tested generic

implementations of GraphNorm and BatchNorm. Furthermore, we test two implementations of our model: AugNorm1.5, which is an implementation of AugNorm with a fixed $\phi = 1.5$, along with DiffPhi, a differentiable ϕ implementation of AugNorm. Both AugNorm1.5 and DiffPhi were applied on top of BatchNorm.

While we could test various other values of ϕ, the compute cost required to run each 10-fold cross validation was too expensive to test numerous values. As a result, we tested several single test/train splits for values of $\phi \in \{1.25, 1.5, 1.75\}$, and reported results associated with the best performing ϕ value.

7.1 Implementation

To implement our code, we utilized a combination of Pytorch Paszke et al. (2019) and Deep Graph Library (DGL) Wang et al. (2019), implementing everything in Python. To build our custom AugNorm layers, we utilized `torch.autograd.Function`, implementing the forward pass with a custom implementation of Newton's method in Pytorch and caching these derivatives to compute the backward pass. Batching was implemented in DGL. These custom layers were then substituted in lieu of standard BatchNorm or GraphNorm layers.

Similarly, we utilize the default hyperparameters proposed in GraphNorm. This consists of a learning rate of 0.01, dropout of 0.05, 400 epochs for all biomedical datasets (which were our main focus due to their smaller scale and popularity as a GNN benchmark Yanardag & Vishwanathan (2015)), 64 features per GCN layer, and a graph pooling type of sum. Furthermore, for both BatchNorm and AugNorm, batch size was set to 128 graphs.

7.2 Results

Table 1: **Validation performance** of GCN with various normalization schemes applied to three graph classification tasks on bioinformatic datasets.

Datasets	MUTAG	PTC	PROTEINS
# graphs	188	344	1113
# classes	2	2	2
Avg # nodes	17.9	25.5	39.1
GCN+BatchNorm	89.3 ± 6.54	73.00±6.84	79.15±3.87
GCN+GraphNorm	90.38±7.13	**74.40±6.31**	79.78±3.90
GCN+AugNorm1.5	**91.00±7.06**	74.15±6.82	**80.14±3.81**
GCN+DiffPhi	90.41±6.55	72.42±6.45	79.06±4.29

By utilizing the default hyperparameters in GraphNorm, we avoid a benchmark that may be unfairly biased towards AugNorm. Furthermore, since we do not explore hyperparameter tuning, we report best averaged validation accuracy (at checkpoints) over a seeded 10-fold cross validation split. This was to help minimize variance across trials, along with providing a fair comparison between AugNorm and GraphNorm. Note that this was also a reported benchmark value in GraphNorm Cai et al. (2021). While we would like to test on specialized datasets such as those with low homophily or low expansion, most of these datasets were too large to reasonable benchmark on a local laptop, and thus we tended to focus solely on the selected datasets. Given our performance benchmarks, we felt that we adequately showcased results that indicated the advantage of our approach.

Typically GNNs utilize testing performance after a complex hyperparameter tuning process, but unfortunately we do not have the computational ability to test this. However, 10-fold cross validation is often used to determine optimal hyperparameters (since it stabilizes performance on a dataset), and thus we believe it serves as a fair comparison between various normalization algorithms.

We do not benchmark against classical heuristics that are popular for graphs since numerous past papers have shown the superiority of graph neural network approaches over these aforementioned heuristics in the particular datasets we test on Cai et al. (2021); Xu et al. (2019).

Our results show that on small-scale graph datasets, we outperform BatchNorm (which our algorithm was based on) and achieve performance comparable and even better than GraphNorm with the fixed ϕ implementation.

Unfortunately, the differentiable ϕ implementation yielded poor performance, and we suspect this is due to poor performance of Newton's method at low ϕ values. When ϕ is sufficiently low, the difference between our initial estimate (which is the mean) and the true minima may be large enough that Newton's method has poor convergence properties.

While we did not observe this for any of our reported results (since we always utilized a relatively large batch size of 128), note that there is the possibility for the differentiable ϕ implementation to have $\phi \leq 1$. For small batch sizes, the sparse number of features can lead to more aggressive gradients that lead to this phenomenon, which results in our generalized geometric median function not being well-defined. This typically results in `NaN` being thrown during backpropagation, stopping the model from learning.

7.3 Training and Testing Convergence

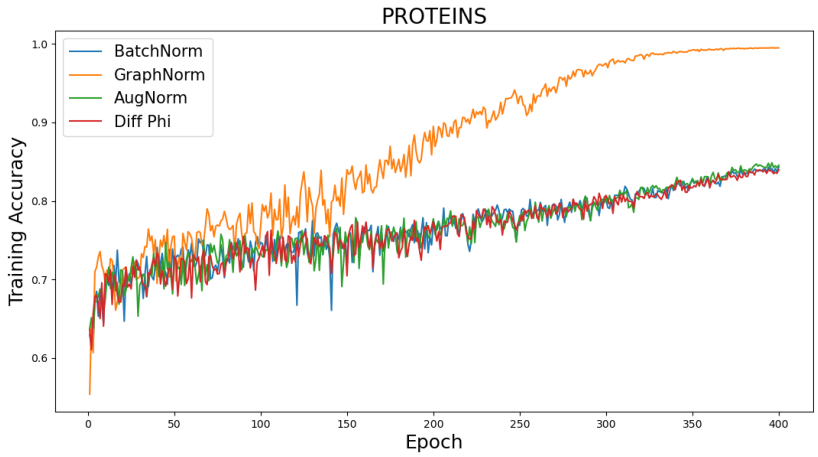

Figure 2: Averaged training accuracy over all 10 folds during cross validation. While batched normalization methods did not reach 100% training accuracy, testing performance was still stronger in AugNorm.

Despite the slower increase in training accuracy of AugNorm (and batched normalization methods in general), we note that we still achieved better testing performance. This is indicative of greater generalizability in AugNorm as compared to GraphNorm, where despite the lack of training convergence we achieved better testing results.

While we could have trained AugNorm/BatchNorm/DiffPhi for a greater number of epochs (since batched methods did not reach 100% train accuracy), we felt this would not yield a fair comparison to the results reported in GraphNorm Cai et al. (2021) where they trained all models for exactly 400 epochs (this additionally helps alleviate concerns with computational limitations). We believe with greater training epochs and fine-tuned augnorm hyperparameters, our performance will be even more promising.

8 Conclusions and Future Work

In this paper, we introduce a new normalization scheme for GNNs called AugNorm. We introduce a novel test statistic that we call the generalized geometric median, showcasing a way to compute the exact minimum of this function using Newton's method. We then compute explicit gradients for backpropagation with respect to the various parameters in AugNorm. These components allow us to construct two distinct implementations of AugNorm: one with a differentiable ϕ and one without. Finally, we benchmark our algorithm against common normalization methods such as GraphNorm and BatchNorm, outperforming both BatchNorm (which our algorithm was based on) and GraphNorm (which currently is the best performing algorithm for node-level classification), along with showing that our algorithm has greater generalizability than GraphNorm from some train dataset.

Given the computational constraints on our current work, we hope to expand to large datasets in the future, where we believe AugNorm would perform even better against state-of-the-art approaches. Furthermore, we hope to more rigorously benchmark our approach with hyperparameter tuning, along with refactoring our code to allow for greater parallelism and efficiency.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

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 Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1204–1215. PMLR, 2021. URL http://proceedings.mlr.press/v139/cai21e.html.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *CoRR*, abs/1801.10247, 2018. URL http://arxiv.org/abs/1801.10247.

Yihao Chen, Xin Tang, Xianbiao Qi, Chun-Guang Li, and Rong Xiao. Learning graph normalization for graph neural networks. *Neurocomputing*, 493:613–625, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.01.003. URL https://www.sciencedirect.com/science/article/pii/S0925231222000030.

Ke Cheng, Yifan Zhang, Xiangyu He, Weihan Chen, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with shift graph convolutional network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 180–189, 2020. doi: 10.1109/CVPR42600.2020.00026.

Simon S. Du, Kangcheng Hou, Barnabás Póczos, Ruslan Salakhutdinov, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *CoRR*, abs/1905.13192, 2019. URL http://arxiv.org/abs/1905.13192.

P. Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi. Robust statistics on riemannian manifolds via the geometric median. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008. doi: 10.1109/CVPR.2008.4587747.

P. Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi. The geometric median on riemannian manifolds with application to robust atlas estimation. *NeuroImage*, 45(1):S143–S152, March 2009. doi: 10.1016/j.neuroimage.2008.10.052. URL https://doi.org/10.1016/j.neuroimage.2008.10.052.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *CoRR*, abs/1607.05447, 2016. URL http://arxiv.org/abs/1607.05447.

J. B. S. Haldane. Note on the median of a multivariate distribution. *Biometrika*, 35(3-4):414–417, 1948. doi: 10.1093/biomet/35.3-4.414. URL https://doi.org/10.1093/biomet/35.3-4.414.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06 (02):107–116, April 1998. doi: 10.1142/s0218488598000094. URL https://doi.org/10.1142/s0218488598000094.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Rohan V Kashyap. A survey of deep learning optimizers-first and second order methods, 2022.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *CoRR*, abs/1905.04943, 2019. URL http://arxiv.org/abs/1905.04943.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Xia Li, Yibo Yang, Qijie Zhao, Tiancheng Shen, Zhouchen Lin, and Hong Liu. Spatial pyramid based graph reasoning for semantic segmentation. *CoRR*, abs/2003.10211, 2020. URL https://arxiv.org/abs/2003.10211.

Aaron Lou, Isay Katsman, Qingxuan Jiang, Serge Belongie, Ser-Nam Lim, and Christopher De Sa. Differentiating through the fréchet mean. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Stanislav Minsker. Geometric median and robust estimation in banach spaces. *Bernoulli*, 21 (4), November 2015. doi: 10.3150/14-bej645. URL https://doi.org/10.3150/14-bej645.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Manikandan S. Measures of central tendency: Median and mode. *Journal of Pharmacology and Pharmacotherapeutics*, 2(3):214–215, September 2011. doi: 10.4103/0976-500x.83300. URL https://doi.org/10.4103/0976-500x.83300.

Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.08.010. URL https://www.sciencedirect.com/science/article/pii/S0893608018302363.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *CoRR*, abs/1909.01315, 2019. URL http://arxiv.org/abs/1909.01315.

Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. *CoRR*, abs/1803.08035, 2018. URL http://arxiv.org/abs/1803.08035.

Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms, and Applications*. Cambridge University Press, 2 edition, 2020. doi: 10.1017/9781108690935.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *CoRR*, abs/1905.13211, 2019. URL http://arxiv.org/abs/1905.13211.

Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783417. URL https://doi.org/10.1145/2783258.2783417.

Xiao-Meng Zhang, Li Liang, Lin Liu, and Ming-Jing Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in Genetics*, 12, July 2021. doi: 10.3389/fgene.2021.690049. URL https://doi.org/10.3389/fgene.2021.690049.

A Appendix

A.1 Proof of Theorem 3.1

We prove that the second derivative of some generalized 1-dimensional geometric median

$$g(y) = \sum_{i=1}^{m} |x_i - y|^{\phi}$$

will always be convex for $\phi \geq 1$ by showing that the second derivative of $g(y)$ with respect to y is strictly non-negative. The first derivative of $g(y)$ is

$$\frac{\partial g}{\partial y} = \sum_{i=1}^{m} \phi |x_i - y|^{\phi - 1} \cdot \frac{x_i - y}{|x_i - y|},$$

however,

$$\frac{x_i - y}{|x_i - y|} = \frac{|x_i - y|}{x_i - y}$$

because $(x_i - y)^2 = |x_i - y|^2$. This gives us

$$\begin{aligned} \frac{\partial g}{\partial y} &= \sum_{i=1}^{m} \phi |x_i - y|^{\phi - 1} \cdot \frac{|x_i - y|}{x_i - y} \\ &= \sum_{i=1}^{m} \phi \frac{|x_i - y|^{\phi}}{x_i - y}. \end{aligned}$$

The second derivative of $g(y)$ w.r.t. y is

$$\begin{aligned}\frac{\partial^2 g}{\partial y^2} &= \frac{\partial g}{\partial y}\left(\sum_{i=1}^{m}\phi\frac{|x_i-y|^\phi}{x_i-y}\right)\\ &= \sum_{i=1}^{m}\phi\left(\frac{\phi|x_i-y|^\phi}{(x_i-y)^2}-\frac{|x_i-y|^\phi}{(x_i-y)^2}\right)\\ &= \sum_{i=1}^{m}\frac{(\phi^2-\phi)|x_i-y|^\phi}{(x_i-y)^2}\end{aligned}$$

which is strictly positive when $\phi^2-\phi$ is positive, which occurs when $\phi \geq 1$. Note that while $\phi < 0$ looks convex, the landscape of the 1-dimensional geometric median is not well-defined at points where $y = x_i$ when $\phi < 0$.

A.2 Proof of Theorem 6.1

We use M_β when referring to the minima along the generalized geometric median function and y as the output to the normalization scheme (similar to the original batch normalization paper where $y = \gamma x + \beta$) Ioffe & Szegedy (2015).

Starting with the easiest derivations,

$$\begin{aligned}\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i}\frac{\partial y_i}{\partial \hat{x}_i}\\ &= \frac{\partial \ell}{\partial y_i}\cdot\gamma\end{aligned}$$

The derivatives of loss with respect to tunable parameters also do not change:

$$\begin{aligned}\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\frac{\partial y_i}{\partial \beta}\\ &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\frac{\partial y_i}{\partial \gamma}\\ &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\hat{x}_i\end{aligned}$$

Next, we compute $\frac{\partial \ell}{\partial \sigma_\beta^2}$ and $\frac{\partial \ell}{\partial M_\beta}$ as follows.

$$\begin{aligned}\frac{\partial \ell}{\partial \sigma_\beta^2} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial \sigma_\beta^2}\\ &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\cdot(x_i-M_\beta)\cdot\frac{-1}{2}(\sigma_\beta^2+\epsilon)^{-3/2}\end{aligned}$$

$$\begin{aligned}\frac{\partial \ell}{\partial M_\beta} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial M_\beta}+\frac{\partial \ell}{\partial \sigma_\beta^2}\frac{\partial \sigma_\beta^2}{\partial M_\beta}\\ &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\frac{-1}{\sqrt{\sigma_\beta^2+\epsilon}}+\frac{\partial \ell}{\partial \sigma_\beta^2}\frac{2(x_i-M_\beta)}{m}\end{aligned}$$

These equations remain mostly the same for both classical batch normalization and median batch normalization, with the small caveat that the mean μ_β is replaced with the median M_β. These prior computations also allow us to calculate $\frac{\partial \ell}{\partial x_i}$.

Again from Ioffe & Szegedy (2015), we know that

$$\begin{aligned}\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial \ell}{\partial \sigma_\beta^2}\frac{\partial \sigma_\beta^2}{\partial x_i} + \frac{\partial \ell}{\partial M_\beta}\frac{\partial M_\beta}{\partial x_i} \\ &= \frac{\partial \ell}{\partial \hat{x}_i}\frac{1}{\sqrt{\sigma_\beta^2+\epsilon}} + \frac{\partial \ell}{\partial \sigma_\beta^2}\frac{2(x_i - M_\beta)}{m} + \frac{\partial \ell}{\partial M_\beta}\frac{\partial M_\beta}{\partial x_i}\end{aligned}$$

A.2.1 Differentiating through the Generalized Geometric Median

Lemma A.1. *Let $f : \mathbb{R}^m \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $g(\mathbf{x}) = \arg\min_y f(\mathbf{x}, y)$, $\mathbf{x} = (x_1, \ldots, x_m)$, with $x_1, \ldots, x_m$ all independent, and $y \in \mathbb{R}$. Then:*

$$\frac{\partial g(\mathbf{x})}{\partial x_i} = -\frac{f_{X_i Y}(\mathbf{x}, g(\mathbf{x}))}{f_{YY}(\mathbf{x}, g(\mathbf{x}))}$$

where $f_{X_i Y} = \frac{\partial^2 f}{\partial x_i \partial y}$ and $f_{YY} = \frac{\partial^2 f}{\partial y^2}$

Proof. We follow a similar proof structure as Gould et al. (2016) to derive $\frac{\partial g(\mathbf{x})}{\partial x_i}$ To begin, we have

$$\frac{\partial f(\mathbf{x}, y)}{\partial y}\Big|_{y=g(\mathbf{x})} = 0$$

so,

$$\frac{d}{dx_i}\frac{\partial f(\mathbf{x}, g(\mathbf{x}))}{\partial y} = 0.$$

Via chain rule, we also have:

$$\begin{aligned}\frac{d}{dx_i}\frac{\partial f(\mathbf{x}, g(\mathbf{x}))}{\partial y} &= \frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial x_i \partial y}\cdot\frac{dx_i}{dx_i} + \sum_{j\neq i, 1\leq j\leq m}\left(\frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial x_j \partial y}\cdot\frac{dx_j}{dx_i}\right) + \frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial^2 y}\cdot\frac{\partial g(\mathbf{x})}{\partial x_i} \\ &= \frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial x_i \partial y} + \frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial^2 y}\cdot\frac{\partial g(\mathbf{x})}{\partial x_i}\end{aligned}$$

Equating this expression to 0 and rewriting in terms of $\frac{\partial g(\mathbf{x})}{\partial x_i}$ yields:

$$\begin{aligned}\frac{\partial g(\mathbf{x})}{\partial x_i} &= -\left(\frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial^2 y}\right)^{-1}\frac{\partial^2 f(\mathbf{x}, g(\mathbf{x}))}{\partial x_i \partial y} \\ &= -\frac{f_{X_i Y}(\mathbf{x}, g(\mathbf{x}))}{f_{YY}(\mathbf{x}, g(\mathbf{x}))}\end{aligned}$$

□

To compute $\frac{\partial \mu_j}{\partial x_{ij}}$ and $\frac{\partial \mu_j}{\partial \phi_j}$, we use Lemma A.1. We define a function $g(\mathbf{x}, \phi)$ for $\mathbf{x} \in \mathbb{R}^m$, $\phi \in \mathbb{R}$ such that

$$g(\mathbf{x}, \phi) = \arg\min_\mu f(\mathbf{x}, \phi, \mu)$$

where

$$f(\mathbf{x}, \phi, y) = \sum_{i=1}^m |x_i - y|^\phi$$

and $y \in \mathbb{R}$. Note than when $\phi = 1$ or $\phi = 2$, analytical solutions exist and they are represented by the median and mean, respectively.
First, we compute $f_{ZZ}(\mathbf{x}, \phi, g(\mathbf{x}, \phi)) \in \mathbb{R}$, which is known from Theorem 3.1:

$$\sum_{i=1}^m \frac{(\phi^2 - \phi)|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}$$

Then we compute $f_{X_i Z}(\mathbf{x}, \phi, g(\mathbf{x}, \phi)) \in \mathbb{R}$: Starting with

$$f_Z = \sum_{i=1}^{m} \phi \frac{|x_i - g(\mathbf{x}, \phi)|^\phi}{x_i - g(\mathbf{x}, \phi)},$$

we then obtain:

$$\begin{aligned} f_{X_i Z} &= \phi \left(\frac{\phi |x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))(g(\mathbf{x}, \phi) - x_i)} - \frac{|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))(g(\mathbf{x}, \phi) - x_i)} \right) \\ &= \frac{(\phi - \phi^2)|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2} \end{aligned}$$

. Finally we have $\frac{\partial g(\mathbf{x}, \phi)}{x_i}$:

$$\begin{aligned} \frac{\partial g(\mathbf{x}, \phi)}{\partial x_i} &= \frac{\frac{(\phi - \phi^2)|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}}{\sum_{i=1}^{m} \frac{(\phi^2 - \phi)|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}} \\ &= \frac{\frac{-|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}}{\sum_{i=1}^{m} \frac{|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}} \\ &= \frac{-|x_i - g(\mathbf{x}, \phi)|^{\phi - 2}}{\sum_{i=1}^{m} |x_i - g(\mathbf{x}, \phi)|^{\phi - 2}} \end{aligned}$$

For $\frac{\partial \mu_j}{\partial \phi_j}$, we first have

$$\frac{f_{YZ}}{f_{ZZ}} = \sum_{i=1}^{m} \frac{|x_i - g(\mathbf{x}, \phi)|^\phi (1 + \ln |x_i - g(\mathbf{x}, \phi)|)}{x_i - g(\mathbf{x}, \phi)}$$

so

$$\begin{aligned} \frac{\partial g(\mathbf{x}, \phi)}{\partial \phi} &= \frac{\sum_{i=1}^{m} \frac{|x_i - g(\mathbf{x}, \phi)|^\phi (1 + \ln |x_i - g(\mathbf{x}, \phi)|)}{x_i - g(\mathbf{x}, \phi)}}{\sum_{i=1}^{m} \frac{(\phi^2 - \phi)|x_i - g(\mathbf{x}, \phi)|^\phi}{(x_i - g(\mathbf{x}, \phi))^2}} \\ &= \frac{\sum_{i=1}^{m} \frac{|x_i - g(\mathbf{x}, \phi)|^\phi (1 + \ln |x_i - g(\mathbf{x}, \phi)|)}{x_i - g(\mathbf{x}, \phi)}}{(\phi^2 - \phi) \sum_{i=1}^{m} |x_i - g(\mathbf{x}, \phi)|^{\phi - 2}} \end{aligned}$$