GRAPH FEATURE NETWORKS

Ting Chen
UCLA
tingchen@cs.ucla.edu

Song Bian
Zhejiang University
songbian@zju.edu.cn

Yizhou Sun
UCLA
yzsun@cs.ucla.edu

ABSTRACT

Graph Neural Nets (GNNs) have received increasing attentions, partially due to their superior performance in many node and graph classification tasks. However, there is a lack of understanding on what they are learning and how sophisticated the learned graph functions are. In this work, we propose Graph Feature Network (GFN), a simple lightweight neural net defined on a set of graph augmented features. GFN can be used to approximate GNNs with improved efficiency, and serve as a tool to access and understand the graph functions learned. To our surprise, we find that, despite its simplicity, GFN could match or exceed the best accuracies produced by recently proposed GNNs on common graph classification benchmarks. Our results seemingly suggest that (1) GFN may serve as an efficient and effective alternative to GNNs, and (2) existing GNNs may not have learned more sophisticated graph functions on these benchmarks. Furthermore, we observe that when treating images as pixels defined in graphs/grids, the same type of GNNs can outperform GFN, indicating GNNs may have learned more complex graph functions in that case.[1]

1 INTRODUCTION

Recent years have seen increasing attentions on Graph Neural Nets (GNNs) (Scarselli et al., 2009; Li et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2016), which have achieved superior performance in many graph tasks, such as node classification (Kipf & Welling, 2016; Wu et al., 2019) and graph classification (Simonovsky & Komodakis, 2017; Xinyi & Chen, 2019). Different from traditional neural networks that are defined on regular structures such as sequences or images, graphs provide more general abstraction of data, which subsume regular structures as special cases. The power of GNNs is that they can directly define learnable compositional function on (arbitrary) graphs, thus extending classic networks (e.g. Convolutional Neural Nets, Recurrent Neural Nets) to more irregular and general domains.

Despite their success, it is unclear what GNNs are learning, and how complex the learned graph functions are. It is shown in (Zeiler & Fergus, 2014) that traditional CNNs used in image recognition have learned complex hierarchical and compositional features, and that deep non-linear computation is beneficial He et al. (2016). Is it also the case when applying GNNs to common graph problems? Recently, Wu et al. (2019) have shown that, for common node classification benchmarks, non-linearity can be removed in GNNs without suffering much loss of performance. The resulting linear GNNs collapse into a logistic regression on propagated graph features. This raises doubts on the necessity of complex GNNs, which require much more expensive computation, for node classification benchmarks. Here we take a step further, and examine necessity of complex GNNs on more challenging graph classification benchmarks (Yanardag & Vishwanathan, 2015; Zhang et al., 2018; Xinyi & Chen, 2019).

In this work, we propose Graph Feature Network (GFN), a simple lightweight neural net defined on a set of graph augmented features. Unlike GNNs, which learn a multi-step neighbor aggregation function on graphs (Dai et al., 2016; Gilmer et al., 2017), the GFN only utilizes graphs in constructing its input features. It first augments nodes with graph structural and propagated features, and then learns a neural net directly on the set of nodes (i.e. a bag of graph pre-processed feature vectors). Therefore, GFN is lightweight, and can be used to approximate GNNs with improved efficiency.

[1] Our code is available at: https://github.com/chentingpc/gfn.

Empirically, we evaluate the model on common graph classification benchmarks (Yanardag & Vishwanathan, 2015; Zhang et al., 2018; Xinyi & Chen, 2019), and find that GFN can achieve results on par with recently proposed GNNs. We interpret these results in two ways: (1) GFN is a good approximation to GNNs with improved efficiency, and (2) the examined GNNs may not have learned more sophisticated patterns for existing benchmarks. Interestingly, when treating images as pixels defined in graphs/grids, the same type of GNNs can outperform GFN, suggesting GNNs may learn more complex graph functions in that case.

2 GRAPH FEATURE NETWORK

The proposed GFN is a neural set function defined on a set of graph augmented features.

Graph augmented features. A GFN can leverage all types of graph augmented features based on $\mathcal{G}$. In this work, we consider two categories as follows: (1) Graph structural/topological features. These are the features related to the intrinsic graph structures which do not rely on node attributes, such as node degrees, node centrality scores[2]; (2) Graph propagated features, which leverage the graph as a medium to propagate node attributes. More specifically, we combine both features, and construct a multi-scale graph augmented features as follows.

$$X^G = \{\boldsymbol{d}, X, \tilde{A}^1 X, \tilde{A}^2 X, \cdots, \tilde{A}^K X\},$$

where $\tilde{A} = D^{-1/2}(A + \epsilon I)D^{-1/2}$ is the normalized adjacency matrix, and $\boldsymbol{d} \in \mathbb{R}^{n\times 1}$ is the degree vector for all nodes. In practice, many graph classification problems have categorical/one-hot node attributes (Yanardag & Vishwanathan, 2015; Kipf & Welling, 2016), which make linear propagation much more sensible. To deal with propagation of continuous features such as node degrees, discretization can be leveraged Xu et al. (2019).

Neural set function. With graph augmented features X^G, GFN discards the graph structures and simply treats the attributed graph as a set of augmented nodes, and learns a neural net on this set. Motivated by the general form of a permutation-invariant set function shown in Zaheer et al. (2017), we define our neural set function for GFN as follows.

$$\text{GFN}(X, G) = \rho\Big(\sum_{v\in\mathcal{V}} \phi\Big(X_v^G\Big)\Big).$$

Both $\phi(\cdot)$ and $\rho(\cdot)$ are parameterized by neural networks. We instantiate this general form of neural set function analogous to that of GNN variants, such as GCN and GIN, in order to facilitate an easier comparison between GFN and GNNs. Concretely, we parameterize $\phi(\cdot)$ function with a multi-layer perceptrons (MLPs), i.e. $\phi(x) = \sigma(\cdots\sigma(x^T W)\cdots)$. Noted that a single layer of $\phi(\cdot)$ is similar to a graph convolution layer $H^{(t+1)} = \sigma(\tilde{A}H^{(t)}W^{(t)})$ with adjacency matrix $\tilde{A}$ replaced by identity matrix I, thus $\phi(x)$ could also be considered as an 1×1 convolution. As for the function $\rho(\cdot)$, we parameterize it with another MLP (a.k.a. fully connected layers in this case), which is also used for the readout function in GNNs.

Computation efficiency. GFN provides a way to approximate GNN with less computation overheads, especially during the training process. Since the graph augmented features can be pre-computed before training even starts, the graph structures are not involved in the iterative training process. This brings the following several advantages. Firstly, since there is no neighbor propagation step in GFN, it reduces computational complexity. For example, compare a single layer feature transformation function in GFN, i.e. $\sigma(HW)$, and a neighbor aggregation function in GCN, i.e. $\sigma(\tilde{A}HW)$. Secondly, since graph augmented features of different scales are readily available from the input layer, GFN can leverage them much earlier, thus may require less transformation layers. Lastly, it also eases the implementation related overhead, since the neighbor aggregation operation in graphs typically implemented by sparse matrix operation.

3 EXPERIMENTS

We conduct experiments on common graph classification benchmarks (Yanardag & Vishwanathan, 2015; Xinyi & Chen, 2019; Xu et al., 2019), which involves two biological and social graphs. We

[2]We do not leverage node centrality features in this work as those can be expensive to calculate during inference time.

Algorithm	MUTAG	NCI1	PROTEINS	D&D	ENZYMES	**Average**
WL	82.05±0.36	82.19±0.18	74.68±0.49	**79.78±0.36**	52.22±1.26	74.18
AWE	87.87±9.76	-	-	71.51±4.02	35.77±5.93	-
DGK	87.44±2.72	80.31±0.46	75.68±0.54	73.50±1.01	53.43±0.91	74.07
PSCN	88.95±4.37	76.34±1.68	75.00±2.51	76.27±2.64	-	-
DGCNN	85.83±1.66	74.44±0.47	75.54±0.94	79.37±0.94	51.00±7.29	73.24
CapsGNN	86.67±6.88	78.35±1.55	76.28±3.63	75.38±4.17	54.67±5.67	74.27
GIN	89.40±5.60	82.70±1.70	76.20±2.80	-	-	-
GCN	87.20 ± 5.11	**83.65 ± 1.69**	75.65 ± 3.24	79.12 ± 3.07	66.50 ± 6.91	78.42
GFN	**90.84 ± 7.22**	82.77 ± 1.49	76.46 ± 4.06	78.78 ± 3.49	**70.17 ± 5.58**	**79.80**
GFN-light	89.89 ± 7.14	81.43 ± 1.65	**77.44 ± 3.77**	78.62 ± 5.43	69.50 ± 7.37	79.38

Table 1: Test accuracies (%) for biological graphs. The best results per dataset and in average are highlighted. - means the results are not available for a particular dataset.

Algorithm	COLLAB	IMDB-B	IMDB-M	RE-M5K	RE-M12K	**Average**
WL	79.02±1.77	73.40±4.63	49.33±4.75	49.44±2.36	38.18±1.30	57.87
AWE	73.93±1.94	74.45±5.83	51.54±3.61	50.46±1.91	39.20±2.09	57.92
DGK	73.09±0.25	66.96±0.56	44.55±0.52	41.27±0.18	32.22±0.10	51.62
PSCN	72.60±2.15	71.00±2.29	45.23±2.84	49.10±0.70	41.32±0.42	55.85
DGCNN	73.76±0.49	70.03±0.86	47.83±0.85	48.70±4.54	-	-
CapsGNN	79.64±0.91	73.10±4.83	50.27±2.65	52.88±1.48	46.62±1.90	60.50
GIN	80.20±1.90	**75.10±5.10**	**52.30±2.80**	57.50±1.50	-	-
GCN	**81.72 ± 1.64**	73.30 ± 5.29	51.20 ± 5.13	56.81 ± 2.37	49.31 ± 1.44	62.47
GFN	81.50 ± 2.42	73.00 ± 4.35	51.80 ± 5.16	**57.59 ± 2.40**	49.43 ± 1.36	**62.66**
GFN-light	81.34 ± 1.73	73.00 ± 4.29	51.20 ± 5.71	57.11 ± 1.46	**49.75 ± 1.19**	62.48

Table 2: Test accuracies (%) for social graphs. The best results per dataset and in average are highlighted. - means the results are not available for a particular dataset.

compare to graph kennel based and graph neural network based baselines () and follow the 10-fold cross-validation setting as in (). We also implement GCN with four graph convolutional layers to mimic the architecture of GFN as in an "AB testing" setting. GFN-light is GFN with a single feature transformation layer. The details of datasets, baseline and our model configurations can be found in the appendix.

Biological and social datasets. Table 1 and 2 show the results of different methods in both biological and social datasets. It is worth noting that in both datasets, GFN achieves similar performances with with our GCN, and match or exceed existing state-of-the-art results on multiple datasets. This suggests that GFN could very well approximate the existing GCNs in these graph benchmark datasets, and also unnecessity of non-linear graph filtering for these benchmarks.

MNIST pixel graphs. We report the accuracies under different total receptive field sizes (the number of hops a pixel could condition its computation on), as the model benefits from larger receptive fields. Results in Table 3 show that GCN significantly outperforms GFN in all three receptive filed sizes. This indicates that non-linear graph filtering is necessary for this graph dataset. Noted that our GNN accuracy is not directly comparable to traditional CNN's, as our GNN does not distinguish the direction in its parameterization, and a global sum pooling does not distinguish spatial information. For the context, when we use coordinates as features both GCN and GFN could achieve nearly 99% accuracy.

Table 3: Results on MNIST graphs.

Receptive size	GCN	GFN
3	**91.47**	87.73
5	**95.16**	91.83
7	**96.14**	92.68

Training time comparisons. We compare the training time of our GCN and the proposed GFNs. Figure 1 shows that a significant speedup by utilizing GFN compared to GCN, especially when there are more edges such as in COLLAB dataset. Also since our GFN could work with less number of transformation layers, GFN-light achieve better speedup.

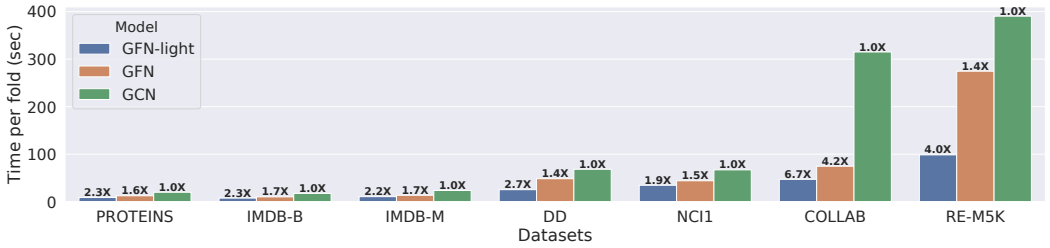


Figure 1: Training time comparisons. The annotation, e.g. $1.0\times$, denotes speedup compared to GCN.

Table 4: Accuracies (%) under various augmented features. Averaged results over multiple datasets are shown here. $A^{1,2,3}X$ is abbreviated for A^1X, A^2X, A^3X, and default node feature X is always used (if available) but not displayed to reduce clutter. Best results per row/block are highlighted.

Graphs	Model	None	$\boldsymbol{d}$	A^1X	$A^{1,2}X$	$A^{1,2,3}X$	$\boldsymbol{d}, A^1X$	$\boldsymbol{d}, A^{1,2}X$	$\boldsymbol{d}, A^{1,2,3}X$
Bio.	GCN	**78.52**	78.51	78.23	78.24	**78.68**	79.10	79.26	**79.69**
	GFN	**76.27**	77.84	78.78	79.09	**79.17**	78.71	**79.21**	79.13
Soical	GCN	**34.02**	**62.35**	59.20	60.39	60.28	62.45	62.71	**62.77**
	GFN	**30.45**	**60.79**	58.04	59.83	60.09	62.47	**62.63**	62.60

Table 5: Accuracies (%) under different number of Conv. layers. Flat denotes the collapsed GFN model into a logistic regression (no non-linear layer is used for the set function).

		Flat	1	2	3	4	5
Bio.	GCN	-	77.17	**79.38**	78.86	78.75	78.21
	GFN	69.54	79.59	79.77	**79.78**	78.99	78.14
Soical	GCN	-	60.69	62.12	62.37	**62.70**	62.46
	GFN	58.41	62.70	**62.88**	62.81	62.80	62.60

Node features. To better understand the impact of features, we test both models with different input node features. Table 4 shows that (1) graph features are important for both models, especially for GFN, and (2) degree of a node itself can provide a very significant improvement, and (3) with multi-scale features they can achieve the best results. More detailed results (per dataset) can be found in appendix.

Architecture depth. We test the effectiveness of the multiple feature transformation layers. We also test the necessity of non-linear set function by constructing GFN-flat, which contains no feature transform layer, but just the global sum pooling followed by a single fully connected layer (mimicking logistic regression). Table 5 shows that (1) GCN benefits from multiple GCN layers with a significant diminishing return, (2) GFN with single feature transformation layer could work pretty well already, likely due to the availability of multi-scale input node features, which otherwise require multiple GCN layers to obtain, and (3) by collapsing GFN into a logistic regression model the performance degenerates significantly. This is in contrast to the finding of Wu et al. (2019) in node classification benchmarks that suggests that logistic regression or a single fully connected layer is enough.

4 Discussions

In this work, we propose Graph Feature Network, a simple and lightweight neural set function based on graph augmented features. GFN well approximates GNNs on graph classification benchmarks with improved efficiency. Our results seemly suggested GNNs may not have learned more complex neighbor aggregation functions than graph augmented features in GFN for common graph classification benchmarks. This raises doubts on the necessity of non-linear graph filtering for these benchmarks. It is possible that GNNs can be improved to learn more sophisticated patterns that require a non-linear graph filtering. However, it is also likely that tested common benchmark datasets are not sufficiently differentiating, thus a linear graph filtering is powerful enough. Superior performance of GNNs for image as graph indicates that graphs constructed from raw visual signals may require more complex neighbor aggregation.

ACKNOWLEDGEMENTS

We would like to thank Yunsheng Bai and Zifeng Kang for their help in a related project prior to this work. This work is partially supported by NSF III-1705169, NSF CAREER Award 1741634, and Amazon Research Award.

REFERENCES

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pp. 2702–2711, 2016.

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

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

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. *arXiv preprint arXiv:1805.11921*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3693–3702, 2017.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Byl8BnRcYm`.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

A PRELIMINARIES

In this section, we start with a definition of the graph classification problem, and then introduces related graph neural network variants.

Graph classification problem. Given an attributed graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V}$ is a set of vertexes/nodes, $\mathcal{E}$ is a set of edges, $X \in \mathbb{R}^{n \times d}$ are node attributes [3]. We can also denote $\mathcal{G} = (G, X)$ that separates the graph structure G and its attributes X. The goal is to predict target class $y \in \mathcal{Y}$, where $\mathcal{Y}$ is a set of pre-defined categories. Many real world problems can be formulated as graph classification problems, such as social and biological graph classification Yanardag & Vishwanathan (2015); Kipf & Welling (2016).

Graph neural networks. Graph Neural Networks (GNNs) define functions on the attributed graph $\mathcal{G}$. Typically, the graph function, GNN(G, X), learns a multiple steps of transformation of the original attributes/signals for node level or graph level representation. In each of the t-th step, a new node presentation, $h_v^{(t)}$ is learned. Initially, $h^{(1)}$ is initialized with the node feature vector, and during each subsequent step, a neighbor aggregation function is applied to generate the new node representation. More specifically, common neighbor aggregation functions for the v-th node take the following form.

$$h_v^{(t)} = f\left(h_v^{(t-1)}, \left\{h_u^{(t-1)} | u \in \mathcal{N}(v)\right\}\right),$$

where $\mathcal{N}(v)$ is a set of neighboring nodes of node v. To instantiate this general neighbor aggregation function, (Kipf & Welling, 2016) proposes the Graph Convolutional Network (GCN) aggregation scheme as follows.

$$h_v^{(t+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \tilde{a}_{uv} (W^{(t)})^T h_u^{(t)}\right),$$

where $W^{(t)} \in \mathbb{R}^{d \times d'}$ is the learnable transformation weight, $\tilde{a}_{uv}$ is the normalized edge weight between node u and v, and it is an entry in the normalized adjacency matrix (Kipf & Welling, 2016). This transformation can be compactly expressed as $H^{(t+1)} = \sigma(\tilde{A} H^{(t)} W^{(t)})$, where $H^{(t)} \in \mathbb{R}^{n \times d}$ are the hidden states of all nodes at t-th step.

More sophisticated neighbor aggregation schemes are also proposed, such as GraphSAGE (Hamilton et al., 2017) which allows pooling and recurrent aggregation over neighboring nodes. And in most

[3] We focus on node attributes in this work, but X can be extended into $(X_{\mathcal{V}}, X_{\mathcal{E}})$ where $X_{\mathcal{V}}$ are node attributes and $X_{\mathcal{E}}$ are edge attributes.

recent Graph Isomorphism Network (GIN) (Xu et al., 2019), a more power aggregation function is proposed as follows.

$$h_v^{(t)} = \text{MLP}^{(t)}\left(\left(1+\epsilon^{(t)}\right)h_v^{(t-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(t-1)}\right),$$

where MLP abbreviates for multi-layer perceptrons. Furthermore, the message function could be extended to incorporate edge features (Gilmer et al., 2017).

Finally, in order to generate graph level representation $h_\mathcal{G}$, a readout function is used, which generally takes the following form:

$$h_\mathcal{G} = g\left(\left\{h_v^{(T)} | v \in \mathcal{G}\right\}\right).$$

This can be instantiated by a global sum pooling, i.e. $h_\mathcal{G} = \sum_{v=1}^{n} h_v^{(T)}$, which is commonly followed by fully connected layers to generate the categorical or numerical output. More sophisticated graph readout functions are also possible, such as attention based ones Li et al. (2015).

B Analysis on the connection between GFNs and GNNs

We now turn to a more formal analysis on the connection between GFNs and GNNs. We generalize the multi-step neighbor propagation and graph readout function discussed above, and consider a graph neural network in two parts/stages: (1) the graph filtering stage that produces a finite set of representations for elements in the graph, and (2) the set function stage that combines signals in the set for graph level prediction.

Definition 1. *(Graph filtering) The graph filtering, $Y = \mathcal{F}_G(X)$, performs a transformation of input signals based on the graph G, that takes a set of signals $X \in \mathbb{R}^{n\times d}$, and outputs another set of filtered signals $Y \in \mathbb{R}^{m\times d'}$.*

Graph filtering in most existing GNNs consists of multi-step neighbor aggregation operations. For example, in GCN Kipf & Welling (2016), the multiple steps aggregation can be expressed as $H^{(T)} = \sigma(A(...\sigma(AXW^{(1)})...)W^{(T)})$. It is also worth noting that given no constraint on m and n in our definition, the graph filtering is allowed to perform a larger set of operations, including graph poolingDefferrard et al. (2016).

Definition 2. *(Set function) The set function, $y = \mathcal{T}(Y)$, takes a set of vector $Y \in \mathbb{R}^{m\times d'}$ where their orders do not matter, and outputs a task specific prediction $y \in \mathbb{R}^{d''}$.*

The graph readout function can be considered as a set function, which enables the graph level prediction that is permutation invariant w.r.t. nodes in the graph[4].

Lemma 1. *A GNN that performs graph-level transformation, i.e. mapping from $\mathcal{G}$ to y, can be decomposed into a graph filtering followed by a set function, i.e. GNN$(X, G) = \mathcal{T} \odot \mathcal{F}_G(X)$.*

A straight-forward proof is to set $\mathcal{F}_G(X) = \text{GNN}(X, G)$ (in which case $Y \in \mathbb{R}^{1\times d''}$), and $\mathcal{T}$ be an identity function. However, despite its correctness, it may not provide any helpful insight. A more practical approach is to scrutinize existing GNN variants under this point of view. Methods with flat aggregation such as GCN, GraphSAGE, GIN, they all have multi-step neighbor aggregation operations which belong to graph filtering, and also a readout function that is a set function. For methods with hierarchical pooling (such as max pooling), where only a subset of nodes are selected in each step, we could consider some nodes are masked out during the process.

By decomposing a GNN as graph filtering followed by a set function, one may wonder how to access the importance of each of these two stages. *Do we need a sophisticated graph filtering function for a particular task or dataset? And if we have a complex set function, is it possible to use a simple graph filtering function?* These questions are difficult to answer directly as the functionality of the two stages may overlap: if the graph filtering stage has fully processed node and graph features,

[4]Even Y contains hidden states of more than one type of objects (such as edge states), we could still augment the hidden states with extra type features such that Y can be treated as a set.

then a simple set function can be used for prediction. So despite the conceptual separation of graph filtering and the set function, they are still coupled computationally. To better decouple these two stages, similar to Wu et al. (2019), we propose to linearize the graph filtering function as follows.

Definition 3. *(Linear graph filtering) We say a graph filtering function $\mathcal{F}_G(X)$ is linear w.r.t. X if it is a linear transformation of $X \in \mathbb{R}^{n\times d}$, and the transformation of graph structure G does not contain any trainable parameters.*

Note that by constraining the learnable parameters in linear graph filtering, they cannot affect the graph structures for neighbor aggregation. An intuitive understanding of linear graph filtering is to take an existing graph filtering function and remove the non-linear operations, such as ReLU, and max pooling. By doing so, the graph filtering becomes linear w.r.t. X, thus could be collapsed into a flat structure.

Proposition 1. *A linear graph filtering function can be written as $\mathcal{F}_G^{linear}(X) = \boldsymbol{\pi}\Gamma(G)X^{(a)}\boldsymbol{\theta}$, where $\Gamma(G)$ is some transformation of G, $X^{(a)} \in \mathbb{R}^{(n+1)\times(d+1)}$ is an augmentation of X with its row and column extended and filled with one in the new entries, and $\boldsymbol{\pi}, \boldsymbol{\theta}$ are the only learnable parameters.*

Proof sketch. According to the definition of linear function w.r.t. X, a graph filtering without learnable parameters can be written as $\mathcal{F}_G(X) = \Gamma(G)X^{(a)}$. Given that the transformation of G does not contain any learnable parameters, the parameters for the linear transformation can only be added on both sides of $\Gamma(G)X^{(a)}$ in order to preserve linearity w.r.t. X, which completes our proof.

It is worth noting that existing GNNs satisfy a desirable permutation invariance condition (w.r.t. nodes), so to satisfy the same condition, we set $\boldsymbol{\pi}$ to an identity matrix. Since the parameters $\boldsymbol{\theta}$ are linearly associated, it could be absorbed into the next layer, leaving only graph propagated features, i.e. $\Gamma(G)X$, in the linear graph filtering stage. This enables us to disentangle graph filtering and the set function more thoroughly: the graph filtering part constructs graph augmented features, and the set function learns to compose them for the graph-level prediction.

Proposition 2. *A GNN that is permutation-invariant and has a linear graph filtering stage can be expressed as GFN with appropriated graph augmented features X^G.*

Proof sketch. We have shown that a GNN that is permutation-invariant, its linear graph filtering function can be expressed as $\Gamma(G)X^{(a)}\boldsymbol{\theta}$. The $\Gamma(G)X^{(a)}$ does not involve any learnable parameters, thus can be treated as graph augmented features X^G. $\boldsymbol{\theta}$ can be absorbed into the set function of GFN.

This is to say, GNNs, once linearized (e.g. removing non-linearity in graph filtering functions), collapse into graph augmented features with simple linear transformation parameterized by $\boldsymbol{\theta}$. And GFN allows a non-linear set function on top of these graph augmented features. For example, a linearized GCN, its K-th layer can be written as $H^{(K)} = \tilde{A}^K X(\Pi_{k=1}^K W^{(k)})$, and the parameters $\Pi_{k=1}^K W^{(k)}$ collapse into a flat $\boldsymbol{\theta}$.

Given the above analysis, we establish the connection between GFNs and GNNs. Generally, one can considered GFNs are equal or less powerful than GNNs as a linear graph filtering is assumed. There should exist scenarios where non-linear graph filtering is crucial for aggregating complex neighborhood information. However, the importance of non-linear graph filtering highly depends on the tasks or datasets at hand, and GFN can be a computationally efficient alternative when linear graph filtering is powerful enough. As a side result, GFN can also be used as a tool to poke at the necessity of a non-linear graph filtering function in a specific task or dataset. And as we will show in our experiments, the existing graph classification benchmarks do not seem to benefit much from a complex non-linear graph filtering.

C Details for datasets, baselines and model configuration

Datasets. The main datasets we consider are commonly used graph classification benchmarks (Yanardag & Vishwanathan, 2015; Xinyi & Chen, 2019; Xu et al., 2019). The graphs in the collection can be categorized into two categories: (1) biological graphs, including MUTAG, NCI1, PROTEINS, D&D, ENZYMES; and (2) social graphs, including COLLAB, IMDB-Binary (IMDB-B), IMDB-Multi (IMDB-M), Reddit-Multi-5K (RE-M5K), Reddit-Multi-12K (RE-M12K). It is worth noting that social graphs have no node attributes, while biological graphs come with node attributes. The

detailed statistics can be found in the appendix. In addition to the common graph benchmarks, we also consider image classification on MNIST where pixels are treated as nodes and nine nearest neighbors in the grid (including self-loop) are used to construct the graph. Detailed statistics of the biological and social graph datasets are listed in Table 6 and 7, respectively.

Dataset	MUTAG	NCI1	PROTEINS	D&D	ENZYMES
# graphs	188	4110	1113	1178	600
# classes	2	2	2	2	6
# features	7	37	3	82	3
Avg # nodes	17.93	29.87	39.06	284.32	32.63
Avg # edges	19.79	32.30	72.82	715.66	62.14

Table 6: Data statistics of Biological dataset

Dataset	COLLAB	IMDB-B	IMDB-M	RE-M5K	RE-12K
# graphs	5000	1000	1500	4999	11929
# classes	3	2	3	5	11
# features	1	1	1	1	1
Avg # nodes	74.49	19.77	13.00	508.52	391.41
Avg # edges	2457.78	96.53	65.94	594.87	456.89

Table 7: Data statistics of Social dataset

Baselines. We compare with two families of baselines. The first family of baselines are kernel-based, namely the Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al., 2011), Deep Graph Kernel (DGK) (Yanardag & Vishwanathan, 2015) and AWE (Ivanov & Burnaev, 2018) that incorporate kernel-based methods with with learning-based approach to learn embeddings. The second family of baselines are graph neural network-based models, which include recently proposed PATCHY-SAN (PSCN) (Niepert et al., 2016), Deep Graph CNN (DGCNN) (Zhang et al., 2018), CapsGNN (Xinyi & Chen, 2019) and GIN (Xu et al., 2019).

For the above baselines, we use their accuracies reported in the original papers, following the same evaluation setting as in (Xu et al., 2019). Different implementation details can make a difference, so to ensure a "A/B testing" between GFN and GNN, we also implement Graph Convolutional Networks (GCN) from (Kipf & Welling, 2016). More specifically, our GCN model contains a dense feature transformation layer, i.e. $H^{(2)} = \sigma(XW^{(1)})$, followed by three GCN layers, i.e. $H^{(t+1)} = \sigma(\tilde{A}H^{(t)}W^{(t)})$ [5]. To enable graph level prediction, we add a global sum pooling, followed by two fully-connected layers that produce categorical probability over pre-defined categories.

Model configurations. For the proposed GFN, we mirror our GCN's to mimic an "A/B testing". Therefore, we use the same architecture, parameterization and training setup, but replace the GCN layer with feature transformation layer, thus total of four feature transformation layers. Converting GCN layer to feature transformation layer is equivalent to setting $A = I$ in in GCN layers. We also construct a faster GFN, namely GFN-light, that contains only a single feature transformation layer, which can further reduce the training time while maintaining similar performance.

For both our GCN and GFN, we utilize ReLU activation and batch normalization Ioffe & Szegedy (2015), and fix the hidden dimensionality to 128. No regularization is applied. Furthermore we use batch size of 128, and a fixed learning rate of 0.001 with Adam optimizer Kingma & Ba (2014). To compare with existing work, we follow (Xinyi & Chen, 2019; Xu et al., 2019) and perform 10-fold cross validation. We report the average and standard deviation of validation accuracies across the 10 folds within the cross-validation. We run the model for 100 epochs, and select the the best epoch in the same way as Xu et al. (2019), i.e., a single epoch with the best cross-validation accuracy averaged over the 10 folds is selected.

[5] The number of GCN layers is selected according on its performance on both graph and biological datasets.

In terms of input node features to the proposed GFN, by default, we use both degree and multi-scale propagated features (up to $K = 3$), that is $\{\boldsymbol{d}, X, \tilde{A}^1X, \tilde{A}^2X, \tilde{A}^3X\}$. We discretize degree features into one-hot vector, as suggested in Fey & Lenssen (2019). As for our GCN, we also augment its node attributes with extra node degree attribute (to counter that the normalized adjacency matrix may not maintain the degree features).

Furthermore, for MNIST, we train and evaluate on given train/test split. Additionally, since MNIST benefits more from deeper GCN layers, we parameterize our GCN model using a residual network (He et al., 2016) with multiple GCN blocks, the number of blocks, and graph augmented features, are varied according to the size of total receptive filed. All experiments are run on Nvidia GTX 1080 Ti GPU.

D DETAILED PERFORMANCES WITH DIFFERENT FEATURES

Table 8: Accuracies (%) under various augmented features. $A^{1..3}X$ is abbreviated for A^1X, A^2X, A^3X, and default node feature X is always used (if available) but not displayed to reduce clutter.

Dataset	Model	None	$\boldsymbol{d}$	A^1X	$A^{1,2}X$	$A^{1..3}X$	$\boldsymbol{d}, A^1X$	$\boldsymbol{d}, A^{1,2}X$	$\boldsymbol{d}, A^{1..3}X$
MUTAG	GCN	83.48	87.09	83.35	83.43	85.56	87.18	87.62	88.73
	GFN	82.21	89.31	87.59	87.17	86.62	89.42	89.28	88.26
NCI1	GCN	80.15	83.24	82.62	83.11	82.60	83.38	83.63	83.50
	GFN	70.83	75.50	80.95	82.80	83.50	81.92	82.41	82.84
PROTEINS	GCN	74.49	76.28	74.48	75.47	76.54	77.09	76.91	77.45
	GFN	74.93	76.63	76.01	75.74	76.64	76.37	76.46	77.09
DD	GCN	79.29	78.78	78.70	77.67	78.18	78.35	78.79	79.12
	GFN	78.70	77.77	77.85	77.43	78.28	77.34	76.92	78.11
ENZYMES	GCN	75.17	67.17	72.00	71.50	70.50	69.50	69.33	69.67
	GFN	74.67	70.00	71.50	72.33	70.83	68.50	71.00	69.33
COLLAB	GCN	39.69	82.14	76.62	76.98	77.22	82.14	82.24	82.20
	GFN	31.57	80.36	76.40	77.08	77.04	81.28	81.62	81.26
IMDB-B	GCN	51.00	73.00	70.30	71.10	72.20	73.50	73.80	73.70
	GFN	50.00	73.30	72.30	71.30	71.70	74.40	73.20	73.90
IMDB-M	GCN	35.00	50.33	45.53	46.33	45.73	50.20	50.73	51.00
	GFN	33.33	51.20	46.80	46.67	46.47	51.93	51.93	51.73
RE-M5K	GCN	28.48	56.99	54.97	57.43	56.55	56.67	56.75	57.01
	GFN	20.00	54.23	51.11	55.85	56.35	56.45	57.01	56.71
RE-M12K	GCN	15.93	49.28	48.58	50.11	49.71	49.73	50.03	49.92
	GFN	17.33	44.86	43.61	48.25	48.87	48.31	49.37	49.39

Table 8 show the performances under different graph features for GNNs and GFNs. It is evident that both model benefit significantly from graph features, especially GFNs.

E DETAILED PERFORMANCES WITH DIFFERENT ARCHITECTURE DEPTHS

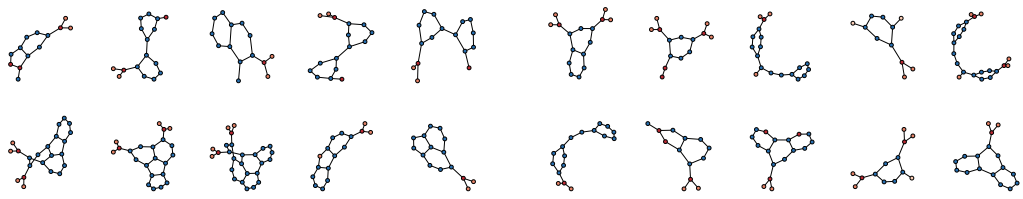
Table 9 shows performance per datasets under different number of layers.

F DETAILED VISUALIZATIONS

Figure 2, 3, 5, and 4 show the random and mis-classified samples for MUTAG, PROTEINS, IMDB-B, and IMDB-M, respectively. In general, it is difficult to find the patterns of each class by visually examining the graphs. And the mis-classified patterns are not visually distinguishable, except for IMDB-B/IMDB-M datasets where there are some graphs seem ambiguous.

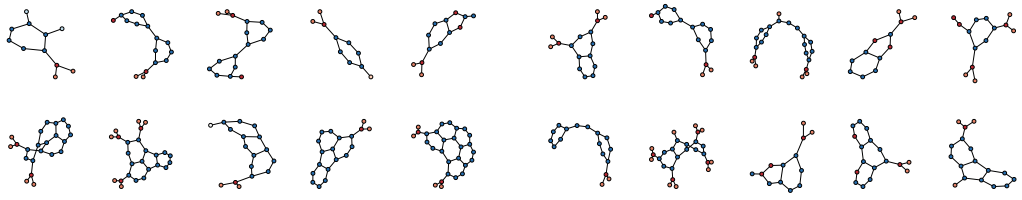
Table 9: Accuracies (%) under different number of Conv. layers. Flat denotes the collapsed GFN model into a logistic regression (no non-linear layer is used for the set function).

Dataset	Method	Flat	1	2	3	4	5
MUTAG	GCN	-	88.32	90.89	87.65	88.31	87.68
	GFN	82.85	90.34	89.39	88.18	87.59	87.18
NCI1	GCN	-	75.62	81.41	83.04	82.94	83.31
	GFN	68.61	81.77	83.09	82.85	82.80	83.09
PROTEINS	GCN	-	76.91	76.99	77.00	76.19	75.29
	GFN	75.65	77.71	77.09	77.17	76.28	75.92
DD	GCN	-	77.34	77.93	78.95	79.46	78.77
	GFN	76.75	78.44	78.78	79.04	78.45	76.32
ENZYMES	GCN	-	67.67	69.67	67.67	66.83	66.00
	GFN	43.83	69.67	70.50	71.67	69.83	68.17
COLLAB	GCN	-	80.36	81.86	81.40	81.90	81.78
	GFN	75.72	81.24	82.04	81.36	82.18	81.72
IMDB-B	GCN	-	72.60	72.30	73.30	73.80	73.40
	GFN	73.10	73.50	73.30	74.00	73.90	73.60
IMDB-M	GCN	-	51.53	51.07	50.87	51.53	50.60
	GFN	50.40	51.73	52.13	51.93	51.87	51.40
RE-M5K	GCN	-	54.05	56.49	56.83	56.73	56.89
	GFN	52.97	57.45	57.13	57.21	56.61	57.03
RE-M12K	GCN	-	44.91	48.87	49.45	49.52	49.61
	GFN	39.84	49.58	49.82	49.54	49.44	49.27



(a) Random samples.

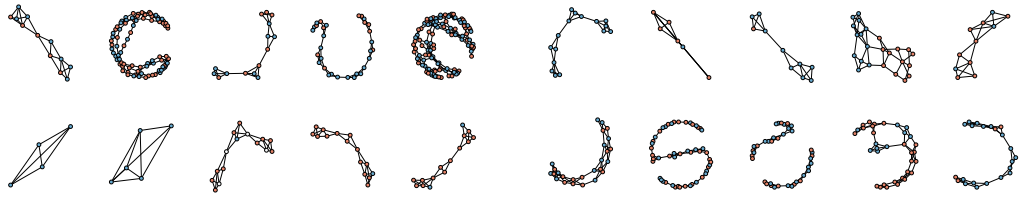
(b) Mis-classified samples by GFN.



(c) Random samples.

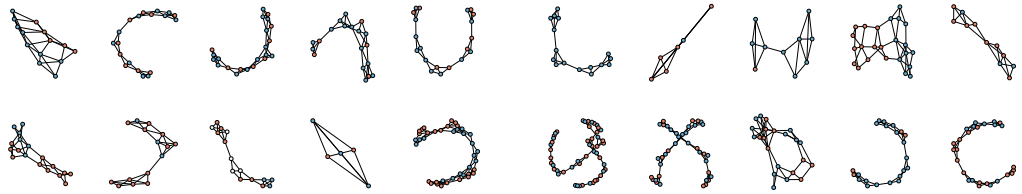
(d) Mis-classified samples by GCN.

Figure 2: Random and mis-classified samples from MUTAG. Each row represents a (true) class.



(a) Random samples

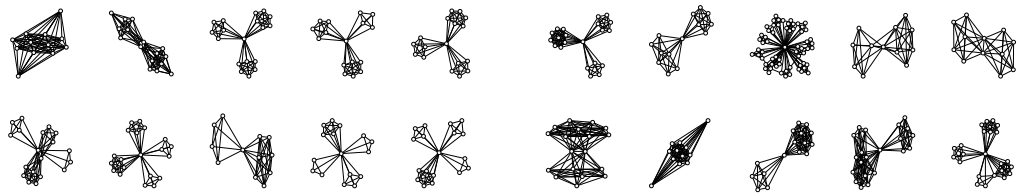
(b) Mis-classified samples by GFN.



(c) Random samples.

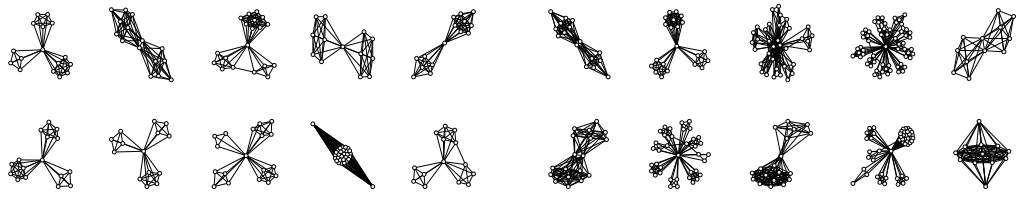
(d) Mis-classified samples by GCN.

Figure 3: Random and mis-classified samples from PROTEINS. Each row represents a (true) class.



(a) Random samples.

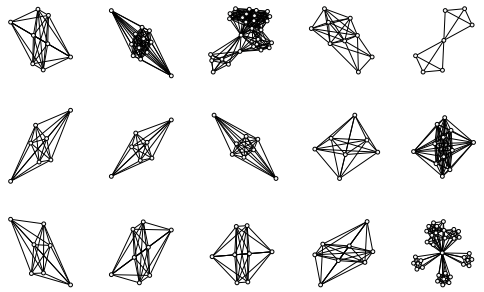
(b) Mis-classified samples by GFN.



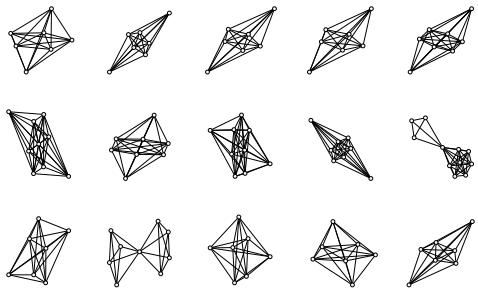
(c) Random samples.

(d) Mis-classified samples by GCN.

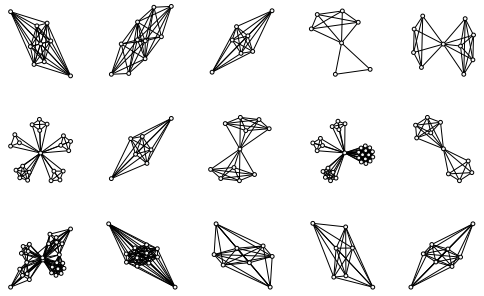
Figure 4: Random and mis-classified samples from IMDB-B. Each row represents a (true) class.



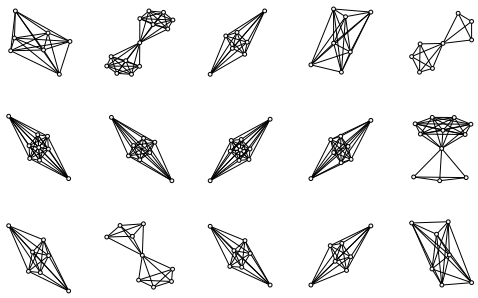
(a) Random samples.



(b) Mis-classified samples by GFN.



(c) Random samples.



(d) Mis-classified samples by GCN.

Figure 5: Random and mis-classified samples from IMDB-M. Each row represents a (true) class.