AdaGNN: Graph Neural Networks with Adaptive Frequency Response Filter

Yushun Dong¹, Kaize Ding², Brian Jalaian³, Shuiwang Ji⁴, Jundong Li¹

¹University of Virginia, ²Arizona State University, ³Army Research Laboratory, ⁴Texas A&M University

09/21/2021
Graph Neural Networks with Adaptive Frequency Response Filter

Overview

- Background Introduction
- Previous Works
- Existing Problems & Challenges
- Our Solutions
- Experiments & Conclusion
- Future Works
Background Introduction: Graph Neural Networks

**Goal of Graph Neural Networks (GNNs):** to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.

\[
\text{Enc} \cdot \text{func}(u) \quad \text{Enc} \cdot \text{func}(v)
\]

GNNs work here
Traditionally, frequency is defined as the number of occurrences of a repeating event per unit of time*, and a basic unit of frequency is Hertz.

\[ y(t) \]

* [https://en.wikipedia.org/wiki/Frequency](https://en.wikipedia.org/wiki/Frequency)
In graphs, we generalize the notion of frequency in the spatial domain to measure how fast a signal changes w.r.t. its graph structure.

- **Low frequency**: the signal changes slowly across edges.
- **High frequency**: the signal changes fast across edges.
Background Introduction: Frequency in Graphs

For time series signals, we have frequency basis. **Cosine function** is a commonly utilized basis for time series signal. For example, it is utilized as one of the basis of Fourier Transform.

\[ y(t) \]

This means the temporal signal has only one frequency component*. 

*Here we only show the positive half axis in the frequency domain.
In graphs, we utilize the eigenvectors of graph Laplacian as the ***basis*** of different frequencies*. 

\[ L = D - A = U\Lambda U^T \]

*Here the frequency notion is defined based on graph Laplacian. Similar notion can also be defined based on adjacency matrix, but larger eigenvalues corresponds to lower frequencies.*
Background Introduction: Frequency in Graphs

In graphs, we utilize the eigenvectors of graph Laplacian as the basis of different frequencies.

\[ L = D - A = U \Lambda U^T \]

We can intuitively understand the functionality of the graph Laplacian eigenvectors as cosine functions.

The corresponding eigenvalue is the frequency.
Background Introduction: Graph (low-pass) Filtering

By projecting \( X \) on different eigenvectors:

\[ X \]

\[ \text{Weight: 0.9} \quad \text{Weight: 0.6} \]
Background Introduction: Graph (low-pass) Filtering

Eigenvectors (of graph Laplacian) are regarded as signal basis.

Graph Neural Networks with Adaptive Frequency Response Filter
Background Introduction: Graph (low-pass) Filtering

Re-weighted components $X$ projected on each basis.

Graph Filter
Graph (low-pass) filtering is a process of reducing the weight on eigenvector basis corresponding to large eigenvalues (of the graph Laplacian matrix) in the graph signal (i.e., $X$ given $A$).
Background Introduction: Graph (low-pass) Filtering

Why we need to do this?

By filtering out "high-frequency" information (i.e., signals with high variances across the graph), the neighbor nodes are made to be similar. This helps us capture the dependencies between linked nodes.
Why we need to do this?

By filtering out "high-frequency" information (i.e., signals with high variances across the graph), the neighbor nodes are made to be similar. This helps us capture the dependencies between linked nodes.

Nevertheless, is low frequency all what we need?
Background Introduction: Graph (low-pass) Filtering

Why we need to do this?

By filtering out "high-frequency" information (i.e., signals with high variances across the graph), the neighbor nodes are made to be similar. This helps us capture the dependencies between linked nodes.

In assortative networks, similar nodes tend to link together; however, in disassortative networks, different nodes tend to link together.
Background Introduction: Graph (low-pass) Filtering

Why we need to do this?

By filtering out "high-frequency" information (i.e., signals with high variances across the graph), the neighbor nodes are made to be similar. This helps us capture the dependencies between linked nodes.

In assortative networks, similar nodes tend to link together; however, in disassortative networks, different nodes tend to link together.

This indicates that only preserving low-frequency components cannot fully capture all useful information [Bo et al. 2021].
Overview

Background Introduction

Previous Works

Existing Problems & Challenges

Our Solutions

Experiments & Conclusion

Future Works
**Previous Works: Spatial GNNs**

**Spatial GNNs:** focus on information aggregation between nodes in the spatial domain;

**Advantages:** explainable and flexible;

**Disadvantages:** limited supporting theoretical basis; more of an empirical method;

**Representative works:** Graph Attention Network [Petar et al. 2017], GraphSAGE [Hamilton et al. 2017], etc.
**Previous Works: Spectral GNNs**

**Spectral GNNs**: treat graph data as a whole and do signal filtering in the spectral domain;

**Advantages**: solid theoretical basis; easy to foresee the performance corresponding to certain type of graphs;

**Disadvantages**: hard to do inductive learning (not impossible though); low localized explainability;

**Representative works**: Fast Localized Graph Spectral Filtering [Defferrard et al. 2016], Graph Convolutional Network [Kipf et al. 2016], etc.
Overview

- Background Introduction
- Previous Works
- Existing Problems & Challenges
- Our Solutions
- Experiments & Conclusion
- Future Works
We focus on the problems of spectral methods: existing models such as GCN can only achieve non-learnable graph filter, which means that it cannot adaptively capture useful information that is not contained in the low-frequency component;
Existing Problems: Non-learnable Graph Filter

We focus on the problems of spectral methods: existing models such as GCN can only achieve non-learnable graph filter, which means that it cannot adaptively capture useful information that is not contained in the low-frequency component;

Layer expression of GCN:

$$\mathbf{Z} = \sigma(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\Theta)$$

$$= \sigma[(I - \tilde{\mathbf{L}})\mathbf{X}\Theta]$$

$$= \sigma[\mathbf{U}(I - \Lambda)\mathbf{U}^\top\mathbf{X}\Theta]$$

Response amplitude

$$f(\lambda) = 1 - \lambda$$
Existing Problems: Non-learnable Graph Filter

We focus on the problems of spectral methods: existing models such as GCN can only achieve **non-learnable graph filter**, which means that it cannot adaptively capture useful information that is not contained in the low-frequency component;

Layer expression of GCN:
\[
Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta) \\
= \sigma[(I - \tilde{L})X\Theta] \\
= \sigma[U(I - \Lambda)U^TX\Theta]
\]

Red component can be well-captured:

\[
f(\lambda) = 1 - \lambda
\]

Frequency-response function
Existing Problems: Non-learnable Graph Filter

We focus on the problems of spectral methods: existing models such as GCN can only achieve non-learnable graph filter, which means that it cannot adaptively capture useful information that is not contained in the low-frequency component;

Layer expression of GCN:
\[
Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta) = \sigma[(I - \tilde{L})X\Theta] = \sigma[U(I - \Lambda)U^TX\Theta]
\]

Red component can be well-captured:
Response amplitude

\[f(\lambda) = 1 - \lambda\]

Red component cannot be well-captured:
Response amplitude

Frequency-response function

Frequency

Frequency

Frequency

Frequency
Existing Problems: Non-learnable Graph Filter

We focus on the problems of spectral methods: existing models such as GCN can only achieve **non-learnable graph filter**, which means that it cannot adaptively capture useful information that is not contained in the low-frequency component;

Layer expression of GCN:

\[
Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta) = \sigma[(I - \tilde{L})X\Theta] = \sigma[U(I - \Lambda)U^TX\Theta]
\]

Problem to be tackled: the frequency response function should be **learnable** and **adaptively adjust** itself to capture useful information.
Existing Problems: Over-smoothing

Such fixed filter would greatly reduce most high-frequency components, i.e., making all nodes to be similar to each other.

\[ f(\lambda) = (1 - \lambda)^n \]

Node embeddings learned from GCN on Cora dataset [Liu et al. 2020]:

- Original data
- MLP
- #layer/hop 1
- #layer/hop 2
- #layer/hop 3
- #layer/hop 4
- #layer/hop 5
Overview

Background Introduction

Previous Works

Existing Problems & Challenges

Our Solutions

Experiments & Conclusion

Future Works
Layer-wise signal filtering operation comparison:

GCN (without learnable matrix)

\[
E = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X = X - \tilde{L}X
\]

Each channel corresponds to a fixed weight factor for filtering.
Our Solutions: AdaGNN Layer-wise Illustration

Layer-wise signal filtering operation comparison:

**GCN (without learnable matrix)**

\[
E = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X = X - \tilde{L}X
\]

**AdaGNN**

\[
E = X - \tilde{L}X\phi
\]

Each channel corresponds to a fixed weight factor for filtering.

Each channel corresponds to a learnable weight factor \( \phi \) for filtering at each specific layer.

Graph Neural Networks with Adaptive Frequency Response Filter
Model-wise signal filtering operation comparison:

**GCN**

- \( \mathbf{E}^{(k+1)} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \mathbf{D}^{-\frac{1}{2}} \mathbf{E}^{(k)} \)
- \( \mathbf{E}^{(k+1)} = \sigma(\mathbf{E}^{(k+1)} \mathbf{\Theta}) \)
- With Softmax as \( \sigma \)

**AdaGNN**

- \( \mathbf{E}^{(1)} = \sigma(\mathbf{E}^{(1)} \mathbf{\Theta}) \)
- \( \mathbf{E}^{(k+1)} = \mathbf{E}^{(k)} - \tilde{\mathbf{L}} \mathbf{E}^{(k)} \mathbf{\Phi} \)
- With Softmax as \( \sigma \)
Our Solutions: Learnable Filter in AdaGNN

**Question 1:** how could this help us to achieve a learnable filter?

<table>
<thead>
<tr>
<th>Layer</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Layer</td>
<td>$f(\lambda) = 1 - \lambda$</td>
</tr>
<tr>
<td>2 Layer</td>
<td>$f(\lambda) = (1 - \lambda)^2$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n Layer</td>
<td>$f(\lambda) = (1 - \lambda)^n$</td>
</tr>
</tbody>
</table>

SGC (without learnable matrix): $f(\lambda) = 1 - \lambda$

AdaGNN* (one feature dimension): $f(\lambda) = 1 - \phi_1 \lambda$

*For simplification purpose, we omit the weight matrix in the first layer.
Our Solutions: Learnable Filter in AdaGNN

**Question 1:** how could this help us to achieve a learnable filter?

<table>
<thead>
<tr>
<th>Layer</th>
<th>Weight Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Layer</td>
<td>( f(\lambda) = 1 - \lambda )</td>
</tr>
<tr>
<td>2 Layer</td>
<td>( f(\lambda) = (1 - \lambda)^2 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n Layer</td>
<td>( f(\lambda) = (1 - \lambda)^n )</td>
</tr>
</tbody>
</table>

SGC (without learnable matrix): \( f(\lambda) = 1 - \lambda \) \( f(\lambda) = (1 - \lambda)^2 \) \( f(\lambda) = (1 - \lambda)^n \)

AdaGNN* (one feature dimension): \( f(\lambda) = 1 - \phi_1 \lambda \) \( f(\lambda) = \prod_{i=1}^{2} (1 - \phi_i \lambda) \) \( f(\lambda) = \prod_{i=1}^{n} (1 - \phi_i \lambda) \)

Assume we have four feature dimensions:

\[ f_k(\tilde{\lambda}_i, \varphi_j) = \prod_{k=1}^{K} (1 - \varphi_{j,k} \tilde{\lambda}_i) \]

*For simplification purpose, we omit the weight matrix in the first layer.*
Question 2: how could this help us to relieve over-smoothing?

Information is aggregated across different feature dimensions indiscriminately, leading to similar nodes only after 2 layers*.

*In this example, we assume the attribute values can only be binary.
Our Solutions: Toy Example for Over-smoothing Relief

Question 2: how could this help us to relieve over-smoothing?

In AdaGNN, information can be aggregated in a dimension-specific manner*. With learnable $\phi$s, the embedding of different nodes can be more distinguishable according to their roles after information aggregation.

*In this example, we assume the attribute values can only be binary.
Overview

Background Introduction

Previous Works

Existing Problems & Challenges

Our Solutions

Experiments & Conclusion

Future Works
Experiments: General Settings

**Downstream tasks:**
- Node classification;

**Datasets:**
- BlogCatalog [Tang et al., 2009], Flickr [Huang et al., 2017], ACM [Tang et al., 2008], Cora, Citeseer and Pubmed [Sen et al., 2008];

**Baselines:**
- Three state-of-the-art GNNs including GCN [Kipf et al. 2016], SGC [Wu et al. 2019] and GraphSAGE [Hamilton et al. 2017]; Two recent approaches tackling over-smoothness including Dropedge [Rong et al. 2019] and Pairnorm [Zhao et al. 2019].

<table>
<thead>
<tr>
<th># Nodes</th>
<th>BlogCatalog</th>
<th>Flickr</th>
<th>ACM</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td># Edges</td>
<td>5,196</td>
<td>7,575</td>
<td>16,484</td>
<td>2,708</td>
<td>3,327</td>
<td>19,717</td>
</tr>
<tr>
<td># Features</td>
<td>173,468</td>
<td>242,146</td>
<td>71,980</td>
<td>5,429</td>
<td>4,732</td>
<td>44,338</td>
</tr>
<tr>
<td># Average Degree</td>
<td>8,189</td>
<td>12,047</td>
<td>8,337</td>
<td>1,433</td>
<td>3,703</td>
<td>500</td>
</tr>
<tr>
<td># Classes</td>
<td>66.8</td>
<td>63.9</td>
<td>8.7</td>
<td>4.0</td>
<td>2.8</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Graph Neural Networks with Adaptive Frequency Response Filter
Our model achieves the **best** performance on prediction accuracy in shallow layer.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>2 Layer</th>
<th>4 Layer</th>
<th>8 Layer</th>
<th>16 Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>GCN</td>
<td>73.98 ± 0.6%</td>
<td>69.71 ± 0.4%</td>
<td>37.61 ± 2.2%</td>
<td>20.61 ± 1.9%</td>
</tr>
<tr>
<td></td>
<td>GraphSAGE</td>
<td>70.41 ± 0.5%</td>
<td>67.03 ± 0.5%</td>
<td>39.15 ± 1.6%</td>
<td>18.34 ± 3.9%</td>
</tr>
<tr>
<td></td>
<td>SGC</td>
<td>73.97 ± 0.6%</td>
<td>68.94 ± 0.8%</td>
<td>47.94 ± 0.9%</td>
<td>29.02 ± 1.7%</td>
</tr>
<tr>
<td></td>
<td>DropEdge-GCN</td>
<td>74.17 ± 0.7%</td>
<td>70.96 ± 1.3%</td>
<td>60.51 ± 2.4%</td>
<td>51.88 ± 0.8%</td>
</tr>
<tr>
<td></td>
<td>Pairnorm-GCN-SI</td>
<td>67.32 ± 0.7%</td>
<td>63.61 ± 0.9%</td>
<td>65.04 ± 0.6%</td>
<td>67.51 ± 0.4%</td>
</tr>
<tr>
<td></td>
<td>Pairnorm-GCN-SCS</td>
<td>71.67 ± 0.3%</td>
<td>67.01 ± 0.2%</td>
<td>69.30 ± 0.7%</td>
<td>69.75 ± 1.2%</td>
</tr>
<tr>
<td></td>
<td>AdaGNN-R</td>
<td><strong>86.80 ± 0.3%</strong></td>
<td><strong>87.04 ± 0.2%</strong></td>
<td><strong>86.68 ± 0.1%</strong></td>
<td><strong>86.44 ± 0.5%</strong></td>
</tr>
<tr>
<td></td>
<td>AdaGNN-S</td>
<td><strong>88.50 ± 0.2%</strong></td>
<td><strong>88.79 ± 0.2%</strong></td>
<td><strong>88.81 ± 0.1%</strong></td>
<td><strong>88.19 ± 0.2%</strong></td>
</tr>
</tbody>
</table>

AdaGNN-R: model with asymmetrically normalized $\tilde{L}$; AdaGNN-S: model with symmetrically normalized $\tilde{L}$;
Experiments: Example Results on BlogCatalog

In deeper layers, our model not only achieves the best performance, but also greatly relieved over-smoothness.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>2 Layer</th>
<th>4 Layer</th>
<th>8 Layer</th>
<th>16 Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>GCN</td>
<td>73.98 ± 0.6%</td>
<td>69.71 ± 0.4%</td>
<td>37.61 ± 2.2%</td>
<td>20.61 ± 1.9%</td>
</tr>
<tr>
<td></td>
<td>GraphSAGE</td>
<td>70.41 ± 0.5%</td>
<td>67.03 ± 0.5%</td>
<td>39.15 ± 1.6%</td>
<td>18.34 ± 3.9%</td>
</tr>
<tr>
<td></td>
<td>SGC</td>
<td>73.97 ± 0.6%</td>
<td>68.94 ± 0.8%</td>
<td>47.94 ± 0.9%</td>
<td>29.02 ± 1.7%</td>
</tr>
<tr>
<td></td>
<td>DropEdge-GCN</td>
<td>74.17 ± 0.7%</td>
<td>70.96 ± 1.3%</td>
<td>60.51 ± 2.4%</td>
<td>51.88 ± 0.8%</td>
</tr>
<tr>
<td></td>
<td>Pairnorm-GCN-SI</td>
<td>67.32 ± 0.7%</td>
<td>63.61 ± 0.9%</td>
<td>65.04 ± 0.6%</td>
<td>67.51 ± 0.4%</td>
</tr>
<tr>
<td></td>
<td>Pairnorm-GCN-SCS</td>
<td>71.67 ± 0.3%</td>
<td>67.01 ± 0.2%</td>
<td>69.30 ± 0.7%</td>
<td>69.75 ± 1.2%</td>
</tr>
<tr>
<td></td>
<td>AdaGNN-R</td>
<td>86.80 ± 0.3%</td>
<td>87.04 ± 0.2%</td>
<td>86.68 ± 0.1%</td>
<td>86.44 ± 0.5%</td>
</tr>
<tr>
<td></td>
<td>AdaGNN-S</td>
<td>88.50 ± 0.2%</td>
<td>88.79 ± 0.2%</td>
<td>88.81 ± 0.1%</td>
<td>88.19 ± 0.2%</td>
</tr>
</tbody>
</table>

AdaGNN-R: model with asymmetrically normalized $\tilde{L}$; AdaGNN-S: model with symmetrically normalized $\tilde{L}$;
Experiments: Ablation Study

Ablation study with AdaGNN-S as an example:

An visualization of the learned filters across different feature dimension of AdaGNN-S on Flickr dataset.

Model ablation study of AdaGNN-S on BlogCatalog.
Conclusion

• AdaGNN **adaptively** learns the smoothness of each feature dimension, and it achieves a **learnable filter** after multiple layers are stacked together.

• The learnable filter contributes to the **performance superiority** and **over-smoothing relief**.
Overview

- Background Introduction
- Previous Works
- Existing Problems & Challenges
- Our Solutions
- Experiments & Conclusion
- Future Works
Future works

• Fairness issue in spectral GNNs.

• Spectral GNNs with better localized explainability.

• GNNs with learnable and more flexible filter.
References


The End

Thanks for listening!