Deep Graph Learning: Foundations, Advances and Applications

Theme II: Advance Topics and Applications

Yu Rong, Tingyang Xu, Junzhou Huang, Tencent AI Lab
Weining Huang, Tsinghua University
Hong Cheng, The Chinese University of Hong Kong

Tutorial website: https://ai.tencent.com/ailab/ml/KDD-Deep-Graph-Learning.html
• Brief History of Graph Neural Networks
• Advanced Topics in GNN:
  • Expressivity of GNNs
  • Training Deep GNNs
  • Scalability of GNNs
  • Self/Un-Supervised Learning of GNNs
• Applications:
  • GNN in Social Networks
  • GNN in Medical Imaging
• Future Directions
The Brief History of Graph Neural Networks
What is the Graph Neural Network?

Neural network model that can deal with graph data.

Applications
- Node Classification
- Link Prediction
- Community Detection
- Graph Generation
- ……
Sperduti, Alessandro and Starita, Antonina. 1997

Supervised Neural Networks for the Classification of Structures

Alessandro Sperduti and Antonina Starita, Member, IEEE

Abstract—Until now neural networks have been used for classifying unstructured patterns and sequences. However, standard neural networks and statistical methods are usually believed to be inadequate when dealing with complex structures because of their feature-based approach. In fact, feature-based approaches usually fail to give satisfactory solutions because of the sensitivity of the approach to the a priori selection of the features, and the incapacity to represent any specific information on the relationships among the components of the structures.

Conceptual Graph

Linear Representation

[HUMAN PROCESS: statement]

→ (AGENT) → [ACTOR: physician]
A Rapidly Growing Area

Number of GNN Papers

ICLR 2020 submissions keyword statistics

- deep learning
- optimization
- neural network
- generative models
- unsupervised learning
- reinforcement learning
- convolutional neural network
- recurrent neural network
- machine learning
- multitask learning
- neural architecture search
- representation learning
- adversarial robustness
- robustness
- self-supervised learning
- nlp
- transformer
- graph neural network

Δ between 2020 and 2019 in %

https://github.com/shaohua0116/ICLR2020-OpenReviewData
The Model of Graph Neural Networks

GNN 1.0

GNN 2.0

GNN 3.0
The Model of Graph Neural Networks

GNN 1.0

• Understanding GNN as RNN

GNN 2.0

GNN 3.0
GNN 1.0: Understanding GNN as RNN

- The RNN on sequences can be generalized to trees and DAGs.
GNN 1.0: Understanding GNN as RNN

Before 2000
Sperduti, Alessandro, and Antonina Starita. (TNN 97) propose the generalized recursive neuron for the graph classification problem on Trees/DAGs.

This generalized recursive neuron can only generate the graph representations.

From 2000 to 2010
Gori et.al (IJCNN 05) and Scarselli et.al (TNN 08) add the output gate for each node to generate the node representation in graphs. This model is called GraphRNN.

After 2010
Li, Yujia, et al. (ICLR 16) add gated recurrent units and modern optimization techniques to improve the performance of Scarselli et.al (TNN 09).
Tai, Kai Sheng et.al. (ACL 2015) extend LSTM to a tree-structured network topologies.
The Brief History of Graph Neural Networks

GNN 1.0
• Understanding GNN as RNN

GNN 2.0
• Understanding GNN as Convolution

GNN 3.0

GNN 2.0: Understanding GNN as Convolution

- How to perform the convolution on graphs?
  - Irregular structures.
  - Weighted edges.
  - No orientation or ordering (in general).
Deep Graph Learning: Foundations, Advances and Applications

GNN 2.0: Understanding GNN as Convolution

ChebyNet (NIPS 2016) [2]
- Build the connection between graph signal processing and graph convolution.
- Use Chebyshev polynomial to fast approximate the graph filtering in the spectral domain.

Graph Convolutional Network (ICLR 2017)
- Approximate 1-order Chebyshev polynomial the in spatial domain.
- Layer-wise convolution to extend receptive field.
- The practical convolutional model for graphs.

Deep Locally Connected Networks (ICLR 2014) [1]
- Discuss two constructions on both spatial and spectral domain.
- Analog the convolution operation based on the Laplacian spectrum.
- Additional eigen decomposition is needed.

PATCHY-SAN (ICML 2016)
- Neighborhood sampling to construct receptive field.

---

The Brief History of Graph Neural Networks

GNN 1.0
• Understanding GNN as RNN

GNN 2.0
• Understanding GNN as Convolution

GNN 3.0
• Variants of Convolutions
• GNN with Attention
• GNN with Graph Pooling
• High-order GNN
**GNN 3.0: Variants of Convolutions**

\[ g_\theta \ast x = U g_\theta U^T x \]

\[ H^{(l+1)} = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} W^{(l)}) \]

|----------------------|----------------------------------|--------------------|

[1] Xu, Bingbing, et al. 2018
**GNN 3.0: Variants of Convolutions**

\[ g_\theta \ast x = U g_\theta U^T x \]

\[ H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \]

- Employ Lanczos algorithm to obtain the low-rank approximation of the graph Laplacian \( \tilde{I} - \frac{1}{2}\tilde{A}\tilde{D}^{-\frac{1}{2}} \).
- Easy to construct multi-scale Graph Convolution.

---

**Lanczos Network [3]**

**Graph Wavelet Neural Network [1]**

**Hyperbolic GCN [2]**

---

GNN 3.0: Variants of Convolutions

\[ g_\theta \ast x = U g_\theta U^T x \]

\[ H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \]

- Employ Lanczos algorithm to obtain the low-rank approximation of the graph Laplacian \( I - \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \).
- Easy to construct multi-scale Graph Convolution.

Graph Wavelet Neural Network [1]

- Use wavelet transform to replace Fourier transform in the original GCN.
- More localized convolution and flexible neighborhood.

Lanczos Network [3]

Hyperbolic GCN [2]
**GNN 3.0: Variants of Convolutions**

\[
g_\theta \ast x = U g_\theta U^T x
\]

\[
H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})
\]

- **Lanczos Network [3]**
  - Employ Lanczos algorithm to obtain the low-rank approximation of the graph Laplacian \( I - \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} \).
  - Easy to construct multi-scale Graph Convolution.

- **Graph Wavelet Neural Network [1]**
  - Use wavelet transform to replace Fourier transform in the original GCN.
  - More localized convolution and flexible neighborhood.

- **Hyperbolic GCN [2]**
  - Smaller distortion.
  - Suitable for scale-free and hierarchical structure.
  - Hyperbolic feature transform.
    \[
h_i^{(l+1),H} = (W^{(l+1)} \otimes K_l) h_i^{(l),H} \otimes K_l b^{(l+1)}
    \]
  - Attention-based hyperbolic aggregation.
    \[
y_i^{(l+1),H} = AGG^K_l(h^{(l),H})_i
    \]

---

[1] Xu, Bingbing, et al. 2018
GNN 3.0: GNN with Attention

Replace the fixed aggregation weight $a_{ij}$ to the learnable self-attention.

$$h_i^{(l+1)} = \sigma(\sum_{j \in N(v_i)} S_{i,j} W^{(l)} h_j^{(l)})$$

$$s = \tilde{D}^{-rac{1}{2}} \tilde{A} \tilde{D}^{-rac{1}{2}}$$

Graph Attention Network [1]

Gated Attention Networks [2]

Spectral Graph Attention Network [3]

GNN 3.0: GNN with Attention

The original form:

\[ h_i^{(l+1)} = \sigma( \sum_{j \in N(v_i)} S_{i,j} W^{(l)} h_j^{(l)}) \]

Replace the fixed aggregation weight \( a_{ij} \) to the learnable self-attention.

\[ h_i^{(l+1)} = \sigma( \sum_{j \in N(v_i)} a_{ij} W^{(l)} h_j^{(l)}) \]

\[ a_{ij} = \exp\left( \frac{\sigma(\alpha^T[W h_i || W h_j])}{\sum_{k \in N(v_i)} \alpha^T[W h_i || W h_k]} \right) \]

Gated Attention Networks [2]

Add a learnable gate \( g_i^k \) to model the importance for each head.

\[ h_i^{(l+1)} = \sigma( \sum_{k=1}^{K} g_i^k \sum_{j \in N(v_i)} a_{ij} W^{(l)} h_j^{(l)}) \]

K is the number of heads.

Spectral Graph Attention Network [3]

Fixed during training

\[ S = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \]
GNN 3.0: GNN with Attention

The original form:

\[ h_i^{(l+1)} = \sigma(\sum_{j \in N(v_i)} S_{i,j} W^{(l)} h_j^{(l)}) \]

\[ S = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \]

Graph Attention Network [1]

Replace the fixed aggregation weight \( a_{ij} \) to the learnable self-attention.

\[ h_i^{(l+1)} = \sigma(\sum_{j \in N(v_i)} a_{ij} W^{(l)} h_j^{(l)}) \]

\[ a_{ij} = \exp(\frac{1}{\sum_{k \in N(v_i)} \alpha^T [W h_i \parallel W h_k]}) \]

Gated Attention Networks [2]

Add a learnable gate \( g_i^k \) to model the importance for each head.

\[ h_i^{(l+1)} = \sigma(\sum_{k=1}^K g_i^k \sum_{j \in N(v_i)} a_{ij} W^{(l)} h_j^{(l)}) \]

\( K \) is the number of heads.

Spectral Graph Attention Network [3]

Apply the attention on the high / low-frequency components in spectral domain.

\[ H^{(l+1)} = \sigma(\text{AGG}(B_L a_L B_L H^{(l)}, B_H a_H B_H H^{(l)}) W^{(l)}) \]

\( B = [B_L, B_H] \) is the spectral graph wavelet bases.

GNN 3.0: GNN with Graph Pooling

Graph Pooling/Coarsening: Convert the node representation to graph representation.

- The most straightforward way: Max/Mean Pooling
- SAGE: Attentive Pooling

Self-Attentive Graph Embedding (SAGE)

\[
\text{Attn} = \text{softmax}(W_s\tanh(W_h^Tz))
\]

Introduce the self-attention to model the node importance during the pooling.
Hierarchical Pooling

Graph Coarsening by Graph Cut [1]

Graph Pooling with pre-defined subgraph by graph cut algorithm.

Differentiable Graph Pooling (DIFFPOOL)[2]

Learn the cluster assignment matrix to aggregate the node representations in a hierarchical way.

EigenPooling [3]

Incorporate the node features and local structures to obtain a better assignment matrix.

[GNN 3.0: GNN with Graph Pooling]

GNN 3.0: High-order GNN

High-order GNN: extending the receptive field to encode high-order proximities in graphs.

DCNN

A $n \times H \times n$ tensor stacking the power series of the transition matrix $P$.

MixHop

$H^{(l+1)} = \prod_{j \in P} \sigma(A_j H^{(l)} W_j^{(l)})$

The normalized $j$-order adjacency matrix

APPNP

Incorporate the personalize page rank to capture the better locality of the target node.

GNN Implementation: Message Passing Framework

• Message Passing Framework:
  • **Step 1**: Gather and transform the messages from neighbors:
    \[ m^{(l+1)}_i = \text{AGG}\{ M^{(l+1)}(h^{(l)}_i, h^{(l)}_j, e_{i,j}) \mid j \in N(v_i) \} \]
  • **Step 2**: Update the state of the target node.
    \[ h^{(l+1)}_i = U^{(l+1)}(h^{(l)}_i, m^{(l+1)}_i) \]

• Most of current spatial GNNs can be formulated as a message passing process.

Yu Rong, Wenbing Huang, Tingyang Xu, Hong Cheng, Junzhou Huang 2020

Deep Graph Learning: Foundations, Advances and Applications

Summary

Advanced topics

Expressivity of GNNs

Training Deep GNNs

Scalability of GNNs

Self/Un-Supervised Learning of GNNs

...

Spatial Domain

Graph Attention Networks

Graph Pooling

Graph Convolutional Networks (GCN)

Relational GCN (r-GCN)

Graph Wavelet Neural Network

Hyperbolic GCN

Spectral Domain

Spectral Graph Convolution

Graph Signal Processing

Recurrent Neural Network

Recursive Neuron on Trees/DAGs

GraphRNN

Gated Graph Sequence Neural Networks

Message Passing Neural Network
Expressivity of GNNs
# What can GNNs compute?

<table>
<thead>
<tr>
<th>I. Graph Isomorphism</th>
<th>2. Function Approximation</th>
<th>3. Graph Property Detection/Optimization/Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph classification</td>
<td>Predicting the chemical property of molecule</td>
<td>Finding the shortest path between two given nodes</td>
</tr>
<tr>
<td>HO-GNN [1]; GIN [2]</td>
<td>IGNs [3,4,5]</td>
<td>GraphMoments [6]; CPNGNN [7]; [8,9]</td>
</tr>
</tbody>
</table>

1. The Graph Isomorphism (GI) view

Given any two graphs, can GNN determine if they are isomorphic or not?
1. The Graph Isomorphism (GI) view

GI is NP problem, mostly solved by Weisfeiler-Lehman (WL) test (1968)

For each iteration:
Step-1: neighborhood aggregation
Step-2: label compression by hashing
Step-3: relabeling

Figures from Shervashidze et al. 2011
1. The Graph Isomorphism (GI) view

Xu et al. (2019) and Morris et al. (2019) proved that,

\[ \text{GNN} \leq \text{WL} \]

Xu et al. (2019) further proved, if the aggregation/readout functions are injective,

\[ \text{GNN} = \text{WL} \]
2. The Function Approximation (FA) view

For any function on graphs, if there is a GNN approximating it up to an arbitrary accuracy?

\[ f(A) \]

\[ \exists \text{GNN, s.t. } |\text{GNN} - f| < \epsilon? \]

This kind of universality theorem has been proved for typical DNNs (Cybenko, 1989; Hornik, 1991)
2. The Function Approximation (FA) view

Function on graphs is symmetric w.r.t. node permutation

$A$

Permutation:

$PAP^T$

1 2 3

1 2 3
2. The Function Approximation (FA) view

**G-invariant** function: Permutation does not change the output, e.g. graph classification

\[
f(PAP^T) = f(A)
\]

\[
f(\begin{array}{c}
\text{ Perm. does not change output }
\end{array}) = f(\begin{array}{c}
\text{ Perm. does not change output }
\end{array})
\]

**G-equivariant** function: Permutation is preserved in the output, e.g. node classification

\[
f(PAP^T) = Pf(A)P^T
\]

\[
f(\begin{array}{c}
\text{ Perm. is preserved in output }
\end{array}) = f(\begin{array}{c}
\text{ Perm. is preserved in output }
\end{array})
\]
2. The Function Approximation (FA) view

What does G-invariant/equivariant function look like?

Linear function on graphs is defined as:

\[ A^{l+1} = f(A^l) = L \ast A^l \]

Enforcing the invariance and equivariance, we have (Maron et al. 2019a):

**G-Invariant Layer:**

\[ P^k \vec{vec}(L) = \vec{vec}(L) \]

**G-Equivariant Layer:**

\[ P^{2k} \vec{vec}(L) = \vec{vec}(L) \]
2. The Function Approximation (FA) view

G-Invariant Network (INN)
(Maron et al. 2019b)

\[ X \in \mathbb{R}^n \rightarrow \text{GEL} \rightarrow \sigma \rightarrow \text{GEL} \rightarrow \ldots \rightarrow \text{GIL} \rightarrow y \]

- **G-Equivariant Layer**
- **Non-linear (ReLu)**
- **G-Invariant Layer**
2. The Function Approximation (FA) view

[Universality Theorem, Maron et al. 2019b] There exists a G-Invariant network (if high-order hidden tensors are allowed) that approximates any G-invariant function to an arbitrary precision.

\[ \forall f, \forall \varepsilon > 0, \exists W, \text{s.t.} |\text{INN}_w - f| < \varepsilon \]
Graph Isomorphism or Function Approximation?

Graph Isomorphism

Isomorphic? (Chen et al. 2019)

Function Approximation

$GEL \xrightarrow{\sigma} GEL \xrightarrow{\ldots} GIL$

$f(A)$
Are GNNs expressive enough to solve the following problems?

Finding the shortest path?  If a graph contains a circle?

Yes, if the **depth** and **width** are beyond certain bounds, with sufficiently discriminative node attributes (Loukas 2020)
Training Deep GNNs
Training Deeper GNNs

Why do we need deeper GNNs?
Can GNNs simply go deeper?
What impedes GNNs to go deeper?
How to alleviate over-smoothing?
How to overcome training dynamics?
Unprecedented success of deep DNNs in computer vision

Deeper DNNs enable larger receptive fields
The Power of Deeper GNNs

- Do GNNs need deeper structures to enable larger receptive fields, too?
- What limits the expressive power of GNNs?
  - The depth $d$
  - The width $w$

GNNs significantly lose their power when *capacity, $d w$, is restricted*

---

The Power of Deeper GNNs

- Do GNNs need deeper structures to enable larger receptive fields, too?
- What limits the expressive power of GNNs?
  - The depth $d$
  - The width $w$

GNNs significantly lose their power when capacity, $d_\text{w}$, is restricted.

## The Power of Deeper GNNs

The boundary of capacity for different problems

<table>
<thead>
<tr>
<th>problem</th>
<th>bound</th>
<th>problem</th>
<th>bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycle detection (odd)</td>
<td>$dw = \Omega(n/\log n)$</td>
<td>shortest path</td>
<td>$d\sqrt{w} = \Omega(\sqrt{n} / \log n)$</td>
</tr>
<tr>
<td>cycle detection (even)</td>
<td>$dw = \Omega(\sqrt{n} / \log n)$</td>
<td>max. indep. set</td>
<td>$dw = \Omega(n^2 / \log^2 n)$ for $w = O(1)$</td>
</tr>
<tr>
<td>subgraph verification*</td>
<td>$d\sqrt{w} = \Omega(\sqrt{n} / \log n)$</td>
<td>min. vertex cover</td>
<td>$dw = \Omega(n^2 / \log^2 n)$ for $w = O(1)$</td>
</tr>
<tr>
<td>min. spanning tree</td>
<td>$d\sqrt{w} = \Omega(\sqrt{n} / \log n)$</td>
<td>perfect coloring</td>
<td>$dw = \Omega(n^2 / \log^2 n)$ for $w = O(1)$</td>
</tr>
<tr>
<td>min. cut</td>
<td>$d\sqrt{w} = \Omega(\sqrt{n} / \log n)$</td>
<td>girth 2-approx.</td>
<td>$dw = \Omega(\sqrt{n} / \log n)$</td>
</tr>
<tr>
<td>diam. computation</td>
<td>$dw = \Omega(n / \log n)$</td>
<td>diam. $3/2$-approx.</td>
<td>$dw = \Omega(\sqrt{n} / \log n)$</td>
</tr>
</tbody>
</table>

Training Deeper GNNs

Why do we need deeper GNNs?

Can GNNs simply go deeper?

- **GCN**: Basic GCN
- **GraphSAGE**: GCN with improved aggregation
- **JKNet**: leverage idea from DenseNet
- **ResGCN**: leverage idea from ResNet
- **IncepGCN**: leverage idea from Inception-v3
- **APPNP**: leverage idea from PageRank

What impedes GNNs to go deeper?

How to alleviate over-smoothing?

How to overcome training dynamics?
GNNs are Shallow

But can they really go deeper? Not all

What is the underlying reason of going deeper?

<table>
<thead>
<tr>
<th>Method</th>
<th>4 layers</th>
<th>16 layers</th>
<th>64 layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>76.7</td>
<td>65.2</td>
<td>44.6</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>77.3</td>
<td>72.9</td>
<td>16.9</td>
</tr>
<tr>
<td>ResGCN</td>
<td>78.9</td>
<td>78.2</td>
<td>21.2</td>
</tr>
<tr>
<td>JKNNet</td>
<td>79.1</td>
<td>78.8</td>
<td>76.7</td>
</tr>
<tr>
<td>IncepGCN</td>
<td>79.5</td>
<td>78.5</td>
<td>79</td>
</tr>
<tr>
<td>APPNP</td>
<td>79.3</td>
<td>81.0</td>
<td>80.4</td>
</tr>
</tbody>
</table>
Training Deeper GNNs

- Why do we need deeper GNNs?
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
  - Over-smoothing (Graph Specific)
  - Overfitting (Common)
  - Training dynamics (Common)
- How to alleviate over-smoothing?
- How to overcome training dynamics?
Training Deeper GNNs

Why do we need deeper GNNs?
Can GNNs simply go deeper?
What impedes GNNs to go deeper?
  - Over-smoothing (Graph Specific)
    - Overfitting (Common)
    - Training dynamics (Common)
How to alleviate over-smoothing?
How to overcome training dynamics?
Over-Smoothing

GNNs suffers from over-smoothing

GNNs suffers from over-smoothing

As the layers go deeper, the hidden variables converge to a subspace

Over-Smoothing of Linear GCN

Why GCN works?
- Laplacian smoothing → symmetric Laplacian smoothing
- The **weighted** average of itself and its **neighbors** → the new feature of a vertex

\[ \tilde{L} = \tilde{D} - \tilde{A} \]
\[ \tilde{D}^{-1}\tilde{L} \rightarrow \tilde{D}^{-\frac{1}{2}}\tilde{L}\tilde{D}^{-\frac{1}{2}} \]
\[ \gamma = 1 \]

Laplacian Smoothing
\[ Y = (I - \gamma \tilde{D}^{-1}\tilde{L})X \]

GCNs
\[ Y = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW \]
Over-Smoothing

When GCNs fail?

- \( H_L \) converges to a certain point with linear activation
- \( H_L \) converges to a certain subspace \( \mathcal{M} \) with ReLU activation
- \( H_L \) converges to a certain sub-cube \( O(\mathcal{M}, r) \) with ReLU and bias

Over-Smoothing of Linear GCN

When GCNs fail?

$H_L$ converges to a certain point with linear activation

$l$-step Random Walk

$$Y = \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right)^l P_0 \text{ where } p_{ij} = 1/d(i) \text{ if } (i,j) \in \mathcal{E}$$

Random Walks on Graph

- $V_{26} - V_{25} - V_{32} - V_3 - V_{10} ...$
- $V_5 - V_7 - V_{17} - V_6 - V_{11} ...$
- $V_{31} - V_{33} - V_{21} - V_{33} - V_{15}$

---

Li, Qimei, Zhichao Han, and Xiao-Ming Wu. "Deeper insights into graph convolutional networks for semi-supervised learning." AAAI 2018.

Over-Smoothing of Linear GCN

When GCNs fail?

$H_L$ converges to a certain point with linear activation

$l$-step Random Walk

$$Y = \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right)^l P_0 \text{ where } p_{ij} = 1/d(i) \text{ if } (i,j) \in \mathcal{E}$$

$l$-layer GCNs

$$Y = \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right)^l XW$$

Learnable Probability
## Over-Smoothing of Linear GCN

### When GCNs fail?

- $H_L$ converges to a certain point with linear activation

### l-step Random Walk

\[ Y = \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right)^l P_0 \text{ where } p_{ij} = 1/d(i) \text{ if } (i,j) \in \mathcal{E} \]

### l-layer GCNs

\[ Y = \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right)^l XW \]

### Eigen decomposition

\[ Y = \sum_{i=1}^{n} \tilde{D}^{-\frac{1}{2}} \left( \lambda_i u_i u_i^T \right)^l XW \]
Over-Smoothing of Linear GCN

Rewrite eigen decomposition

\[
\tilde{D}^{-\frac{1}{2}}(\lambda_1 u_1 u_1^T)XW + \cdots \tilde{D}^{-\frac{1}{2}}(\lambda_m u_m u_m^T)XW + \tilde{D}^{-\frac{1}{2}}(\lambda_{m+1} u_{m+1} u_{m+1}^T)XW + \cdots \tilde{D}^{-\frac{1}{2}}(\lambda_n u_n u_n^T)XW
\]
Over-Smoothing of Linear GCN

Rewrite eigen decomposition

\[ \tilde{D}^{-\frac{1}{2}}(\lambda_1 u_1 u_1^T)^l X W + \cdots + \tilde{D}^{-\frac{1}{2}}(\lambda_m u_m u_m^T)^l X W + \tilde{D}^{-\frac{1}{2}}(\lambda_{m+1} u_{m+1} u_{m+1}^T)^l X W + \cdots + \tilde{D}^{-\frac{1}{2}}(\lambda_n u_n u_n^T)^l X W \]

Suppose graph \( g \) has \( m \) connected components. It indicates

Eigenvalues

\[ 1 = \lambda_1 = \cdots = \lambda_m > \lambda_{m+1} > \cdots > \lambda_n > -1 \]
Rewrite eigen decomposition

\[ \mathbf{D}^{-\frac{1}{2}}(\lambda_1 \mathbf{u}_1 \mathbf{u}_1^\top)^l \mathbf{XW} + \cdots + \mathbf{D}^{-\frac{1}{2}}(\lambda_m \mathbf{u}_m \mathbf{u}_m^\top)^l \mathbf{XW} + \mathbf{D}^{-\frac{1}{2}}(\lambda_{m+1} \mathbf{u}_{m+1} \mathbf{u}_{m+1}^\top)^l \mathbf{XW} + \cdots + \mathbf{D}^{-\frac{1}{2}}(\lambda_n \mathbf{u}_n \mathbf{u}_n^\top)^l \mathbf{XW} \]

Suppose graph \( g \) has \( m \) connected components. It indicates

1 = \( \lambda_1 = \cdots = \lambda_m \ > \lambda_{m+1} \ > \cdots > \lambda_n \ > -1 \)

When \( l \to +\infty \), \( \lambda_{m+1}, \ldots, \lambda_n \to 0 \)

\[ \lim_{l \to \infty} \mathbf{D}^{-\frac{1}{2}}(\lambda_1 \mathbf{u}_1 \mathbf{u}_1^\top)^l \mathbf{XW} + \cdots + \mathbf{D}^{-\frac{1}{2}}(\lambda_m \mathbf{u}_m \mathbf{u}_m^\top)^l \mathbf{XW} + \mathbf{D}^{-\frac{1}{2}}(\lambda_{m+1} \mathbf{u}_{m+1} \mathbf{u}_{m+1}^\top)^l \mathbf{XW} + \cdots + \mathbf{D}^{-\frac{1}{2}}(\lambda_n \mathbf{u}_n \mathbf{u}_n^\top)^l \mathbf{XW} \]
Over-Smoothing of Non-Linear GCN

\( H_L \) converges to a certain subspace \( \mathcal{M} \) with ReLU activation

We define a subspace \( \mathcal{M} \) first

\( \mathcal{M} \) subspace

**Definition 1** (subspace). Let \( \mathcal{M} := \{ EC | C \in \mathbb{R}^{M \times C} \} \) be an \( M \)-dimensional subspace in \( \mathbb{R}^{N \times C} \), where \( E \in \mathbb{R}^{N \times M} \) is orthogonal, i.e. \( E^T E = I_M \), and \( M \leq N \).

\( d_{\mathcal{M}}(\cdot) \) refers to the distance to the subspace \( \mathcal{M} \)

\[ d_{\mathcal{M}}(\cdot) \]

(b) Non-Linear Case

---

Over-Smoothing of Non-Linear GCN

\( H_L \) converges to a certain subspace \( \mathcal{M} \) with ReLU activation

- We define a subspace \( \mathcal{M} \) first

\( \mathcal{M} \) subspace

**Definition 1** (subspace). Let \( \mathcal{M} := \{ EC \mid C \in \mathbb{R}^{M \times C} \} \) be an \( M \)-dimensional subspace in \( \mathbb{R}^{N \times C} \), where \( E \in \mathbb{R}^{N \times M} \) is orthogonal, i.e. \( E^T E = I_M \), and \( M \leq N \).

- \( d_\mathcal{M}(\cdot) \) refers to the distance to the subspace \( \mathcal{M} \)
- \( d_\mathcal{M}(\cdot) \) converges as the layers go deeper

**Convergence**

\[
d_\mathcal{M}(H_{l+1}) = d_\mathcal{M}\left(\sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H_lW_l)\right) \leq s_l\lambda_{m+1}d_\mathcal{M}(H_l)
\]

- \( \lambda_{m+1} < 1 \) is the largest non-one eigenvalue
- \( s_l \leq 1 \) is the maximum singular value of \( W_l \)

---

Over-Smoothing of Non-Linear GCN

**Convergence of $\tilde{A}$**

$$d_{\mathcal{M}} \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \right) \leq \lambda_{m+1} d_{\mathcal{M}} (X), \quad \lambda_{m+1} < 1$$

**Convergence**

$$d_{\mathcal{M}} (H_{l+1}) = d_{\mathcal{M}} \left( \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l \right) \right) \leq \lambda_{m+1} s_l d_{\mathcal{M}} (H_l)$$

Over-Smoothing of Non-Linear GCN

**Convergence of $\tilde{A}$**

$$d_{\mathcal{M}}\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\right) \leq \lambda_{m+1} d_{\mathcal{M}}(X), \quad \lambda_{m+1} < 1$$

**Convergence of $W$**

$$d_{\mathcal{M}}(XW_l) \leq s_l d_{\mathcal{M}}(X), \quad s_l \leq 1$$

**Convergence**

$$d_{\mathcal{M}}(H_{l+1}) = d_{\mathcal{M}}\left(\sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H_lW_l)\right) \leq \lambda_{m+1}s_l d_{\mathcal{M}}(H_l)$$

---

Oono, Kenta, and Taiji Suzuki. "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification." *ICLR 2020*. 
Over-Smoothing of Non-Linear GCN

Convergence of $\tilde{A}$

$$d_M(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X) \leq \lambda_{m+1} d_M(X), \quad \lambda_{m+1} < 1$$

Convergence of $W$

$$d_M(XW_l) \leq s_l d_M(X), \quad s_l \leq 1$$

Convergence of ReLU

$$d_M(\sigma(X)) \leq d_M(X)$$

Convergence

$$d_M(H_{l+1}) = d_M\left(\sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H_lW_l)\right) \leq \lambda_{m+1} s_l d_M(H_l)$$
Over-Smoothing of GCNs with bias

- $H_L$ converges to a certain sub-cube $O(\mathcal{M}, r)$ with ReLU and bias

\[
H_{l+1} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l + b_l \right)
\]
Over-Smoothing of GCNs with bias

- \(H_L\) converges to a certain sub-cube \(O(\mathcal{M}, r)\) with ReLU and bias

\[
H_{l+1} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l + b_l \right)
\]

Convergence of bias

\[
d_{\mathcal{M}}(H_{l+1}) \leq \lambda_{m+1} s_l d_{\mathcal{M}}(H_l) + d_{\mathcal{M}}(b_l)
\]
Over-Smoothing of GCNs with bias

- $H_L$ converges to a certain sub-cube $O(\mathcal{M}, r)$ with ReLU and bias

\[
H_{l+1} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l + b_l \right)
\]

Convergence of bias

\[
d_{\mathcal{M}}(H_{l+1}) \leq \lambda_{m+1} s_l d_{\mathcal{M}}(H_l) + d_{\mathcal{M}}(b_l)
\]

GCN with bias

\[
\lim_{l \to +\infty} d_{\mathcal{M}}(H_l) \leq \begin{cases} 
\frac{d_{\mathcal{M}}(b_{\text{max}})}{1 - \lambda_{m+1} s_{\text{max}}}, \text{with } \lambda_{m+1} s_l < 1 \\
\infty, \text{with } \lambda_{m+1} s_l > 1
\end{cases}
\]
Overall Over-Smoothing?

Linear GCN
\[
\lim_{l \to +\infty} (I - \gamma L_{sym})^l X = \tilde{D}^{-\frac{1}{2}} [\mathbf{1}^i (XW)^i]_{i=1}^m
\]

Non-linear GCN
\[
\lim_{l \to +\infty} d_M(H_l) \leq (\lambda_{m+1}s_{\text{max}})^l d_M(X) = \begin{cases} 
0, \text{ with } \lambda_{m+1}s_l < 1 \\
\infty, \text{ with } \lambda_{m+1}s_l > 1 
\end{cases}
\]

GCN with bias
\[
\lim_{l \to +\infty} d_M(H_l) \leq \begin{cases} 
\frac{d_M(b_{\text{max}})}{1 - \lambda_{m+1}s_{\text{max}}}, \text{ with } \lambda_{m+1}s_l < 1 \\
\infty, \text{ with } \lambda_{m+1}s_l > 1 
\end{cases}
\]
Universal Over-Smoothing

General Case

\[ d_M(H_{l+1}) - r \leq \nu (d_M(H_l) - r) \]

Basic GCN

Non-linear GCN

\[ \nu = s_{\text{max}} \lambda_{m+1} \]
\[ r = 0 \]

GCN with bias

\[ \nu = s_{\text{max}} \lambda_{m+1} \]
\[ r = \frac{d_M(b_{\text{max}})}{1 - \nu} \]

Training Deeper GNNs

- Why do we need deeper GNNs?
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
  - Over-smoothing (Graph Specific)
  - Overfitting (Common)
  - Training dynamics (Common)
- How to alleviate over-smoothing?
- How to overcome training dynamics?
GNNs suffer from Overfitting

Too many parameters are established but only few of data points are provided

Training Deeper GNNs

Why do we need deeper GNNs?
Can GNNs simply go deeper?
What impedes GNNs to go deeper?
  - Over-smoothing (Graph Specific)
  - Overfitting (Common)
  - Training dynamics (Common)
How to alleviate over-smoothing?
How to overcome training dynamics?
Training dynamics

The gradients vanish as the model go deeper because \( s_1 \ldots l \lambda_{m+1} < 1 \)

\[
\frac{dH_{l+1}}{dH_l} \cdot \frac{dH_l}{dH_{l-1}} \cdots \frac{dH_0}{dW_0} \leq (s_l\lambda_{m+1}) \cdot (s_{l-1}\lambda_{m+1}) \cdots \frac{dH_0}{dW_0}
\]
Training Deeper GNNs

Why do we need deeper GNNs?
Can GNNs simply go deeper?
What impedes GNNs to go deeper?

- **Over-smoothing (Graph Specific)**
- Overfitting (Common)
- Training dynamics (Common)

How to alleviate over-smoothing?
- Deal with adjacency matrix
- Deal with weights

How to overcome training dynamics?
Over-smoothing Layer

When does over-smoothing happen?

Let’s take basic GCN as example: \( v = \lambda_{m+1} s_{max}, r = 0 \)
Over-smoothing Layer

When does over-smoothing happen?

Let’s take basic GCN as example: \( v = \lambda_{m+1} s_{max}, r = 0 \)

\[ d_M(H_l) \leq (\lambda_{m+1} s_{max})^l d_M(X) < \epsilon, \forall l \geq L \]
Over-smoothing Layer

When does over-smoothing happen?

Let’s take basic GCN as example: \( v = \lambda_{m+1}s_{max}, r = 0 \)

\( \epsilon \)-smoothing

\[
d_M(H_l) \leq (\lambda_{m+1}s_{max})^l d_M(X) < \epsilon, \forall l \geq L
\]

\( \epsilon \)-smoothing layer

\[
l^*(M, \epsilon) := \left\{ \min_l d_M(H_l) < \epsilon \right\}
\]

Relaxed \( \epsilon \)-smoothing layer

\[
\hat{l}(M, \epsilon) = \left\lceil \frac{\log \left( \frac{\epsilon}{d_M(X)} \right)}{\log(\lambda_{m+1}s_{max})} \right\rceil
\]

Over-smoothing Layer

• How to alleviate over-smoothing?

Relaxed $\epsilon$-smoothing layer

$$\hat{l}(\mathcal{M}, \epsilon) = \left[ \frac{\log \left( \frac{\epsilon}{d_{\mathcal{M}}(X)} \right)}{\log(\lambda_{m+1} s_{max})} \right]$$

Adjacency Matrix, Weights

Training Deeper GNNs

- Why do we need deeper GNNs?
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
- How to alleviate over-smoothing?
  - Deal with adjacency matrix
  - Deal with weights
- How to overcome training dynamics?
Alleviate Over-Smoothing by Adjacency Matrix

Relaxed $\epsilon$-smoothing layer

$$\hat{l}(\mathcal{M}, \epsilon) = \left[ \frac{\log\left(\frac{\epsilon}{d_{\mathcal{M}}(X)}\right)}{\log(\lambda_{m+1} s_{max})} \right]$$

How adjacency matrix affects on over-smoothing?

$$\lambda_{m+1} \uparrow \Rightarrow \log(\lambda_{m+1} s_{max}) \uparrow \Rightarrow \hat{l}(\mathcal{M}, \epsilon) \uparrow$$

So how to increase $\lambda_{m+1}$?
Alleviate Over-Smoothing by Adjacency Matrix

So how to increase $\lambda_{m+1}$? Drop Edges!

When drop edges:

- The spread speed of the information is decreased

\[
\text{The relaxed smoothing layer only increases: } \hat{l}(\mathcal{M}, \epsilon) \leq \hat{l}(\mathcal{M}', \epsilon);
\]

- The dimension of subspace increases as the number of connected components increases

\[
\text{The information loss is decreased: } N - \text{dim}(\mathcal{M}) > N - \text{dim}(\mathcal{M}').
\]

## Alleviate Over-Smoothing by Adjacency Matrix

### DropEdge results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>4 layers</th>
<th>DropEdges</th>
<th>16 layers</th>
<th>DropEdges</th>
<th>64 layers</th>
<th>DropEdges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citeseer</td>
<td>76.7</td>
<td>79.2(+2.5)</td>
<td>65.2</td>
<td>76.8(+11.6)</td>
<td>44.6</td>
<td>45.6(+1.0)</td>
</tr>
<tr>
<td>GCN</td>
<td>78.9</td>
<td>78.8(-0.1)</td>
<td>78.2</td>
<td>79.4(+1.2)</td>
<td>21.2</td>
<td>75.3(+54.1)</td>
</tr>
<tr>
<td>ResGCN</td>
<td>79.1</td>
<td>80.2(+1.1)</td>
<td>78.8</td>
<td>80.1(+1.3)</td>
<td>76.7</td>
<td>80.0(+3.3)</td>
</tr>
<tr>
<td>JKNet</td>
<td>79.5</td>
<td>79.9(+0.4)</td>
<td>78.5</td>
<td>80.2(+1.7)</td>
<td>79.0</td>
<td>79.9(+0.9)</td>
</tr>
<tr>
<td>IncepGCN</td>
<td>77.3</td>
<td>79.2(+1.9)</td>
<td>72.9</td>
<td>74.5(+1.6)</td>
<td>16.9</td>
<td>25.1(+8.2)</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Training Deeper GNNs

を持っている理由は何ですか？

GNNs が簡単に深くなると行うことができるでしょうか？

何が GNNs の深さを制限していますか？

過ぼかしを和らげる方法はありますか？

- 隣接マトリックスを扱う
- 重みを扱う

どのように訓練ダイナミクスを克服しますか？
Alleviate Over-Smoothing by Weights

Similarly, increasing $s_{\text{max}}$ will increase the $\epsilon$-smoothing layer. So how to increase $s_{\text{max}}$? Increase the initial $W_l$s.

Weights
Alleviate Over-Smoothing by Weights

Try different $s_{max}$ as initial
Training Deeper GNNs

- Why do we need deeper GNNs?
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
  - Over-smoothing (Graph Specific)
  - Overfitting (Common)
  - Training dynamics (Common)
- How to alleviate over-smoothing?
- How to overcome training dynamics?
  - PairNorm
  - Shortcuts in Structures
Pair Norm: Center and Rescale

PairNorm: Center and rescale (normalize) GCN outputs $\tilde{X} := \text{GCN}(A, X)$ to keep the total pairwise squared distance unchanged

Center

$$\tilde{x}_i^c = \tilde{x}_i - \frac{1}{n} \sum_{i=1}^{n} \tilde{x}_i$$

Rescale

$$\hat{x}_i = s \sqrt{n} \frac{\tilde{x}_i^c}{\sqrt{\|	ilde{X}^c\|_F^2}}$$
Training Deeper GNNs

- Why do we need deeper GNNs?
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
- How to alleviate over-smoothing?
- How to overcome training dynamics?
  - Pair norm
  - Shortcuts in Structures
Structures with Shortcuts: Results

<table>
<thead>
<tr>
<th>Model</th>
<th>4 layers</th>
<th>16 layers</th>
<th>64 layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>76.7</td>
<td>65.2</td>
<td>44.6</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>77.3</td>
<td>72.9</td>
<td>16.9</td>
</tr>
<tr>
<td>ResGCN</td>
<td>78.9</td>
<td>78.2</td>
<td>21.2</td>
</tr>
<tr>
<td>JKNet</td>
<td>79.1</td>
<td>78.8</td>
<td>76.7</td>
</tr>
<tr>
<td>IncepGCN</td>
<td>79.5</td>
<td>78.5</td>
<td>79</td>
</tr>
<tr>
<td>APPNP</td>
<td>79.3</td>
<td>81.0</td>
<td>80.4</td>
</tr>
</tbody>
</table>

### Shortcuts in Structures

<table>
<thead>
<tr>
<th>Model</th>
<th>Input</th>
<th>Output</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResGCN</td>
<td>$H_{l+1} = f(A, H_l) + H_l$</td>
<td>$H_L = agg(H_1, ..., H_L)$</td>
<td></td>
</tr>
<tr>
<td>JKNet</td>
<td>$H_{l+1} = f(A, H_l)$</td>
<td>$H_{L+1} = agg(H_1, ..., H_L)$</td>
<td></td>
</tr>
<tr>
<td>IncepGCN</td>
<td>$H_{p,l+1} = f(A, H_{p,l}) p &gt; l + 1$</td>
<td>$H_{P+1} = agg(H_{1,1}, ..., H_{P,P})$</td>
<td></td>
</tr>
<tr>
<td>APPNP</td>
<td>$Z_{l+1} = (1 - \beta)AZ_l + \beta H_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GCN with Structures

**General Case**

\[ d_M(H_{l+1}) - r \leq \nu(d_M(H_l) - r) \]

**Basic GCN**

- Non-linear GCN
  \[
  \nu = s_l \lambda_{m+1} \\
  r = 0
  \]

- GCN with bias
  \[
  \nu = s_l \lambda_{m+1} \\
  r = \frac{d_M(b_{max})}{1 - \nu}
  \]

**Different Structures**

- ResGCN
  \[
  \nu = s_l \lambda_{m+1} + \alpha \\
  r = 0
  \]

- APPNP
  \[
  \nu = (1 - \beta) \lambda_{m+1} \\
  r = \frac{\beta d_M(H_0)}{1 - \nu}
  \]

Training Deeper GNNs

- Why do we need deeper GNNs?
  - Deeper GNNs gain **expressive power** with larger receptive fields
- Can GNNs simply go deeper?
- What impedes GNNs to go deeper?
- How to alleviate over-smoothing?
- How to overcome training dynamics?
Why do we need deeper GNNs?
- Deeper GNNs gain **expressive power** with larger receptive fields

Can GNNs simply go deeper?
- Not all. Some **underlying reasons** prevent GNNs from getting deeper

What impedes GNNs to go deeper?

How to alleviate over-smoothing?

How to overcome training dynamics?
Training Deeper GNNs

Why do we need deeper GNNs?
- Deeper GNNs gain **expressive power** with larger receptive fields

Can GNNs simply go deeper?
- Not all. Some **underlying reasons** prevent GNNs from getting deeper

What impedes GNNs to go deeper?
- **Over-smoothing (Graph Specific)**
  - Overfitting (Common)
  - Training dynamics (Common)

How to alleviate over-smoothing?

How to overcome training dynamics?
Training Deeper GNNs

Why do we need deeper GNNs?
- Deeper GNNs gain expressive power with larger receptive fields

Can GNNs simply go deeper?
- Not all. Some underlying reasons prevent GNNs from getting deeper

What impedes GNNs to go deeper?
- Over-smoothing (Graph Specific)
- Overfitting (Common)
- Training dynamics (Common)

How to alleviate over-smoothing?
- Deal with adjacency matrix
- Deal with weights

How to overcome training dynamics?
Training Deeper GNNs

Why do we need deeper GNNs?
- Deeper GNNs gain **expressive power** with larger receptive fields

Can GNNs simply go deeper?
- Not all. Some **underlying reasons** prevent GNNs from getting deeper

What impedes GNNs to go deeper?
- **Over-smoothing (Graph Specific)**
- Overfitting (Common)
- Training dynamics (Common)

How to alleviate over-smoothing?
- Deal with adjacency matrix
- Deal with weights

How to overcome training dynamics?
- Pair norm
- Shortcuts in Structures
Scalability of GNNs
Graph in the Real-world Can be Very Large

- **Large scale:**

- **Large number:**

  ![Graph showing the number of monthly active Facebook users worldwide as of 2nd quarter 2020](image-url)

  **ZINC15**

  Welcome to ZINC, a free database of commercially-available compounds for virtual screening. ZINC contains over 230 million purchasable compounds in ready-to-dock, 3D formats. ZINC also contains over 750 million purchasable compounds you can search for analogs in under a minute.

  **Getting Started**

  - Getting Started
  - What's New
  - About ZINC 15 Resources
  - Current Status / In Progress
  - Why are ZINC results "estimates"?

  **Ask Questions**

  You can use ZINC for general questions such as:

  - How many substances in current clinical trials have failed?
  - How many natural products have names in ZINC and are SMILES, names and calculated logP?
  - How many endogenous human metabolites are there?
Three Paradigms

• Why the original GNN fails on large graph?
  • Large memory requirement.
  • Inefficient gradient update.

• Three paradigms toward large-scale GNN:
  - Node-wise Sampling
  - Layer-wise Sampling
  - Graph-wise Sampling

- Full GNN
Two issues towards the large-scale GNNs

• How to design efficient sampling algorithm?
• How to guarantee the sampling quality?
Overview

GraphSAGE
Node-wise

FastGCN
Layer-wise

ClusterGCN
Graph-wise

2017

VRGCN
Node-wise

ASGCN
Layer-wise

GraphSAINT
Graph-wise

2020

Acknowledgement: The idea of this slide is taken from: https://drive.google.com/file/d/1Va7WaX0c5S37v7z79WllbZ-H54mXEly8x8/view?usp=sharing
Node-wise Sampling: GraphSAGE

- The architecture:

\[ h^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{h^{k-1}_u, \forall u \in \mathcal{N}(v)\}) \]

\[ h^k_v \leftarrow \sigma(W^k \text{CONCAT}(h^{k-1}_v, h^k_{\mathcal{N}(v)})) \]

Generalized Aggregator
- Mean aggregator (GCN)
- Pooling aggregator
- LSTM aggregator
- ....

Use Concatenation instead of SUM
Towards large-scale GraphSAGE

Mini-batch training

- Sample target nodes as a mini-batch;
- Only consider the nodes that used to compute the representation in the batch.
## Towards large-scale GraphSAGE

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Mini-batch training" /></td>
<td><img src="image2" alt="Mini-batch training" /></td>
<td><img src="image3" alt="Mini-batch training" /></td>
</tr>
<tr>
<td>Sampled Nodes: 9</td>
<td>Sampled Nodes: 6</td>
<td>Sampled Nodes: 1</td>
</tr>
</tbody>
</table>

- Sample target nodes as a mini-batch;
- Only consider the nodes that used by computing the representation in the batch.

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4" alt="Fixed-size neighbor sampling" /></td>
<td><img src="image5" alt="Fixed-size neighbor sampling" /></td>
<td><img src="image6" alt="Fixed-size neighbor sampling" /></td>
</tr>
<tr>
<td>Sampled Nodes: 5</td>
<td>Sampled Nodes: 3</td>
<td>Sampled Nodes: 1</td>
</tr>
</tbody>
</table>

- Sample fixed-size ($S_i$ for layer $i$) set of neighbors for computing.
- Number of nodes at input layer: $O(|V|^K) \rightarrow O(\prod_{i=1}^{K} S_i)$

---

Node-wise Sampling: GraphSAGE

- Pros:
  - Generalized aggregator.
  - Mini-batch training and fixed-size neighbor sampling.

- Cons:
  - Neighborhood expansion on deeper GNNs.
  - No guarantees for the sampling quality.

Node-wise Sampling: VR-GCN

• GraphSAGE: NS Sampler

\[ \text{NS}_u^{(l)} := R \sum_{v \in \hat{N}(u)} p_{uv} h_v^{(l)}, \quad R = N(u)/D^{(l)} \]

- biased sampling / larger variance \( \rightarrow \) larger sample size \( \hat{N}^{(l)}(u) \).

• Control Variate Based Estimator (CV Sampler):
  - Maintain the historical hidden embedding \( \bar{h}_v^{(l)} \) for a better estimation.
  - Variance reduction \( \rightarrow \) Variance elimination \( \rightarrow \) Smaller sample size \( \hat{n}^{(l)}(u) \).

• VR-GCN:

\[ H^{(l+1)} = \sigma \left( \tilde{P}^{(l)} (H^{(l+1)} - \bar{H}^{(l)}) + P \bar{H}^{(l)} \right) W^{(l)} \]

- One more thing: CVD Sampler -- Control Variate for Dropout.
Node-wise Sampling: VR-GCN

- **Pros:**
  - Analyze the variance reduction on node-wise sampling.
  - Successfully reducing the size of samples.

- **Cons:**
  - Additional memory consuming for storing the historical hidden embeddings.
Layer-wise Sampling: FastGCN

The functional generalization of GCN

\[ L = E_{v \sim P} \left[ g \left( h^{(M)}(v) \right) \right] = \int g \left( h^{(M)}(v) \right) dP(v) \]

FastGCN: sampling fixed number of nodes at each layer.

Integral Transform

The estimation

\[ L_{t_0, t_1, \ldots, t_M} := \frac{1}{t_M} \sum_{i=1}^{t_M} g(h^{(M)}_{t_M}(u^{(M)}_i)) \]
Layer-wise Sampling: FastGCN

- Towards the variance reduction: importance sampling.

\[ q(u) = ||\hat{A}(\cdot, u)||^2 / \sum_{u' \in V} ||\hat{A}(\cdot, u')||^2, \quad u \in V \]

This sampling probability keeps the same for each layer.
Layer-wise Sampling: FastGCN

• Pros:
  • Avoid neighborhood expansion problem.
  • Sample method with quality guarantee.

• Cons:
  • Failed to capture the between-layer correlations.
  • Performance compromise.

FastGCN
The number of samples per layer: 3

Chen, Jie, Tengfei Ma, and Cao Xiao. “FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling.”
Layer-wise Sampling: ASGCN

- Adaptive layer-wise sampling:
  - The sampling probability of lower layers depends on the upper ones.

\[
q(u_j) = \frac{p(u_j | v_i)}{p(u_j | v_1 \ldots v_n)} \quad s.t. \quad p(u_j | v_i) = \frac{\hat{a}(v_i, u_j)}{N(v_i)} , N(v_i) = \sum_{j=1}^{n} \hat{a}(v_i, u_j)
\]

- The probability of sampling node \(u_j\) given node \(v_i\).
- The entry of node \(v_i\) and \(u_j\) in re-normalization of the adjacency matrix \(\hat{A}\).
- The output hidden embeddings of node \(v_i\).
- \(x(u_j)\): the node feature of node \(u_j\).
- \(p(u_j | v_i)\): the probability of sampling node \(u_j\) given node \(v_i\).
- \(\hat{a}(v_i, u_j)\): The entry of node \(v_i\) and \(u_j\) in re-normalization of the adjacency matrix \(\hat{A}\).
- \(\mu_q(v_i)\): the output hidden embeddings of node \(v_i\).

Layer-wise Sampling: ASGCN

• Parameterized for explicit variance reduction.

\[ q^*(u_j) = \frac{\sum_{i=1}^{n} p(u_j|v_i)g(x(u_j))}{\sum_{j=1}^{N} \sum_{i=1}^{n} p(u_j|v_i)g(x(v_j))} \cdot g(x(u_j)) = W_g x(u_j) \]

• Optimize the sampler \( q^*(u_j) \) to minimize the variance:

\[ \mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_c(y_i, \hat{y}(\hat{\mu}_q(v_i))) + \lambda \text{Var}_q(\hat{\mu}_q(v_i)) \]

\( x(u_j) \): the node feature of node \( u_j \).
\( p(u_j|v_i) \): the probability of sampling node \( u_j \) given node \( v_i \).
\( \hat{a}(v_i,u_j) \): The entry of node \( v_i \) and \( u_j \) in re-normalization of the adjacency matrix \( \hat{A} \).
\( \hat{\mu}_q(v_i) \): the output hidden embeddings of node \( v_i \).
Layer-wise Sampling: ASGCN

- Self-dependent shared attention:
  \[ a(x(v_i), x(u_j)) = \frac{1}{n} \text{ReLU}(W_1g(x(v_i)) + W_2g(x(u_j))) \]

- Preserving second-order proximities by skip connections:
  \[ h_{\text{skip}}^{(l+1)}(v_i) = \sum_{j=1}^{n} \hat{a}_{\text{skip}}(v_i, s_j) h^{(l-1)}(s_j) W_{\text{skip}}^{(l-1)}, \quad i = 1, \ldots, n, \]
  where, \( \hat{a}_{\text{skip}} \) is estimated by:
  \[ \hat{a}_{\text{skip}}(v_i, s_j) \approx \sum_{k=1}^{n} \hat{a}(v_i, u_k) \hat{a}(u_k, s_j). \]
Layer-wise Sampling: ASGCN

- **Pros:**
  - Good performance.
  - Better variance control.

- **Cons:**
  - Additional dependence during sampling.

---

Github: https://github.com/huangwb/AS-GCN

Graph-wise Sampling: ClusterGCN

- Extract small clusters based on efficient clustering algorithms.

\[ \tilde{G} = [G_1, \ldots, G_c] = \{ \{ V_1, E_1 \}, \ldots, \{ V_c, E_c \} \}, \]

\[ \tilde{A} = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix}, \]

- Random batching at the subgraph level.

Graph-wise Sampling: ClusterGCN

- Neighbor expansion control.

Fix-size neighbor sampling $S=2$

Only sample the nodes in the clusters

Graph-wise Sampling: ClusterGCN

- **Pros:**
  - Good performance / Good memory usage.
  - Alleviate the neighborhood expansion problem in traditional mini-batch training.

- **Cons:**
  - Empirical results without analyzing the sampling quality.

Graph-wise Sampling: GraphSAINT

- Directly sample a subgraph for mini-batch training according to subgraph sampler.
- Sampler construction

**Node sampler**

Uniformly sample nodes.

**Edge sampler**

Sample edge with probably $p_{u,v} \propto 1/d_u + 1/d_v$

**Random walk sampler**

Sample edge with probably $p_{u,v} \propto B_{u,v} + B_{v,u}$
- $B_{u,v}$: the probability of a random walk to start at $u$ and end at $v$ in $L$ hops.
Graph-wise Sampling: GraphSAINT

• How to eliminate the bias introduce by the sampler?
  • Loss normalization:
    \[ \mathcal{L}_{\text{batch}} = \sum_{v \in G} L_v / \lambda_v, \quad \lambda_v = |V| p_v. \]
  • Aggregation normalization:
    \[ a(u, v) = p_{u,v} / p_v \]

\( p_v \): the probability of a node \( v \in V \) being sampled.
\( p_{u,v} \): the probability of an edge \( (u, v) \in E \) being sampled.
Graph-wise Sampling: GraphSAINT

• How to reduce the sampling variance?
  • The optimal edge probability for variance minimization:
    \[ p_{u,v} \propto 1/d_u + 1/d_v \]

\( p_v \): the probability of a node \( v \in V \) being sampled.
\( p_{u,v} \): the probability of an edge \((u, v) \in E\) being sampled.

Sample “—” more
Sample “—” less
Graph-wise Sampling: GraphSAINT

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Degree</th>
<th>Feature</th>
<th>Classes</th>
<th>Train / Val / Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPI</td>
<td>14,755</td>
<td>225,270</td>
<td>15</td>
<td>50</td>
<td>121 (m)</td>
<td>0.66 / 0.12 / 0.22</td>
</tr>
<tr>
<td>Flickr</td>
<td>89,250</td>
<td>899,756</td>
<td>10</td>
<td>500</td>
<td>7 (s)</td>
<td>0.50 / 0.25 / 0.25</td>
</tr>
<tr>
<td>Reddit</td>
<td>232,965</td>
<td>11,606,919</td>
<td>50</td>
<td>602</td>
<td>41 (s)</td>
<td>0.66 / 0.10 / 0.24</td>
</tr>
<tr>
<td>Yelp</td>
<td>716,847</td>
<td>6,977,410</td>
<td>10</td>
<td>300</td>
<td>100 (m)</td>
<td>0.75 / 0.10 / 0.15</td>
</tr>
<tr>
<td>Amazon</td>
<td>1,598,960</td>
<td>132,169,734</td>
<td>83</td>
<td>200</td>
<td>107 (m)</td>
<td>0.85 / 0.05 / 0.10</td>
</tr>
</tbody>
</table>

- Highly flexible and extensible (graph samplers, GNN architectures, etc.)
- Good performance (accuracy, speed)

Figure 2: Convergence curves of 2-layer models on GraphSAINT and baselines

Zeng, Hanqing, et al. "GraphSAINT: Graph Sampling Based Inductive Learning Method."
Summary

GraphSAGE
Node-wise
[o] Theory
[o] Performance
[o] Efficiency

FastGCN
Layer-wise
[+] Theory
[-] Performance
[+] Efficiency

ClusterGCN
Graph-wise
[-] Theory
[+] Performance
[o] Efficiency

VRGCN
Node-wise
[+] Theory
[o] Performance
[-] Efficiency

ASGCN
Layer-wise
[+] Theory
[o] Performance
[+] Efficiency

GraphSAINT
Graph-wise
[o] Theory
[+] Performance
[+] Efficiency

Future Directions
More efficient sampling...
Heterogenous/Dynamic graph...
System deployment …
Complex GNN architectures…
....

Acknowledgement: The idea of this slide is taken from: https://drive.google.com/file/d/1Va7WaXOcNS37z779WLbZ-H54mXEJy8cB/view?usp=sharing
Self/Un-Supervised Learning of GNNs
What we discussed before are supervised

- **Labels are scarce**, e.g. molecular property
- **Training/Testing tasks are Non I.I.D.**
## Existing Self-Supervised GNNs

<table>
<thead>
<tr>
<th>Predictive Methods</th>
<th>Node-Classification</th>
<th>Graph-Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information-based Methods</td>
<td>DGI [8], GMI [9]</td>
<td>InfoGraph [10]</td>
</tr>
</tbody>
</table>

“In self-supervised learning, the system learns to predict part of its input from other parts of its input.” ---- by Yann Lecun

Graphs are highly structured!
Node classification

- Two typical ways to formulate training loss

I. Enforcing Adjacent Similarity
- VGAE, GraphSAGE

II. Reconstruction from Neighbors
- EP-B
I Enforcing Adjacent Similarity

• GraphSAGE (Hamilton et al. 2017)

Enforcing nearby nodes to have similar representations, while enforcing disparate nodes to be distinct:

\[
\min \ -E_{u \sim N(v)} \log(\sigma(h^T_u h_v)) - \lambda E_{v_n \sim P_n(v)}[\log(\sigma(-h^T_{v_n} h_v))]
\]

Positive Samples \quad \text{Negative Samples}

- \(h_v\): representation of target node;
- \(h_u\): representation of neighbor/positive node;
- \(h_{v_n}\): representation of disparate/negative node;
- \(P_n(v)\): negative sampling.
II Reconstruction from neighbors

• EP-B (Durán & Niepert, 2017)

The objective is to minimize the reconstruction error (regulated by the error to other nodes):

\[
\min \sum_{u \in V \setminus \{v\}} \left[ \gamma + d(\hat{h}_v, h_u) - d(\bar{h}_v, h_u) \right]
\]

Hinge loss

Positive Samples
Negative Samples

\( h_v \): representation of target node;
\( h_u \): representation of nodes except \( v \);
\( \hat{h}_v \): \( \text{AGG}(h_l | l \in N(v)) \) is the reconstruction from neighbors;
\( \gamma \): the bias
How about graph classification/regression?

Toxicity?
Solubility?
…
N-Gram Graph

- (Liu et al. 2019)

Stage I: Node Representation

First learn node representations by CBOW-like pipeline

\[ h_v = \sum_{i \in \mathcal{P}} f'_i \]

Stage II: Graph Representation

For all n-gram paths:

\[ f_p = \prod_{i \in \mathcal{P}} h_i \]

\[ f(n) = \sum_{p \in \text{n-gram}} f_p \]

Graph Representation: \[ F = [f(1), \ldots, f(T)] \]

Equivalent to a GNN that needs no training
PreGNN: node- and graph-level pretraining

- (Hu et al. 2020)

Stage I: Node Representation

Stage II: Graph Representation

First learn node representations by *Context Prediction or Attribute Masking*

Then perform graph-level multi-task *Supervised Training*

\[
h_G = \text{Readout}(h_v | v \in G)
\]

\[
\min \text{ CrossEntropy}(h_G, y_G)
\]

Both node- and graph-level training are crucial!
(Hu et al. 2020)

Stage I: Node Representation

Enforcing node representation to be similar to its contextual structures:

\[
\min - \log \left( \sigma \left( (h_v^K)^T C_v^G \right) \right) - I(v \neq v') \log \left( 1 - \sigma \left( (h_v^K)^T C_{v'}^{G'} \right) \right)
\]

Positive Samples  Negative Samples

Degenerates to EP-B, if \( r1=0, r2=K=1 \)

**Context Prediction**

**Attribute Masking**

- \( h_v^K \): K-hop information
- \( C_v^G \): Structures between \( r1 \) and \( r2 \)-hop

Hu et al. Strategies for Pre-training Graph Neural Networks.
• (Hu et al. 2020)

Stage I: Node Representation

Mask random node/edge attribute and predict it, just like Bert:
GCC: contrastive learning

• (Qiu et al. 2020)

Both N-Gram Graph and PreGNN do not perform graph-level unsupervised training:

\[
\min - \log \frac{\exp(q^T k_+ / \tau)}{\sum_{i=0}^{K} \exp(q^T k_i / \tau)}
\]

\(q\): representation of different graphs;
\(k\): key of different graphs;
\(k_+\): positive key generated by random graph perturbation
\(\tau\): temperature

But, GCC only conducts graph-level pre-training, without node-level distinguishment

Qiu et al. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training.
## GROVER (Rong et al. 2020)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Node-Level Self-Supervised</th>
<th>Graph-Level Self-Supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Gram Graph</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>PreGNN</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>GCC</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Grover</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
• (Rong et al. 2020)

Stage I: Node/Edge-level pretraining

Predicting node/edge contexts instead of node labels can better capture local structures (multi-label)

Rong et al. GROVER: Self-supervised Message Passing Transformer on Large-scale Molecular Data.
Stage II: Graph-level pretraining

Predicting a graph if contains pre-defined graph motifs.

(Rong et al. 2020)
• One more thing: GTransformer

We build a more expressive and transformer-alike model: GTransformer

- Output for both node embedding and edge embedding.
- Multi-Head Attention: model global interaction between nodes/edges.
- Long-range Residual Connection: alleviating the vanishing gradient and over-smoothing.
- MPNN: Extract local structural information of graphs.
- dyMPN: Randomize the message passing hops for the dynamic receptive field modeling.

Rong et al. GROVER: Self-supervised Message Passing Transformer on Large-scale Molecular Data.
We pre-train GROVER with 100 million parameters on 10 million unlabeled molecules collected from ZINC15 and Chembl
Molecular classification

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BBBP # Molecules</th>
<th>SIDER 1427</th>
<th>ClinTox 1478</th>
<th>BACE 1513</th>
<th>Tox21 7831</th>
<th>ToxCast 8575</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF_Robust [39]</td>
<td>0.860 (0.087)</td>
<td>0.607 (0.033)</td>
<td>0.765 (0.085)</td>
<td>0.824 (0.022)</td>
<td>0.698 (0.012)</td>
<td>0.585 (0.031)</td>
</tr>
<tr>
<td>GraphConv [23]</td>
<td>0.877 (0.036)</td>
<td>0.593 (0.035)</td>
<td>0.845 (0.051)</td>
<td>0.854 (0.011)</td>
<td>0.772 (0.041)</td>
<td>0.650 (0.025)</td>
</tr>
<tr>
<td>Weave [22]</td>
<td>0.837 (0.065)</td>
<td>0.543 (0.034)</td>
<td>0.823 (0.023)</td>
<td>0.791 (0.008)</td>
<td>0.741 (0.044)</td>
<td>0.678 (0.024)</td>
</tr>
<tr>
<td>SchNet [44]</td>
<td>0.847 (0.024)</td>
<td>0.545 (0.038)</td>
<td>0.717 (0.042)</td>
<td>0.750 (0.033)</td>
<td>0.767 (0.025)</td>
<td>0.679 (0.021)</td>
</tr>
<tr>
<td>MPNN [13]</td>
<td>0.913 (0.041)</td>
<td>0.595 (0.030)</td>
<td>0.879 (0.054)</td>
<td>0.815 (0.044)</td>
<td>0.808 (0.024)</td>
<td>0.691 (0.013)</td>
</tr>
<tr>
<td>DMPNN [61]</td>
<td>0.919 (0.030)</td>
<td>0.632 (0.023)</td>
<td>0.897 (0.040)</td>
<td>0.852 (0.053)</td>
<td>0.826 (0.023)</td>
<td>0.718 (0.011)</td>
</tr>
<tr>
<td>MGCN [29]</td>
<td>0.850 (0.064)</td>
<td>0.552 (0.018)</td>
<td>0.634 (0.042)</td>
<td>0.734 (0.030)</td>
<td>0.707 (0.016)</td>
<td>0.663 (0.009)</td>
</tr>
<tr>
<td>AttentiveFP [59]</td>
<td>0.908 (0.050)</td>
<td>0.605 (0.060)</td>
<td>0.933 (0.020)</td>
<td>0.863 (0.015)</td>
<td>0.807 (0.020)</td>
<td>0.579 (0.001)</td>
</tr>
<tr>
<td>N-GRAM [28]</td>
<td>0.912 (0.013)</td>
<td>0.632 (0.005)</td>
<td>0.855 (0.037)</td>
<td>0.876 (0.035)</td>
<td>0.769 (0.027)</td>
<td>0.714 (0.019)</td>
</tr>
<tr>
<td>HU. et.al. [18]</td>
<td>0.915 (0.040)</td>
<td>0.614 (0.006)</td>
<td>0.762 (0.058)</td>
<td>0.851 (0.027)</td>
<td>0.811 (0.015)</td>
<td>0.714 (0.019)</td>
</tr>
<tr>
<td>GROVERbase</td>
<td>0.936 (0.008)</td>
<td>0.656 (0.06)</td>
<td>0.925 (0.013)</td>
<td>0.878 (0.016)</td>
<td>0.819 (0.020)</td>
<td>0.723 (0.010)</td>
</tr>
<tr>
<td>GROVERlarge</td>
<td>0.940 (0.019)</td>
<td>0.658 (0.023)</td>
<td>0.944 (0.021)</td>
<td>0.894 (0.028)</td>
<td>0.831 (0.025)</td>
<td>0.737 (0.010)</td>
</tr>
</tbody>
</table>

Rong et al. GROVER: Self-supervised Message Passing Transformer on Large-scale Molecular Data.
## Existing Self-supervised models

<table>
<thead>
<tr>
<th>Predictive Methods</th>
<th>Node-Classification</th>
<th>Graph-Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information-based Methods</td>
<td>DGI [8], GMI [9]</td>
<td>InfoGraph [10]</td>
</tr>
</tbody>
</table>

What makes a good representation?

Auto-Encoder (AE)

“One natural criterion that we may expect any good representation to meet, at least to some degree, is to **retain a significant amount of information about the input.**” by Vincent et al. 2010

Hinton & Salakhutdinov 2006; Vincent et al. 2010
What makes a good representation?

- A more direct way, other than AE?
  - Yes, **Mutual Information (MI)**.

\[
I(X; Y) = D_{KL}(p(X)p(Y)||p(X, Y)) = H(X) - H(X|Y)
\]

- \( 0 \leq I(X; Y) \leq H(X) \) or \( H(Y) \);
- \( I(X; Y) = 0 \) iff \( X \) and \( Y \) are independent random variables;
- \( I(X; Y) = H(X) = H(Y) \), if \( X \) and \( Y \) are determinately related, i.e. \( H(X|Y) = 0 \)
AE is a lower bound of MI

(Hjelm et al. 2019)

\[
I(X; Y) = H(X) - H(X|Y) \geq H(X) - R(X|Y)
\]

Mutual Information

Reconstruction error

Computing MI is hard and not end-to-end, until recently (CPC, Oord et al., 2018; MINE, Belghazi et al., 2018; Nowozin et al., 2016; Hjelm et al. 2019)
Estimating/Maximizing MI (Hjelm et al. 2019)

\[ X, Y \xrightarrow{T_w} \hat{I}_{T_w}(X; Y) \]

\[ \max_{T_w} \hat{I}_{T_w}(X; Y) \rightarrow \max I(X; Y) \]

MINE (Belghazi et al., 2018):

\[ I_{\text{MINE}}(X; Y) \triangleq E_{p(x,y)}[T_w(x, y)] - \log E_{p(x)p(y)}[\exp(T_w(x, y))] \]

JSD MI estimator (Nowozin et al., 2016):

\[ I_{\text{JSD}}(X; Y) \triangleq E_{p(x,y)}[\log \sigma(T_w(x, y))] + E_{p(x)p(y)}[\log(1 - \sigma(T_w(x, y)))] \]

infoNCE MI estimator (Oord et al., 2018):

\[ I_{\text{NCE}}(X; Y) \triangleq E_{p(x,y)}[\log \frac{\exp T_w(x, y)}{\sum_{x' \sim p(x)} \exp T_w(x', y)}] \]
Deep Graph Infomax (DGI)

• (Velickovic et al. 2019)

The JSD MI estimator is applied:

$$\max_{GNN} I(X; A; h_i) \approx \max \log(D(h_i; X, A)) + \log(1 - D(\tilde{h}_i; X, A))$$

$$h_i = \text{GNN}(X, A)$$

$$\tilde{h}_i$$ negative sample
Deep Graph Infomax (DGI)

It is hard to directly compute $D(\tilde{h}_i; X, A)$, thus DGI resorts to readout $s = R(X, A)$:

\[
\max_{\text{GNN}} I(X, A; h_i) \approx \max \log(D(h_i; X, A)) + \log(1 - D(\tilde{h}_i; X, A))
\]

\[
\max_{\text{GNN}} I(X, A; h_i) = \max_{\text{GNN}} \log(D(h_i; s)) + \log(1 - D(\tilde{h}_i; s))
\]
It can be proved that, if the readout $s = R(X, A)$ is injective,

$$\log(D(h_i; s)) + \log\left(1 - D(\tilde{h}_i; s)\right) = \log(D(h_i; X, A)) + \log\left(1 - D(\tilde{h}_i; X, A)\right)$$

It can be also proved that, if $|X| = |s|$ is finite,

$$\max \log(D(h_i; s)) + \log\left(1 - D(\tilde{h}_i; s)\right) = \max I(h_i; X, A)$$
• Some issues in DGI

➢ Computing MI requires the injectivity of readout function

➢ It resorts to graph corruption to generate negative samples

➢ Distinct encoders and corruption functions for different tasks
GMI: Graphical Mutual Information

- (Peng et al. 2020)

The basic idea of GMI is to compute the MI directly.
The basic idea of GMI is to compute the MI directly.

$$I(X, A; h_i) \approx I(X; h_i) + \sum_{j \in N(i)} I(\sigma(h_i^T h_j); A_{ij})$$

- It is both feature- and edge-aware;
- No need to readout or corruption;
- Feature MI can be further decomposed;

The basic idea of GMI is to compute the MI directly.
GMI: Graphical Mutual Information

• (Peng et al. 2020)

It can be proved that, if certain mild condition meets,

\[ I(X; h_i) = \sum_{j \in N(i)} w_{ij} I(x_j; h_i), \text{ for } 0 \leq w_{ij} \leq 1 \]

The global MI is decomposed into a weighted sum of local MIs. It is not a bad idea to let \( w_{ij} = \sigma(h_i^T h_j) \)

We then apply the JSD MI estimator to compute \( I(x_j; h_i) \) and \( I(\sigma(h_i^T h_j); A_{ij}) \)
**GMI: Graphical Mutual Information**

- **Node Classification**

We use a universal backbone (GCN) for all tasks, different from DGI

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cora</th>
<th>Transductive Citeseer</th>
<th>PubMed</th>
<th>Inductive Reddit</th>
<th>Inductive PPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP-B loss</td>
<td>79.4 ± 0.1</td>
<td>69.3 ± 0.2</td>
<td>78.6 ± 0.2</td>
<td>93.8 ± 0.03</td>
<td>61.8 ± 0.04</td>
</tr>
<tr>
<td>DGI loss</td>
<td>82.2 ± 0.2</td>
<td>72.2 ± 0.2</td>
<td>78.9 ± 0.3</td>
<td>94.3 ± 0.02</td>
<td>62.3 ± 0.02</td>
</tr>
<tr>
<td>FMI (ours)</td>
<td>78.3 ± 0.1</td>
<td>72.0 ± 0.2</td>
<td>79.1 ± 0.3</td>
<td>94.7 ± 0.03</td>
<td>64.8 ± 0.03</td>
</tr>
<tr>
<td>GMI-mean (ours)</td>
<td>82.7 ± 0.1</td>
<td>73.0 ± 0.3</td>
<td>80.1 ± 0.2</td>
<td>95.0 ± 0.02</td>
<td>65.0 ± 0.02</td>
</tr>
<tr>
<td>GMI-adaptive (ours)</td>
<td>83.0 ± 0.3</td>
<td>72.4 ± 0.1</td>
<td>79.9 ± 0.2</td>
<td>94.9 ± 0.02</td>
<td>64.6 ± 0.03</td>
</tr>
</tbody>
</table>

Codes: [https://github.com/zpeng27/GMI](https://github.com/zpeng27/GMI)
GMI: Graphical Mutual Information

• Link Prediction

We use an universal backbone (GCN) for all tasks

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cora 20.0%</th>
<th>Cora 50.0%</th>
<th>Cora 70.0%</th>
<th>BlogCatalog 20.0%</th>
<th>BlogCatalog 50.0%</th>
<th>BlogCatalog 70.0%</th>
<th>Flickr 20.0%</th>
<th>Flickr 50.0%</th>
<th>Flickr 70.0%</th>
<th>PPI 22.7%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGI</td>
<td>95.6±0.3</td>
<td>94.6±0.4</td>
<td>94.4±0.2</td>
<td>77.2±0.4</td>
<td>76.4±0.4</td>
<td>75.5±0.3</td>
<td>90.3±0.3</td>
<td>89.0±0.4</td>
<td>74.1±0.7</td>
<td>77.4±0.1</td>
</tr>
<tr>
<td>FMI (ours)</td>
<td>97.2±0.2</td>
<td>95.2±0.1</td>
<td>95.0±0.1</td>
<td>81.2±0.2</td>
<td>79.5±0.4</td>
<td>75.1±0.2</td>
<td>92.7±0.3</td>
<td>92.2±0.3</td>
<td>90.6±0.4</td>
<td>79.8±0.2</td>
</tr>
<tr>
<td>GMI (ours)</td>
<td>97.9±0.3</td>
<td>96.4±0.2</td>
<td>96.3±0.1</td>
<td>84.1±0.3</td>
<td>83.6±0.2</td>
<td>82.5±0.1</td>
<td>92.0±0.2</td>
<td>90.1±0.3</td>
<td>88.5±0.2</td>
<td>80.0±0.2</td>
</tr>
</tbody>
</table>

Codes: https://github.com/zpeng27/GMI
Summarization

• Node classification
  • EP-B, GraphSAGE
  • DGI, GMI

• Graph classification
  • N-gram Graph, PreGNN, GCC, Grover
  • InfoGraph

Sun et al. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization, ICLR 2020
Applications

GNN in Social Networks
GNN in Social Networks

• "Semi-supervised graph classification: A hierarchical graph perspective." WWW 2019

• "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020
Hierarchical Graph Classification

- **Hierarchical Graph**: A set of graph instances are interconnected via edges.
  - Social network with group structure.
  - Document (graph-of-words) collection with citation relation.

- **The Problem**: predicts the class label of graph instances in a hierarchical graph.

Hierarchical Graph Classification

• **The Problem:** predicts the class label of graph instances in a hierarchical graph.

• **Challenges:**
  - How to represent the graphs with arbitrary size into a fixed-length vector?
  - How to incorporate the information of instance level and hierarchical level?
Self-Attentive Graph Embedding

• How to represent the graphs with arbitrary size into a fixed-length vector?
• Graph representation learning at different level:
  • Node Level: \( G(V, E) \rightarrow H^{n \times v} \)
  • Graph Level: \( G(V, E) \rightarrow e^v \)
• SAGE: Self-Attentive Graph Embedding
  • Size invariance ---- Self-attention
  • Permutation invariance ---- GCN Smoothing
  • Node importance ---- Self-attention
• Self-attention \( S : r \) opinions about node importance.

The Unified Model

• How to incorporate the information of instance level and hierarchical level?
  • **Instance Level Model**: Graph Level Learning (SEGA)
  • **Hierarchical Level Model**: Node Level Learning (GCN)

• **Feature Sharing**: Concatenate the output of SEGA to the input of GCN.

• **Disagreement Loss**: The disagreement between instance classifier and hierarchical classifier should be minimized.

$$\gamma_4$$

The overall loss: \(\min \zeta(G_I) + \zeta(G_U)\),

The supervised loss:

$$\zeta(G_I) = \sum_{g_i \in G_I} (\mathcal{L}(y_i, \psi_i) + \mathcal{L}(y_i, \gamma_i)),$$

The disagreement loss:

$$\zeta(G_U) = \sum_{g_i \in G_U} D_{KL}(\psi_i || \gamma_i),$$

One More Thing: SEAL-AI/CI

- How to deal with the small mount of class label?
- Semi-supervised Graph Classification (SEAL)
  - Active Learning Setting (SEAL-AI): Choose the instances with large disagreement loss to annotate.

• How to deal with the small mount of class label?
• Semi-supervised Graph Classification (SEAL)
  • Cautious Iteration Setting (SEAL-CI): Choose the instances with maximum predicted probability and annotate them with the predicted class label.

Algorithm 1: SEAL-CI

Input: A, X, Θ,
Output: Ψ(t), Γ(t).
1 Initial: G_{tmp} = ∅, G_0^t = G_I, t = 0;
2 while t ≤ U do
3 \( W^{t+1} \leftarrow \arg \min (G_0^t | W^t); \)
4 \( \Psi^{t+1}, E^{t+1} \leftarrow IC(A, X | W^{t+1}); \)
5 \( Γ^{t+1} \leftarrow HC(E^{t+1}, Θ | W^{t+1}); \)
6 \( G_{tmp} \leftarrow h(t, Γ^{t+1}); \)
7 \( G_1^{t+1} \leftarrow G_I \cup G_{tmp}; \)
8 \( G_{tmp} = ∅; \)
9 Return Ψ(t), Γ(t);
Performance of SAGE

- Performance of synthesized graphs
  
  Two-dimensional visualization of graph embeddings generated from the synthesized graph instances using SAGE.

- Performance on the protein classification task:

  Table 1: Statistics of PROTEINS and D&D

<table>
<thead>
<tr>
<th></th>
<th>PROTEINS</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max number of nodes</td>
<td>620</td>
<td>5748</td>
</tr>
<tr>
<td>Avg number of nodes</td>
<td>39.06</td>
<td>284.32</td>
</tr>
<tr>
<td>Number of graphs</td>
<td>1113</td>
<td>1178</td>
</tr>
</tbody>
</table>

  Table 2: Accuracy of different classifiers

<table>
<thead>
<tr>
<th>Approach</th>
<th>PROTEINS</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>75.07±0.54%</td>
<td>-</td>
</tr>
<tr>
<td>RW</td>
<td>74.22±0.42%</td>
<td>-</td>
</tr>
<tr>
<td>GK</td>
<td>71.67±0.55%</td>
<td>78.45±0.26%</td>
</tr>
<tr>
<td>WL</td>
<td>72.92±0.56%</td>
<td>77.95±0.70%</td>
</tr>
<tr>
<td>PSCN</td>
<td>75.89±2.76%</td>
<td>77.12±2.41%</td>
</tr>
<tr>
<td>graph2vec</td>
<td>73.30±2.05%</td>
<td>-</td>
</tr>
<tr>
<td>SAGE</td>
<td>77.26±2.28%</td>
<td>80.88±2.33%</td>
</tr>
</tbody>
</table>

Performance of SEAL

- Real Datasets
  - User-Group Data
  - # of users: 18M
  - # of groups: 37K
  - Target: Predicting group label.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Macro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*1 GK-SVM</td>
<td>48.8%</td>
</tr>
<tr>
<td>*1 WL-SVM</td>
<td>47.8%</td>
</tr>
<tr>
<td>*2 graph2vec-GCN</td>
<td>48.1%</td>
</tr>
<tr>
<td>*3 cautious-SAGE-Cheby</td>
<td>64.3%</td>
</tr>
<tr>
<td>*3 active-SAGE-Cheby</td>
<td>66.7%</td>
</tr>
<tr>
<td>*4 SAGE</td>
<td>54.7%</td>
</tr>
<tr>
<td>*4 SEAL-CI</td>
<td>70.8%</td>
</tr>
<tr>
<td><strong>SEAL-AI</strong></td>
<td><strong>73.2%</strong></td>
</tr>
</tbody>
</table>

- The visualization of the hierarchical graph for a “game” group.
GNN in Social Networks

• "Semi-supervised graph classification: A hierarchical graph perspective." WWW 2019

• "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020
**Bi-GCN: Rumor Detection**

- Rumor tree: rumors spread like a tree in the social network. Rumor has two major characteristics:
  - **Propagation**: deep spread *along a relationship chain*
  - **Dispersion**: widespread *across a social community*

Bi, Tian, et al. "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020.
Bi-GCN: Rumor Detection

• Bi-directed GCN:
  1. Construct directed **Propagation** and **Dispersion** graphs for rumors
  2. Calculate high-level node representations via **GCNs**
  3. Concatenate **root features** to enhance the performance
  4. Classify rumors from **Representations** of propagation and dispersion

---

Bian, Tian, et al. "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020.
Bi-GCN: Rumor Detection: results

• We tested on 3 datasets:
  • Twitter15
  • Twitter16
  • Weibo

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Acc.</th>
<th>Prec.</th>
<th>Rec.</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTC</td>
<td>F</td>
<td>0.831</td>
<td>0.847</td>
<td>0.815</td>
<td>0.831</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.815</td>
<td>0.824</td>
<td>0.819</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>F</td>
<td>0.879</td>
<td>0.777</td>
<td>0.656</td>
<td>0.708</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.579</td>
<td>0.708</td>
<td>0.615</td>
</tr>
<tr>
<td>SVM-TS</td>
<td>F</td>
<td>0.885</td>
<td>0.950</td>
<td>0.932</td>
<td>0.938</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.124</td>
<td>0.047</td>
<td>0.059</td>
</tr>
<tr>
<td>RvNN</td>
<td>F</td>
<td>0.908</td>
<td>0.912</td>
<td>0.897</td>
<td>0.905</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.904</td>
<td>0.918</td>
<td>0.911</td>
</tr>
<tr>
<td>PPC_RNN+CNN</td>
<td>F</td>
<td>0.916</td>
<td>0.884</td>
<td>0.957</td>
<td>0.919</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.955</td>
<td>0.876</td>
<td>0.913</td>
</tr>
<tr>
<td>Bi-GCN</td>
<td>F</td>
<td>0.961</td>
<td>0.961</td>
<td>0.964</td>
<td>0.961</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td>0.962</td>
<td>0.962</td>
<td>0.960</td>
</tr>
</tbody>
</table>

Twitter15

<table>
<thead>
<tr>
<th>Method</th>
<th>Acc.</th>
<th>N $F_1$</th>
<th>F $F_1$</th>
<th>T $F_1$</th>
<th>U $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTC</td>
<td>0.454</td>
<td>0.415</td>
<td>0.355</td>
<td>0.733</td>
<td>0.317</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>0.318</td>
<td>0.225</td>
<td>0.082</td>
<td>0.455</td>
<td>0.218</td>
</tr>
<tr>
<td>SVM-TS</td>
<td>0.544</td>
<td>0.796</td>
<td>0.472</td>
<td>0.404</td>
<td>0.483</td>
</tr>
<tr>
<td>SVM-TK</td>
<td>0.750</td>
<td>0.804</td>
<td>0.698</td>
<td>0.765</td>
<td>0.733</td>
</tr>
<tr>
<td>RvNN</td>
<td>0.723</td>
<td>0.682</td>
<td>0.758</td>
<td>0.821</td>
<td>0.654</td>
</tr>
<tr>
<td>PPC_RNN+CNN</td>
<td>0.477</td>
<td>0.359</td>
<td>0.507</td>
<td>0.300</td>
<td>0.640</td>
</tr>
<tr>
<td>Bi-GCN</td>
<td>0.886</td>
<td>0.891</td>
<td>0.860</td>
<td>0.930</td>
<td>0.864</td>
</tr>
</tbody>
</table>

Twitter16

<table>
<thead>
<tr>
<th>Method</th>
<th>Acc.</th>
<th>N $F_1$</th>
<th>F $F_1$</th>
<th>T $F_1$</th>
<th>U $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTC</td>
<td>0.473</td>
<td>0.254</td>
<td>0.080</td>
<td>0.190</td>
<td>0.482</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>0.553</td>
<td>0.670</td>
<td>0.085</td>
<td>0.117</td>
<td>0.361</td>
</tr>
<tr>
<td>SVM-TS</td>
<td>0.574</td>
<td>0.755</td>
<td>0.420</td>
<td>0.571</td>
<td>0.526</td>
</tr>
<tr>
<td>SVM-TK</td>
<td>0.732</td>
<td>0.740</td>
<td>0.709</td>
<td>0.836</td>
<td>0.686</td>
</tr>
<tr>
<td>RvNN</td>
<td>0.737</td>
<td>0.662</td>
<td>0.743</td>
<td>0.835</td>
<td>0.708</td>
</tr>
<tr>
<td>PPC_RNN+CNN</td>
<td>0.564</td>
<td>0.591</td>
<td>0.543</td>
<td>0.394</td>
<td>0.674</td>
</tr>
<tr>
<td>Bi-GCN</td>
<td>0.880</td>
<td>0.847</td>
<td>0.869</td>
<td>0.937</td>
<td>0.865</td>
</tr>
</tbody>
</table>

Bian, Tian, et al. "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020.
Bi-GCN: Rumor Detection: early detection

• Early detection of rumors

Github: https://github.com/TianBian95/BiGCN

Bian, Tian, et al. "Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks." AAAI 2020.
Applications

GNN in Medical Imaging
GNN in Medical Imaging

• "Graph CNN for Survival Analysis on Whole Slide Pathological Images", MICCAI 2018

• "Graph Convolutional Nets for Tool Presence Detection in Surgical Videos", IPMI 2019

• "Graph Attention Multi-instance Learning for Accurate Colorectal Cancer Staging", MICCAI 2020
• **Survival Prediction**
  
  • Predict the risk of a certain event occurs.
  • Event: part failure, drug adverse reaction or death.
  • Application: provides suggestion for clinical inte

• **Whole Slide Images**
  
  • Large: single WSI size >0.5 GB.
  • Complicated: millions of cells.
  • Combine local and global features.
\( \textbf{Cox} \ \lambda(t|X_i) = \lambda_0(t) \exp(\beta_1 X_{i1} + \cdots + \beta_p X_{ip}) = \lambda_0(t) \exp(X_i \cdot \beta) \)

\( \) \( \) \( \) \( \) \( \) \( \)

**Partial likelihood for event happens on subject** \( i \) : 

\[
L_i(\beta) = \frac{\lambda(Y_i|X_i)}{\sum_{j:Y_j \geq Y_i} \lambda(Y_i|X_j)} = \frac{\lambda_0(Y_i)\theta_i}{\sum_{j:Y_j \geq Y_i} \lambda_0(Y_i)\theta_j} = \frac{\theta_i}{\sum_{j:Y_j \geq Y_i} \theta_j}
\]

where, \( Y \) is the observation time.

**Join likelihood of all subjects:** 

\[
L(\beta) = \prod_{i:C_i=1} L_i(\beta)
\]

**Log likelihood as object function:** 

\[
\ell(\beta) = \sum_{i:C_i=1} \left( X_i \cdot \beta - \log \sum_{j:Y_j \geq Y_i} \theta_j \right)
\]
• Pathological Images and Patient Survival Time and Label
  • TCGA, The Cancer Genome Atlas
  • NLST, National Lung Screening Trials

<table>
<thead>
<tr>
<th>Database</th>
<th>Cancer Subtype</th>
<th>No. Patient</th>
<th>No. WSI</th>
<th>Quality</th>
<th>Avg. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCGA</td>
<td>LUSC</td>
<td>463</td>
<td>535</td>
<td>medium</td>
<td>0.72 GB</td>
</tr>
<tr>
<td>TCGA</td>
<td>GBM</td>
<td>365</td>
<td>491</td>
<td>low</td>
<td>0.50 GB</td>
</tr>
<tr>
<td>NLST</td>
<td>ADC &amp; SCC</td>
<td>263</td>
<td>425</td>
<td>high</td>
<td>0.74 GB</td>
</tr>
</tbody>
</table>

• Evaluation Metrics- C-index: the fraction of all pairs of patients whose predicted survival times are correctly ordered.
• Yellow regions: high attention values
• High attention patches: values > 0.9 (attention values (0, 1))
<table>
<thead>
<tr>
<th>Model</th>
<th>LUSC</th>
<th>GBM</th>
<th>NLST</th>
</tr>
</thead>
<tbody>
<tr>
<td>LASSO-Cox [19]</td>
<td>0.5280</td>
<td>0.5574</td>
<td>0.4738</td>
</tr>
<tr>
<td>LASSO-Cox*</td>
<td><strong>0.5663</strong></td>
<td>0.5165</td>
<td><strong>0.5663</strong></td>
</tr>
<tr>
<td>BoostCI [17]</td>
<td>0.5633</td>
<td>0.5543</td>
<td>0.5705</td>
</tr>
<tr>
<td>BoostCI*</td>
<td><strong>0.5800</strong></td>
<td>0.5130</td>
<td><strong>0.5716</strong></td>
</tr>
<tr>
<td>EnCox [20]</td>
<td>0.5216</td>
<td>0.5597</td>
<td>0.4883</td>
</tr>
<tr>
<td>EnCox*</td>
<td><strong>0.5740</strong></td>
<td>0.5231</td>
<td><strong>0.5742</strong></td>
</tr>
<tr>
<td>RSF [12]</td>
<td>0.5066</td>
<td>0.5570</td>
<td>0.5964</td>
</tr>
<tr>
<td>RSF*</td>
<td><strong>0.5492</strong></td>
<td>0.5193</td>
<td>0.5491</td>
</tr>
<tr>
<td>MTLSA [16]</td>
<td>0.5386</td>
<td>0.5787</td>
<td>0.6042</td>
</tr>
<tr>
<td>MTLSA*</td>
<td>0.5247</td>
<td>0.5630</td>
<td>0.5573</td>
</tr>
<tr>
<td>WSISA [21]</td>
<td>0.6380</td>
<td>0.5760</td>
<td>0.6539</td>
</tr>
<tr>
<td>GCN-Cox [8]</td>
<td>0.6280</td>
<td>0.5901</td>
<td>0.6845</td>
</tr>
<tr>
<td>DeepGraphSurv</td>
<td><strong>0.6606</strong></td>
<td><strong>0.6215</strong></td>
<td><strong>0.7066</strong></td>
</tr>
</tbody>
</table>

* Use our graph features for the survival model.
Future Directions

Graph Multi-instance Learning

Graph Attack

Inverse Graph Identification

Graph Defense

Hierarchical Graph

Hypergraph

Subgraph Recognition


https://en.wikipedia.org/wiki/Hypergraph

Attn
Bibliography
Bibliography

• Morris et al. Weisfeiler and Leman Go Neural Higher-order Graph Neural Networks. AAAI, 2019.
• Xu et al. How Powerful Are Graph Neural Networks. ICLR, 2019.
• Maron et al. Invariant and Equivariant Graph Networks. ICLR, 2019.
• Dehmamy et al. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology. NeurIPS, 2019.
• Garg et al. Generalization and Representational Limits of Graph Neural Networks. ICML, 2020.
• Shervashidze et al. Weisfeiler-Lehman Graph Kernels. JRML, 2011.
• Chen et al. On the equivalence between graph isomorphism testing and function approximation with GNNs. NeurIPS, 2019.
• Hu et al. Strategies for Pre-training Graph Neural Networks. ICLR, 2020.
• Qiu et al. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. KDD, 2020.
Bibliography

- Zeng, et al. Graph convolutional networks for temporal action localization. ICCV, 2019