Learning GNNs with CogDL

Yukuo Cen, Jie Tang

Knowledge Engineering Group (KEG)

Computer Science and Technology

Tsinghua University

CogDL is publicly available at https://github.com/THUDM/cogdl

Outline

- Preliminary
- Basic GNNs
- Advanced GNNs
- All with CogDL

Networked World

facebook

- 2.7 billion MAU
- 17 billion photos/day

Alibaba Group 阿里巴巴集团

- 2.3 billion trans. on 11/11
- ~500 billion GMV on 11/11

twitter

- 350 million MAU
- 5 million tweets/day

新浪微博 weibo.com

- 550 million MAU
- 12 billion pageview/day

Instagram

- 1.15 billion MAU
- 95 million pics/day

ByteDance 字节跳动

- ~1.9 billion MAU
- 70 minutes/user/day

snapchat

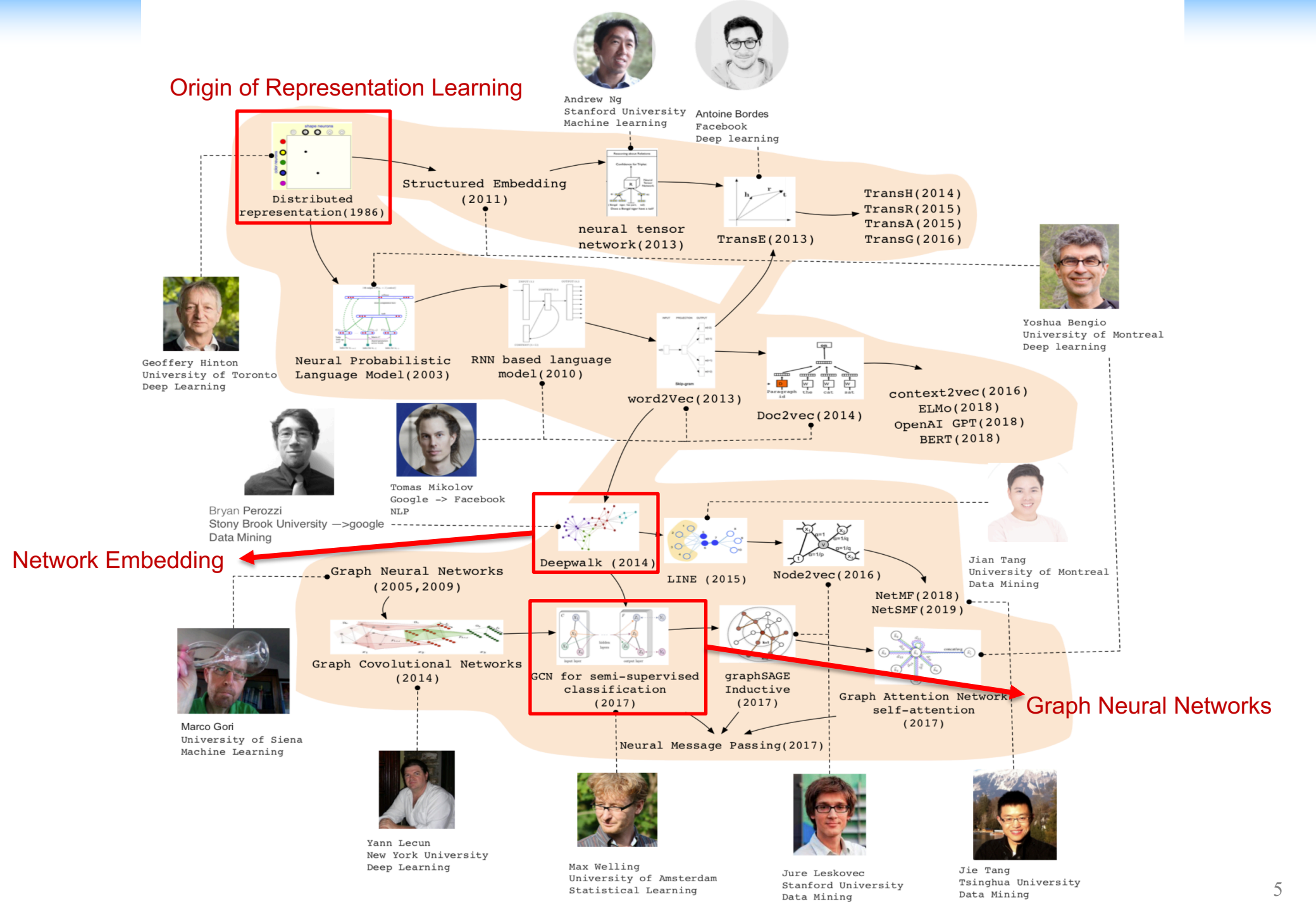
- 500 million MAU
- 30 minutes/user/day

WeChat

- QQ: 600 million MAU
- WeChat: 1.2 billion MAU

Machine Learning with Networks

- ML tasks in networks:
  - Node classification
    - Predict a type of a given node
  - Link prediction
    - Predict whether two nodes are linked
  - Community detection
    - Identify densely linked clusters of nodes
  - Network similarity
    - How similar are two (sub)networks?


Origin of Representation Learning
Andrew Ng
Stanford University
Machine learning
Antoine Bordes
Facebook
Deep learning
Distributed representation(1986)
Structured Embedding (2011)
neural tensor network(2013)
TransE(2013)
TransH(2014)
TransR(2015)
TransA(2015)
TransG(2016)
Geoffery Hinton
University of Toronto
Deep Learning
Neural Probabilistic Language Model(2003)
RNN based language model(2010)
word2Vec(2013)
Doc2vec(2014)
context2vec(2016)
ELMo(2018)
OpenAI GPT(2018)
BERT(2018)
Yoshua Bengio
University of Montreal
Deep learning
Bryan Perozzi
Stony Brook University —>google
Data Mining
Tomas Mikolov
Google -> Facebook
NLP
Network Embedding
Deepwalk (2014)
LINE (2015)
Node2vec(2016)
NetMF(2018)
NetSMF(2019)
Jian Tang
University of Montreal
Data Mining
Graph Neural Networks (2005,2009)
Graph Covolutional Networks (2014)
GCN for semi-supervised classification (2017)
graphSAGE Inductive (2017)
Graph Attention Network self-attention (2017)
Graph Neural Networks
Marco Gori
University of Siena
Machine Learning
Neural Message Passing(2017)
Yann Lecun
New York University
Deep Learning
Max Welling
University of Amsterdam
Statistical Learning
Jure Leskovec
Stanford University
Data Mining
Jie Tang
Tsinghua University
Data Mining

Why is it hard?

- Modern deep learning toolkit is designed for simple sequences or grids.
  - CNNs for fixed-size images/grids...
  - RNNs or word2vec for text/sequences...

- But networks are far more complex!
  - Complex topographical structure
  - No fixed node ordering or reference point
  - Often dynamic and have multimodal features.

Why is it hard? (cont.)

- Billion-scale real-world graphs!
  - How to store large-scale graphs
  - Large cost of model training
  - GPU memory bounded
- Potential issues in training GNNs
  - Over-fitting issue
  - Over-smoothing issue

CogDL Introduction

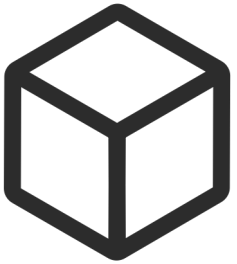
Vision

CogDL aims at providing researchers and developers with easy-to-use APIs, reproducible results, and high efficiency for most graph tasks and applications.

Philosophy

Easy-to-use

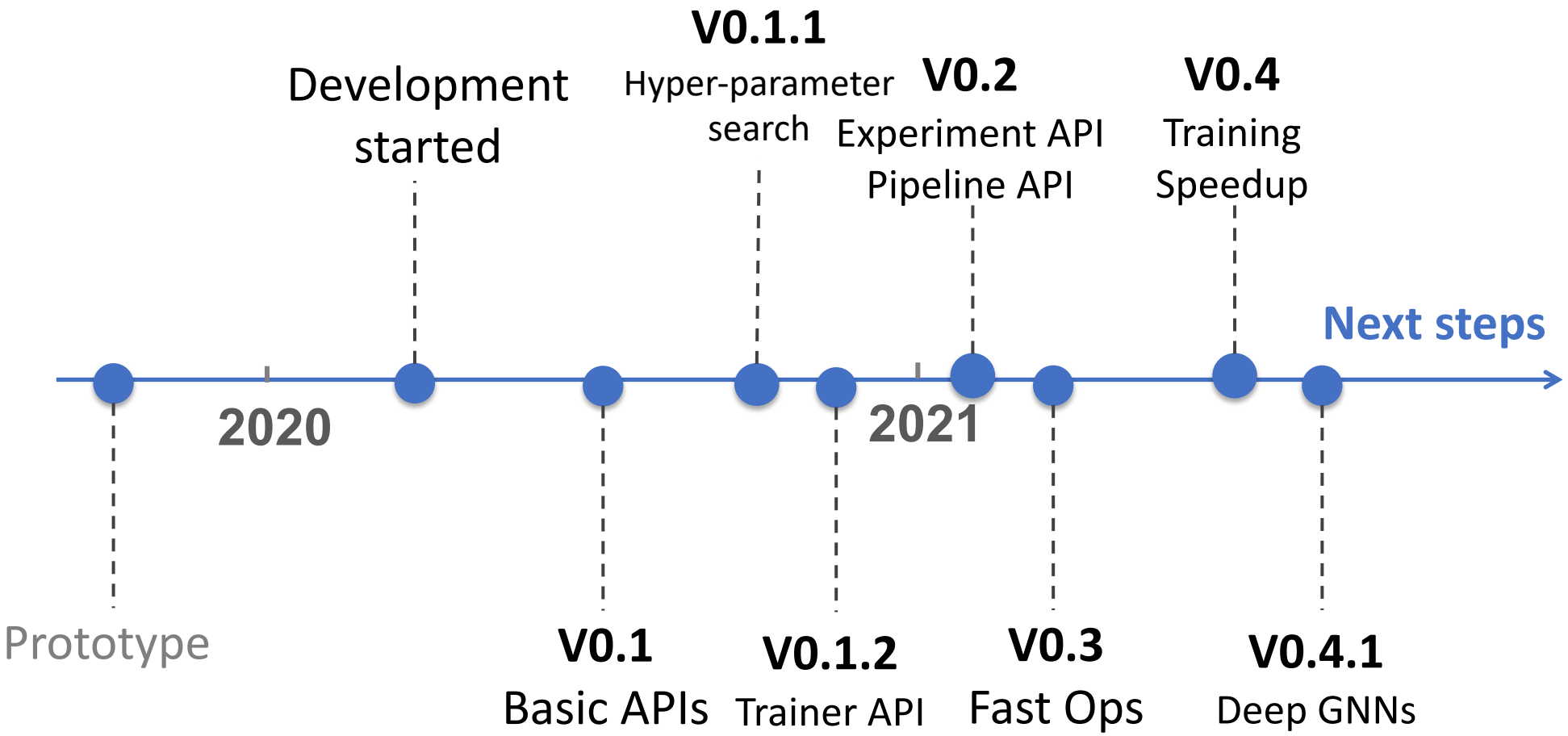
Reproducibility



Efficiency

Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Yang Yang, Peng Zhang, Guohao Dai, Yu Wang, Chang Zhou, Hongxia Yang, and Jie Tang. CogDL: An Extensive Toolkit for Deep Learning on Graphs. arXiv preprint 2021.

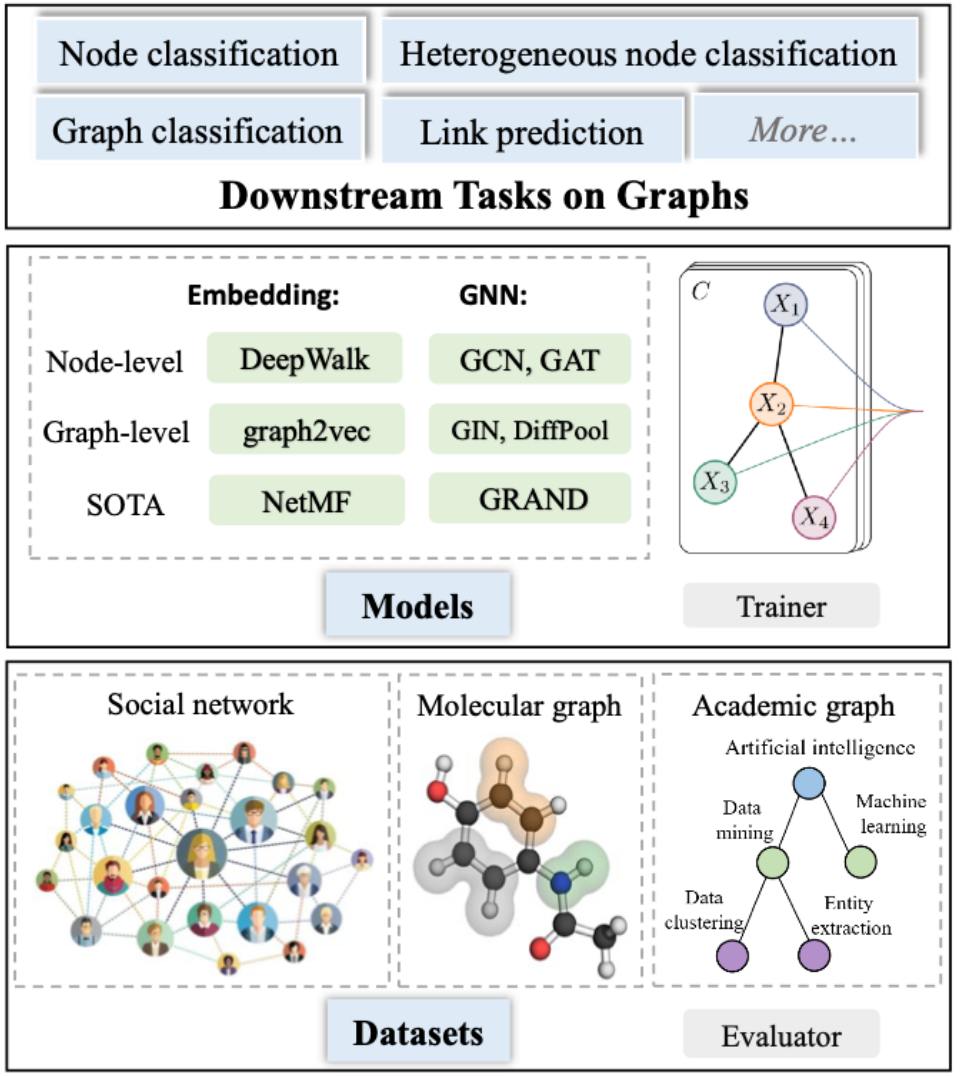
CogDL Development



Prerequisite: PyTorch environment
CogDL installation: **pip install cogdl**
or git clone https://github.com/THUDM/cogdl

Tasks, Datasets, Models in CogDL

- >10 Tasks:
  - node classification
  - graph classification
- >60 Datasets:
  - Social networks
  - Academic graphs
  - Molecular graphs
- >70 models:
  - Network embedding
  - Graph Neural Networks



Experiment API

- Feed task, dataset, model, (hyper-parameters), (search space)

```python
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")

# set other hyper-parameters
experiment(task="node_classification", dataset="cora", model="gcn", hidden_size=32, max_epoch=200)

# run over multiple models on different seeds
experiment(task="node_classification", dataset="cora", model=["gcn", "gat"], seed=[1, 2])

# automl usage
def func_search(trial):
    return {
        "lr": trial.suggest_categorical("lr", [1e-3, 5e-3, 1e-2]),
        "hidden_size": trial.suggest_categorical("hidden_size", [32, 64, 128]),
        "dropout": trial.suggest_uniform("dropout", 0.5, 0.8),
    }

experiment(task="node_classification", dataset="cora", model="gcn", seed=[1, 2], func_search=func_search)
```

Results of Experiment API

In [3]:
```
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")
```

```
Failed to load C version of sampling, use python version instead.
Namespace(activation='relu', checkpoint=None, cpu=False, dataset='cora', device_id=[0], dropout=0.5, fast_spmm=False, hidden_s
ize=64, inference=False, lr=0.01, max_epoch=500, missing_rate=0, model='gcn', norm=None, num_classes=None, num_features=None,
num_layers=2, patience=100, residual=False, save_dir='.', save_model=None, seed=1, task='node_classification', trainer=None, u
se_best_config=False, weight_decay=0.0005)
Downloading https://cloud.tsinghua.edu.cn/d/6808093f7f8042bfa1f0/files/?p=%2Fcora.zip&dl=1
unpacking cora.zip
Processing...
Done!
Epoch: 455, Train: 1.0000, Val: 0.7920, ValLoss: 0.7300:  89%|████████▉ | 446/500 [00:03<00:00, 120.41it/s]
Valid accurracy =  0.7940
Test accuracy = 0.8100
| Variant          | Acc            | ValAcc         |
|------------------|----------------|----------------|
| ('cora', 'gcn') | 0.8100±0.0000 | 0.7940±0.0000 |
```

Out[3]:
```
defaultdict(list, {('cora', 'gcn'): [{'Acc': 0.81, 'ValAcc': 0.794}]})
```

Results of Node Classification

- Two kinds of models：
  - Semi-supervised：GCN, GAT, GRAND, …
  - Self-supervised：MVGRL, DGI
- Citation networks：Cora, Citeseer, Pubmed

Rank	Method	Cora	Citeseer	Pubmed	*Reproducible*
1	GRAND [12]	84.8	75.1	82.4	*Yes*
2	GCNII [7]	85.1	71.3	80.2	*Yes*
3	MVGRL [20]	83.6 ↓	73.0	80.1	*Partial*
4	APPNP [26]	**84.3** ↑	72.0	80.0	*Yes*
5	Graph-Unet [15]	83.3 ↓	71.2 ↓	79.0	*Partial*
6	GDC [27]	82.5	72.1	79.8	*Yes*
7	GAT [53]	82.9	71.0	78.9	*Yes*
8	DropEdge [38]	82.1	72.1	79.7	*Yes*
9	GCN [25]	**82.3** ↑	**71.4** ↑	79.5	*Yes*
10	DGI [52]	82.0	71.2	76.5	*Yes*
11	JK-net [58]	81.8	69.5	77.7	*Yes*
12	Chebyshev [8]	79.0	69.8	68.6	*Yes*

Results of Graph Classification

- Two kinds of models
  - Self-supervised：InfoGraph, graph2vec, DGK
  - Supervised：GIN, DiffPool, SortPool, …
- Two types of graphs
  - Bioinformatics：MUTAG, PTC, NCI1, PROTEINS
  - Social networks：IMDB-B/M, COLLAB, REDDIT-B

Algorithm	MUTAG	PTC	NCI1	PROTEINS	IMDB-B	IMDB-M	COLLAB	REDDIT-B	*Reproducible*
GIN [57]	92.06	67.82	81.66	75.19	76.10	51.80	79.52	83.10 ↓	*Yes*
InfoGraph [42]	88.95	60.74	76.64	73.93	74.50	51.33	79.40	76.55	*Yes*
DiffPool [62]	85.18	58.00	69.09	75.30	72.50	50.50	79.27	81.20	*Yes*
SortPool [67]	87.25	62.04	**73.99** ↑	74.48	75.40	50.47	**80.07** ↑	78.15	*Yes*
graph2vec [31]	83.68	54.76 ↓	71.85	73.30	73.90	52.27	**85.58** ↑	91.77	*Yes*
PATCHY_SAN [32]	86.12	61.60	69.82	75.38	**76.00** ↑	46.40	74.34	60.61	*Yes*
DGCNN [56]	83.33	56.72	65.96	66.75	71.60	49.20	77.45	86.20	*Yes*
SAGPool [28]	71.73 ↓	59.92	72.87	74.03	74.80	51.33	/	89.21	*Yes*
DGK [59]	85.58	57.28	/	72.59	55.00 ↓	40.40 ↓	/	/	*Partial*

Pipeline API

- Feed application, model, (hyper-parameters)

```python
import numpy as np
from cogdl import pipeline

# build a pipeline for generating embeddings
# pass model name with its hyper-parameters to this API
generator = pipeline("generate-emb", model="prone")

# generate embedding by an unweighted graph
edge_index = np.array([[0, 1], [0, 2], [0, 3], [1, 2], [2, 3]])
outputs = generator(edge_index)
print(outputs)

# build a pipeline for generating embeddings using unsupervised GNNs
# pass model name and num_features with its hyper-parameters to this API
generator = pipeline("generate-emb", model="dgi", num_features=8, hidden_size=4)
outputs = generator(edge_index, x=np.random.randn(4, 8))
print(outputs)
```

```
- prone
- netmf
- netsmf
- deepwalk
- line
- node2vec
- hope
- sdne
- grarep
- dngr
- spectral
```

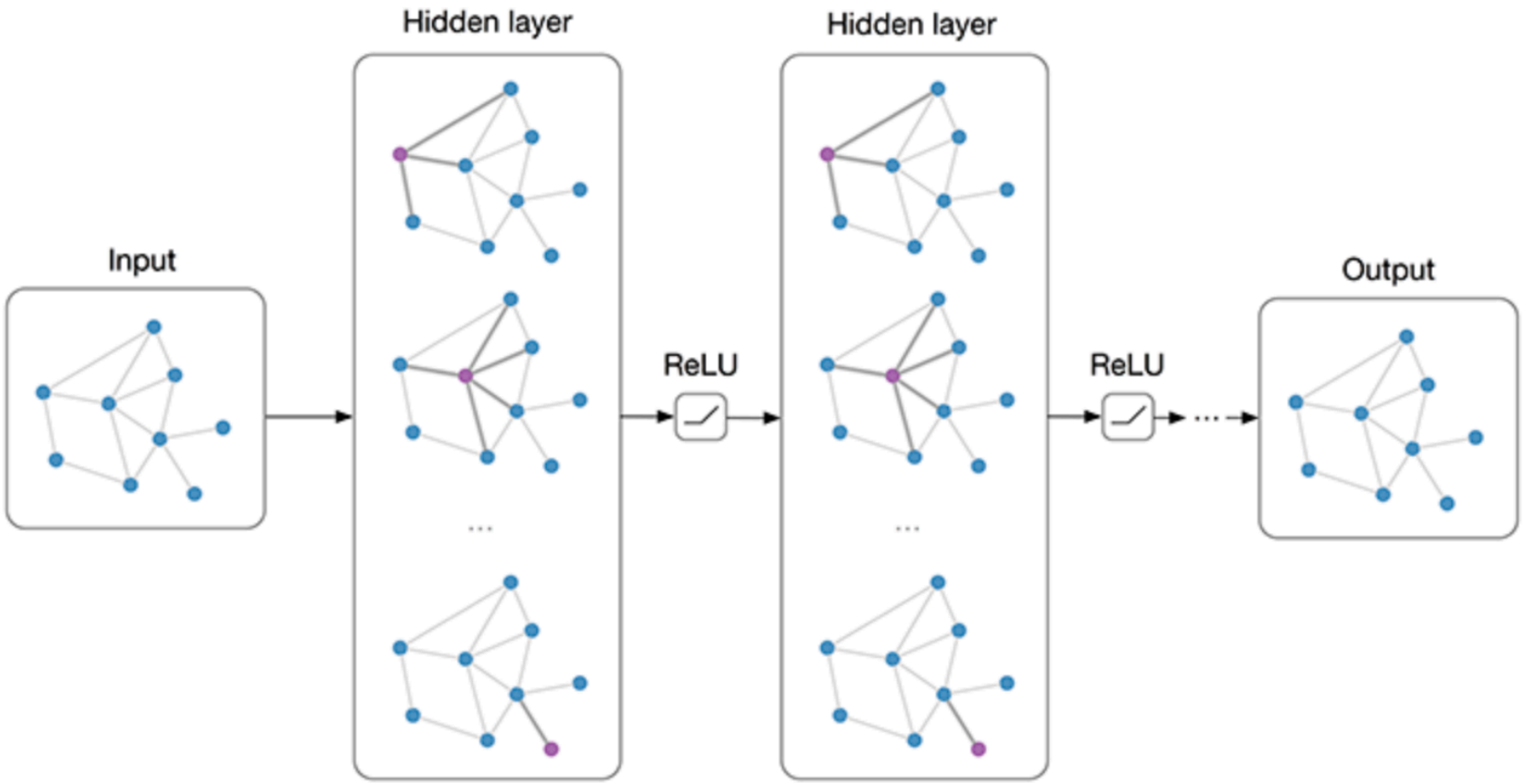
```
- unsup_graphsage
- dgi
- mvgrl
- grace
```

Outline

- Preliminary
- **Basic GNNs**
- Advanced GNNs
- All with CogDL

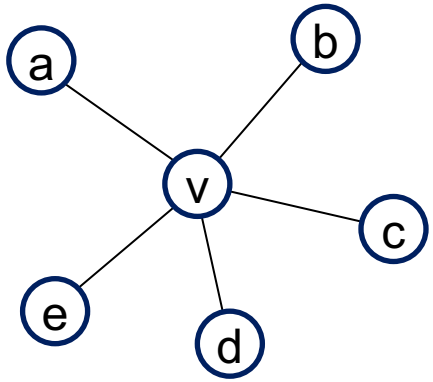
Graph Neural Networks

- Layer-wise propagation: $f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$



Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In ICLR '17.

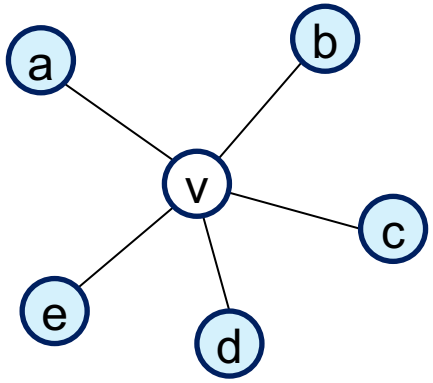
Graph Neural Networks



$$\boldsymbol{h}_v = f(\boldsymbol{h}_a, \boldsymbol{h}_b, \boldsymbol{h}_c, \boldsymbol{h}_d, \boldsymbol{h}_e)$$

- **Neighborhood Aggregation:**
  - Aggregate neighbor information and pass into a neural network
  - It can be viewed as a center-surround filter in CNN---graph convolutions!

GCN: Graph Convolutional Networks



parameters in layer k

Non-linear activation function (e.g., ReLU)

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

node v's embedding at layer k

the neighbors of node v

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

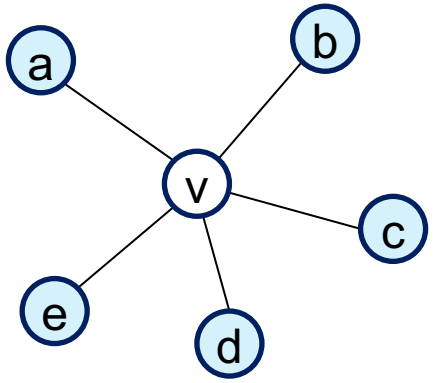
GCN Performance

- 2-layer GCN: $\boldsymbol{Z} = \mathrm{softmax}\big(\tilde{\boldsymbol{A}}\,\sigma\big(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_0\big)\boldsymbol{W}_1\big)$

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

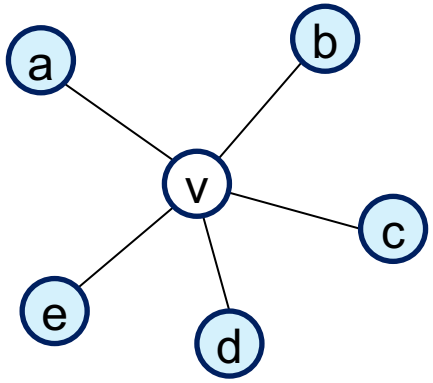
GraphSAGE

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}_k \cdot \mathrm{AGG}(\{h_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}_k h_v^{k-1}])$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

GraphSAGE

Instead of summation, it concatenates neighbor & self embeddings

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}_k \cdot \mathrm{AGG}(\{h_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}_k h_v^{k-1}])$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

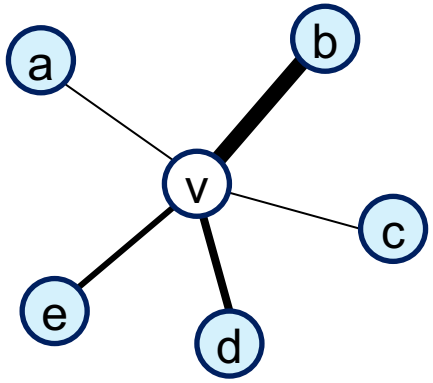
1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSAGE Performance

- AGGs: GCN / mean / LSTM / max-pooling
- Supervised (Sup.), Unsupervised (Unsup.)

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

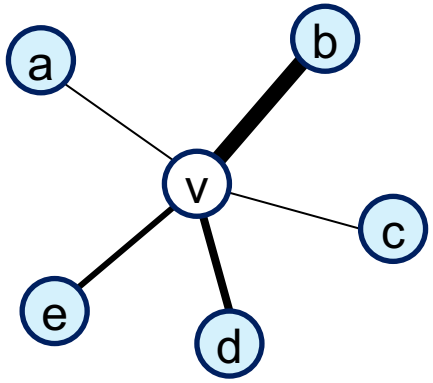
Graph Attention Networks (GAT)



Realistically, neighbors play different influences

1. Velickovic et al. Graph Attention Networks. ICLR 2018

Graph Attention Networks (GAT)



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

Graph Attention

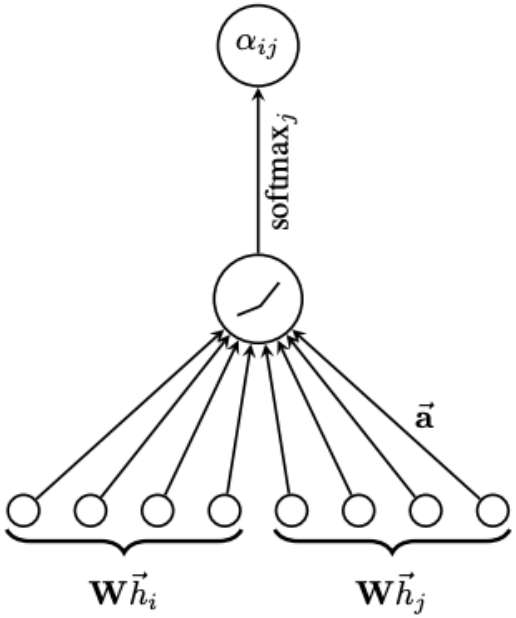
$$\boldsymbol{h}_v^k = \sigma(\sum_{u \in \mathrm{N}(v) \cup v} \alpha_{v,u} \boldsymbol{W}^k \boldsymbol{h}_u^{k-1})$$

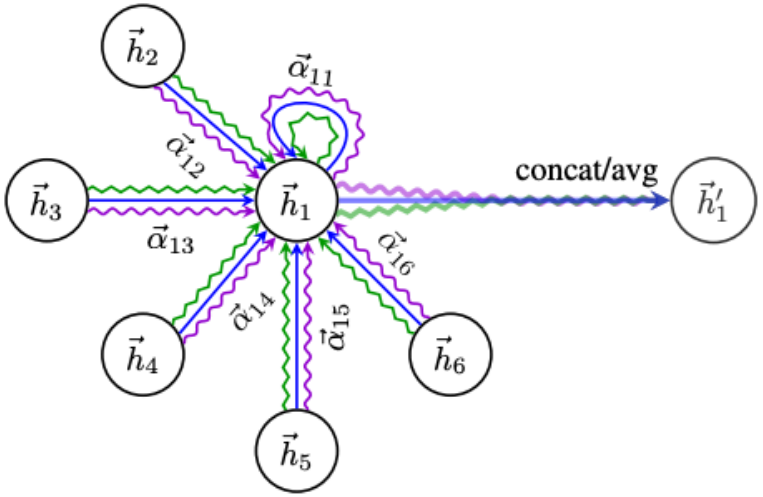
Learned attention weights

1. Velickovic et al. Graph Attention Networks. ICLR 2018

Graph Attention Networks (GAT)

- How to compute attention coefficients?
- $e_{vu} = \text{LeakyReLU}\left(\boldsymbol{r}^{\boldsymbol{T}}[\boldsymbol{W}\boldsymbol{h}_v || \boldsymbol{W}\boldsymbol{h}_u]\right)$
- $\alpha_{vu} = \text{softmax}(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$





GAT Performance

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Inductive

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

Outline

- Preliminary
- Basic GNNs
- **Advanced GNNs**
- All with CogDL

Advanced GNNs

- JKNet (ICML'18)
- APPNP (ICLR'19)
- DropEdge (ICLR'20)
- GRAND (NeurIPS'20)
- GCNII (ICML'20)
- DeeperGCN (Arxiv 2020)
- RevGNN (ICML'21)

JKNet (ICML'18)

- Connections between influence distributions and random walk distribution:

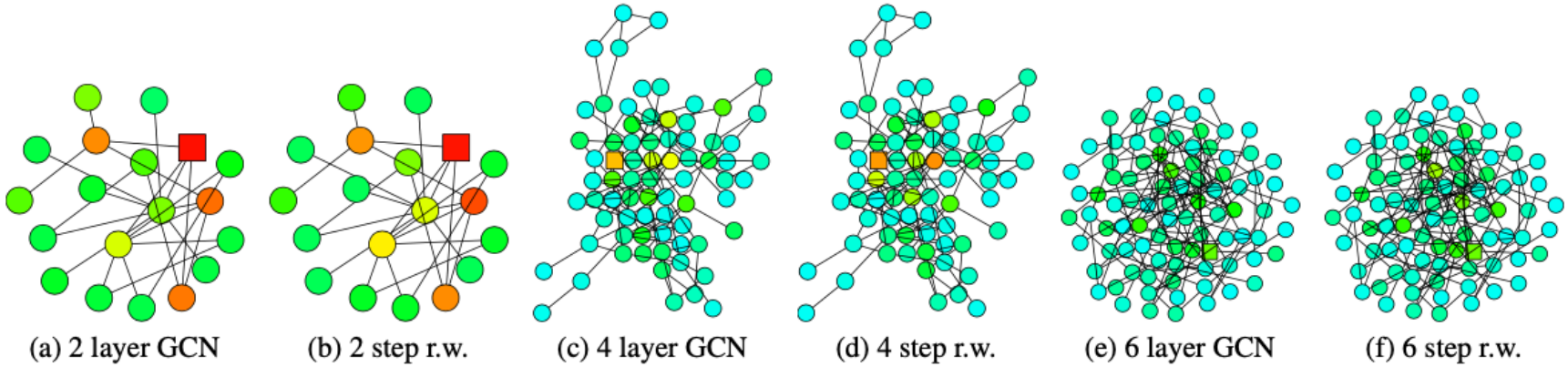


Figure 2. Influence distributions of GCNs and random walk distributions starting at the square node

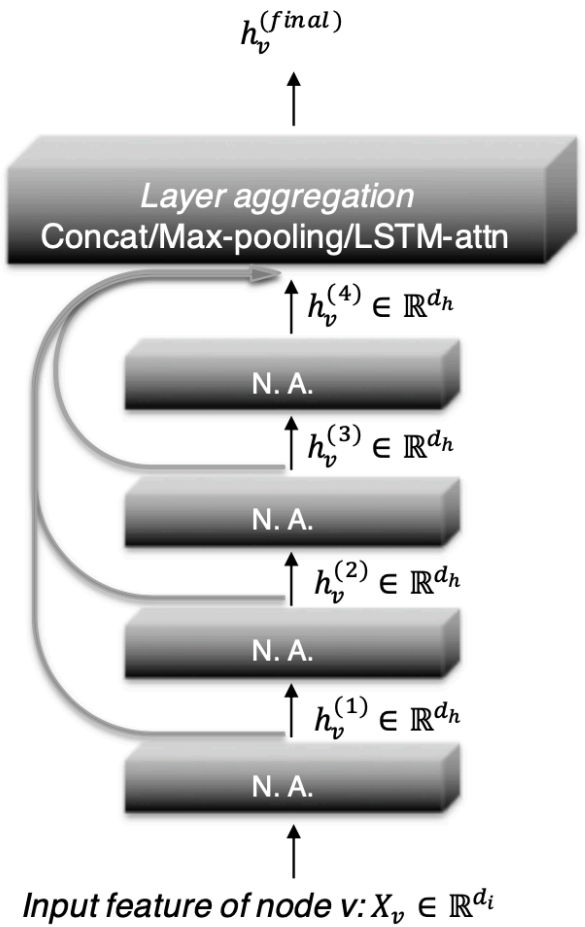
- Hard to determine propagation step!
- Layer aggregation!

Xu, Keyulu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. "Representation learning on graphs with jumping knowledge networks." In ICML'18.

JKNet (ICML'18)

- Layer-aggregation Mechanism
  - Concatenation
  - Max-pooling
  - LSTM-attention

Model	Citeseer	Model	Cora
GCN (2)	77.3 (1.3)	GCN (2)	88.2 (0.7)
GAT (2)	76.2 (0.8)	GAT (3)	87.7 (0.3)
JK-MaxPool (1)	77.7 (0.5)	JK-Maxpool (6)	**89.6** (0.5)
JK-Concat (1)	**78.3** (0.8)	JK-Concat (6)	89.1 (1.1)
JK-LSTM (2)	74.7 (0.9)	JK-LSTM (1)	85.8 (1.0)



APPNP (ICLR'19)

- Personalized PageRank (PPR):

$$\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x) = (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x)+\alpha\boldsymbol{i}_x$$

- By solving the equation, we obtain:

$$\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x) = \alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{i}_x$$

- Personalized propagation of neural predictions (PPNP):
  - generate **predictions** based on its own features and then **propagate** them via PPR:

$$\boldsymbol{Z}_{\text{PPNP}} = \text{softmax}\left(\alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{H}\right), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{X}_{i,:}),$$

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. In ICLR'19.

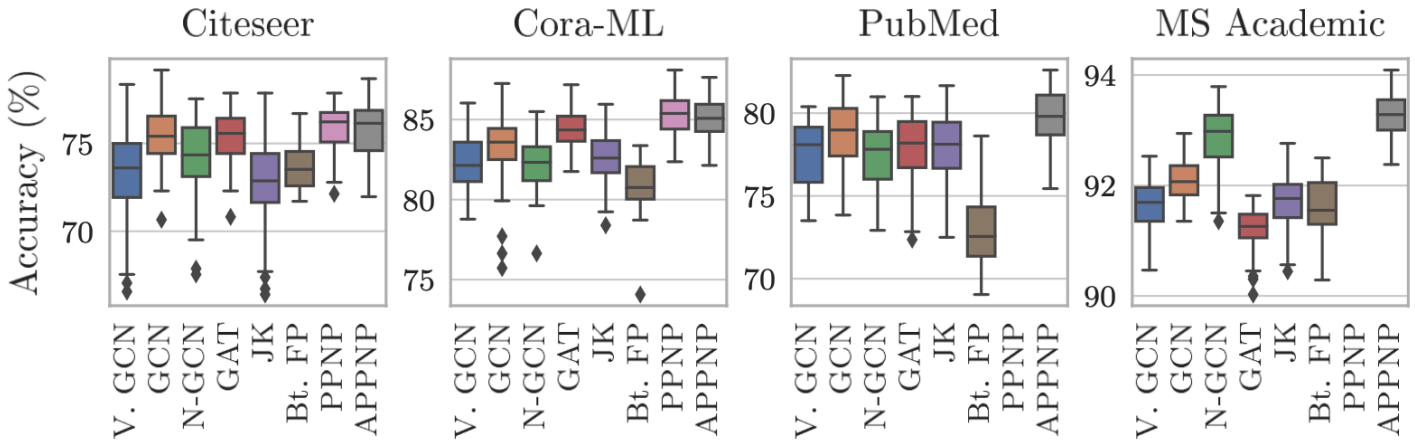
APPNP (ICLR'19)

- PPNP needs $O(n^2)$ to calculate the full PPR matrix:

$$\boldsymbol{\Pi}_{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}})^{-1}$$

- Approximate PPNP (APPNP):
  - via power iteration (random walk/propagation) step

$$\begin{aligned}
\boldsymbol{Z}^{(0)} &= \boldsymbol{H} = f_\theta(\boldsymbol{X}), \\
\boldsymbol{Z}^{(k+1)} &= (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{Z}^{(k)} + \alpha\boldsymbol{H}, \\
\boldsymbol{Z}^{(K)} &= \text{softmax}\left((1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{Z}^{(K-1)} + \alpha\boldsymbol{H}\right),
\end{aligned}$$



DropEdge (ICLR'20)

- Two issues: over-fitting and over-smoothing
- DropEdge: $\boldsymbol{A}_{\text{drop}} = \boldsymbol{A} - \boldsymbol{A}'$
- Prevent over-fitting:
  - unbiased data augmentation
- Alleviate over-smoothing:
  - Slow down the convergence of over-smoothing
  - Reduce information loss

Rong, Yu, Wenbing Huang, Tingyang Xu, and Junzhou Huang. "Dropedge: Towards deep graph convolutional networks on node classification." arXiv preprint arXiv:1907.10903 (2019).

DropEdge Discussion

- DropEdge vs Dropout
  - Dropout: no help to prevent over-smoothing
- DropEdge vs DropNode
  - GraphSAGE, FastGCN, ASGCN
- DropEdge vs Graph-Sparsification
  - Random vs Fixed

DropEdge Performance

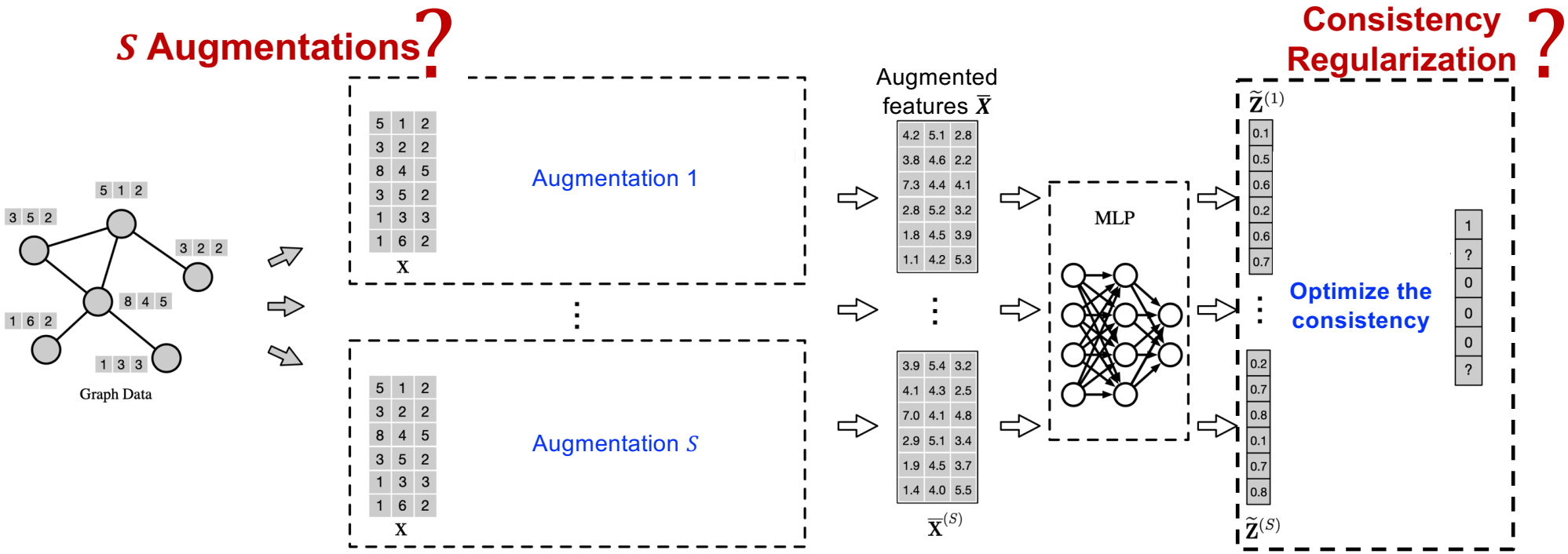
Table 1: Testing accuracy (%) comparisons on different backbones w and w/o DropEdge.

Dataset	Backbone	2 layers		8 layers		32 layers	
		Orignal	DropEdge	Orignal	DropEdge	Orignal	DropEdge
Cora	GCN	86.10	**86.50**	78.70	**85.80**	71.60	**74.60**
	ResGCN	-	-	85.40	**86.90**	85.10	**86.80**
	JKNet	-	-	86.70	**87.80**	87.10	**87.60**
	IncepGCN	-	-	86.70	**88.20**	87.40	**87.70**
	GraphSAGE	87.80	**88.10**	84.30	**87.10**	31.90	**32.20**
Citeseer	GCN	75.90	**78.70**	74.60	**77.20**	59.20	**61.40**
	ResGCN	-	-	77.80	**78.80**	74.40	**77.90**
	JKNet	-	-	79.20	**80.20**	71.70	**80.00**
	IncepGCN	-	-	79.60	**80.50**	72.60	**80.30**
	GraphSAGE	78.40	**80.00**	74.10	**77.10**	37.00	**53.60**
Pubmed	GCN	90.20	**91.20**	90.10	**90.90**	84.60	**86.20**
	ResGCN	-	-	89.60	**90.50**	90.20	**91.10**
	JKNet	-	-	90.60	**91.20**	89.20	**91.30**
	IncepGCN	-	-	90.20	**91.50**	OOM	**90.50**
	GraphSAGE	90.10	**90.70**	90.20	**91.70**	41.30	**47.90**
Reddit	GCN	96.11	**96.13**	96.17	**96.48**	45.55	**50.51**
	ResGCN	-	-	96.37	**96.46**	93.93	**94.27**
	JKNet	-	-	96.82	**97.02**	OOM	OOM
	IncepGCN	-	-	96.43	**96.87**	OOM	OOM
	GraphSAGE	96.22	**96.28**	96.38	**96.42**	96.43	**96.47**

GRAND (NeurIPS'20)

Graph Random Neural Network (GRAND)

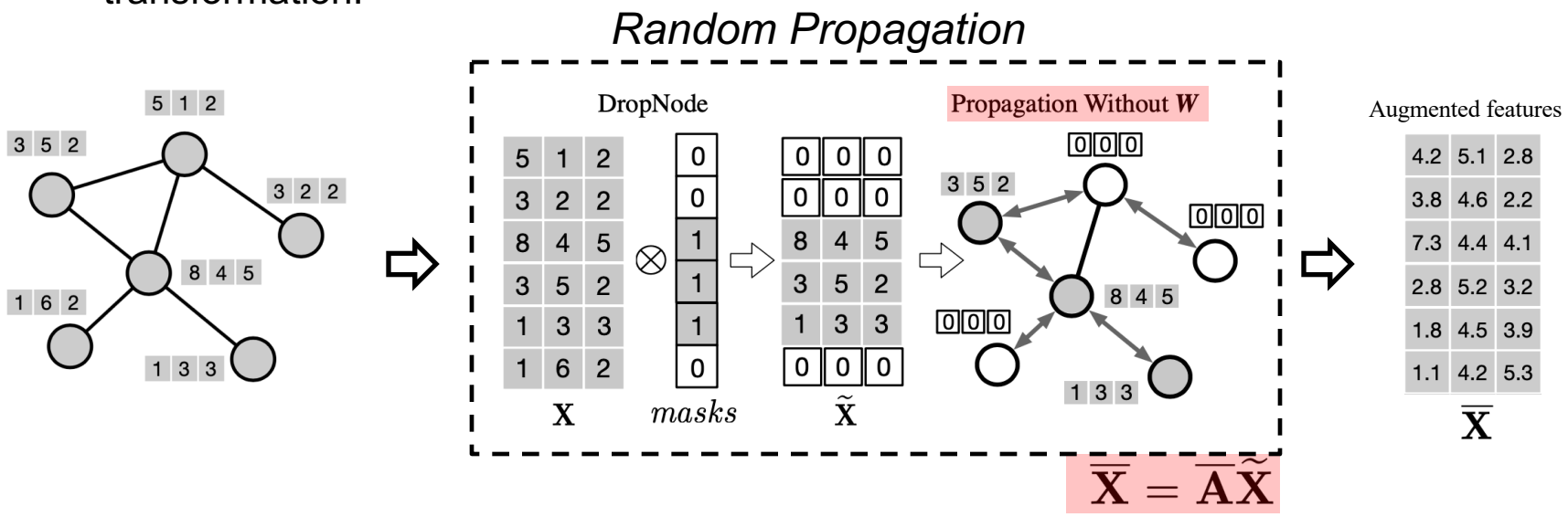
- Consistency Regularized Training:
  - Generates S data augmentations of the graph
  - Optimizing the consistency among S augmentations of the graph.



- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

GRAND (NeurIPS'20)

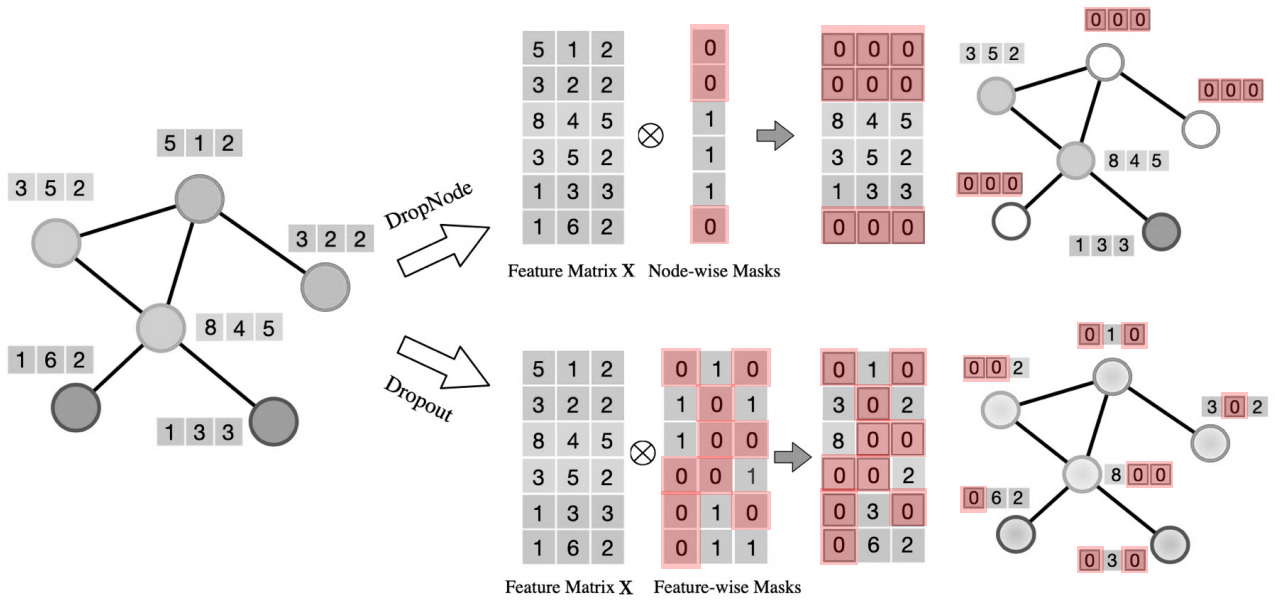
- **Random Propagation** (DropNode + Propagation):
  - Enhancing robustness: Each node is enabled to be not sensitive to specific neighborhoods.
  - Mitigating over-smoothing and overfitting: Decouple feature propagation from feature transformation.



- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

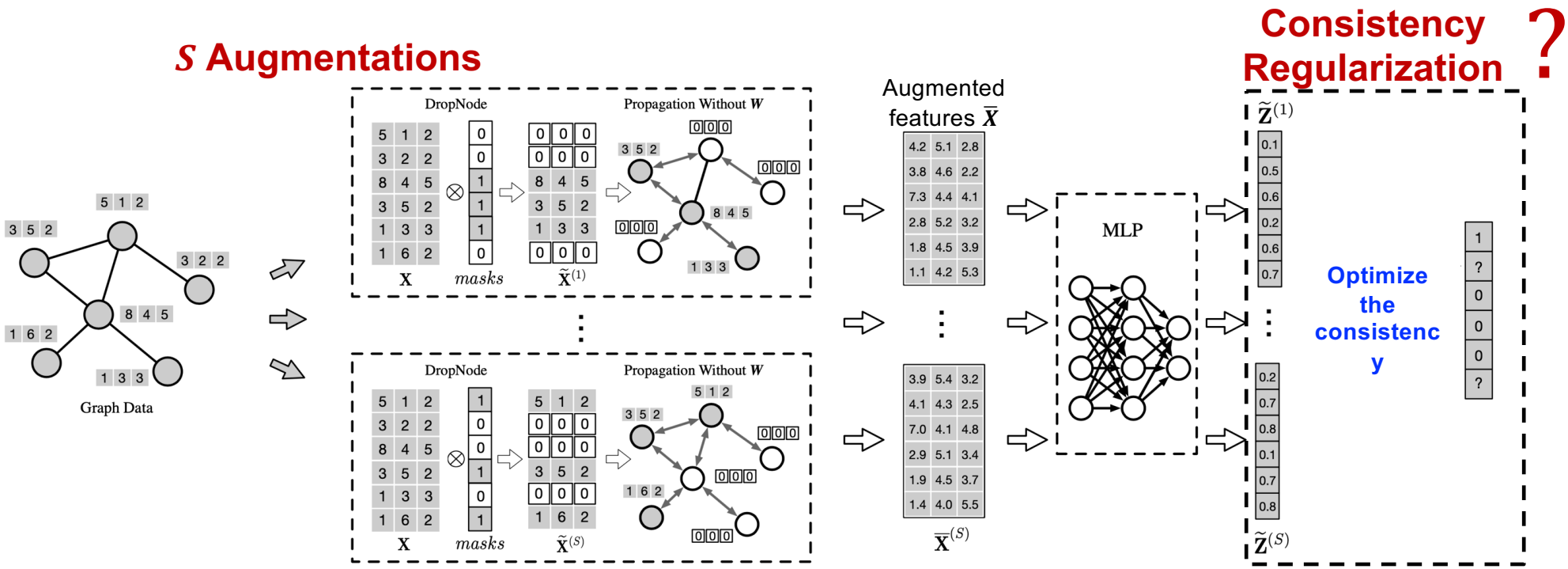
Random propagation: DropNode vs Dropout

- Dropout drops each element in $\boldsymbol{X}$ independently
- DropNode drops the entire features of selected nodes, i.e., the row vectors of $\boldsymbol{X}$, randomly



- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

GRAND (NeurIPS'20)



Random Propagation as data augmentation

- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

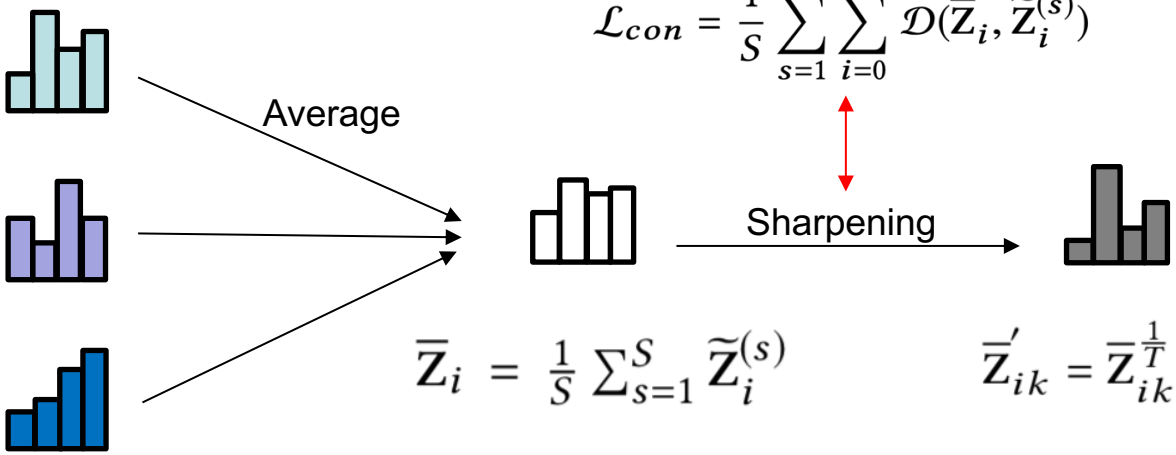
GRAND: Consistency Regularization

Distributions of a node after augmentations

$$\mathcal{L}_{sup} = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{m-1}\mathbf{Y}_i^{\top}\log\widetilde{\mathbf{Z}}_i^{(s)}$$

$$\mathcal{L}_{con} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{n-1}\mathcal{D}(\overline{\mathbf{Z}}_i', \widetilde{\mathbf{Z}}_i^{(s)})$$

$$\Rightarrow \quad \mathcal{L} = \mathcal{L}_{sup} + \lambda\mathcal{L}_{con}$$



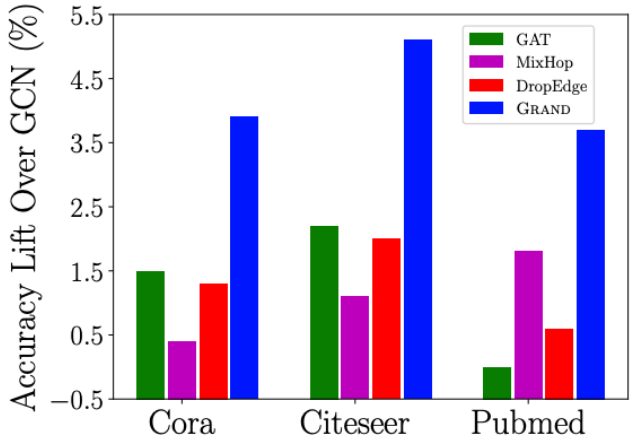
$$\overline{\mathbf{Z}}_i = \frac{1}{S}\sum_{s=1}^{S}\widetilde{\mathbf{Z}}_i^{(s)}$$

$$\overline{\mathbf{Z}}'_{ik} = \overline{\mathbf{Z}}_{ik}^{\frac{1}{T}} \Big/ \sum_{j=0}^{C-1}\overline{\mathbf{Z}}_{ij}^{\frac{1}{T}}$$

- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

GRAND Results

	Method	Cora	Citeseer	Pubmed
GCNs	GCN [19]	81.5	70.3	79.0
	GAT [32]	83.0±0.7	72.5±0.7	79.0±0.3
	APPNP [20]	83.8±0.3	71.6± 0.5	79.7 ± 0.3
	Graph U-Net [11]	84.4±0.6	73.2±0.5	79.6±0.2
	SGC [36]	81.0 ±0.0	71.9 ± 0.1	78.9 ± 0.0
	MixHop [1]	81.9± 0.4	71.4±0.8	80.8±0.6
	GMNN [28]	83.7	72.9	81.8
	GraphNAS [12]	84.2±1.0	73.1±0.9	79.6±0.4
Sampling GCNs	GraphSAGE [16]	78.9±0.8	67.4±0.7	77.8±0.6
	FastGCN [7]	81.4±0.5	68.8±0.9	77.6±0.5
Regularization GCNs	VBAT [10]	83.6±0.5	74.0±0.6	79.9±0.4
	G^3NN [24]	82.5±0.2	74.4±0.3	77.9 ±0.4
	GraphMix [33]	83.9±0.6	74.5±0.6	81.0±0.6
	DropEdge [29]	82.8	72.3	79.6
	GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**



Instead of the marginal improvements by conventional GNN baselines over GCN, ***GRAND*** *achieves* ***much more significant*** *performance lift in* ***all three datasets***!

- Feng et al. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. https://arxiv.org/abs/2005.11079, 2020
- Code & data for Grand: https://github.com/Grand20/grand

GCNII (ICML'20)

- Initial residual connection:
  - Similar approach to APPNP (but APPNP remains shallow)
  - Combine the smoothed representations with initial features

$$(1-\alpha)\tilde{\boldsymbol{P}}\boldsymbol{H}^{(\ell)}+\alpha\boldsymbol{H}^{(0)}$$

- Identity mapping:
  - Similar to the motivation of ResNet
  - Add an identity matrix to the weight matrix

$$(1-\beta_\ell)\mathbf{I}_n+\beta_\ell\mathbf{W}^{(\ell)}$$

Chen, Ming, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. "Simple and deep graph convolutional networks." In ICML'20.

GCNII Results

- GCNII (Combine the two techniques)

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha_\ell \mathbf{H}^{(0)}\right)\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}^{(\ell)}\right)\right)$$

- GCNII*: employ different weights for PH and $H^{(0)}$:

$$\mathbf{H}^{(\ell+1)} = \sigma\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}_1^{(\ell)}\right) + \right.$$
$$\left. + \alpha_\ell \mathbf{H}^{(0)}\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}_2^{(\ell)}\right)\right).$$

Method	Cora	Cite.	Pumb.	Cham.	Corn.	Texa.	Wisc.
GCN	85.77	73.68	88.13	28.18	52.70	52.16	45.88
GAT	86.37	74.32	87.62	42.93	54.32	58.38	49.41
Geom-GCN-I	85.19	**77.99**	90.05	60.31	56.76	57.58	58.24
Geom-GCN-P	84.93	75.14	88.09	60.90	60.81	67.57	64.12
Geom-GCN-S	85.27	74.71	84.75	59.96	55.68	59.73	56.67
APPNP	87.87	76.53	89.40	54.3	73.51	65.41	69.02
JKNet	85.25 (16)	75.85 (8)	88.94 (64)	60.07 (32)	57.30 (4)	56.49 (32)	48.82 (8)
JKNet(Drop)	87.46 (16)	75.96 (8)	89.45 (64)	62.08 (32)	61.08 (4)	57.30 (32)	50.59 (8)
Incep(Drop)	86.86 (8)	76.83 (8)	89.18 (4)	61.71 (8)	61.62 (16)	57.84 (8)	50.20 (8)
GCNII	**88.49** (64)	77.08 (64)	89.57 (64)	60.61 (8)	74.86 (16)	69.46 (32)	74.12 (16)
GCNII*	88.01 (64)	77.13 (64)	**90.30** (64)	**62.48** (8)	**76.49** (16)	**77.84** (32)	**81.57** (16)

DeeperGCN

- Generalized Aggregation Function (Mean-Max)
  - Find a better aggregator than *mean* and *max*
- SoftMax_Agg: $\sum_{u \in \mathcal{N}(v)} \frac{exp(\beta \mathbf{m}_{vu})}{\sum_{i \in \mathcal{N}(v)} exp(\beta \mathbf{m}_{vi})}$
  - $\lim_{\beta \to 0} \text{SoftMax\_Agg}_{\beta} = \text{Mean}$
  - $\lim_{\beta \to \infty} \text{SoftMax\_Agg}_{\beta} = \text{Max}$
- PowerMean: $\left(\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{m}_{vu}^{p}\right)^{1/p}$
  - $\text{PowerMean\_Agg}_{p=1} = \text{Mean}$
  - $\lim_{p \to \infty} \text{PowerMean\_Agg}_{p} = \text{Max}$

DeeperGCN

- Better residual connections:
  - pre-activation variant of residual connections
  - BN/LN → ReLU → GraphConv → Addition

	PlainGCN			ResGCN			ResGCN+		
#layers	Sum	Mean	Max	Sum	Mean	Max	Sum	Mean	Max
3	0.824	0.793	**0.834**	0.824	0.786	0.824	0.830	0.792	0.829
7	0.811	0.796	0.823	0.831	0.803	0.843	0.841	0.813	0.845
14	0.821	0.802	0.824	0.843	0.808	0.850	0.840	0.813	0.848
28	0.819	0.794	0.825	0.837	0.807	0.847	0.845	0.819	0.855
56	0.824	0.808	0.825	0.841	0.813	**0.851**	0.843	0.810	0.853
112	0.823	0.810	0.824	0.840	0.805	**0.851**	0.853	0.820	**0.858**
avg.	0.820	0.801	0.826	0.836	0.804	0.844	0.842	0.811	**0.848**

	SoftMax_Agg			PowerMean_Agg		
#layers	Fixed	β	β&s	Fixed	p	p&s
3	0.821	0.832	0.837	0.802	0.818	0.838
7	0.835	0.846	0.848	0.797	0.841	0.851
14	0.833	0.849	0.851	0.814	0.840	0.849
28	0.845	0.852	0.853	0.816	0.847	**0.854**
56	0.849	**0.860**	0.854	0.818	0.846	-
112	0.844	0.858	0.858	0.824	-	-
avg.	0.838	**0.850**	**0.850**	0.812	0.838	**0.848**

Deep GNNs (From 112 to 1000 layers)

- DeeperGCN: All You Need to Train Deeper GCNs
  - Li et al., June 2020
  - Generalized Message Aggregation
  - Pre-activation residual connections
  - Up to 112 layers (GPU memory bounded)

- Training Graph Neural Networks with 1000 Layers
  - Li et al., ICML 2021
  - Reversible connections
  - Up to 1000 layers

Li, Guohao, et al. "Deepergcn: All you need to train deeper gcns." *arXiv preprint arXiv:2006.07739* (2020).
Li, Guohao, et al. "Training Graph Neural Networks with 1000 Layers." arXiv preprint arXiv:2106.07476 (2021).

Recall Backpropagation in GNNs

- Graph convolution：

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{A}\boldsymbol{H}^{(l)}\boldsymbol{W}_l$$

- Backward of graph convolution：

$$\nabla_{\boldsymbol{H}^{(l)}} = \boldsymbol{A}^T \nabla_{\boldsymbol{H}^{(l+1)}} \boldsymbol{W}_l^T$$

$$\nabla_{\boldsymbol{W}_l} = \boldsymbol{H}^{(l)^T} \boldsymbol{A}^T \nabla_{\boldsymbol{H}^{(l+1)}}$$

$$\nabla_{\boldsymbol{A}} = \nabla_{\boldsymbol{H}^{(l+1)}} \boldsymbol{W}_l^T \boldsymbol{H}^{(l)^T}$$

- We need to save $\boldsymbol{H}^{(l)}$ for each layer, which costs O(ND) memory per layer。

GNNs with 1000 Layers (ICML'21)

- Challenges: O(LND) memory, linear to the number layers
- **Reversible connections!**
- (similar to NeurIPS 2017: The reversible residual network: Backpropagation without storing activations)
- Grouped Reversible GNN block:

$$\langle X_1, X_2, ..., X_C \rangle \mapsto \langle X_1', X_2', ..., X_C' \rangle$$

Forward (from X_i to X_i')

$$X_0' = \sum_{i=2}^{C} X_i$$

$$X_i' = f_{w_i}(X_{i-1}', A, U) + X_i,\ i \in \{1, \cdots, C\}$$

Backward (from X_i' to X_i)

$$X_i = X_i' - f_{w_i}(X_{i-1}', A, U),\ i \in \{2, \cdots, C\}$$

$$X_0' = \sum_{i=2}^{C} X_i$$

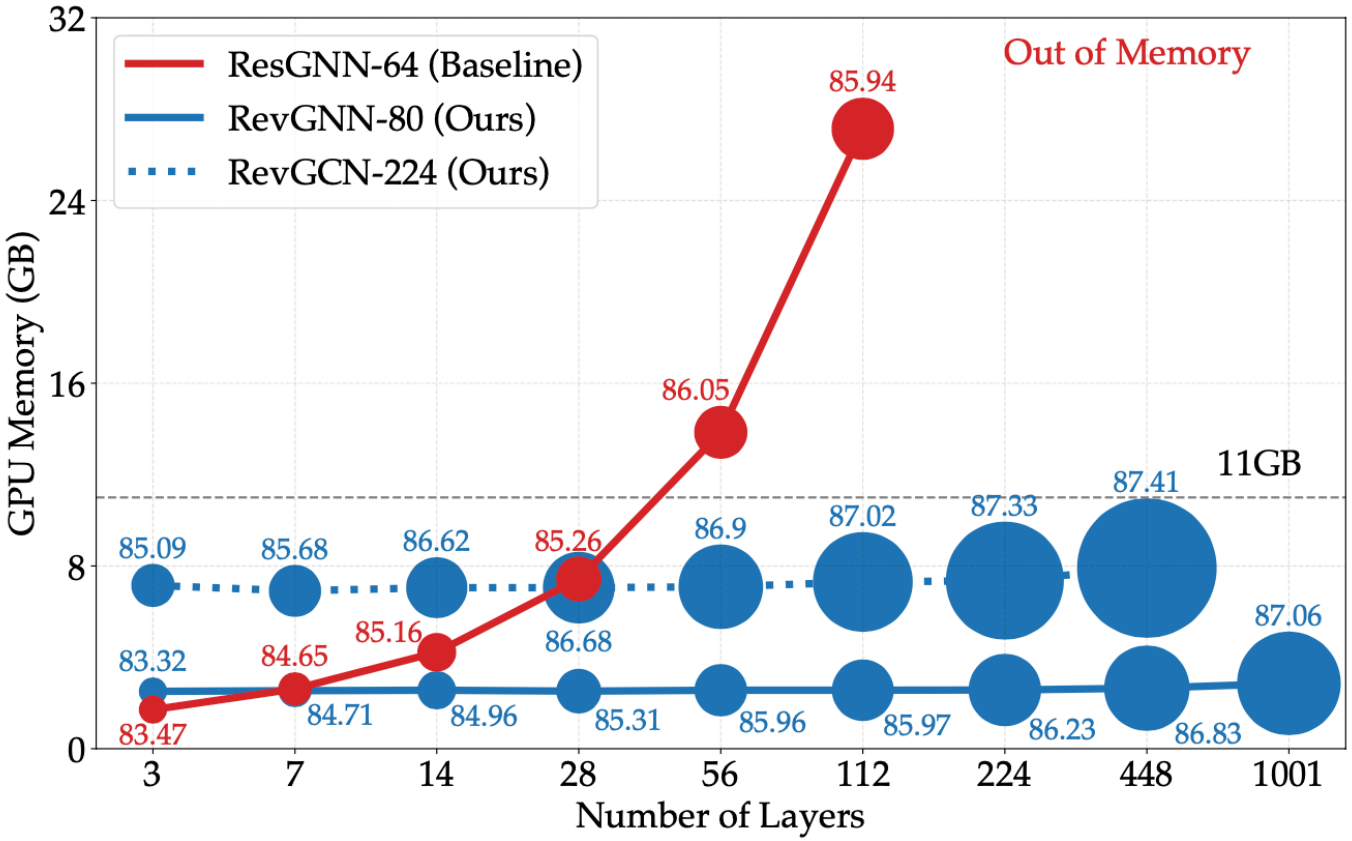
$$X_1 = X_1' - f_{w_1}(X_0', A, U).$$

RevGNN on ogbn-proteins

- ogbn-proteins dataset：
  - Node: proteins
  - Edge: biologically meaningful associations (e.g., homology)

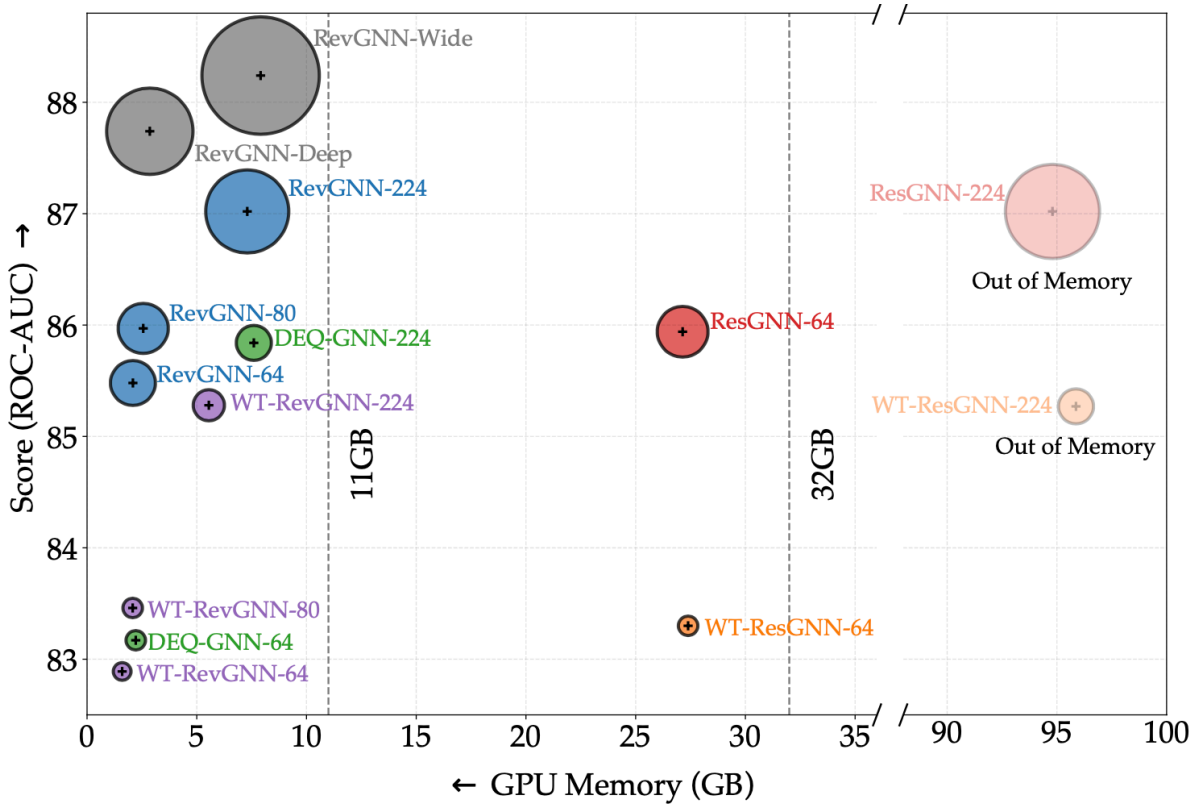
Model	ROC-AUC ↑	Mem ↓	Params
GCN (Kipf & Welling)	72.51 ± 0.35	4.68	96.9k
GraphSAGE (Hamilton et al.)	77.68 ± 0.20	3.12	193k
DeeperGCN (Li et al.)	86.16 ± 0.16	27.1	2.37M
UniMP (Shi et al.)	86.42 ± 0.08	27.2	1.91M
GAT (Veličković et al.)	86.82 ± 0.21	6.74	2.48M
UniMP+CEF (Shi et al.)	86.91 ± 0.18	27.2	1.96M
Ours (RevGNN-Deep)	87.74 ± 0.13	**2.86**	20.03M
Ours (RevGNN-Wide)	**88.24** ± 0.15	7.91	68.47M

RevGNN v.s. ResGNN



RevGNN v.s. all variants

- **RevGNN-Wide**
  - 448 layers+224 hidden
- **RevGNN-Deep**
  - 1001 layers+80 hidden
- Compared with RevGNN/ResGNN/WT/DEQ-x (x: hidden)
- Datapoint size is proportional to \sqrt(#parameters)



Outline

- Preliminary
- Basic GNNs
- Advanced GNNs
- **All with CogDL**

All with CogDL

- Efficiency
  - Graph storage in CogDL
  - Sparse operators in CogDL
  - Training on large-scale graphs
  - Training very deep GNNs
- Customization
  - Customized usage in CogDL
- Benchmarks:
  - Self-supervised learning
  - Heterogeneous Graph Benckmark (HGB)
  - Graph Robustness Benchmark (GRB)
- Applications:
  - Recommendation

Sparse Storage of Adjacency Matrix

- COO format：
  - (row, col) or (row, col, value)，size: |E|*2/3
  - [[0,0,1], [0,2,2], [1,2,3], [2,0,4], [2,1,5], [2,2,6]]
- CSR format：
  - row_ptr：size |V|+1
  - col_indices: size |E|
  - value: size |E|
  - [0, 2, 3, 6], [0, 2, 2, 0, 1, 2], [1, 2, 3, 4, 5, 6]

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Graph Storage in CogDL

class Graph: (defined in cogdl.data)

- x：node feature matrix
- y：node labels
- edge_index：COO format matrix
- edge_weight：edge weight (if exists)
- edge_attr：edge attributes (if exists)
- row_ptr：row index pointer for CSR matrix
- col_indices：column indices for CSR matrix

Usage of CogDL's Graph

- Graph Initialization
  - g = Graph(edge_index=edge_index)
  - g.edge_weight = torch.rand(n)
- Commonly used operators：
  - add_self_loops()
  - sym_norm()
  - degrees()
  - subgraph()
  - …

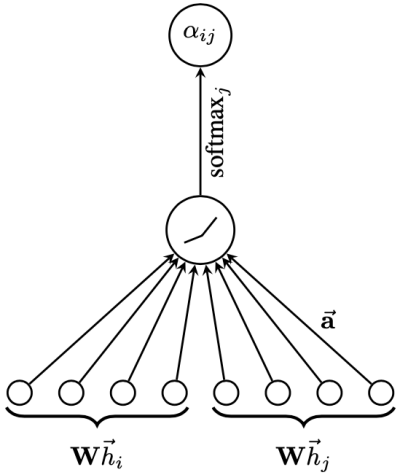
Recall Sparse Operators in GNNs

- GCN（Sparse Maxtrix-Matrix Multiplication, SpMM）

$$\boldsymbol{H}^{(i+1)} = \boldsymbol{A}\boldsymbol{H}^{(i)}\boldsymbol{W}$$

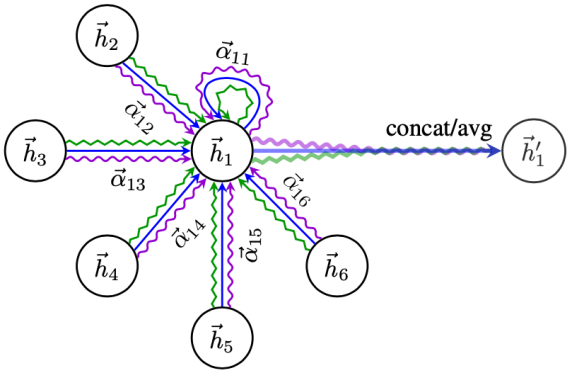
- GAT（ Edge-wise-softmax）

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$



- GAT（Multi-Head SpMM）

$$\boldsymbol{h}_i = CONCAT\left(\sigma\left(\sum_{j \in N_i} \alpha_{ij}^k \boldsymbol{W}^k \boldsymbol{h}_j\right)\right)$$



GCN/GAT Layer in CogDL

```python
# GCN Layer
# graph: cogdl.data.Graph
# x: node featurs
# weight: parameters

h = torch.mm(x, weight)
h = spmm(graph, h)
out = torch.relu(h)
```

```python
# GAT Layer
# graph: cogdl.data.Graph
# h: node featurs
# h_score: importance score of edge

edge_attention = mul_edge_softmax(graph, edge_score)
h = mh_spmm(graph, edge_attention, h)
out = torch.cat(h, dim=1)
```

$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i}\alpha_{ij}^{k}\boldsymbol{W}^{k}h_j\right)\right)$$

Implementation of GCN/GAT Layer

```python
class GCNLayer(nn.Module):
    """
    Simple GCN layer, similar to https://arxiv.o
    """

    def __init__(self, in_features, out_features,

    def reset_parameters(self):…

    def forward(self, graph, x):
        support = torch.mm(x, self.weight)
        out = spmm(graph, support)
```

```python
class GATLayer(nn.Module):
    """
    Sparse version GAT layer, similar to https://arxiv.org/ab
    """

    def __init__(self, in_features, out_features, nhead=1, al

    def reset_parameters(self):…

    def forward(self, graph, x):
        h = torch.matmul(x, self.W)
        edge_score = self.compute_edge_score(graph, x, h)
        # edge_attention: E * H
        edge_attention = mul_edge_softmax(graph, edge_score)
        edge_attention = self.dropout(edge_attention)

        out = mh_spmm(graph, edge_attention, h)
```

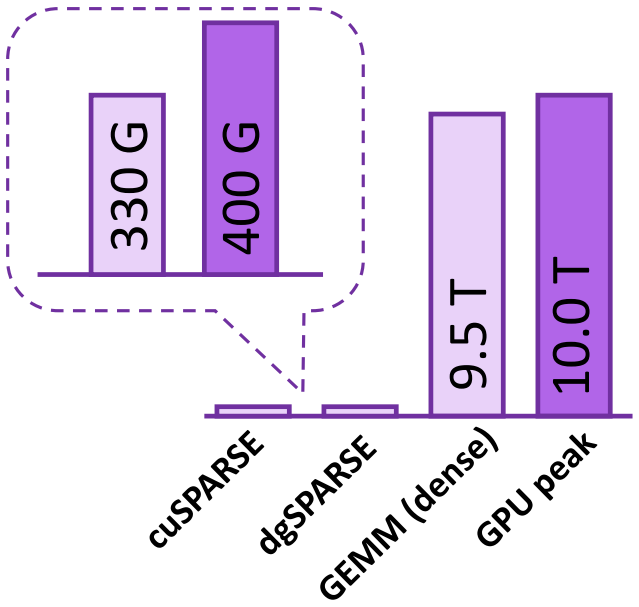
$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i}\alpha_{ij}^{k}\boldsymbol{W}^{k}h_j\right)\right)$$

Efficient Sparse Kernels

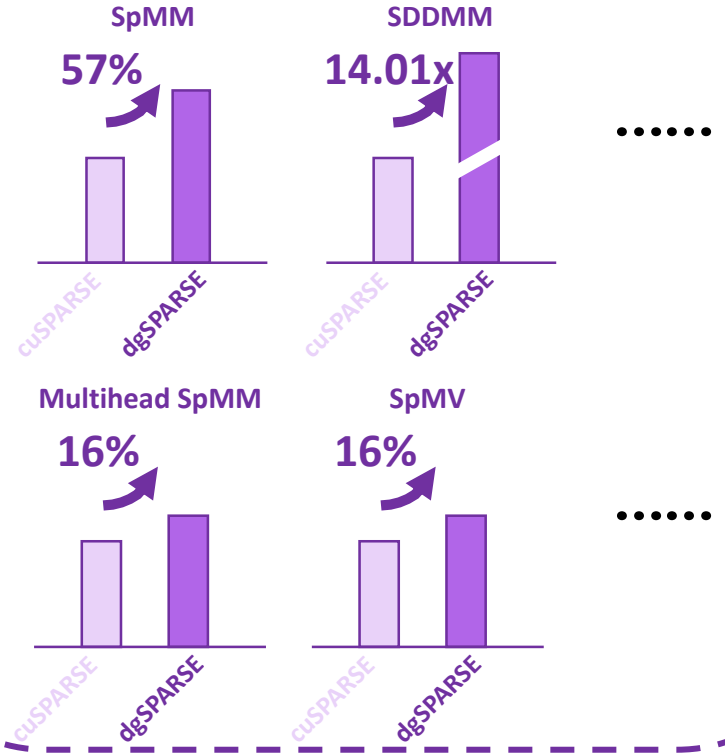
Lack efficient sparse kernels



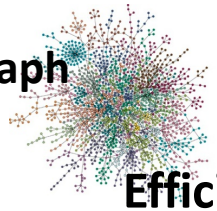
Big gap between FLOPS of sparse kernels and hardware peak performance.

dgSPARSE, Deep Graph SPARSE

Efficient Implementation on GPUs



Deep Graph Sparse (dgSPARSE) Library



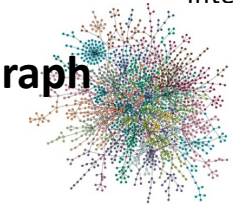
Efficient map

- Workload balance
- Thread coarsening
- ……

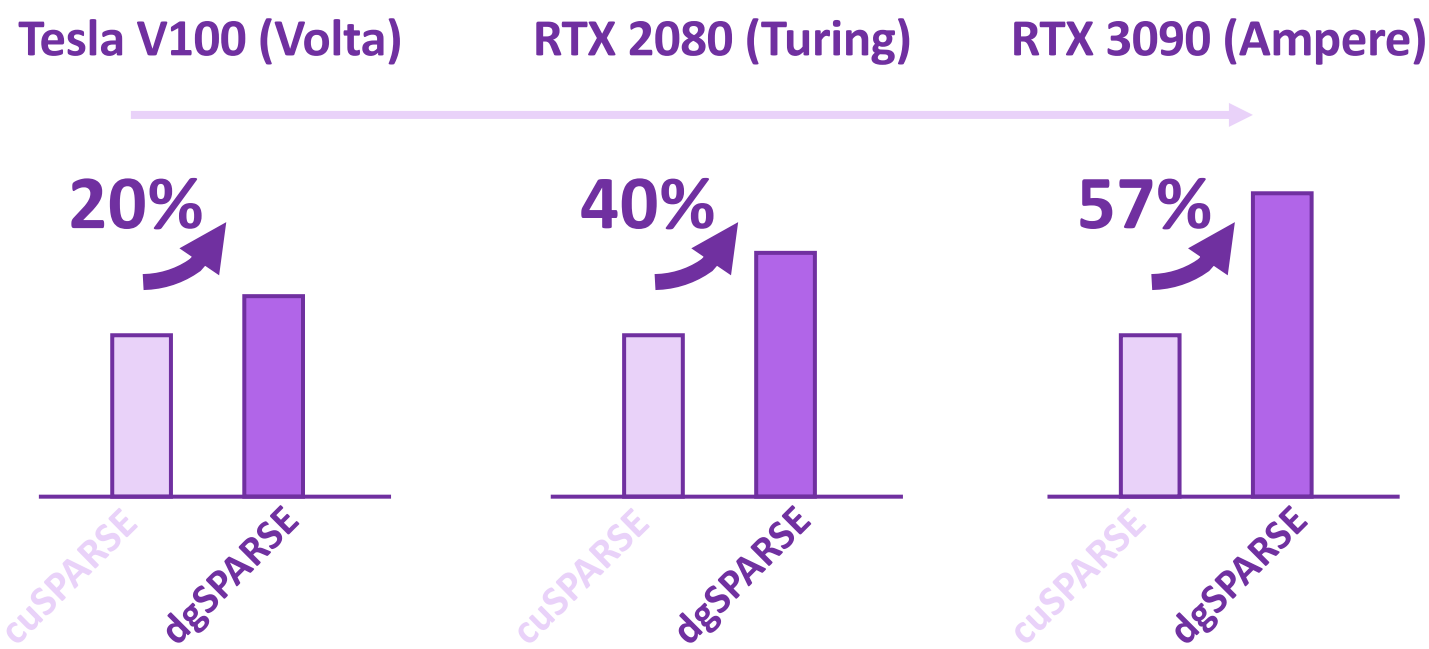
SIMT GPU architecture

Efficient reduce

- Inner-warp reduction
- Inter-warp reduction



Tiny Effort to new GPU architecture



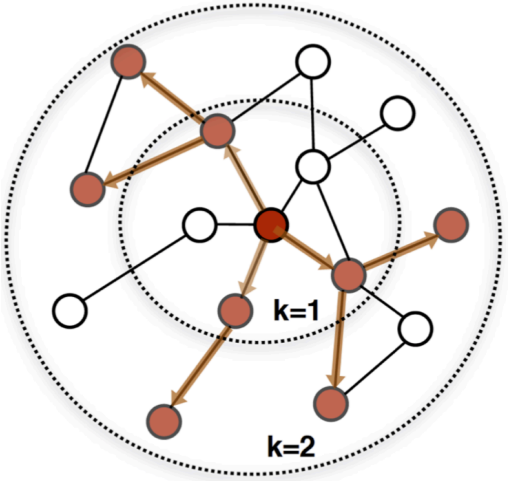
Performance of GCN/GAT model

- Setting：2-layer GCN/GAT，hidden size=128
- Supported by dgSPAESE

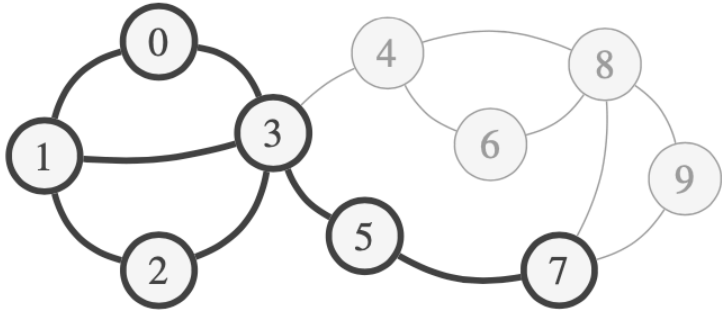
Model	GPU	Dataset	Tranining per epoch (s)				Inference per epoch (s)			
			torch	PyG	DGL	CogDL	torch	PyG	DGL	CogDL
GCN	2080Ti (11G)	Flickr	0.025	0.0084	0.012	0.085	0.012	0.0034	0.007	0.0035
		Reddit	0.445	0.122	0.102	0.081	0.218	0.045	0.049	0.039
		Yelp	0.412	0.151	0.151	0.110	0.191	0.053	0.063	0.040
	3090 (24G)	Flickr	0.017	0.006	0.008	0.007	0.008	0.002	0.004	0.002
		Reddit	0.263	0.062	0.060	0.050	0.127	0.022	0.0314	0.022
		Yelp	0.230	0.081	0.080	0.062	0.106	0.029	0.036	0.023
GAT	2080Ti (11G)	PubMed	0.017	0.016	0.011	0.012	0.004	0.006	0.004	0.003
		Flickr	0.082	0.090	0.047	0.056	0.023	0.030	0.019	0.014
		⋆Reddit	__‡	__‡	0.406	0.537	__‡	__‡	0.163	0.086
	3090 (24G)	PubMed	0.043	0.011	0.011	0.016	0.004	0.004	0.003	0.002
		Flickr	0.097	0.059	0.033	0.044	0.021	0.024	0.013	0.009
		Reddit	0.671	__‡	0.301	0.373	0.112	__‡	0.113	0.088
		Yelp	0.614	__‡	0.294	0.404	0.118	__‡	0.105	0.113

Training on Large-scale Graphs

- Billion-scale social networks and recommender systems
- Main challenge: GPU memory bounded!
- Training GNNs via mini-batch sampling

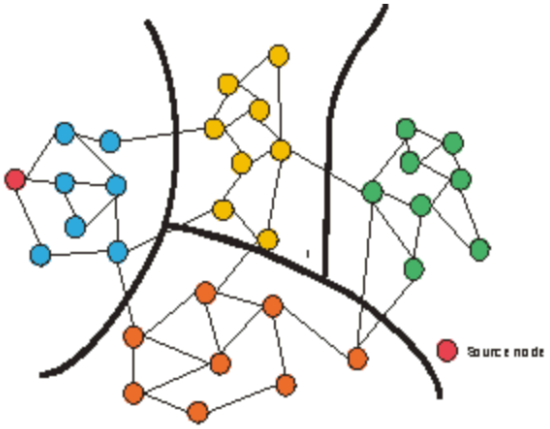


Neighbor Sampling
(NeurIPS '17)



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$

GraphSAINT
(ICLR '20)

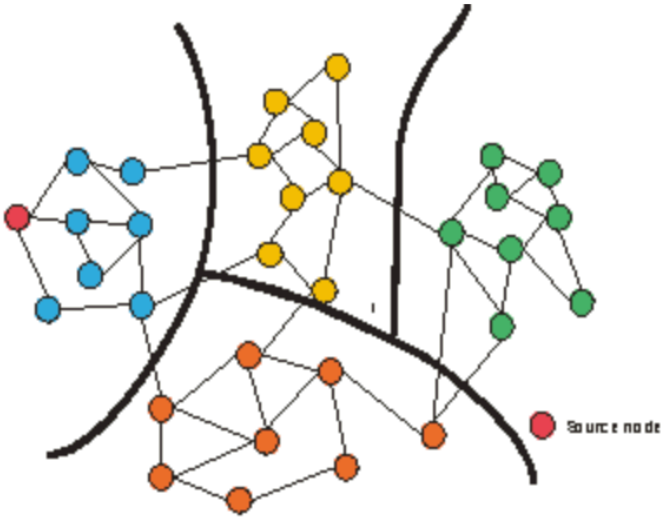


ClusterGCN
(KDD '19)

1. Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In NeurIPS '17.
2. Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In KDD '19.
3. Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In ICLR '20.

ClusterGCN

- Graph partition (METIS)
- Train GNNs via mini-batch



- Memory: O(NFL) -> O(bFL)

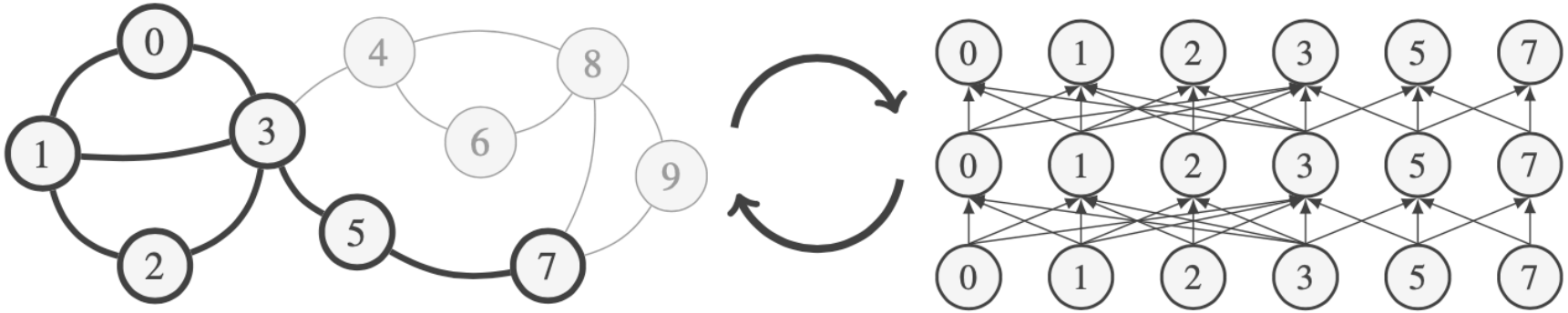
	GCN [9]	Vanilla SGD	GraphSAGE [5]	FastGCN [1]	VR-GCN [2]	Cluster-GCN
Time complexity	$O(L\|\|A\|\|_0F + LNF^2)$	$O(d^LNF^2)$	$O(r^LNF^2)$	$O(rLNF^2)$	$O(L\|\|A\|\|_0F + LNF^2 + r^LNF^2)$	$O(L\|\|A\|\|_0F + LNF^2)$
Memory complexity	$O(LNF + LF^2)$	$O(bd^LF + LF^2)$	$O(br^LF + LF^2)$	$O(brLF + LF^2)$	$O(LNF + LF^2)$	$O(bLF + LF^2)$

	Time		Memory		Test F1 score	
	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN
Amazon2M (2-layer)	337s	1223s	7476 MB	2228 MB	89.03	89.00
Amazon2M (3-layer)	1961s	1523s	11218 MB	2235 MB	90.21	90.21
Amazon2M (4-layer)	N/A	2289s	OOM	2241 MB	N/A	90.41

GraphSAINT

- Unbiased sampler
  - Random node/edge/walk sampler
- Unbiased aggregated representations

$$\zeta_v^{(\ell+1)} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \left(\boldsymbol{W}^{(\ell)}\right)^{\mathsf{T}} \boldsymbol{x}_u^{(\ell)} \mathbb{1}_{u|v} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \tilde{\boldsymbol{x}}_u^{(\ell)} \mathbb{1}_{u|v}$$



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$

Full GCN on $\mathcal{G}_s$

GraphSAINT Performance

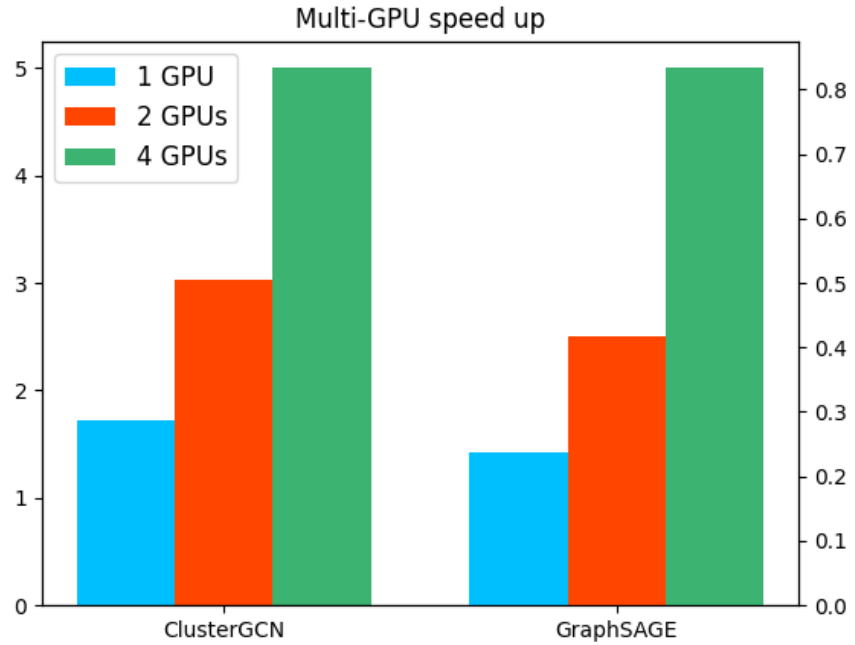
Dataset	Nodes	Edges	Degree	Feature	Classes	Train / Val / Test
PPI	14,755	225,270	15	50	121 (m)	0.66 / 0.12 / 0.22
Flickr	89,250	899,756	10	500	7 (s)	0.50 / 0.25 / 0.25
Reddit	232,965	11,606,919	50	602	41 (s)	0.66 / 0.10 / 0.24
Yelp	716,847	6,977,410	10	300	100 (m)	0.75 / 0.10 / 0.15
Amazon	1,598,960	132,169,734	83	200	107 (m)	0.85 / 0.05 / 0.10
PPI (large version)	56,944	818,716	14	50	121 (m)	0.79 / 0.11 / 0.10

Method	PPI	Flickr	Reddit	Yelp	Amazon
GCN	0.515±0.006	0.492±0.003	0.933±0.000	0.378±0.001	0.281±0.005
GraphSAGE	0.637±0.006	0.501±0.013	0.953±0.001	0.634±0.006	0.758±0.002
FastGCN	0.513±0.032	0.504±0.001	0.924±0.001	0.265±0.053	0.174±0.021
S-GCN	0.963±0.010	0.482±0.003	0.964±0.001	0.640±0.002	—‡
AS-GCN	0.687±0.012	0.504±0.002	0.958±0.001	—‡	—‡
ClusterGCN	0.875±0.004	0.481±0.005	0.954±0.001	0.609±0.005	0.759±0.008
GraphSAINT-Node	0.960±0.001	0.507±0.001	0.962±0.001	0.641±0.000	0.782±0.004
GraphSAINT-Edge	**0.981**±0.007	0.510±0.002	**0.966**±0.001	**0.653**±0.003	0.807±0.001
GraphSAINT-RW	**0.981**±0.004	**0.511**±0.001	**0.966**±0.001	**0.653**±0.003	**0.815**±0.001
GraphSAINT-MRW	0.980±0.006	0.510±0.001	0.964±0.000	0.652±0.001	0.809±0.001

Multi-GPU Training

Multi-GPU implementation:

sampling + PyTorch DDP



✓ Sampler in CogDL

[+] NeighborSampling

[+] ClusterGCN

[+] GraphSAINT

✓ 4 GPUs ~ 3x↑ speedup

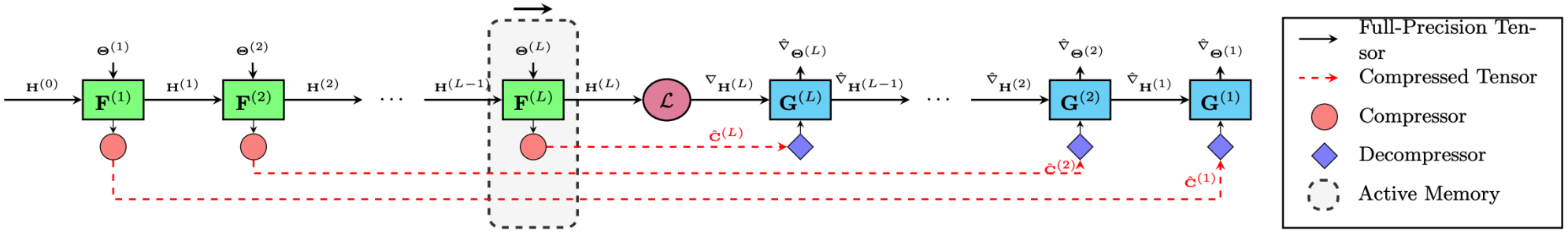
Usage: python scripts/train.py --model gcn --task node_classification --dataset reddit --trainer dist_clustergcn

Other Solutions for very Deep GNNs?

- GPU memory is the bottleneck for training very deep GNNs.
- Recall RevGNN uses reversible blocks.
- Are there other solutions?
- Activation Compressed Training!

ActNN : Activation Compressed Training

- ActNN : Reducing Training Memory Footprint via **2-Bit Activation** Compressed Training (By Jianfei Chen, Tsinghua)
- “ActNN reduces the memory footprint of the activation by 12×.”
- https://github.com/ucbrise/actnn



Chen, Jianfei, et al. "ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training." *arXiv preprint arXiv:2104.14129* (2021).

ActNN Theory

Theorem 1. *(Unbiased Gradient) There exists random quantization strategies for* $\hat{\mathbf{C}}$*, such that*

$$\mathbb{E}\left[\hat{\nabla}_{\mathbf{\Theta}}\right] = \nabla_{\mathbf{\Theta}} \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}).$$

Theorem 2. *(Convergence) If A1-A3 holds, and* $0 < \alpha \leq \frac{1}{\beta}$*, take the number of iterations* t *uniformly from* $\{1, \ldots, T\}$*, where* T *is a maximum number of iterations. Then*

$$\mathbb{E}\left\|\nabla \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}_t)\right\|^2 \leq \frac{2(\mathcal{L}(\Theta_1) - \mathcal{L}_{inf})}{\alpha T} + \alpha\beta\sigma^2. \tag{6}$$

Theorem 3. *(Gradient Variance)*

$$\mathrm{Var}\left[\hat{\nabla}_{\mathbf{\Theta}^{(l)}}\right] = \mathrm{Var}\left[\nabla_{\mathbf{\Theta}^{(l)}}\right] + \sum_{m=l}^{L} \mathbb{E}\left[\mathrm{Var}\left[\mathbf{G}_{\mathbf{\Theta}}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right) \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(m)}}\right]\right].$$

ActNN Implementation

```
class RegularLayer:
  def forward(context, input):
    context.save_for_backward(input)
    return compute_output(input)

  def backward(context, grad_output):
    input = context.saved_tensors
    return compute_gradient(grad_output, input)

class ActivationCompressedLayer:
  def forward(context, input):
    context.save_for_backward(compress(input))
    return compute_output(input)

  def backward(context, grad_output):
    input = decompress(context.saved_tensors))
    return compute_gradient(grad_output, input)
```

ActNN Performance

- Experiment on ImageNet

Bits	32	4	3	2	1.5	1.25
FP	**77.1**	N/A	N/A	N/A	N/A	N/A
BLPA	N/A	**76.6**	Div.	Div.	N/A	N/A
ActNN (L2)	N/A	-	**77.4**	0.1	N/A	N/A
ActNN (L2.5)	N/A	-	-	**77.1**	75.9	75.1
ActNN (L3)	N/A	-	-	**76.9**	76.4	75.9

Network	Batch	Total Mem. (GB)			Act. Mem. (GB)		
		FP	ActNN (L3)	R	FP	ActNN (L3)	R
ResNet-152	32	6.01	1.18	5×	5.28	0.44	12×
	64	11.32	1.64	7×	10.57	0.88	12×
	96	OOM	2.11	/	OOM	1.32	/
	512	OOM	8.27	/	OOM	7.01	/
FCN-HR-48	2	5.76	1.39	4×	4.76	0.39	12×
	4	10.52	1.79	6×	9.52	0.79	12×
	6	OOM	2.17	/	OOM	1.18	/
	20	OOM	4.91	/	OOM	3.91	/

When SpMM meets ActNN (in CogDL)

$$H^{(i+1)} = AH^{(i)}W$$

```python
class SPMMFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, rowptr, colind, feat, edge_weight_csr=None, sym=False):
        if edge_weight_csr is None:
            out = spmm.csr_spmm_no_edge_value(rowptr, colind, feat)
        else:
            out = spmm.csr_spmm(rowptr, colind, edge_weight_csr, feat)
        ctx.backward_csc = (rowptr, colind, feat, edge_weight_csr, sym)
        return out

    @staticmethod
    def backward(ctx, grad_out):
        rowptr, colind, feat, edge_weight_csr, sym = ctx.backward_csc
        if edge_weight_csr is not None:
            grad_out = grad_out.contiguous()
            if sym:
                colptr, rowind, edge_weight_csc = rowptr, colind, edge_weigh
            else:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colin
            grad_feat = spmm.csr_spmm(colptr, rowind, edge_weight_csc, grad_
            grad_edge_weight = sddmm.csr_sddmm(rowptr, colind, grad_out, fea
        else:
            if sym is False:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colin
                grad_feat = spmm.csr_spmm_no_edge_value(colptr, rowind, grad
            else:
                grad_feat = spmm.csr_spmm_no_edge_value(rowptr, colind, grad
            grad_edge_weight = None
        return None, None, grad_feat, grad_edge_weight, None
```

```python
class ActSPMMFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, rowptr, colind, feat, edge_weight_csr=None, sym=False):
        if edge_weight_csr is None:
            out = spmm.csr_spmm_no_edge_value(rowptr, colind, feat)
        else:
            out = spmm.csr_spmm(rowptr, colind, edge_weight_csr, feat)
        quantized = quantize_activation(feat, None)
        ctx.backward_csc = (rowptr, colind, quantized, edge_weight_csr, sym)
        ctx.other_args = feat.shape

        return out

    @staticmethod
    def backward(ctx, grad_out):
        rowptr, colind, quantized, edge_weight_csr, sym = ctx.backward_csc
        q_input_shape = ctx.other_args
        feat = dequantize_activation(quantized, q_input_shape)
        del quantized, ctx.backward_csc

        if edge_weight_csr is not None:
            grad_out = grad_out.contiguous()
            if sym:
                colptr, rowind, edge_weight_csc = rowptr, colind, edge_weight_cs
            else:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colind, e
            grad_feat = spmm.csr_spmm(colptr, rowind, edge_weight_csc, grad_out)
            grad_edge_weight = sddmm.csr_sddmm(rowptr, colind, grad_out, feat)
```

Experimental Results

- Default setting of CogDL

Dataset	Origin GCN	GCN + actnn
Cora	81.30 ± 0.22	81.27 ± 0.19
Citeseer	71.73 ± 0.54	71.70 ± 0.28
Pubmed	79.17 ± 0.12	79.10 ± 0.08
Flickr	50.74 ± 0.10	50.89 ± 0.04
Reddit	95.01 ± 0.02	94.89 ± 0.01

Activation Memory (GCN + ActNN)

- Setting： $H = Dropout\left(ReLU\left(BN(AHW)\right)\right)$

#dataset, #layers, #hidden	Origin GCN	GCN + actnn	ratio	Idea ratio
PPI, 5, 2048	3704	420	8.8x	raw: 32*2 / (2.125*2+1) =12.2x +bn: 32*3 / (2.125*3+1) = 13.0x +bn+dropout: 32*4 / (2.125*3+2)=15.3x
PPI, 5, 2048 (+bn)	5484	539	10.2x	
PPI, 5, 2048 (+bn, +dropout)	7711	594	13.0x	
Flickr, 5, 512	1420	154	9.2x	
Flickr, 5, 512 (+bn)	2117	201	10.5x	
Flickr, 5, 512 (+bn, +dropout)	2991	223	13.4x	
Flickr, 10, 512	3178	311	10.2x	
Flickr, 10, 512 (+bn)	4747	415	11.4x	
Flickr, 10, 512 (+bn, +dropout)	6712	465	14.4x	

Activation Memory (GraphSAGE + ActNN)

- Setting：$H = Dropout\left(ReLU\left(BN(Concat(AH, H)W)\right)\right)$

#dataset, #layers, #hidden	Origin SAGE	SAGE + actnn	ratio	Idea ratio
PPI, 5, 2048	5524	580	9.5x	raw: 32*2 / (2.125*2+1) =12.2x +bn: 32*3 / (2.125*3+1) = 13.0x +bn+dropout: 32*4 / (2.125*3+2)=15.3x
PPI, 5, 2048 (+bn)	7304	698	10.5x	
PPI, 5, 2048 (+bn, +dropout)	OOM	754	-	
Flickr, 5, 512	2457	209	11.8x	
Flickr, 5, 512 (+bn)	3155	255	12.4x	
Flickr, 5, 512 (+bn, +dropout)	4027	278	14.5x	
Flickr, 10, 512	5090	430	11.8x	
Flickr, 10, 512 (+bn)	6659	534	12.5x	
Flickr, 10, 512 (+bn, +dropout)	8624	584	14.8x	

Activation Memory (GCNII + ActNN)

- Setting：

$$\mathbf{H}^{(\ell+1)}=\sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}+\alpha_\ell\mathbf{H}^{(0)}\right)\left((1-\beta_\ell)\mathbf{I}_n+\beta_\ell\mathbf{W}^{(\ell)}\right)\right)$$

#dataset, #layers, #hidden	GCNII	GCNII + actnn	ratio	ideal
PPI, 10, 512	4008	340	11.8x	(32*3) / (2*2.125+2)=15.36
PPI, 20, 512	7708	619	12.5x	
PPI, 20, 1024	7879	603	13.1x	
Flickr, 5, 512	3421	229	14.9x	
Flickr, 10, 512	6268	413	15.2x	
Flickr, 5, 1024	6660	438	15.2x	

#layers	Origin GCNII	GCNII + actnn
32	84.83 ± 0.33	84.67 ± 0.12
64	85.13 ± 0.61	85.00 ± 0.37
128	84.83 ± 0.58	85.20 ± 0.36
256	85.00 ± 0.08	85.37 ± 0.54

Activation Memory (GIN + ActNN)

- Setting: $H^{(l)} = MLP^{(l)}\left((1+\epsilon)H^{(l-1)} + AH^{(l-1)}\right)$

$$h_G = CONCAT(READOUT(H^{(l)}), l = 0, 1, \ldots L$$

#dataset, #batch, #layers, #hidden	GIN	GIN + actnn	ratio	ideal
NCI1, 512, 20, 512	2723	262	10.4x	(32*3) / (2.125*3+1) = 13.0x
NCI1, 512, 20, 1024	5735	540	10.6x	
NCI1, 1024, 20, 512	5502	528	10.4x	
NCI1, 512, 40, 512	6231	590	10.6x	

Solutions to Very Deep GNNs

	Memory complexity	Extra time	limitations	Suitable scenario
Reversible GNNs	O(ND)	One additional forward pass	Limited feature crosses; Some operations are limited (e.g, dropout)	Arbitrary layers
ActNN + GNN	O((L/C)ND), C: compression ratio (<16)	Quantize + Dequantize	Gradient approximation; Memory complexity is still linear to L	Hundreds of layers
GNN with checkpointing	O(sqrt(L)ND)	One additional forward pass	N/A	Thousands of layers

How to Design Customized Layer?

- Message Passing!

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

- Message: func$(h_v, h_w, e_{vw}) \rightarrow m_{vw}$ # |E|*d
- Aggregate: func$(m_{vw}) \rightarrow m_v$ # |V|*d
- Update: func$(h_v, m_v) \rightarrow h_v$ # |V|*d

Message-passing Layer in CogDL

- SpMM (AX) cannot handle complex propagation operators such as involving edge features
- Implementation via message passing

```python
class BaseLayer(nn.Module):
    def __init__(self, **kwargs) -> None:
        super().__init__(**kwargs)

    def forward(self, graph, x):
        m = self.message(x[graph.edge_index[0]])
        return self.aggregate(graph, m)

    def message(self, x):
        return x

    def aggregate(self, graph, x):
        result = torch.zeros(graph.num_nodes, x.shape[1], dtype=x.dtype).to(x.device)
        result.scatter_add_(0, graph.edge_index[1].unsqueeze(1).expand(-1, x.shape[1]), x)
        return result
```

Compare GIN with GINE

$$\mathbf{x}_i' = h_{\mathbf{\Theta}}\left((1+\epsilon)\cdot\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)}\mathbf{x}_j\right)$$

```
class GINLayer(nn.Module):
    def __init__(self, apply_func=None, eps=0, train_eps=True):
        super(GINLayer, self).__init__()
        if train_eps:
            self.eps = torch.nn.Parameter(torch.FloatTensor([eps]))
        else:
            self.register_buffer("eps", torch.FloatTensor([eps]))
        self.apply_func = apply_func

    def forward(self, graph, x):
        out = (1 + self.eps) * x + spmm(graph, x)
        if self.apply_func is not None:
            out = self.apply_func(out)
        return out
```

$$\mathbf{x}_i' = h_{\mathbf{\Theta}}\left((1+\epsilon)\cdot\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)}\mathrm{ReLU}(\mathbf{x}_j + \mathbf{e}_{j,i})\right)$$

```
class GINELayer(BaseLayer):
    def __init__(self, apply_func=None, eps=0, train_eps=True):
        super(GINELayer, self).__init__()
        if train_eps:
            self.eps = torch.nn.Parameter(torch.FloatTensor([eps]))
        else:
            self.register_buffer("eps", torch.FloatTensor([eps]))
        self.apply_func = apply_func

    def forward(self, graph, x):
        m = self.message(x[graph.edge_index[0]], graph.edge_attr)
        out = self.aggregate(graph, m)
        out += (1 + self.eps) * x
        if self.apply_func is not None:
            out = self.apply_func(out)
        return out

    def message(self, x, attr):
        return F.relu(x + attr)
```

Customized Models

```python
@register_model("mygcn")
class GCN(BaseModel):

    @staticmethod
    def add_args(parser):
        parser.add_argument("--hidden-size", type=int, default=64)
        parser.add_argument("--dropout", type=float, default=0.5)

    @classmethod
    def build_model_from_args(cls, args):
        return cls(args.num_features, args.hidden_size, args.num_classes, args.dropout)
```

1. Define hyper-parameters

```python
    def __init__(self, in_feats, hidden_size, out_feats, dropout):
        super(GCN, self).__init__()
        self.conv1 = GraphConvolution(in_feats, hidden_size)
        self.conv2 = GraphConvolution(hidden_size, out_feats)
        self.dropout = nn.Dropout(dropout)
```

2. Define modules

```python
    def forward(self, graph):
        graph.sym_norm()
        h = graph.x
        h = F.relu(self.conv1(graph, self.dropout(h)))
        h = self.conv2(graph, self.dropout(h))
        return h

if __name__ == "__main__":
    experiment(task="node_classification", dataset="cora", model="mygcn")
```

3. Define forward function

Customized Datasets

```python
@register_dataset("mydataset")
class MyNodeDataset(NodeDataset):
    def __init__(self, path="data.pt"):
        self.path = path
        super(MyNodeDataset, self).__init__(path, scale_feat=False, metric="accuracy")

    def process(self):
        """You need to load your dataset and transform to `Graph`"""
        # Load and preprocess data
        edge_index = torch.LongTensor([[0, 1], [0, 2], [1, 2], [1, 3]]).t()
        x = torch.randn(4, 10)
        y = torch.LongTensor([0, 0, 1, 1])
        # provide attributes as you need
        data = Graph(x=x, y=y, edge_index=edge_index)
        torch.save(data, self.path)
        return data

if __name__ == "__main__":
    # Run with self-loaded dataset
    experiment(task="node_classification", dataset="mydataset", model="gcn")
```

Load your own data

Self-supervised Learning on Graphs

- Types of self-supervision：
  - generative learning
  - contrastive learning
- Learning paradigm：
  - Pre-training & Fine-tuning
  - Joint learning
  - Self-training
- Encoders：GCN, GAT, GIN
- Downstream tasks

Survey of Graph Self-supervised Learning

Table 1: An overview of recent generative models on graphs. For acronyms used, "PT+FT" refers to Pre-training and Fine-tuning, "JL" refers to Joint Learning, "ST" refers to Self-Training.

Model	Self-Supervision Tasks	Training Scheme	Graph Encoders	Downstream Tasks
GAE/VGAE[1]	Graph Reconstruction	PT+FT	GCN	Node/Link
EdgeMask[2]	Graph Reconstruction	JL/PT+FT	GCN	Node
AttributeMask[2]	Graph Reconstruction	JL/PT+FT	GCN	Node
GraphCompletion[3]	Graph Reconstruction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
GPT-GNN[4]	Graph Reconstruction	PT+FT	Heterogeneous Graph Transformer	Node/Link
S^2GRL[5]	Graph Property Prediction	PT+FT	GCN/GraphSAGE	Node
Distance2Cluster[2]	Graph Property Prediction	JL/PT+FT	GCN	Node
Node clustering[3]	Graph Property Prediction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
Graph partitioning[3]	Graph Property Prediction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
SuperGAT[6]	Graph Property Prediction	JL	SuperGAT	Node
Un_GraphSAGE[7]	Graph Property Prediction	PT+FT	GraphSAGE	Node
M3S[8]	Graph Property Prediction	ST	GCN	Node
AdvT[3]	Adversial Attack and Defense	JL	GCN	Node
Graph-Bert[9]	Hybrid	PT+FT	Graph Transformer	Node

Table 2: An overview of recent contrastive models on graphs.

Model	Augmentation Method	Contrastive Framework	Graph Encoders	Downstream Tasks
InfoGraph[10]	Original Graph	Deep Infomax	GIN	Graph
GraphCL[11]	Graph Corruption		GCN	Graph
GRACE[12]	Graph Corruption	InfoNCE	GCN	Node
GCC[13]	Random walk Sampling	MoCo	GIN	Node/Graph
DGI[14]	Graph Corruption + Sampling	Deep Infomax	GCN	Node
MVGRL[15]	Graph Corruption + Sampling	Deep Infomax	GCN/GraphSAGE	Node
DwGCL[16]	Graph Corruption + Sampling		GCN	Node

Results of Self-supervised Learning

- Learning paradigm:
  - Self-supervised (SL), Joint Learning (JL), unsupervised representation learning (URL)
- Semi-supervised datasets：Cora, Citeseer, PubMed
- Supervised datasets：Flickr, Reddit

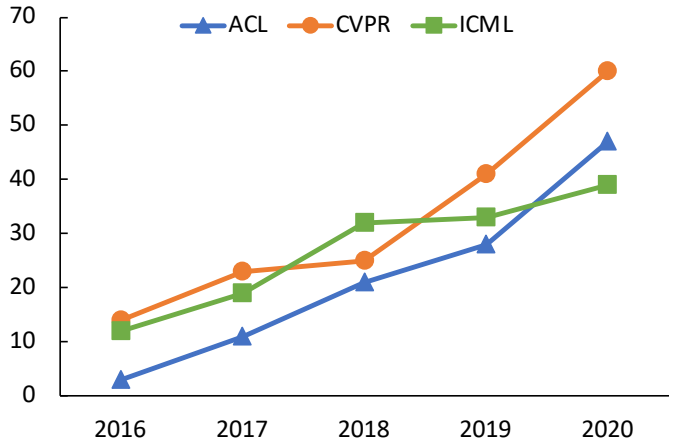
Model	Training Scheme	Cora	Citeseer	PubMed	Flickr	Reddit
Linear	SL	47.86±0.02	49.25±0.04	69.17±0.03	45.81±0.00	67.95±0.01
GCN	SL	81.53± 0.26	71.75±0.05	79.30±0.31	52.74±0.13	95.16±0.01
EdgeMask	JL	81.28±0.31	71.53±0.24	79.55±0.11	52.55±0.06	95.07±0.01
AttributeMask	JL	81.20±0.32	71.45±0.35	78.93±0.25	52.45±0.15	95.00±0.01
S^2GRL	JL	83.42±0.63	72.37±0.11	81.20±0.37	52.59±0.05	95.17±0.00
Distance2Clusters	JL	82.47±0.48	71.55±0.30	81.53±0.22	52.24±0.07	/
SuperGAT	JL	82.77±0.53	72.25±0.52	80.30±0.31	52.80±0.07	95.42±0.01
MVGRL	URL	80.76±0.80	65.84±2.72	76.01±2.46	46.08±0.39	92.76±0.22
DGI	URL	81.91±0.17	70.01±0.87	76.49±1.04	46.35±0.11	93.12±0.18
EdgeMask	URL	75.23±1.14	68.96±0.96	79.41±1.15	50.48±0.06	93.68±0.01
AttributeMask	URL	76.12±0.91	70.69±0.44	75.16±1.71	51.57±0.15	93.37±0.01
S^2GRL	URL	81.56±0.49	69.48±0.91	80.83±0.57	50.85±0.03	93.88±0.05
Distance2Cluster	URL	73.86±0.29	66.53±0.29	79.44±0.34	50.24±0.07	/

Heterogeneous Graph Benchmark (HGB)

- A unified benchmark datasets and evaluation pipelines for heterogeneous graph research.
- **Paper:** Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. *(KDD'21)*
- **Code & Data:** https://github.com/THUDM/HGB
- **Leaderboard:** https://www.biendata.xyz/hgb/
- There is also a simple baseline Simple-HGN in HGB. We find that a rather simple design of heterogeneous GNN can reach SOTA.

Simple-HGN

- GAT + relation type attention + residual connection + L2 norm
  - cogdl implementation
  - dgl implementation
- "Simple" is an interesting trend in recent years
  - The table shows the number of papers with "simple" in title or abstract for different years



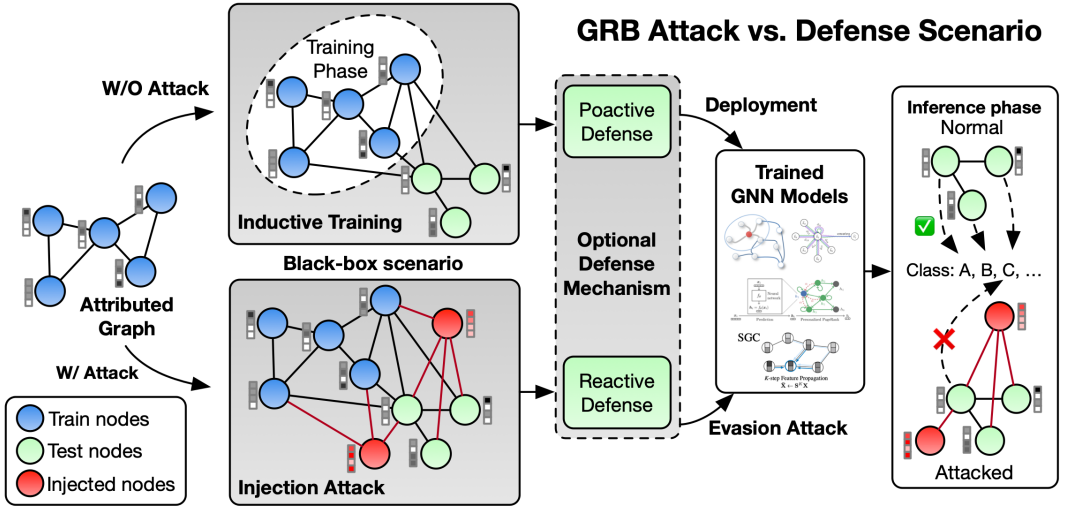
GRB
GRAPH ROBUSTNESS BENCHMARK

Background:

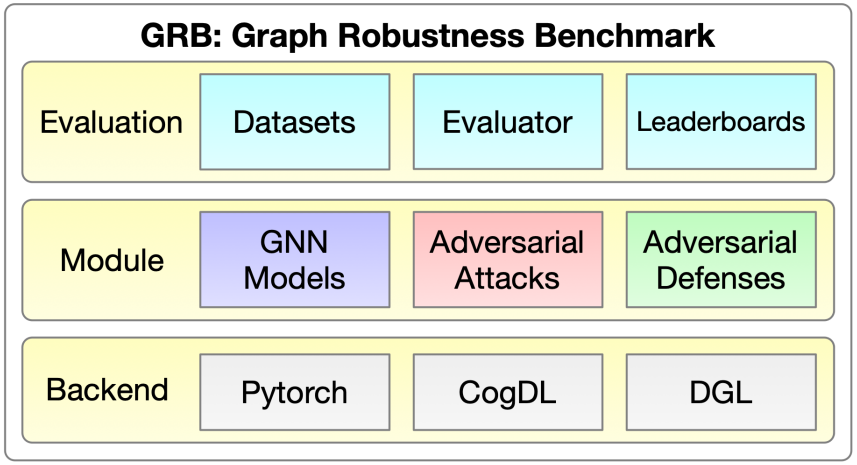
Recently, works have proved that adversarial attacks can threat the *robustness* of graph ML models in various tasks.

Problems:

1. Ill-defined threat model in previous works.
2. Absence of unified and standard evaluation approach.



Example of GRB evaluation scenario



GRB framework

Solution: **Graph Robustness Benchmark (GRB)**

Scalable, *general*, *unified*, and *reproducible* benchmark on *adversarial robustness* of graph ML models, which facilitates fair comparisons among various attacks & defenses and promotes future research in this field.

Graph Robustness Benchmark: Rethinking and Benchmarking Adversarial Robustness of Graph Neural Networks
Qinkai Zheng, Xu Zou, Yuxiao Dong, Yukuo Cen, Jie Tang

GRB
GRAPH ROBUSTNESS BENCHMARK

Graph Robustness Benchmark (GRB): Key Features				
Elaborated Datasets	***Scalability***	Datasets from small to large scales.		
	Specificity	Novel splitting scheme + Preprocessing.		
Modular Framework	***Implementation***	GNNs (GCN, SAGE, GAT, GIN, APPNP, ...)		
		Attacks (FGSM, PGD, SPEIT, TDGIA, ...)		
		Defenses (Adversarial Training, GNNGuard, ...)		
	Backend	Pytorch	CogDL	DGL
Unified Evaluation	***Scenario***	Well-defined realistic threat model.		
	Fairness	Unified settings for attackers and defenders.		
	Pipeline	Easy-to-use evaluation pipeline.		
Reproducible Leaderboard	***Reproducibility***	Availability of methods and related materials.		
	Up-to-date	Continuously maintained to track progress.		

All discussions and contributions are highly welcome!

Homepage:
https://cogdl.ai/grb/home

Github:
https://github.com/THUDM/grb

Leaderboard:
https://cogdl.ai/grb/leaderboard/

Docs: https://grb.readthedocs.io/

Google Group:
https://groups.google.com/g/graph-robustness-benchmark

Contact:
cogdl.grbteam@gmail.com
qinkai.zheng1028@gmail.com

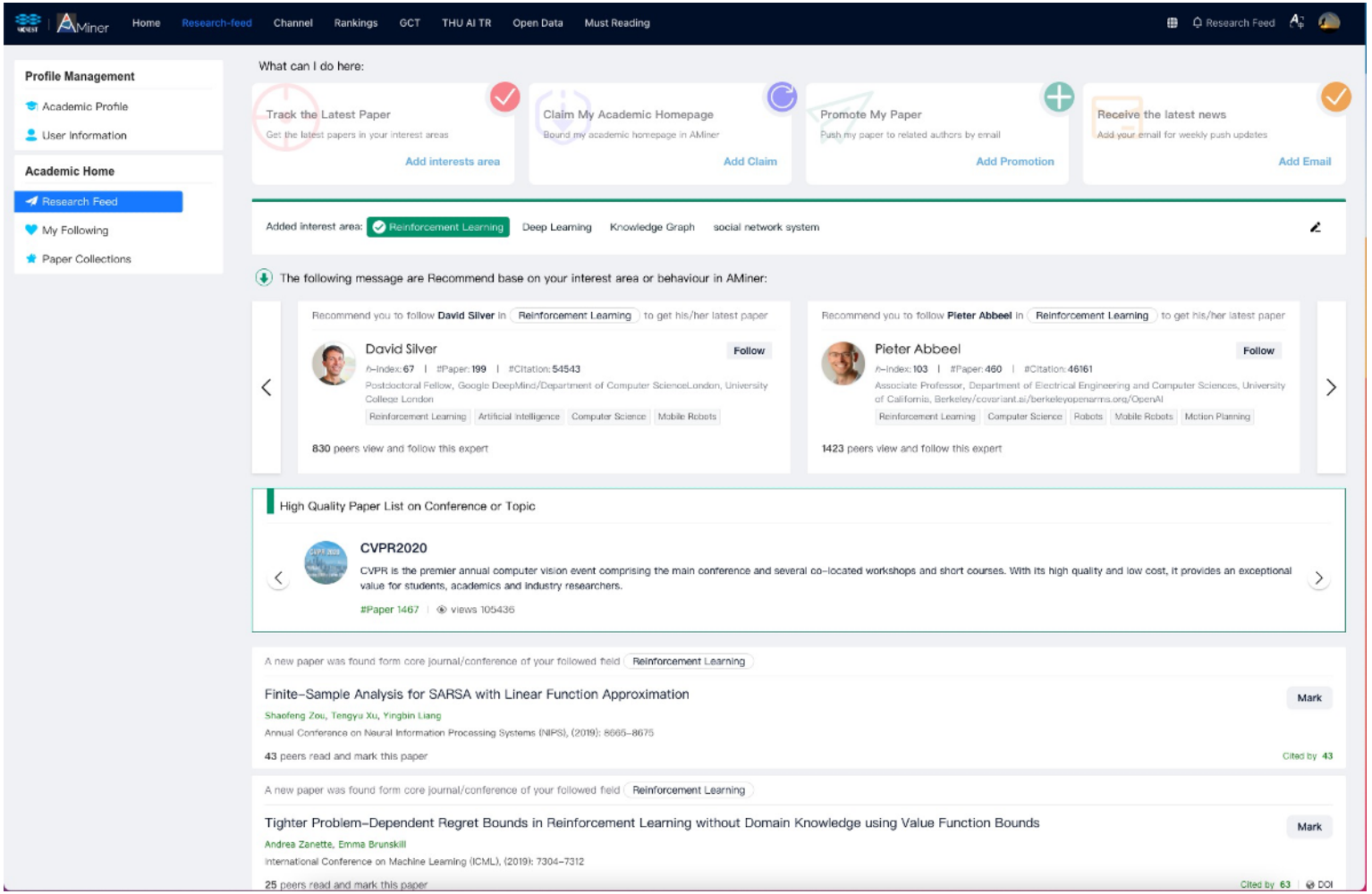
Recommendation Application

- Build recommendation via pipeline API
- Integrate LightGCN (SIGIR'20)
- Similar to Amazon Personalize

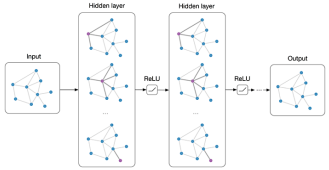
```
import numpy as np
from cogdl import pipeline

data = np.array([[0, 0], [0, 1], [0, 2], [1, 1], [1, 3], [1, 4], [2, 4], [2, 5], [2, 6]])
rec = pipeline("recommendation", model="lightgcn", data=data, max_epoch=1)
print(rec([0]))

rec = pipeline("recommendation", model="lightgcn", dataset="ali", max_epoch=1)
print(rec([0]))
```

AMiner Subscribe

- Recommend papers, scholars to users



CogDL & dgSPARSE



GNNs

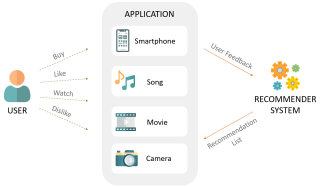
- Experiment API
- Validate ideas

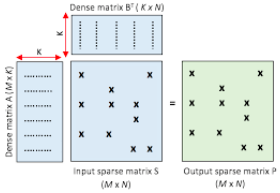
CogDL

- Fast computation
- Fast development

Apps

- Pipeline API
- Deploy Apps

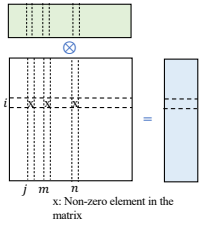




- Operator APIs
- Fast deployment

dgSPARSE

- Sparse APIs
- Hardware support



Ops

Hardware

Summary

- Preliminary
- Basic GNNs
- Advanced GNNs
  - Over-fitting and over-smoothing issues
  - From shallow GNNs to very deep GNNs
- All with CogDL
  - Efficiency (Time / Memory)
  - Customization (Layer / Model / Dataset)
  - Benchmarks (HGB / GRB)
  - Applications

Thank you !

Collaborators:

Zhenyu Hou, Guohao Dai, Yu Wang et al. (**THU**)

Yang Yang (**ZJU**)

Peng Zhang (**Zhipu**)

Hongxiao Yang, Chang Zhou, et al. (**Alibaba**)

Yukuo Cen, KEG, Tsinghua U.
Jie Tang, KEG, Tsinghua U.

https://github.com/THUDM/cogdl
http://keg.cs.tsinghua.edu.cn/jietang