Training Graph Neural Networks with 1000 Layers

Jerry Gu

Introduction

The high memory and computational complexity of GNNs make them unusable on real-world datasets which are large

- One method to fix this is lowering the parameters; however, this is counterproductive, as larger graphs would benefit from more parameters

- Recent works try to overcome it by using minibatch training or partitioning the graph; however, this introduces more hyperparameters that need to be tuned, and still does not scale well, as memory complexity still depends on the layers

Introduction

Different methods inspired from efficient CV and NLP architectures to increase memory efficiency are explored and analyzed.

Related Work

Issues with training deep GNNs

Previous work in GNNs were limited to shallow depths due to overfitting and vanishing gradients in training deep GNNs.

Issues with training deep GNNs

Current work explores different approaches to fix this:

- Skip connections adaptively select intermediate representations to the last layer

- Dilated convolutions inserts gaps between elements to expand receptive field without increasing parameters

- Several works on residual connections shown it helps with training deep GNNs

Memory complexity of training GNNs

Not viable for large, real world graphs

# of layers in network (depth)

# of nodes in graph

# of hidden channels in network (width)

$$O(LND)$$

Node-wise sampling

GraphSAGE uses node-wise sampling, which perform graph convolutions on partial node neighborhoods, and is paired with minibatch training

# of nodes per batch

# of layers

# of channels

# of sampled neighbors per node

$$O(R^L B D)$$

Layer-wise sampling

FastGCN uses layer-wise sampling, where nodes are sampled for each layer independently

$$O(\overset{\text{\# of neighbors}}{R}\overset{\text{\# of layers}}{L}\overset{\text{Batch size}}{B}\overset{\text{\# of channels}}{D})$$

Subgraph-wise sampling

ClusterGCN and GraphSAINT uses subgraph-wise sampling to split the graph into subgraphs and only train on one subgraph per iteration

$$O(\overset{\text{\# of layers}}{\downarrow \atop L}\overset{\text{Batch size}}{\downarrow \atop B}\overset{\text{\# of channels}}{\downarrow \atop D})$$

Goal

The memory complexity of these sampling methods depends on layer count, so they still do not scale well. The following will discuss methods to remove this dependency.

$$O(\overset{\text{Batch size}}{B}\overset{\text{\# of channels}}{D})$$

Building deep GNNs

Preliminaries

For graph with vertices V and edges E given as

$$G = (V = v_1, \ldots, v_n, E \subseteq V \times V)$$

with N nodes and M edges:

- Can define graph by adjacency matrix A
- Can associate V and E with their feature matrices X and U
- Use GNN operators $f_w(X, A, U) = X'$, where U is optional, w are learnable parameters, for simplicity X and X' have same dimensions, and A is the same in all layers

Residual connections

- Recent work shows that adding residual connections to vertex features can give promising results, given by $X_0 = f_w(X, A, U) + X$

- Still has high memory complexity since it is linear in number of layers

Grouped reversible GNN (RevGNN)

- The nodes’ feature matrix can be partitioned along the channels by splitting it vertically into C equal groups $X = (X_1, \ldots, X_C)$

- Grouped reversible GNN blocks (inspired by reversible networks and grouped convolutions) operates on this group of inputs and produces group of outputs $(X_1', \ldots, X_C')$

Grouped reversible GNN (RevGNN)

The forward pass is defined as

$$X'_0 = \sum_{i=2}^{C} X_i$$
$$X'_i = f_{w_i}(X'_{i-1}, A, U) + X_i, \ i \in \{1, \cdots, C\},$$

where only the output features of the last block need to be saved, so memory complexity is constant in number of layers (much lower).

Grouped reversible GNN (RevGNN)

Then, the backward pass is

$$
\begin{aligned}
X_i &= X_i' - f_{w_i}(X_{i-1}', A, U),\ i \in \{2, \cdots, C\} \\
X_0' &= \sum_{i=2}^{C} X_i \\
X_1 &= X_1' - f_{w_1}(X_0', A, U).
\end{aligned}
$$

These can be computed in parallel; then, gradients can be derived through backpropagation

Grouped reversible GNN (RevGNN)

- Normalization and dropout layers are essential for training; to avoid extra memory usage, they are embedded into the block

- The GNN block can be designed by

$$\widehat{X}_i = \mathrm{Dropout}(\mathrm{ReLU}(\mathrm{Norm}(X'_{i-1})))$$
$$\widetilde{X}_i = \mathrm{GraphConv}(\widehat{X}_i, A, U).$$

Grouped reversible GNN (RevGNN)

- Randomness of the classic dropout layers can cause reconstruction errors during backward pass
- Can fix this by storing dropout patterns, but this leads to memory complexity linear to number of layers
- Instead, the dropout pattern is shared across all layers

Weight-tied (WT-) GNN

- Improves parameter efficiency in GNNs by sharing weights across layers

- As with the original residual and reversible GNNs, the memory complexity of their weight-tied versions are O(LND) and O(ND), respectively

Deep equilibrium GNN (DEQ-GNN)

Another way to train weight-tied GNNs with memory complexity constant in layers is implicit differentiation, assuming the model's state converges. The GNN block can be built as follows:

$$
\begin{aligned}
Z' &= \mathrm{GraphConv}(Z_{\mathrm{in}}, A, U) \\
Z'' &= \mathrm{Norm}(Z' + X) \\
Z''' &= \mathrm{GraphConv}(\mathrm{Dropout}(\mathrm{ReLU}(Z'')), A, U) \\
Z_{\mathrm{o}} &= \mathrm{Norm}(\mathrm{ReLU}(Z''' + Z')),
\end{aligned}
$$

where Z represents node states (initialized as 0) and X represents injected input (initial node features)

Deep equilibrium GNN (DEQ-GNN)

- Forward pass can be implemented with a root-finding algorithm (e.x. Broyden’s method)

- Then, one can implicitly differentiate through equilibrium state to get gradients for backward pass

Analysis of Deep GNN architectures

- The performance, memory efficiency and parameter efficiency were evaluated on ogbn-proteins dataset from the Open Graph Benchmark (OGB)

- Same GNN operator, hyperparameters, and optimizers were used for fairness

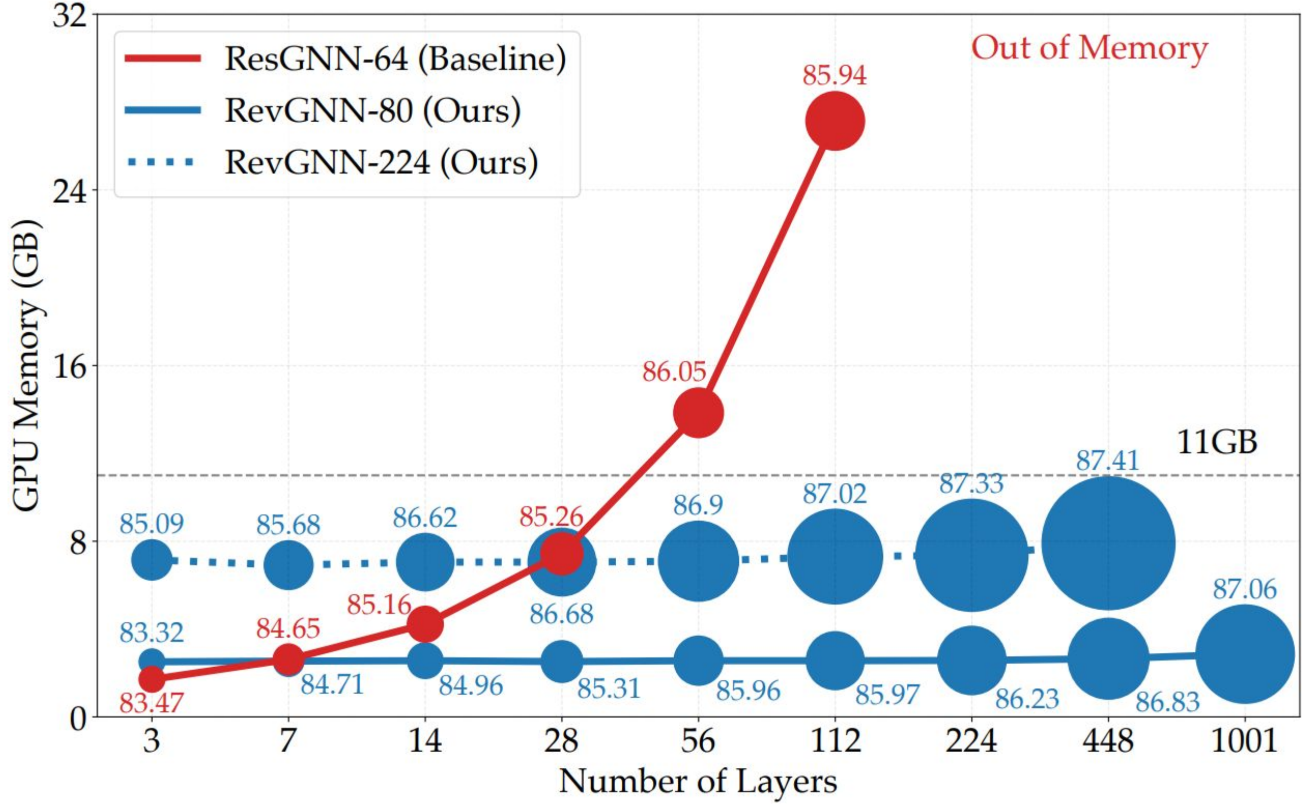
- Used mini-batch training where graphs were randomly partitioned into 10 parts for training and 5 for testing

Analysis of Deep GNN architectures

- The following plots show the reported peak GPU memory during first training epoch plotted against layer count, with point sizes representing parameter count and number representing ROC-AUC scores and the suffixes in model names represents channel count

- A recent pre-activation residual GNN (ResGNN) that consistently achieves state-of-the-art performance on OGB was used as the baseline

RevGNN


ResGNN-64 (Baseline)
RevGNN-80 (Ours)
RevGNN-224 (Ours)
Out of Memory
11GB
GPU Memory (GB)
Number of Layers
32
24
16
8
0
3
7
14
28
56
112
224
448
1001
85.94
86.05
85.26
85.16
84.65
83.47
85.09
85.68
86.62
86.9
87.02
87.33
87.41
87.06
83.32
84.71
84.96
86.68
85.31
85.96
85.97
86.23
86.83

RevGNN

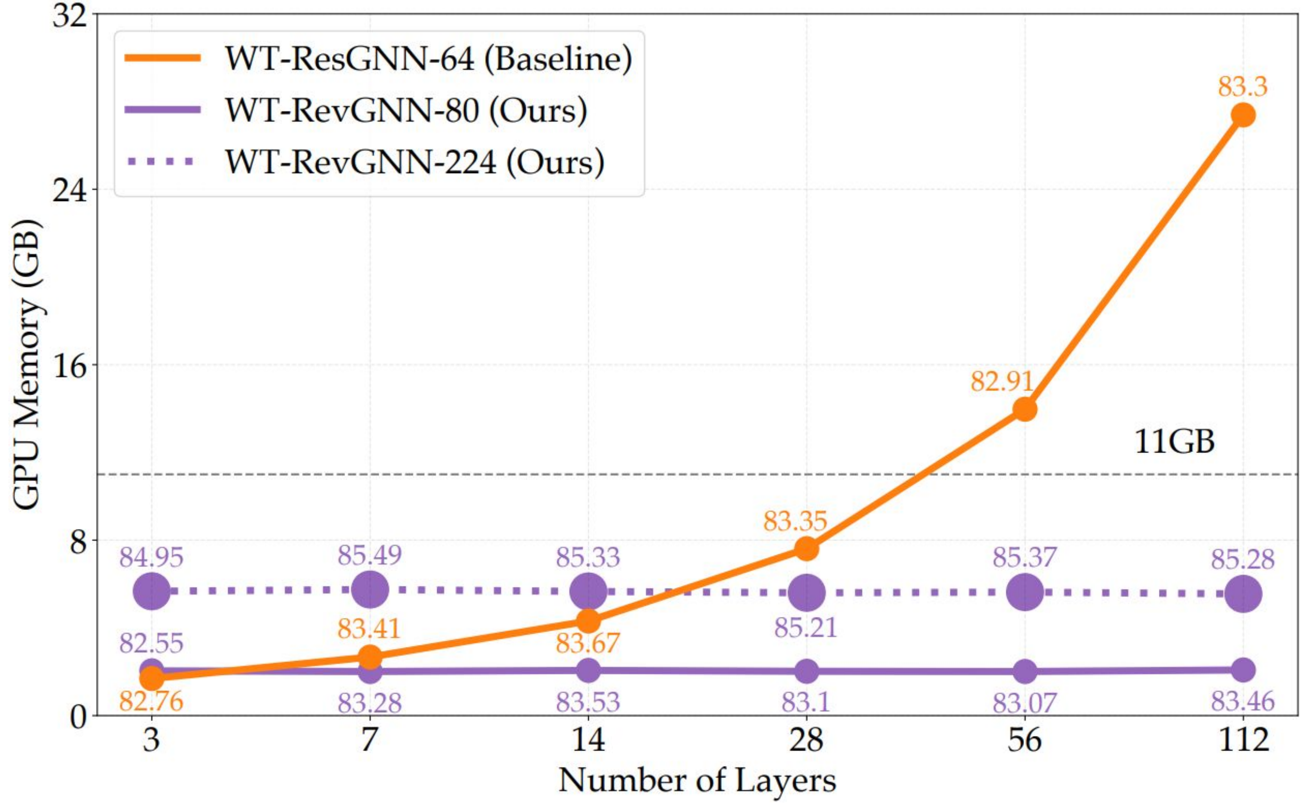
- 2 groups were used in RevGNNs
- For any # of layers, RevGNN-80 has similar performance and # of parameters as the baseline ResGNN-64, both of which increase in # of layers
- But memory consumption does not increase, unlike the baseline which runs out of memory after 112 layers, reaching a score of 85.94%
- Meanwhile RevGNN-80 can go more than 1000 layers, with an improved score of 87.06%

RevGNN

- The saved memory can be invested in making the model wider

- Increasing the channel size to 224 improves the performance further, with a score of 87.41% at 448 layers

WT-RevGNN



WT-ResGNN-64 (Baseline)
WT-RevGNN-80 (Ours)
WT-RevGNN-224 (Ours)
GPU Memory (GB)
Number of Layers
0
8
16
24
32
3
7
14
28
56
112
11GB
84.95
85.49
85.33
85.37
85.28
85.21
82.55
83.41
83.67
83.35
82.91
83.3
82.76
83.28
83.53
83.1
83.07
83.46

WT-RevGNN

- Now the # of parameters is constant in # of layers

- However, their training time and memory consumption is similar, while performance is worse as there is diminishing returns after 7 layers

WT-RevGNN

- For example at 112 layers, WT-RevGNN-224 has 337k parameters with a score of 85.28%, while RevGNN-224 has 17.1M parameters with a score of 87.02%

- Along with the results for RevGNN, this shows correlation between parameter count and performance

DEQ-GNN

- Equivalent to a weight-tied network with infinite depth

- Therefore, they have the same parameters and memory as a single layer, yet the expressiveness of a very deep network

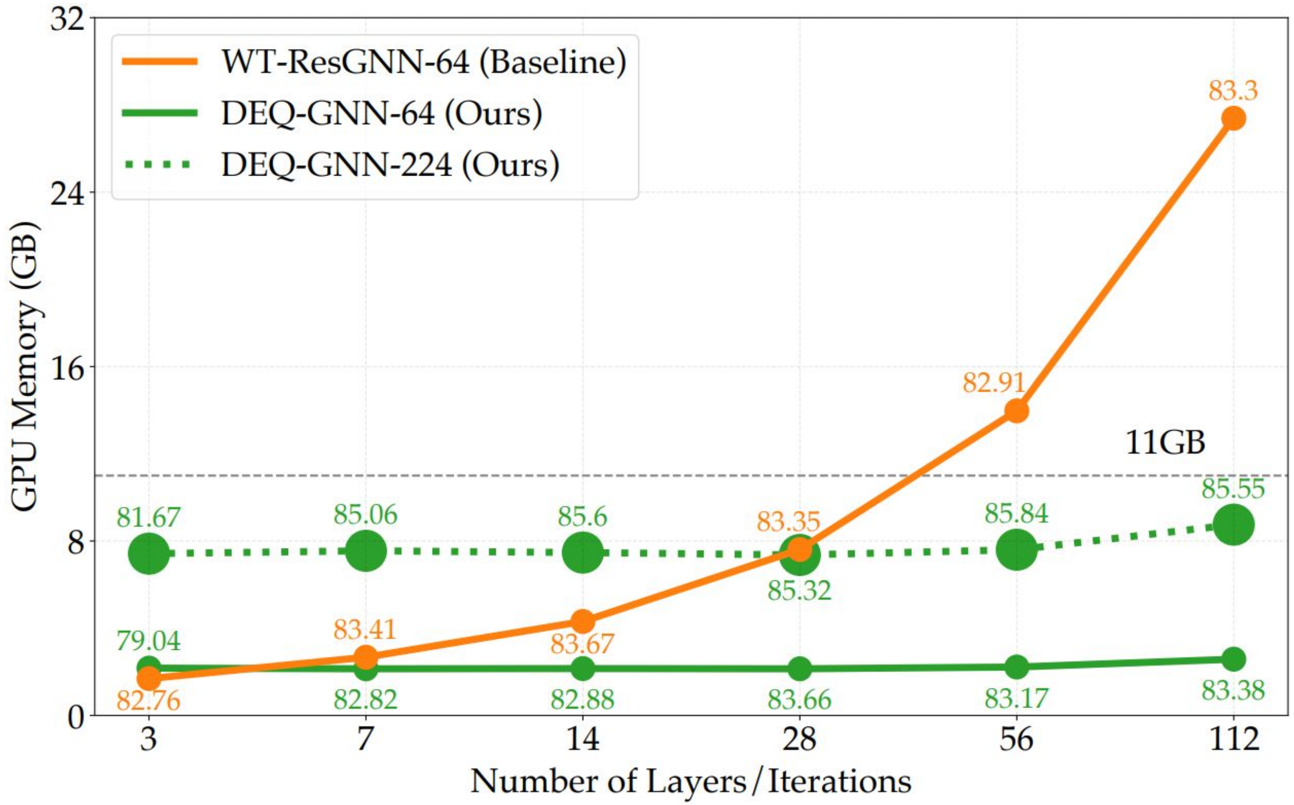
DEQ-GNN

- Broyden’s method used to find equilibrium states and inverse Jacobian for forward and backward passes, respectively

- Broyden iterations terminate when objective is less than some threshold or a maximum iteration threshold is reached

- Results for DEQ-GNN plotted against maximum iteration threshold

DEQ-GNN


WT-ResGNN-64 (Baseline)
DEQ-GNN-64 (Ours)
DEQ-GNN-224 (Ours)
GPU Memory (GB)
Number of Layers / Iterations
0
8
16
24
32
3
7
14
28
56
112
11GB
82.76
83.41
83.67
83.35
82.91
83.3
81.67
85.06
85.6
85.32
85.84
85.55
79.04
82.82
82.88
83.66
83.17
83.38

DEQ-GNN

- DEQ-GNN-64 is very similar to WT-RevGNN-80 in terms of performance, parameter count and memory consumption

- DEQ-GNN-224 has similar performance to ResGNN-64 with only 28% of the memory and 23% of the parameters, and slightly outperforms WT-RevGNN-80 with only 60% of the training time

Discussion

Reversible networks are the most promising approach in deep GNNs

- Compared to the baseline, they consume much less memory for the same parameter count while retaining state-of-the-art performance

- So by increasing the layers, they outperform the baseline more

Discussion

- Reversible networks cannot practically go to arbitrary depths since training time increases

- Increasing the group count greater than 2 can lower the parameter count and training time, but it was found to not help performance, and sometimes even harm it

Discussion

Weight-tied networks limits parameter count to 1 layer regardless of depth

- Going deeper boosts performance only up to some number of layers

- Graph equilibrium model is an extension that effectively has infinite depth, are faster to train but have more hyperparameters to tune

Discussion

- For fairness, results presented in the next section does not include pretraining, although that can further help performance

- While weight-tied networks don’t achieve state-of-the-art performance, they are parameter-efficient, which can be practically useful

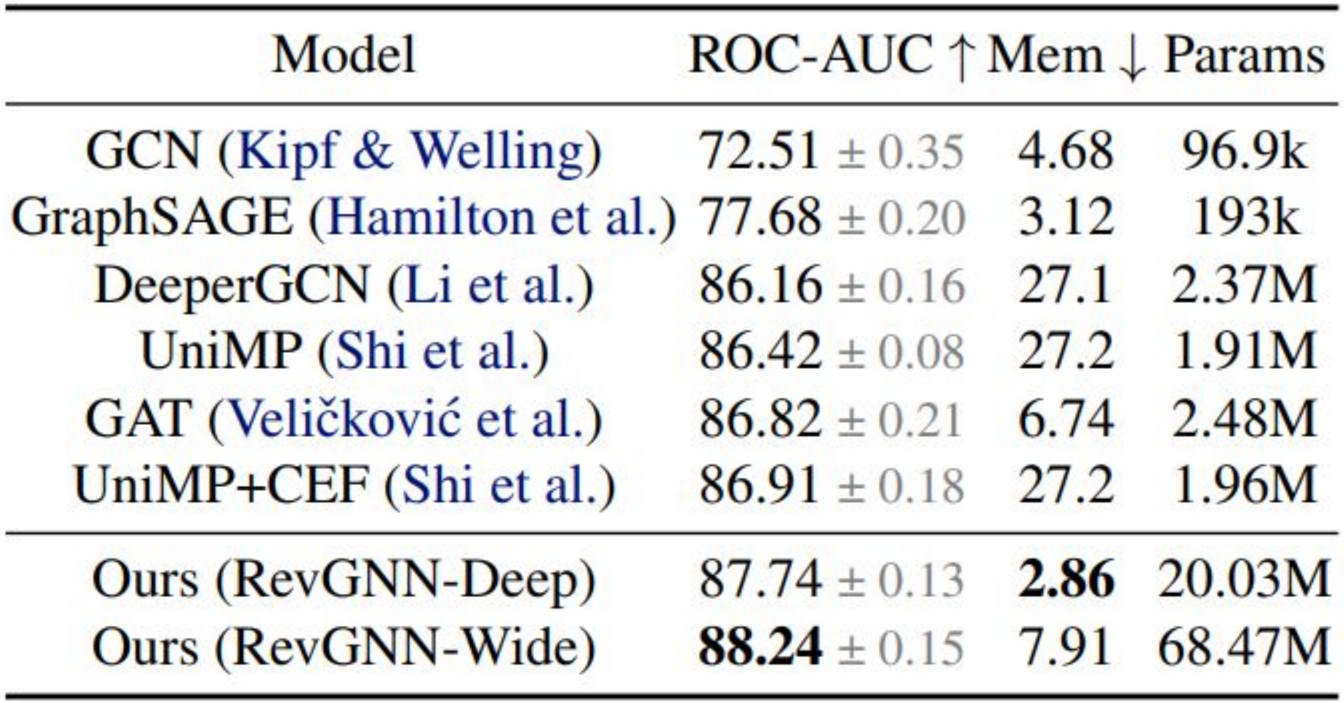
Over-parameterized Deep GNNs

State-of-the-art (SOTA) results

RevGNNs achieve new SOTA results on ogbn-proteins dataset in the OGB leaderboards

- The variants used were RevGNN-Deep which has 1001 layers and 80 channels, and RevGNN-Wide which has 448 layers and 224 channels
- Multi-view inference was used to boost performance
- Takes 13.5 and 17.1 days, respectively, to train 2000 epochs on NVIDIA V100
- These huge over-parameterized models were demonstrated to be able to train on single GPUs

State-of-the-art (SOTA) results



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

State-of-the-art (SOTA) results

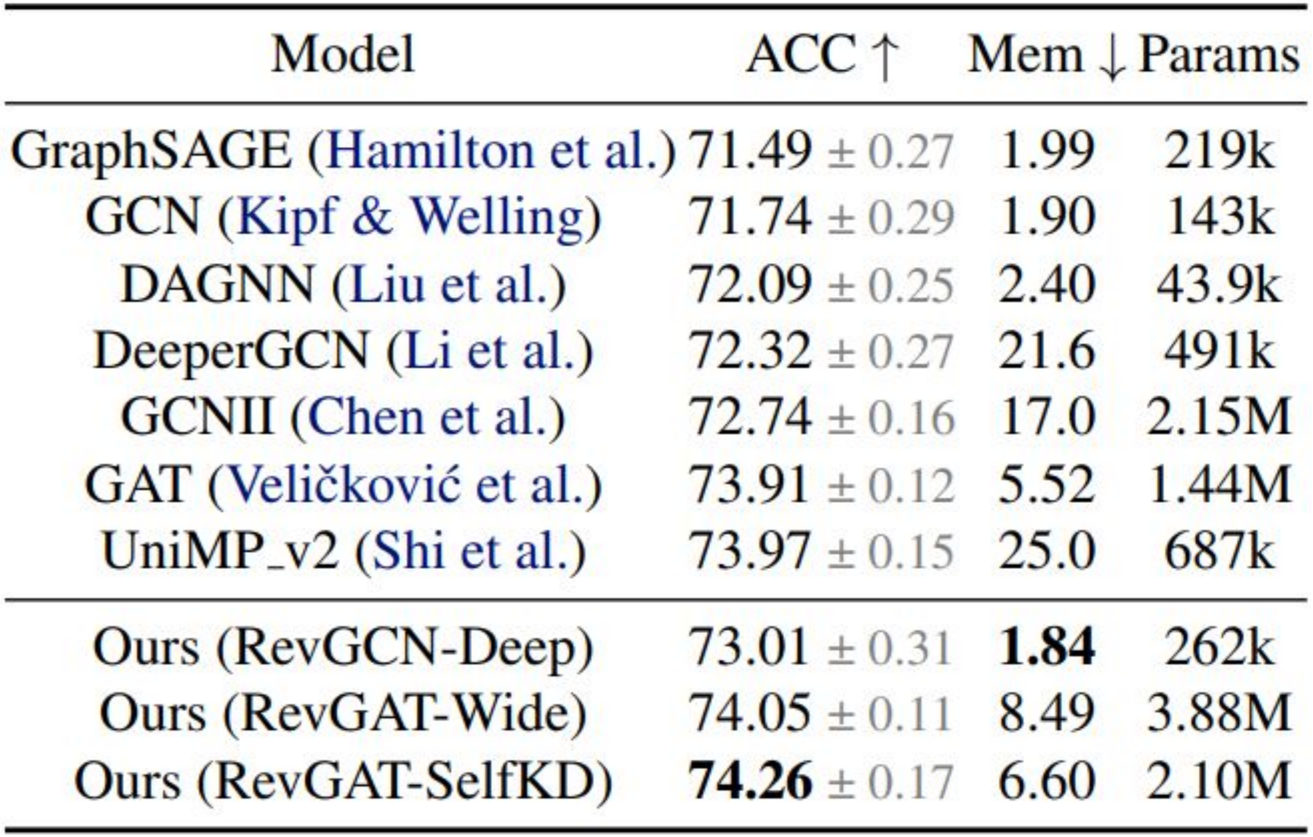
RevGNNs also achieve new SOTA results on ogbn-arxiv dataset. The variants used were:

- RevGCN-Deep which has 28 GCN layers and 128 channels

- RevGAT-Wide which uses 5 attention layers, 3 heads with 356 channels each

- RevGAT-SelfKD which uses self-knowledge distillation with 5 attention layers, 3 heads with 256 channels each

State-of-the-art (SOTA) results



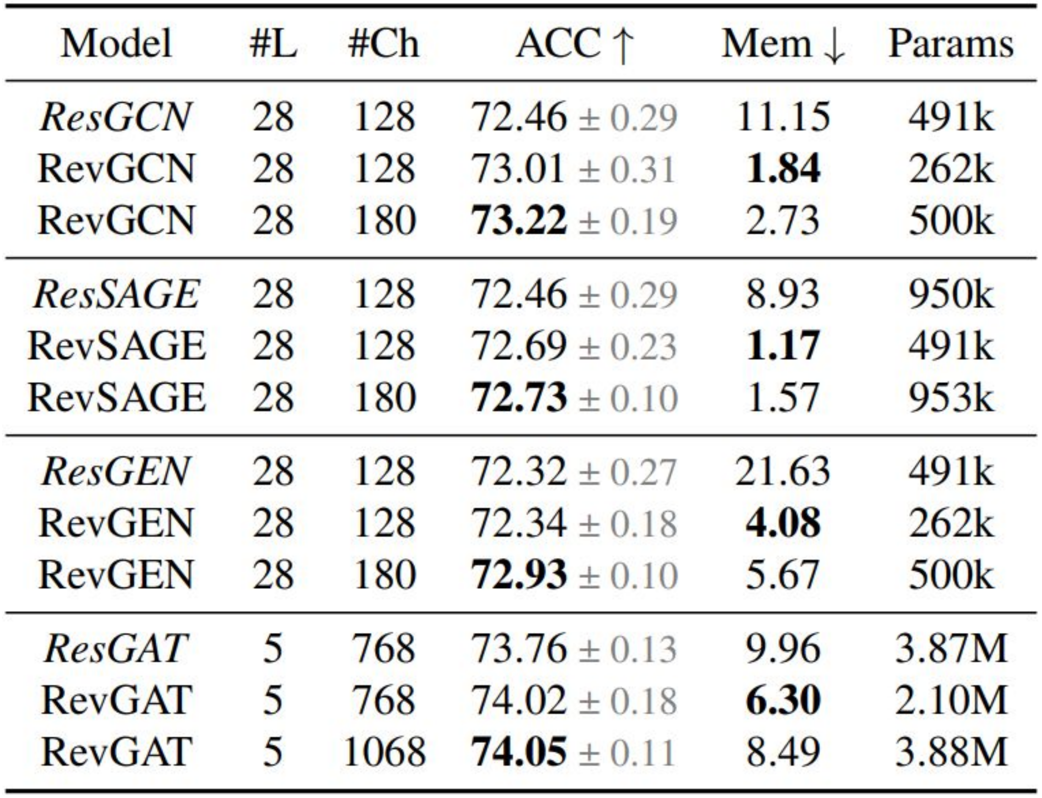
Model	ACC ↑	Mem ↓	Params
GraphSAGE (Hamilton et al.)	71.49 ± 0.27	1.99	219k
GCN (Kipf & Welling)	71.74 ± 0.29	1.90	143k
DAGNN (Liu et al.)	72.09 ± 0.25	2.40	43.9k
DeeperGCN (Li et al.)	72.32 ± 0.27	21.6	491k
GCNII (Chen et al.)	72.74 ± 0.16	17.0	2.15M
GAT (Veličković et al.)	73.91 ± 0.12	5.52	1.44M
UniMP_v2 (Shi et al.)	73.97 ± 0.15	25.0	687k
Ours (RevGCN-Deep)	73.01 ± 0.31	**1.84**	262k
Ours (RevGAT-Wide)	74.05 ± 0.11	8.49	3.88M
Ours (RevGAT-SelfKD)	**74.26** ± 0.17	6.60	2.10M

Application to different GNN operators

- The proposed methods can be applied to any GNN generally, to boost performance

- Tested on GAT, GCN, GraphSAGE, and ResGEN, and consistently outperforms their non-reversible residual counterparts with less memory used when tested on ogbn-arxiv, using full batch training as the dataset is small enough

Application to different GNN operators



Model	#L	#Ch	ACC ↑	Mem ↓	Params
ResGCN	28	128	72.46 ± 0.29	11.15	491k
RevGCN	28	128	73.01 ± 0.31	**1.84**	262k
RevGCN	28	180	**73.22** ± 0.19	2.73	500k
ResSAGE	28	128	72.46 ± 0.29	8.93	950k
RevSAGE	28	128	72.69 ± 0.23	**1.17**	491k
RevSAGE	28	180	**72.73** ± 0.10	1.57	953k
ResGEN	28	128	72.32 ± 0.27	21.63	491k
RevGEN	28	128	72.34 ± 0.18	**4.08**	262k
RevGEN	28	180	**72.93** ± 0.10	5.67	500k
ResGAT	5	768	73.76 ± 0.13	9.96	3.87M
RevGAT	5	768	74.02 ± 0.18	**6.30**	2.10M
RevGAT	5	1068	**74.05** ± 0.11	8.49	3.88M

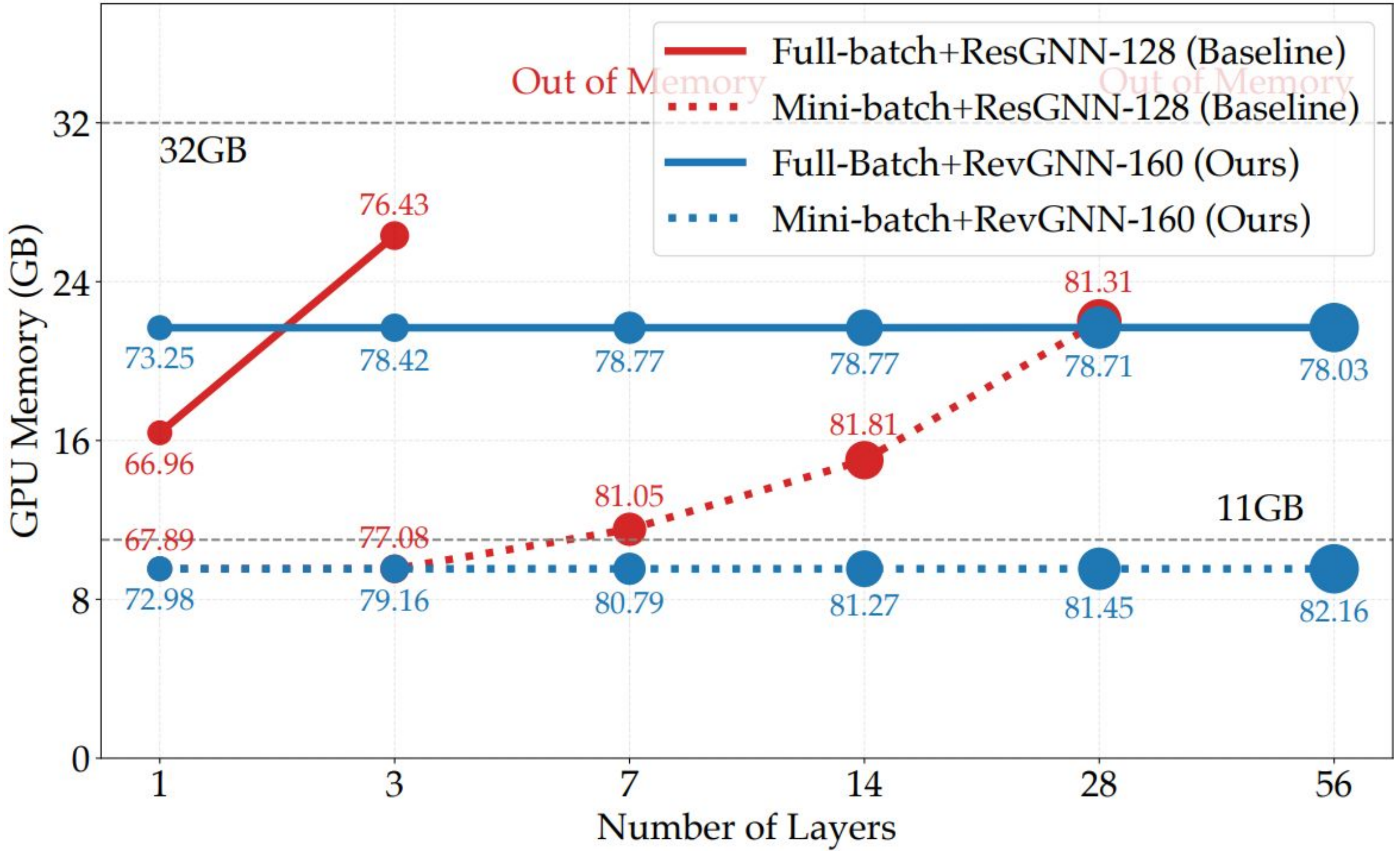
Full-batch vs mini-batch training

- The techniques introduced in this study can be applied with sampling-based approaches which also reduce memory consumption

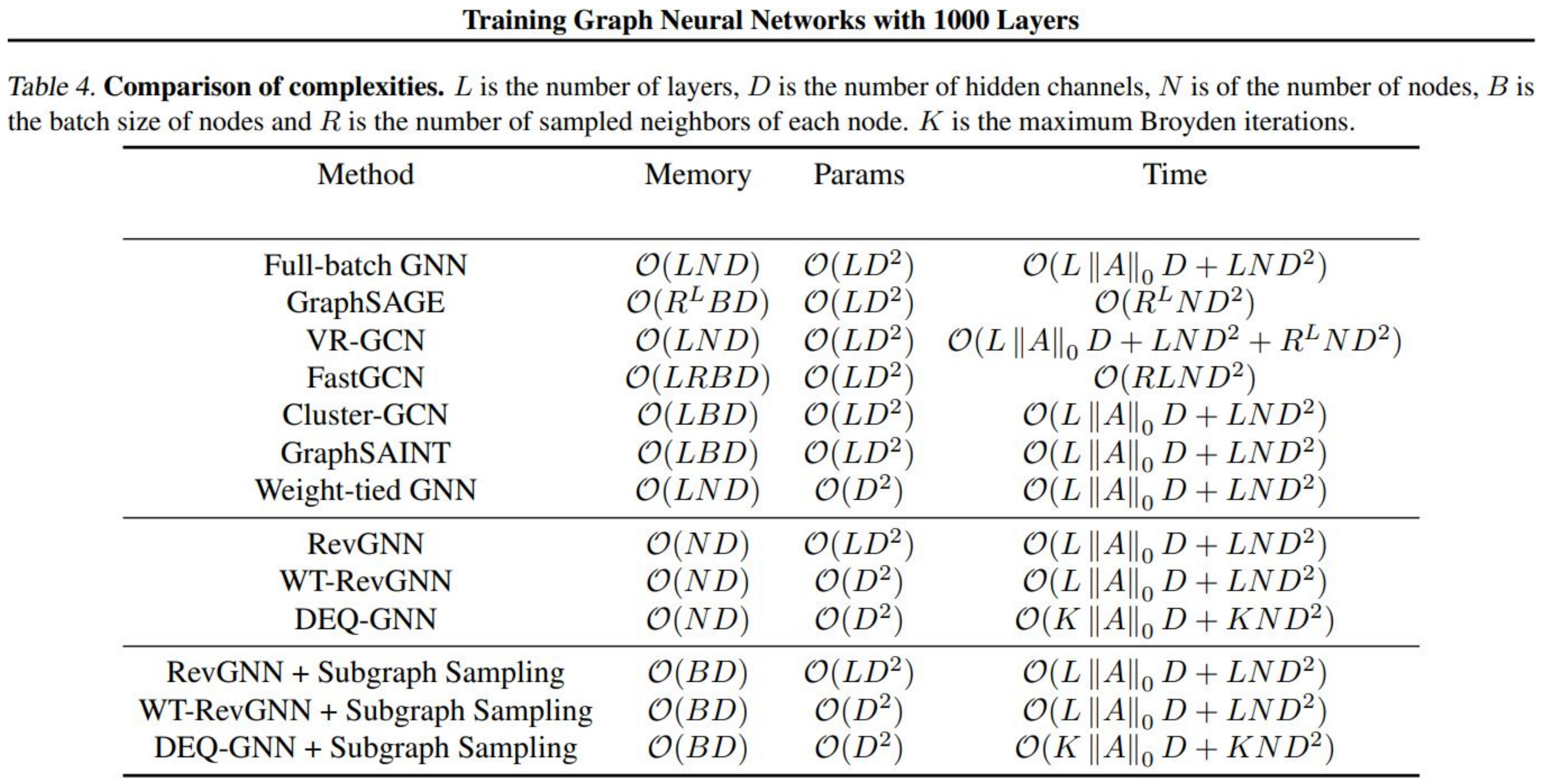
- Therefore, for example, reversible GNNs can be used with mini-batch training to further optimize memory

- An ablation study was done on ogbn-products with full-batch and a simple random-clustering mini-batch training

Full-batch vs mini-batch training



Analysis of complexities



Training Graph Neural Networks with 1000 Layers

Table 4. **Comparison of complexities.** L is the number of layers, D is the number of hidden channels, N is of the number of nodes, B is the batch size of nodes and R is the number of sampled neighbors of each node. K is the maximum Broyden iterations.

Method	Memory	Params	Time
Full-batch GNN	$\mathcal{O}(LND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
GraphSAGE	$\mathcal{O}(R^L BD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(R^L ND^2)$
VR-GCN	$\mathcal{O}(LND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2 + R^L ND^2)$
FastGCN	$\mathcal{O}(LRBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(RLND^2)$
Cluster-GCN	$\mathcal{O}(LBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
GraphSAINT	$\mathcal{O}(LBD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
Weight-tied GNN	$\mathcal{O}(LND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
RevGNN	$\mathcal{O}(ND)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
WT-RevGNN	$\mathcal{O}(ND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
DEQ-GNN	$\mathcal{O}(ND)$	$\mathcal{O}(D^2)$	$\mathcal{O}(K\left\|A\right\|_0 D + KND^2)$
RevGNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(LD^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
WT-RevGNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(D^2)$	$\mathcal{O}(L\left\|A\right\|_0 D + LND^2)$
DEQ-GNN + Subgraph Sampling	$\mathcal{O}(BD)$	$\mathcal{O}(D^2)$	$\mathcal{O}(K\left\|A\right\|_0 D + KND^2)$

Ablation studies

RevGNN-160 (with 56 layers and mini-batch training) achieved a score of 82.16% on ogbn-products, which outperforms S-GCN and SIGN, whose scores as reported in the SIGN paper was 74.87% and 77.60%, respectively

Ablation studies

ResGNN and RevGNN with same layers and parameters (28 and ~2.3M, respectively) both achieve ~77% accuracy on ogbg-ppa dataset, but RevGNN only uses 16% of GPU memory compared to ResGNN

Ablation studies

BatchNorm and GraphNorm were compared by applying them on RevGNN-256 (with 14 layers, dropout of 0.3 and learnable softmax aggregation functions) on ogbg-molhiv dataset; GraphNorm slightly outperforms BatchNorm (78.62% vs 77.82%)

Conclusion

- Several techniques were investigated to fix the high GPU memory consumption of GNNs, one of their fundamental bottlenecks

- Reversible GNNs allows them to practically go an order of magnitude deeper with less memory cost, which can be used to increase width as well, further outperforming current models

- However, this increases the training time; although by less than 40%, this is a few days on large datasets

Sources

https://arxiv.org/pdf/2106.07476.pdf