ON THE BOTTLENECK OF GRAPH NEURAL NETWORKS AND ITS PRACTICAL IMPLICATIONS

Uri Alon, Eran Yahav

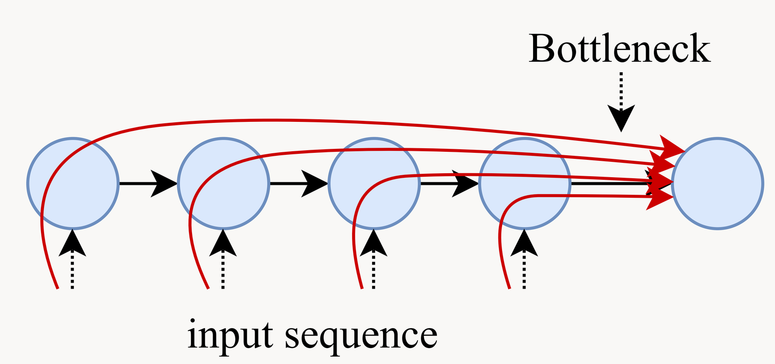
TECHNION Israel Institute of Technology

Code: https://github.com/tech-srl/bottleneck/

Contributions:

1. Introducing the **over-squashing** phenomenon of GNNs.
2. We show that **GCN** and **GIN** are more susceptible to over-squashing than **GAT** and **GGNN**.
3. We show that extensively tuned prior models suffer from over-squashing.
4. A synthetic problem of the worst case of over-squashing and its theoretical analysis.

The Bottleneck of RNNs:



The Bottleneck of GNNs:

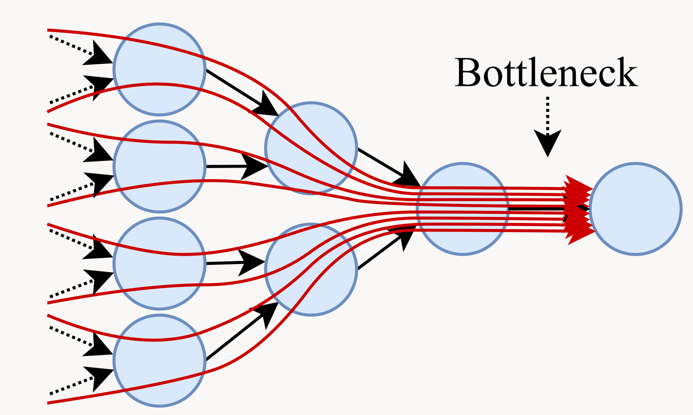


Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Synthetic Problem: NeighborsMatch

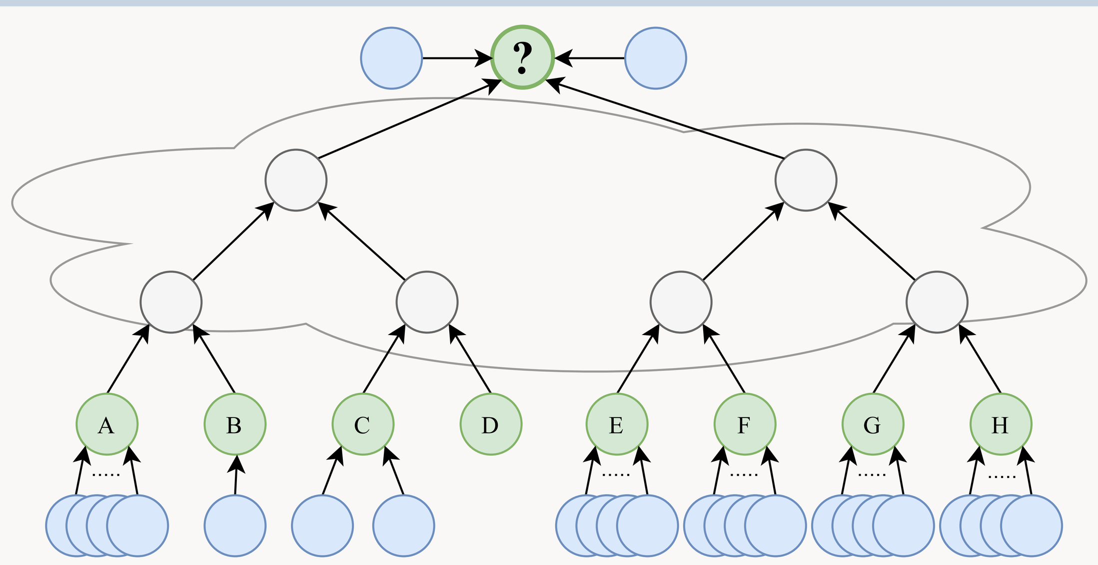


Figure 2: The NeighborsMatch: green nodes (A, B, C) have blue neighbors ( ) and an alphabetical label. The goal is to predict the label (A, B, or C) of the green node that has the same number of blue neighbors as the target node (?) in the same graph.
In this example, the correct label is **C**, because the target node has *two* blue neighbors, like the node marked with C in the same graph. Every example in the dataset has a different mapping from numbers of blue neighbors to alphabetical labels. Since information from the entire graph needs to flow into the target node, a bottleneck at the target node is inevitable.

Over-squashing Prevents GNNs from Fitting the Training Data

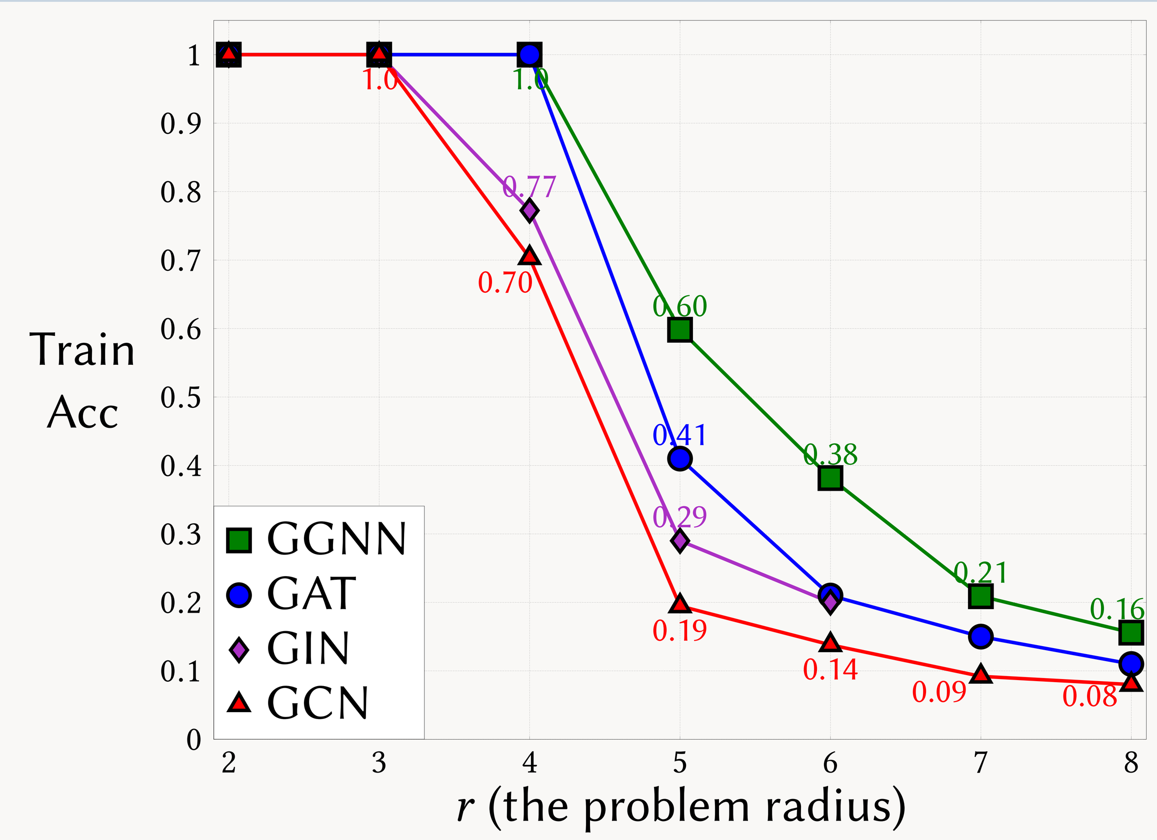


Figure 3: Accuracy across *problem radius* (tree depth) in the NeighborsMatch problem. Over-squashing starts to affect GCN and GIN even at $r = 4$.

Different GNNs are Affected by Over-Squashing *Differently*

GCN and GIN perform a temporary summation of neighbors:

- GCN: $h_v^{(k)} = ReLU\left(W^{(k)} \boxed{\sum_{u\in\mathcal{N}_v\cup\{v\}} \frac{1}{c_{v,u}} h_u^{(k-1)}}\right)$
- GIN: $h_v^{(k)} = MLP^{(k)}\left(\left(1+\epsilon^{(k)}\right) h_v^{(k-1)} + \boxed{\sum_{u\in\mathcal{N}_v} h_u^{(k-1)}}\right)$
- GAT: $h_v^{(k)} = ReLU\left(\boxed{MultiHeadAttention}\left(\mathcal{N}_v \mid h_v^{(k-1)}\right)\right)$
- GGNN: $h_v^{(k)} = \boxed{GRU}\left(h_v^{(k-1)}, \sum_{u\in\mathcal{N}_v} W_{neighbor} h_u^{(k-1)}\right)$

Theoretical Analysis: The GNN's Hidden Size Must Grow Much Faster than the Number of Layers

$\boldsymbol{d}$ — the GNN's hidden size

$\boldsymbol{r}$ — the problem radius

$$\boxed{2^{32\cdot d} > \frac{(2^r)!}{2^{2^r-1}}}$$

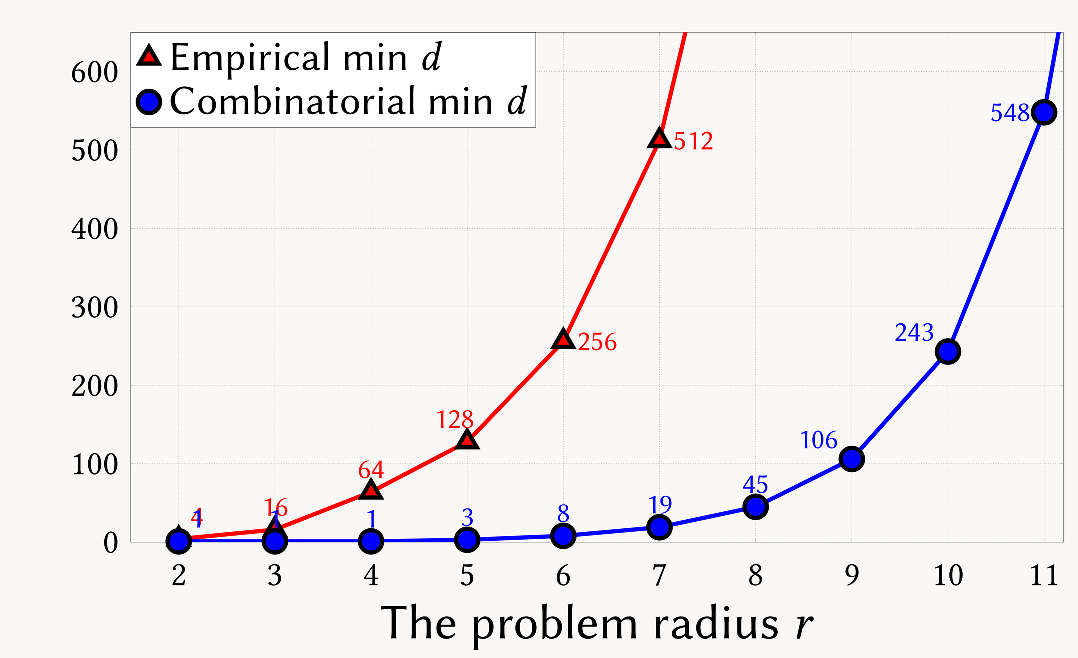


Figure 4: Combinatorial and empirical lower bounds of the model dimension given the problem radius.

A Simple Temporary Solution: Modifying the Last Layer to be Fully-Adjacent (FA) Improves SoTA Results Without Tuning

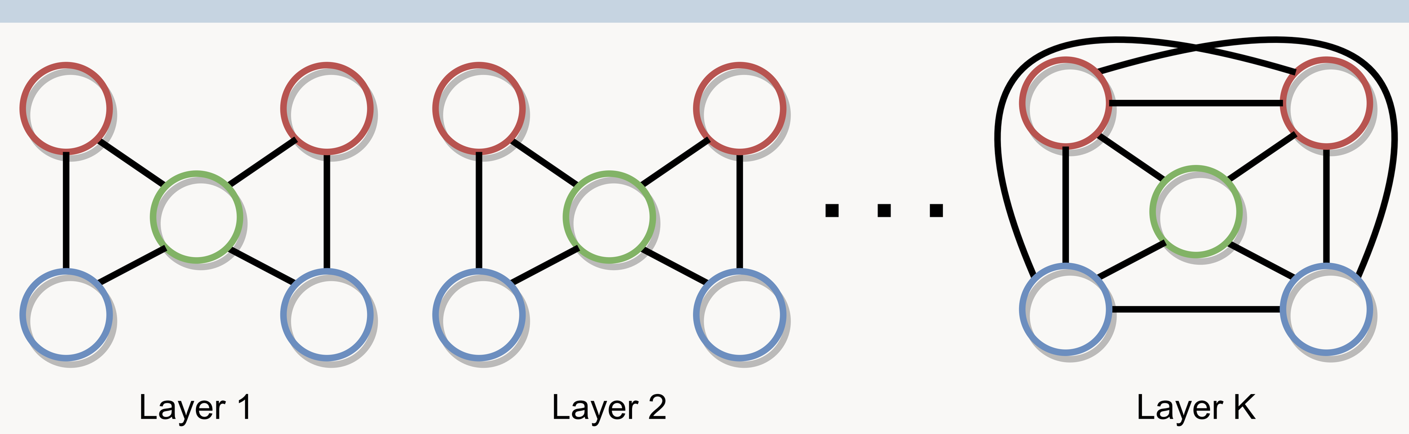


Figure 5: A possible solution: modifying the last layer to be a *fully adjacent layer.* The $K-1$ GNN layers exploit the graph structure using their original sparse topology, and only the K-th layer is an FA layer that allows the topology-aware node-representations to interact directly and consider nodes beyond their original neighbors. This simple solution's purpose is merely to demonstrate that over-squashing in GNNs is so prevalent and untreated that *even the simplest solution helps.*

This simple modification improves previous models without tuning and without adding weights:

- **+1% accuracy increase** in VarMisuse
- **-40% error reduction** in QM9
- **-5% error reduction** in NCI1
- **-12% error reduction** in ENZYMES

Over-squashing vs. Over-smoothing

$\boldsymbol{r}$ — the problem radius
$\boldsymbol{K}$ – the number of layers

$K < r$	$K >> r$	$K = R >> 1$
Under-reaching	Over-smoothing	**Over-squashing**
The GNN *cannot* fit long-range patterns	Nodes go in-distinguishable	The GNN *fails* to fit long-range patterns