

# DropGNN

Random Dropouts Increase the Expressiveness of Graph  
Neural Networks

NeurIPS 2021

Pál András Papp, Karolis Martinkus, Lukas Faber, Roger Wattenhofer

<https://arxiv.org/abs/2111.06283>  
<https://github.com/KarolisMart/DropGNN>

# Outline

- Introduction - Motivation
- Brief Overview of Graph Neural Networks (GNN)
- Brief Overview of The Weisfeiler-Lehman Isomorphic Test
- DropGNN - Method
- DropGNN - Results

# DropGNN - Introduction and Motivation

The advancement of Graph Neural Networks (GNNs) have achieved state-of-the-art performance in many disparate fields such as chemistry, physics and social networks.

However, a major drawback to typical message passing GNNs is their limitation in expressiveness often even with simple graph structures.

For example, often times distinct graph structures produce indistinguishable results when using GNNs.

# DropGNN - Introduction and Motivation (cont.)

DropGNN's novel innovation is the ability to distinguish various graph neighborhoods that cannot be separated by traditional message passing GNNs.

The Main idea being, to identify atypical special cases in graph neighborhoods through perturbations such that graphs which otherwise seem identical to standard GNN methods can be distinguished.

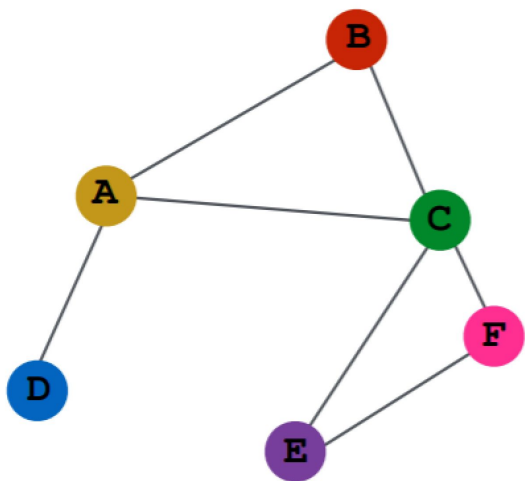
# Background

Before we get to DropGNN's contributions we need to understand 2 concepts:

- Graph Neural Network (GNN)
- Weisfeiler-Lehman Test

# Graph Neural Network (GNN)

The problem:



**Input Graph**

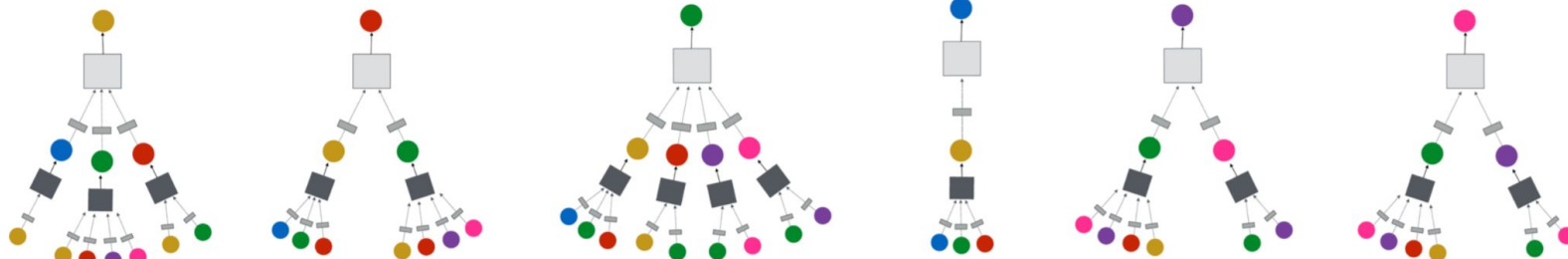
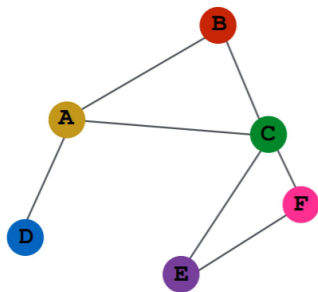


<b>A</b>	1	0.2	4	0.234	1.3
<b>B</b>	4.2	0.6	3	0.32	1
<b>C</b>	1.2	0.3	6	0.43	7
<b>D</b>	4.7	0.5	1	3	3
<b>E</b>	2	0.34	1	0.34	9.1
<b>F</b>	7	1.1	9	0.4	0.2

**Learned Graph Embeddings**

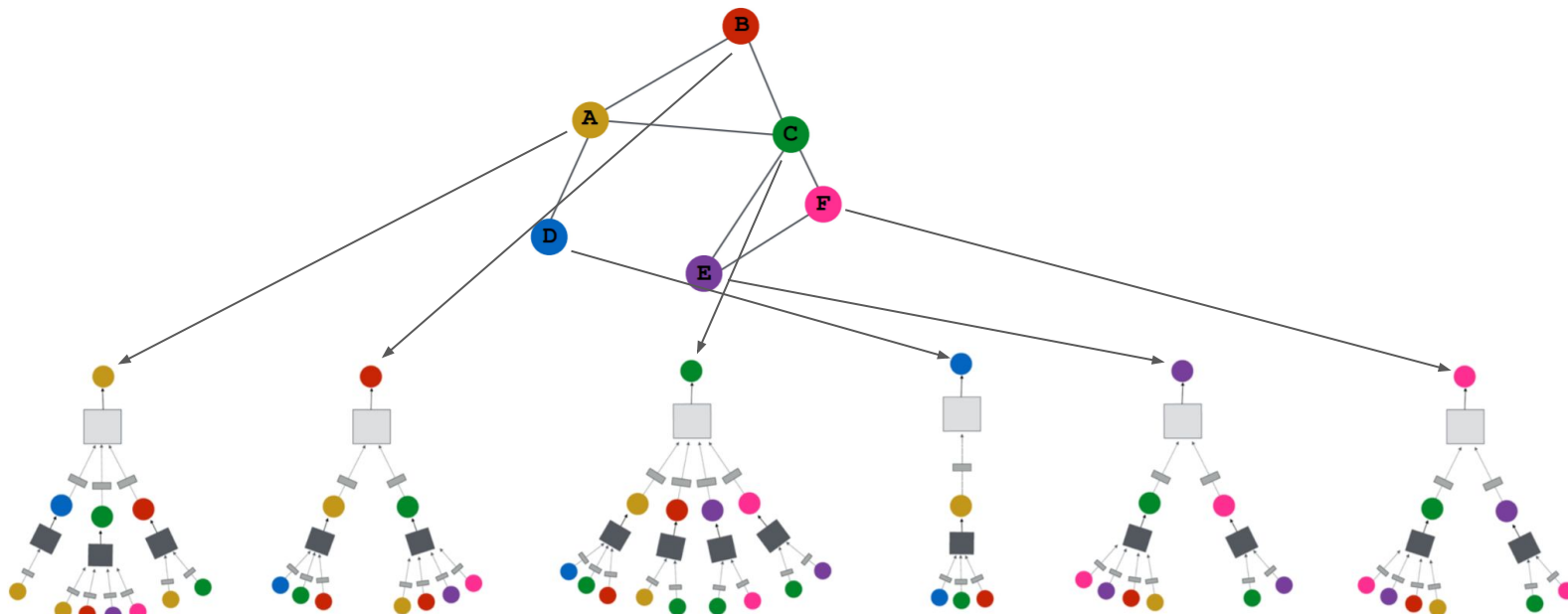
# Graph Neural Network (GNN)

Every node defines a computation graph based on its neighborhood



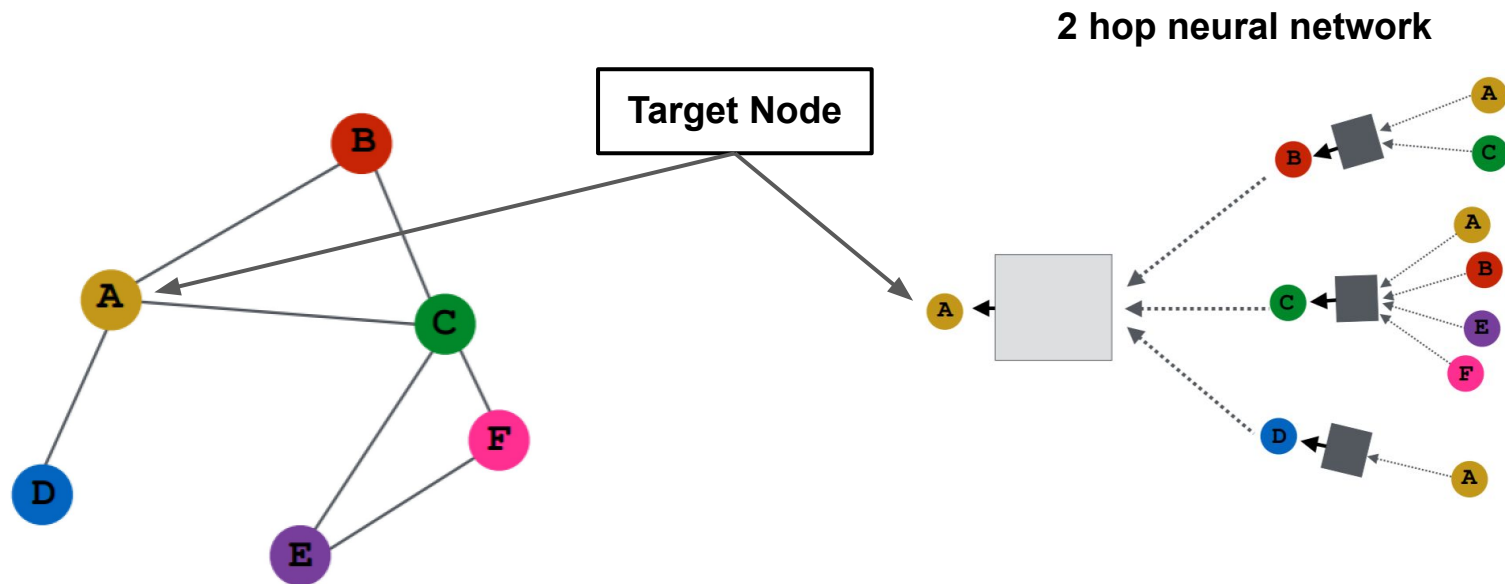
# Graph Neural Network (GNN)

Every node defines a computation graph based on its neighborhood

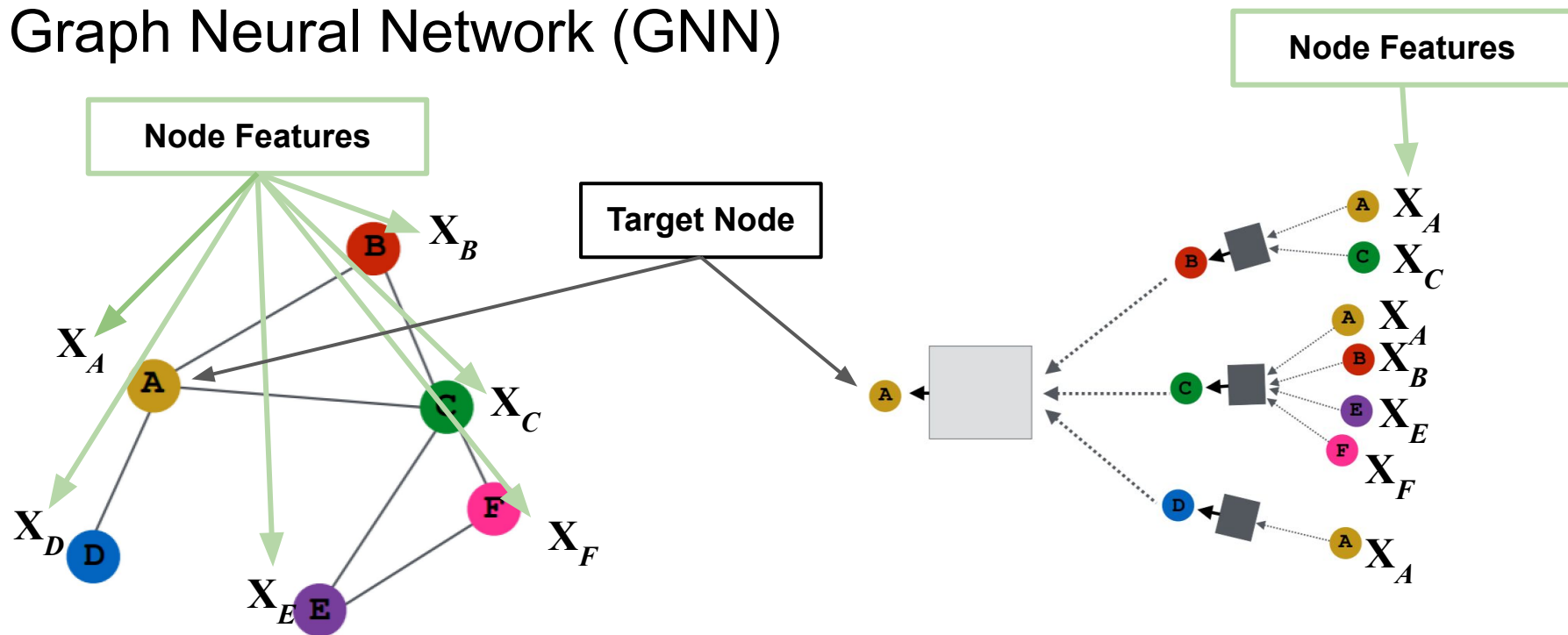




# Graph Neural Network (GNN)



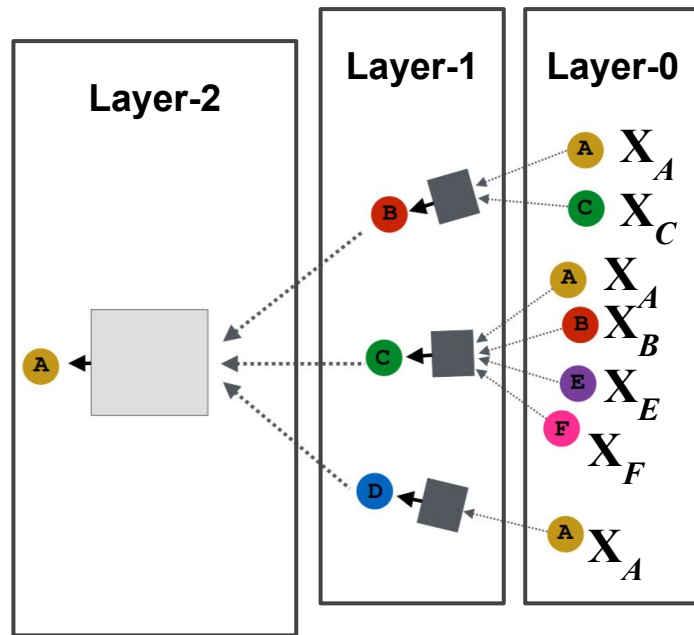
# Graph Neural Network (GNN)



# Graph Neural Network (GNN)

- Nodes have embeddings at each layer
- Layer-0 embedding of node  $u$  is the input feature vector  $\mathbf{X}_u$
- Layer- $k$  embedding gets its information from all the nodes that are  $k$ -hops away

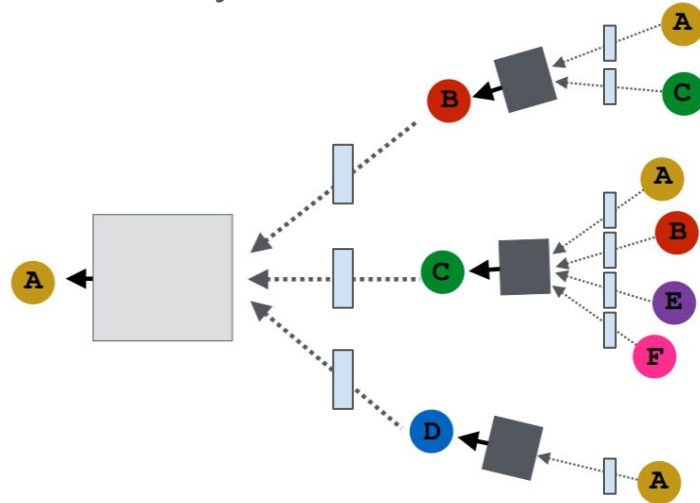
\* Key Point: Embeddings for node  $u$  at Layer-0 is different from embeddings for node  $u$  at Layer- $k$



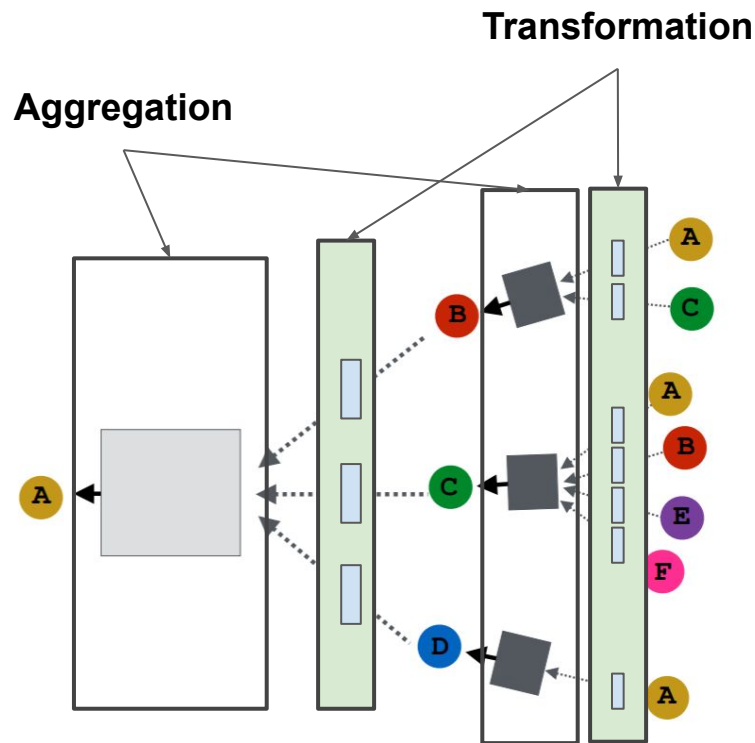
# Graph Neural Network (GNN)

## Neighborhood Aggregation:

- The key differentiator between many GNN methods is in the transformation and aggregation between layers



# Graph Neural Network (GNN)



# Graph Neural Network (GNN)

General computation

$$h_v^0 = x_v$$

$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

# Graph Neural Network (GNN)

Start with the Node Feature Vector as Embeddings  
at Layer 0

$$h_v^0 = x_v$$

Node Embedding  
at each layer

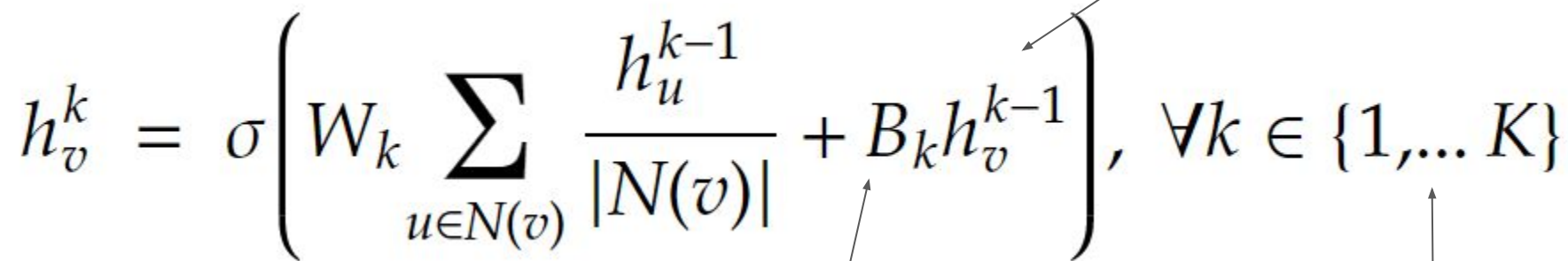
$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

Final (Output) Embedding at Layer K

# Graph Neural Network (GNN)

$$h_v^0 = x_v$$

$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$


$$z_v = h_v^K$$

Transform

Previous Layer's Embedding

For Layers 1 to K



# Graph Neural Network (GNN)

Average Previous Layer's Neighbor Embeddings  
(Aggregate)

Previous Layer's Embedding

$$h_v^0 = x_v$$
$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

For Layers 1 to K

$$z_v = h_v^K$$

Transform

Transform

# Graph Neural Network (GNN)

Non-linearity  
(ReLU)

Average Previous Layer's Neighbor Embeddings  
(Aggregate)

Previous Layer's Embedding

$$h_v^0 = x_v$$
$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

Transform

Transform

For Layers 1 to K

# Graph Neural Network (GNN)

$$h_v^0 = x_v$$

$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

Learn Weight Matrices  $W$  and  $B$



# Graph Neural Network (GNN)

$$h_v^0 = x_v$$

$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

$$z_v = h_v^K$$

Learn Weight Matrices  $W$  and  $B$ . These matrices balance between learning from your neighbors vs learning from yourself

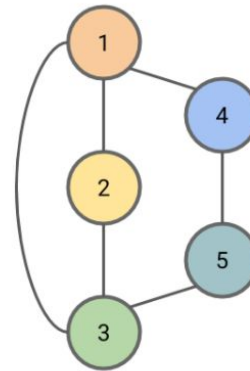
# Weisfeiler-Lehman Test (WL-test)

The WL-test attempts to determine if two graphs are isomorphic.

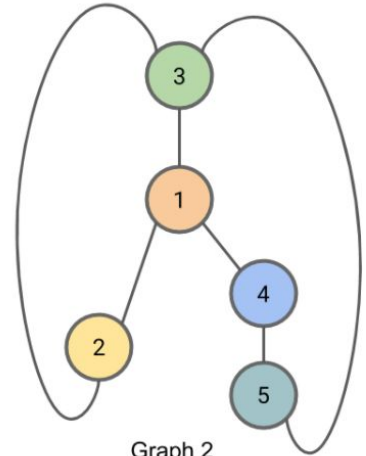
Two graphs are considered isomorphic if there is a mapping between the nodes of the graphs that preserves node adjacencies.

The WL-test is not perfect, it can only tell you:

- The graphs are not isomorphic
- The graphs are possibly isomorphic



Graph 1



Graph 2

# Weisfeiler-Lehman Test (WL-test)

Why is determining isomorphism important?

It has been shown that GNNs are at most as powerful as the WL test in distinguishing graph structures <sup>[1]</sup>.

[1] Xu, Keyulu, et al. "How powerful are graph neural networks?." arXiv preprint arXiv:1810.00826 (2018).

# Weisfeiler-Lehman Test (WL-test)

## Algorithm:

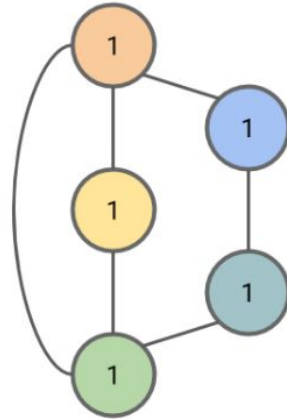
1. We initialize  $C_{0,n} = 1$  for all nodes  $n$ .
2. At iteration  $i$  of the algorithm (beginning with  $i = 1$ ), for each node  $n$ , we set  $L_{i,n}$  to be a tuple containing the node's old label  $C_{i-1,n}$  and the multiset\* of compressed node labels  $C_{i-1,m}$  from all nodes  $m$  neighboring  $n$  from the previous iteration ( $i - 1$ ).
3. We then complete iteration  $i$  by setting  $C_{i,n}$  to be a new “compressed” label, such as a hash of  $L_{i,n}$ . Any two nodes with the same labels  $L_{i,n}$  must get the same compressed label  $C_{i,n}$ .
4. Partition the nodes in the graph by their compressed label. Repeat 2 + 3 for up to  $N$  (the number of nodes) iterations, or until there is no change in the partition of nodes by compressed label from one iteration to the next.

\* Note: Multisets are similar to sets but they allow duplicate entries

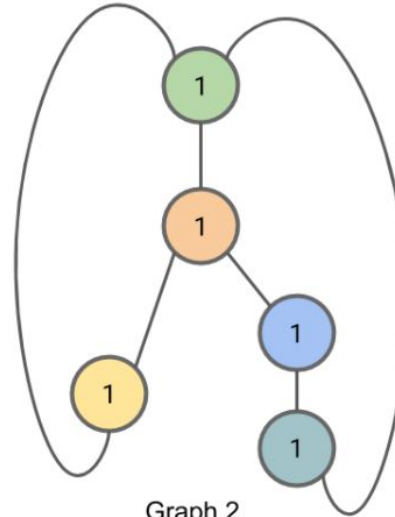
# Weisfeiler-Lehman Test (WL-test)

To initialize the algorithm (Step 1), we set  $C_{0,n} = 1$  for all nodes  $n$ .

$C_0$



Graph 1

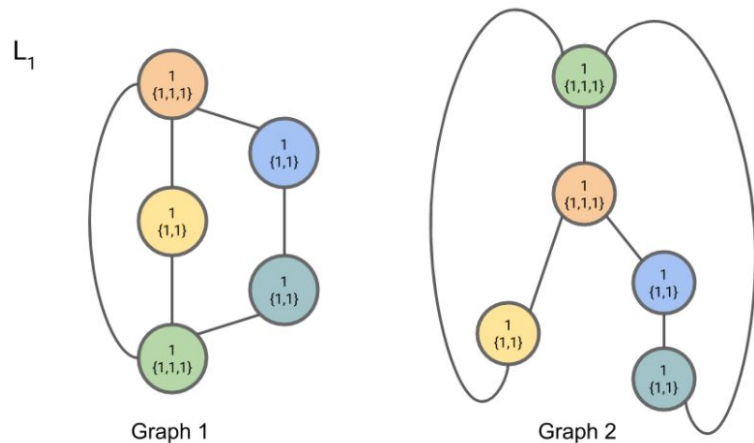


Graph 2

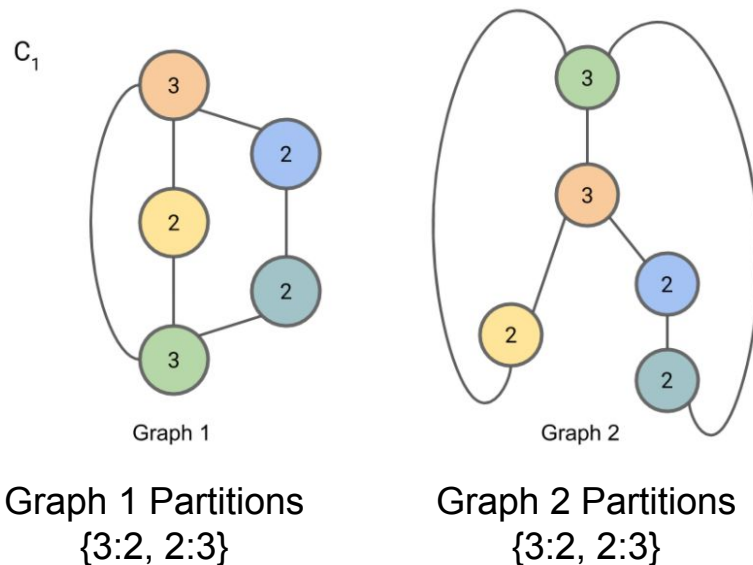


# Weisfeiler-Lehman Test (WL-test)

For iteration 1, Step 2, we compute  $L_{1,n}$ . The first part of a node's  $L$  is the node's old compressed label; the second part of a node's  $L$  is the multiset of the neighboring nodes' compressed labels.

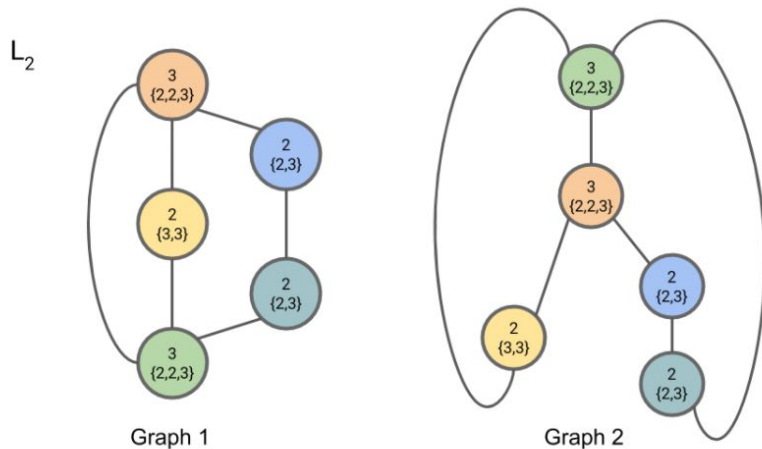


For iteration 1, Step 3, we introduce “compressed” labels  $C_{1,n}$  for the nodes:

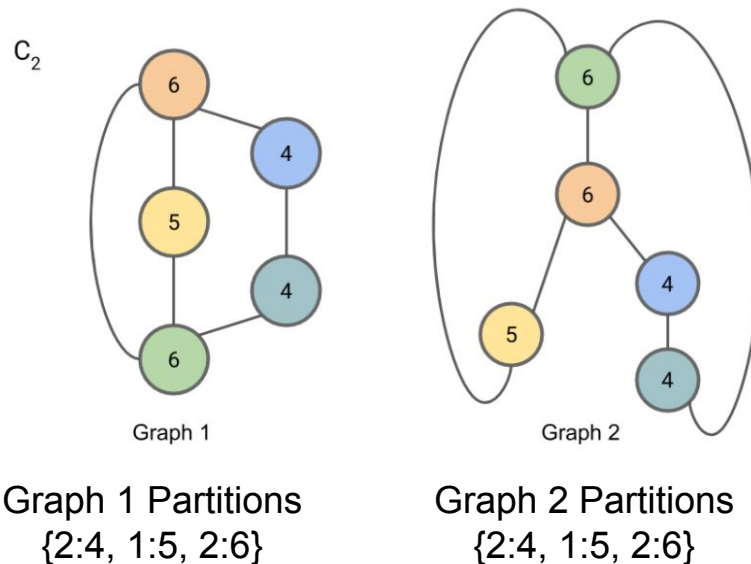


# Weisfeiler-Lehman Test (WL-test)

We now begin iteration 2. In iteration 2, Step 2, we compute  $L_{2,n}$ :



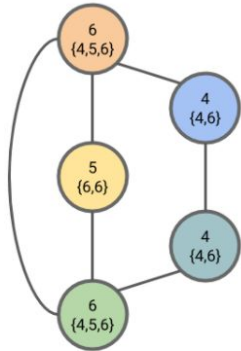
In iteration 2, Step 3, we compute  $C_{2,n}$ :



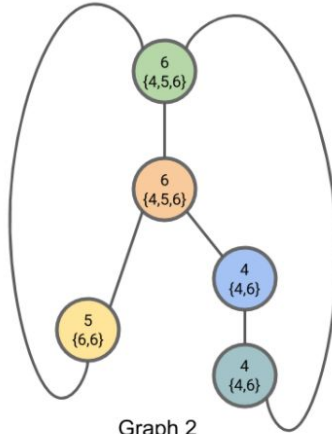
# Weisfeiler-Lehman Test (WL-test)

In iteration 3, Step 2, we compute  $L_{3,n}$ :

$L_3$



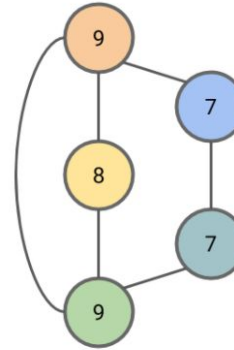
Graph 1



Graph 2

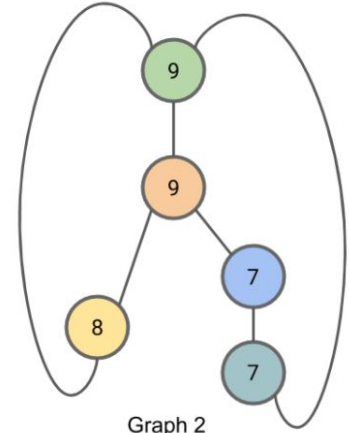
In iteration 3, Step 3, we compute  $C_{3,n}$ :

$C_3$



Graph 1

Graph 1 Partitions  
 $\{2:7, 1:8, 2:9\}$



Graph 2

Graph 2 Partitions  
 $\{2:7, 1:8, 2:9\}$

# Weisfeiler-Lehman Test (WL-test)

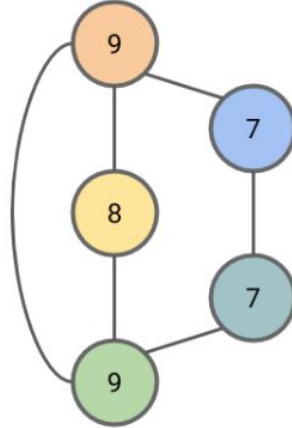
In iteration 3, Step 3, we compute  $C_{3,n}$ :

We stop at iteration 3 because the partition of the nodes did not change from the previous step

Since the 2 partitions are the same we can state that the 2 graphs are possibly identical.

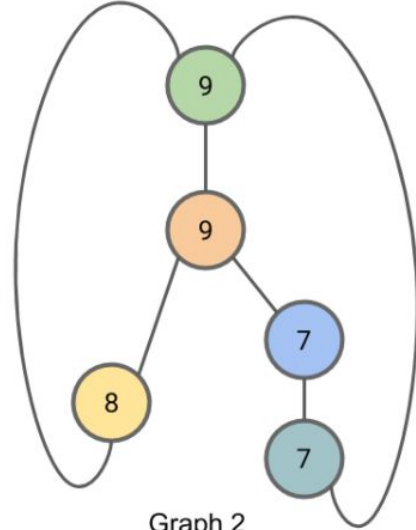
However, if the 2 partitions were different we could definitively conclude that the 2 graphs are different.

$C_3$



Graph 1

Graph 1 Partitions  
 $\{2:7, 1:8, 2:9\}$



Graph 2

Graph 2 Partitions  
 $\{2:7, 1:8, 2:8\}$

# Weisfeiler-Lehman Test (WL-test)

Why is determining isomorphism important?

It has been show that GNNs are at most as powerful as the WL test in distinguishing graph structures <sup>[1]</sup>.

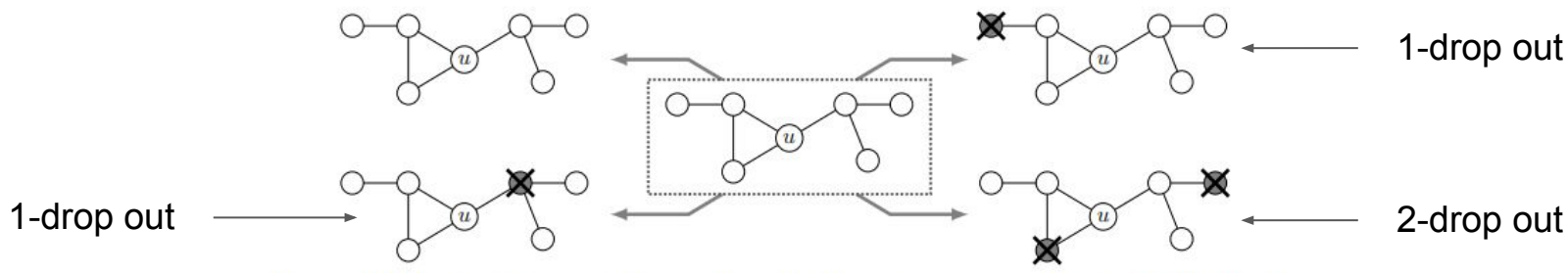
This is an important limitation with respect to the expressive power of GNNs

To address this limitation, DropGNN can be used to increase the expressive power of traditional message passing GNNs

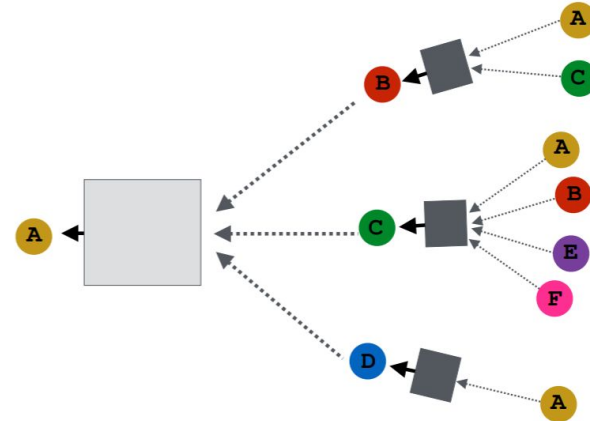
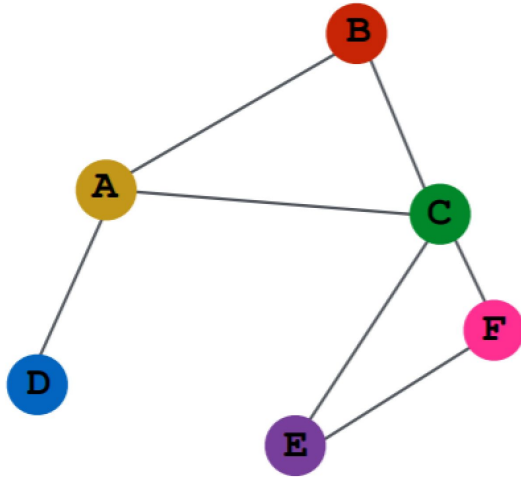
[1] Xu, Keyulu, et al. "How powerful are graph neural networks?." arXiv preprint arXiv:1810.00826 (2018).

# DropGNN

The main idea of DropGNNs is to execute multiple independent runs of the GNN. In each run, every node of the GNN is removed with probability  $p$ , independently from all other nodes. If a node  $v$  is removed during a run, then  $v$  does not send or receive any messages to/from its neighbors and does not affect the remaining nodes in any way. Essentially, the GNN behaves as if  $v$  (and its incident edges) were not present in the graph in the specific run, and no embedding is computed for  $v$  in this run

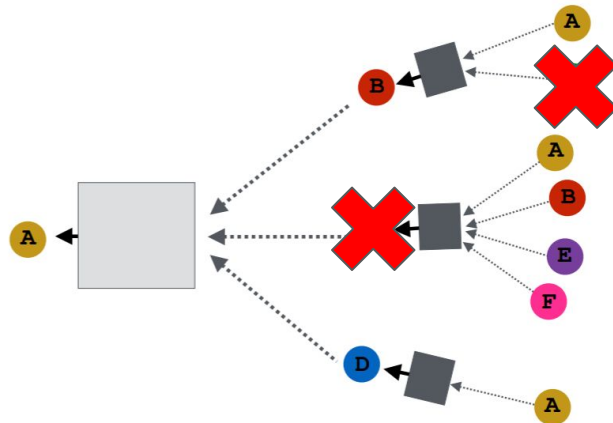
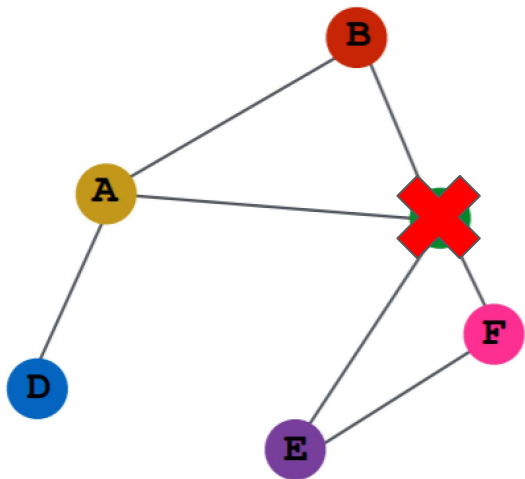


# DropGNN - 1-dropout



# DropGNN - 1-dropout

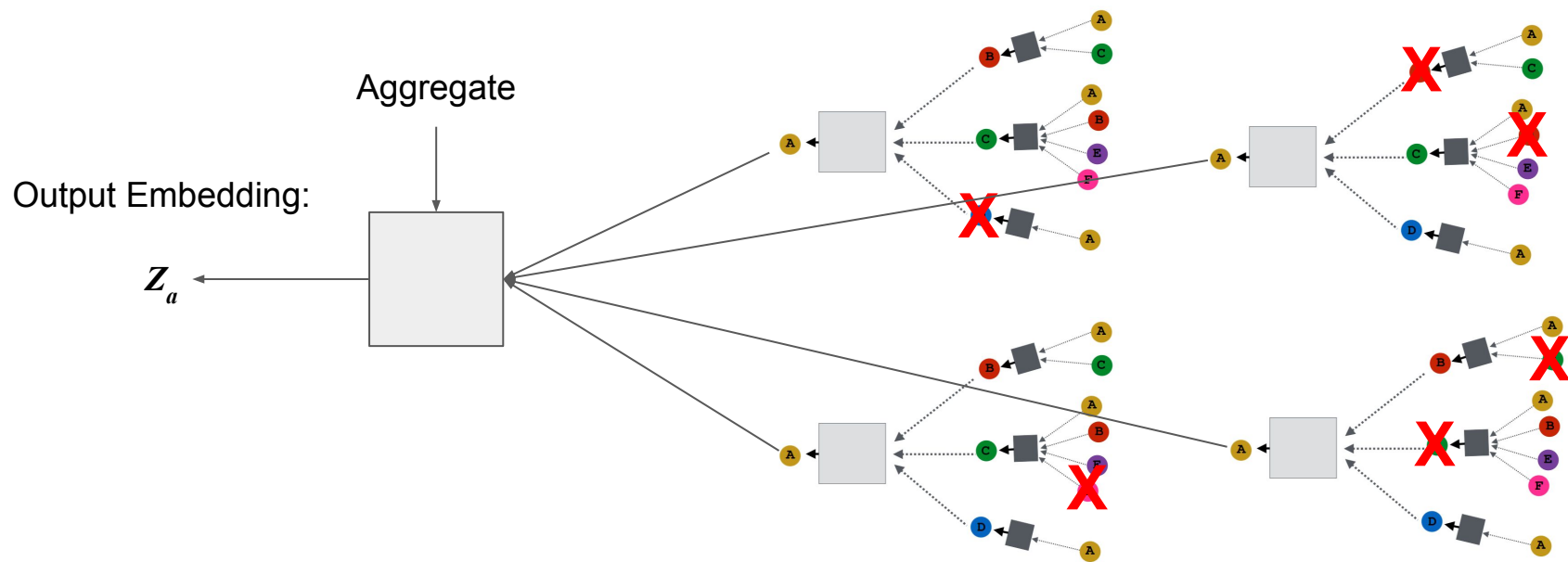
For a specific run we drop out node  $C$   
No messages from  $C$  propagate back to  $A$





# DropGNN - 1-dropout

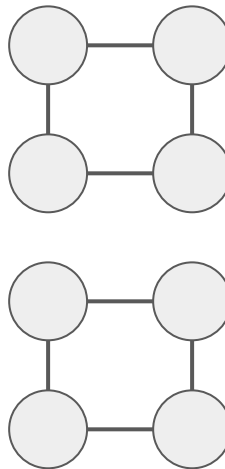
$r$  Random Runs



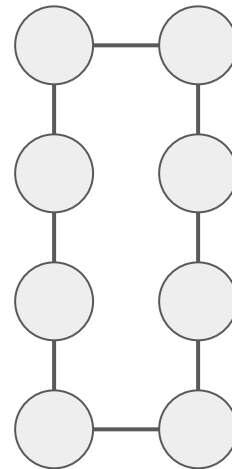
# DropGNN

## Example:

Here is an example where the WL-test cannot distinguish between Graph A and B.



**A**



**B**

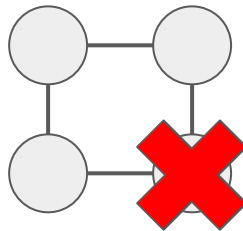
# DropGNN

## Example:

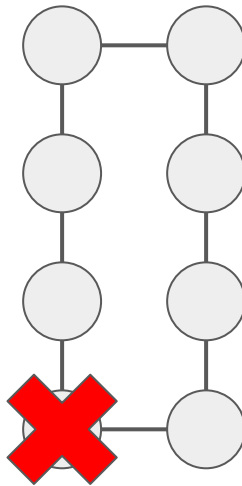
Here is an example where the WL-test cannot distinguish between Graph **A** and **B**.

Using DropGNN, with a neighborhood depth of 2 we can differentiate.

In each 1-dropout, A will always produce a graph with zero 2-hop neighbors



**A**



**B**

# DropGNN - Picking $r$ (the optimal number of runs)

For a specific neighborhood  $\Gamma$  around node  $u$  (region of interest) we want to ensure that we adequately sample from the dropout combinations.

Therefore the choice of  $\Gamma$  then determines our ideal  $p$  and  $r$ . Where  $p$  is the probability that a specific node is dropped and  $r$  is the number of runs.

The size of our dropout neighborhood  $\gamma = |\Gamma|$  does not necessarily scale with the graph. It is possible for  $\Gamma$  to be less than the GNN neighborhood.

For example:  $\Gamma$  may be a 2-hop neighborhood whereas our GNN may be a 5-hop neighborhood.

# DropGNN - Picking $r$ for 1-complete dropout

**1-complete case:** We want to have enough runs to ensure that at least every 1-dropout is observed a few times.

For any specific node  $v \in \Gamma$ , the probability of a 1-dropout for  $v$  is (including the probability that  $u$  is not dropped out):

$$p \cdot (1 - p)^\gamma$$

Therefore we want to maximize this probability, to find the max we differentiate:

$$p^* = \frac{1}{1 + \gamma}$$

## DropGNN - Picking $r$ for 1-complete dropout

From our dropout probability  $p \cdot (1 - p)^\gamma$  plug in  $p^* = \frac{1}{1 + \gamma}$

$$\frac{1}{1 + \gamma} \cdot \left( \frac{\gamma}{1 + \gamma} \right)^\gamma$$

We can constrain this probability:

$$\frac{1}{1 + \gamma} \cdot \left( \frac{\gamma}{1 + \gamma} \right)^\gamma \geq \frac{1}{1 + \gamma} \cdot \frac{1}{e}$$

Hence, if we execute  $r \geq e \cdot (\gamma + 1)$  runs, then the expected number of times we observe a specific 1-dropout is at least  $r \cdot \frac{1}{e} \cdot \frac{1}{1 + \gamma} \geq 1$ .

# DropGNN - Results

Dataset	GIN		+Ports		+IDs		+Random feat.		+Dropout	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
LIMITS 1 [11]	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	1.00 $\pm$ 0.00	0.59 $\pm$ 0.19	0.66 $\pm$ 0.19	0.66 $\pm$ 0.22	1.00 $\pm$ 0.00	<b>1.00 <math>\pm</math>0.00</b>
LIMITS 2 [11]	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	1.00 $\pm$ 0.00	0.61 $\pm$ 0.26	0.72 $\pm$ 0.17	0.64 $\pm$ 0.19	1.00 $\pm$ 0.00	<b>1.00 <math>\pm</math>0.00</b>
4-CYCLES [20]	0.50 $\pm$ 0.00	0.50 $\pm$ 0.00	1.00 $\pm$ 0.01	0.84 $\pm$ 0.07	1.00 $\pm$ 0.00	0.58 $\pm$ 0.07	0.75 $\pm$ 0.05	0.77 $\pm$ 0.05	0.99 $\pm$ 0.03	<b>1.00 <math>\pm</math>0.01</b>
LCC [29]	0.41 $\pm$ 0.09	0.38 $\pm$ 0.08	1.0 $\pm$ 0.00	0.39 $\pm$ 0.09	1.00 $\pm$ 0.00	0.42 $\pm$ 0.08	0.45 $\pm$ 0.16	0.46 $\pm$ 0.08	1.00 $\pm$ 0.00	<b>0.99 <math>\pm</math>0.02</b>
TRIANGLES [29]	0.53 $\pm$ 0.15	0.52 $\pm$ 0.15	1.0 $\pm$ 0.00	0.54 $\pm$ 0.11	1.00 $\pm$ 0.00	0.63 $\pm$ 0.08	0.57 $\pm$ 0.08	0.67 $\pm$ 0.05	0.93 $\pm$ 0.12	<b>0.93 <math>\pm</math>0.13</b>
SKIP-CIRCLES [6]	0.10 $\pm$ 0.00	0.10 $\pm$ 0.00	1.00 $\pm$ 0.00	0.14 $\pm$ 0.08	1.00 $\pm$ 0.00	0.10 $\pm$ 0.09	0.16 $\pm$ 0.11	0.16 $\pm$ 0.05	0.81 $\pm$ 0.28	<b>0.82 <math>\pm</math>0.28</b>

DropGNN was compared against 4 other GNN methods on 6 synthetically generated datasets.

- Limits 1 and 2: Compares smaller structures to larger structure to differentiate between them
- 4-cycles: Classifies graphs that are part of a cycle and of length 4
- LCC: Predicts Local Clustering Coefficient
- Triangles: the model must predict whether a node is part of a triangle
- Skip-Circles: The model needs to classify if a given circular graph has skip links of length {2, 3, 4, 5, 6, 9, 11, 12, 13, 16}.