

Stanford CS224W: **Advanced Topics in** **Graph Neural Networks**

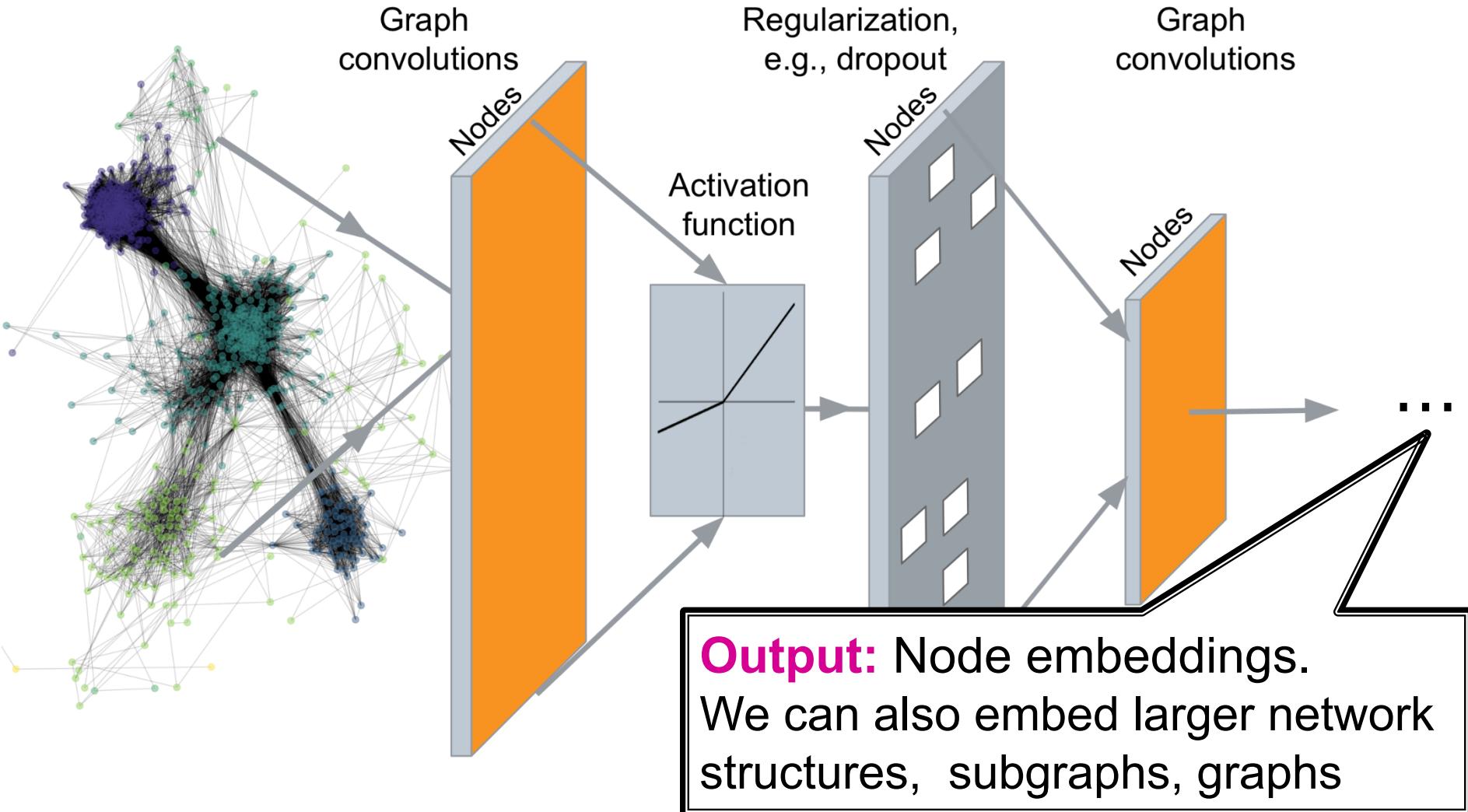
CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

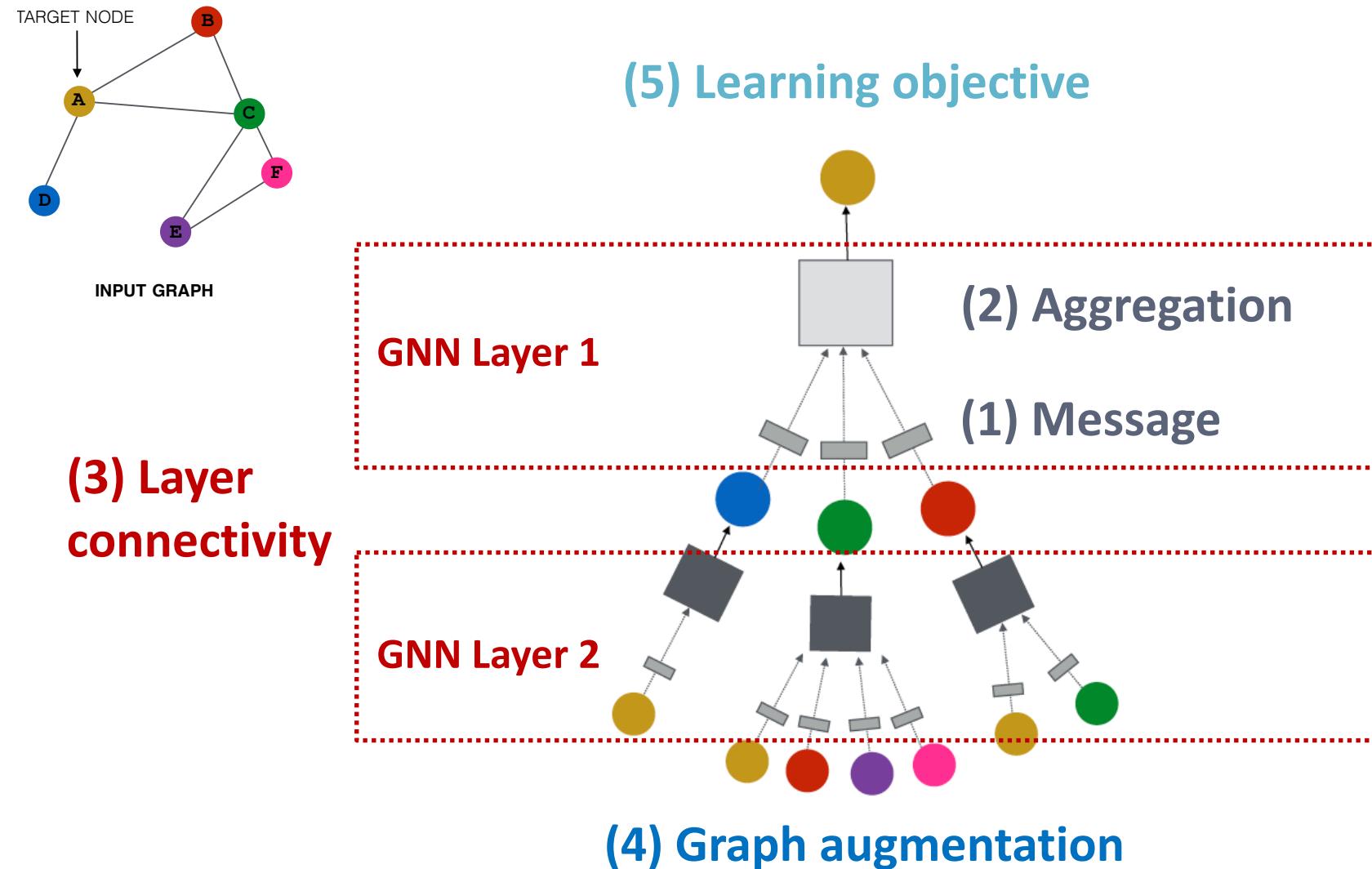
<http://cs224w.stanford.edu>



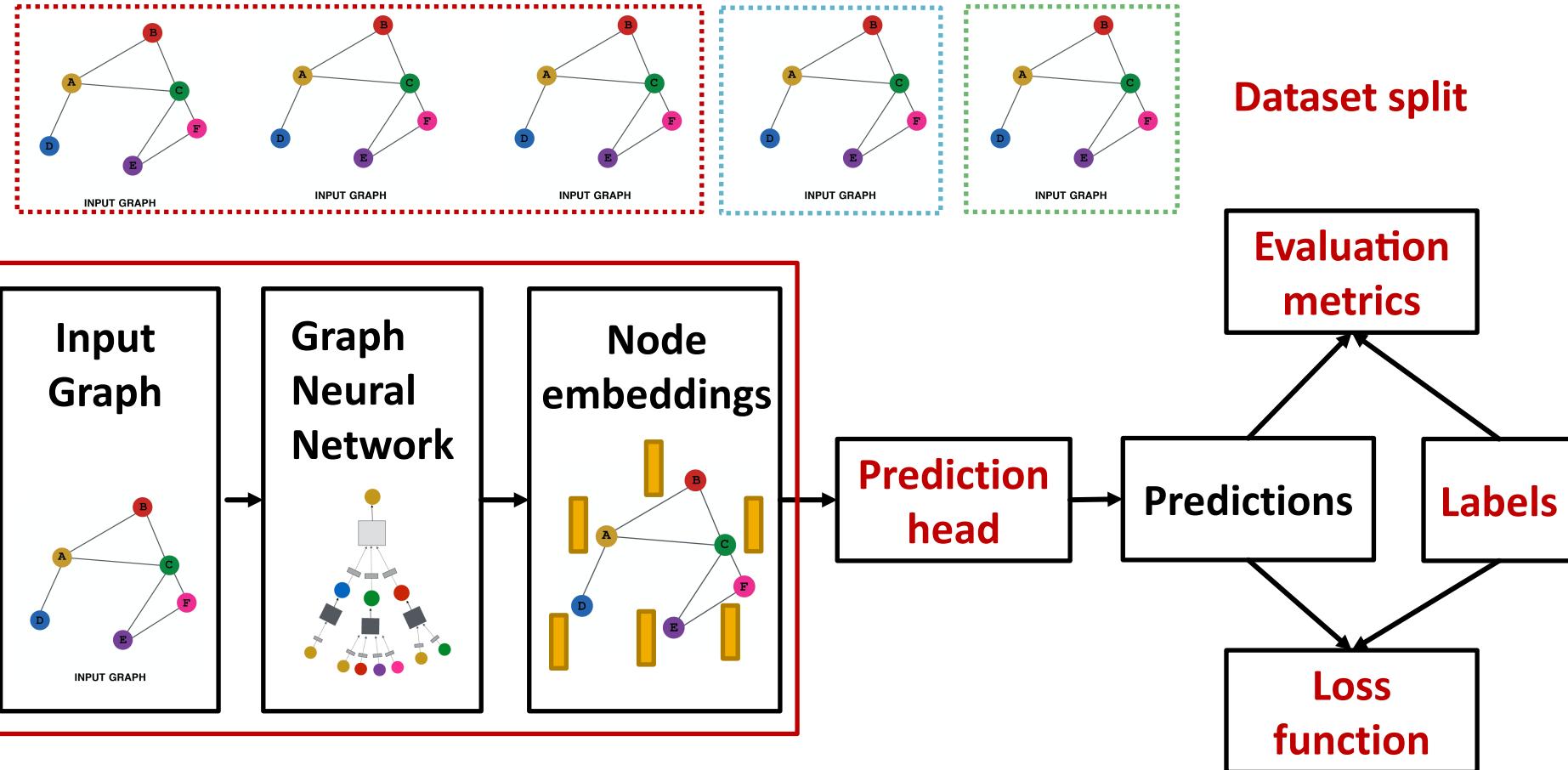
Recap: Graph Neural Networks



Recap: A General GNN Framework



Recap: GNN Training Pipeline



Today's lecture: Can we make GNN representation more expressive?

Stanford CS224W: Limitations of Graph Neural Networks

CS224W: Machine Learning with Graphs

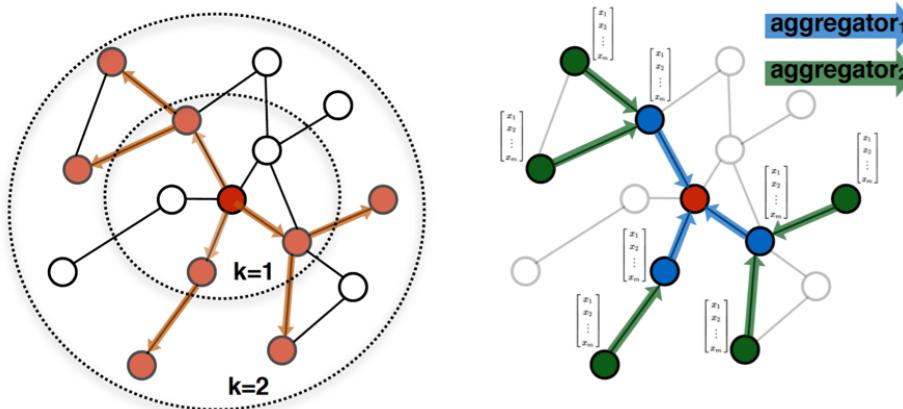
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



A “Perfect” GNN Model

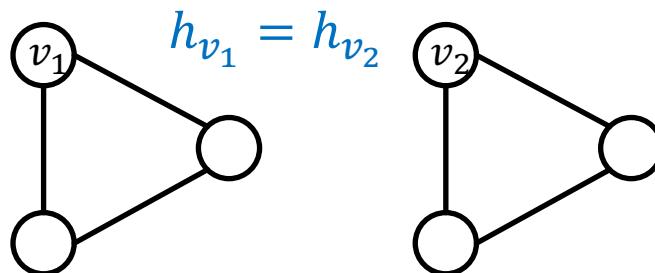
- A thought experiment: What should a perfect GNN do?
 - A k -layer GNN embeds a node based on the K -hop neighborhood structure



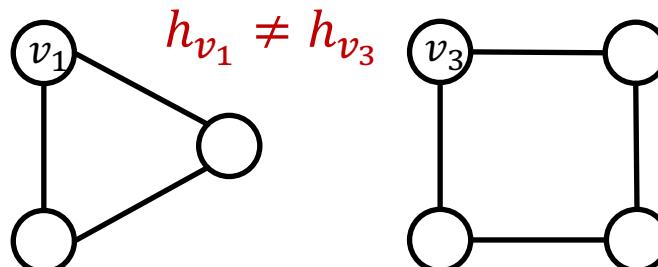
- A perfect GNN should build an **injective function** between **neighborhood structure** (regardless of hops) and **node embeddings**

A “Perfect” GNN Model

- Therefore, for a perfect GNN:
 - **Observation 1:** If two nodes have the same neighborhood structure, they must have the same embedding

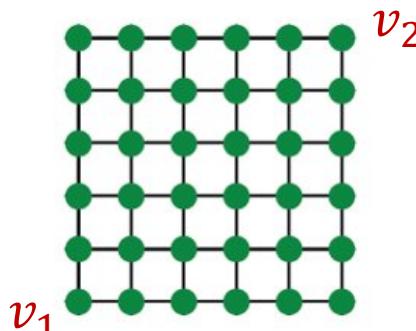


- **Observation 2:** If two nodes have different neighborhood structure, they must have different embeddings



Imperfections of Existing GNNs

- However, Observations 1 & 2 are imperfect
- Observation 1 could have issues:
 - Even though two nodes may have the same neighborhood structure, we may want to assign different embeddings to them
 - Because these nodes appear in different positions in the graph
 - We call these tasks Position-aware tasks
 - Even a perfect GNN will fail for these tasks:



A grid graph

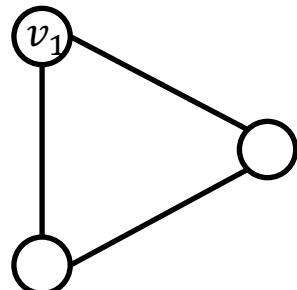


NYC road network

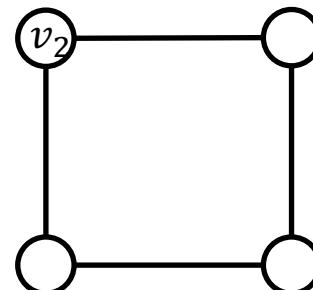
Imperfections of Existing GNNs

- **Observation 2 often cannot be satisfied:**
 - The GNNs we have introduced so far are not perfect
 - In Lecture 9, we discussed that their expressive power is **upper bounded by the WL test**
 - For example, message passing GNNs **cannot count cycle length**:

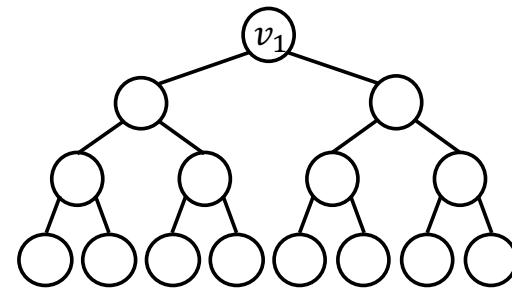
v_1 resides in a cycle with length 3



v_2 resides in a cycle with length 4



The computational graphs for nodes v_1 and v_2 are always the same



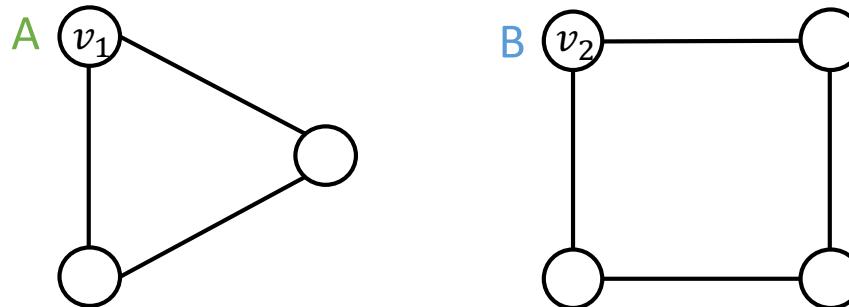
Plan for the Lecture

- We will resolve both issues by **building more expressive GNNs**
- **Fix issues in Observation 1:**
 - Create node embeddings based on their positions in the graph
 - Example method: Position-aware GNNs
- **Fix issues in Observation 2:**
 - Build message passing GNNs that are more expressive than WL test
 - Example method: Identity-aware GNNs

Our Approach

- We use the following thinking:

- Two different inputs (nodes, edges, graphs) are labeled differently
- A “failed” model will always assign the same embedding to them
- A “successful” model will assign different embeddings to them
- Embeddings are determined by GNN computational graphs:



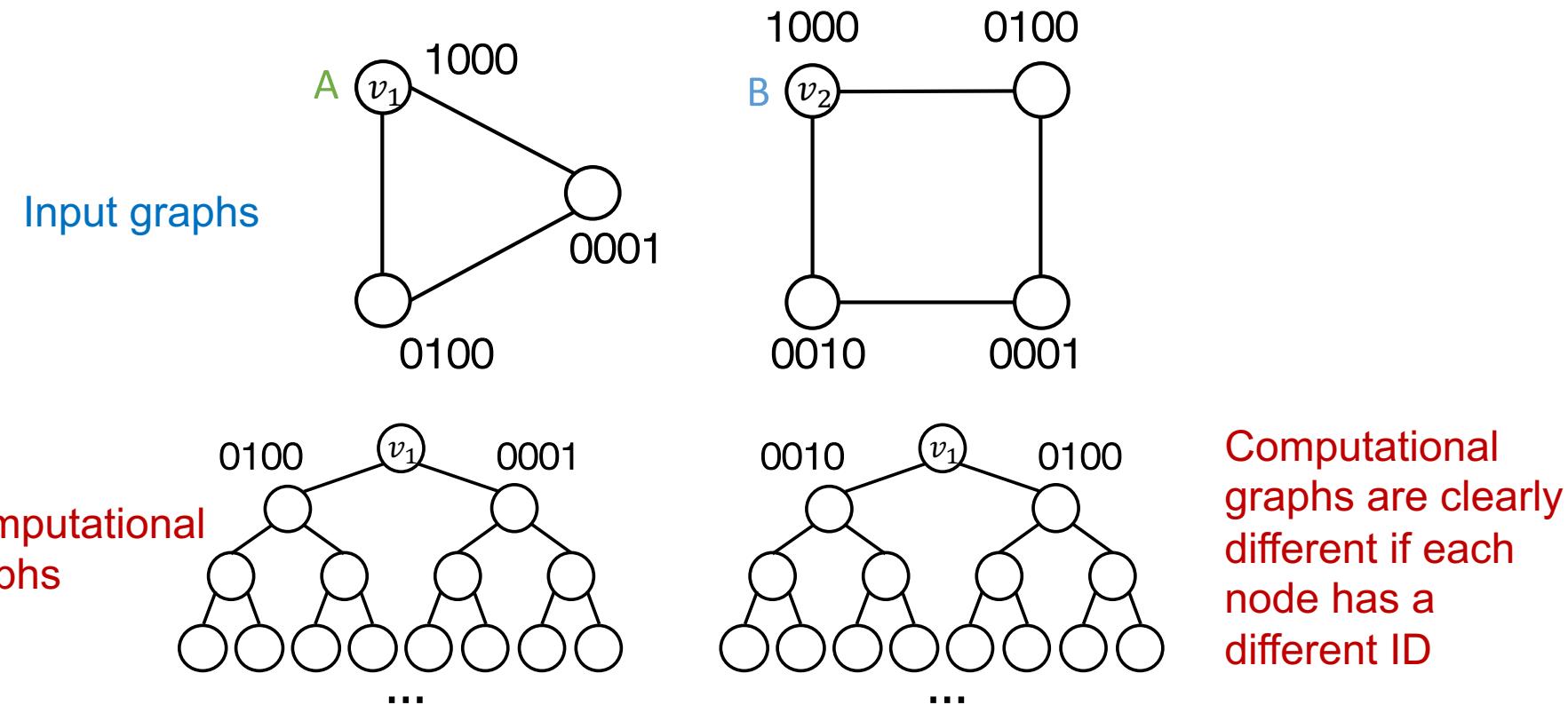
Two inputs: nodes v_1 and v_2

Different labels: A and B

Goal: assign different embeddings to v_1 and v_2

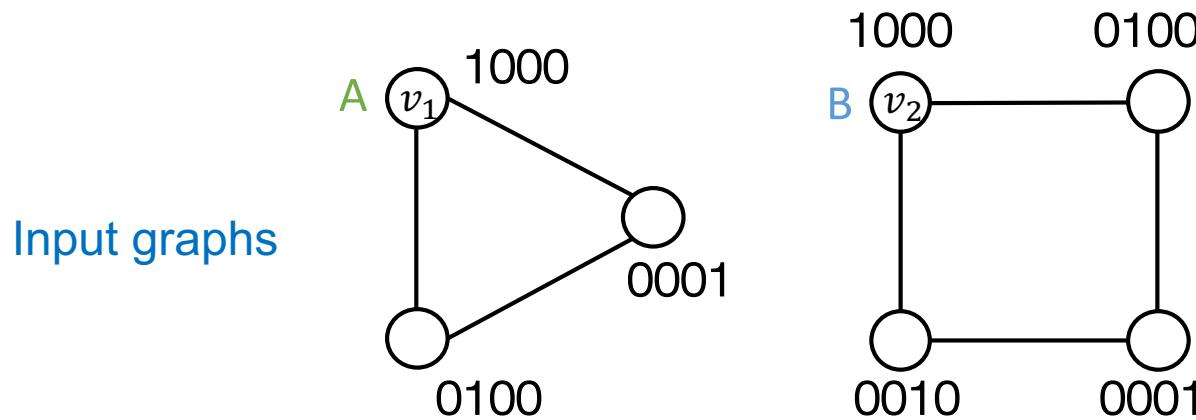
Naïve Solution is not Desirable

- A naïve solution: One-hot encoding
 - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs



Naïve Solution is not Desirable

- A naïve solution: One-hot encoding
 - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs



- Issues:
 - Not scalable: Need $O(N)$ feature dimensions (N is the number of nodes)
 - Not inductive: Cannot generalize to new nodes/graphs

Stanford CS224W: Position-aware Graph Neural Networks

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

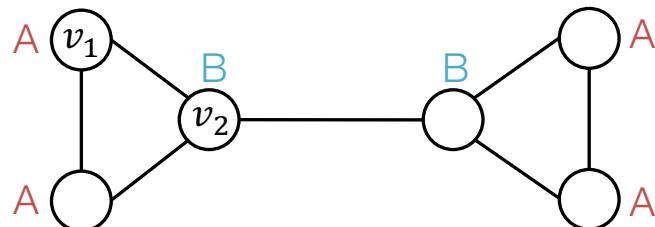
<http://cs224w.stanford.edu>



Two Types of Tasks on Graphs

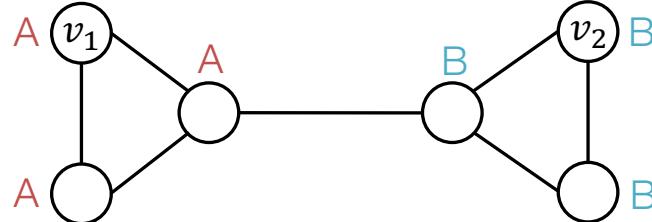
- There are two types of tasks on graphs

Structure-aware task



- Nodes are labeled by their **structural roles** in the graph

Position-aware task

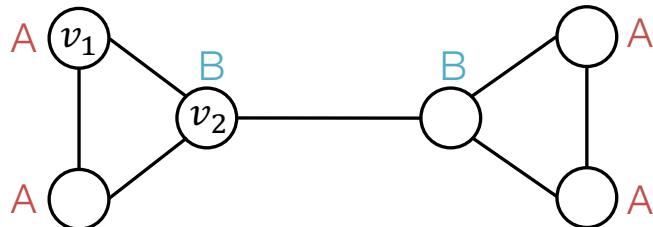


- Nodes are labeled by their **positions** in the graph

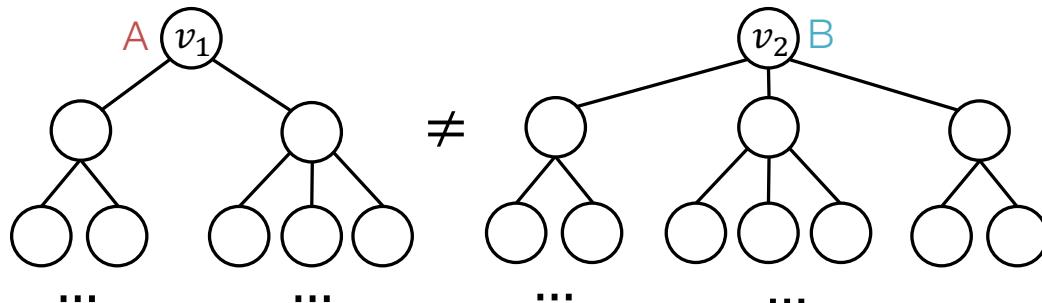
Structure-aware Tasks

- GNNs often work well for structure-aware tasks

Structure-aware task



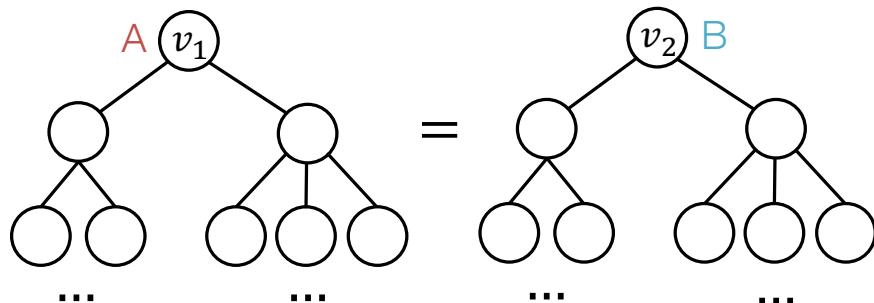
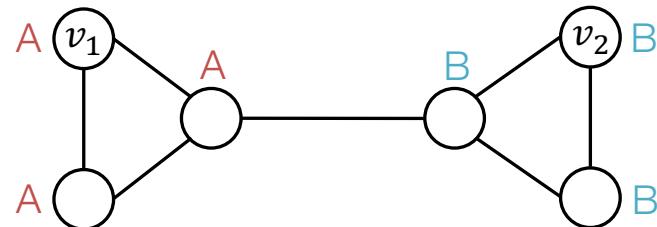
- GNNs work 😊
- Can differentiate v_1 and v_2 by using different computational graphs



Position-aware Tasks

- GNNs will always fail for position-aware tasks

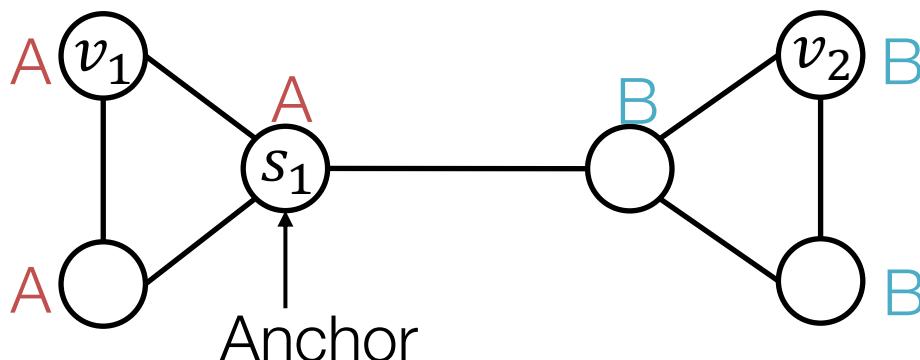
Position-aware task



- GNNs fail 😞
- v_1 and v_2 will always have the same computational graph, due to structure symmetry
- Can we define deep learning methods that are position-aware?

Power of “Anchor”

- Randomly pick a node s_1 as an **anchor node**
- Represent v_1 and v_2 via their relative distances w.r.t. the anchor s_1 , **which are different**
- An anchor node serves as **a coordinate axis**
 - Which can be used to **locate nodes in the graph**

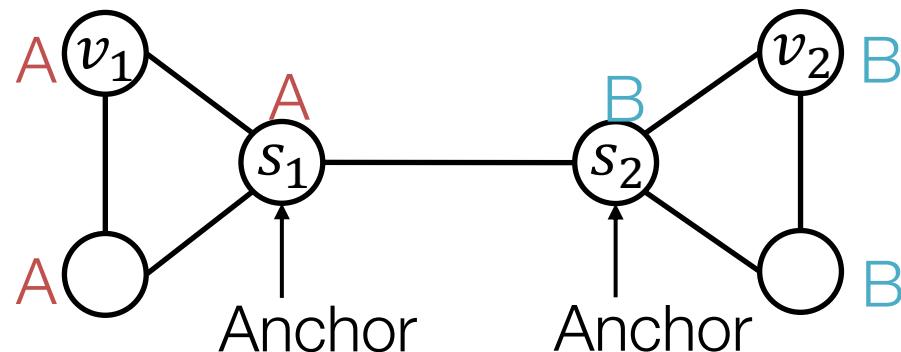


Relative
Distances

	s_1
v_1	1
v_2	2

Power of “Anchors”

- Pick more nodes s_1, s_2 as **anchor nodes**
- **Observation:** More anchors can better characterize node position in different regions of the graph
- Many anchors → Many coordinate axes

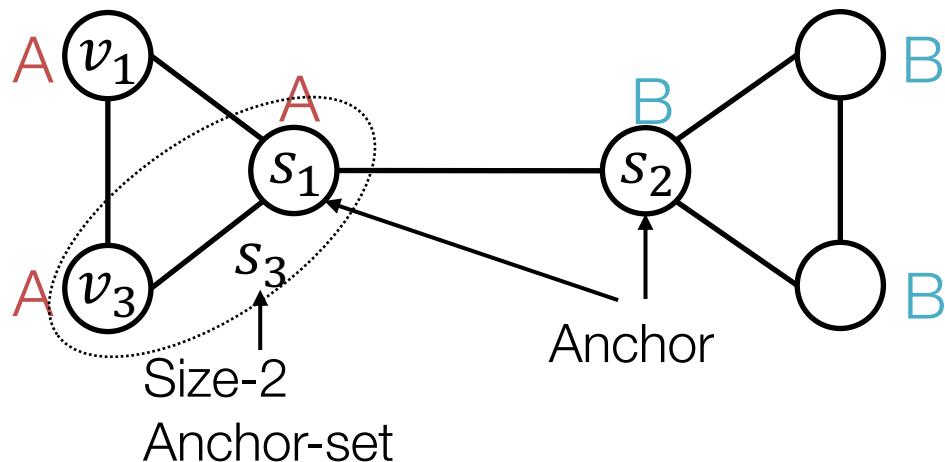


Relative
Distances

	s_1	s_2
v_1	1	2
v_2	2	1

Power of “Anchor-sets”

- Generalize anchor from a single node to a **set of nodes**
 - We define distance to an anchor-set as the minimum distance to all the nodes in the anchor-set
- **Observation:** Large anchor-sets can sometimes provide more precise position estimate
 - We can save the total number of anchors



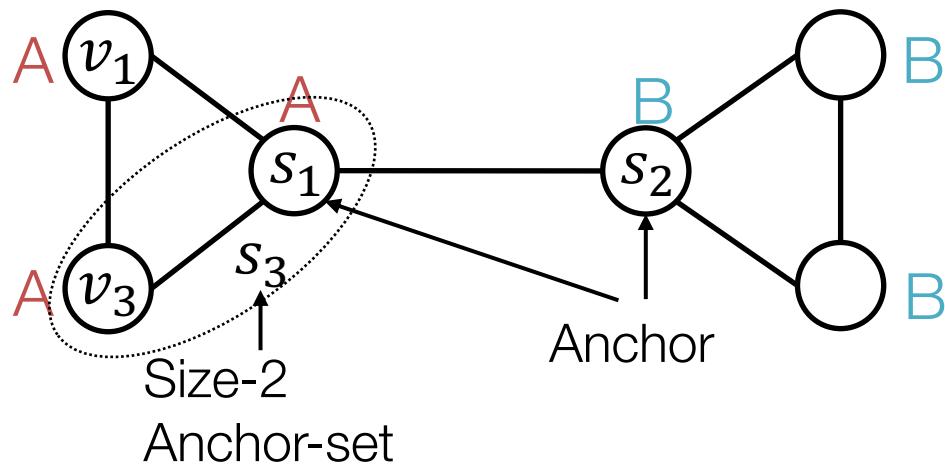
Relative Distances

	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

Anchor s_1, s_2 cannot differentiate node v_1, v_3 , but anchor-set s_3 can

Position Information: Summary

- **Position encoding for graphs:** Represent a node's position by its distance to randomly selected anchor-sets
 - Each dimension of the position encoding is tied to an anchor-set



	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

v_1 's Position
encoding

v_3 's Position
encoding

How to Use Position Information

- **The simple way:** Use position encoding as an **augmented node feature** (works well in practice)
 - **Issue:** since each dimension of position encoding is tied to a random anchor, **dimensions of positional encoding can be randomly permuted, without changing its meaning**
 - Imagine you permute the input dimensions of a normal NN, the output will surely change

How to Use Position Information

- **The rigorous solution:** requires a special NN that can maintain the **permutation invariant property of position encoding**
 - Permuting the input feature dimension will **only result in the permutation of the output dimension**, the value in each dimension won't change
 - Refer to the Position-aware GNN paper for more details

Stanford CS224W: Identity-Aware Graph Neural Networks

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



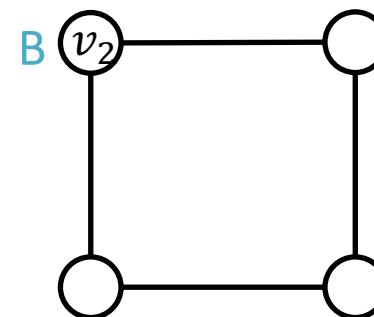
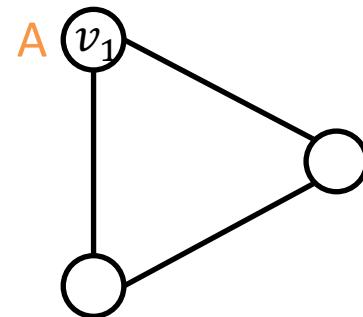
More Failure Cases for GNNs

- We learned that **GNNs would fail for position-aware tasks**
- **But can GNN perform perfectly in structure-aware tasks?**
 - Unfortunately, **NO**
- GNNs exhibit three levels of failure cases in structure-aware tasks:
 - Node level
 - Edge level
 - Graph level

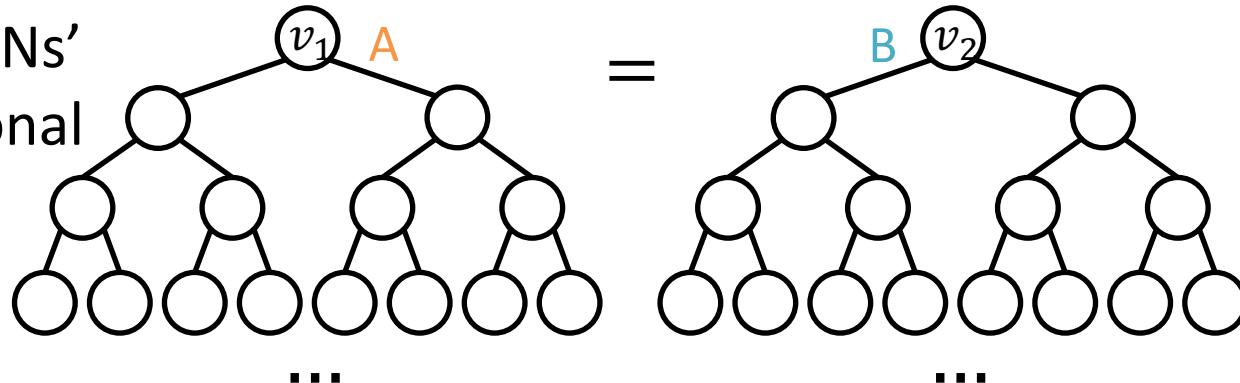
GNN Failure 1: Node-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input
graphs



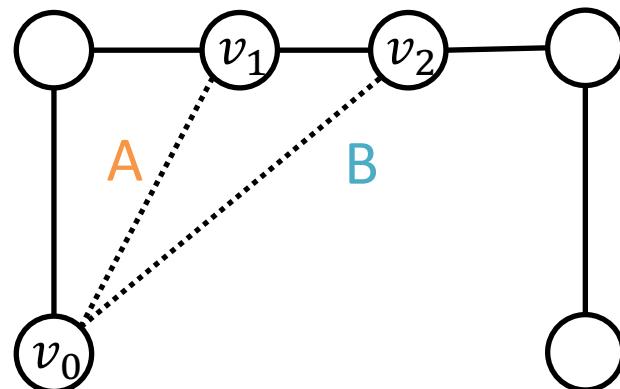
Existing GNNs'
computational
graphs



GNN Failure 2: Edge-level Tasks

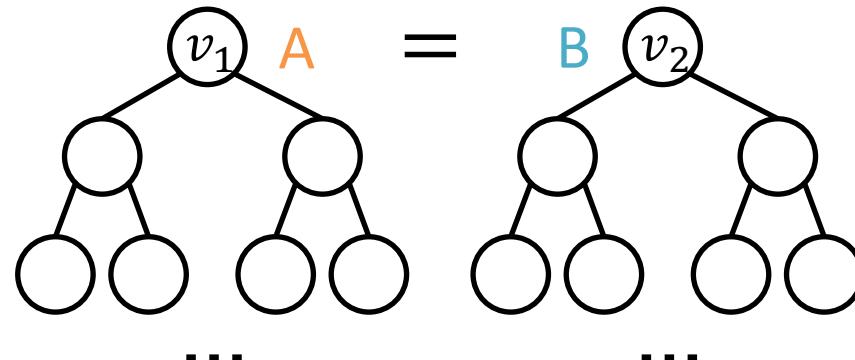
Different Inputs but the same computational graph → GNN fails

Example input graphs



Edge A and B share node v_0
We look at embeddings for v_1 and v_2

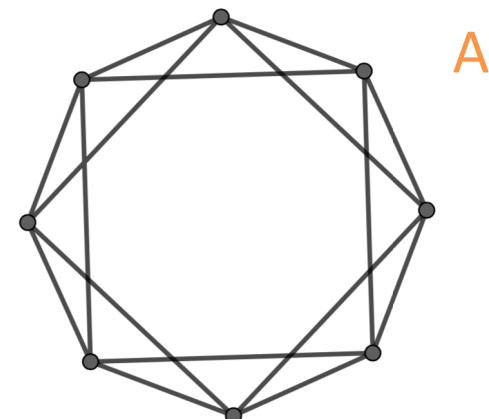
Existing GNNs' computational graphs



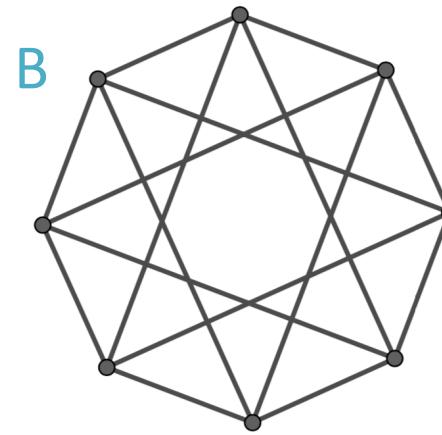
GNN Failure 3: Graph-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input graphs



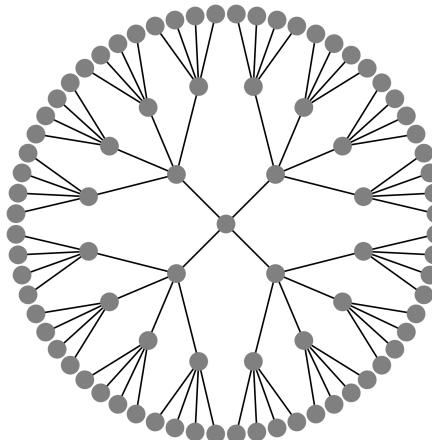
A



B

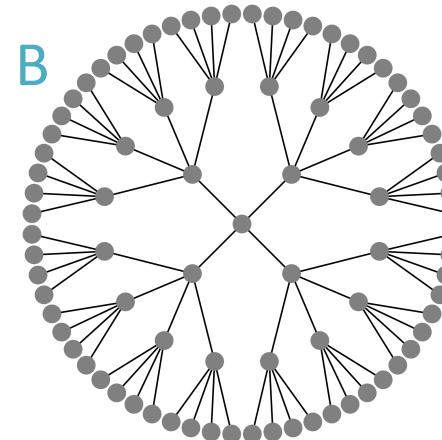
We look at embeddings for each node

For each node:



A

=



For each node:

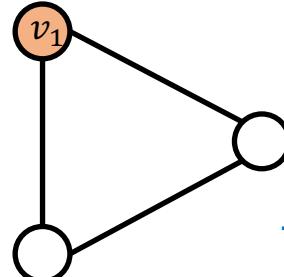
Existing GNNs' computational graphs

Idea: Inductive Node Coloring

- Idea: We can assign a color to the node we want to embed

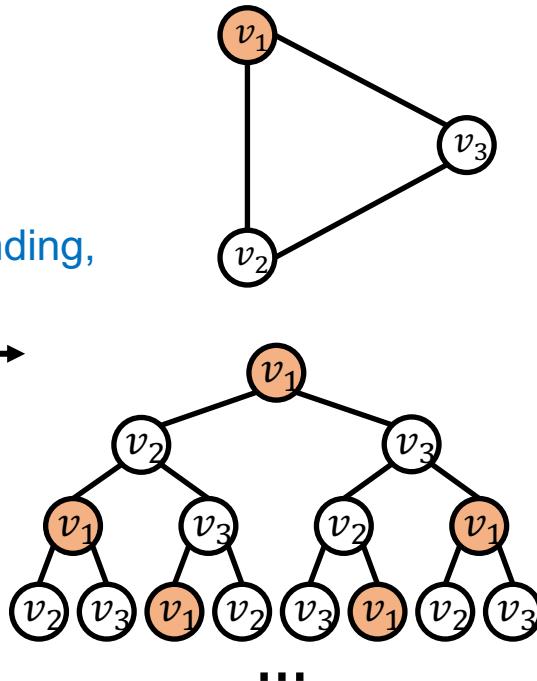
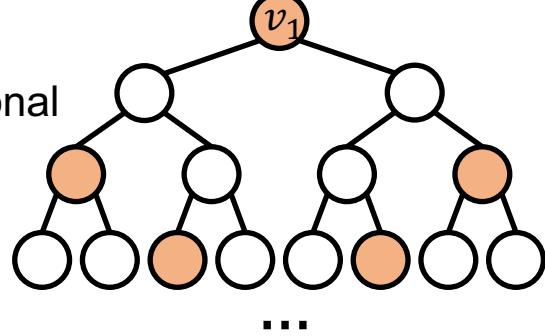
- The node we want to embed
- The rest of nodes

Input graph



To assist understanding,
we label the nodes

Computational
graph

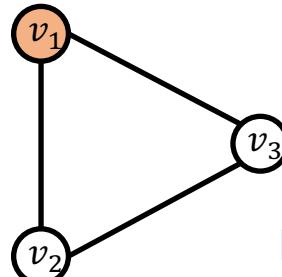


Idea: Inductive Node Coloring

- This coloring is **inductive**:
 - It is invariant to node ordering/identities

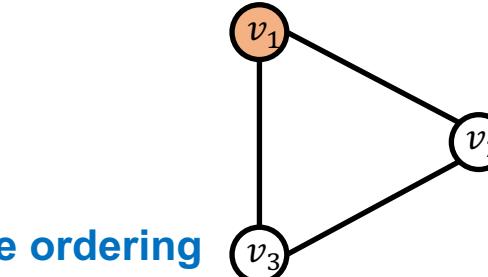
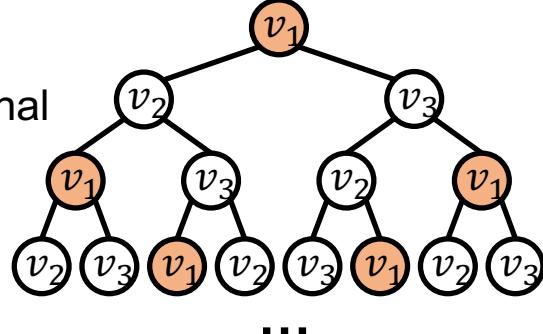
- The node we want to embed
- The rest of nodes

Input graph



Permute the node ordering
between v_2 and v_3

Computational
graph

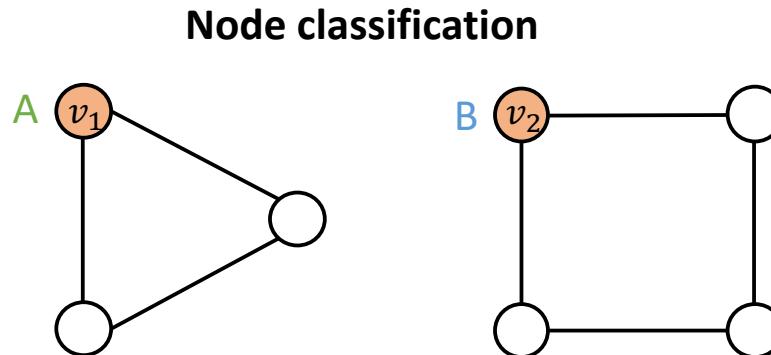


The computational graph stays the same

Inductive Node Coloring – Node level

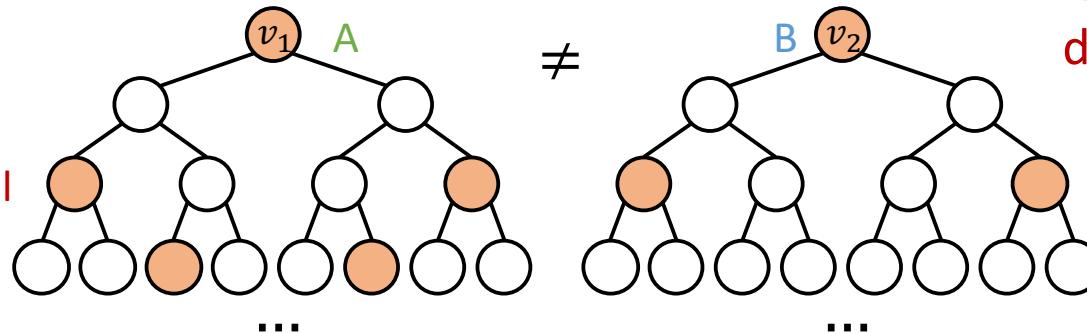
- Inductive node coloring can help **node classification**

Example input graphs



Different computational graphs
→ Successfully differentiate nodes

ID-GNNs' computational graphs



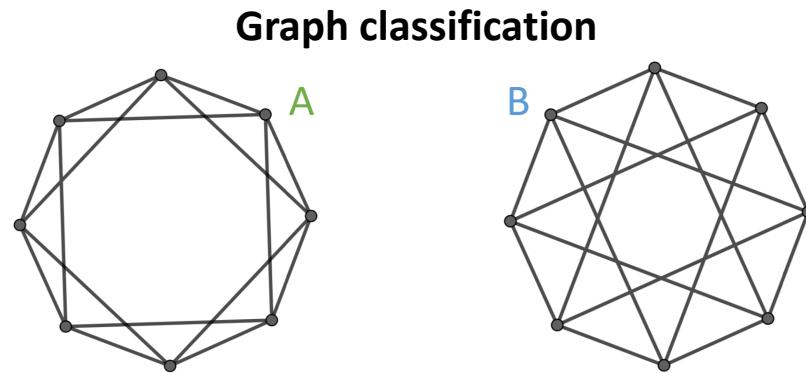
Two types of nodes:

- node with augmented identity (orange circle)
- node without augmented identity (white circle)

Inductive Node Coloring – Graph Level

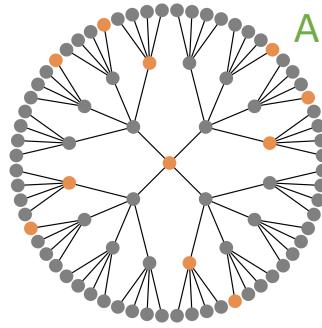
- Inductive node coloring can help **graph classification**

Example input graphs



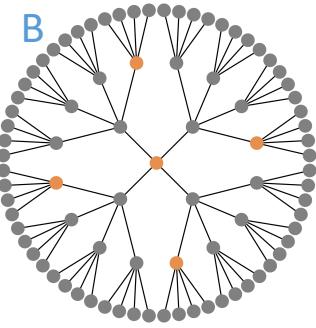
We color root nodes with identity

For each node:



≠

For each node:



Different computational graphs
→ Successful differentiate graphs

ID-GNNs' computational graphs

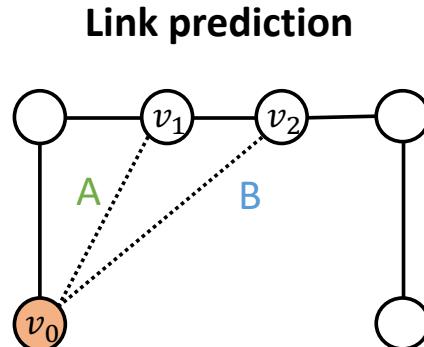
Two types of nodes:

- node with augmented identity
- node without augmented identity

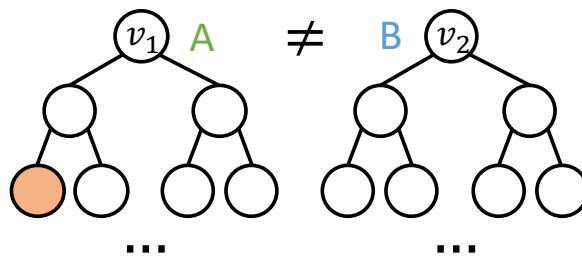
Inductive Node Coloring – Edge Level

- Inductive node coloring can help **link prediction**

Example input graphs



ID-GNNs'
computational
graphs



Two types of nodes:

- node with augmented identity
- node without augmented identity

An edge-level task involves classifying a pair of nodes:

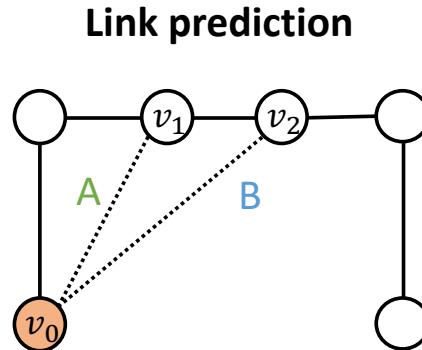
- We color one of the node (v_0)
- We then embed the other node in the node pair (v_1 or v_2)
- We use the node embedding for v_1 or v_2 conditioned on v_0 being colored or not to make edge-level prediction

Different
computational graphs
→ Successfully
differentiate edges

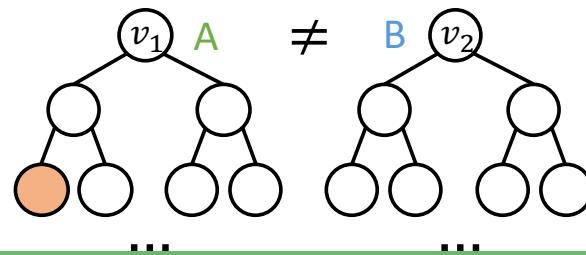
Inductive Node Coloring – Edge Level

- Inductive node coloring can help **link prediction**

Example input graphs



ID-GNNs'
computational
graphs



An edge-level task involves classifying a pair of nodes:

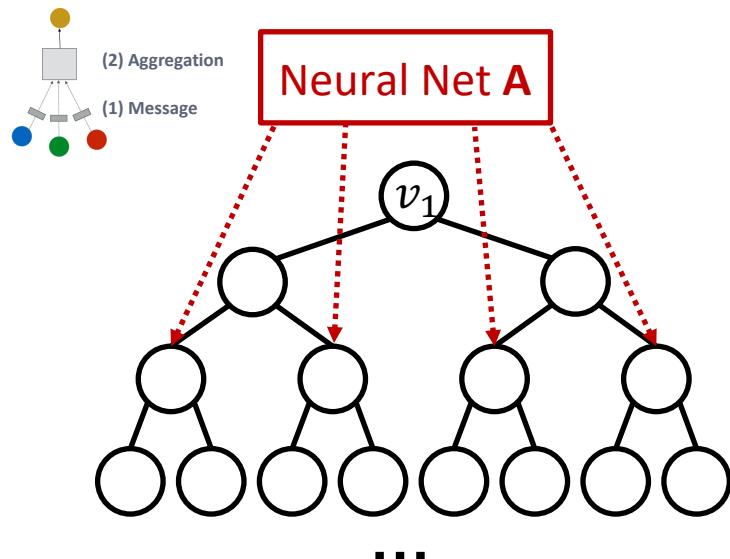
- We color one of the nodes (v_0)
- We then embed the other node in the node pair (v_1 or v_2)
- We use the node embedding for v_1 or v_2 conditioned on v_0 being colored or not to make edge-level prediction

Different

Two How to build a GNN using node coloring?

Identity-aware GNN

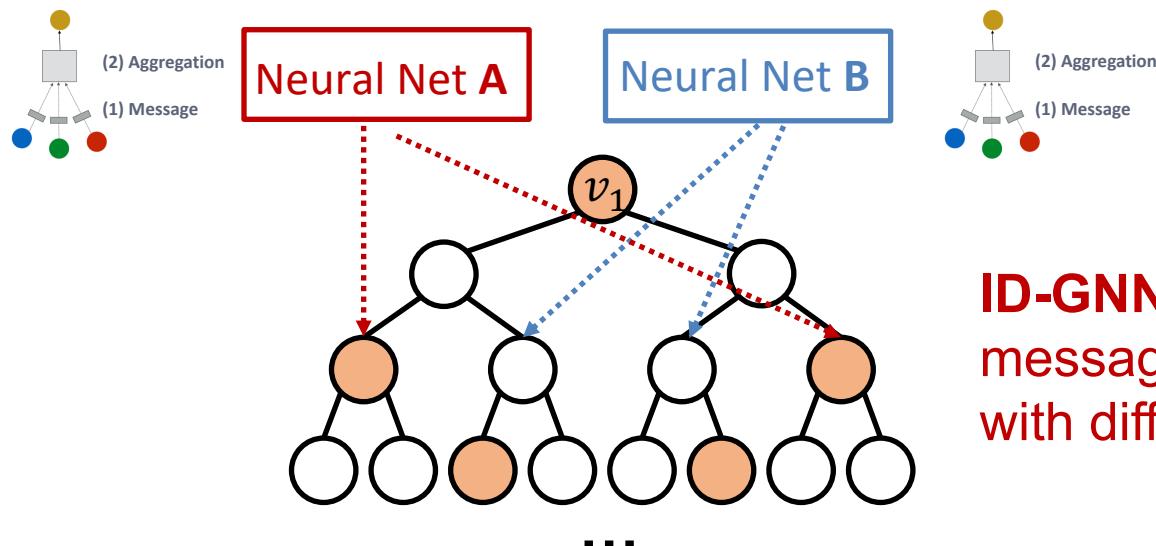
- Utilize **inductive node coloring** in embedding computation
 - Idea: **Heterogenous message passing**
 - Normally, a GNN applies **the same message/aggregation computation to all the nodes**



GNN: At a given layer, we apply the same message/aggregation to each node

Identity-aware GNN

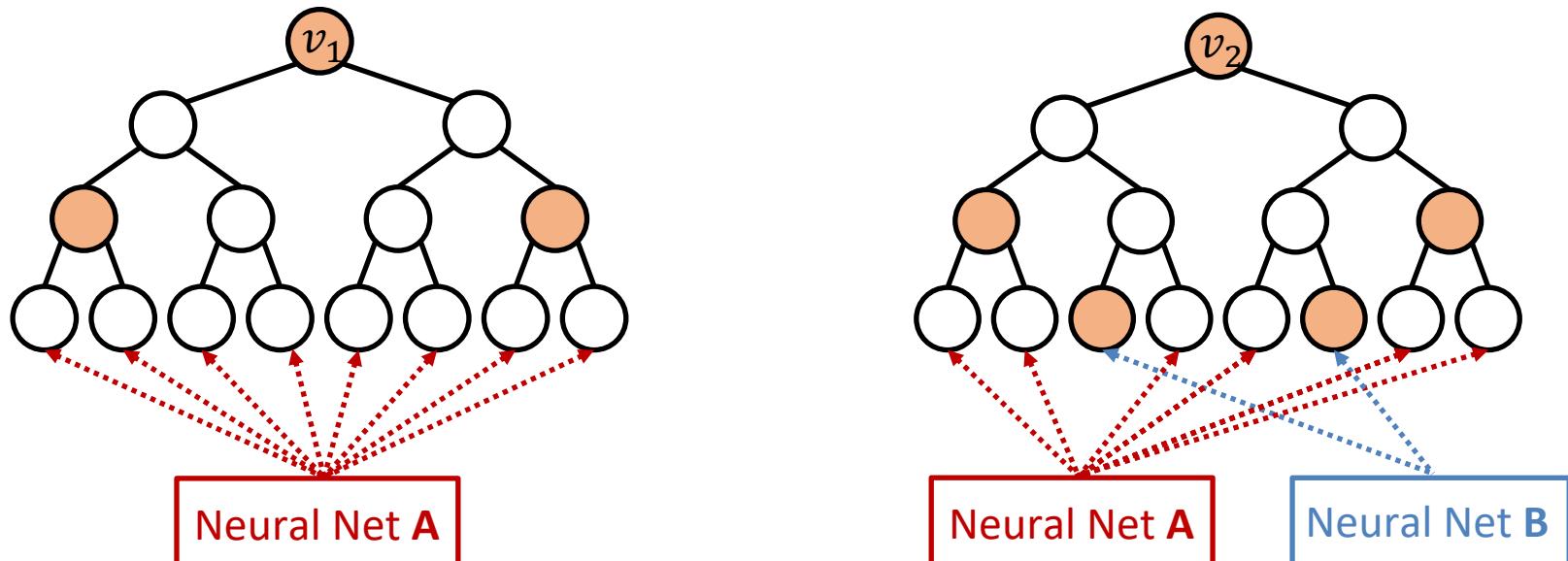
- Idea: **Heterogenous message passing**
 - **Heterogenous:** different types of message passing is applied to different nodes
 - An ID-GNN applies **different message/aggregation to nodes with different colorings**



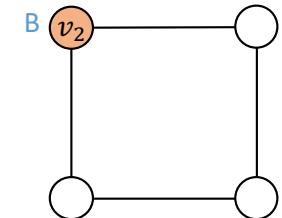
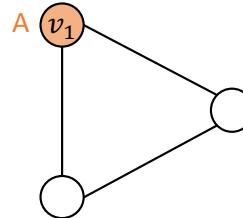
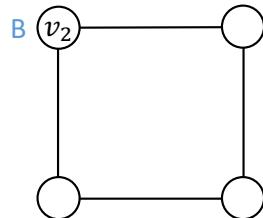
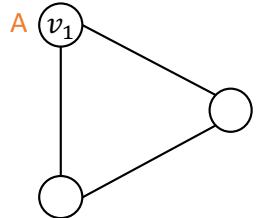
ID-GNN: At a given layer, different message/aggregation to nodes with different colorings

Identity-aware GNN

- Why does heterogenous message passing work:
 - Suppose two nodes v_1, v_2 have the same computational graph structure, but have different node colorings
 - Since we will apply different neural network for embedding computation, their embeddings will be different

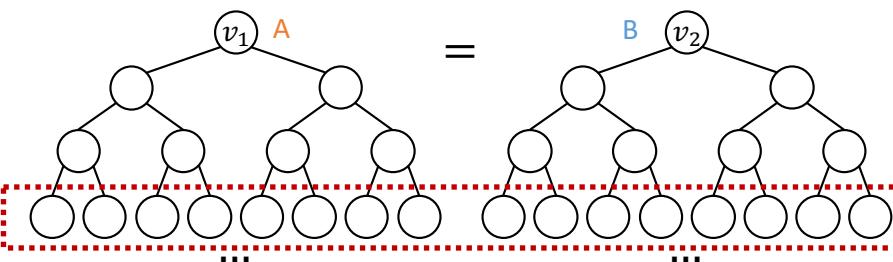


GNN vs ID-GNN

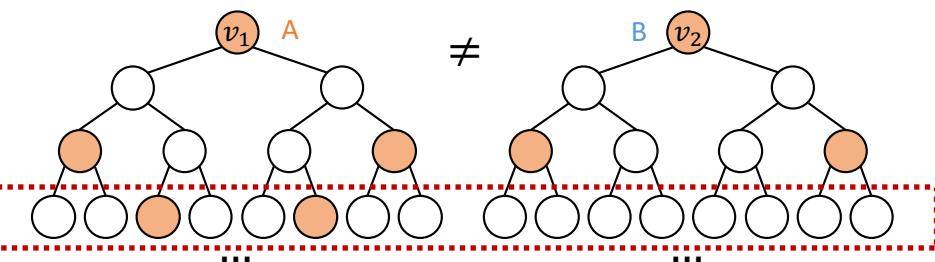


Goal: classify v_1 and v_2

GNN computational graph



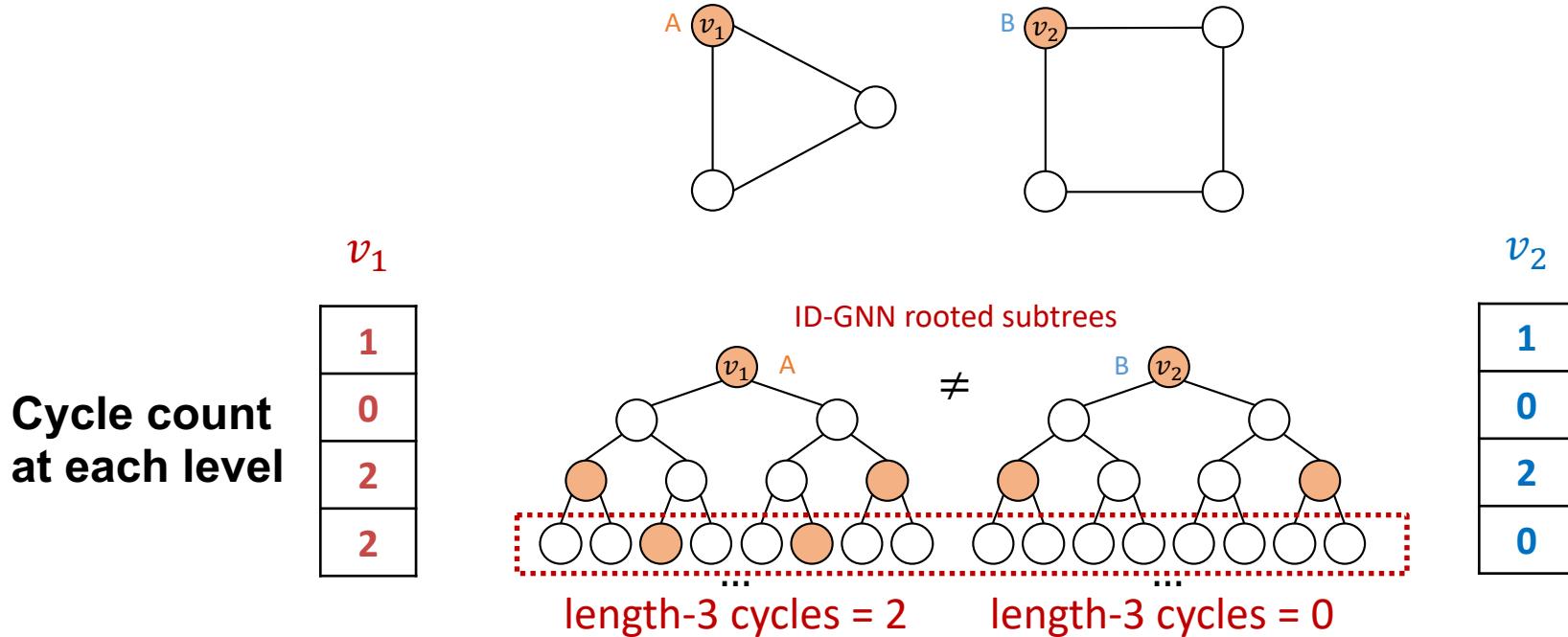
ID-GNN rooted subtrees



From the node coloring, we can tell that:
 v_1 : length-3 cycles = 2 v_2 : length-3 cycles = 0

- Why does ID-GNN work better than GNN?
- Intuition: ID-GNN can count cycles originating from a given node, but GNN cannot

Simplified Version: ID-GNN-Fast



- Based on the intuition, we propose a simplified version **ID-GNN-Fast**
 - Include identity information as an **augmented node feature** (no need to do heterogenous message passing)
 - **Use cycle counts in each layer as an augmented node feature.** Also can be used together with any **GNN**

Identity-aware GNN

- **Summary of ID-GNN: A general and powerful extension to GNN framework**
 - We can apply ID-GNN on **any message passing GNNs** (GCN, GraphSAGE, GIN, ...)
 - ID-GNN provides **consistent performance gain** in node/edge/graph level tasks
 - ID-GNN is **more expressive** than their GNN counterparts. ID-GNN is **the first message passing GNN that is more expressive than 1-WL test**
 - We can **easily implement** ID-GNN using popular GNN tools (PyG, DGL, ...)

Stanford CS224W: **Robustness of** **Graph Neural Networks**

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>

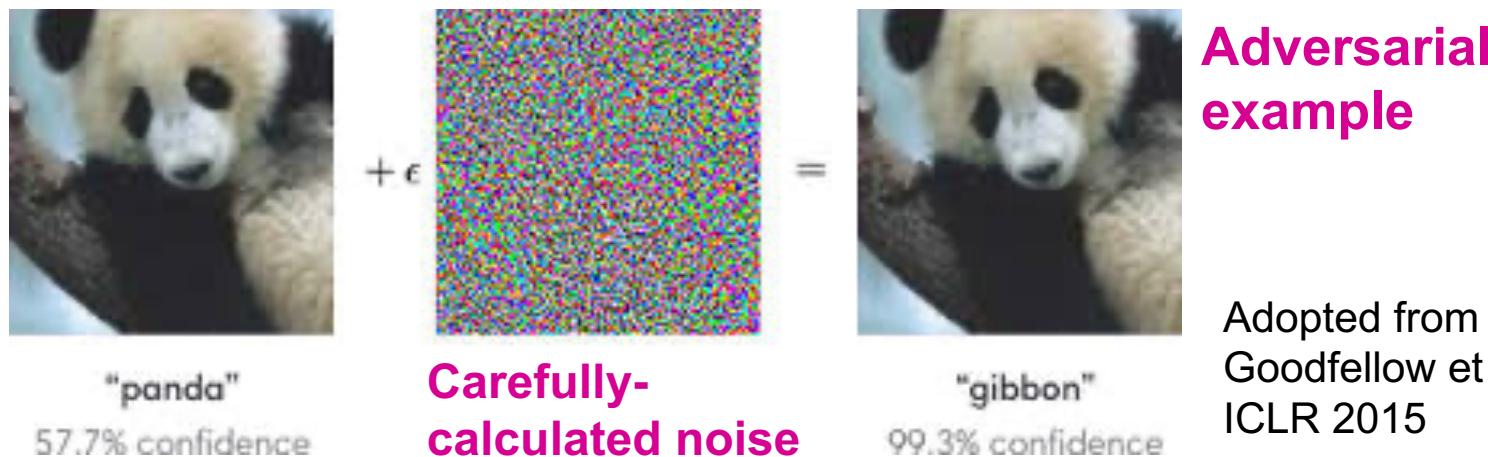


Deep Learning Performance

- Recent years have seen **impressive performance of deep learning models in a variety of applications.**
 - Ex) In computer vision, **deep convolutional networks** have achieved human-level performance on ImageNet (image category classification)
- **Are these models ready to be deployed in real world?**

Adversarial Examples

- Deep convolutional neural networks are vulnerable to **adversarial attacks**:
 - Imperceptible noise changes the prediction.



Adopted from
Goodfellow et al.
ICLR 2015

- Adversarial examples are also reported in natural language processing [Jia & Liang et al. EMNLP 2017] and audio processing [Carlini et al. 2018] domains.

Implication of Adversarial Examples

- **The existence of adversarial examples prevents the reliable deployment of deep learning models to the real world.**
 - Adversaries may try to actively hack the deep learning models.
 - The model performance can become much worse than we expect.
- **Deep learning models are often not robust.**
 - In fact, it is an active area of research to make these models robust against adversarial examples

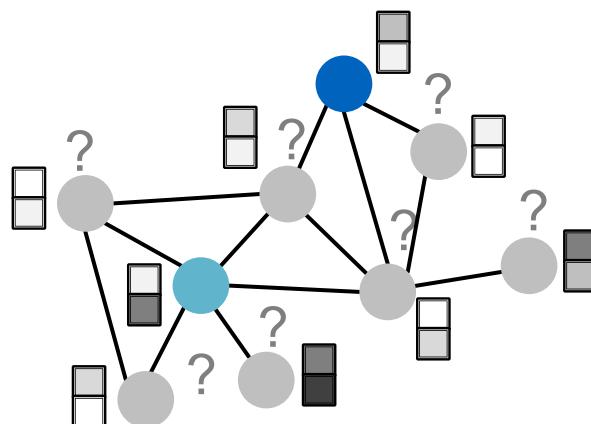
Robustness of GNNs

- **This lecture: How about GNNs? Are they robust to adversarial examples?**
- **Premise:** Common applications of GNNs involve **public platforms** and **monetary interests**.
 - Recommender systems
 - Social networks
 - Search engines
- Adversaries **have the incentive** to manipulate input graphs and hack GNNs' predictions.

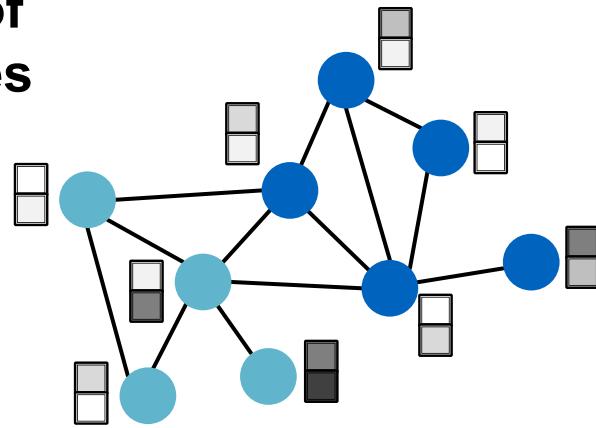
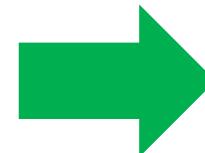
Setting to Study GNNs' Robustness

- To study the robustness of GNNs, we specifically consider the following setting:
 - Task:** Semi-supervised node classification
 - Model:** GCN [Kipf & Welling ICLR 2017]

?: Unlabeled



Predict labels of
unlabeled nodes

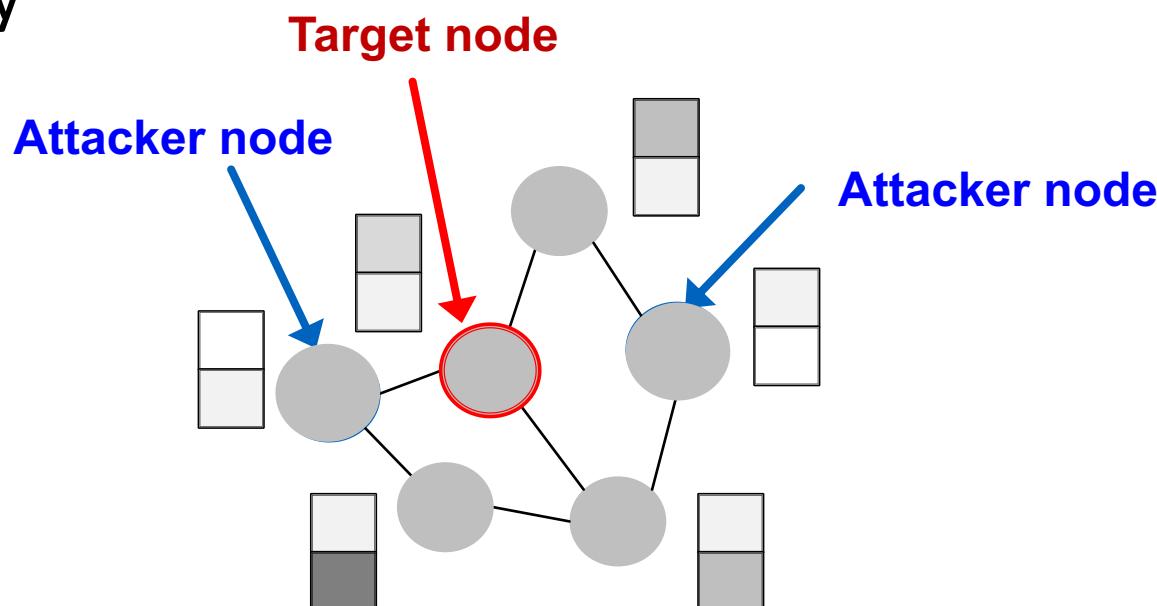


Roadmap

- We first describe several real-world **adversarial attack possibilities.**
- We then review the GCN model that we are going to attack (**knowing the opponent**).
- We mathematically **formalize the attack problem as an optimization problem.**
- **We empirically see how vulnerable GCN's prediction is to the adversarial attack.**

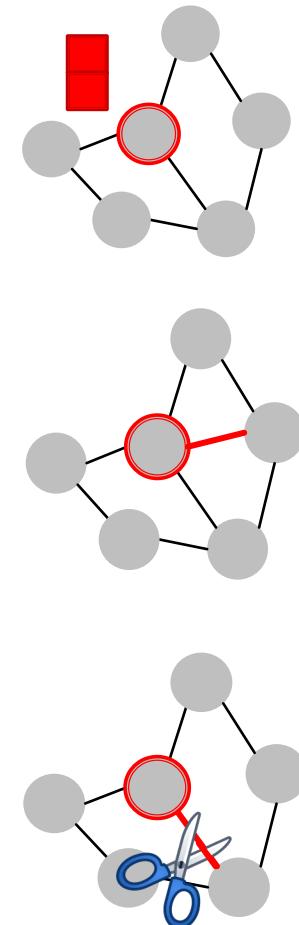
Attack Possibilities

- What are the attack possibilities in real world?
 - **Target node $t \in V$** : node whose label prediction we want to change
 - **Attacker nodes $S \subset V$** : nodes the attacker can modify



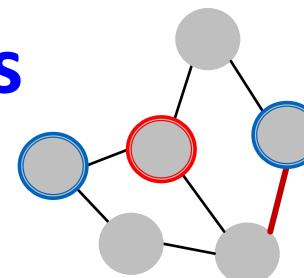
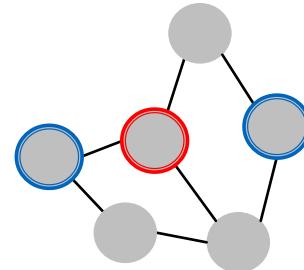
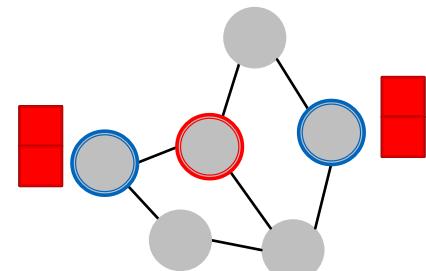
Attack Possibilities: Direct Attack

- **Direct Attack: Attacker node is the target node:** $S = \{t\}$
- Modify **target** node feature
 - Ex) Change website content
- Add connections to **target**
 - Ex) Buy likes/followers
- Remove connections from **target**
 - Ex) Unfollow users



Attack Possibilities: Indirect Attack

- **Indirect Attack:** The **target** node is not in the **attacker** nodes: $t \notin S$
- Modify **attacker** node features
 - Ex) Hijack friends of targets
- Add connections to **attackers**
 - Ex) Create a link, link farm
- Remove connections from **attackers**
 - Ex) Delete undesirable link

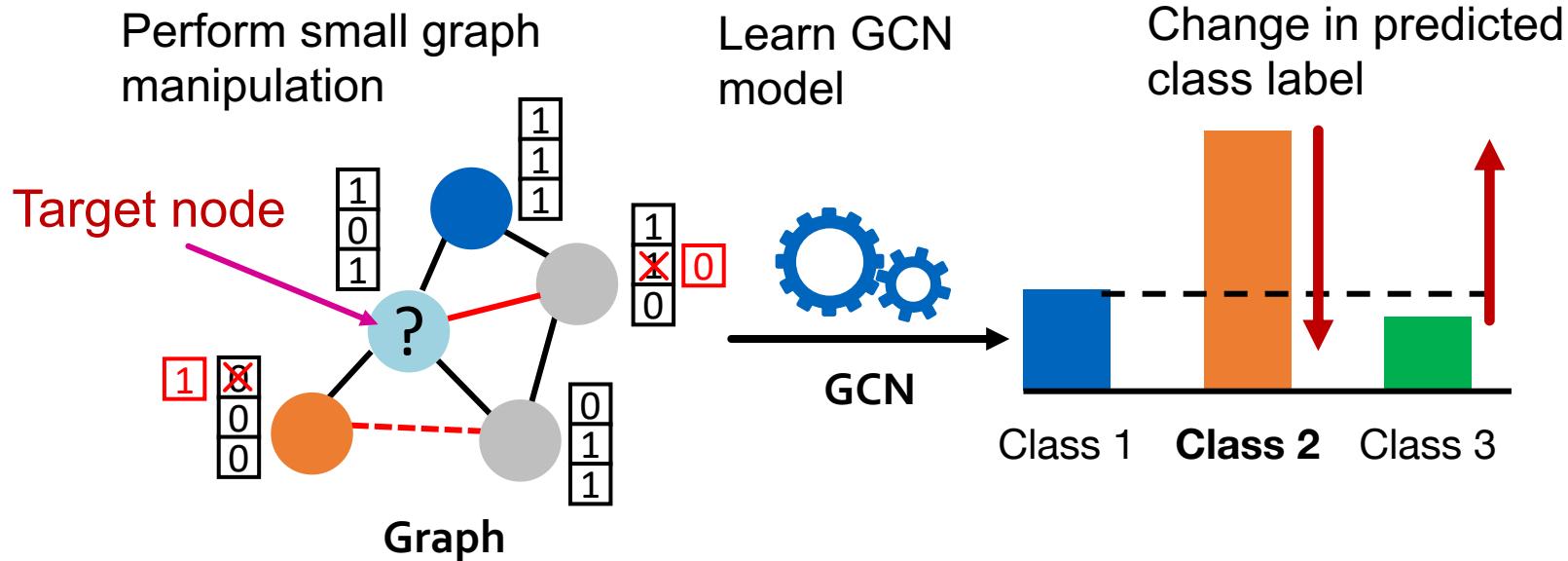


Formalizing Adversarial Attacks

■ Objective for the attacker:

Maximize (**change of target node label prediction**)
Subject to (**graph manipulation is small**)

If graph manipulation is too large, it will easily be detected.
Successful attacks should change the target prediction
with “unnoticeably-small” graph manipulation.



Mathematical Formulation (1)

- **Original graph:**
 - A : adjacency matrix, X : feature matrix
- **Manipulated graph (after adding noise):**
 - A' : adjacency matrix, X' : feature matrix
- **Assumption:** $(A', X') \approx (A, X)$
 - Graph manipulation is **unnoticeably small**.
 - Preserving basic graph statistics (e.g., degree distribution) and feature statistics.
 - Graph manipulation is either **direct** (changing the feature/connection of target nodes) or **indirect**.

Mathematical Formulation (2)

- **Target node:** $v \in V$
- GCN learned over the **original graph**
$$\theta^* = \operatorname{argmin}_{\theta} \mathcal{L}_{train}(\theta; A, X)$$
- GCN's original prediction on the **target node**:

$$c_v^* = \operatorname{argmax}_c f_{\theta^*}(A, X)_{v,c}$$

Predict the class c_v^* of vertex v that has the highest predicted probability

Mathematical Formulation (3)

- GCN learned over the **manipulated graph**

$$\boldsymbol{\theta}^{*''} = \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}_{train}(\boldsymbol{\theta}; \mathbf{A}', \mathbf{X}')$$

- GCN's prediction on the **target node v** :

$$c_v^{*''} = \operatorname{argmax}_c f_{\boldsymbol{\theta}^{**}}(\mathbf{A}', \mathbf{X}')_{v,c}$$

- We want the prediction to change after the graph is manipulated:

$$c_v^{*''} \neq c_v^*$$

Mathematical Formulation (4)

- Change of prediction on target node v :

$$\Delta(v; A', X') =$$

$$\log f_{\theta^{*'}}(A', X')_{v, c_v^{*'}} - \log f_{\theta^{*'}}(A', X')_{v, c_v^*}$$

Predicted (log)
probability of the
newly-predicted
class $c_v^{*'}$

Predicted (log)
probability of the
originally-predicted
class c_v^*



Want to increase
this term



Want to decrease
this term

Mathematical Formulation (5)

- **Final optimization objective:**

$$\operatorname{argmax}_{A', X'} \Delta(v; A', X')$$

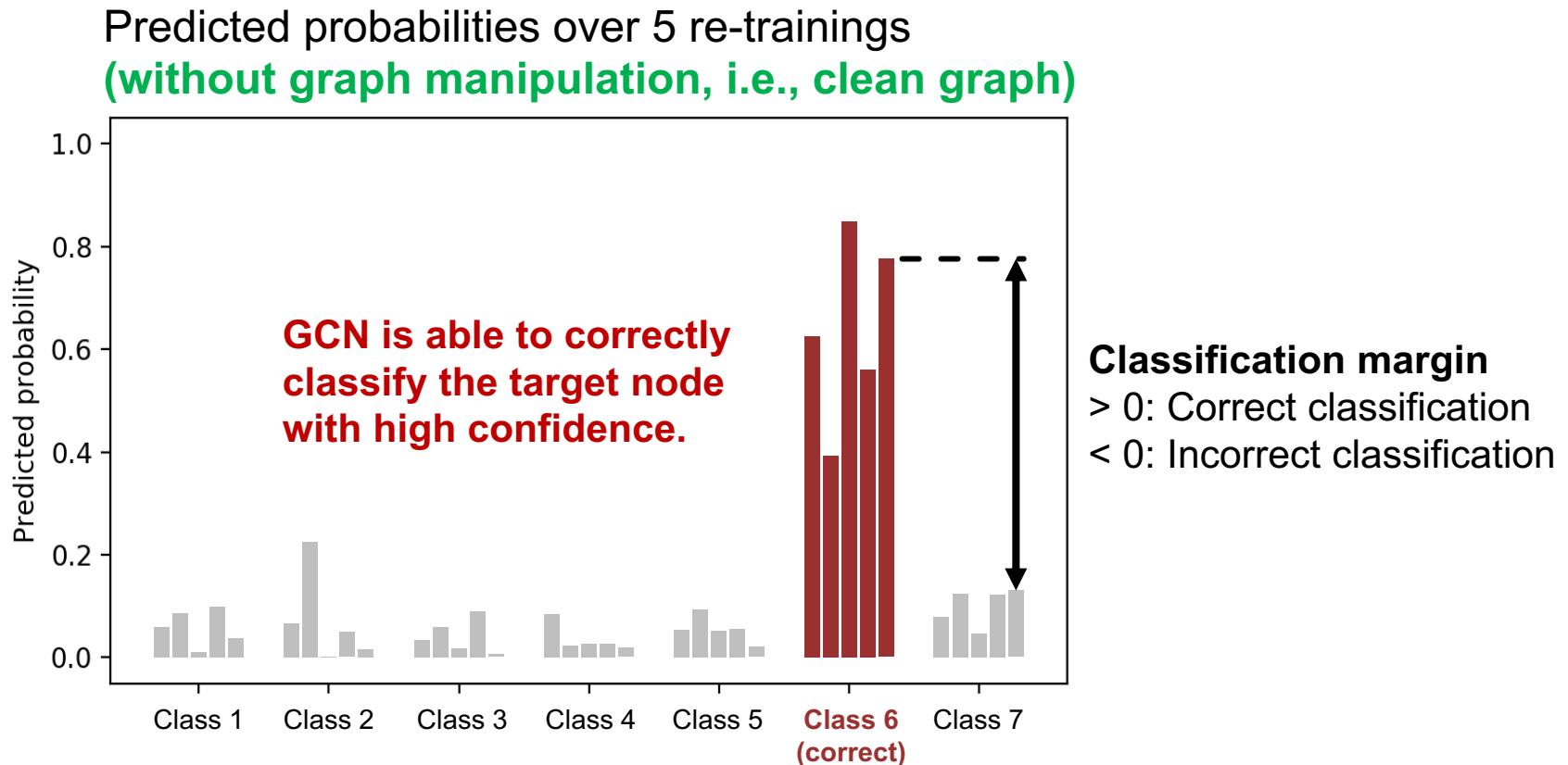
subject to $(A', X') \approx (A, X)$

- **Challenges in optimizing the objective**

- Adjacency matrix A' is a discrete object: gradient-based optimization cannot be used.
- For every modified graph A' and X' , GCN needs to be re-trained (this is computationally expensive):
 - $\theta^{*'} = \operatorname{argmin}_{\theta} \mathcal{L}_{train}(\theta; A', X')$
- Several approximations are proposed to make the optimization tractable [Zügner et al. KDD2018].

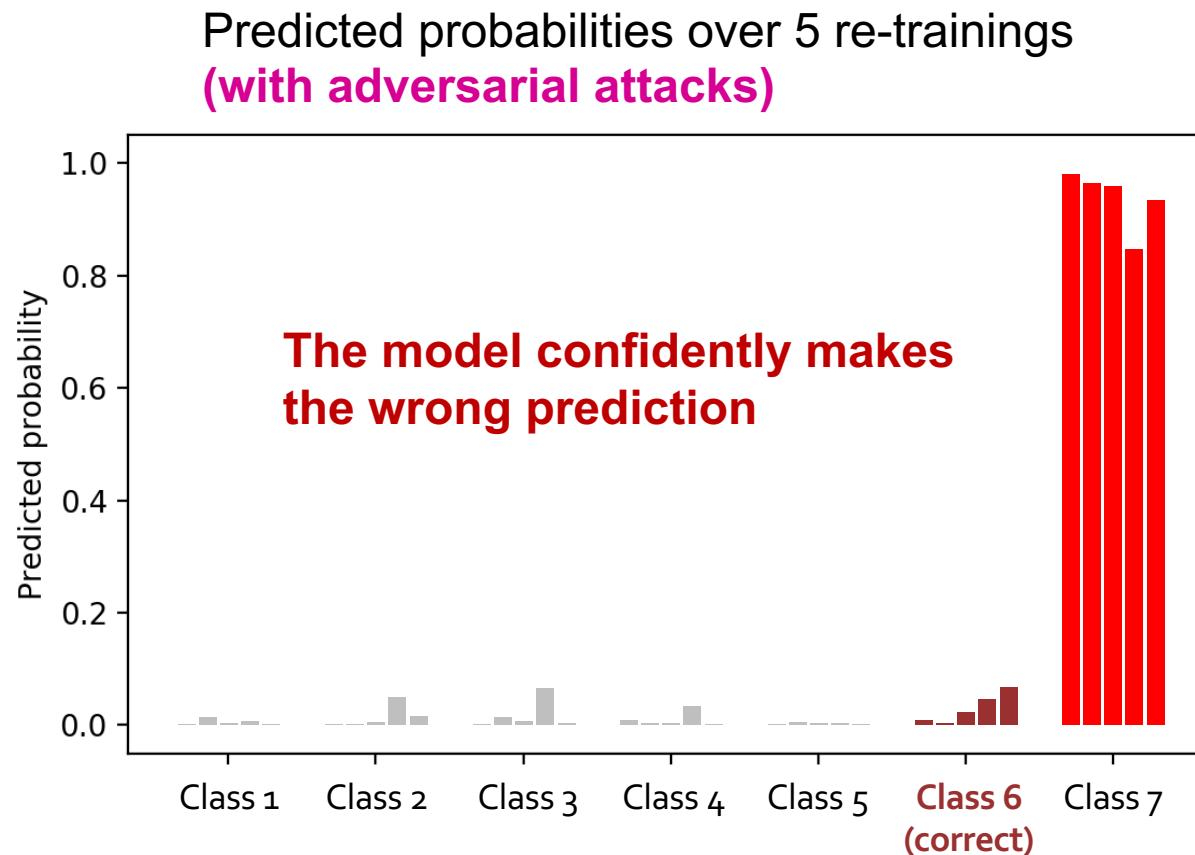
Experiments: Adversarial Attack

Semi-supervised node classification with GCN on a paper citation network (2,800 nodes, 8,000 edges).



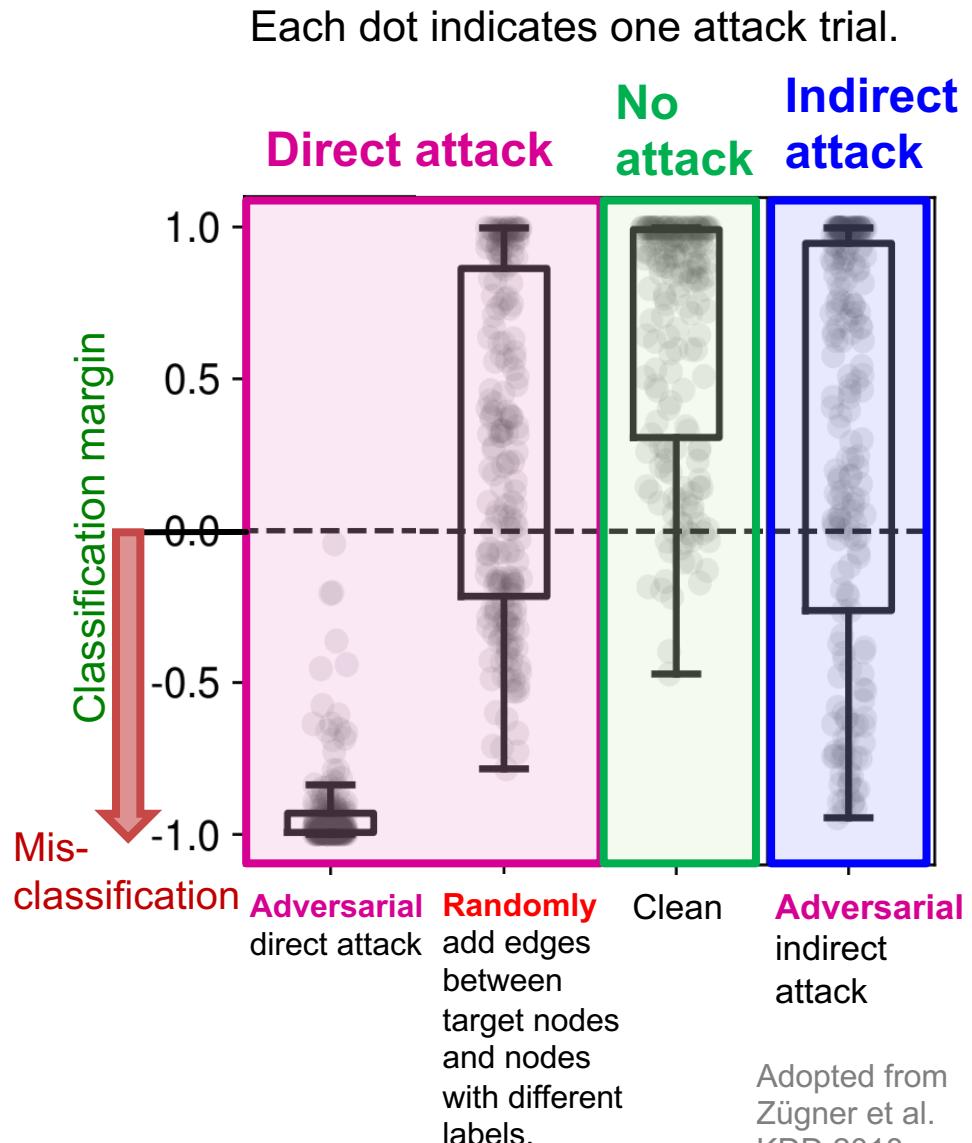
Experiments: Adversarial Attack

GCN's prediction after carefully modifying just 5 edges attached to the target node (**direct adversarial attack**).



Experiments: Attack Comparison

- **Adversarial direct attack** is the strongest attack, significantly worsening GCN's performance (compared to **no attack**).
- **Random** attack is much weaker than **adversarial** attack.
- **Indirect attack** is more challenging than direct attack.



Summary

- We study the adversarial robustness of GCN applied to semi-supervised node classification.
- We consider different **attack possibilities on graph-structured data.**
- We mathematically **formulate the adversarial attack as an optimization problem.**
- We empirically demonstrate that GCN's prediction performance can be significantly harmed by adversarial attacks.
- **GCN is *not* robust to adversarial attacks but it is somewhat robust to indirect attacks and random noise.**