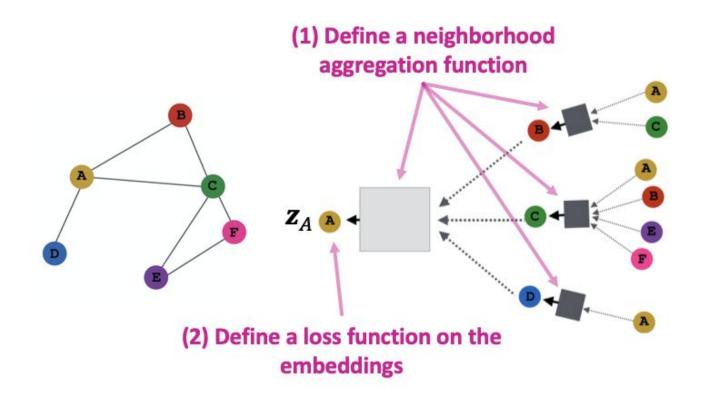
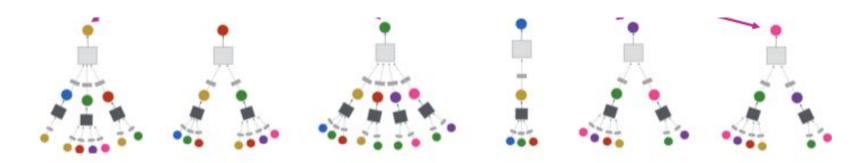
Анализ графовых данных и глубокое обучение

В предыдущих сериях



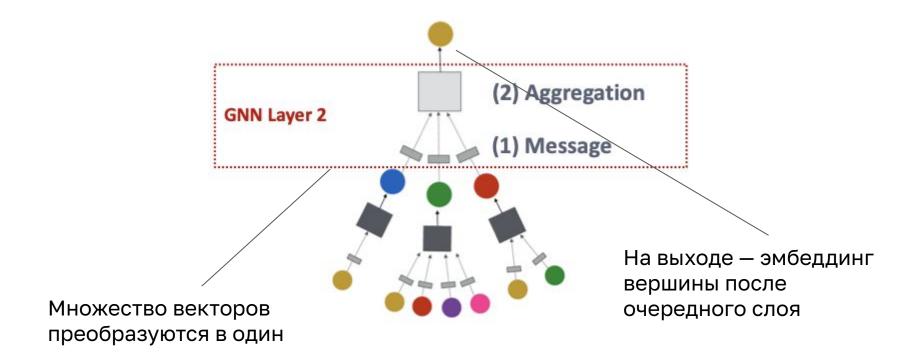
В предыдущих сериях

- Вершины агрегируют информацию от соседей с помощью нейронных сетей
- Топология графа определяет свой вычислительный граф для каждой вершины



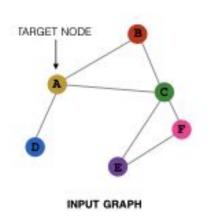
GNN Layer Architecture

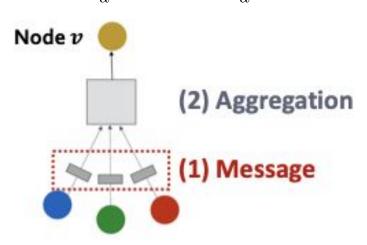
GNN Layer



Message Computation

- Каждая соседняя вершина посылает сообщение, основываясь на своём эмбеддинге с предыдущего слоя
 - \circ Например, линейный слой $m_u^{(l)} = W^{(l)} h_u^{(l-1)}$

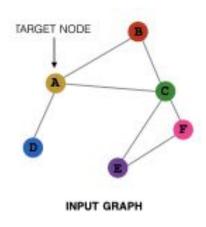


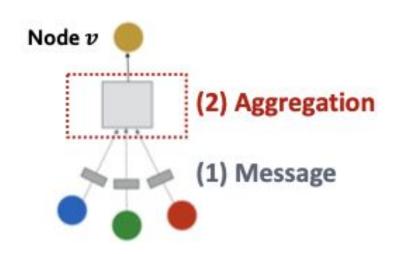


Aggregation

- Сообщения от соседей агрегируются
 - о Например, Sum, Mean, Max

$$h_v^{(l)} = AGG^{(l)}(\{m_u^{(l)}, u \in N(v)\})$$





Aggregation

- Информация о вершине v из предыдущих слоев может потеряться
- ullet Поэтому часто добавляют $\,h_v^{(l-1)}$ при подсчёте $\,h_v^{(l)}$
- Можно обучать отдельные параметры для таких сообщений

• Также можно использовать отдельную агрегацию

Then aggregate from node itself
$$\mathbf{h}_v^{(l)} = \text{CONCAT}\left(\text{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right), \mathbf{m}_v^{(l)}\right)$$
 First aggregate from neighbors

Архитектура слоя GNN

- Вычисления сообщений + Агрегация
- Нелинейность может быть добавлена в любую из этих двух частей
 - Например, Relu
- Классические архитектуры
 - GCN
 - GraphSAGE
 - GAT

GCN

Graph Convolutional Networks (GCNs)

$$\mathbf{h}_{v}^{(l)} = \sigma \left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_{u}^{(l-1)}}{|N(v)|} \right)$$

• Что тут сообщения, а что агрегация?

GCN

Message

$$\mathbf{h}_{v}^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_{u}^{(l-1)}}{|N(v)|} \right)$$
Aggregation

- Сообщения от соседей: $\mathbf{m}_{u}^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)}$
- $oldsymbol{h}_v^{(l)} = \sigma \left(\operatorname{Sum} \left(\left\{ oldsymbol{m}_u^{(l)}, u \in N(v)
 ight\}
 ight)
 ight)$
- В GCN использовали петли для каждой вершины, чтобы одним из соседей была сама вершина

GraphSAGE

$$\mathbf{h}_{v}^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_{v}^{(l-1)}, \text{AGG}\left(\left\{\mathbf{h}_{u}^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

- Двухэтапная агрегация
- Агрегация сообщений от соседей:

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \mathrm{AGG}\left(\left\{\mathbf{h}_{u}^{(l-1)}, \forall u \in N(v)\right\}\right)$$

• Агрегация с сообщением от самой вершины:

$$\mathbf{h}_{v}^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_{v}^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

GraphSAGE: Агрегация соседей

• Взвешенная сумма соседей

$$AGG = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$
Aggregation Message computation

• Преобразование и агрегация

AGG = Mean(
$$\{MLP(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\}$$
)
Aggregation Message computation

Применение LSTM на перемешанных соседях

AGG = LSTM(
$$[\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))]$$
)
Aggregation

GraphSAGE: L2 нормализация

• Опционально можно применять нормализацию для эмеддингов на каждом слое

$$\mathbf{h}_{v}^{(l)} \leftarrow \frac{\mathbf{h}_{v}^{(l)}}{\left\|\mathbf{h}_{v}^{(l)}\right\|_{2}} \ \forall v \in V \ \text{where} \ \left\|u\right\|_{2} = \sqrt{\sum_{i} u_{i}^{2}} \ \left(\ell_{2}\text{-norm}\right)$$

 Так можно масштабировать все эмбеддинги, что иногда приводит к улучшению производительности

Graph Attention Networks (GATs)

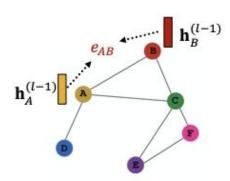
$$\mathbf{h}_{v}^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)})$$
Attention weights

- B GCN/GraphSAGE: $\alpha_{vu} = \frac{1}{|N(v)|}$
 - Эти веса определяются по структуре графа
 - Важность всех соседей одинакова
 - Основная идея GAT обучать эти параметры и фокусироваться на важных частях данных

- Let α_{vu} be computed as a byproduct of an attention mechanism α:
 - (1) Let a compute attention coefficients e_{vu} across pairs of nodes u, v based on their messages:

$$\boldsymbol{e_{vu}} = a(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\boldsymbol{h}_v^{(l-1)})$$

• e_{vu} indicates the importance of u's message to node v



$$e_{AB} = a(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)})$$

- Normalize e_{vu} into the final attention weight α_{vu}
 - Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

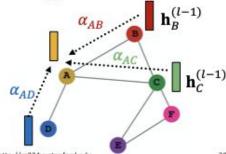
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

• Weighted sum based on the final attention weight α_{vv} :

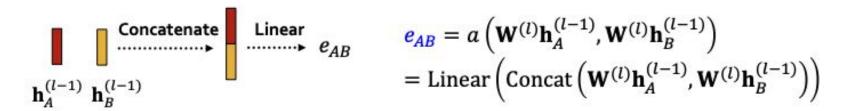
$$\mathbf{h}_{v}^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)})$$

Weighted sum using α_{AB} , α_{AC} , α_{AD} :

$$\mathbf{h}_{A}^{(l)} = \sigma(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_{B}^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_{C}^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_{D}^{(l-1)})$$



 Пробуйте различные способы вычислить эти коэффициенты, например линейный слой



 Для стабилизации обучения можно параллельно обучать несколько коэффициентов

$$\mathbf{h}_{v}^{(l)}[1] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{1} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)})$$

$$\mathbf{h}_{v}^{(l)}[2] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{2} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)})$$

$$\mathbf{h}_{v}^{(l)}[3] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{3} \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)})$$

$$\mathbf{h}_{v}^{(l)} = AGG(\mathbf{h}_{v}^{(l)}[1], \mathbf{h}_{v}^{(l)}[2], \mathbf{h}_{v}^{(l)}[3])$$

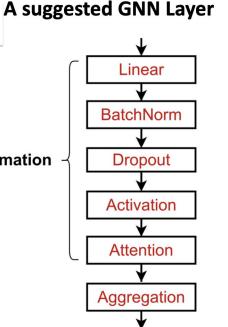
• Такие эмбеддинги агрегируются, суммой или конкатенацией

GNN Advanced Modules

GNN Layer Design

Для улучшения производительности можно использовать современные модули глубокого обучения
 Batch Normalization — стабилизация обучения Transformation
 Dropout — борьба с переобучением

- Attention/Gating контроль важности сообщений
- Любые другие модули глубокого обучения



Batch Normalization

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$

N node embeddings

Trainable Parameters:

 $\gamma, \beta \in \mathbb{R}^D$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$

Normalized node embeddings

Step 1:

Compute the mean and variance over N embeddings

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} \mathbf{X}_{i,j}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{X}_{i,j} - \mu_j)^2$$

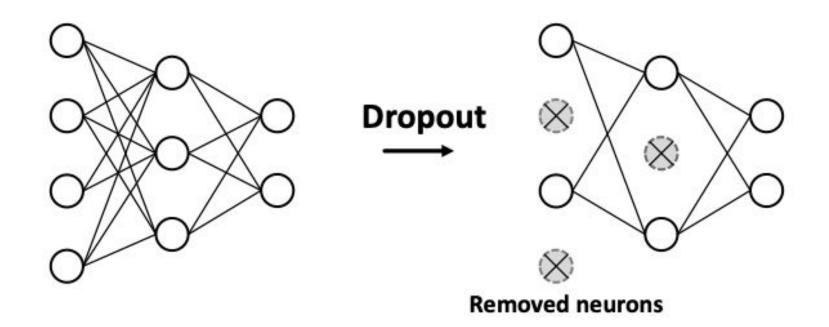
Step 2:

Normalize the feature using computed mean and variance

$$\widehat{\mathbf{X}}_{i,j} = \frac{\mathbf{X}_{i,j} - \mathbf{\mu}_j}{\sqrt{\mathbf{\sigma}_j^2 + \epsilon}}$$

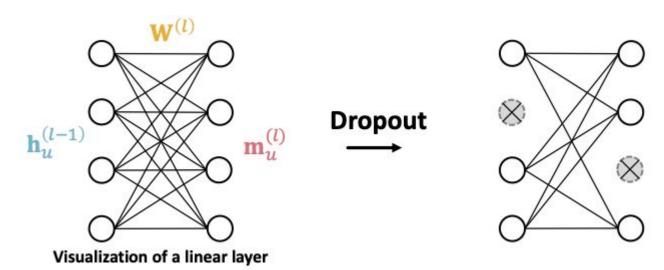
$$\mathbf{Y}_{i,j} = \mathbf{\gamma}_j \widehat{\mathbf{X}}_{i,j} + \mathbf{\beta}_j$$

Dropout



Dropout

- В GNN используют Dropout при вычислении сообщений
- Например, для линейного слоя: $\mathbf{m}_{u}^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_{u}^{(l-1)}$



Функции активации

Часто используют ReLU

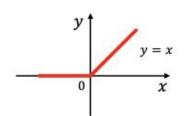
$$ReLU(\mathbf{x}_i) = max(\mathbf{x}_i, 0)$$

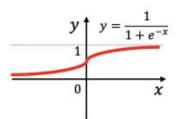
 Сигмоида, когда хотим ограничить область значений эмбеддингов

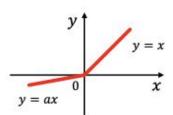
$$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}}$$

 Параметрический ReLU часто на практике эффективнее ReLU

$$PReLU(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + \mathbf{a}_i \min(\mathbf{x}_i, 0)$$

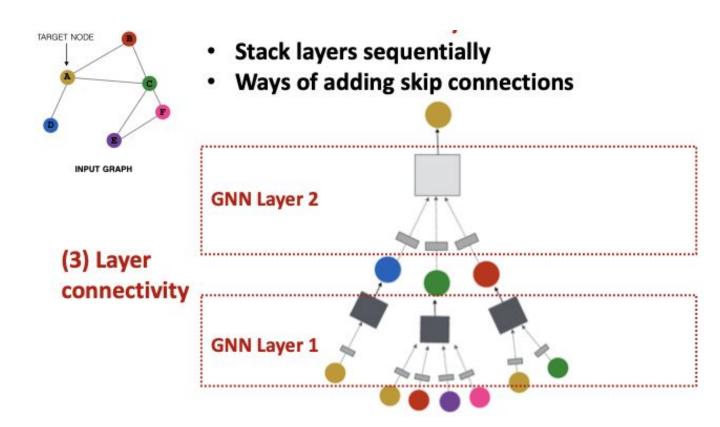






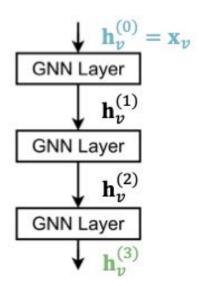
Multilayer GNN

Как связывать слои GNN?



Последовательные слои

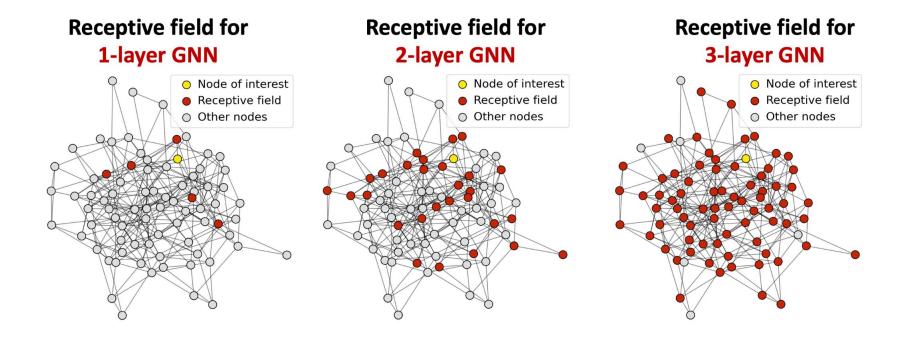
 Классическое решение – последовательно применять слои и трансформировать эмбеддинги вершин



Over-smoothing Problem

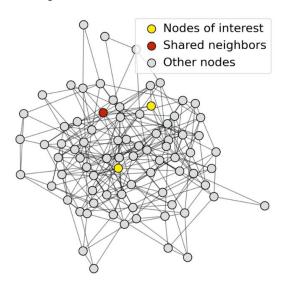
- При росте количества слоев эмбеддинги вершин становятся похожими
 - Похожие эмбеддинги → слабо различимые вершины → плохое качество модели
- Почему так происходит?

Receptive Field of a Node

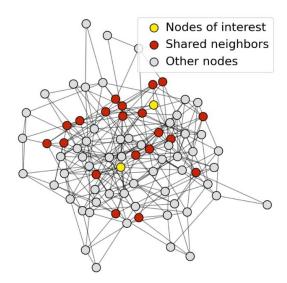


Receptive Field Overlap

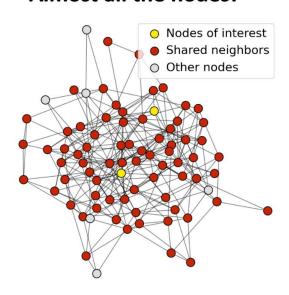
1-hop neighbor overlap Only 1 node



2-hop neighbor overlap About 20 nodes



3-hop neighbor overlap Almost all the nodes!

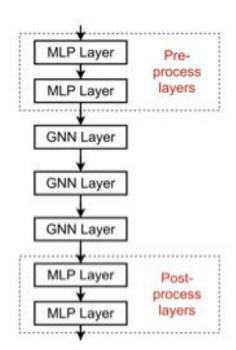


GNN Expressive Power

- Для увеличения выразительности GNN добавляем GNN слои, но не слишком много
 - Смотрим на диаметр графа и receptive fields
- Если хочется усложнить GNN, но слоёв больше не добавить, то можно
 - Усложнить сами GNN слои (например, 3-слойные MLP)
 - Добавить слои предобработки и постобработки

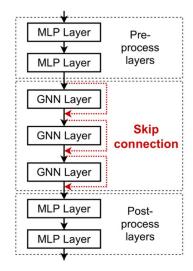
Дополнительные слои

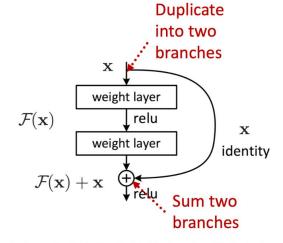
- Препроцессинг полезен для первичной обработки данных в вершинах, например изображения/текст
- Постпроцессинг для решения задачи, с использованием эмбеддингов вершин, например классификация графов



Skip Connections

- Информация с более ранних слоёв может быть важна и позволяет лучше различать вершины
- Напрямую передаём их в последующие слои





Idea of skip connections:

Before adding shortcuts:

$$F(\mathbf{x})$$

After adding shortcuts:

$$F(\mathbf{x}) + \mathbf{x}$$

GCN with Skip Connections

A standard GCN layer

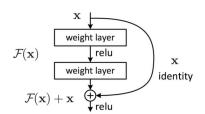
$$\mathbf{h}_{v}^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_{u}^{(l-1)}}{|N(v)|}\right)$$

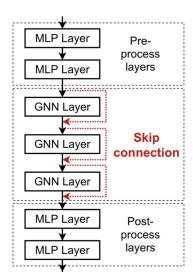
This is our F(x)

A GCN layer with skip connection

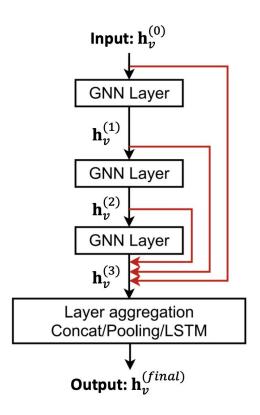
$$\mathbf{h}_{v}^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_{u}^{(l-1)}}{|N(v)|} + \mathbf{h}_{v}^{(l-1)} \right)$$

$$F(\mathbf{x}) + \mathbf{x}$$





Другой способ



Graph Manipulation in GNNs

Graph Manipulation

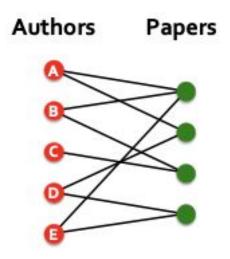
- Изначальный граф вряд ли оптимален для построения вычислительного графа
- Граф слишком разрежен → слабая передача информации
- Граф слишком плотный → большая вычислительная нагрузка
- Граф слишком большой → не влезает на GPU

Graph Manipulation

- Граф слишком разрежен → добавить искусственные вершины/рёбра
- Граф слишком плотный → делать выборку из соседей
- Граф слишком большой → обучаться только на подграфе
- Также можно добавить аугментацию самих данных, если изначально мало признаков в вершинах

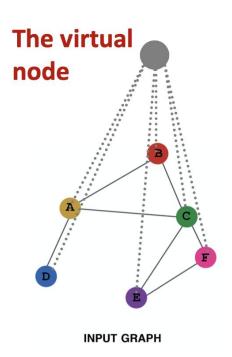
Добавление рёбер

• Например, использовать вместо матрицы смежности A использовать $A+A^2$



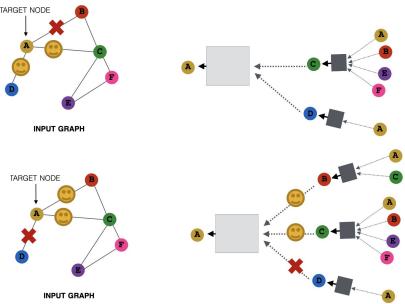
Добавление вершин

- Существенно улучшить передачу сообщений в разреженном графе можно добавив вершину, связанную со всеми остальными
- Кратчайшие расстояния в таком графе станут равны 2



Выборка соседей

• Существенно улучшить производительность можно сэмплированием соседей случайным образом каждый раз, вычисляя эмбеддинги



Feature augmentation

Feature augmentation: **constant** vs. **one-hot**

	Constant node feature	One-hot node feature
Expressive power	Medium. All the nodes are identical, but GNN can still learn from the graph structure	High. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	High. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	Low. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	Low. Only 1 dimensional feature	High. High dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

Feature augmentation

- GNN трудно различать некоторые структуры без дополнительной информации, например длина цикла, в котором находится вершина
- Можно добавить такую информацию как признак вершины

 v_1 resides in a cycle with length 3 v_1 resides in a cycle with length 4

 Можно также добавить другие признаки (Clustering coefficient, centrality, PageRank)

Заключение

