

Graph Neural Networks

DeepSchool Alumni

1

Всем привет. Рад всех видеть. Я работаю в области машинного обучения на графах. Решаю задачи на экономических графах, строю графовые нейронные сети.

И буду рад рассказать про то методы работы с графиками. Я подготовил некоторую презентацию, на протяжении которой мы можем в свободном формате, в режиме диалога провести нашу встречу.

Примерный план

- Графы
- Области применения
- Как работает GNN (На примере GCN)
- Напишем свою GCN на чистом PyTorch и обучим
- Формулировка задач и обучим пару нейронных сетей
- Inductive learning и GraphSAGE
- Разреженные матрицы и сэмплирование
- Изоморфизм графов и GNN
- Динамические графовые нейронные сети

Применение графов

Сначала посмотрим зачем нужны графы.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision

4

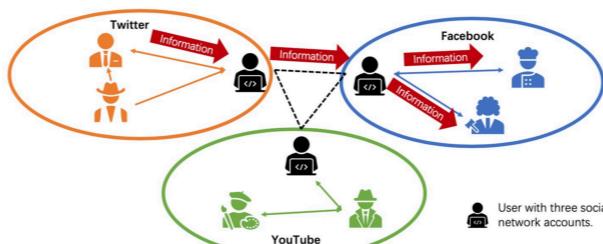
По сути машинное обучение на графах это надстройка над любой задачей, где можно найти связи между объектами. Поэтому их можно встретить много где.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision



Influence Maximisation



5

Есть задачи связанные с машинным обучением. Например довольно интересная задача максимизация влияния. Когда мы ищем людей, которые максимально распространяют информацию или инновацию на всех людей.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision



RecSys: JODIE

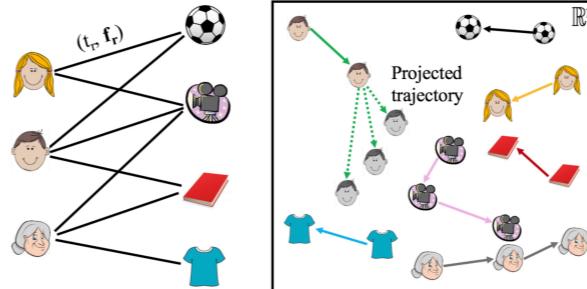


Figure 1: Left: a temporal interaction network of three users and four items. Each arrow represents an interaction with associated timestamp t and a feature vector f . Right: embedding trajectory of the users and items. We predict the future trajectory of users (the dotted line shown for one user) by training an embedding projection operator.

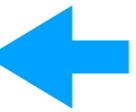
6

Или в рекомендательных сетях. Например есть графовая рекомендательная система JODIE, которая формирует граф между пользователями и товарами. f - это вектор признаков отношения между пользователем и товаром: например текст отзыва, размер покупки. А также формирует траекторию эмбеддинга пользователя и эмбеддинга товара, чтобы предсказать будущие отношения пользователя к товару.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision

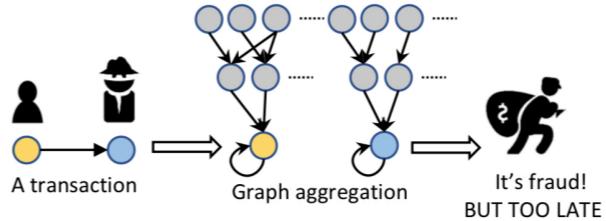
Traffic forecasting
ETA(Estimated Time of Arrival) prediction



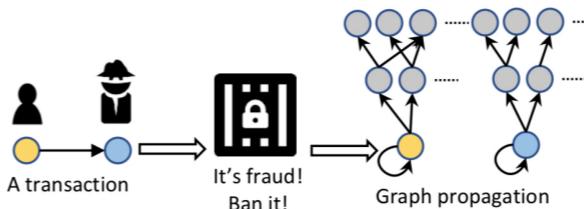
Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision

Fraud detection:



(a) Offline-deployed synchronous CTDG algorithm like TGAT [29]



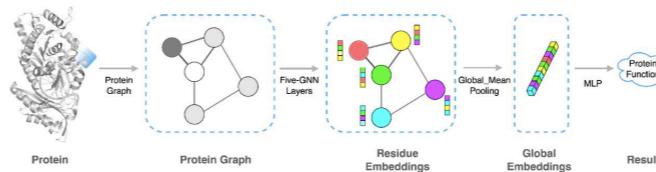
(b) Online-deployed asynchronous CTDG algorithm APAN

8

Графовые сети используются для экономических графов. Например модель APAN для предсказания мошенничества. Эта отличилась тем, что в ней ускорили алгоритм по выявлению мошенников.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/... ←
- NLP
- Computer Vision



- AlphaFold
- Предсказание структуры белка/РНК
- Молекулярные графы
- Белок-белковые взаимодействия

9

Графы используются в биологии, химии.

Применение

- Социальные сети
- Рекомендательные системы
- Транспортные сети
- Экономические графы
- Химия/биология/...
- NLP
- Computer Vision

Knowledge Graph Question Answering
LLM+KG: например DRAGON
Graph Based Dependency Parsing
Text2KG
...



10

В NLP есть интересные модели по применению графов знаний, чтобы перебороть галлюцинировавшие в языковых моделях.

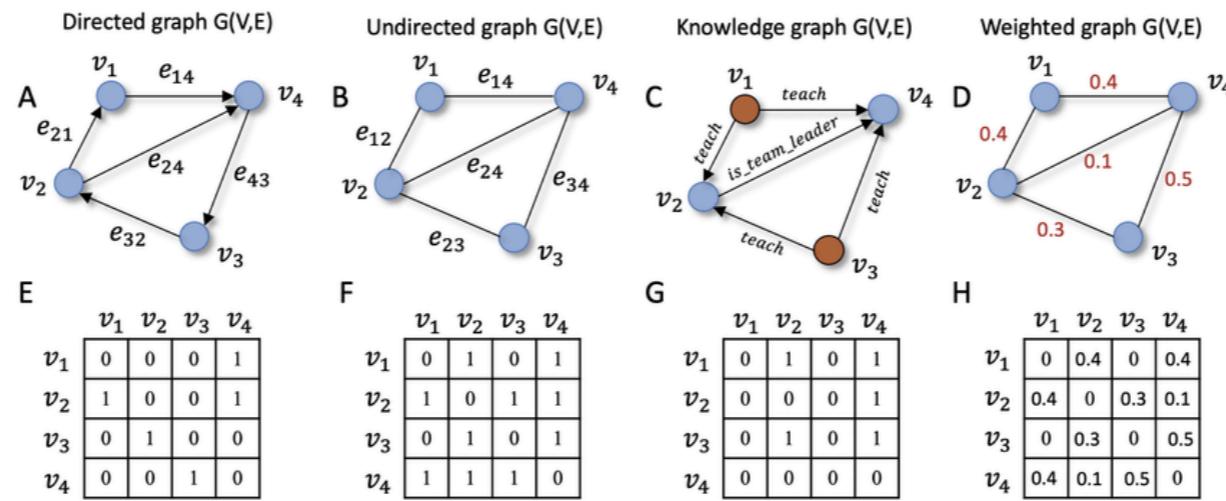
Применение

- Социальные сети
 - Рекомендательные системы
 - Транспортные сети
 - Экономические графы
 - Химия/биология/...
 - NLP
 - Computer Vision
- 
- **Pose Estimation**
 - **3D Semantic segmentation**
 - **3D shape generation**
 - **Annotation tools**
 - **Image matching**
 - **Structure from Motion**

11

И так же применяются в компьютерном зрении и трехмерном компьютерном зрении. Например для оценки позы человека, через наложение графа в виде простого скелета.

Виды графов



https://www.researchgate.net/publication/347300725_Understanding_graph_embedding_methods_and_their_applications?tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Ii9kaXJIY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

12

Графы бывают разных видов. Направленные графы где небо имеет направление.

В противоположность им ненаправленные, у которых матрица смежности симметрична.

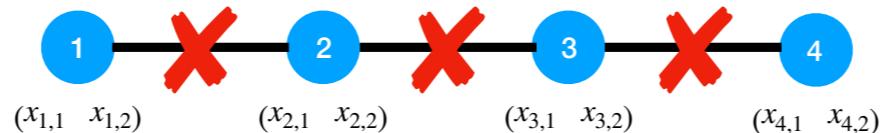
Графы знаний очень интересная область. Она отображается в виде так называемых гетерогенных графов. То есть узлы делятся на разные виды: например узлы-пользователи и узлы-товары. И также ребра. На рисунке мы видим следующие типы ребер: обучает, является лидером по отношению к .

И если у ребер имеются веса, то мы можем их записать в матрицу смежности.

Имеются ли у Вас вопросы?

Как работает GNN?

Игнорируем ребра



Linear Layer:
$$Y_{n,1} = X_{n,m} \cdot W_{m,1} = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{pmatrix} \begin{pmatrix} w_{1,1} \\ w_{2,1} \end{pmatrix} = \begin{pmatrix} w_{1,1}x_{1,1} + w_{2,1}x_{1,2} \\ w_{1,1}x_{2,1} + w_{2,1}x_{2,2} \\ w_{1,1}x_{3,1} + w_{2,1}x_{3,2} \end{pmatrix}$$

2-слойный MLP:
$$Y_{n,1} = Sigmoid(ReLU(X_{n,m} \cdot W_{m,h}) \cdot W_{h,1})$$

n - число узлов/объектов

m - число признаков у каждого узла/объекта

14

Пусть у нас имеется очень простой дотаяет из 4-х объектов. Который представляет собой очень простой граф: цепь из неориентированных ребер.

Каждый объект имеет пару признаков, например первый узел имеет признаки $x[1,1]$ и $x[1,2]$, второй узел имеет признаки $x[2,1]$ и $x[2,2]$, и т.д.

И каждый узел может принадлежать одному из двух классов: 0, 1.

Первое что мы можем сделать это проигнорировать связи между узлами. И тогда можно например использовать линейный слой.

Если добавить нелинейность и еще умножение на матрицу весов, то получаем многослойный перцептрон.

Есть ли здесь вопросы?

Матрица смежности



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad A^2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$(A^k)_{ij}$ – number of paths of length k from i to j

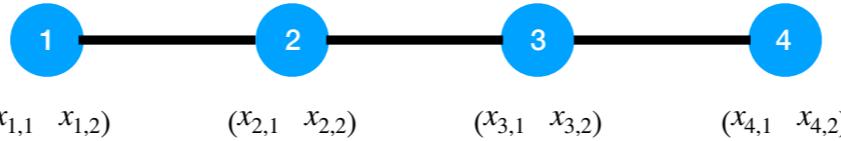
15

Но если мы хотим воспользоваться информацией о связях, то нам нужно их как-то закодировать. Один из вариантов как это сделать - через матрицу смежности.

Если на i строке и j столбце стоит 1, то значит что есть ребро между i и j узлами.

Если мы возведем матрицу в квадрат, то увидим что в матрице появились числа больше 1. Эти числа это число путей из i узла в j .

GCN



$$Y_{n,1} = A_{n,n} \cdot X_{n,m} \cdot W_{m,1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{pmatrix} \begin{pmatrix} w_{1,1} \\ w_{2,1} \end{pmatrix} =$$

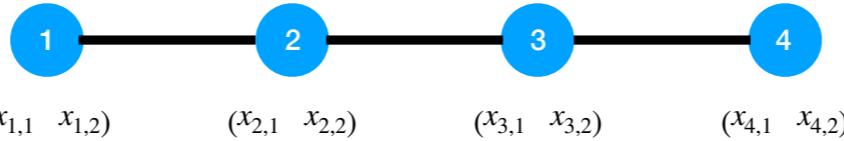
$$\begin{pmatrix} x_{2,1} & x_{2,2} \\ x_{1,1} + x_{3,1} & x_{1,2} + x_{3,2} \\ x_{2,1} + x_{4,1} & x_{2,2} + x_{4,2} \\ x_{3,1} & x_{3,2} \end{pmatrix} \begin{pmatrix} w_{1,1} \\ w_{2,1} \end{pmatrix} = \begin{pmatrix} w_{1,1}x_{2,1} + w_{2,1}x_{2,2} \\ w_{1,1}(x_{1,1} + x_{3,1}) + w_{2,1}(x_{1,2} + x_{3,2}) \\ w_{1,1}(x_{2,1} + x_{4,1}) + w_{2,1}(x_{2,2} + x_{4,2}) \\ w_{1,1}x_{3,1} + w_{2,1}x_{3,2} \end{pmatrix}$$

16

Так вот. Мы можем умножить матрицу смежности на матрицу признаком и на обучаемы веса как на слайде. И видим что в конечном счете на каждой позиции узла у нас получилась взвешенная сумма по соседям этого узла.

Если MLP обучал веса на основе признаков этого объекта, то матрица смежности добавила для обучения только соседей.

GCN



$$Y_{n,1} = A_{n,n} \cdot X_{n,m} \cdot W_{m,1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{pmatrix} \begin{pmatrix} w_{1,1} \\ w_{2,1} \end{pmatrix} = \begin{pmatrix} w_{1,1}x_{2,1} + w_{2,1}x_{2,2} \\ w_{1,1}(x_{1,1} + x_{3,1}) + w_{2,1}(x_{1,2} + x_{3,2}) \\ w_{1,1}(x_{2,1} + x_{4,1}) + w_{2,1}(x_{2,2} + x_{4,2}) \\ w_{1,1}x_{3,1} + w_{2,1}x_{3,2} \end{pmatrix}$$

Веса $w[i,j]$ учатся предсказывать метку узла обучаясь на признаках его соседей.

Что не так с этим решением?

Подсказка: этой проблемы не было у MLP

17

Получается что для каждого узла веса учатся предсказывать класс опираясь на признаки соседей этого узла.

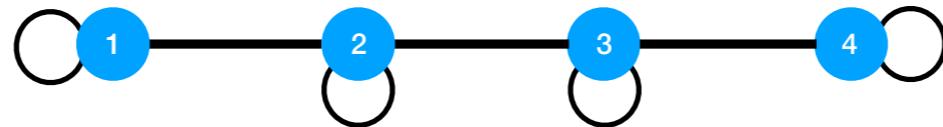
Но что не так с этой моделью?

Подсказка: этой проблемы нет у MLP.

Ответ: Когда мы предсказываем класс узла, мы игнорируем признаки этого узла.

Как решить эту проблему?

GCN: Adding self loops



$$\tilde{A} = I + A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$Y_{n,1} = \tilde{A}_{n,n} \cdot X_{n,m} \cdot W_{m,1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{pmatrix} \begin{pmatrix} w_{1,1} \\ w_{2,1} \end{pmatrix}$$

$$Y_{n,1} = \begin{pmatrix} w_{1,1}(x_{1,1} + x_{2,1}) + w_{2,1}(x_{1,2} + x_{2,2}) \\ w_{1,1}(x_{1,1} + x_{2,1} + x_{3,1}) + w_{2,1}(x_{1,2} + x_{2,2} + x_{3,2}) \\ w_{1,1}(x_{2,1} + x_{3,1} + x_{4,1}) + w_{2,1}(x_{2,2} + x_{3,2} + x_{4,2}) \\ w_{1,1}(x_{3,1} + x_{4,1}) + w_{2,1}(x_{3,2} + x_{4,2}) \end{pmatrix}$$

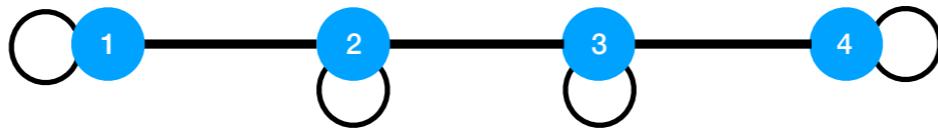
18

Мы вводим матрицу а-тильда, которая просто является суммой единичной матрицы и матрицы смежности. Графически это выглядит как если на каждом узле графа.

И теперь мы видим что эта проблема исчезла.

Есть еще одна проблема, не такая очевидна. То что узлы с большей степенью доминируют над узлами с меньшей степенью, то есть с меньшим числом ребер.

GCN: Adding self loops



$$\text{degree matrix : } D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad D^{-1/2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\tilde{A} = I + A$ ← Сначала добавили петли чтобы веса учились на признаках текущего узла

$\hat{A} = D^{-1/2} \tilde{A} D^{-1/2}$ ← Затем убрали доминирование узлов с большей степенью над узлами с меньшей степенью

19

Чтобы решить вторую проблему, нужно просто нормировать в матрице смежности каждое значение на степени узлов. Для этого введем матрицу степеней. У этой матрицы заполнена только диагональ, где на i элементе диагонали проставлена степень i узла.

Поскольку это диагональная матрица. Когда мы возводим матрицу в степень, то результатом будет матрица где в эту степень возведены элементы диагонали.

Если мы возьмем матрицу тильда-а и умножим слева и справа матрицу степеней узлов, возведенную в степень минус одна вторая. То тем самым нормируем каждый элемент на среднее геометрическое степеней.

И тогда получим новую матрицу - а-крышка. Ее мы и будем использовать.

Multilayer GCN

Первый слой
графовой
свертки GCNConv

$$Y_{n,1} = \text{Sigmoid}(\hat{A}_{n,n} \cdot \text{ReLU}(\hat{A}_{n,n} \cdot X_{n,m} \cdot W_{m,h}^{(0)}) \cdot W_{h,1}^{(1)})$$

Второй слой графовой
свертки GCNConv

Разделим по слоям

Признаки: $H_{n,m}^{(0)} = X_{n,m}^{(0)}$

Первый слой: $H_{n,h}^{(1)} = \text{ReLU}(\hat{A}_{n,n} \cdot H_{n,m}^{(0)} \cdot W_{m,h}^{(0)})$

Второй слой: $Y_{n,1} = \text{Sigmoid}(\hat{A}_{n,n} \cdot H_{n,h}^{(1)} \cdot W_{h,1}^{(1)})$

20

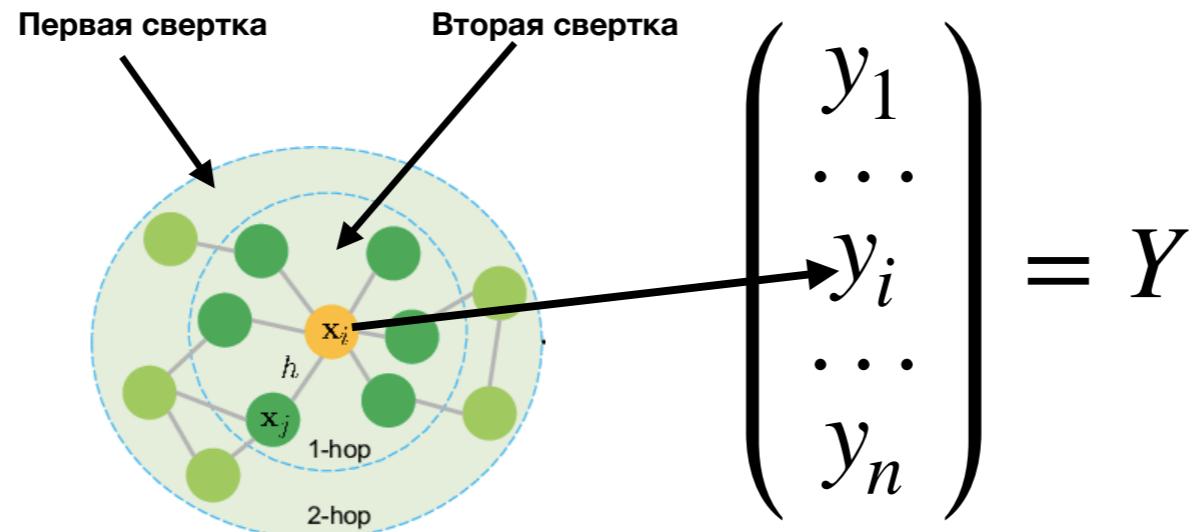
Выше уже показано как сделать нейрону уже с двумя слоями грабовой свертки. Где в желтом прямоугольнике первый слой, а в синем - второй.

Но такой формулой не удобно пользоваться, гораздо удобнее когда есть отдельные слои.

И ниже мы их как раз добавили.

GCN

$$H_{n,m}^{(0)} = X_{n,m}^{(0)} \rightarrow H_{n,h}^{(1)} = \text{ReLU}(\hat{A}_{n,n} \cdot H_{n,m}^{(0)} \cdot W_{m,h}^{(0)}) \rightarrow Y_{n,1} = \text{Sigmoid}(\hat{A}_{n,n} \cdot H_{n,h}^{(1)} \cdot W_{h,1}^{(1)})$$



21

Когда мы пользовались только одной графовой сверткой, использовались признаки узлов, достижимых в рамках одного прыжка от текущего узла. То есть соседи.

Если же мы используем сеть из двух графовых сверток, то уже используются узлы достижимые в два прыжка от текущего узла.

От трех сверток - в три прыжка. И так далее.

Но что интересно это происходит в обратном порядке. Например, мы хотим предсказать класс i -го узла. Тогда вторая графовая свертка будет использовать соседей этого узла, то есть узлы достижимые за один прыжок. А первая грабовая свертка будет использовать соседей соседей узла i . То есть узлы на расстоянии двух прыжков от узла i .

В силу этого мы не можем создавать графовые сети с большим числом слоев. Так как в какой-то момент мы получим сеть, в которой каждый узел будет агрегировать информацию от всего графа целиком. То есть получается что грабовая сеть вырождайся в обычный многослойный перцептрон.

По опыту получалось что достаточно 2-3 слоев, максимум 4-5. Что похоже на теорию шести рукопожатий, согласно которой любые два человека на Земле разделены не более чем на 5 рукопожатий.

GCN

$$H^{(l+1)} = Activation(\hat{A}_{n,n} \cdot H^{(l)} \cdot W^{(l)})$$

```
3 class GCNConv(th.nn.Module):
4     def __init__(self, in_features, out_features):
5         super().__init__()
6         self.weight = th.nn.Parameter(th.FloatTensor(in_features, out_features))
7         self.bias = th.nn.Parameter(th.FloatTensor(out_features))
8         self.reset_parameters()
9
10    def forward(self, input, adj):
11        support = th.spmm(adj, input)
12        out = th.spmm(support, self.weight)+self.bias
13        return out
14
15    def reset_parameters(self):
16        stdv = 1. / math.sqrt(self.weight.size(1))
17        self.weight.data.uniform_(-stdv, stdv)
18        self.bias.data.uniform_(-stdv, stdv)
19
```

Здесь на слайде показана общая формула слоя GCNConv и ниже код, который реализовывает этот слой.

Как видите для имплементации GNN так же достаточно просто перемножить 3 матрицы согласно формуле. Нюанс только в том, что в коде использовалось произведение для разреженных матриц эс-пи-эм-эм (spmm), чтобы слой поддерживал работу с большими графами.

Демонстрация кода

1

Теперь применим наши знания на практике. Рассмотрим такую задачу как классификацию статей в сети цитирований.

Cora Dataset

- Каждый узел это статья о Машинном Обучении
- Каждое ребро отображает ссылку одной статьи на другую
- Каждый узел попадает в одну из следующих категорий:
 - Case based
 - Genetic algorithms
 - Neural networks
 - Probabilistic methods
 - Reinforcement learning
 - Rue leaning
 - Theory
- Признаки это мешок слов статьи. В оригиналe это просто multi-hot вектор. В коде этот вектор просто нормирован по строкам, поэтому значения там не 0 или 1, а некоторое число между 0 и 1.

Для этого будем использовать датасет Cora. (Зачитываем факты из слайда)

Демонстрация кода

<https://github.com/roman-4erkasov/gnn-base/blob/main/GraphConvolutionNetwork.ipynb>

Формулировка задач

Мы посмотрели как работают графовые нейронные сети на базе такой задачи как классификация узлов. Теперь давайте рассмотрим какие есть еще формулировки задач на языке графовых сетей.

Виды GraphML

- Статистика на графах
- Классический ML на графах
- Моделирование процессов на графах: болезни (SIR, MSEIR, SEIS), влияние, инновации, транспортные потоки
- Блуждание на графах (node2vec,...)
- Нейронные сети на графах

Помимо графовых нейронных сетей есть другие области машинного обучения на графах:

Статистика на графах, изучение графов с помощью подсчета разных статистик: плотность графа, диаметр графа. Поиск компонент связности. Можно найти минимальное число ребер, чтобы разрезать граф.

Затем, изучив статистику графа, можно генерировать искусственные графы с такой же статистикой.

Классический ML это применение относительно простых алгоритмов обучения:

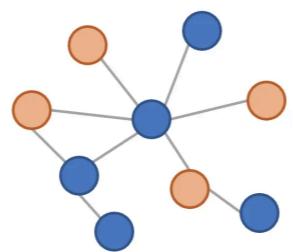
- например классификацию узла можно решить с помощью label propagation
- Можно подобрать функцию близости узлов, которая будет максимально коррелировать с вероятностью ребра между узлами
- И т.д.

На графах также можно моделировать процессы перемещения между узлами по ребрам. Например, эпидемии моделируется с помощью моделей SIR, MSEIR, SEIS.

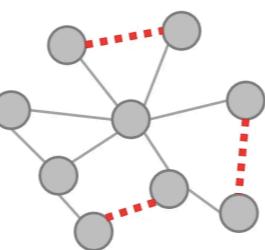
Есть еще методы генерации эмбеддингов с помощью блужданий на графах. Например: node2vec. Они похожи на word2vec. Можно использовать вектора, что упрощает работу. Их проблема в том что они учитывают только топологию без признаков узлов или ребер.

Формулировка задач GNN

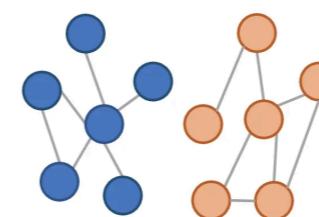
Node Classification



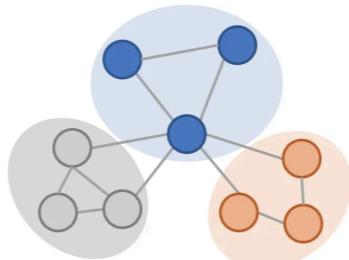
Link Prediction



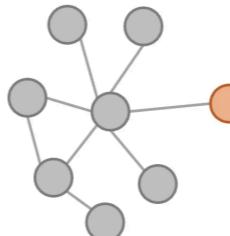
Graph Classification



Community Detection



Anomaly Detection

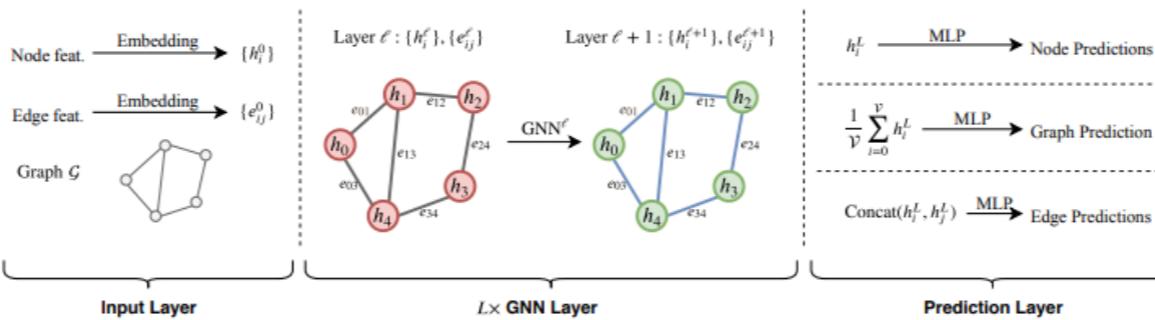


28

Здесь все понятно, просто кратко перечисляем задачи.

Только упоминает что сообщества могут пересекаться.

GNN Pipeline



Задачу GNN можно разбить на 2 этапа:

- 1) Составить эмбеддинги
- 2) Решить с помощью эмбеддингов поставленную задачу

<https://arxiv.org/pdf/2003.00982.pdf>

Здесь слайд показывает идею того что мы сначала информацию о графе превращается в вектора узлов - газовые эмбеддинги.

И далее уже решаем задачу используя эти вектора. То мы сначала упрощаем задачу от работы со сложными графиками до работы с простой "таблицей". А с просто "таблицей" решить задачу проще чем с графиками.

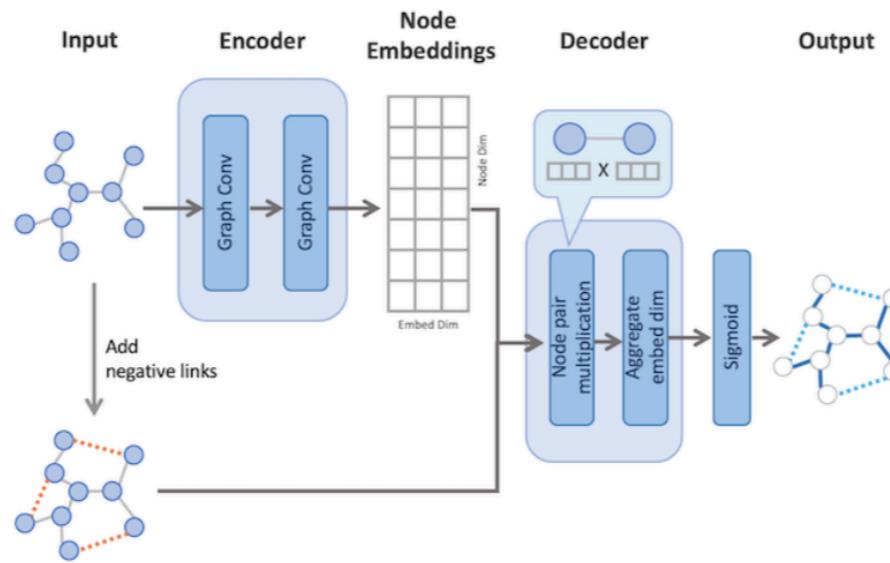
Дальше если мы хотим решить задачу классификации узлов, то прогоняем через MLP (как вариант) каждый вектор.

Дальше если мы хотим решить задачу классификации графов, то прогоняем через MLP (как вариант) агрегацию векторов (например среднее).

Если предсказание ребра, то как вариант можно конкатенировать вектора в пары и прогонять также через MLP. Обучаем MLP выдавать для пары 1, если ребро есть. И 0, если ребра нет. Соответственно MLP будет выдавать какую-то "вероятность" ребра.

Link Prediction

Link Prediction



Думаю, это лишний слайд. Лучше показать пайплайн предсказания ребра через демонстрацию кода, которая на следующем слайде.

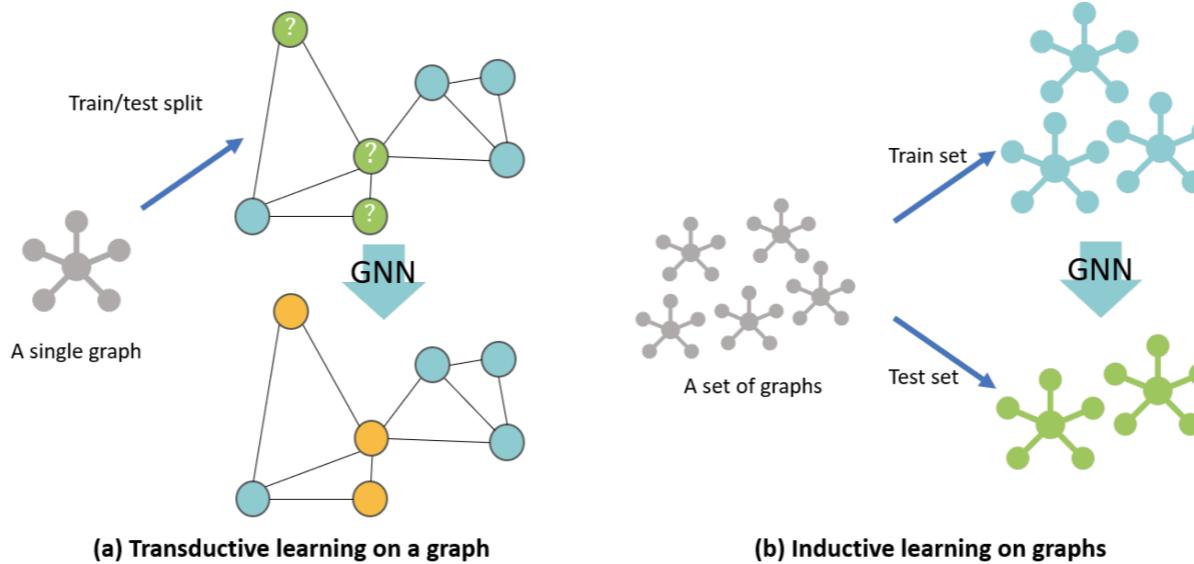
Демонстрация кода

2

<https://github.com/roman-4erkasov/gnn-base/blob/main/LinkPrediction.ipynb>

Inductive learning and GraphSAGE

Transductive learning vs Inductive learning



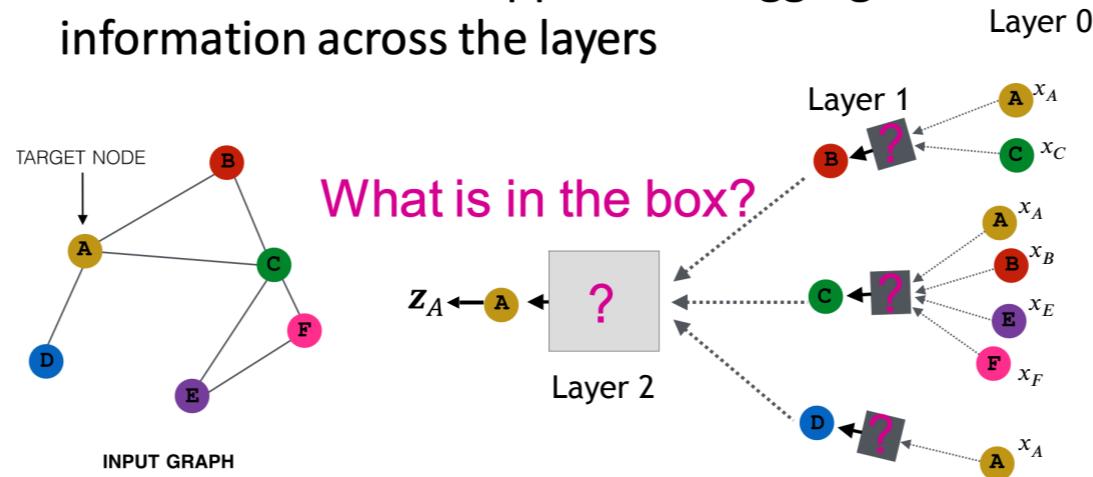
<https://www.mdpi.com/2220-9964/10/2/97>

Мы пока только рассматривали обучение, когда все объекты находятся в одном графе. Просто у одних объектов была разметка, а у других нет. И мы должны были обучиться на первых, чтобы заполнить вторые. Трансдуктивное обучение это такое обучение когда модель учится работать в рамках одного и того же графа. Такое обучение изображено на слайде слева.

Но можно обучать модель выполнять задачи в условиях того, что могут быть разные графы. Что модель учила закономерности в условиях того, что мы не ограничены одним и тем же графиком. Такое обучение называется индуктивным.

По крайней мере это верно в рамках графовых нейронных сетей. Далее мы как раз изучим как работает трансдуктивное обучение.

■ **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



Слева у нас представлен некий граф и мы хотим принять решение касательно узла A.

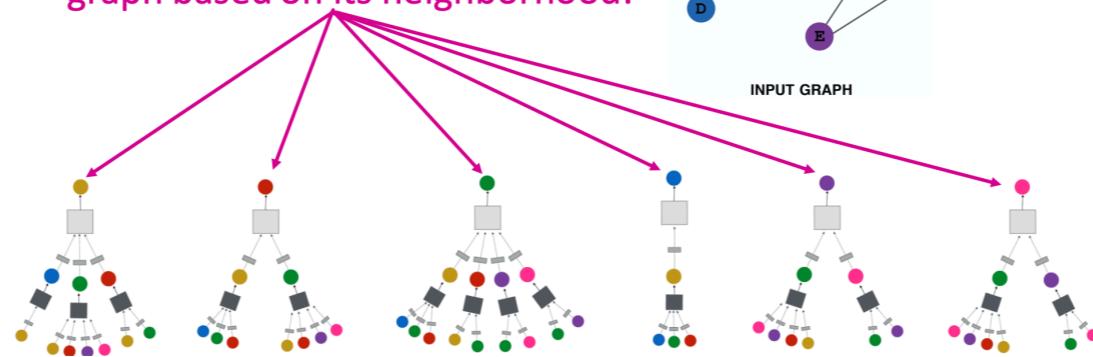
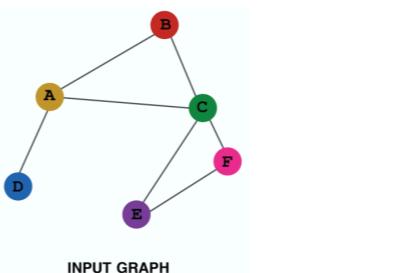
Тогда для этого узла у нас получится граф вычислений изображенный справа. Если у нас двухслойная сеть:

- Кружки это узлы,
- Квадраты это некие функции агрегации, которые агрегируют информацию от соседей в виде какого-то представления. Что это за функции мы пока не знаем. В конечном счете мы получаем эмбеддинг для нашего узла A.

Каждая агрегация определяет свой слой, а на вход первого слоя подаются неграфовые признаки. В данном случае они обозначены как икс с индексом узла, из которого они взяты.

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



10/7/21

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, <http://cs224w.stanford.edu>

56

35

Мы рассмотрели граф вычислений для узла A, но вообще подобные вычисления проводятся для каждого узла.

Дальше мы нам надо решить как выглядят эти функции агрегации.

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k : the hidden representation of node v at layer k

- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

36

Самый прямой подход как задать агрегации это рассматривать агрегации как full connected layers от соседей.

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Для того, чтобы построить $(k+1)$ -ый слой для эмбеддинга вершины v .

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Нужно перемножить обучаемую матрицу весов $W[k]$ на среднее по эмбеддингам соседей с предыдущего слоя.

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Плюс обучаемая матрица весов $B[k]$ умножить на эмбединг самого узла v с предыдущего слоя.

То есть, мы раздели веса для представления от соседних вершин - это матрица W . И отдельно веса для представления текущего узла - это матрица B .

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

В результате мы получим некоторый вектор, который подается на вход в функцию активации чтобы добавить нелинейность.

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

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

$\Rightarrow z_v = h_v^{(K)}$

Final node embedding

We can feed these **embeddings** into any loss function and run SGD to **train the weight parameters**

- h_v^k : the hidden representation of node v at layer k
- W_k : weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Так повторяем пока не дойдем до узла K -большое. Это и будет искомое представление - вывод нашей графовой нейронки. То есть эмбеддинг узла v .

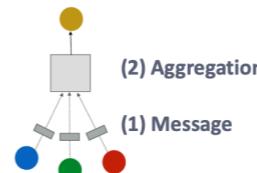
■ (1) Graph Convolutional Networks (GCN)

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

■ How to write this as Message + Aggregation?

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

Message
Aggregation



А GCN в рамках такого подхода будет выглядеть следующим образом.

Слои графовой свертки еще называют Message Passing слоями. На слайде ниже показано что имеется ввиду. Если матрицу весов принести под знак суммы. То получается:

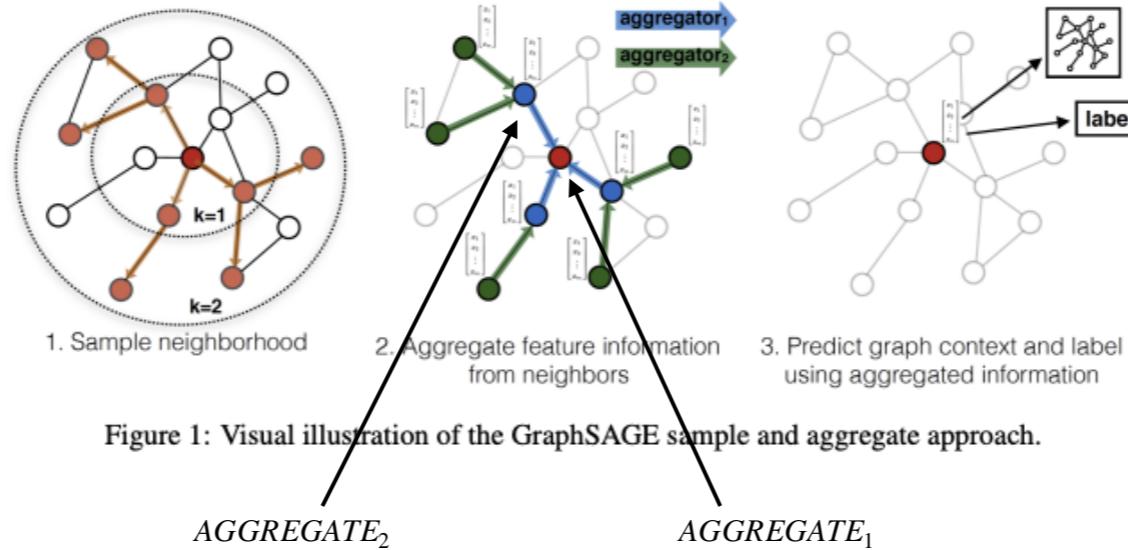
- 1) что сначала вектор каждого соседнего кодируется в новое представления. Это представление и является сообщением.
- 2) Затем все сообщения агрегируются , в данном случае такой агрегирующей функцией является сумма.

Справа внизу показана схема этих двух этапов.

Такой подход к представлению графовой сети используется в методе GraphSAGE.

GraphSAGE: Sample & aggreGatE

K - глубина соседства, на слайде она равна 2



Название происходит от двух первых букв слова SAmple и двух букв слова aggreGatE.

Когда мы сэмплируем мы задаем следующие:

- 1) Первый параметр это K - глубина соседства, насколько далеких соседей мы будем брать. То есть мы будем рассматривать узлы в пределах ка-большое прыжков.
- 1) И серия параметров это сколько соседей мы будем брать на каждом прыжке. Мы берем не всех соседей, а какое-то фиксированное число. Как это изображено на слайде. Мы на первом прыжке взяли только три соседа, на втором прыжке - уже только по два соседа от каждого.

Затем мы агрегируем представления

GraphSAGE Algorithm

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K ; weight matrices $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$; non-linearity σ ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \dots, K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations \mathbf{z}_v for all $v \in \mathcal{V}$

```
1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$  ;
2 for  $k = 1 \dots K$  do
3   for  $v \in \mathcal{V}$  do
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$ 
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$ 
6   end
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ 
8 end
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 
```

GCN: $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}))$

Pooling: $\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma(\mathbf{W}_{\text{pool}} \mathbf{h}_{u_i}^k + \mathbf{b}), \forall u_i \in \mathcal{N}(v)\})$

Так выглядит сам алгоритм.

Есть два вложенных цикла:

- внешний - по слоям
- и внутренний - по узлам.

Для каждого узла мы сначала на 4 строке агрегируем информацию от соседей.

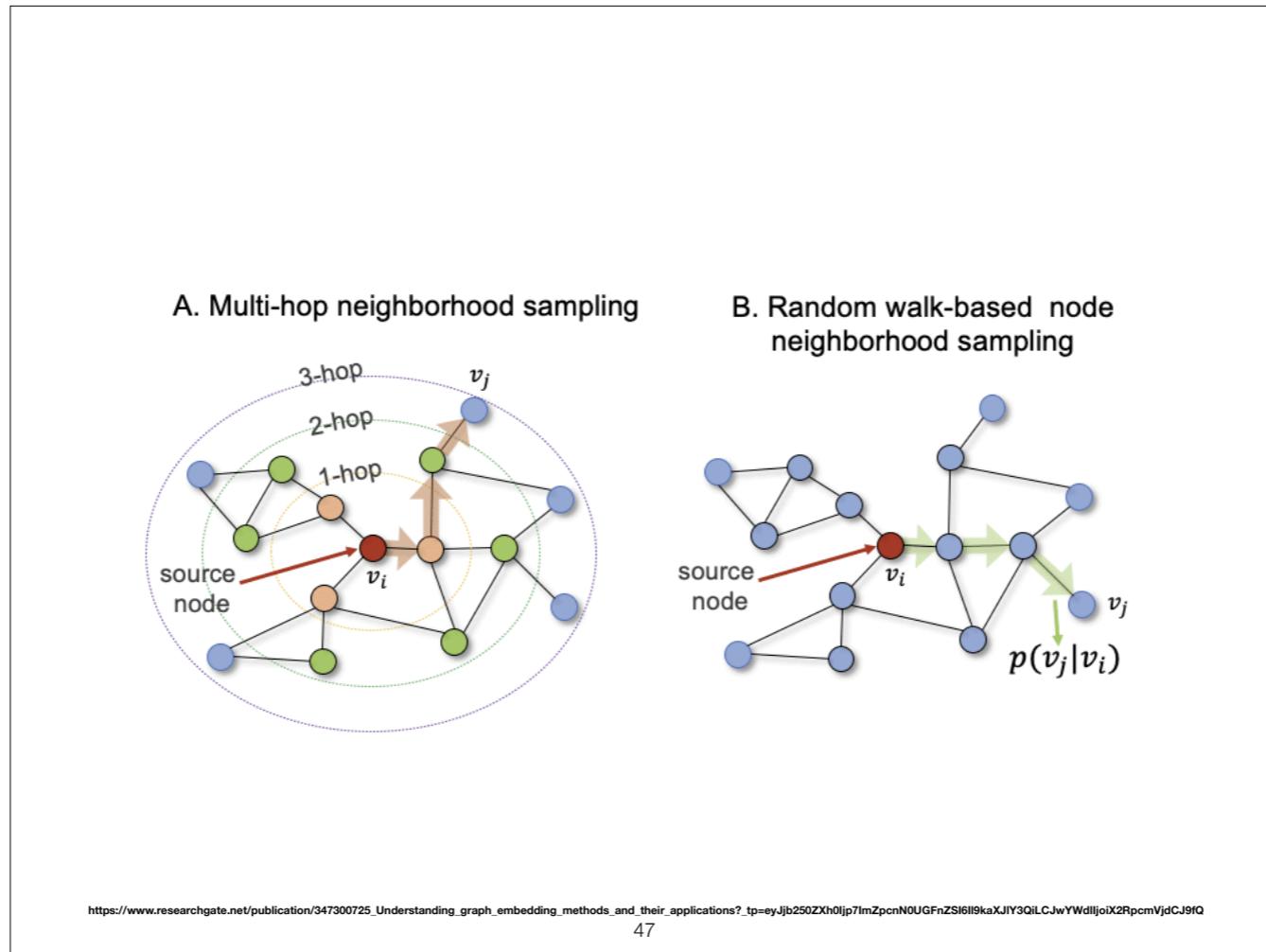
Затем на пятой строке получаем новое представление через матрицу весов. Причем мы сохраняем идею что можно использовать отдельные веса для представления соседей и для представления текущего узла.

Функция агрегации может быть разной. Если мы воспользуемся усреднением и откажемся разделения весов. То получим индуктивную версию GCN.

В пулинг агрегации мы выбираем максимальные значения для каждой компоненты среди соседей.

Есть также LSTM агрегация. Поскольку между соседями нету какого-то порядка, то в LSTM соседи передаются в случайном порядке.

Graph Sampling



Когда мы имеем дело с настолько большими графами, что для обучения целиком на всем кафе не хватает памяти. То один из вариантов как можно обучить сеть это использовать разреженные матрицы.

Но есть также второй способ, который появляется благодаря возможности обучать GNN индуктивно. Это сэмплировать из большого графа подграф и обучаться на таких подграфах.

Если мы просто будем брать случайные узлы, то мы можем с немалой вероятностью набрать много узлов между которыми не будет ребер. На таком нереально учиться.

Если мы будем брать подграф, то возникнет другая проблема. В подграф будут узлы, соседи которых не попали в подграф. Обучение на таких узлах тоже будет возможно но это будет снижать качество.

Поэтому есть методы сэмплирования, которые решают эту проблему.

Первый популярный вариант это Neibor Sampling, он был предложен в статье про GraphSAGE мы его только что рассматривали. Добавлю что мы лишь используем добавленных соседей в агрегации, но не добавляем их в целевые узлы.

Второй вариант это сэмплирования на случайном блуждании. Когда от целевого узла выполняются случайные переходы и новые посещенные узлы добавляются в сэмпл.

Graph Isomorphism

Изоморфные графы

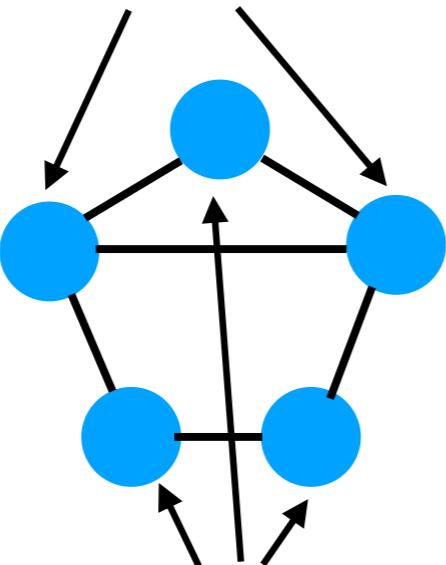
Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned}f(a) &= 1 \\f(b) &= 6 \\f(c) &= 8 \\f(d) &= 3 \\f(e) &= 2 \\f(g) &= 5 \\f(h) &= 2 \\f(i) &= 4 \\f(j) &= 7\end{aligned}$

Два графа считаются изоморфными, если мы можем идеально сопоставить вершины одного графа с вершинами другого.

На слайде изображены два изоморфных графа и в правом столбце - как найти изоморфизм между ними.

1-WL Test

Соседи: три синих узла



Алгоритм: First Order Weisfeller-Lehman Test

Тех, у кого соседи это три синих узла,
покрасим в желтый цвет.

Соседи: два синих узла

<https://www.youtube.com/watch?v=e4e6h9arD78>

Один из способов тестировать графы на изоморфизм это тест Вейсфеллера-Лемана первого порядка. Идея заключается в том чтобы сначала выполняем раскраску двух графов и затем пытаемся сопоставить цвета между ними. Если это не получается то графы не изоморфны. Иначе они частично изоморфны. Этот тест не совершенен, у него есть ограничения по способности различать топологию между графиками.

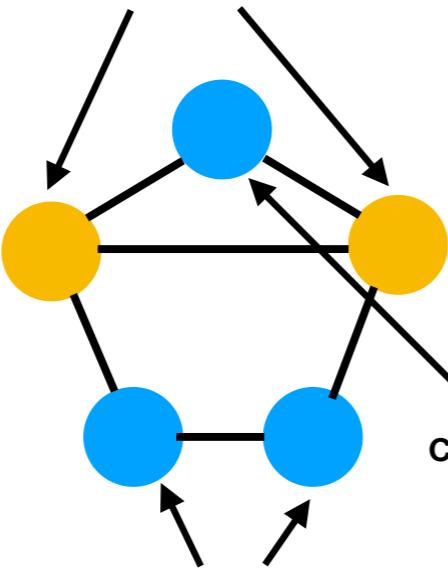
Мы делим узлы на группы, так чтобы узлы с одинаковым набором типов соседей попадали в одну группу. Каждую группу красим в отдельный цвет.

Рассмотрим это на примере.

(описываем слайд)

1-WL Test

Соседи: 2 синих и 1 жёлтый



Тех, у кого соседи это два желтых узла,
покрасим в зелёный цвет

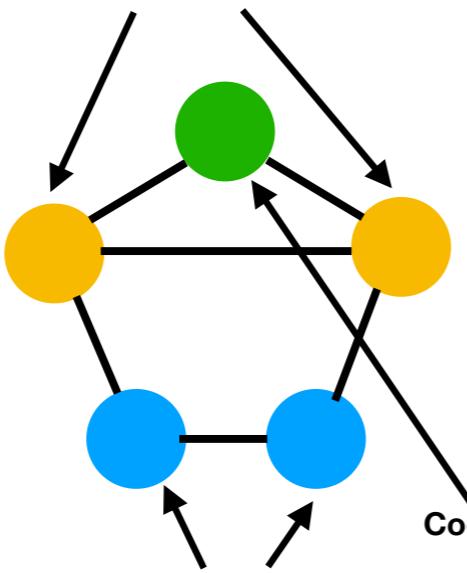
Соседи: 2 жёлтых

Соседи: 1 синий и 1 жёлтый

<https://www.youtube.com/watch?v=e4e6h9arD78>

1-WL Test

Соседи: 1 синий, 1 жёлтый и 1 зелёный



1) Дальнейшие итерации перекрашивания не поменяют разбиение улов.
Поэтому прекратим перекрашивание.

2) Если у двух покрашенных графов разное распределение цветов, то они не изоморфны.

Если одинаковое, то графы частично изоморфны.

<https://www.youtube.com/watch?v=e4e6h9arD78>

Graph Isomorphism Network

GCN и GraphSAGE хуже по мощности чем 1-WL Test

Graph Isomorphism Network

Keyulu Xu proposed a choice of the aggregation and update functions that make message passing neural networks equivalent to the WL algorithm, calling them *Graph Isomorphism Networks (GIN)*.

This is as powerful as a standard message-passing neural network can get.

<https://www.youtube.com/watch?v=e4e6h9arD78>

Graph Isomorphism Network

У графовой свертки есть два этапа

$$a_v^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right)$$

Агрегируем представление соседей

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, a_v^{(k)} \right)$$

Комбинируем
представление соседей
И представление текущего узла

Рассмотрим варианты AGGREGATE:

GCN: $h_v^{(k)} = \text{ReLU} \left(W \cdot \text{MEAN} \left\{ h_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\} \right\} \right).$

GraphSAGE
(а именно max pooling): $a_v^{(k)} = \text{MAX} \left(\left\{ \text{ReLU} \left(W \cdot h_u^{(k-1)} \right), \forall u \in \mathcal{N}(v) \right\} \right).$

И сравним такой агрегацией как просто сумма эмбеддингов соседей

Если резюмировать всю статью то у нас получится несколько формальных логических и простых шагов.

Шаг 1.

Графовую свертку можно разбить на 2 этапа:

- AGGREGATE: Агрегируем представление соседей
- COMBINE: Комбинируем представление соседей и представление текущего узла

Здесь ничего нового, в GraphSAGE была такая же идея.

И далее мы подумаем как лучше агрегировать. Авторы решили взять три варианта:

- 1 Бать среднее от соседей, то есть как в GCN
- 2 Брать max-pooling это один предложенных вариантов в статье про GraphSAGE
- 3 Давайте просто сложим все вектора

Graph Isomorphism Network

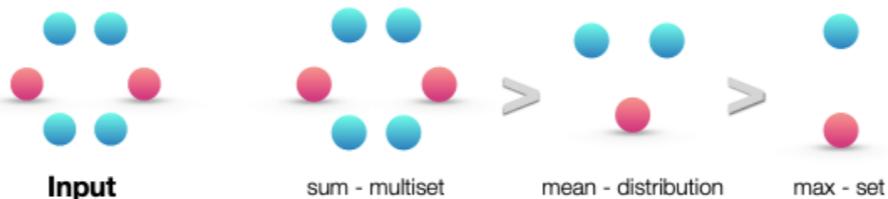


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, i.e., the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set).

arXiv:1810.00826v3 [cs.LG] 22 Feb 2019

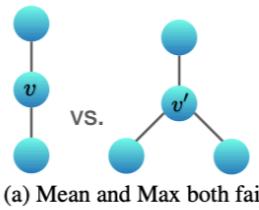
Обнаружили что между этими тремя методами агрегации есть отличие по expressive power (выразительность, способность выразить граф без потери информации).

Больше информации теряем max-pooling, так как ему не важно сколько было соседей насколько они были разнообразны, какие были пропорции между разными представлениями (векторами, разные соседи могут кодироваться в один и тот же вектор) соседей. Это все можно изменить и данная агрегация это не заметит. Главное не менять максимум.

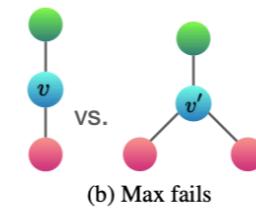
Усреднение сохраняет уже информацию о пропорциях. То есть можно поменять количество соседей, главное сохранить пропорции между разными представлениями соседей. Тогда агрегация через усреднение ничего не заметит.

Но сумма из рассмотренных агрегаций самая выразительная. Подметим это, это будет **Шаг 2**.

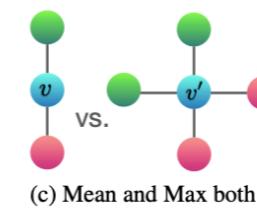
Graph Isomorphism Network



(a) Mean and Max both fail



(b) Max fails



(c) Mean and Max both fail

Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.** Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ. Figure 2 gives reasoning about how different aggregators “compress” different multisets and thus fail to distinguish them.

Почему в указанных примерах не срабатывают агрегации через усреднение и через максимум?

arXiv:1810.00826v3 [cs.LG] 22 Feb 2019

Рассуждение с слушателями по усвоению предыдущего слайда

Graph Isomorphism Network

Условия чтобы слой был равен по мощности 1-WL Test:

$$1) \quad h_v^{(k)} = \phi \left(h_v^{(k-1)}, f \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right) \right)$$

2) **Инъективность (для каждого узла можно сгенерировать эмбеддинг)**

Также упоминается что рассматриваются **permutation-invariant** функции, т.е. не важен порядок эмбеддингов.

В GIN множество эмбеддингов от соседних узлов рассматривается как мульти множества, так как вектора хоть от разных узлов могут повторяться.

Авторы нашли два условия, которые достаточны чтобы достичь мощность WL-теста.

Шаг 3 - это формула из первого условия. Это по сути те самые COMBINE и AGGREGATE . Где f - это AGGREGATE а фи - это COMBINE. Просто мы сразу подставили одну в другую.

Graph Isomorphism Network

Lemma 5. Assume \mathcal{X} is countable. There exists a function $f : \mathcal{X} \rightarrow \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \phi(\sum_{x \in X} f(x))$ for some function ϕ .

Кратко доказательство этой леммы:

Введем такое число N что N превышает размер мульти множества X .

Тогда есть как минимум две подходящие под Лемму функции f :

1. $f(x) = N^{-Z(x)}$ где $Z : \mathcal{X} \rightarrow \mathbb{N}$

2. f это one-hot вектора длины N

Graph Isomorphism Network

Это и есть GIN:
$$h_v^{(k)} = \text{MLP}^{(k)} \left(\left(1 + \epsilon^{(k)} \right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right)$$

Важность представления текущего узла по отношению к представлению соседей.

В PyG по-умолчанию равна нулю, причем можно задать как гиперпараметр а можно как обучаемый параметр

Следующий ключевой момент **Шаг 4** это 5-ая лемма.

В лекции чуть подробнее описал лемму 5. Грубыми словами она описывает следующую идею:

1. Можно подобрать такое отображение представлений набора узлов в другие вектора f что просто сумма по таким векторам будет однозначно определять каждый набора.

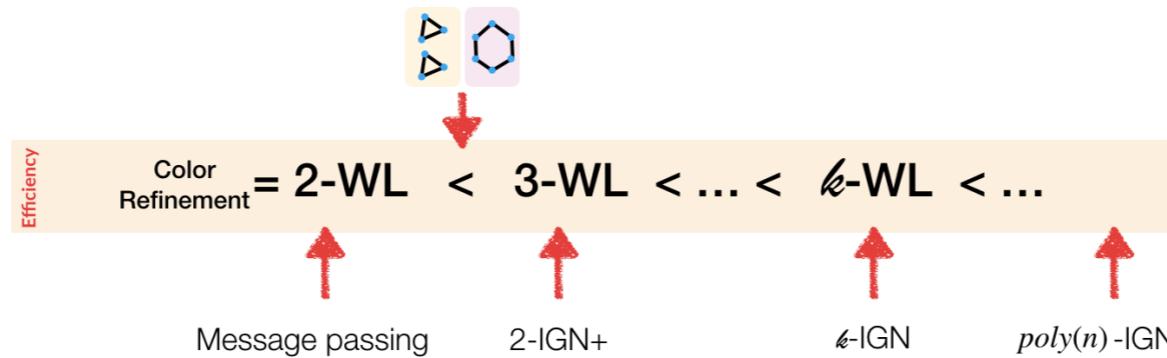
2. Более того если мы в функцию ϕ подадим просто сумму f от эмбеддингов соседей и текущего узла вместо той сложной композиции из "Шага 3" (показываем формулу на предыдущем слайде из 1-го условия). То мы по-прежнему сможем получить свертку равную по мощности WL-тесту.

У этой леммы на самом деле очень постое доказательство, если убрать все формальности то они в качестве доказательства просто приводят два примера такой новой f , которую можно суммировать.

И последние два шага. Выделяем из суммы представление текущего узла и умножаем на коэффициент важности информации текущего узла по сравнению с соседями. И затем вместо функции ϕ просто ставим многослойный перцептрон с достаточным числом слоев. Имеем на это право благодаря одной из теорем Вейерштрасса, что любую непрерывную функцию можно приблизить полиномом. А значит и MLP при достаточном числе слоев.

Этот коэффициент в PyG по умолчанию вообще равен нулю и у нас снова получается просто сумма из 5-ой леммы. Но его можно менять как гиперпараметр. И что еще круче, можно обучать как параметр сетки.

k-WL GNNs



1. Можно применять WL Test другого порядка.

Например, 2-WL Test более мощный, он рассматривает пары вместо отдельных узлов. Чем больше k тем тест более мощный.

2. Есть GNN превосходящие 1-WL Тест, это k-WL GNNs, в частности k-IGN.

arXiv:2201.07083v2 [stat.ML] 1 Nov 2022
[https://irregulardeep.org/How-expressive-are-Invariant-Graph-Networks-\(2-2\)/](https://irregulardeep.org/How-expressive-are-Invariant-Graph-Networks-(2-2)/)

Есть ли более мощные тесты чем 1-WL?

Да, это например 2-WL. Или более общий вариант - k-WL Test.

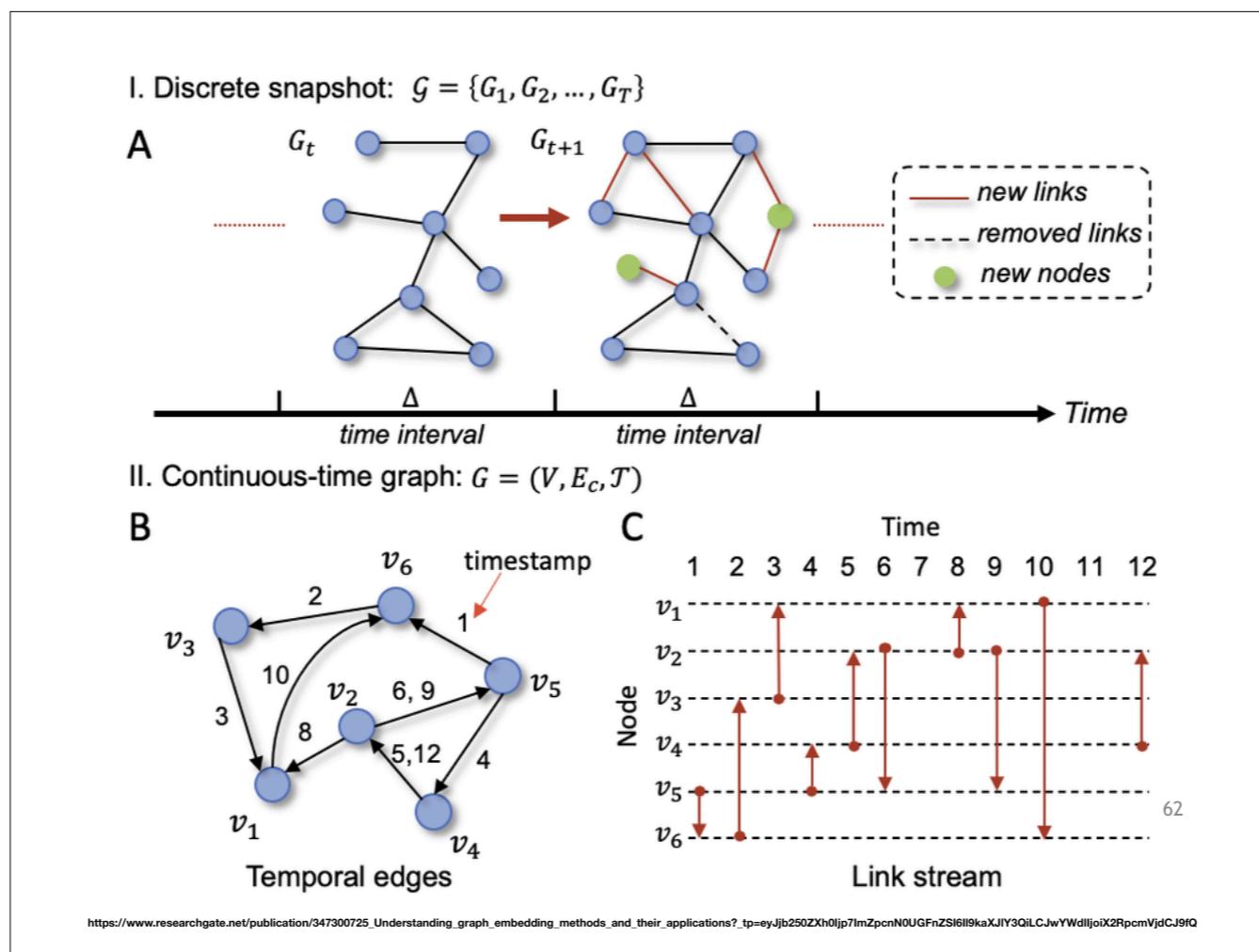
И чем больше k число в k-WL тесте, тем он мощнее.

И также есть графовые свертки более мощные чем 1-WL. Например k-IGN по мощности равна k-WL тесту.

Здесь еще не хватает рассуждений:

- 1) А нужны ли такие мощные графовые свертки?
- 2) Если мы просто в каждый узел к признакам добавим 1-hot вектор , не исчезнет ли эта проблема сама собой?
- 3) Есть проблемы с вычислительными затратами по сравнению с обычными message passing свертками?

Dynamic GNN



Бывает ситуация что граф, меняется со временем и нужно как-то это учитывать.

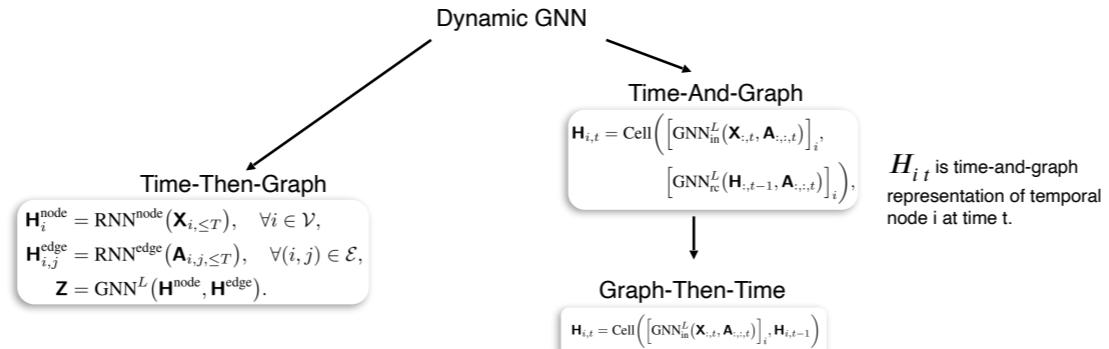
Самый простой способ, попробовать проигнорировать что график меняется и по-прежнему обучать его как статический.

Далее есть варианты упрощенно закодировать время. Например можно ребра разных моментов времени по-разному взвешивать. Чем дальше ребро от текущего момента, тем меньше его вес. Можно закодировать время на уровне признаков, добавив например время в виде PositionalEncoding.

И еще можно применить динамические графовые нейронные сети. Они бывают двух видов:

- Дискретные. Время разбито на интервалы, каждый интервал описывается статическим графиком
- Непрерывные. Мы знаем в какой момент времени произошло то или иное событие и для таких сетей график представляет собой такой поток событий. Например, появление / исчезновение узла или ребра, изменение в признаках.

Объединение: графов и времени



Cell (...) - is a RNN cell embedding evolutions of those GNN representation

<https://arxiv.org/abs/2103.07016>

Нам нужно как-то объединить информацию о динамике и информацию графовой топологии. Как это сделать?

Есть два подхода:

1. Time-Than-Graph
2. Time-And-Graph

В Time-Than-Graph мы сначала копируем динамику, а затем информацию о графе. (Описываем формулу на слайде.)

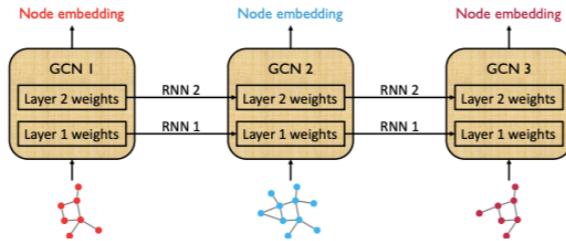
Time-And-Graph - устроена чуть сложнее. (Описываем смысл формулы на слайде.)

И есть еще Graph-Then-Time - частный случай Time-And-Graph. (Описываем отличие второго аргумента на слайде.)

EvolveGCN

$$\underbrace{W_t^{(l)}}_{\text{hidden state}} = \text{GRU}(\underbrace{H_t^{(l)}}_{\text{node embeddings}}, \underbrace{W_{t-1}^{(l)}}_{\text{hidden state}})$$

$$H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$$



Грубый алгоритм модели

W - матрица весов
snapshots - снэпшоты графа

```
1 def forward(snapshots, W):
2
3     H = [] # кэш эмбеддингов для слоя
4     for graph in snapshots:
5         H.append(graph.node_features)
6
7     for l in layers:
8         for i, g in enumerate(snapshots):
9             W = RNN[l](H[i], W)
10            H[i] = GCONV(g, H[i], W)
11
12    return H[-1]
```

[arXiv:1902.10191](https://arxiv.org/abs/1902.10191): EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs

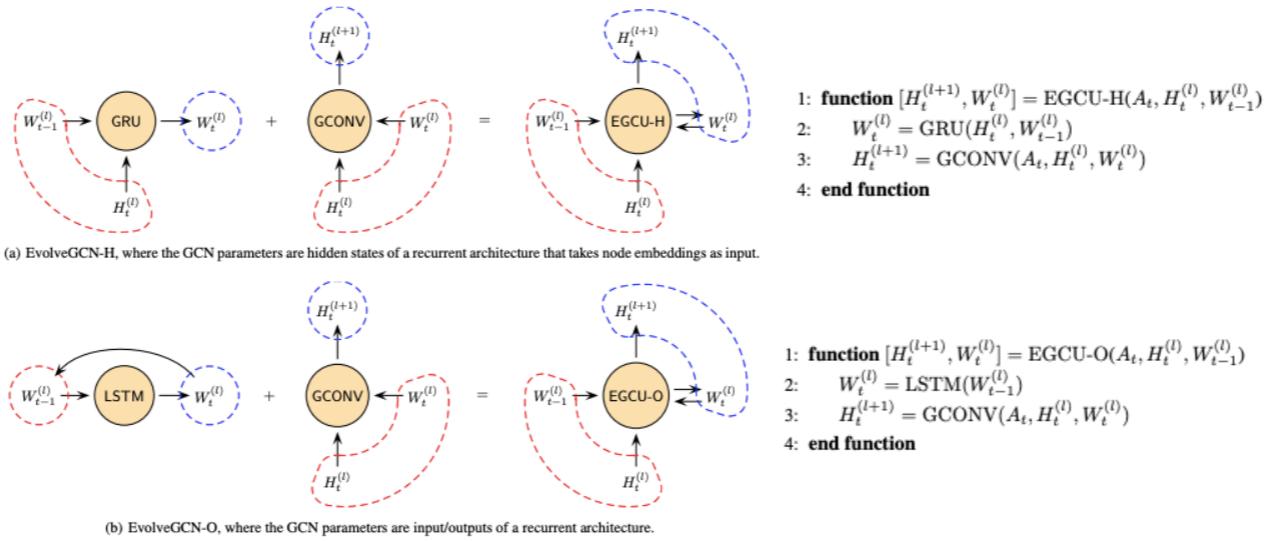
Рассмотрим такой пример динамической графовой нейронной сети как EvolveGCN.

Это дискретная графовая нейронная сеть. Показываем на примере слева.

По способу объединения времени и графов это Graph-Then-Time. Далее просто показываем схожесть с формулой из предыдущего слайда.

Далее описываем алгоритм справа, используя номера строк в качестве указателя.

EvolveGCN: O vs H



[arXiv:1902.10191](https://arxiv.org/abs/1902.10191): EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs

В статье описано два варианта реализации EvolveGCN:

1. EvolveGCN-H. Это довольно интересный вариант, что когда мы кодируем динамику через RNN и кодируем граф через GCN, то и для RNN и для GCN используется одна и та же общая матрица весов.

А в качестве входных данных в RNN подаются эмбеддинги узлов.

2. EvolveGCN-O. Здесь идея проще, матрица весов от GCN просто подается на вход в RNN, который выдает новую версию матрицы весов для GCN.

Авторы задались вопросом какой вариант лучше. Ответ такой.

Выбор правильной версии зависит от набора данных. Когда признаки узла информативны, версия -H может быть более эффективной, поскольку она включает дополнительное встраивание узлов в рекуррентную сеть.

С другой стороны, если характеристики узла не очень информативны, но структура графа играет роль более важную роль, версия -O фокусируется на изменении структуры и может быть более эффективной.

**Спасибо за
внимание!**

Ссылки

- https://mdpi-res.com/d_attachment/futureinternet/futureinternet-11-00112/article_deploy/futureinternet-11-00112-v2.pdf?version=1558954241
- https://openaccess.iyte.edu.tr/bitstream/11147/9418/1/Application_Areas.pdf
- <https://www.biorxiv.org/content/10.1101/2023.04.02.534383v1.full>
- <https://arxiv.org/abs/2212.10207>
- KkkX. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. In Proceedings of the IEEE International Conference on Computer Vision, pages 5199–5208, 2017. 2D. Xu, Y. Zhu, C.
- B. Choy, and L. Fei-Fei. Scene graph generation by iterative message passing. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5410–5419, 2017. 2
- J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh. Graph rcnn for scene graph generation. In Proceedings of the European Conference on Computer Vision (ECCV), pages 670– 685, 2018. 2
- Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang. Factorizable net: an efficient subgraph-based framework for scene graph generation. In Proceedings of the European Conference on Computer Vision (ECCV), pages 335–351, 2018. 2
- <https://github.com/senadkurtisi/pytorch-GCN/tree/main>
- <https://arxiv.org/pdf/2212.10207.pdf>
- <https://graph-neural-networks.github.io/static/file/chapter20.pdf>
- <https://cv-blog.ru/?p=341>
- <https://www.youtube.com/watch?v=kBZVZbKtIk4>
- https://www.researchgate.net/publication/343471056_Graph_Signal_Processing_for_Geometric_Data_and_Beyond_Theory_and_Applications?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZS16Ii9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCj9fQ

Appendix A: Учебные материалы

- tg: @sberlogawithgraphs
- Graph ML: <http://networksciencebook.com/>
- Graph DL: <https://web.stanford.edu/class/cs224w/>
- Graph ML/DL: <https://youtu.be/1T5-BG6yngM?si=RYNOwSTFwleuuptB>
- Network Science: https://youtu.be/hNEND_OMIL4?si=rJMvGeDIMCVznWwO
- <https://youtu.be/TLVhGs2ckPY?si=C-JvvNie6rTCIb5r>
- <https://arxiv.org/abs/2212.10207>

Appendix B: Инструментарий

NetworkX

Манипуляции с графами, классический
GraphML

iGraph

Манипуляции с графами, классический
GraphML

GraphX

Манипуляции с графами, классический
GraphML

PytorchGeometric

Нейронные сети

DGX

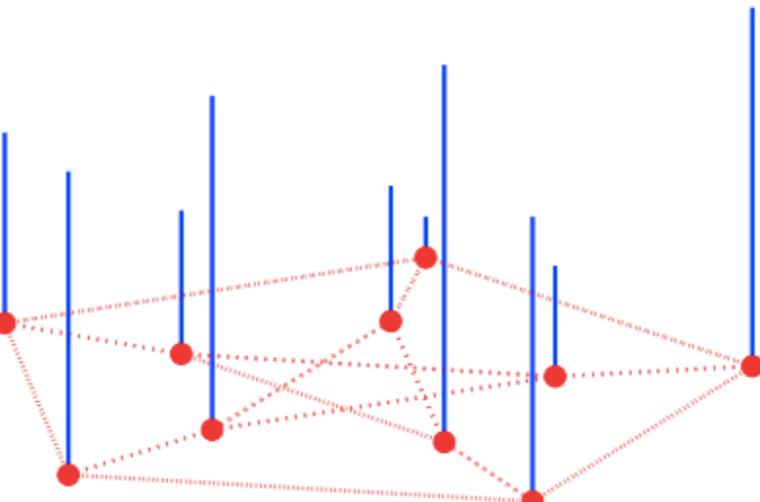
Нейронные сети

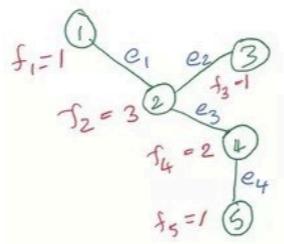
Appendix C: Laplacian Matrix of a Graph

На случай вопросов про лапласиан

Graph function

$f: V \rightarrow \mathbb{R}$





$$f = \begin{bmatrix} 1 & f_1 \\ 3 & f_2 \\ 1 & f_3 \\ 2 & f_4 \\ 1 & f_5 \end{bmatrix}_{5 \times 1} \quad K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}_{5 \times 4}$$

$$\text{grad}(f) = K^T f = \begin{bmatrix} -2 \\ 2 \\ 1 \\ 1 \end{bmatrix}_{4 \times 1} \begin{array}{l} e_1 : f_1 - f_2 \\ e_2 : f_2 - f_3 \\ e_3 : f_2 - f_4 \\ e_4 : f_4 - f_5 \end{array}$$

$$L = D - W$$

$$g = \begin{bmatrix} -2 \\ 2 \\ 1 \\ 1 \end{bmatrix}_{4 \times 1} \quad \text{div}(g) = Kg = \begin{bmatrix} -2 \\ 5 \\ -2 \\ 0 \\ 1 \end{bmatrix}_{5 \times 1}^{\textcolor{red}{v_1}}_{\textcolor{red}{v_2}}_{\textcolor{red}{v_3}}_{\textcolor{red}{v_4}}_{\textcolor{red}{v_5}}$$

$$KK^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}_{5 \times 4} \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}_{4 \times 5} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -3 & 1 & -1 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}_{5 \times 5} = L$$

<https://arxiv.org/pdf/1211.0053.pdf>