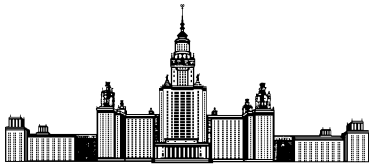
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

ЭССЕ СТУДЕНТА 317 ГРУППЫ

«Graph Neural Networks»

Выполнил:

студент 3 курса 317 группы

Висков Василий Алексеевич

Москва, 2021

Содержание

Аннотация

За последние 4 года область машинного обучения над графами обрела новую жизнь после нескольких публикаций. Востребованность данного направления очевидна, ведь все крупные социальные, компьютерные или, например, поисковые системы представимы в виде графа. В данном эссе раскрывается тема графовых нейронных сетей, одного из подходов к решению различных задач машинного обучения над этой нестандратной структурой данных.

При написании эссе автор вдохновлялся курсом CS224W [8] и блогом DeepFindr [4].

1 Введение

В этой части мы рассмотрим базовые понятия теории графов, опишем, почему вообще важно рассматривать графы как представление данных отдельно от остальных их типов и подведем обсуждение к графовым нейронным сетям (GNN).

1.1 Теория графов

Граф G - структура данных, моделирующая множество объектов и их связей между собой. Каждый объект отождествляется с одной из вершин $v \in V$, где V - их множество, связи между некоторыми двумя объектами описываются с помощью ребер $e = (v_1, v_2) \in E$, где E - их множество.

Определение 1. *$G = (V, E)$*

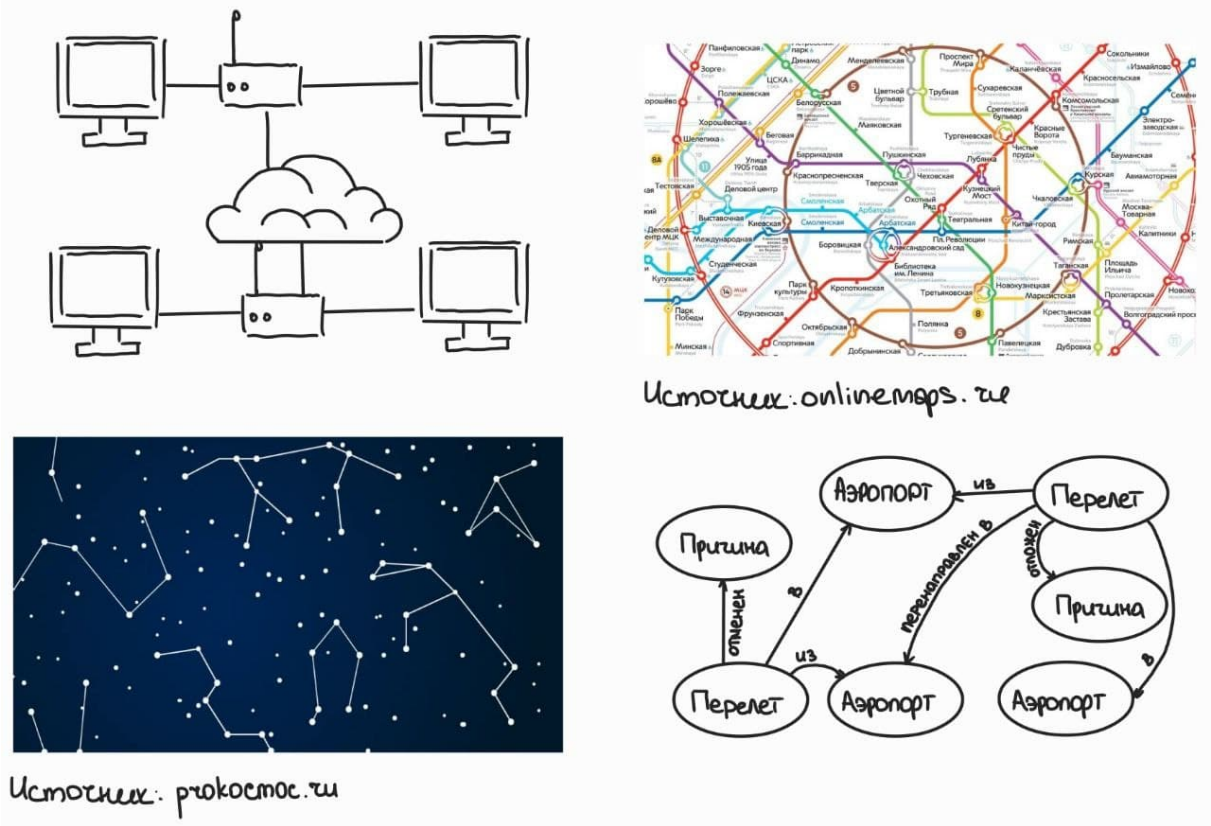


Рис. 1: Примеры систем, представимых в виде графа

В виде графа можно представить большое количество разнообразных систем: компьютерные сети, молекулы, 3D-фигуры, схемы финансовых операций, планы ме-

роприятия и т.д - причем представления эти по мощности множеств вершин и ребер могут быть внушительными. К примеру, количество активных пользователей социальной сети «ВКонтакте» на начало 2021 года составляло 74 млн. человек, больше половины всего населения РФ [1], а общее число связей между нейронами в мозгу человека может доходить до 100 трлн [2].

Порой степень связанности вершин может различаться, например, пропусная способность сети может варьироваться в зависимости от географического положения сервера. В таком случае каждому ребру $e = (v_i, v_j)$ можно приписать некоторый вес $w_{i,j}$, а такой граф называют *взвешенным*. В случае графа с равновзвешенными ребрами весами попросту можно пренебречь, т.к. нам важны не абсолютные, а относительные их величины, поэтому такой граф принято называть *невзвешенным*.

В введенном определении графа фигурирует множество пар вершин, которое обобщается до более привычной для прикладной математики сущности, матрицы.

Определение 2. ***Матрицей смежности*** $A = ||a_{i,j}||$ *графа* $G = (V, E)$ *называается матрица* $A_{[|V|\times|V|]}$, *в которой* $a_{i,j}$ *равняется:*

- *для невзвешенного графа: количеству ребер, соединяющих вершины* v_i *и* v_j*;*
- *для взвешенного графа: весу ребра* $w_{i,j}$

Вершины, имеющие общее ребро с рассматриваемой, будем называть *соседними*.

1.2 Типы задач МО для графов

Переходя к постановкам задач машинного обучения для графов, нам потребуется введение отображения из множества вершин и ребер в множество признаков. Для примера рассмотрим молекулу в виде графа, где вершинами являются атомы, а ребрами - связи между ними. Тогда признаками вершин могут выступать количество

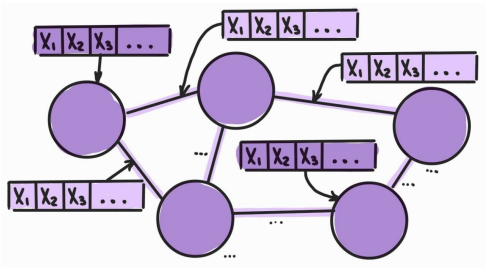


Рис. 2: Признаковое описание графа

[1] https://www.web-canape.ru/business/internet-i-socseti-v-rossii-v-2021-godu-vsya-statistika/

[2] https://theoryandpractice.ru/posts/17816-10-faktov-o-neyronakh

протонов, нейтронов и валентных электронов, а

признаками ребер - тип связи (ковалентная, донорно-акцепторная, . . .), типы связываемых молекул и др.

Различают три типа постановок задач МО для графов:

- node-level predictions - задачи предсказания откликов вершин;
- edge-level predictions (еще называют link predictions) - задачи предсказания откликов ребер;
- graph-level predictions - задачи предсказания отклика для всего графа, т.е. конкретного объекта;
- community detection - задачи кластеризации вершин графа;

В качестве примера на рисунке ниже приведены задача бинарной классификации вершины, задача бинарной классификации ребра в графе с различными типамм вершин и ребер, задачи бинарной классификации, регрессии (энтальпия принимает вещественнозначные значения) и кластеризации графа, а также задача кластеризации вершин одного графа (примером может служить задача сегментации пользователей социальной сети по сообществам на основе их профиля).

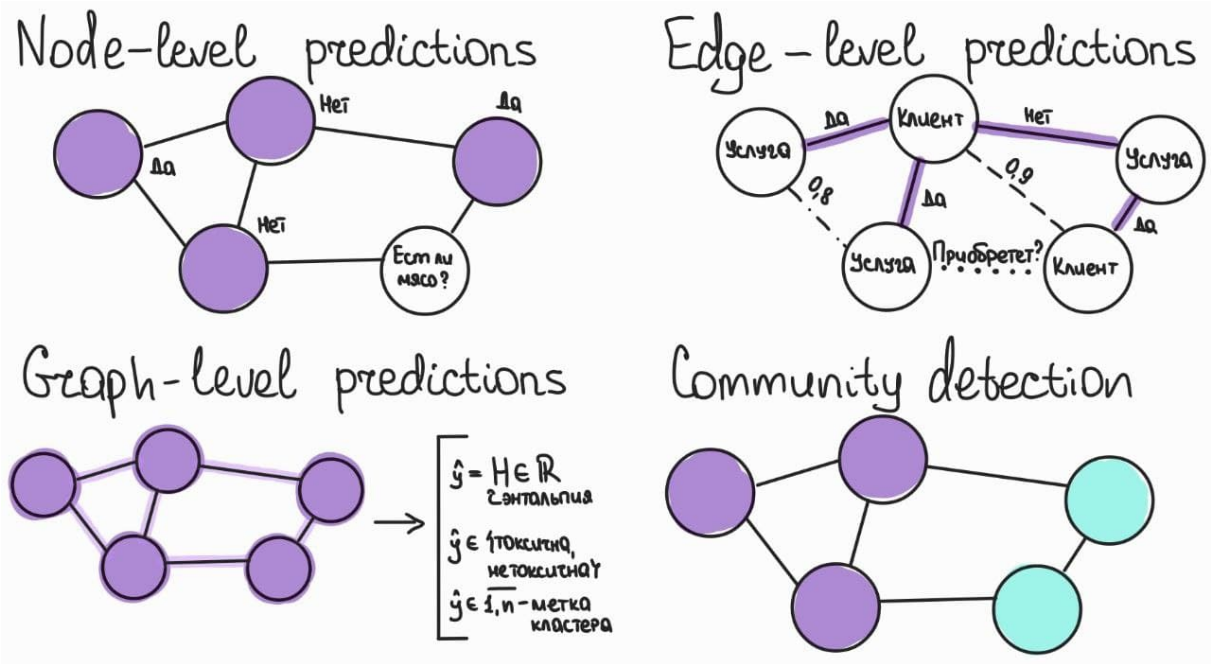


Рис. 3: Примеры задач МО для графов

1.3 Проблемы применения классических подходов

Имея определенную постановку задачи, мы можем искать решение среди известных нам нейросетевых моделей, таких как CNN, MLP ил RNN, но есть несколько фундаментальных отличий графовых представлений данных от других их типов, из-за которых нам необходимо искать новые подходы к построению моделей. Классические модели глубокого обучения разработаны для решения задач над линейными последовательностями данных: тексты, звук - и сеточнообазными данными, картинками. Но эти типы данных являются лишь частным, простым случаем графа.

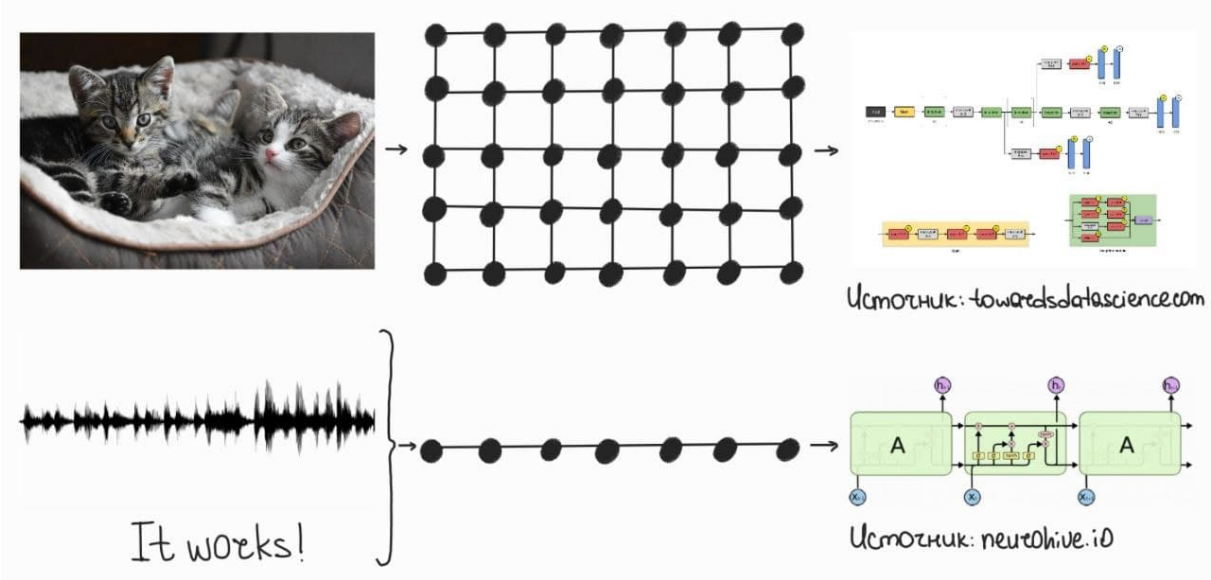


Рис. 4: Обобщение данных до графов

Итак, разберем основные отличия графовых данных от других их типов:

1. Размер и форма: при появлении новой вершины мы не сможем без знания о самих данных без больших потерь сохранить прежние размеры, как это, например, возможно в случае картинок, ведь для них определены простые операции обрезки и масштабирования.

2. Изоморфизм (инвариантность к перестановкам вершин): отсутствие ориентаций «лево, право, верх и низ» у графа как таковых.

   Замечание. *Это сразу же делает невозможным применение* ***наивного подхода*** *с использованием матрицы смежности как входа для MLP: переста-*

новка двух строк влечет автоматическую перестановку столбцов и, соответственно, перепутывание признаков (тут же можно указать, что число параметров такой модели имеет асимптотику $O(|V|)$, что делает ее невероятно сложной).

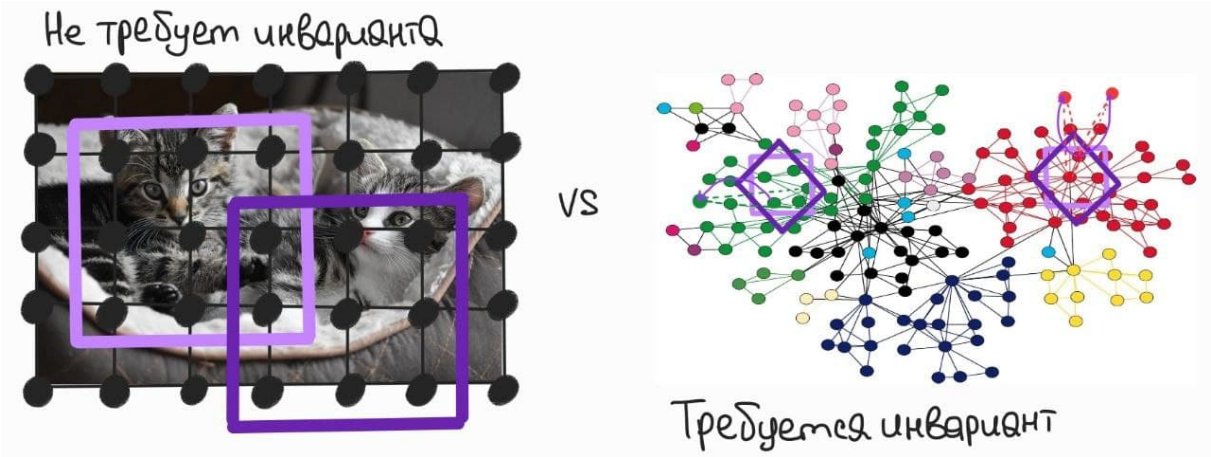


Рис. 5: Плавающее окно для изображения и графа

3. Неевклидова структура графа: требование изоморфизма приводит к отсутствию единственного положения в пространстве, из-за чего для графа нельзя ввести систему координат и задать метрику (поэтому область МО, связанную с графами, называют *Geometric DL*, что не позволяет, например, применять для графа механизм классической свертки и, следовательно, CNN).

1.4 Representation Learning

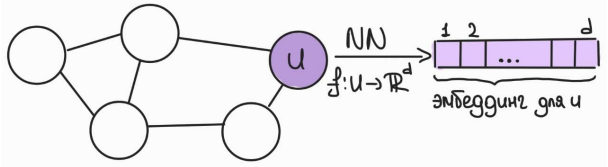


Рис. 6: Задача обучения представлений

В силу наличия фундаментальных проблем напрашивается и фундаментальная идея описываемой темы.

Нам необходимо найти удачное представление для графа, которое учитывало бы свойства этой структуры данных, хорошо описывало граф и которое бы вписывалось в уже имеющуюся теорию ML и DL в качестве признакового описания объекта.

Representation Learning (обучение представлений) – механизм автоматического извлечение и обучения признаков для вершин графа. Эти признаки будут содержать как информацию о самих вершинах, так и структурную информацию о взаимодействиях вершин друг с другом.

Способ использования этих признаков зависит от решаемой задачи:

- если решается node-level-задача или задача community detection, то мы можем использовать полученные признаки как есть;
- если решается edge-level-задача, то мы можем для каждого ребра определенным образом агрегировать представления инцидентных ей вершин;
- если решается graph-level-задача, то мы можем агрегировать представления всех вершин графа;

Размер эмбеддинга является гиперпараметром, но во многом зависит от размера исходного признакового описания вершин. Увеличения признакового описания можно добиться, например, конкатенацией результатов различных функций агрегации.

Если ребрам также приписаны некоторые признаки, то их можно или приписать всем инцидентным вершинам и решать после этого задачу обучения представлений или после этого этапа, обучаясь только на признаках вершин, векторы ребер можно внедрить в полученные эмбеддинги вершин.

2 Основная часть

В данном разделе мы рассмотрим все основные свойства GNN, способы построения, обучения и применения, изучим методы регуляризации и нормализации графовых нейронных сетей, а также изучим вопрос предела возможностей этого класса функций.

2.1 Графовая конволюция

Основные проблемы, возникающие при применении наивного подхода с использованием матрицы смежности для решения поставленной задачи – это невыполнение

свойства инвариантности графа к перестановкам вершин и колоссальное количество параметров. Проблема сложности модели возникала при изучении изображений как входных данных для нейросетевых моделей, и тогда решением оказалось применение конволюции. Исходя из того, что картинка является частным случаем графа, но имеет фиксированное положение и ориентацию, следующим шагом является поиск обобщения операции свертки за пределы множества простых решеток, которыми и являются изображения.

Идеей модификации конволюции является отход от фиксированной прямоугольной формы и переход к «топологической» свертке. Мы хотим использовать признаки соседних вершин и агрегировать их с некоторыми весами с признаками рассматриваемой вершине. Агрегируемую информацию с текущей и соседних вершин называют *сообщениями* (*messages*). После агрегации всех сообщений следует этап обновления эмбеддинга текущей вершины. Совокупность этапов построения сообщений и агрегации называется этапом *передачи сообщений* (*message passing*).

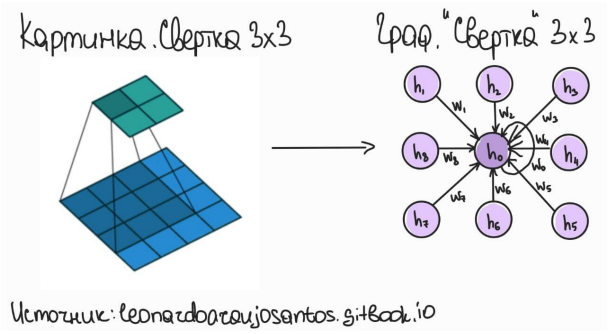


Рис. 7: Модификация свертки для графа

Замечание. *В случае ориентированного графа, когда не существует ребра, ориентированного в сторону какой-то вершины, можно произвести следующий трюк: дополним множество ребер противоположно ориентированными (**backwards edges**).*

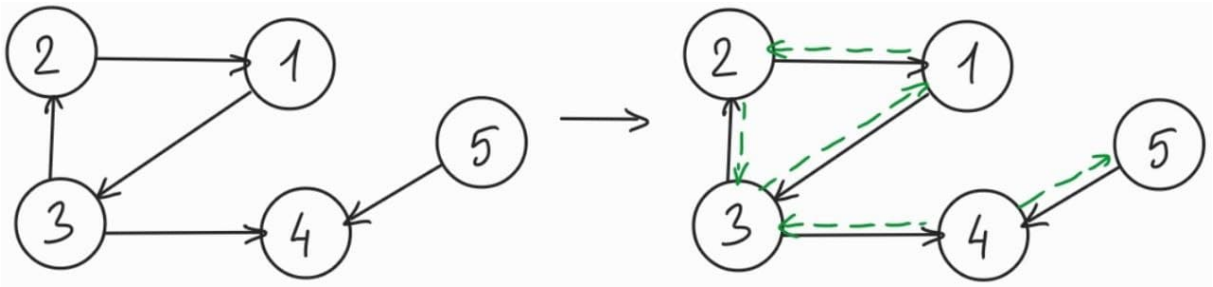


Рис. 8: Добавление развернутых ребер

2.2 Message Passing

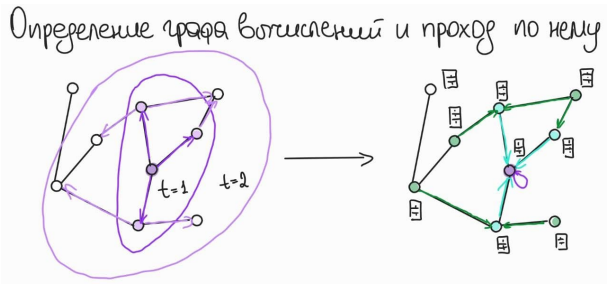


Рис. 9: Граф вычислений

В сущности на этапе k передачи сообщений текущая вершина получает информацию о вершинах на расстоянии в k ребер от нее. Зафиксировав некоторое число T_i для каждой вершины v_i (пусть везде далее для удобства будем рассматривать число шагов $T = T_i \quad \forall i \in 1, \ldots, |V|$), мы можем построить граф вычислений на основе локального соседства, а после пройтись по нему и получить конечное признаковое описание рассматриваемой вершины. И так можно сделать для любой вершины графа.

Если message passing этап на каждом шаге от 1 до T промоделировать некоторой нейронной сетью, например, однослойным перцептроном для каждой вершины, граф вычислений можно отождествить с полносвязной нейронной сетью. Тогда граф вычислений становится моделью MLP, в дальнейшем будем считать этот подход *базовым*.

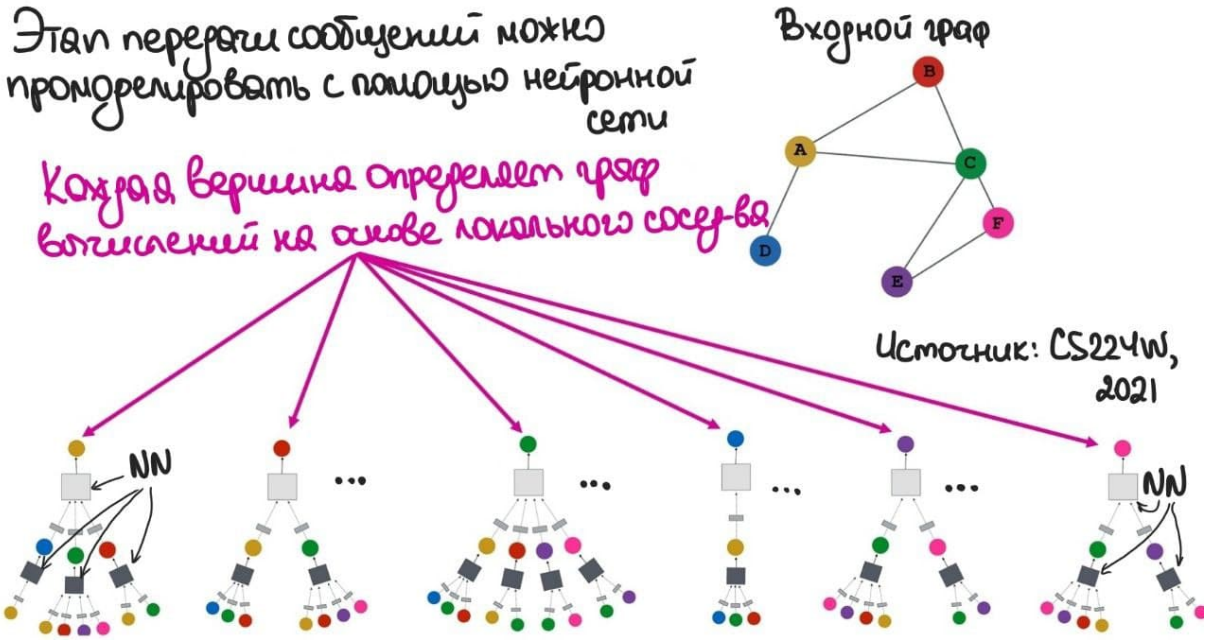


Рис. 10: Сети соседей определяют граф вычислений

Каждый слой нейронной сети, моделирующей этапы передачи сообщений, называют *слоями передачи сообщений* (*message passing layers*). На нулевом слое эмбеддинги каждой вершины v являются исходными признаковыми описаниями, а на слое k - векторами, полученными после прохода по k слоям нейронной сети (k *hops*).

Итоговая формула для эмбеддинга $h_v^{(k)}$ вершины v на шаге k выглядит следующим образом:

$$h_v^{(k)} = AGG^{(k)}(MSG^{(k)}(h_v^{(k-1)}), \quad \{MSG^{(k)}(h_u^{(k-1))} \quad \forall u \in \mathcal{N}(v)\}) \tag{1}$$

Здесь и далее $\mathcal{N}(v)$ - множество соседей вершины v. Можно сказать, что мы перешли от исходной задачи генерации признакового описания к $|V|$ независимым (в данном случае не учитываем возможные повторные вычисления) по каждому слою message passing задачам оптимизации композитных функций.

Заметим, что с помощью указанной выше формулы, задав соответствующие функции $MSG^{(k)}$ и $AGG^{(k)}$, мы можем получить все известные варианты message passing слоев (слоев GNN).

2.3 Базовый подход

Рассмотрим подробнее базовый подход к решению задачи и продемонстрируем возможные оптимизации вычислений.

Algorithm 1: GNN: basic approach

Data: $\forall v \in V: \quad h_v^1 = x_v, \quad T$– hyperparameter;

$f(x)$ – nonlinear differentiable function (ex., $\max(0, x)$);

Result: $\forall v \in V: \quad z_v = h_v^{(T)}$;

$k := 1$;

while $k \leq T$ **do**

  for $v \in V$ **do**

    $h_v^{(k+1)} = f(W_k \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(k)}}{|\mathcal{N}(v)|} + B_k h_v^{(k)})$

  end

end

Параметрами этой модели являются матрицы W_k и B_k для каждого шага k. Заметим, что эти веса не зависят от вершин и являются общими для них всех на одном и том же шаге. Выбрав некоторую функцию ошибки и алгоритм оптимизации, мы можем выучить эти параметры, о чем будет написано ниже.

Следующим шагом будет оптимизация этапа агрегирования за счет использования матричной нотации.

Пусть $H^{(k)} = [h_1^{(k)}, \dots, h_{|V|}^{(k)}]^T$, $D = \mathrm{diag}(|\mathcal{N}(v_1), \dots, \mathcal{N}(v_{|V|})|)$.

Тогда

$$\{\sum_{u \in \mathcal{N}(v)} h_u^{(k)}\}_v = AH^{(k)}$$

$$\sum_{u \in \mathcal{N}(v)} \frac{h_u^{(k-1)}}{|\mathcal{N}(v)|} \Rightarrow H^{(k)} = D^{-1}AH^{(k-1)}$$

Если принять $\hat{A} = D^{-1}A$, то

$$H^{(k)} = f(\hat{A}H^{(k)}W_k^T + H^{(k)}B_k^T)$$

На практике выясняется, что $\hat{A}$ - разреженная матрица, что позволяет применять эффективное перемножение разреженных матриц.

Замечание. *В случае нетривиальной функции агрегации не факт, что GNN может быть описана с помощью матричной нотации.*

2.4 Функция сообщения

$$m_v^{(k)} = MSG^{(k)}(h_v^{(k-1)}$$

Требования, предъявляемые к функции, состоят в дифференцируемости на рассматриваемом множестве.

Интуиция: каждая вершина создает сообщение, отправляемое всем соседним вершинам.

Пример: уже рассмотренное линейное преобразование $m_v^{(k)} = W^{(k)}h_v^{(k-1)}$

Для добавления сложности функция сообщения может быть композитной, внешней же функцией f могут служить $ReLU(\cdot)$, $\sigma(\cdot)$, ...

2.5 Функция агрегации соседей

$$h_v^{(k)} = AGG^{(k)}(m_v^{(k)}, \{m_u^{(k)} \quad \forall u \in \mathcal{N}(v)\})$$

Требования, предъявляемые к этой функции, состоят в дифференцируемости функции на рассматриваемом множестве и в симметричности по всем аргументам, т.е. она должна быть инвариантна к порядку последовательного агрегирования вершин.

Примеры:

- суммирование с весами $w(v)$;
- среднее арифметическое;
- максимум по каждому аргументу;

Добавленное среди аргументов сообщение от текущей вершины с предыдущего шага позволяет сохранять накопленную информацию. Обычно этот аргумент является трансформацией эмбеддинга с предыдущего слоя, причем преобразование отличается в плане используемых параметров от трансформаций эмбеддингов соседей. Иногда производят и нетривиальную агрегацию сообщения рассматриваемой вершины с сообщениями соседей, например, можно конкатенировать сагрегированные сообщения соседей с сообщением текущей вершины.

Для добавления сложности, как это было, например, в базовом подходе, можно использовать композитную функцию агрегации, внешней же функцией f могут служить $ReLU(\cdot)$, $\sigma(\cdot)$, ...

2.6 Примеры слоев GNN

Весь список архитектур можно изучить в этом подробном обзоре [18].

1. Graph Convolutional Networks [7] (его вариация рассматривалась выше как базовая)

$$h_v^{(k)} = f(W^{(k)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{h_u^{(k-1)}}{\sqrt{|\mathcal{N}(v)|}\sqrt{|\mathcal{N}(u)|}}$$

2. GraphSAGE [6]

$$h_v^{(k)} = f(W^{(k)} \cdot CONCAT(h_v^{(k-1)}, AGG(\{h_u^{(k-1)} \quad \forall u \in \mathcal{N}(v)\})))$$

Варианты использования функции AGG:

- средневзвешенное (вариант из GCN);
- $mean(\{MLP(h_u^{(k-1)}) \quad \forall u \in \mathcal{N}(v)\})$;
- $LSTM([h_u^{(k-1)} \quad \forall u \in \pi(\mathcal{N}(v))])$ (здесь $\pi(\cdot)$ - случаная перестановка множества);

Авторы советуют применять l_2–нормализацию для сохранения одинакового масштаба всех векторов;

3. Graph Attention Networks [14]

$$h_v^{(k)} = f(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^{(k)} h_u^{(k-1)})$$

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

$$e_{vu} = a(W^{(k)} h_u^{(k-1)}, W^{(k)} h_v^{(k-1)})$$

Варианты использования функции a:

- $Linear(Concat(\dots))$;
- $1LP(\dots)$;

Предлагается учить параметры модели a совместно с параметрами GNN end–to–end. Для стабилизации процесса обучения можно применять механизм *multu–head attention*: параллельно рассчитываются несколько вариантов эмбеддингов $h_v^{(k)}[i], \quad i = \overline{1, M}$, а после конечный результат получается агрегированием всех вариантов.

4. Graph Isomorphism Network [15]

$$h_v^{(k)} = MLP^{(k)}((1 + \varepsilon^{(k)}) h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)})$$

GIN имеет наибольшие возможности в плане решения задач на графах (но, как описано в статье, в классе GNN это не единственная функция, обладающая этим свойством).

	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	**86.0 ± 1.8** *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	**92.6 ± 4.2** *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	SUM–MLP (**GIN-0**)	**75.1 ± 5.1**	**52.3 ± 2.8**	**92.4 ± 2.5**	**57.5 ± 1.5**	**80.2 ± 1.9**	**89.4 ± 5.6**	**76.2 ± 2.8**	**64.6 ± 7.0**	**82.7 ± 1.7**
	SUM–MLP (**GIN-ϵ**)	**74.3 ± 5.1**	**52.1 ± 3.6**	**92.2 ± 2.3**	**57.0 ± 1.7**	**80.1 ± 1.9**	**89.0 ± 6.0**	**75.9 ± 3.8**	63.7 ± 8.2	**82.7 ± 1.6**
	SUM–1-LAYER	74.1 ± 5.0	**52.2 ± 2.4**	90.0 ± 2.7	55.1 ± 1.6	**80.6 ± 1.9**	**90.0 ± 8.8**	**76.2 ± 2.6**	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	**52.3 ± 3.1**	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	**66.6 ± 6.9**	80.9 ± 1.8
	MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

Рис. 11: Результаты экспериментов, проведенных в исследовании

5. LA–GCN [9]

$$h_v^{(k)} = f(W^{(k)}(h_v^{(k-1)} + s_v^{(k-1)}))$$

$$s_v^{(k-1)} = \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} * m_u^{(k-1)}$$

$$m_u^{(k-1)} = MLP^{(k)}(CONCAT(h_v^{(k-1)}, h_u^{(k-1)})) \in [0, 1]$$

Авторы заметили, что наиболее удачным способ агрегации является суммирование сообщений (что показано в предыдущей статье, в целом), они внедрили модель мета-обучения, представленную в статье в виде полносвязной нейронной сети, которая по конкатенированному эмбеддингу текущей и соседней вершины выдает, можно сказать, вес внимания;

Table 1: Node classification accuracy (%)

Methods	Cora	Citeseer	PubMed	Reddit
GCN	88.0	77.8	86.8	93.0
GAT	80.4	75.7	85.0	–
FastGCN	85.0	77.6	88.0	93.7
GraphSAGE_mean	82.2	71.4	87.1	94.6
LGCL	86.9	77.5	84.1	–
MixHop	88.3	73.9	85.6	–
LA-GCN	**89.1 %**	**78.7%**	**89.1%**	**95.1%**

Table 2: Graph classification accuracy (%)

Methods	MUTAG	PROTEIN	PTC
WL subtree	90.4	75.0	59.9
DCNN	67.0	61.3	56.6
PATCHYSAN	**92.6**	75.9	60.0
DGCNN	85.8	75.5	58.6
AWL	87.9	–	–
GIN	89.4	76.2	64.6
LA-GCN	90.0	**80.5**	**72.2**

Рис. 12: Результаты экспериментов, проведенных в исследовании

2.7 Композицирование слоев GNN

После изучения популярных архитектур слоев GNN нам необходимо понять, каким образом можно последовательно (или не последовательно) соединять слои GNN для образования полноценной модели.

Простейшим методом здесь будет обыкновенное «наслаивание» одинаковых слоев друг на друга, но при выборе слишком большого параметра T эмбеддинги к концу процесса обучения признаков получатся неотличимыми друг от друга. Такое явление называется *over–smoothing* (*пересглаживание*). Проблема возникает из-за того, что локальные сети соседей (*receptive fields*, *области чувствительности*) нескольких вершин начинают пересекаться по очень большому числу элементов с ростом числа шагов, что и приводит в конечном итоге к одинаковым пределам последовательностей эмбеддингов. Бороться с этим эффектом можно следующими способами:

- поиск баланса между областью чувствительности каждой вершины и числом слоев в GNN;
- усложение моделей, включенных в слой GNN;
- добавление слоев, которые не передают сообщения: эти слои могут отвечать за предобработку данных (важно для кодирования признаков вершин) или за постобработку (важно при необходимости анализа или трансформации вершин) - и это отлично работает на практике;

- добавление skip connections, основавывается на наблюдении, что эмбеддинги в первых слоях GNN лучше различают вершины - добавление shortcut–ов позволяет увеличивать вклад первых слоев при предсказании конечных эмбеддингов и создает т.н. *смеси моделей*;

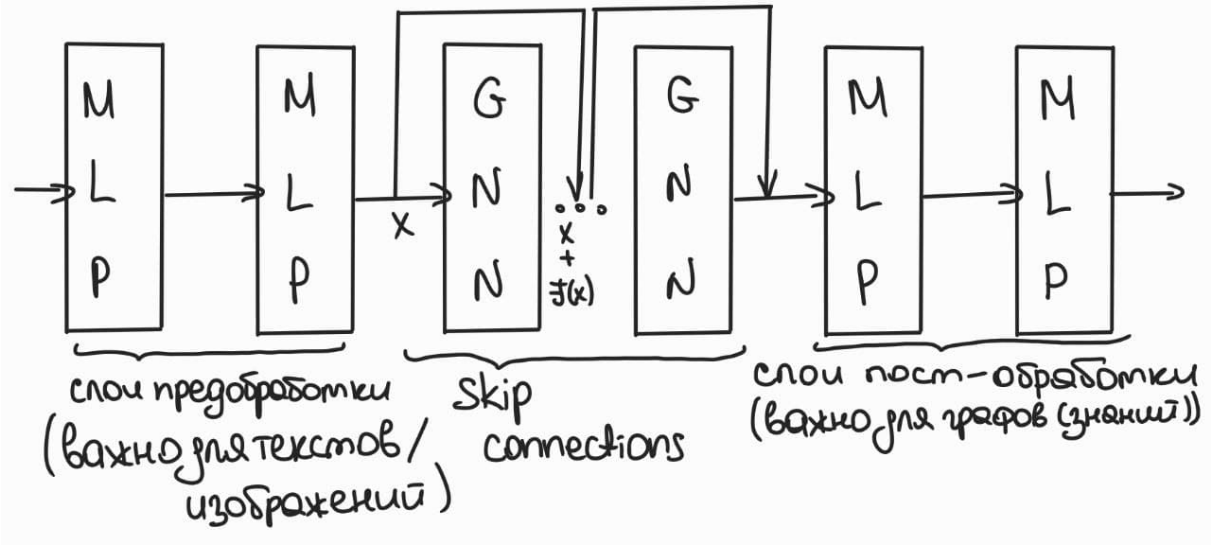


Рис. 13: Добавление новых слоев и skip connections

Замечание. *GCN с skip connection:*

$$h_v^{(k)} = f\left(\sum_{u \in N(v)} W^{(k)} \frac{h_u^{(k-1)}}{|N(v)|} + h_v^{(k-1)}\right)$$

возможны и другие варианты skip connections;

Существуют и другие методы борьбы с эффектом over–smoothing [17] [12].

2.8 Аугментации графа

Основная идея, стоящая за этим подразделом: исходный граф и граф вычислений - не тождественные понятия. Везде ранее мы строили граф вычислений, напрямую опираясь на изначальную структуру графа. Но есть ряд причин, по которым имеет смысл найти новые методы построения:

- иногда в данных присутствуют пропуски;
- граф может быть слишком разреженнным, что приводит к неэффективной процедуре message passing;

- граф может быть слишком плотным, что приводит к слишком дорогой процедуре message passing;
- граф может быть слишком большим, из-за чего мы не сможем обучить его на GPU;

Рассмотрим несколько подходов к аугментации данных:

1. Пропуски в данных → аугментация признаков.

   Ситуация может возникать в случаях, когда нам дана только матрица смежностей. Стандартными подходами тут являются приписывание константных величин вершинам и one–hot кодирование вершин.

   Другая ситуация связана с наличием в графе циклов или змеевидных подграфов, для вершин которых сложно выучить эмбеддинги. Решением в данном случае будет взятие в качестве признаков вектора, описывающего количество циклов определенной длины в каждой ячейке (от 0 до заданного N).

2. Граф слишком разреженный → добавление виртуальных вершин или ребер. Стандартными подходами являются добавление виртуальных вершин, которые соединяются со всеми вершинами в графе, или соединение всех соседей на расстоянии 2, что подразумевает, на самом деле, замену матрицы A на $A + A^2$ при расчетах в GNN.

3. Граф слишком плотный → сэмплирование соседей при проходе по слою GNN для каждой вершины.

4. Граф слишком большой → сэмплирование подграфов для вычисления эмбеддингов.

2.9 Методы регуляризации и нормализации

На практике мы можем использовать классические модули глубокого обучения, осуществляющие регуляризацию и нормализацию. Мы рассмотрим несколько стратегий регуляризации: *Dropout*, *DropNode* [3] и *DropEdge* [12] - и стратегию нормализации *Batch Normalization*.

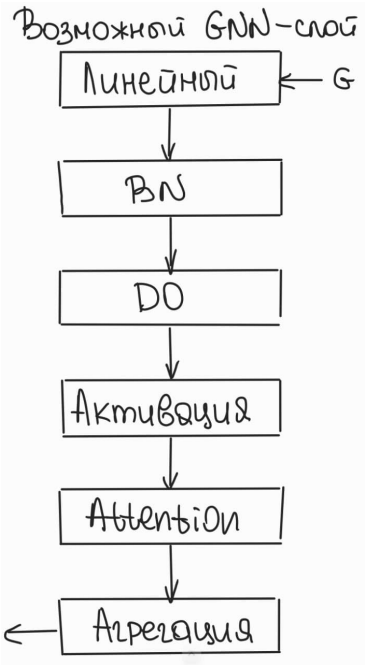


Рис. 14: Этапы трансформации данных

Batch Normalization.

Цель: стабилизировать процесс обучения.

Идея: стандратизируем батч входных данных (эмбеддинги вершин).

Вход: $X \in \mathbb{R}^{N \times D}$ - N эмбеддингов.

Выход: $Y \in \mathbb{R}^{N \times D}$ - стандартизованные эмбеддинги.

Обучаемые параметры: $\gamma, \beta \in \mathbb{R}^{D}$.

Первый шаг:

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

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

Второй шаг:

$$\hat{X}_{i,j} = \frac{X_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$Y_{i,j} = \gamma_j \hat{X}_{i,j} + \beta_j$$

Регуляризация

Цель: хотим избежать переобучения нейронной сети.

Dropout

Идея: во время обучения с некоторой вероятностью p отключать нейроны, во время теста учитывать смещение.

Обучение:

$$y = f(Wx) \cdot m$$

Тест:

$$y = (1 - p) f(Wx)$$

Замечание. *Dropout должен применяться к линейному слою в message функции.*

DropNode

Идея: случайно сэмплировать подграф из исходного графа на каждом шаге обучения с помощью следующего алгоритма:

1. Фиксируем величину $p \in (0, 1)$ - вероятность того, что вершина останется в подграфе.

2. Строим случайный вектор $r = (r_1, \ldots, r_{|V|} \quad | \quad r_i \sim Bern(p) \quad \forall i = \overline{1, |V|})^T$.

3. Отбрасываем вершины из V в соответствии со значениями случайного вектора. Если из графа удаляются две соседние вершины, ребро, инцидентное им, также удаляется.

4. Проводим процесс обучения на оставшемся подграфе.

Аналогично тому, что происходит в Dropout, на этапе применения необходимо шкалировать выход каждого слоя на величину $\frac{1}{p}$.

DropEdge

Метод регуляризации применим для моделей, содержащих в явном виде матрицу смежности (в статье рассматривается GCN).

Идея: случайно отбросить подмножество ребер графа, каждое из которых выбрано с определенной вероятностью p, на каждом шаге обучения.

Формальный алгоритм для каждого шага:

1. Фиксируем величину $p \in (0, 1)$ - вероятность того, что ребро будет отброшено.

2. Сэмплируем из $Bern(p)$ $|E|$ случайных величин, в соответствии с ними строим целочисленную матрицу A' с неотрицательными элементами, в которой ненулевой элемент находится только на месте существующего в E ребра. Ненулевые элементы равны сумме случайных величин, соответствующих этой ячейке. В случае отсутствия в графе параллельных ребер в среднем эта матрица будем содержать Vp ненулевых элементов.

3. $A_{drop} = A - A'$.

4. $\hat{A}_{drop} = \hat{D}^{-\frac{1}{2}}(A_{drop} + I)\hat{D}^{-\frac{1}{2}}, \quad \hat{D} = Deg(A_{drop} + I)$ - degree matrix (renormalization [7], опционально).

5. Полученная матрица используется в формулах для обучения.

На этапе инференса DropEdge не применяется.

2.10 Обучение

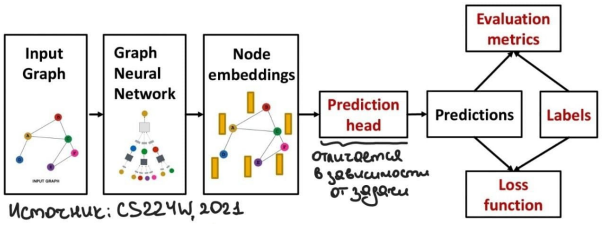


Рис. 15: Pipeline обучения

Последовательность действий при обучении GNN не отличается принципиально от последовательности для других моделей. В графовых нейронных сетях в конечном счете нам важен способ преобразования полученных эмбеддингов в предсказанные отклики, соответствующие задаче.

Как и в случае постановок различаю, различают три типа т.н. *prediction heads*: node–level, edge–level и graph–level.

Node–level. Пусть полученные эмбеддинги $z_v = h_v^T = g(G)$, где g - GNN, дифференцируемая функция, а G - исходный граф.

В случае задачи обучения с учителем мы будем оптимизировать эмпирический риск выборки

$$\sum_{v \in V} \mathcal{L}(y_v, f(z_v) \to \min_{\Theta}$$

Здесь Θ - совокупность параметров g и f. В случае задачи регресси в качестве $\mathcal{L}$ мы можем взять L_2, а в случае задачи классификации - кросс–энтропию.

В случае задачи обучения без учителя мы можем использовать структуру графа в качестве учителя, к примеру, $z_v^T z_u \approx sim(v, u)$, где функцию близости можно взять как веса соответствующих ребер. Можно воспользовать альтернативным подходом к обучению представлений и учиться на близостях других вариантов признаковых описаний:

- алгоритмы, основанные на случайных блужданиях (*randow walks*): node2vec [5], DeepWalk [11];
- матричная факторизация [1];

В общем виде задачу можно записать в следующем виде:

$$L = \sum_{z_u, z_v} CE(y_{u,v}, sim(z_u, z_v)),$$

где $y_{u,v} \in \{0, 1\}$ равняется единице тогда и только тогда, когда вершины u и v похожи. CE - кросс–энтропия, sim - функция близости (для простоты можно взять скалярное произведение).

Рассмотрим еще один пример. В [2] решается задача graph-level бинарной классификации в два этапа, первым из которых выступает этап обучения представлений с помощью триплетной функции ошибки:

$$\sum_{(a,p,n)} \max_{\Theta}(||g(G_a) - g(G_p)||_2^2 - ||g(G_a) - g(G_n)||_2^2 + \alpha, 0)$$

Обучение на такой функции ошибки называется обучением метрики, впервые использованной для обучения модели FaceNet [13].

Второй этап состоит в обучении модели бинарной классификации при фиксированных представлениях.

Edge–level. В данном случае для каждого ребра (v, u) нас будут интересовать сконкатенированные векторы (z_v, z_u) и построенное с помощью функции $Head_{edge}$ над ними некоторое предсказание. Первым подходом является применение линейного преобразования над ними, вторым подходом является применение функции $Head_{edge}(z_u, z_v) = z_u^T W z_v$, W может быть как априорной (единичной, например), так и обучаемой матрицей.

Graph–level. Способ предсказания по эмбеддингам схож с этапом агрегации в слое GNN, поэтому тут применимы те же варинты, которые мы рассматривали для функции агрегации.

Рассмотрим и другой способ агрегации, называемый *DiffPool* [16]. Идея состоит в применении *hierarchical global pooling*, способе агрегации эмбеддингов, результот которой будет зависить от порядка агрегации (например, можно использовать функцию $Head_{graph} = ReLU(\sum(\dots))$. Будем строить модель последовательной агрегации, где на каждом уровне нам необходимо получить модель, которая бы сообщала, какие эмбеддинги вершин необходимо агрегировать. Идея *DiffPool* состоит в постоении иерархической модели, где на каждом уровне параллельно строятся модель для решения задачи *community detection* и модель вычисления эмбеддингов. Первая модель, строя кластеризацию вершин внутри графа, будет сообщать нам, как правильно агрегировать эмбеддинги вершин.

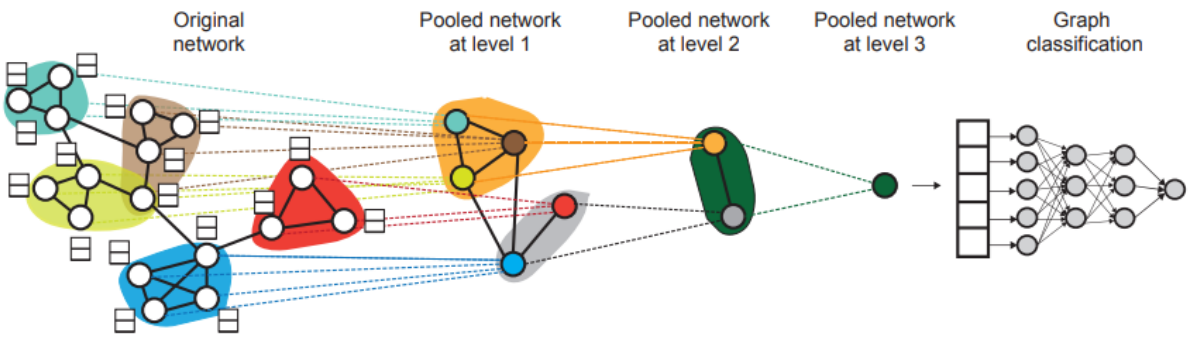


Рис. 16: Модель *DiffPool*

Отдельно поговорим о методе оптимизации модели, а именно о SGD. Для его применения на каждом шаге мы можем оптимизировать GNN не по всему множеству V, а по некоторому батчу мощности $m < |V|$. Возможно это потому, что нам удалось декомпозировать исходную задачу до оптимизации некоторой композитной функции для каждой вершины графа. В остальном же методы оптимизации для GNN не отличаются от уже нам известных.

2.11 Индуктивная способность

Графовые нейронные сети в силу отсутствия зависимости параметров от конкретных вершин и наличия зависимости только от номера шага могут обобщаться на ранее ненаблюдаемые вершины или графы. Например, модель обучалась на белковой структуре одного организма A и выдала эмбеддинг z_A, а после может быть обобещна на другой организм B, для которого будет построен эмбеддинг z_B.

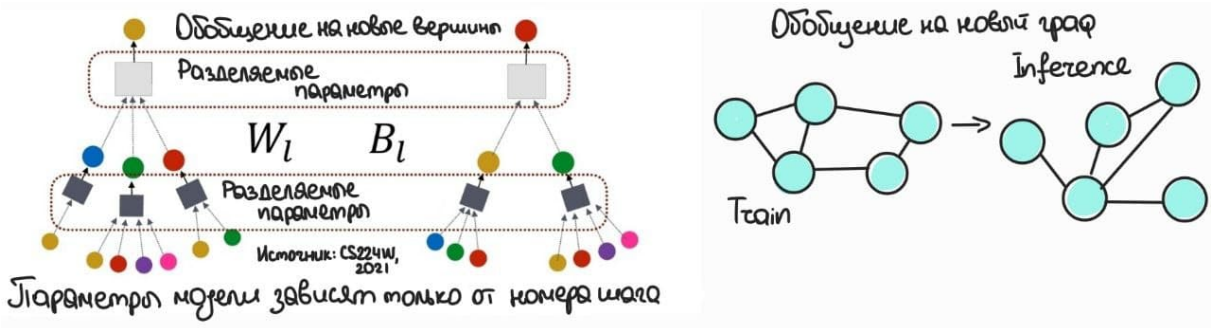


Рис. 17: Inductive Capability

Это свойство модели позволяет «на лету» строить эмбеддинги для вершин, пришедших к нам на этапе применения. Например, такой вершиной может выступить новый фильм на Netflix или новое видео на Youtube.

2.12 Валидация

Давайте сначала поймем, почему необходимо отдельно рассматривать валидирование GNN.

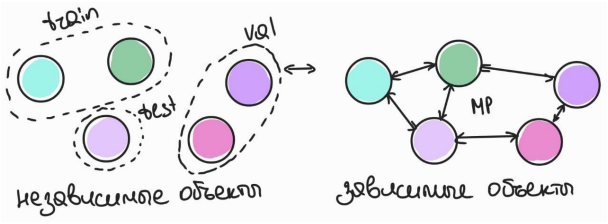


Рис. 18: Для GNN разбиение нужно строит иначе

Представим, что нам необходимо классифицировать 5 изображений. При использовании классических моделей мы предполагаем, что объекты независимы, поэтому можем каким-то образом разделить выборку на train/val/test. В случае же с GNN у нас есть априорная информация о взаимодействиях объектов, которую мы будем постоянно использовать при поиске эмбеддингов, поэтому выбранное в первом случае разбиение для второго оказывается неподходящим.

Первым подходом является т.н. *трансдуктивный способ разбиения*, когда модель наблюдает весь граф во время обучения, а делим мы на три множества только набор откликов для вершин.

Вторым подходом является т.н. *индуктивный способ разбиения*, основанный на способности модели обощаться на ранее ненаблюдаемые графы. То есть мы делим исходный граф на три подграфа, аналогично поступаем с множеством меток с сохранением соответствий, и проводим классический процесс обучения, валидации и тестирования на трех независимых подграфах.

Рассмотрим отдельно задачу *link prediction*. Нам необходимо «спрятать» от модели некоторые ребра и предложить ей их предсказать, т.е. ставится задача из ряда *self–training*. Ребра, которые подаются на вход модели, называют *message ребра*, а те, которые хотим предсказать, - *supervision ребра*.

Для оценки такой модели мы можем воспользоваться индуктивным способом разбиения в классическом варианте, для трансдуктивного же способа нам потребуется

выделить валидационные и тестовые множества, а также из обучающего множества выделить supervision ребра. На изображении ниже приведено наглядное описание того, как нужно на каждом этапе использовать ребра графа (цвета частей прямоугольников соответствуют ребрам графа, пунктирная линия, разделяющая цветные части прямоугольников, «разделяет» ребра на обучающую и тестовую выборки на конкретном этапе (слева обучающее множество, справа тестовое)).

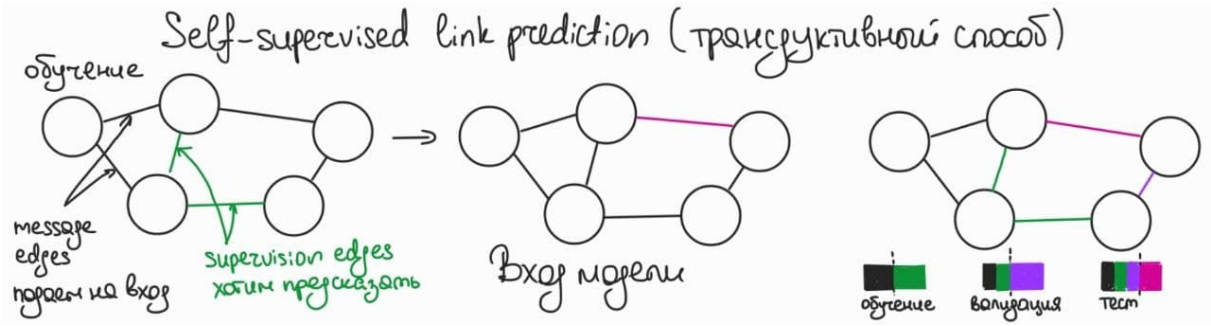


Рис. 19: Self-supervised link prediction

2.13 Возможности GNN

Мы хотим понять, где находится предел применимости графовых нейронных сетей, оценить возможности модели по разделению различных графовых структур и как построить наиболее «сильную» модель.

Рассмотрим модельную задачу.

Пусть нам дан граф, вершины которого являются неотличимыми, и мы хотим понять, сможет ли GNN по структуре графа различить локальные структуры соседей при наличии отличий и где находится предел возможностей модели.

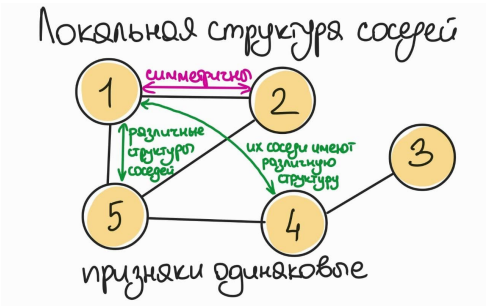


Рис. 20: Локальная структура соседей

Основной сущностью, которая поможет ответить на поставленные вопросы, является граф вычислений. Его построение основано на локальной сетке соседей. Но в модельной задаче граф вычислений для вершин 1 и 2 будет одинаковым для любого выбора T. Это означает, что без каких-либо признаков GNN построит одинако-

вые эмбеддинги для этих вершин и не сможет их отличить.

GNN, обладающие наибольшими возможностями по решению задач на графах, должны отображать различные вершины с различными графами вычислений, в сущности являющиеся корневыми поддеревьями, в различные эмбеддинги (на рисунке отображены цветом).

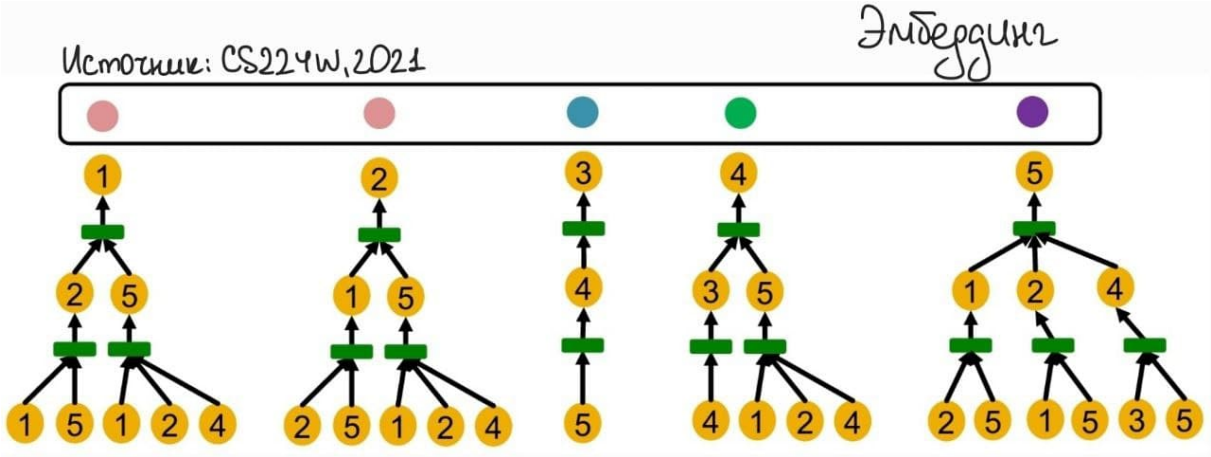


Рис. 21: Отображение графов вычислений в множество эмбеддингов

Основным наблюдением в этой задаче является тот факт, что наличие отличий в графах вычислений хотя бы на одном уровне позволяет посредством рекурсии различать их полностью. Так как на каждом слое GNN единственным механизмом, позволяющим нам отличать различные совокупности вершин на одинаковых уровнях графов друг от друга, является функция агрегации, нам необходимо найти такую функцию, которая бы имела способность отображать различные объекты в различные элементы. Такие функции называются *инъективными*.

Перейдем к поиску наиболее «сильного» варианта GNN.

В [8] подробно разбираются случаи, в которых рассмотренные выше архитектуры GCN и GraphSAGE не отрабатывают верным образом.

Как уже упоминалось, GIN обладает наибольшими возможностями в плане решения задач МО на графах. Запишем ее в ином виде.

$$MLP_{\Phi}(\sum_{x \in S} MLP_f(x))$$

Можно показать, что выполнена следующая теорема, из которой и следует, что GIN является наиболее «сильной» функцией в классе функций GNN.

Теорема 2.1. *Функция агрегации GIN является инъективной, то есть отображение различных вершин происходит в различные элементы множества эмбеддингов.*

Замечание. *Ключ к успеху - в применении в качестве функции агрегации поэлементного суммирования вместо усреднения и поэлементной максимазиции.*

2.14 Актуальные результаты приложения

Список взят из следующей статьи [3].

1. рекомендательные системы:
    - Uber Eats использовал GraphSAGE для рекомендаций блюд;
    - Alibaba использует Aligraph [19], собственную разработку, для построения за несколько минут коллосальных по количеству вершин графов;
2. задачи дискретной оптимизации:
    - DeepMind использовал GNN в задаче, связанной с MILP (mixed-integer linear program) solver и обогнал по скорости существующие решения [10];
3. физика/химия:
    - исследователи из DeepMind применили GNN для симуляции сложной динамической системы частиц (например, песка или воды);
    - ученые из Fermilab проводят работы по внедрению GNN в актуальные исследования на БАК;

3 Заключение

Мы рассмотрели класс подходов к решению задач машинного обучения на графах, именуемый графовыми нейронными сетями, показали, что в классе функций

[3] https://bit.ly/3yQleWR

GNN существуют инъективные отображения, позволяющие достигать идеальных результатов построения эмбеддингов (по крайней мере, на бумаге). Но, как и у любой модели, у GNN есть ряд недостатков в силу специфики работы модели, поэтому применение графовых нейронных сетей необходимо подкреплять другими графовыми моделями не из класса нейросетевых функций.

Список литературы

[1] D. Cai и др. “Graph Regularized Non-negative Matrix Factorization for Data Representation”. в: (2010). URL: http://www.cad.zju.edu.cn/home/dengcai/Publication/Journal/TPAMI-GNMF.pdf.

[2] M. T. Do, N. Park и K. Shin. “Two-Stage Training of Graph Neural Networks for Graph Classification”. в: *arXiv:2011.05097v3* (2021). URL: https://arxiv.org/pdf/2011.05097.pdf.

[3] T. H. Do и др. “Graph Convolutional Neural Networks with Node Transition Probability-based Message Passing and DropNode Regularization”. в: *arXiv:2008.12578v2* (2021). URL: https://arxiv.org/pdf/2008.12578.pdf.

[4] Florian. *Understanding Graph Neural Networks*. 2020. URL: https://deepfindr.com/understanding-graph-neural-networks-part-1-3/.

[5] A. Grover и J. Leskovec. “node2vec: Scalable Feature Learning for Networks”. в: (2016). URL: https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf.

[6] W. L. Hamilton, R. Ying и J. Leskovec. “Inductive Representation Learning on Large Graphs”. в: *NIPS* (2017). URL: https://arxiv.org/pdf/1706.02216.pdf.

[7] T.N. Kipf и M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. в: *ICLR* (2017). URL: https://arxiv.org/pdf/1609.02907.pdf.

[8] J. Leskovec и др. *CS224W: Machine Learning with Graphs*. 2021. URL: http://web.stanford.edu/class/cs224w/.

[9] Z. Li и H. Lu. “Learnable Aggregator for GCN”. в: *NIPS* (2019). URL: https://grlearning.github.io/papers/134.pdf.

[10] V. Nair и др. “Solving Mixed Integer Programs Using Neural Networks”. в: *arXiv:2012.13349v1* (2020). URL: https://arxiv.org/pdf/2012.13349.pdf.

[11] B. Perozzi, R. Al-Rfou и S. Skiena. “DeepWalk: Online Learning of Social Representations”. в: *arXiv:1403.6652v2* (2014). URL: https://arxiv.org/pdf/1403.6652.pdf.

[12] Y. Rong и др. “DropEdge: Towards Deep Graph Convolutional Networks on Node Classification”. в: *ICLR* (2020). URL: https://openreview.net/pdf?id=Hkx1qkrKPr.

[13] F. Schroff, D. Kalenichenko и J. Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. в: *arXiv:1503.03832v3* (2015). URL: https://arxiv.org/pdf/1503.03832.pdf.

[14] P. Velockovic и др. “Graph Attention Networks”. в: *arXiv:1710.10903v3* (2018). URL: https://arxiv.org/pdf/1710.10903.pdf.

[15] K. Xu и др. “How Powerful are Graph Neural Networks?” в: *ICLR* (2019). URL: https://arxiv.org/pdf/1810.00826.pdf.

[16] R. Ying и др. “Hierarchical Graph Representation Learning with Differentiable Pooling”. в: *arXiv:1806.08804v4* (2019). URL: https://arxiv.org/pdf/1806.08804.pdf.

[17] L. Zhao и L. Akoglu. “PairNorm: Tackling Oversmoothing in GNNs”. в: *ICLR* (2020). URL: https://arxiv.org/pdf/1909.12223.pdf.

[18] J. Zhou и др. “Graph neural networks: A review of methods and applications”. в: *AI Open* (2020). URL: https://arxiv.org/ftp/arxiv/papers/1812/1812.08434.pdf.

[19] R. Zhu и др. “AliGraph: A Comprehensive Graph Neural Network Platform”. в: *arXiv:1902.08730v1* (2019). URL: https://arxiv.org/pdf/1902.08730.pdf.