

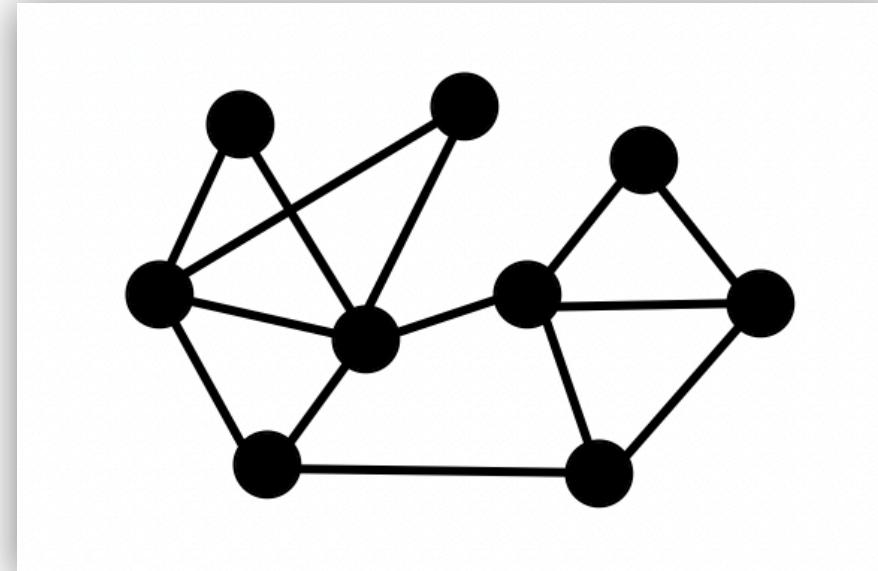
ML на графах

Графы в ML

ML на графах

Что такое граф

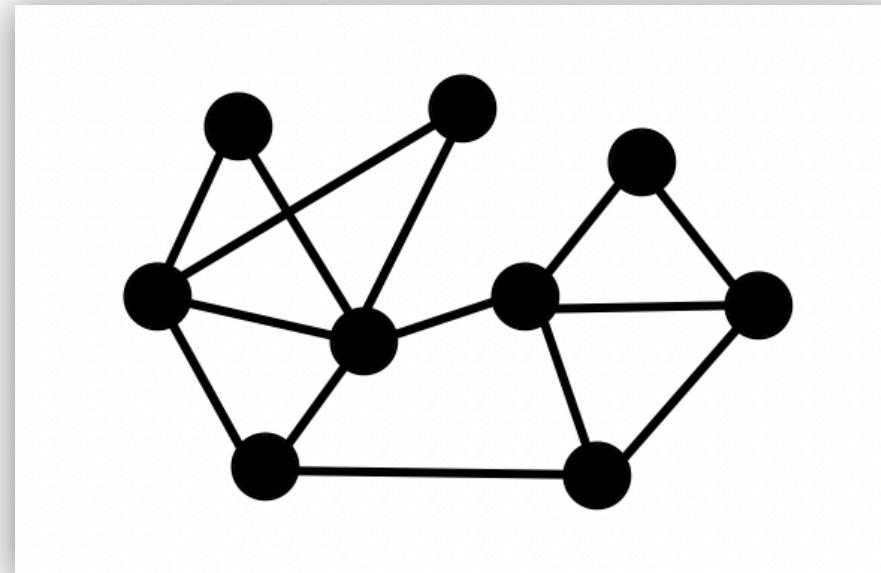
- Множество вершин V
- Множество ребер
объединяющих эти
вершины E



A BEAUTIFUL TITLE HERE

Зачем нам граф

Мы используем графы,
чтобы описать и
анализировать элементы с
отношениями/
взаимодействиями



ML на графах

Данные, которые уже граф

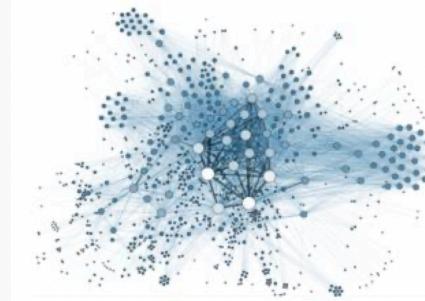
Естественные графы (Natural graphs) - данные которые уже представлены в форме графа



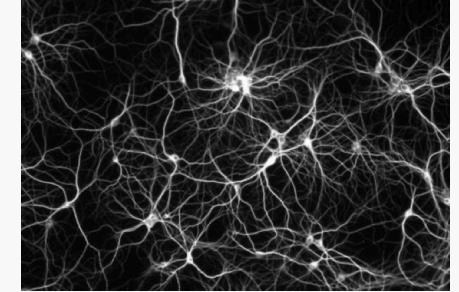
Социальные сети



Компьютерные сети



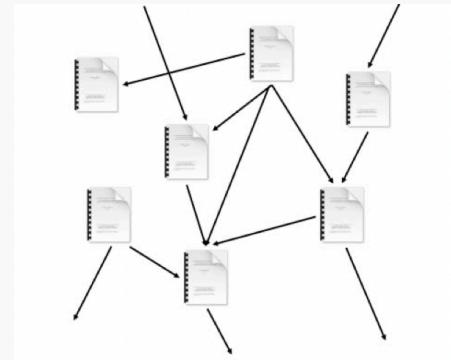
Сети коммуникаций



Граф нейронов



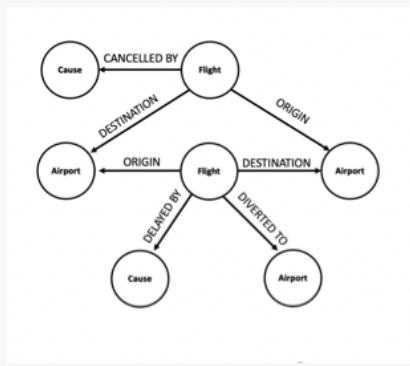
Интернет



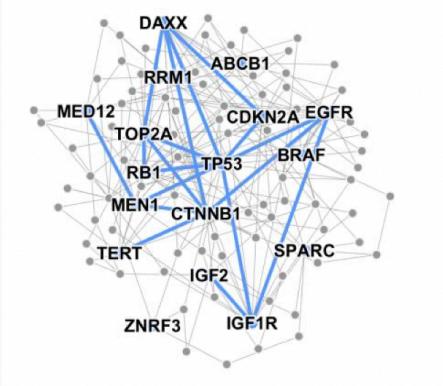
Цитирования

Данные, которые не очень граф

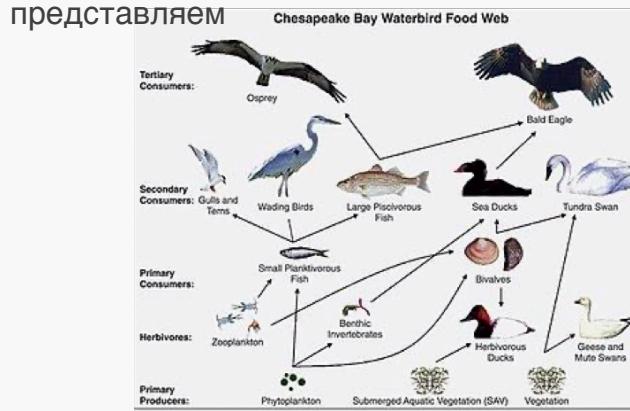
Репрезентация в виде графа - данные которые еще не представлены в форме графа, но мы их так



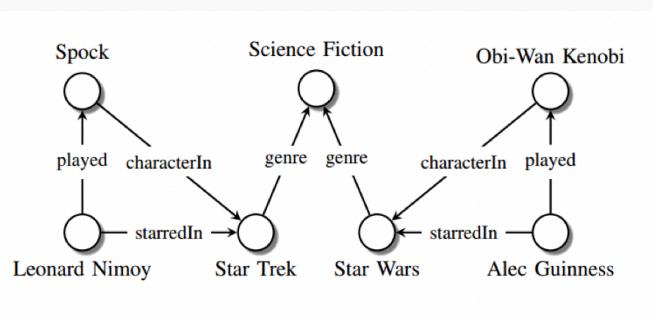
Граф событий



Распространение болезни



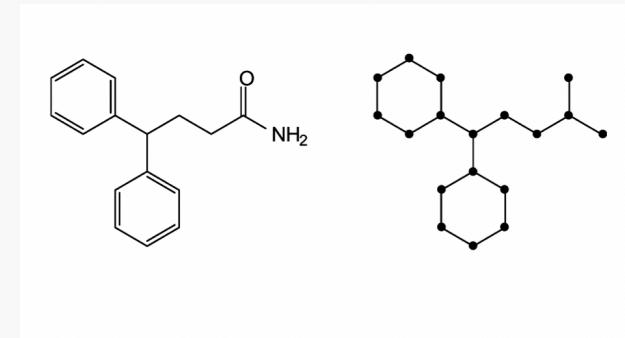
Particle networks(?)



Граф знаний



Карта метро

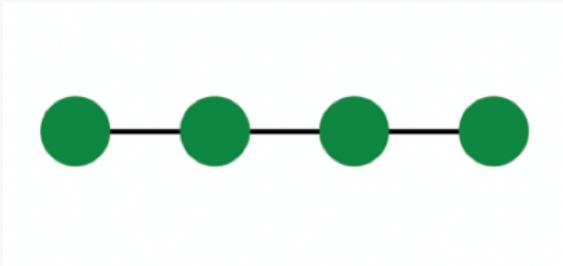


Молекулы

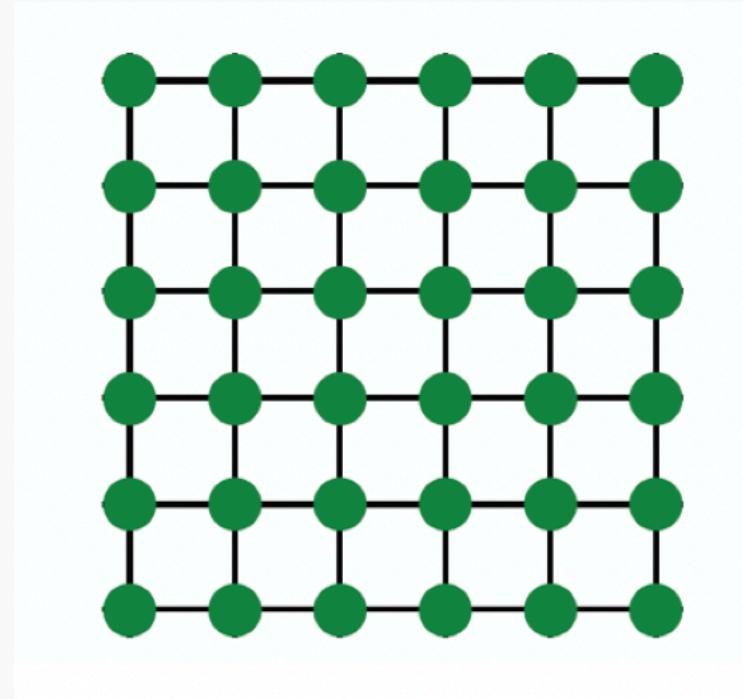
Данные, которые не очень граф

Репрезентация в виде графа - данные которые еще не представлены в форме графа, но мы их так представляем

Граф текста

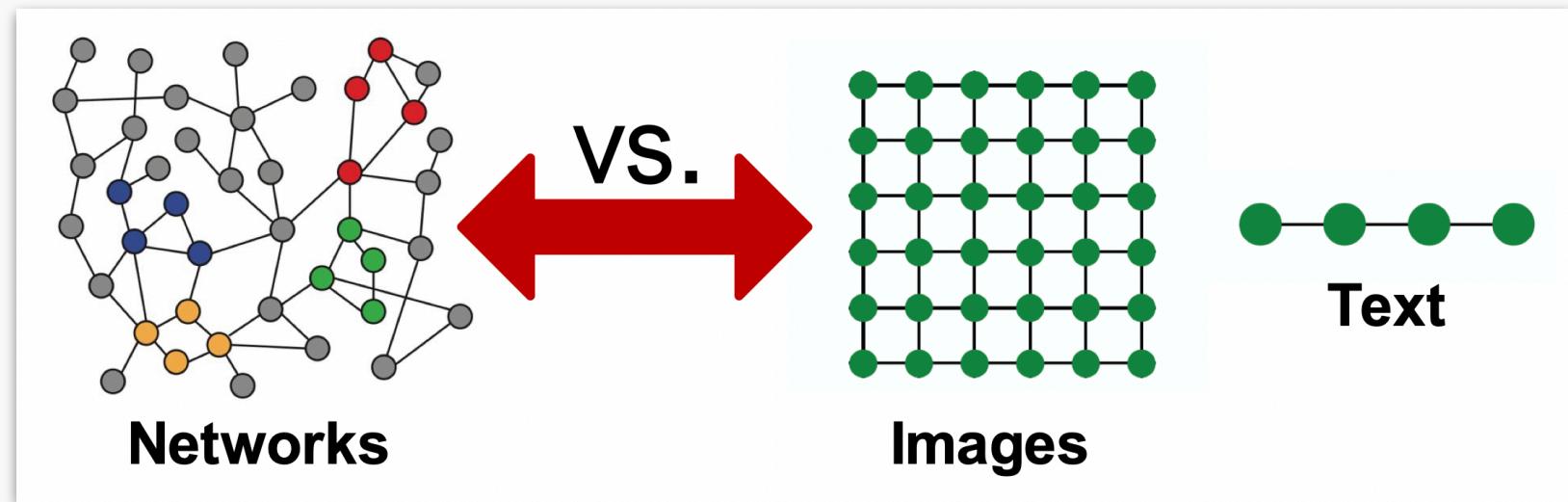


Граф картинки



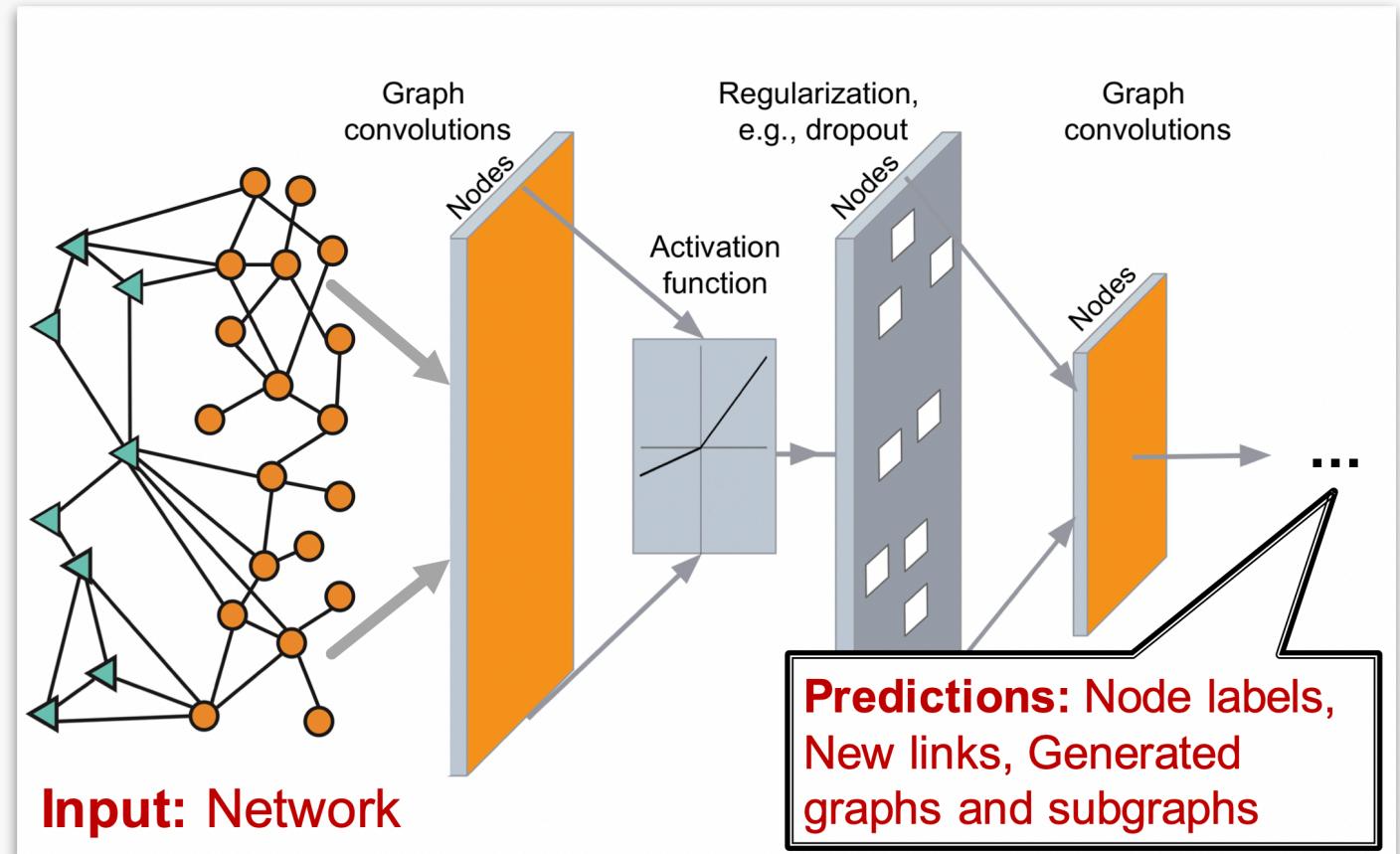
Почему это сложнее

- Произвольный размер и сложная топологическая структура
- Нет фиксированного размера или точки отсчета
- Часто они изменяющиеся в процессе



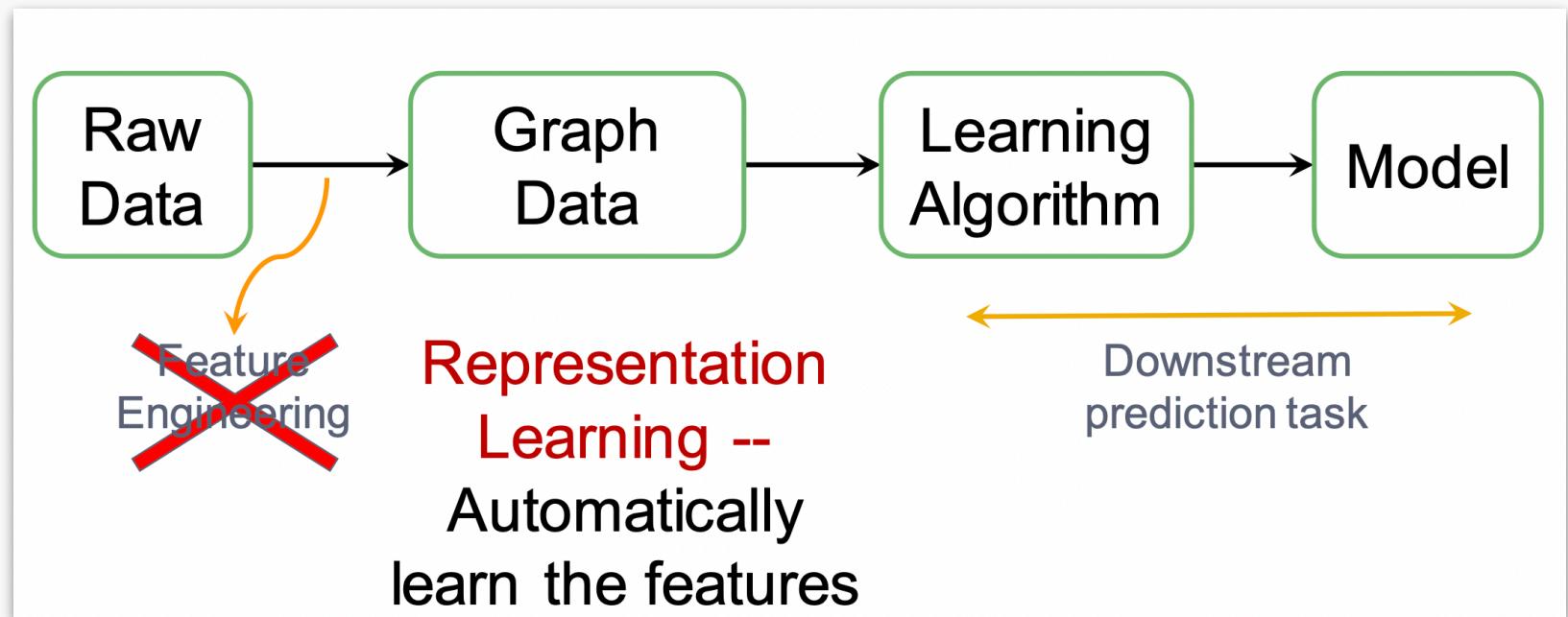
Что хотим получить?

- Получать на вход график
- Что-то делать внутри
- Предсказывать что-то по графу (Вершины / ребра / графы)



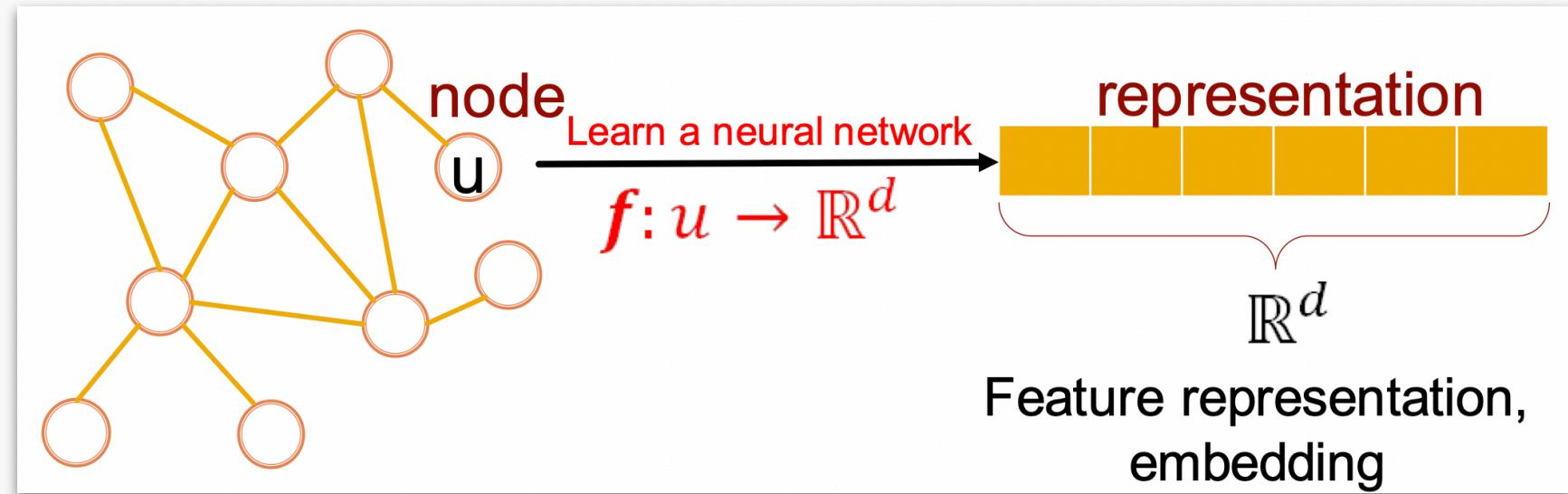
Но как это сделать?

- Мы не хотим смотреть на граф и придумывать его свойства
- Мы хотим как-то его преобразовать чтобы наша модель сама смогла выявить признаки



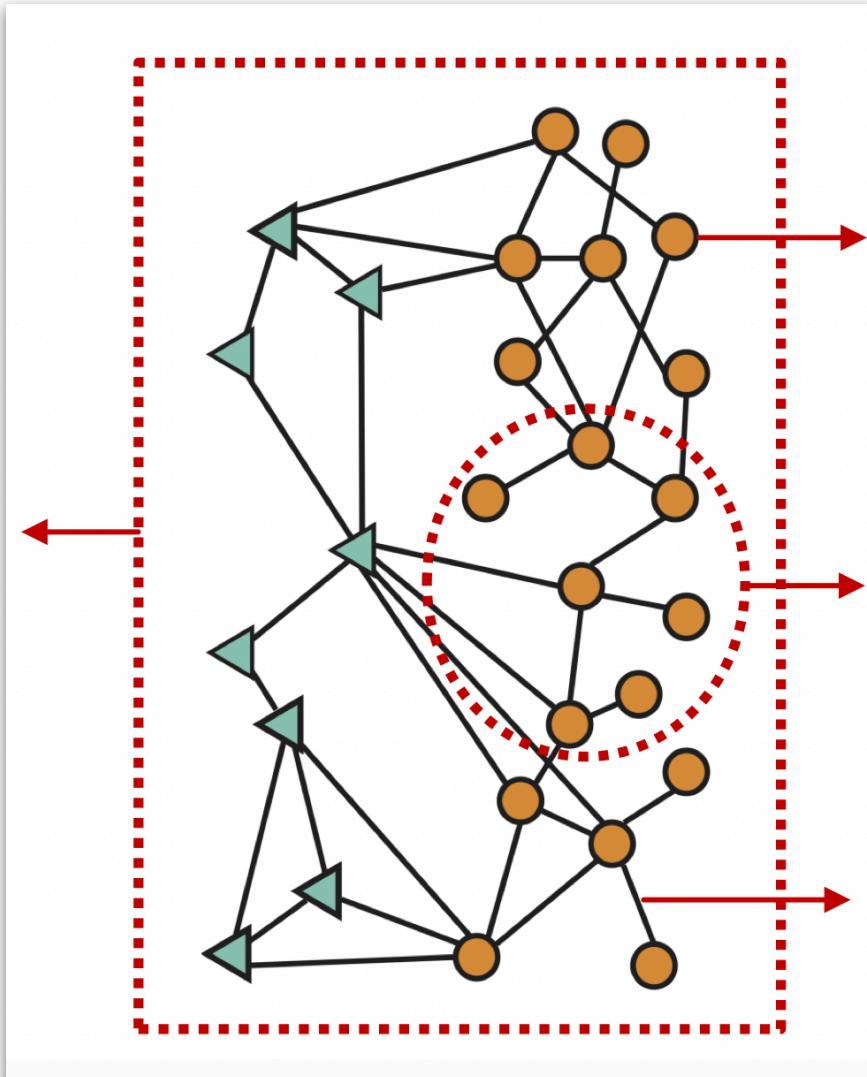
Но как это сделать?

Давайте каждую вершину представлять в виде вектора, но так, чтобы близкие вершины находились близко к друг другу



Какие бывают задачи?

Предсказание уровня графа



Предсказание уровня вершин

Предсказание уровня подграфа

Предсказание уровня ребер

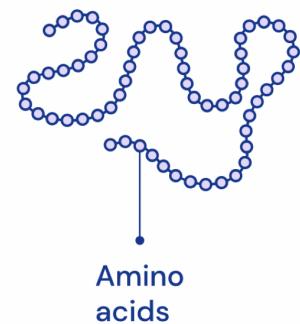
Какие бывают задачи?

- Классификация вершин
 - Категоризация пользователей
- Предсказание ребер
 - Полный ли граф знаний
- Классификация графа
 - Классификация молекул
- Кластеры (подграфы)
 - Принадлежность социальному кругу
- Другие задачи
 - Генерация графов
 - Изобретение лекарств
 - Эволюция графа
 - Физические симуляции

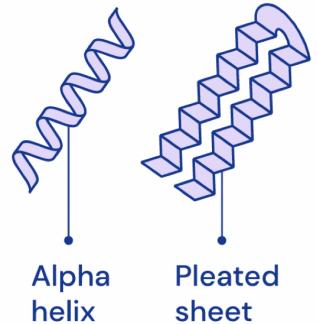
Предсказания уровня вершин

Можем ли мы получить 3d структуру белка, зная только последовательность его аминокислот

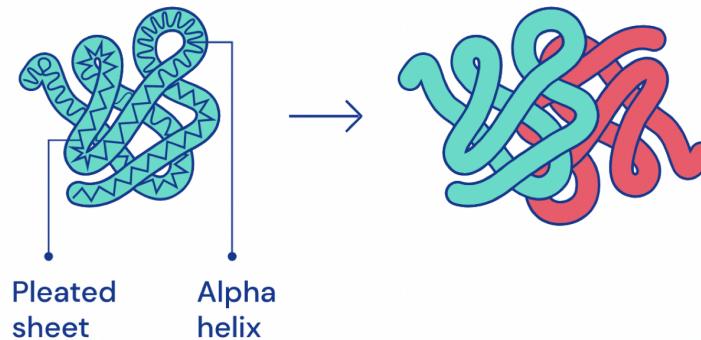
Every protein is made up of a sequence of amino acids bonded together



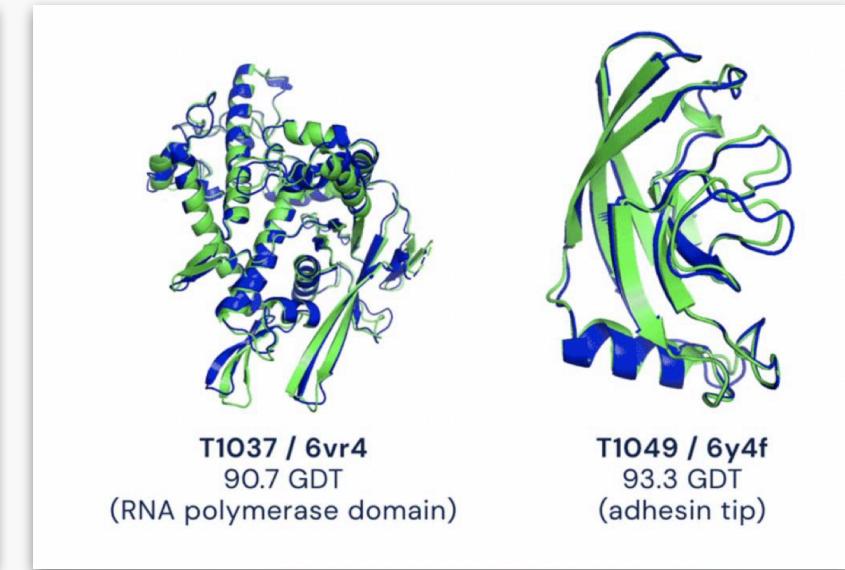
These amino acids interact locally to form shapes like helices and sheets



These shapes fold up on larger scales to form the full three-dimensional protein structure



Proteins can interact with other proteins, performing functions such as signalling and transcribing DNA

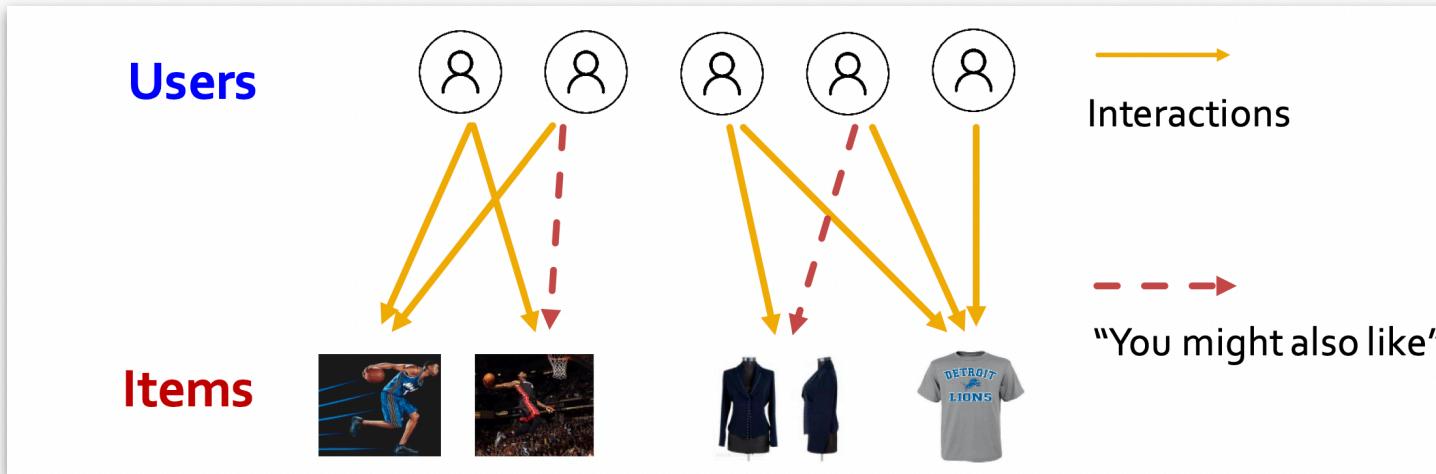


Идея:

Представить аминокислоты в виде вершины,
а в виде ребер представить их близость

Предсказания уровня ребер

Предсказание рекомендаций для пользователя

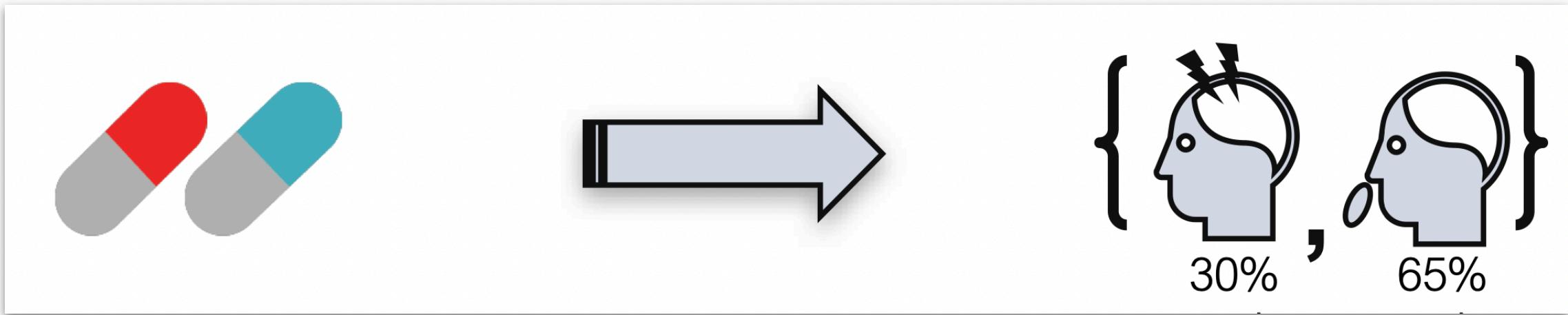


Идея:

Представить пользователей и предметы в виде вершины,
а в виде ребер представить то, что им нравится
(Возможные ребра между предметами должны быть короче, чем невозможные)

Предсказания уровня ребер

Предсказание побочных эффектов от лекарств

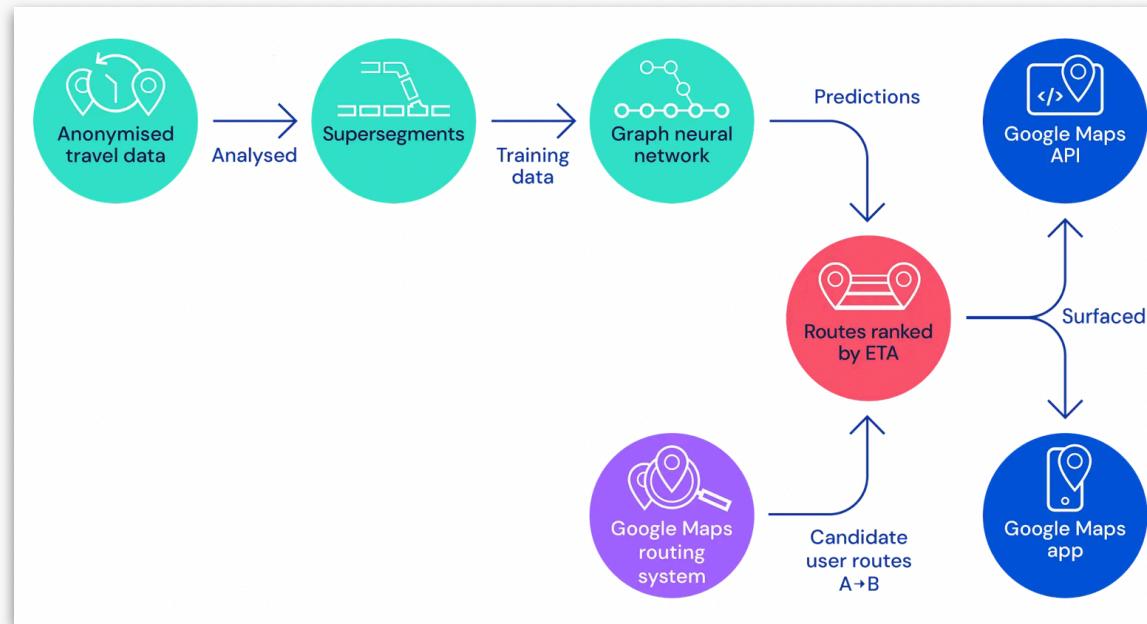


Идея:

Представить лекарства, как вершины,
а в виде ребер представить побочные эффекты

Предсказания уровня подграфов

Предсказание пробок

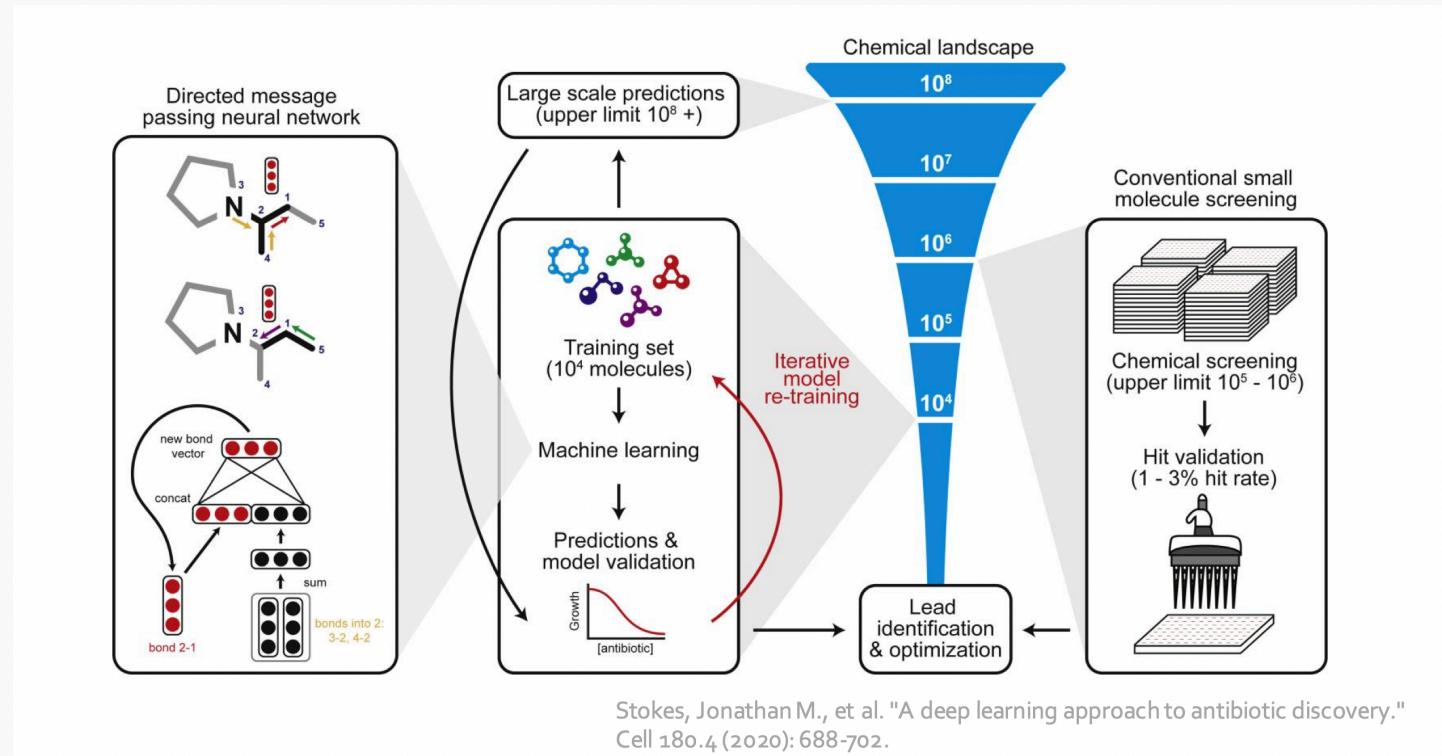


Идея:

Представить кусочки дорог, как вершины,
а в виде ребер представить соединения

Предсказания уровня графов

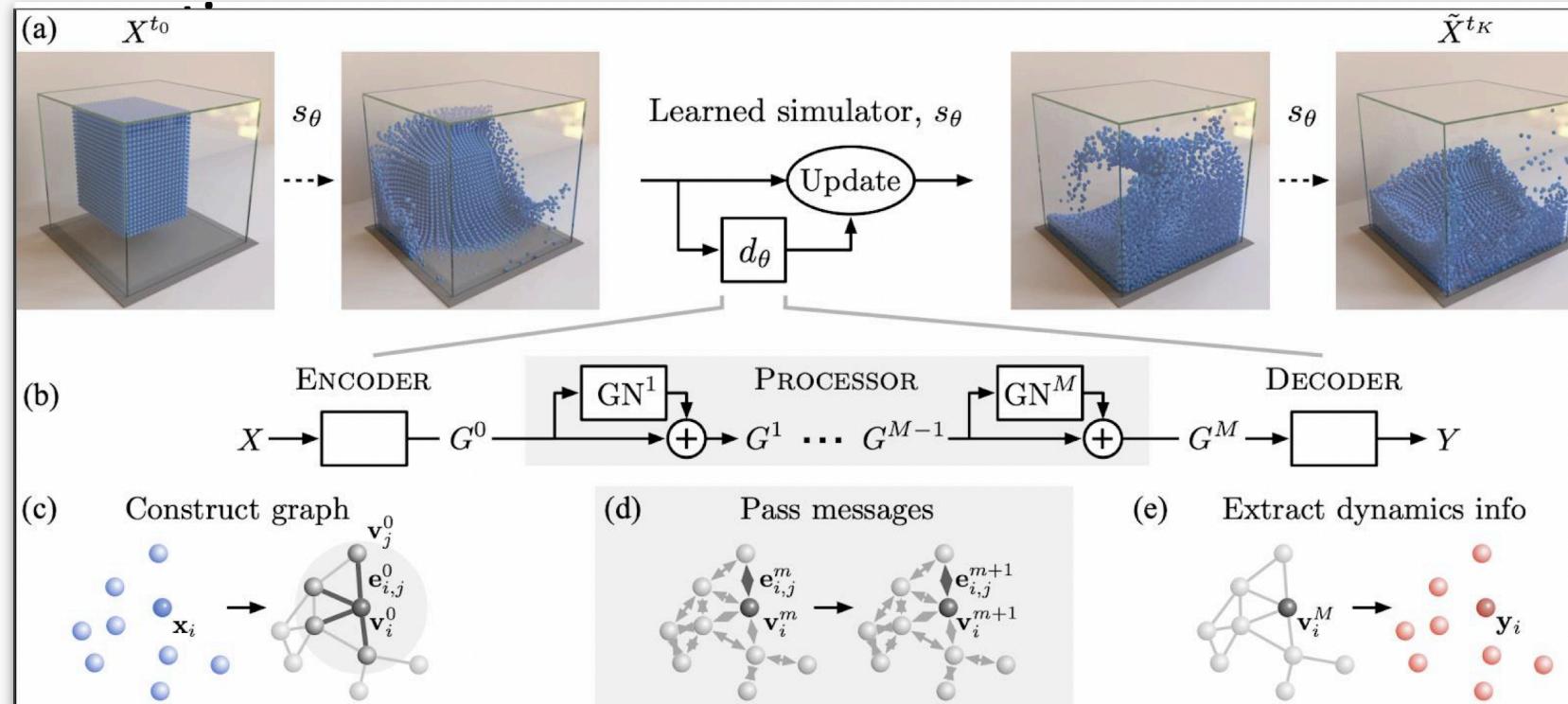
Создание лекарств



Идея:
Представить атомы, как вершины,
а в виде ребер представить соединения

Предсказания эволюции графов

Предсказание движения частиц



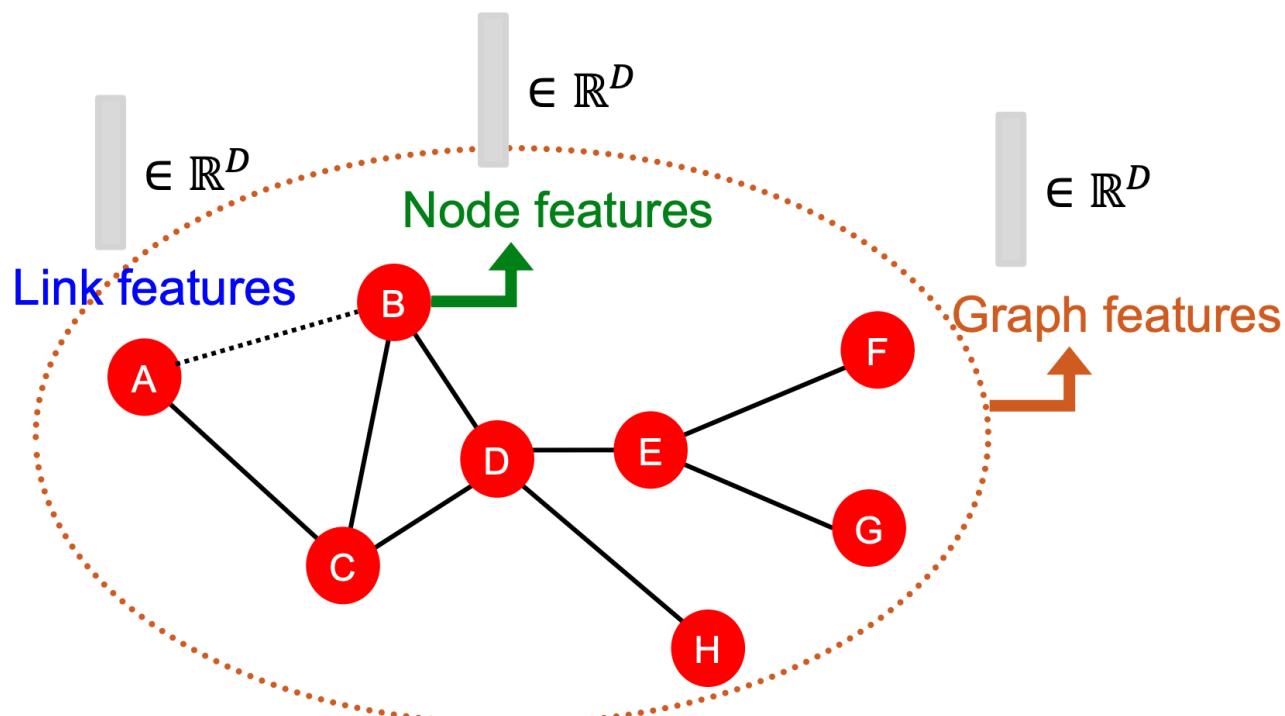
Идея:
Представить частицы, как вершины,
а в виде ребер представить их близость

ML на графах

Классический ML на графах

Конструирование признаков

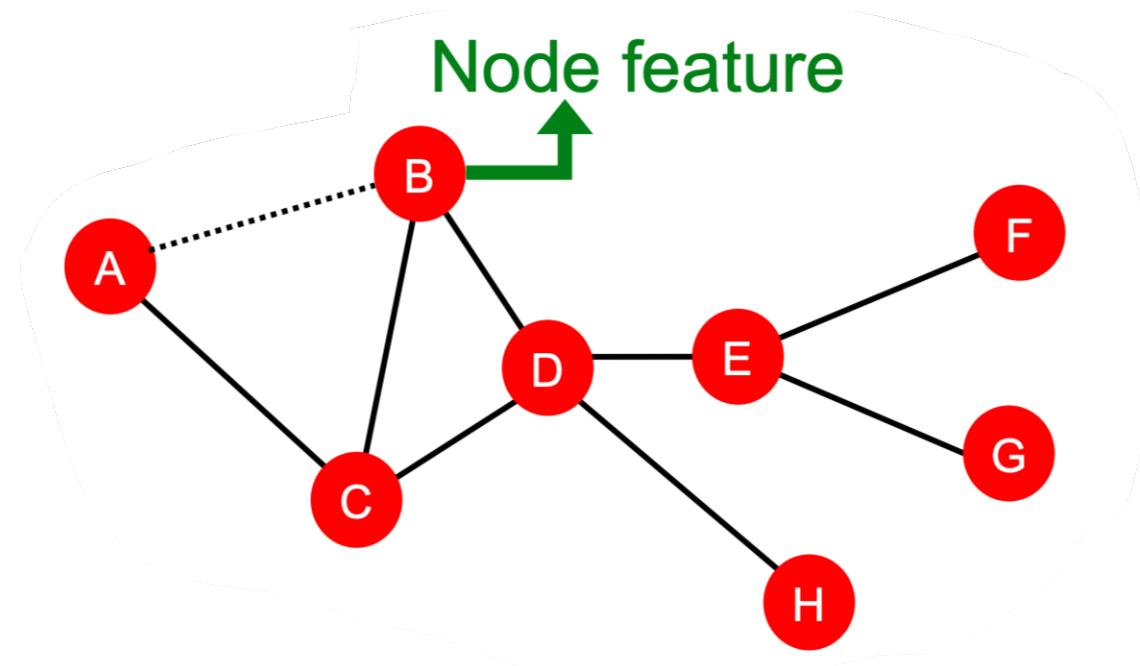
1. Признаки для вершин
2. Признаки для ребер
3. Признаки для целого графа



Признаки для вершин

Хотим охарактеризовать структуру и положение
вершины в графе:

- Степень вершины
- Центральность вершины
- Коэффициент кластеризации
- Подграфы



Центральность

Когда мы считаем степень вершины, мы не
учитываем важность ее соседей

Разные меры вершинной центральности решают эту
проблему:

- Центральность на основе собственных векторов
(Engienvector centrality)
- Степень посредничества (Betweenness centrality)
- Степень близости (Closeness centrality)
- Другие меры центральности

Engienvector centrality

Считаем, что вершина важная, если соединена с важными вершинами: будем моделировать центральность, как сумму центральностей соседних вершин

Центральность это собственный вектор! В качестве самой меры используют собственный вектор `c_max`, соответствующий максимальному собственному значению

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

λ is some positive constant

- \longleftrightarrow
- A : Adjacency matrix
 - $A_{uv} = 1$ if $u \in N(v)$
 - c : Centrality vector

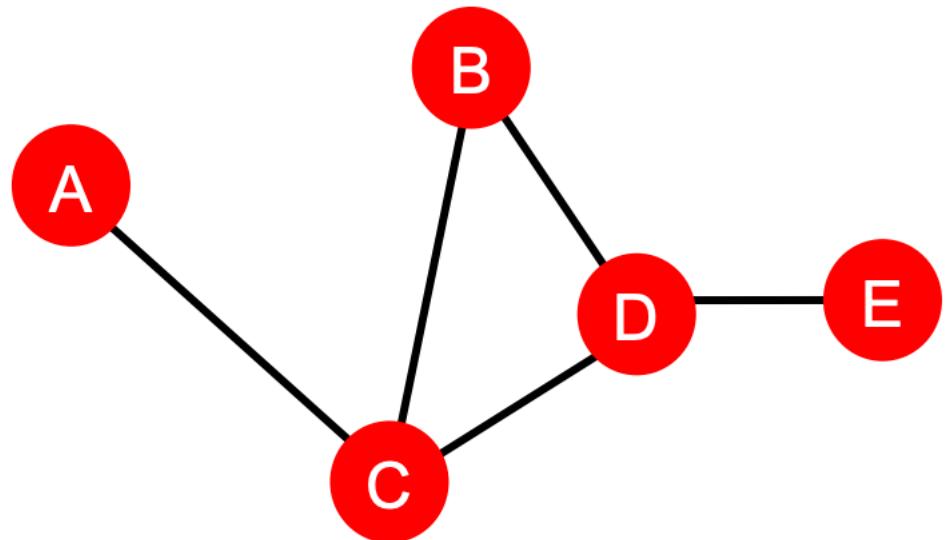
Betweenness centrality

Считаем, что вершина важная, если она находится на большом количестве кратчайших путей между другими вершинами

$$c_v = \sum_{s \neq v \neq t} \frac{\text{#(shortest paths between } s \text{ and } t \text{ that contain } v)}{\text{#(shortest paths between } s \text{ and } t)}$$

Betweenness centrality

Пример:



$$c_A = c_B = c_E = 0$$

$$c_C = 3$$

(A-C-B, A-C-D, A-C-D-E)

$$c_D = 3$$

(A-C-D-E, B-D-E, C-D-E)

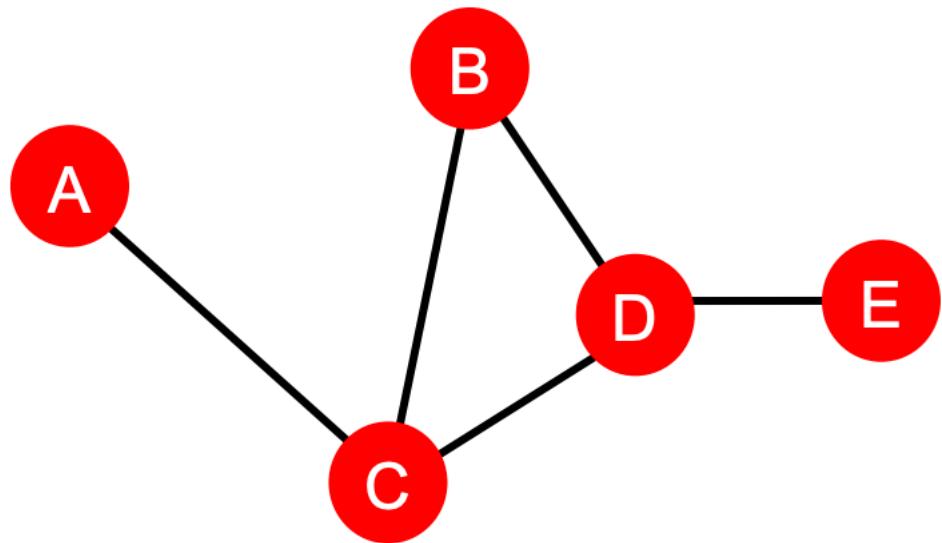
Closeness centrality

Считаем, что вершина важная, если в сумме она имеет маленькую длину кратчайшего пути ко всем остальным узлам.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

Closeness centrality

Пример:



$$c_A = 1/(2 + 1 + 2 + 3) = 1/8$$

(A-C-B, A-C, A-C-D, A-C-D-E)

$$c_D = 1/(2 + 1 + 1 + 1) = 1/5$$

(D-C-A, D-B, D-C, D-E)

Коэффициент кластеризации

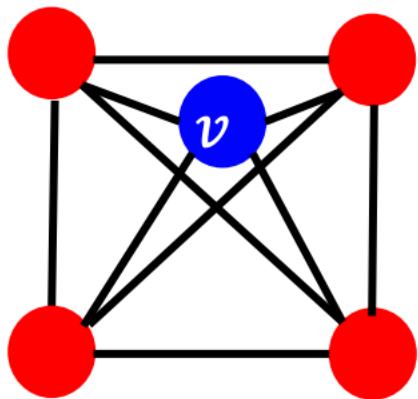
Коэффициент кластеризации измеряет долю связей
между соседними вершинами:

$$e_v = \frac{\text{#(edges among neighboring nodes)}}{\binom{k_v}{2}} \in [0,1]$$

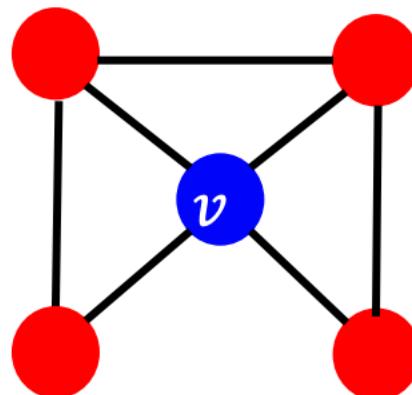
#(node pairs among k_v neighboring nodes)

Коэффициент кластеризации

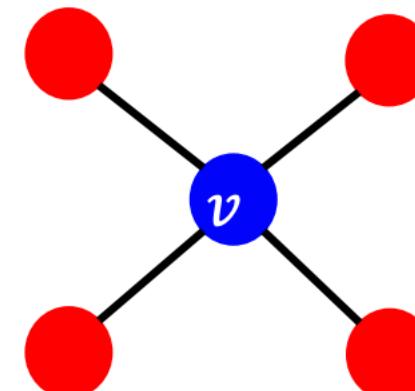
Пример:



$$e_v = 1$$



$$e_v = 0.5$$

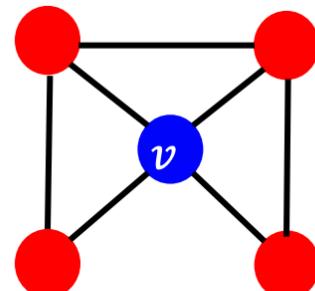


$$e_v = 0$$

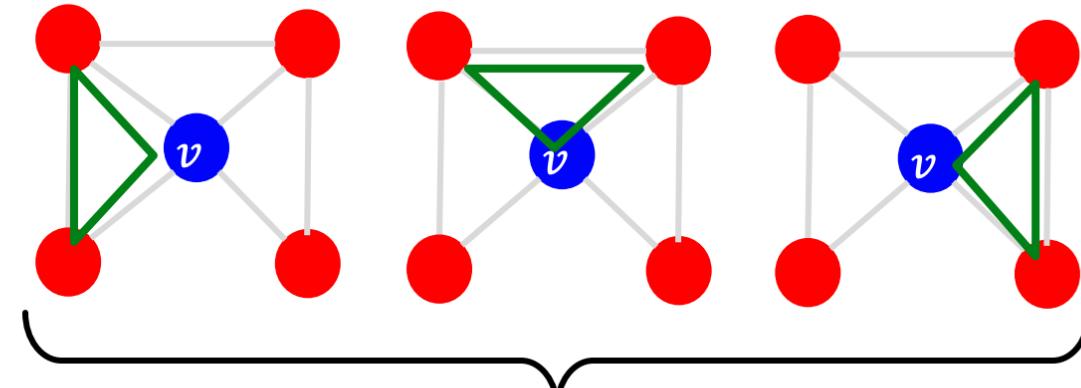
Коэффициент кластеризации

Можно заметить, что коэффициент кластеризации на самом деле считает количество треугольников в подграфе индуцированном вершиной и её соседями

Мы можем обобщить коэффициент кластеризации, подсчитывая не треугольники, а произвольные индуцированные подграфы, которые тут называют графлетами

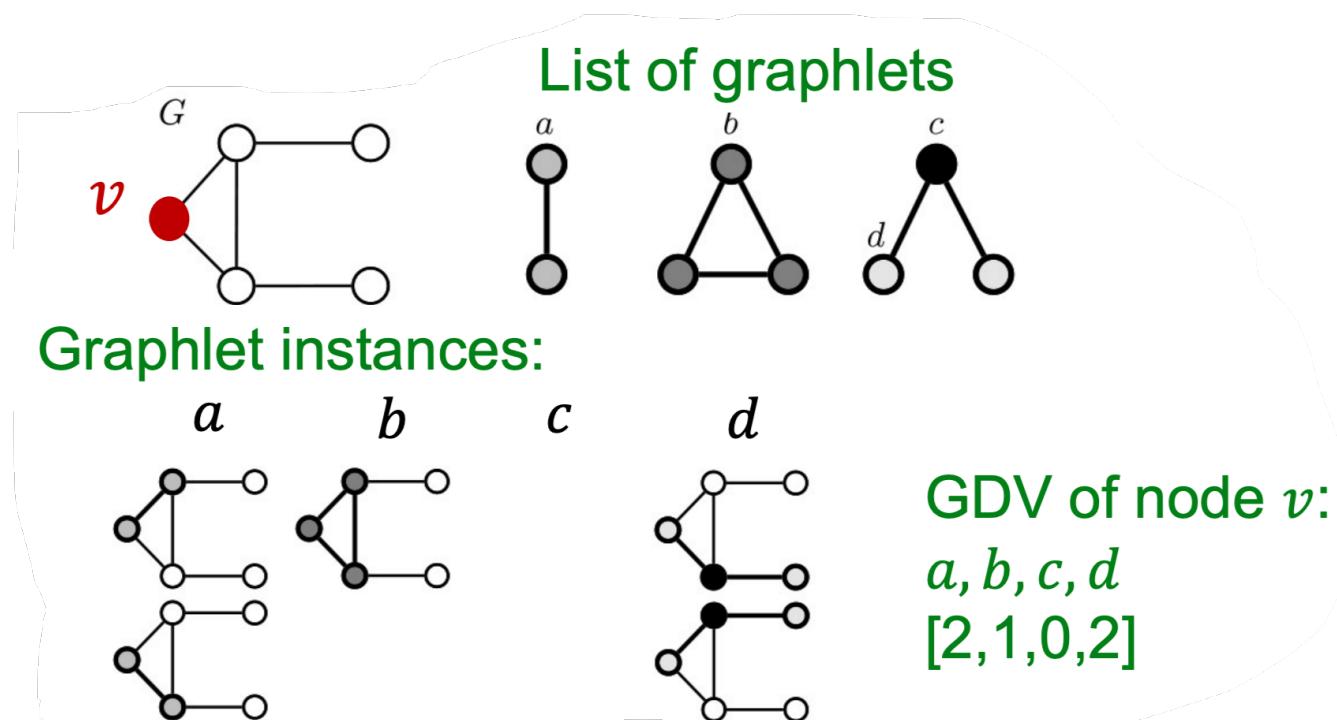


$$e_v = 0.5$$



Коэффициент кластеризации

Graphlet Degree Vector (GDV): считает количество смежных к вершине графлетов



Признаки для вершин

Итоги

Признаки, измеряющие важность вершин:

- Степень вершины: просто считает количество смежных вершин
- Меры центральности: моделирует важность соседних вершин

Пример: С помощью таких признаков можно прогнозировать влиятельные вершины в графе (селеб в социальной сети)

Признаки, измеряющие структуру вершины внутри графа:

- Степень вершины
- Коэффициент кластеризации: измеряет, как сильно связаны соседние вершины
- Подсчет графлетов

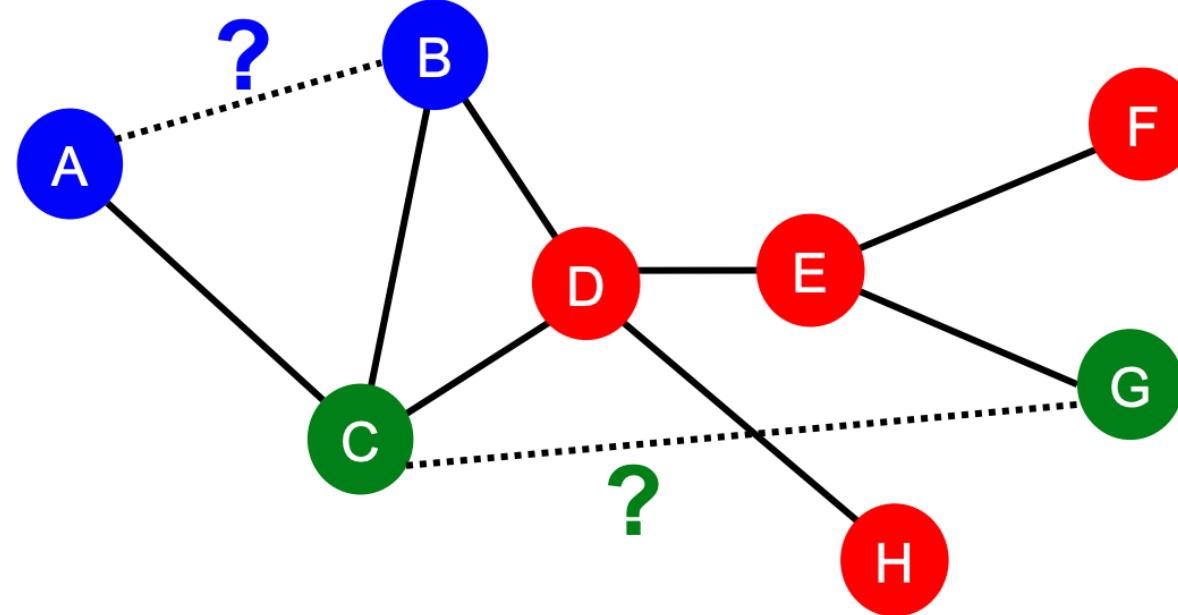
Пример: С помощью таких признаков можно прогнозировать определенную роль, которую вершина играет в графе (функции белка, в сети взаимодействий белок-белок)

Признаки для ребер

Будем решать задачу предсказания ребер

Хотим составить признаки для пары вершин:

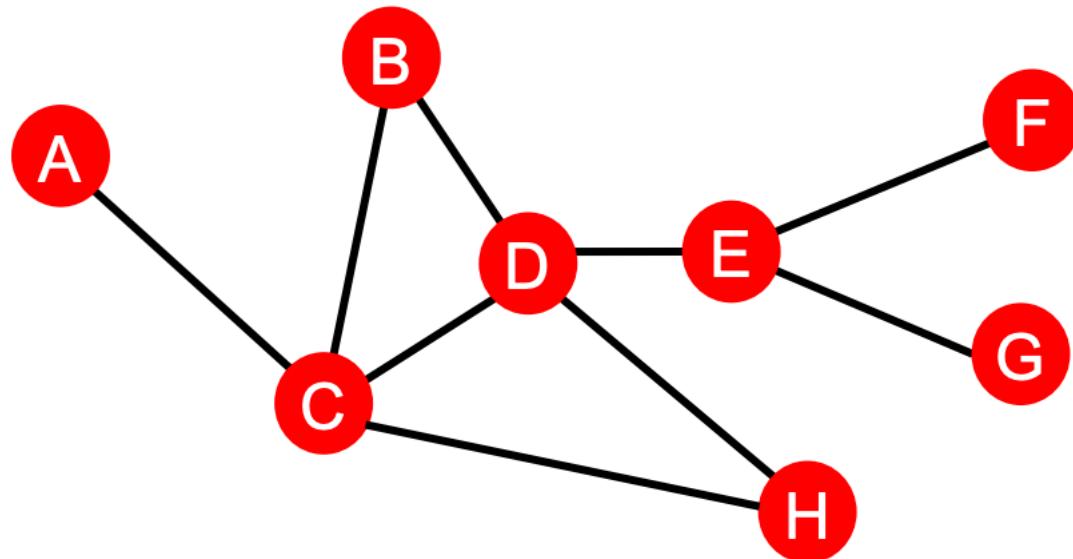
- Признаки основанные на расстояниях
- Local neighborhood overlap
- Global neighborhood overlap



Кратчайшее расстояние между двумя вершинами

Считаем кратчайшие пути в графе между двумя вершинами

Такой признак не учитывает связи по соседним вершинам



$$S_{BH} = S_{BE} = S_{AB} = 2$$

$$S_{BG} = S_{BF} = 3$$

Local neighborhood overlap

Теперь считаем количество пересекающихся соседних вершин

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$

- Example: $|N(A) \cap N(B)| = |\{C\}| = 1$

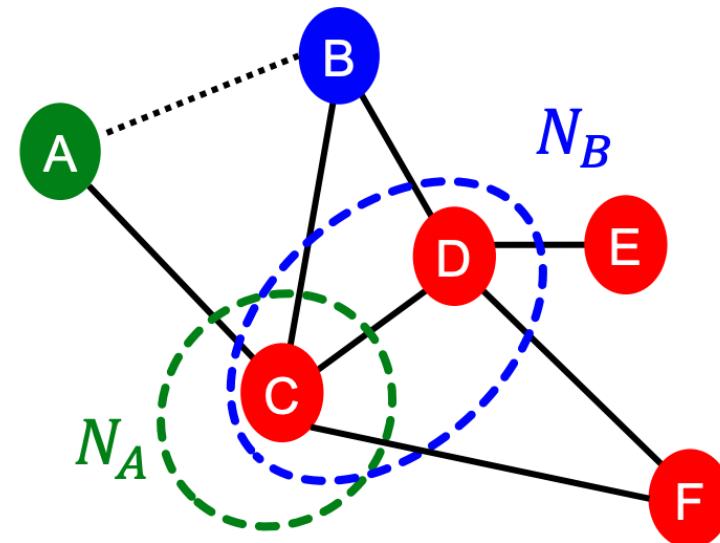
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$

- Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{A, B, C, D\}|} = \frac{1}{2}$

- **Adamic-Adar index:**

$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$

- Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



Global neighborhood overlap

Минус Local neighborhood overlap:

- Метрика всегда нулевая, если у вершин нет общих соседей
- При этом они все еще могут быть потенциально связаны

Метрики Global neighborhood overlap решают эту проблему:

- Katz index: считает количество путей всех длин между парой вершин

Katz index

Как посчитать количество путей между вершинами?

Используем матрицу смежности!

A_{uv}^l specifies #paths of length l .

Тогда Katz index можно посчитать так:

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{A_{v_1 v_2}^l}$$

#paths of length l
between v_1 and v_2

$0 < \beta < 1$: discount factor

Признаки для графа

Мы хотим построить наши признаки так, чтобы они описывали структуру всего графа

Будем конструировать **ядра** вместо векторов признаков

$$K(G, G') = \phi(G)^T \phi(G')$$

Будем конструировать ядра, через признаки в таком виде

Graphlet Kernel

Используем мешок слов для графа, только тут мы будем считать вхождения различных графлетов

Для каждого заданного списка графлетов можем посчитать вектор количества вхождений в граф

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1, 2, \dots, n_k$$

$$\mathbf{h}_G = \frac{\mathbf{f}_G}{\text{Sum}(\mathbf{f}_G)} \quad K(G, G') = \mathbf{h}_G^T \mathbf{h}_{G'}$$

Weisfeiler-Lehman Kernel

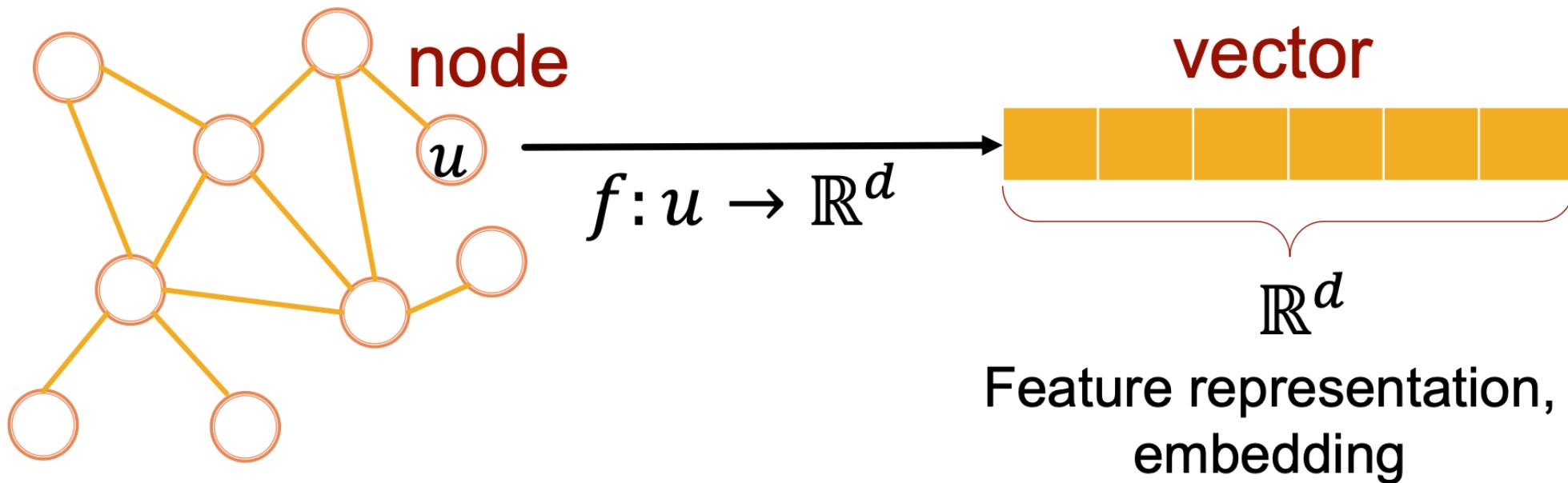
Так как считать подграфы в графе очень сложно, люди решили придумать что-то еще
Будем считать для вершины так называемый цвет

$$c^{(k+1)}(v) = \text{HASH} \left(\left\{ c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)} \right\} \right)$$

После k шагов цвет будет содержать в себе информацию о k -окрестности вершины
После процедуры соберем вектор количества вершин определенного цвета, его и будем использовать в ядре

Node Embeddings

Хотим заняться Representation learning и автоматически создавать признаки для вершин



Node Embeddings

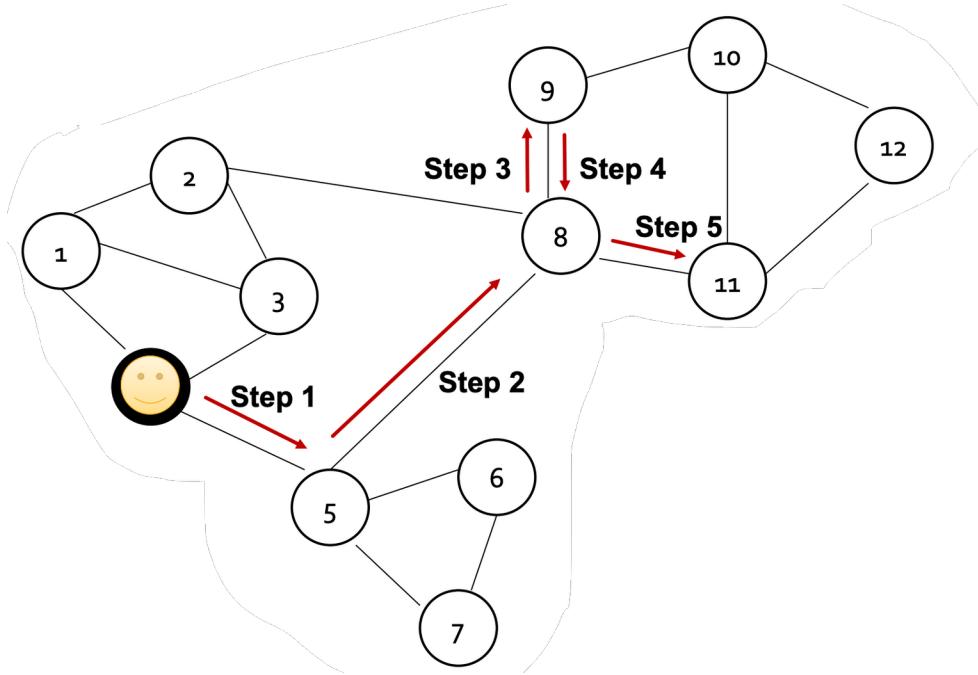
1. Придумываем, как определять схожесть вершин в исходном графе
2. Придумываем Encoder, которые переводят вершину в вектор
3. Делаем Decoder, который переводит векторные представления в меру схожести
4. Настраиваем параметры энкодера так, чтобы выполнялось:

$$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

in the original network

Similarity of the embedding

Node Embeddings



Будем считать, что вершины близки, если с большой вероятностью они возникнут во время **случайного блуждания** по графу

DeepWalk

1. Запускаем случайное блуждание короткой фиксированной длины из каждой вершины
2. Для каждой вершины собираем список вершин, которые мы успели посетить во время случайного блуждания из нее
3. Оптимизируем представления с помощью SGD

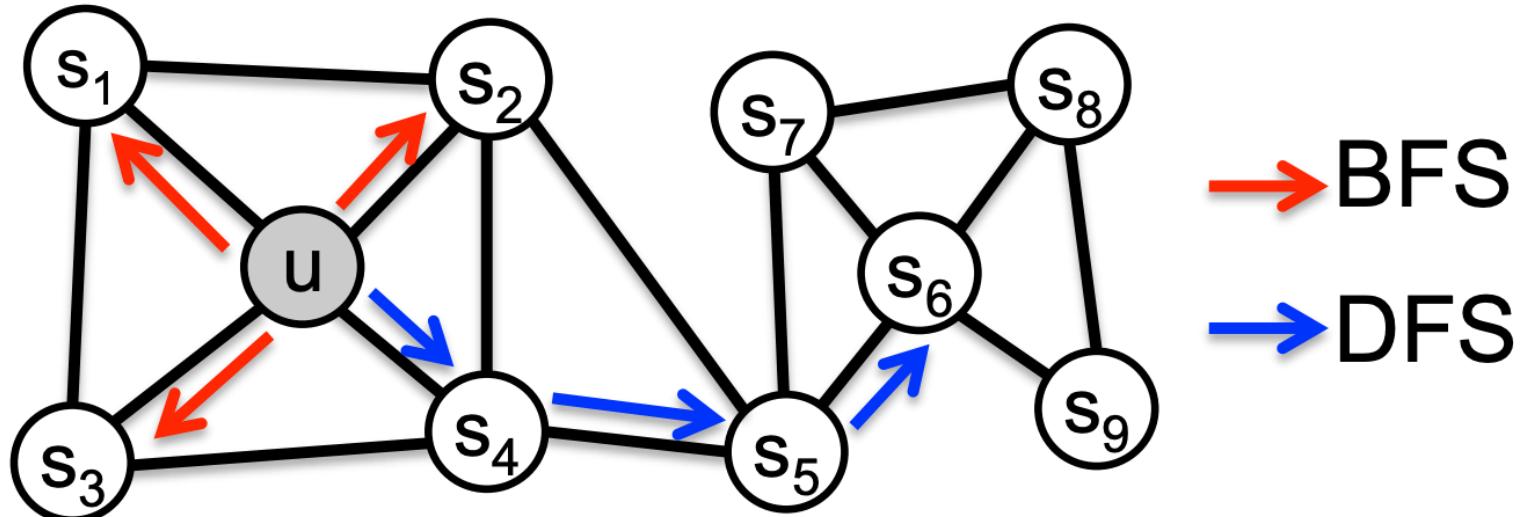
$$\max_f \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u) \rightarrow \text{Maximum likelihood objective}$$

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}$$

node2vec

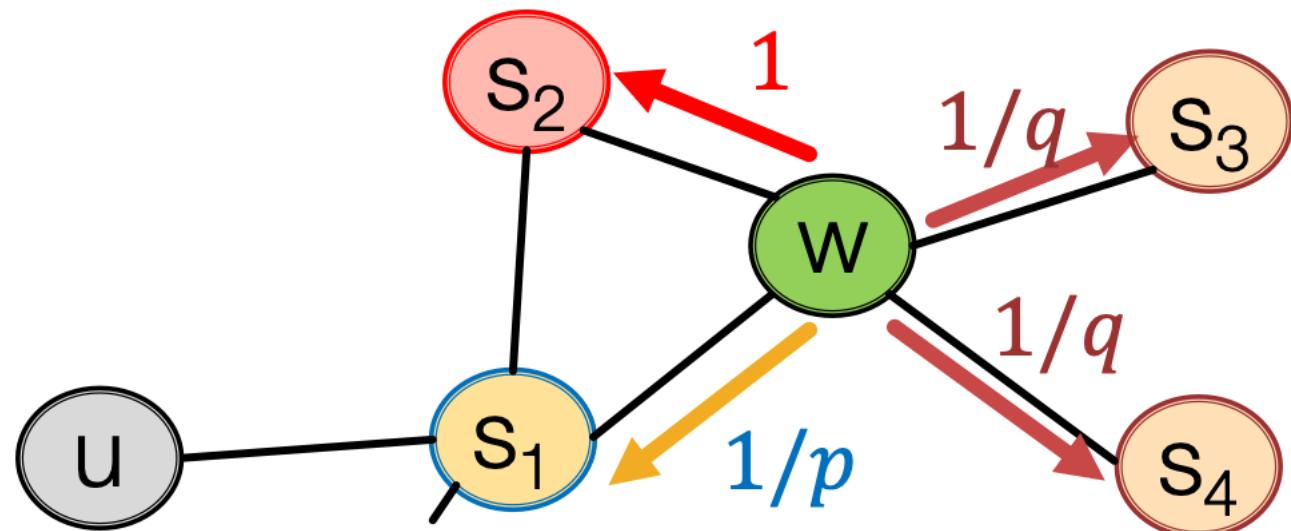
В отличие от DeepWalk тут мы используем несимметричное случайное блуждание с двумя параметрами:

1. Вероятность шагнуть в предыдущую вершину
2. Соотношение того, как часто мы шагаем в глубину или в ширину



node2vec

Пример:



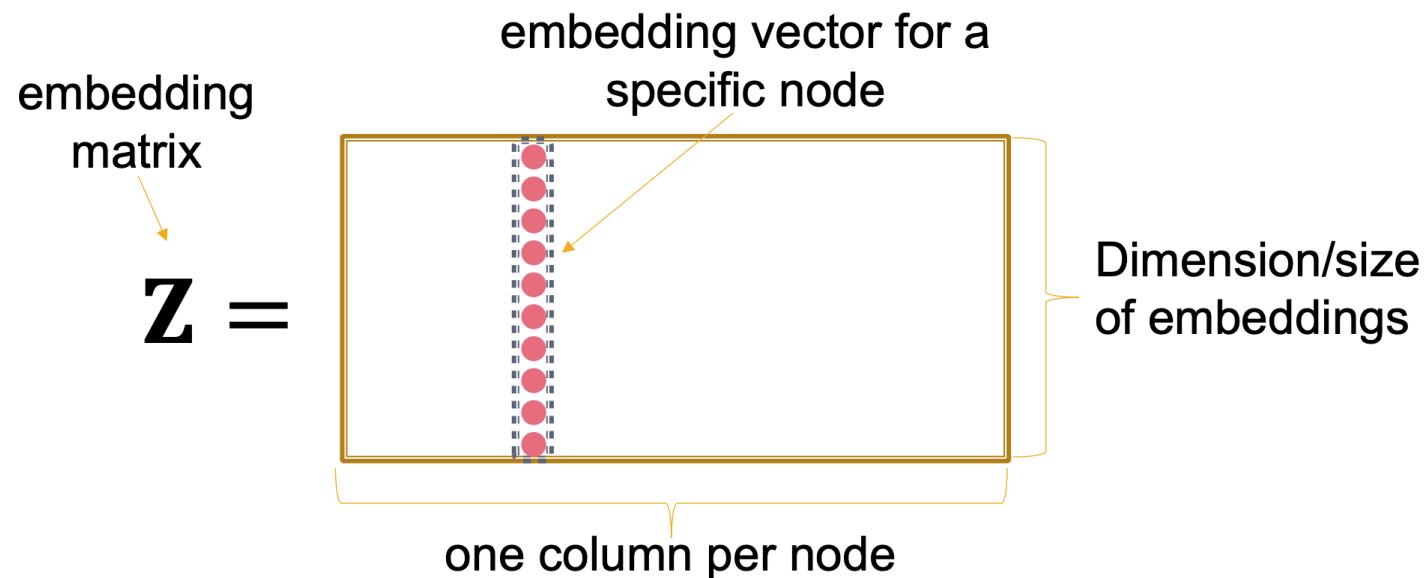
node2vec

Всегда ли эмбеддинги лучше конструирования других признаков ручками?

Нет, как показывает практика node2vec лучше подходит для задач классификации вершин,
а в задачах предсказания лучше использовать другие меры схожести вершин

Связь с матрицами

Самый простой способ определить похожесть вершин это сказать, что они похожи, если соединены ребром



$$\mathbf{z}_v^T \mathbf{z}_u = A_{u,v}$$

$$\mathbf{Z}^T \mathbf{Z} = A$$

Связь с матрицами

Такая задача сводится к матричному разложению матрицы смежности!

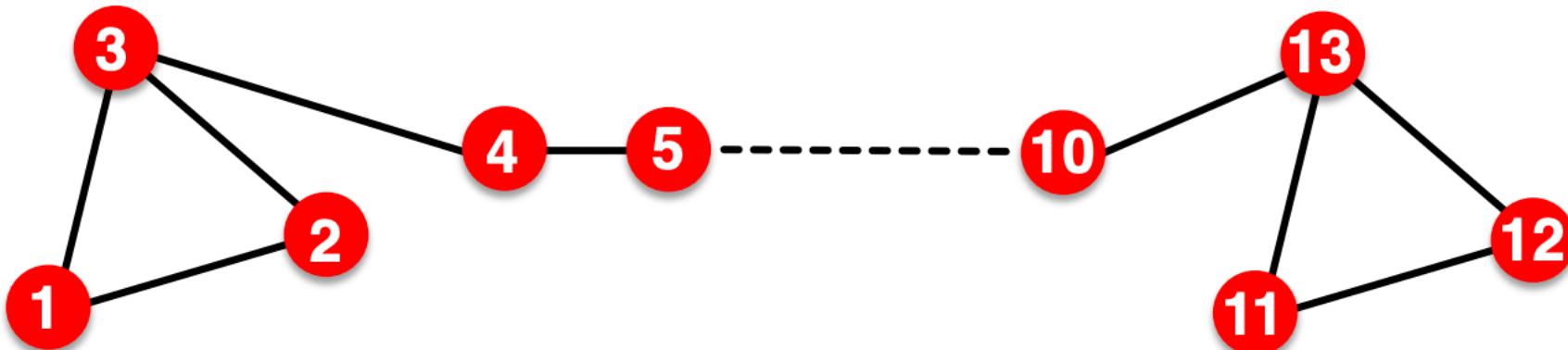
Тут можно использовать различные приближенные разложения из линейной алгебры, например искать что-то такое (приближение по фробениусовой норме):

$$\min_{\mathbf{Z}} \|\mathbf{A} - \mathbf{Z}^T \mathbf{Z}\|_2$$

Также интересно, что DeepWalk и node2vec тоже можно свести к матричным разложениям определенных матриц

Ограничения Node Embeddings

- Не получится найти представление для вершины не из обучающей выборки
- Рассмотренные методы не улавливают структурную схожесть вершин:
Вероятность попасть из 1 вершины в 11 мала
- Нет возможности использовать дополнительные признаки



Все эти проблемы решают Deep Representation Learning и Графовые нейронные сети!

Спасибо за внимание!

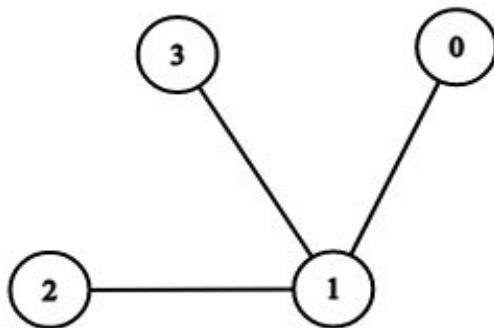
ML на графах

Графовые нейронные сети

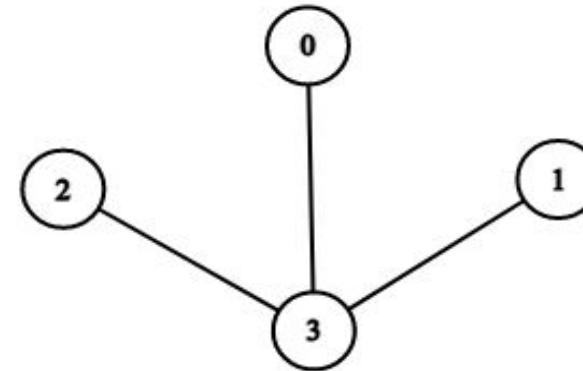
Марк Рофин, БПМИ-191

Необходимые свойства

1. Устойчивость к перестановке вершин
2. Обработка данных с нерегулярной структурой
3. Масштабирование на гигантские графы



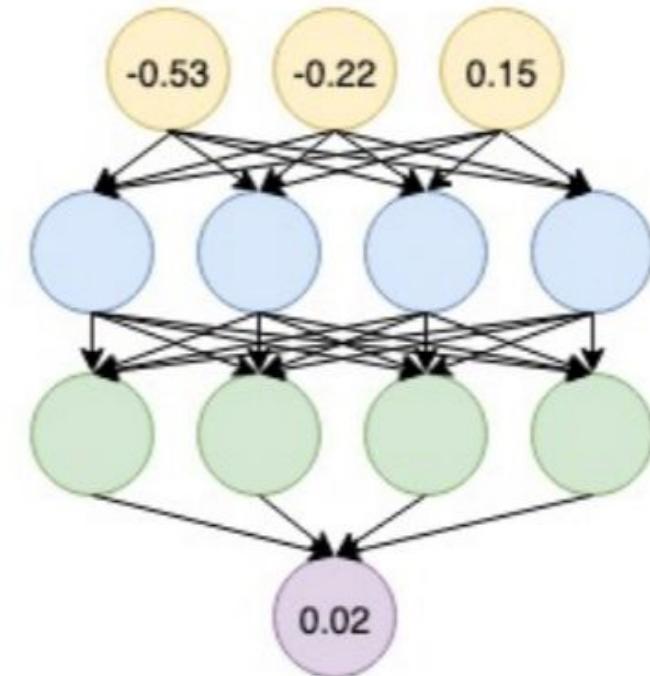
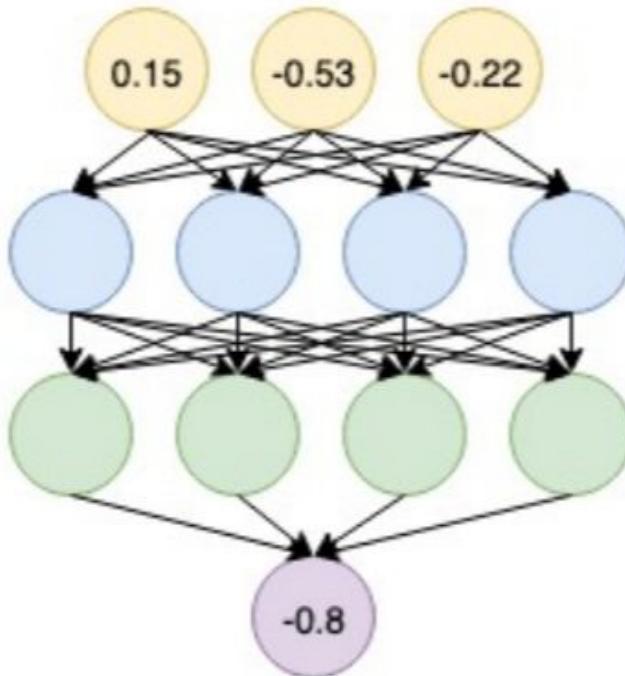
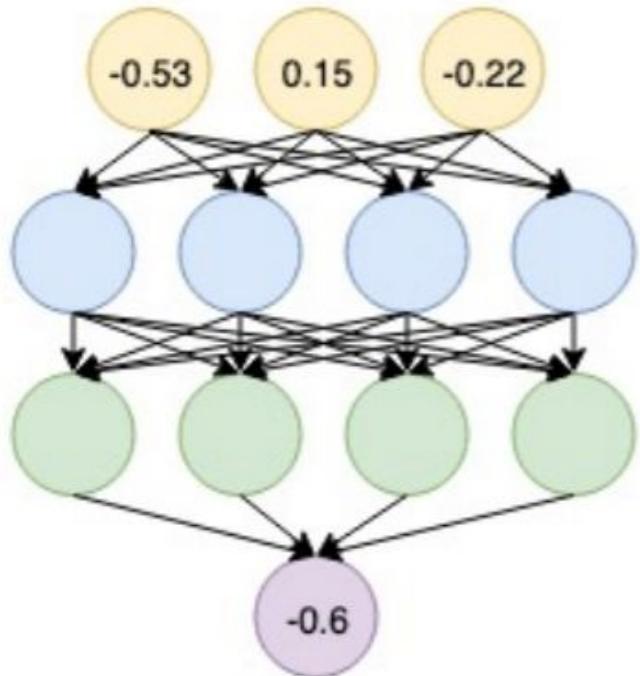
$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$



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

Необходимые свойства

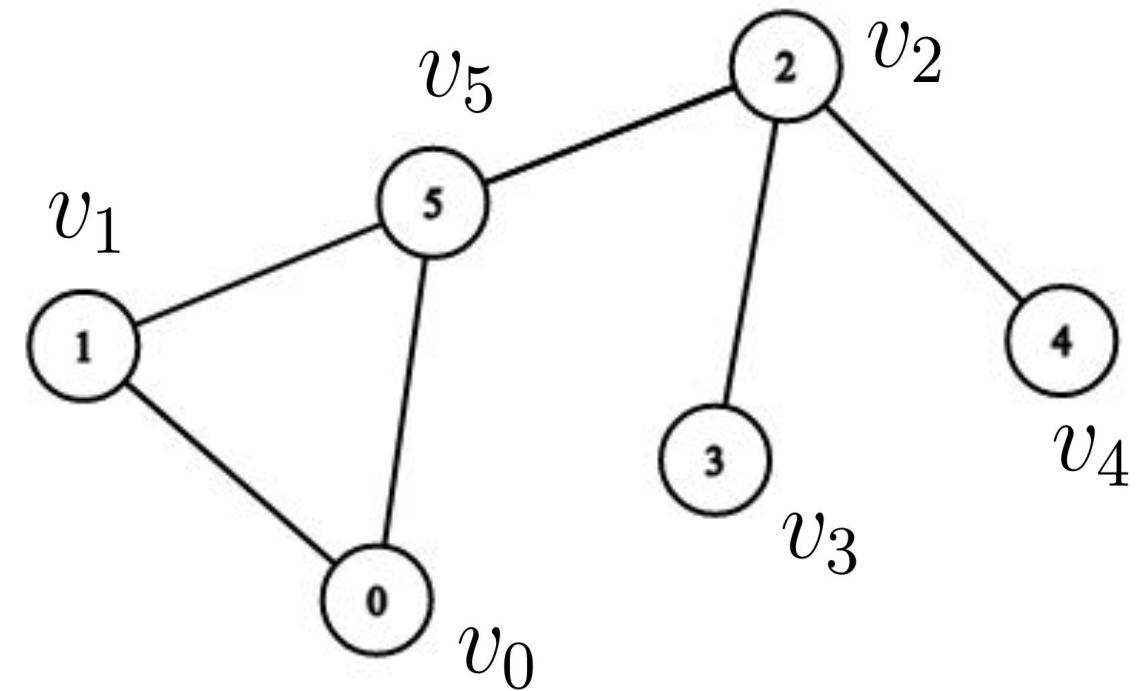
Полносвязные (и все прочие известные нам) сети
этим свойствам не удовлетворяют!



Graph Convolutional Networks: Идея

Идея: эмбеддинг вершины — это функция от признаков этой вершины и признаков всех её соседей.

$$h_5 = f(v_0, v_1, v_2, v_5)$$



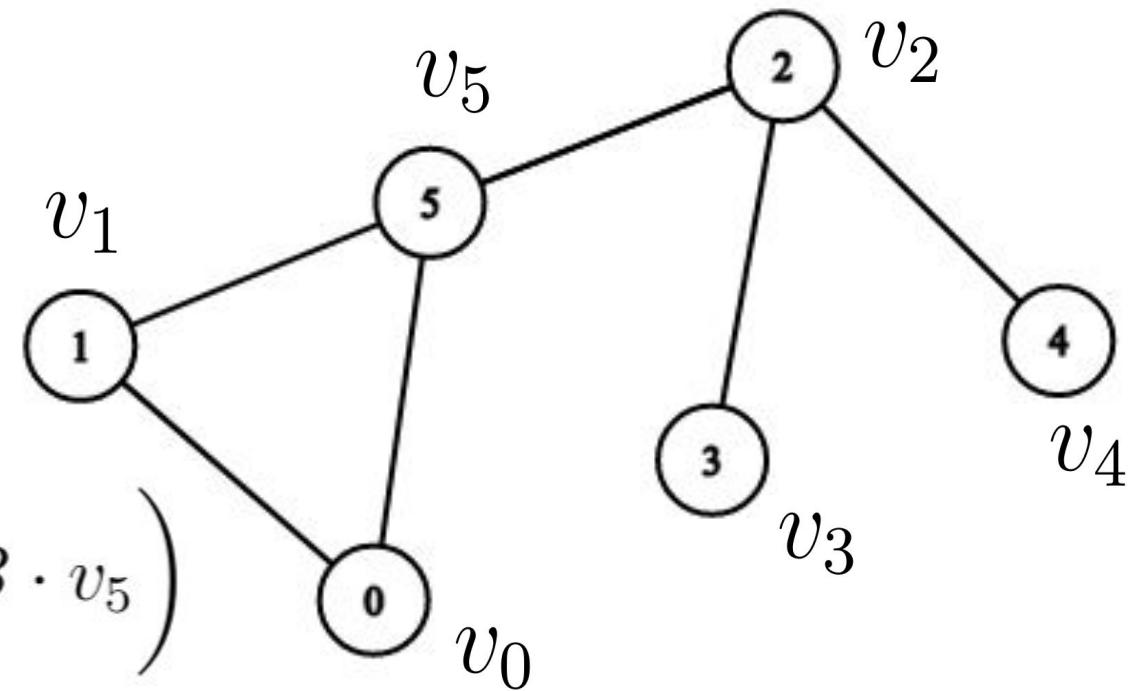
Graph Convolutional Networks: Идея

Чтобы получить эмбеддинги данной вершины, мы:

1. Агрегируем признаки всех соседей
2. Складываем с признаками текущей вершины
(возможно, трансформированным)
3. Применяем нелинейность

например, так:

$$h_5 = \text{ReLU} \left(W \cdot \frac{1}{3}(v_0 + v_1 + v_2) + B \cdot v_5 \right)$$



Graph Convolutional Networks: Идея

А в общем случае вот так:



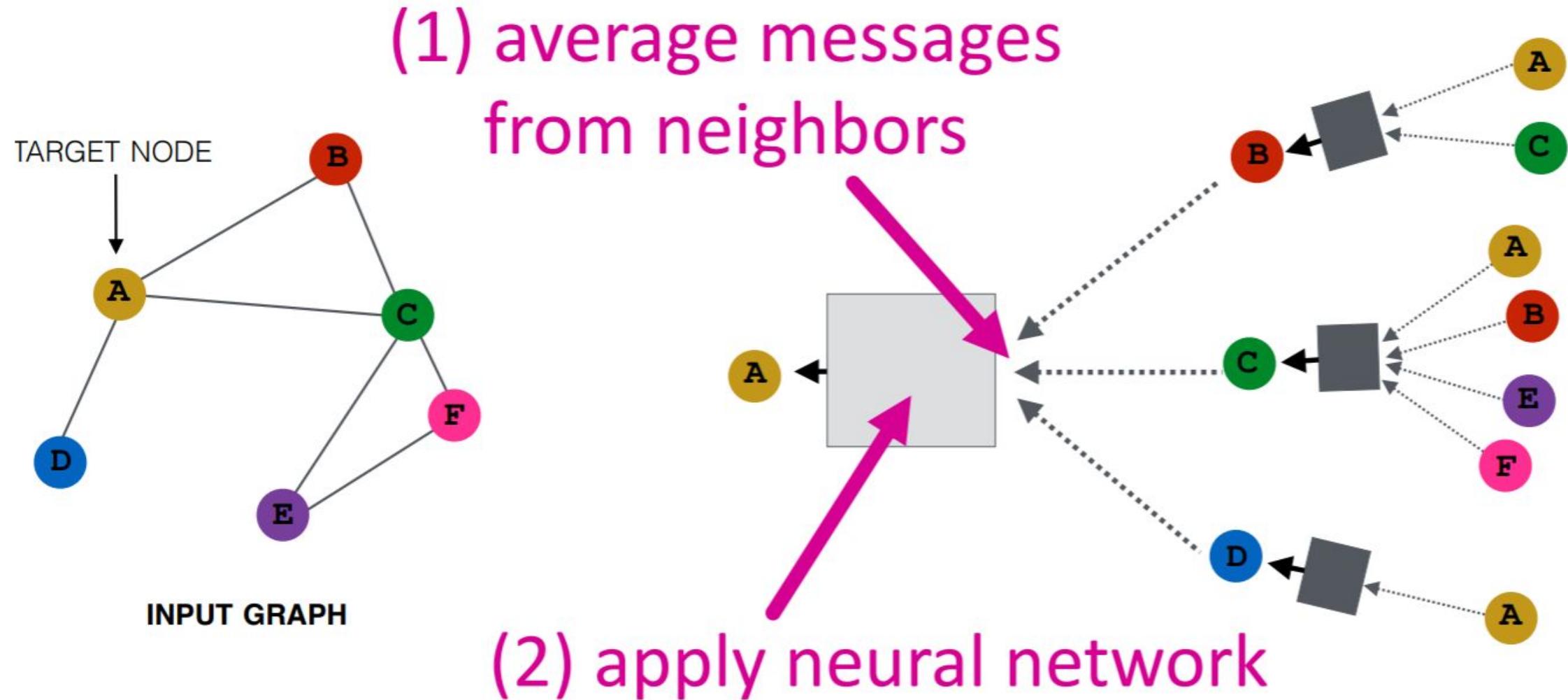
Graph Convolutional Networks: Идея

Что у нас получилось:

- Совсем мало параметров 😊
- Устойчивость к перестановкам вершин и нерегулярной структуре 😊
- Смотрим только на ближайших соседей 😞

Добавим слоев!

Graph Convolutional Networks: Идея

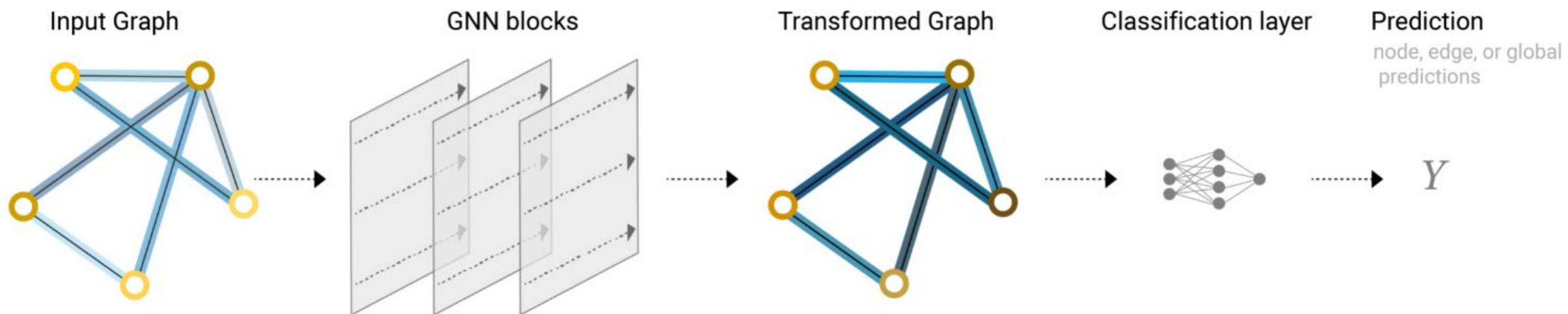


Graph Convolutional Networks: Алгоритм

$$h_v^{(k+1)} = \text{MLP} \left(\text{AGG}(\{h_u^{(k)} \mid u \in N(v)\}), h_v^{(k)} \right) \quad \forall k \in \{0, \dots, K-1\}$$

- Пересчитываем эмбеддинги $(k + 1)$ -ого слоя через эмбеддинги k -ого
- В качестве эмбеддингов на нулевом шаге берём исходные фичи
- Делаем предсказания на основе эмбеддингов последнего слоя
- На каждом слое используем разные параметры

Graph Convolutional Networks: Алгоритм

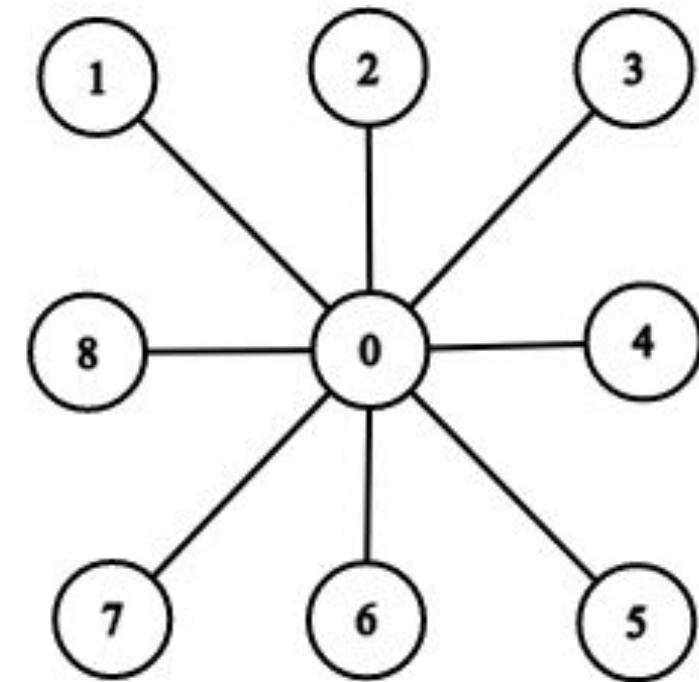
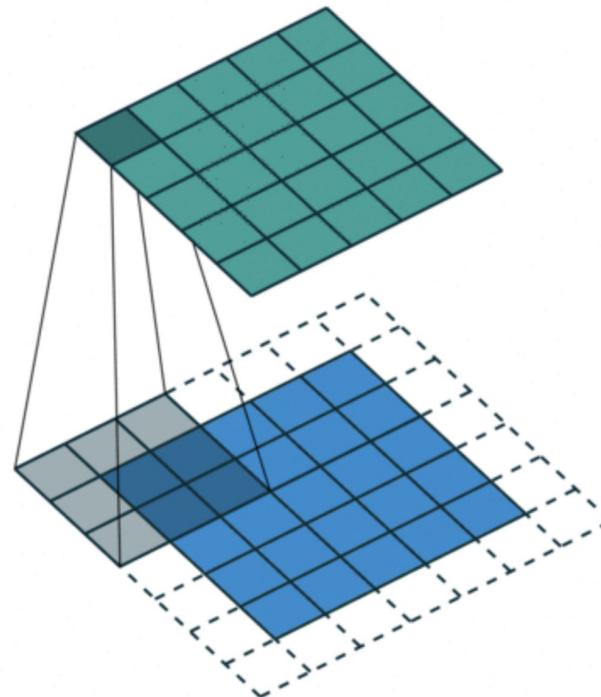


An end-to-end prediction task with a GNN model.

Как можно думать о GCN

Способ 1: для диплёрнеров

Graph Convolutional Networks — это
общий случай свёрточных сетей!

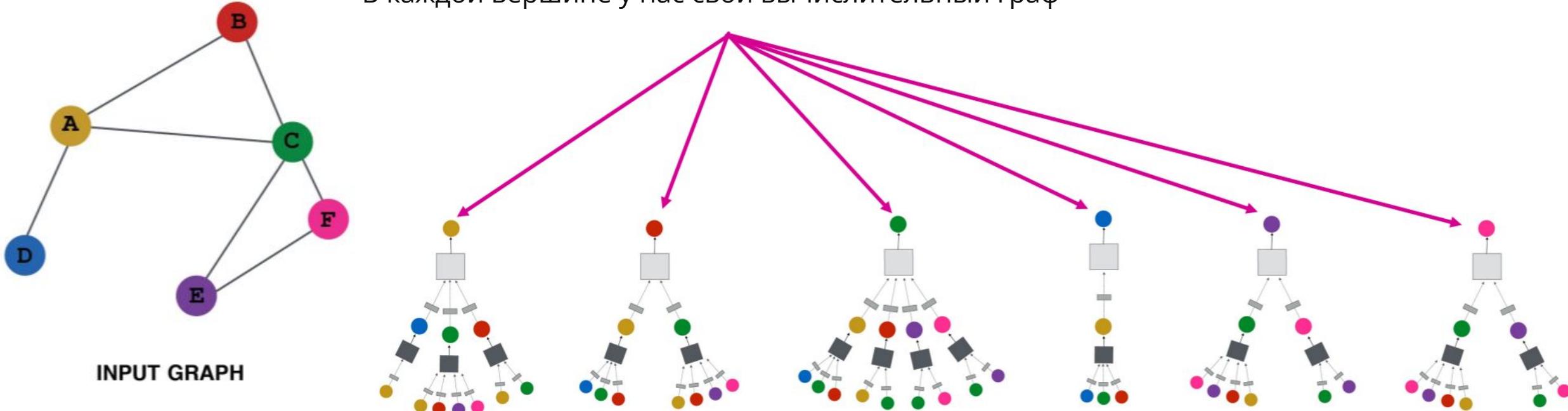


Как можно думать о GCN

Способ 2: для теоретических информатиков

Graph Convolutional Networks — это полносвязные нейронные сети, использующие разные вычислительные графы для разных объектов!

В каждой вершине у нас свой вычислительный граф

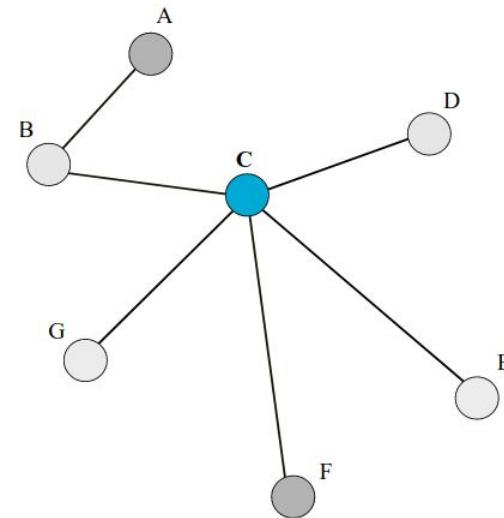


Как можно думать о GCN

Способ 3: для математиков

Graph Convolutional Networks — это полиномы от лапласиана графа!

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \dots + w_d L^d = \sum_{i=0}^d w_i L^i$$

Input Graph G

	A	B	C	D	E	F	G
A	1	-1					
B	-1	2	-1				
C		-1	5	-1	-1	-1	-1
D			-1	1			
E				-1		1	
F				-1		1	
G				-1			1

Laplacian L of G

Продвинутые варианты GNN: GraphSAGE

$$h_v^{(k+1)} = \text{MLP} \left(\text{CONCAT} \left(\text{AGG} \left(\{h_u^{(k)} \mid u \in N(v)\} \right), h_v^{(k)} \right) \right)$$

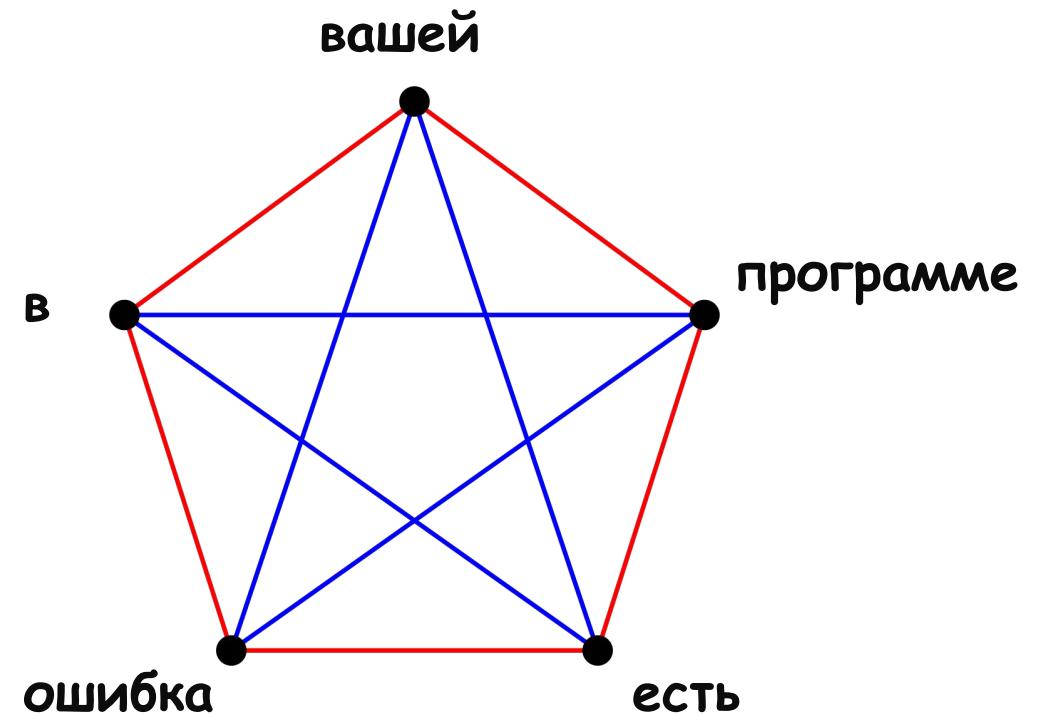
$$h_v^{(l)} \longleftarrow \frac{h_v^{(l)}}{\|h_v^{(l)}\|_2}$$

- Формула такая же, что и у GCN, но мы конкатенируем эмбеддинги соседей с эмбеддингом данной вершины
- Используем не всех соседей, а случайную подвыборку
- Нормируем эмбеддинги

Продвинутые варианты GNN: GAT

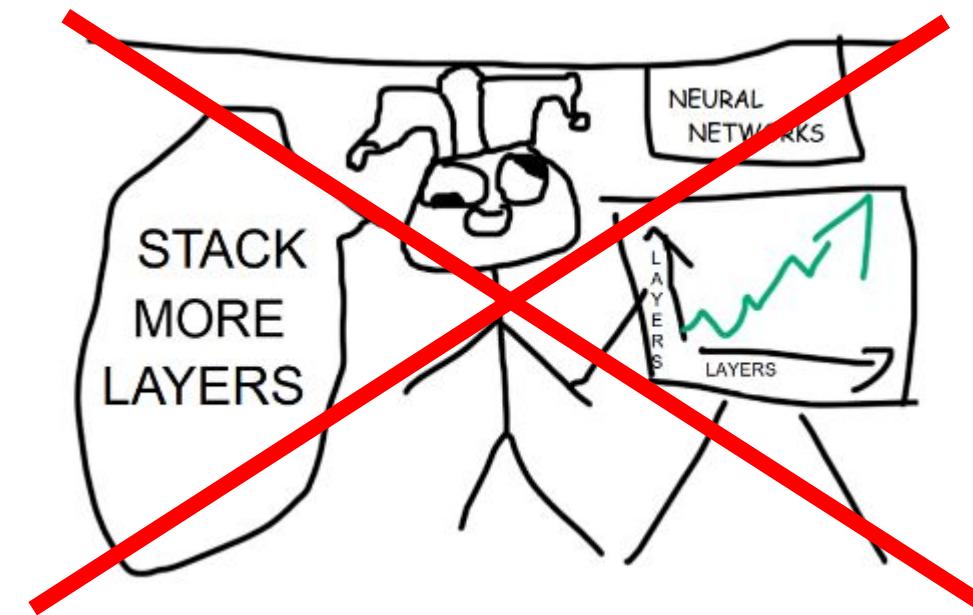
- GAT for Graph Attention Network (for some reason)
- Всё то же самое, что в GCN, но эмбеддинги соседей и самой вершины мы агрегируем при помощи attention
- Трансформеры — это GAT!

в вашей программе
есть ошибка



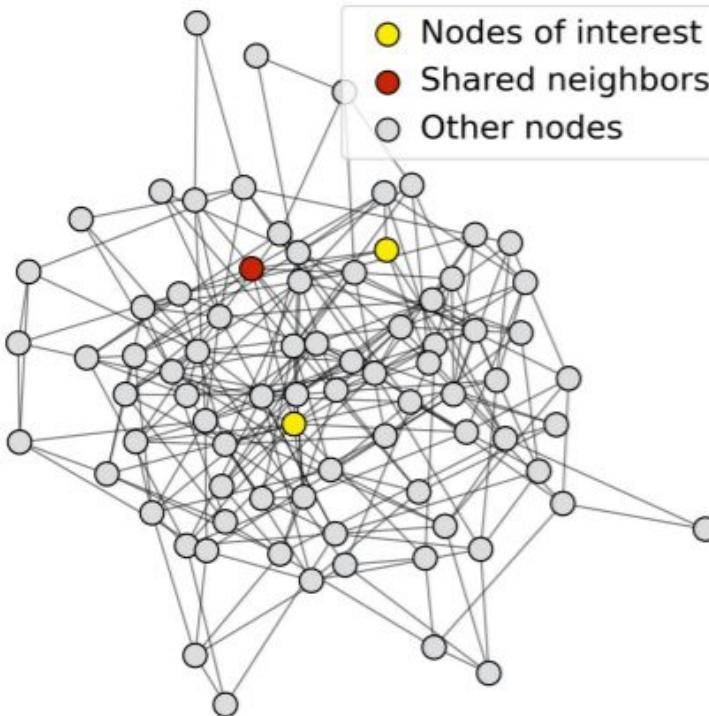
GNN на практике: Stack More Layers не работает!

- Вершины в k -слойной GNN получают информацию от всех соседей на расстоянии k
- Чем больше число слоёв, тем более похожими получаются эмбеддинги разных вершин
- Это плохо, потому что так предсказания для всех вершин становятся одинаковыми

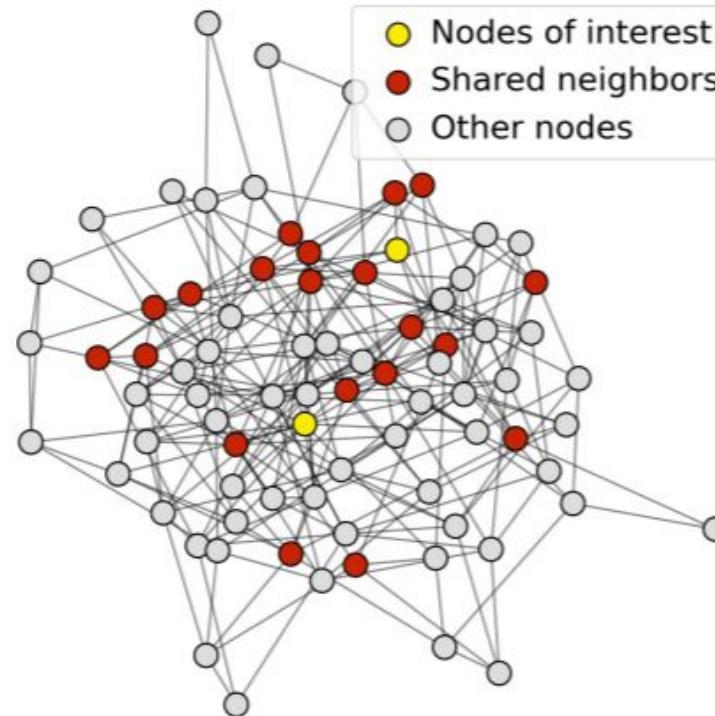


GNN на практике: Stack More Layers не работает!

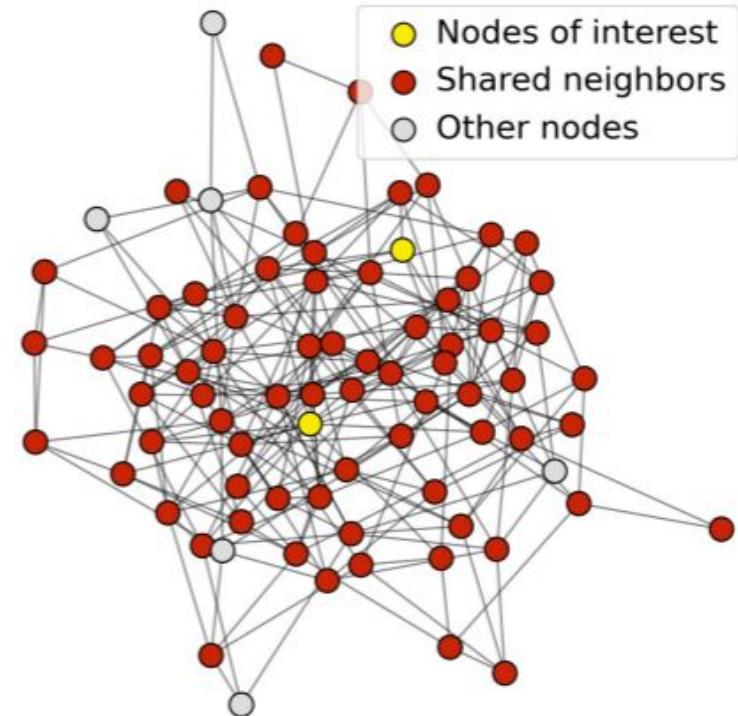
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes

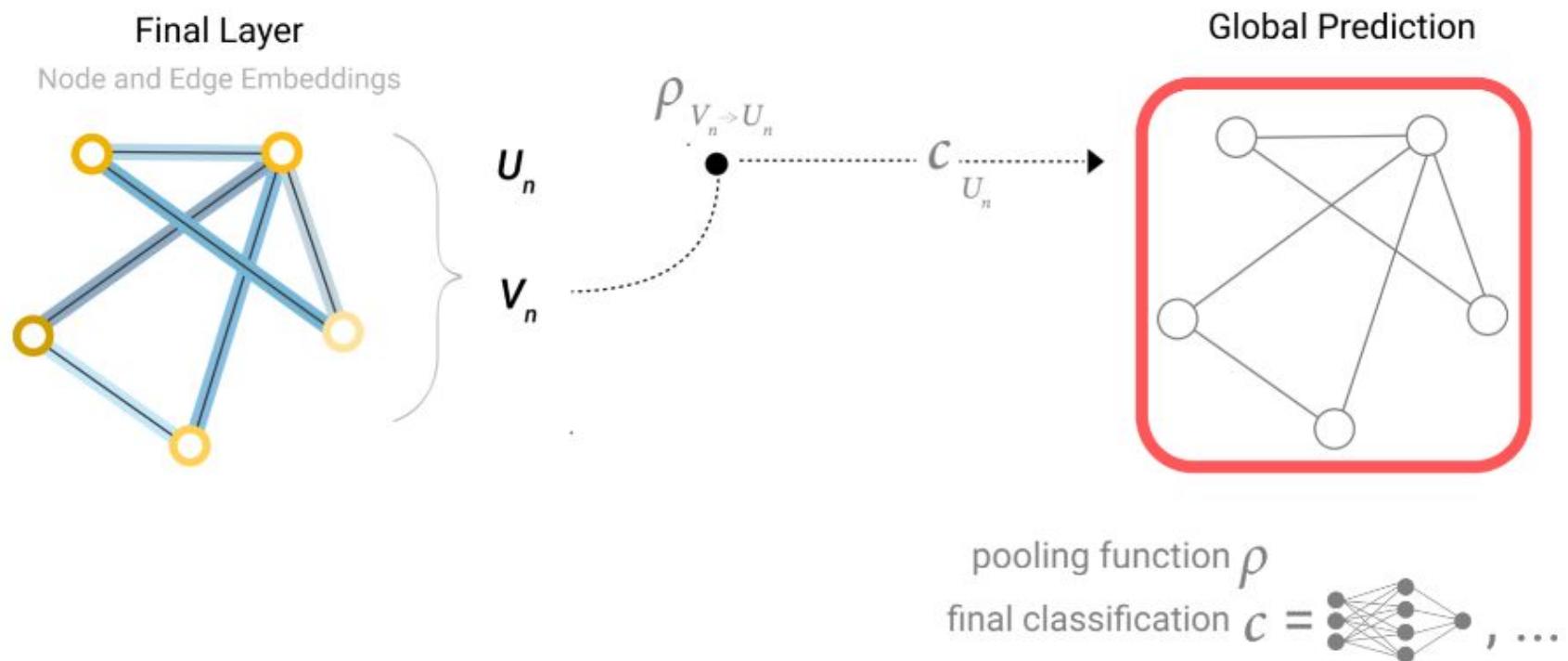


3-hop neighbor overlap
Almost all the nodes!



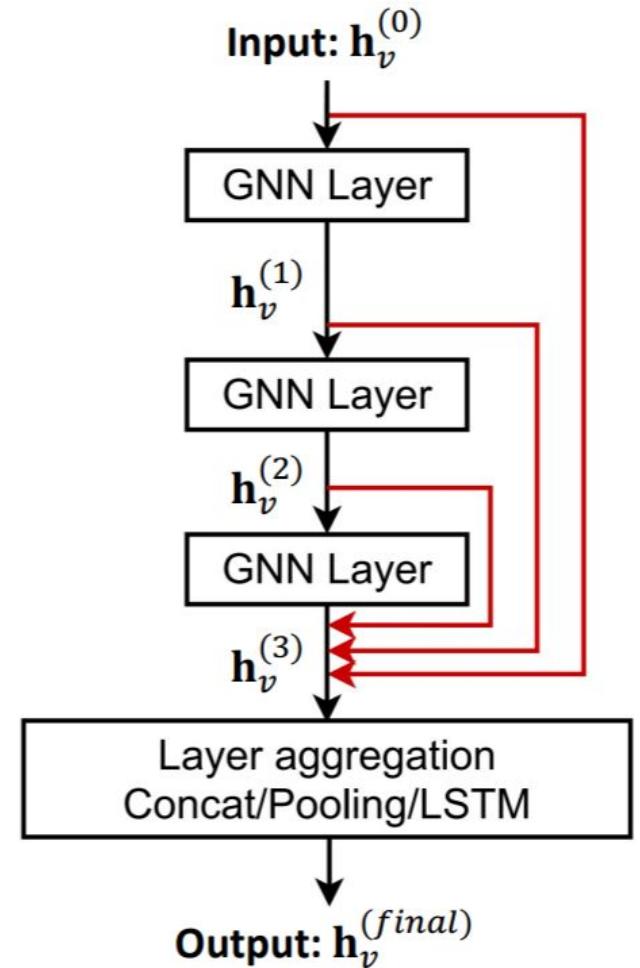
GNN на практике: эмбеддинги не только у вершин

- У рёбер тоже могут быть эмбеддинги
- Можем добавить фиктивную вершину, соседями которой мы будем считать все вершины в графе, и считать её эмбеддинг эмбеддингом всего графа
- Или мы можем просто усреднять эмбеддинги всех вершин



GNN на практике: усложняем модель

- Для ускорения обучения можно брать не всех соседей, а случайную подвыборку
- В графовые нейронные сети тоже можно добавлять skip connections
- А также batch normalization, dropout, etc
- И всё что угодно, что не противоречит здравому смыслу



Список источников

- [Материалы курса CS224w](#) (многие картинки в презентации оттуда)
- [Understanding Convolutions on Graphs](#)
- [A Gentle Introduction to Graph Neural Networks](#)
- [Graph neural networks: A review of methods and applications](#)
- [KG Course 2021](#) (курс по графикам знаний и GNN на русском!)