

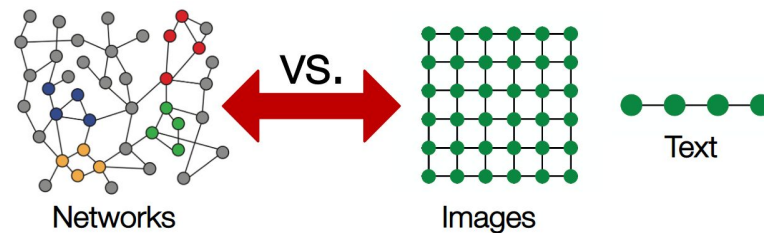
Лекция 7. Методы обучения представлений графов

Дробышевский Михаил
ИСП РАН
3 ноября 2021

Графы и задачи на них

Во многих областях данные имеют графовую структуру:

- социальные: граф дружбы в соцсети, граф научных цитирований;
- техногенные: Интернет, веб, сети дорог, сети авиасообщений;
- в биологии: взаимодействия белков, сложные молекулы.



Задачи машинного обучения на графах и их приложения

supervised, semi-supervised		unsupervised
node classification	link prediction	community detection
• является ли аккаунт ботом? • предсказание возраста/пола/профессии пользователя в соцсети • предсказание функции нового белка на основе его взаимодействий с другими • предсказание тематики статьи на основе ее цитирований	• рекомендация контента в онлайн-платформе • предсказание побочных эффектов лекарств	• поиск пользователей со схожими интересами • выявление функциональных групп белков

Цель: извлечь из графа признаки в виде, пригодном для алгоритмов машинного обучения

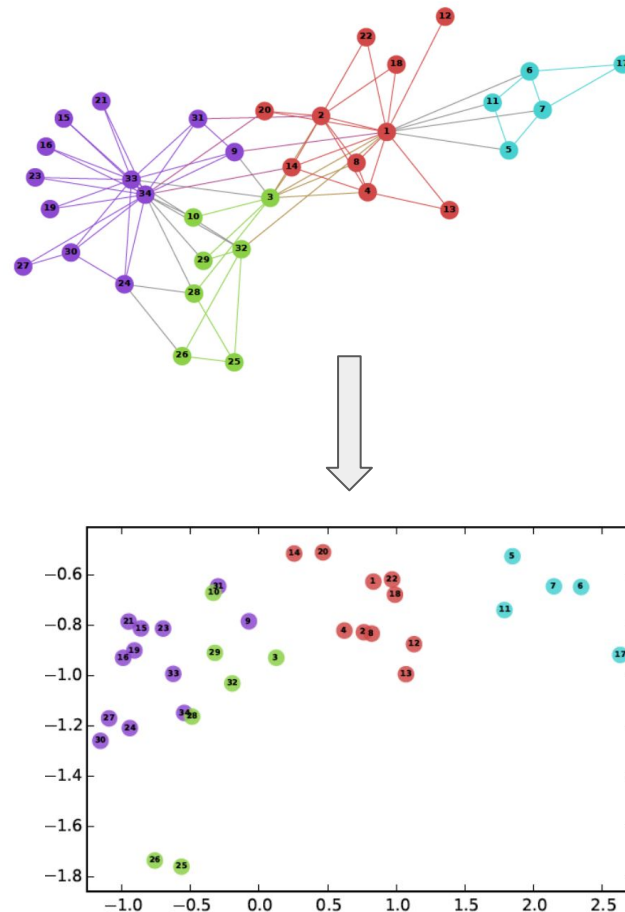
Подходы к обучению представления графов

Задача: найти представление вершин графа в виде векторов (низкоразмерного) пространства, сохраняющее полезную информацию. Обычно, вектора близки в пространстве, если вершины близки в графе.

вложение графа $\approx$ обучение представления
(graph embedding) (representation learning)

Подходы:

1. основанные на матричных разложениях;
2. основанные на случайных блужданиях;
3. основанные на максимизации вероятности восстановления ребер;
4. основанные на глубоком обучении;
5. графовые нейронные сети.



1. Матричные разложения

Задача представления вершин как задача понижения размерности с сохранением информации

Общая идея: представить граф в виде матрицы и разложить ее

1) Locally Linear Embedding (2000)

$$Y_i \approx \sum_j W_{ij} Y_j \quad \varphi(Y) = \sum_i \|Y_i - \sum_j W_{ij} Y_j\|_2^2$$

сводится к нахождению наименьших собственных векторов разреженной матрицы $(I - W)^T(I - W)$

2) Laplacian Eigenmaps (NIPS, 2002)

Идея: представления вершин близки, если вершины связаны

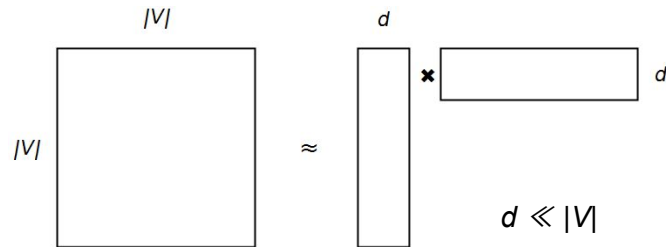
$$\varphi(Y) = \frac{1}{2} \sum_{i,j} \|Y_i - Y_j\|_2^2 W_{ij} = \text{Tr}(Y^T L Y) \quad Y^T D Y = I$$

сводится к нахождению наименьших собственных векторов нормализованного Лапласиана $L_{norm} = D^{-1/2} L D^{-1/2}$

3) Cauchy Graph Embedding (ICML, 2011)

Другая метрика близости $distance = \frac{|Y_i - Y_j|^2}{|Y_i - Y_j|^2 + \sigma^2}$

$$\varphi(Y) = \frac{1}{2} \sum_{i,j} \frac{W_{ij}}{|Y_i - Y_j|^2 + \sigma^2} \quad (\text{max при единичной норме})$$



Обозначения:

$G(V, E)$ – граф с вершинами V , ребрами E

W – матрица смежности с весами,

D – диагональная матрица степеней,

$L = D - W$ – Лапласиан графа,

Y_i – векторное представление вершины i
размерности $d \ll |V|$

$\varphi(Y)$ – функция потерь

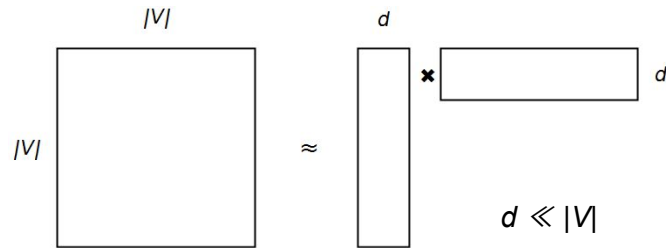
1. Матричные разложения

Основная проблема – сохранение только близости 1-го порядка

Определения:

Близость 1-го порядка между вершинами i и j = вес ребра W_{ij}

Пусть $s_i = [s_{i1}, s_{i2}, \dots, s_{iN}]$ – близость k -го порядка. Тогда близость $(k+1)$ -го порядка между вершинами i и j = мера сходства векторов s_i и s_j



(ориентированный граф – двойные представления Y_s и Y_t)

4) GraRep (CIKM, 2015)

Нормированная матрица переходов $X_{i,j}^k = \log \frac{A_{i,j}^k}{\sum_i A_{i,j}^k} - \log \beta$ $\varphi(Y) = \|X^k - Y_s^k Y_t^{kT}\|_F^2$

Представления для всех k конкатенируются. Недостаток – сложность алгоритма $O(|V|^3)$.

5) High-Order Proximity preserved Embedding (HOPE) (KDD, 2016)

Вместо матрицы смежности взять матрицу близости S (Katz Index, Rooted Page Rank, Common Neighbors, Adamic-Adar score)

$\varphi(Y) = \|S - Y_s Y_t^T\|_F^2$ Сложность алгоритма $O(|E|d^2)$

Основные недостатки алгоритмов матричного разложения:

сохранение близости только 1-го порядка и/или большая сложность алгоритма

Отступление про word2vec

Source Text

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

Training Samples

(the, quick)
(the, brown)

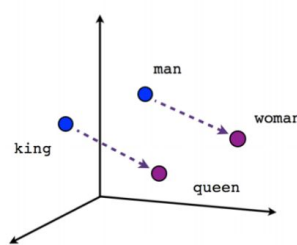
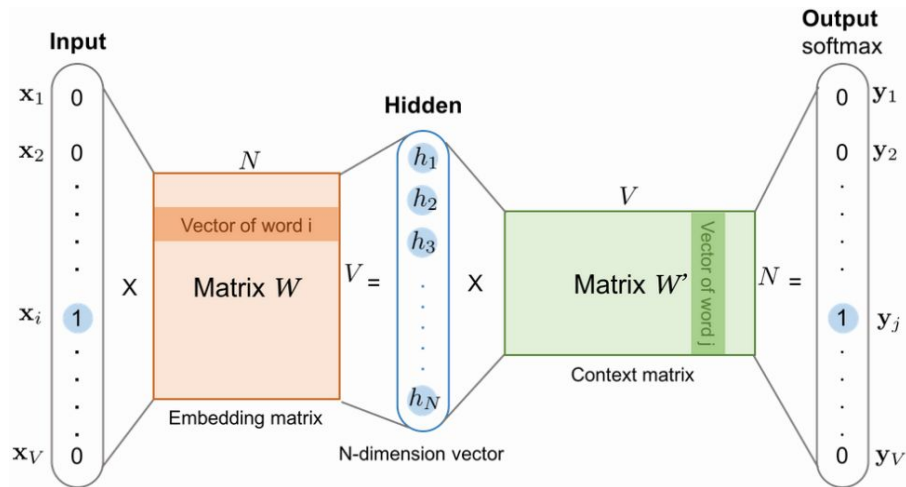
(quick, the)
(quick, brown)
(quick, fox)

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

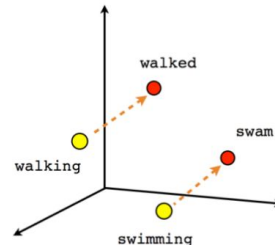
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

$$p(w_O|w_I) = \frac{\exp(v'_{w_O} \top v_{w_I})}{\sum_{i=1}^V \exp(v'_{w_i} \top v_{w_I})}$$

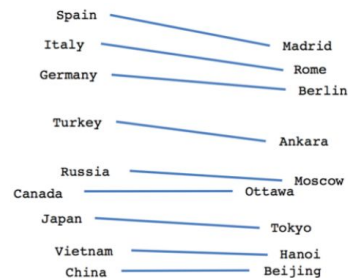
word2vec выучивает векторные представления слов, полезные в прикладных задачах. Вектора показывают интересные семантические свойства



Male-Female



Verb tense

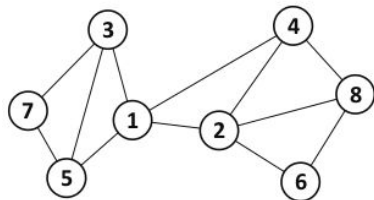


Country-Capital

2. Случайные блуждания (random walks)

1) Deepwalk (KDD, 2014)

Суть: случайные блуждания по графу + word2vec



(1) Random walk

Corpus

#seq.	traveling path
seq.1	1,3,7,3,5,7,5,1
seq.2	4,8,4,2,8,4,1,4
seq.3	5,7,5,1,5,1,2,1
seq.4	8,6,2,6,2,6,8,6
...	...
seq.32	1,3,1,5,7,5,1,5

(2) Sliding window

Training set

target	context
3	1, 7
7	3
3	7, 5
5	3, 7
7	5
...	...

$$\frac{1}{|V|} \sum_{i=1}^{|V|} \sum_{-c \leq j \leq c} \log p(v_{i+j} | Y_i) \rightarrow \max$$

$$p(v_{i+j} | Y_i) = \frac{\exp(Y_{i+j}^T Y_i)}{\sum_{k=1}^{|V|} \exp(Y_k^T Y_i)}$$

Проблема: софтмакс требует $O(|V|)$ операций, на практике используют аппроксимации:

- Иерархический софтмакс (работает за $O(\log |V|)$)
- **Негативное сэмплирование** (контрастное оценивание, Noise-contrastive estimation) – метод оценки параметров

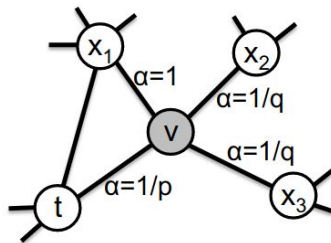
статистической модели $p_d(\cdot) = p_m(\cdot, \alpha^*)$ $Z(\alpha) = \int p_m^0(u, \alpha) du$

Суть: заменить интеграл на новый параметр модели и свести задачу к разделению настоящего и шумового распределений (в данном случае ребра графа vs случайные пары вершин)

2. Случайные блуждания (random walks)

2) **node2vec** (KDD, 2016)

более гибкая версия deerwalk за
счет параметризации блужданий

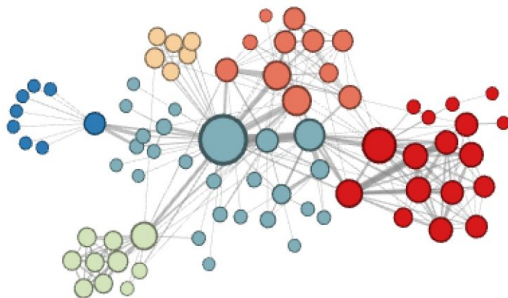


$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

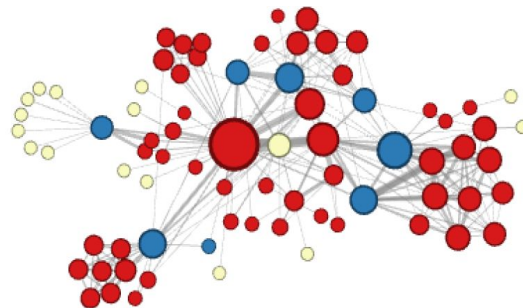
$p \ll 1$ – локальные блуждания (типа BFS)

$q \ll 1$ – исследование в глубину (типа DFS)

Эксперимент:
граф связей персонажей
романа “Отверженные”;
эмбединги кластеризованы



$p=1, q=0.5$
гомофилия, структура сообществ



$p=1, q=2$
структурные роли вершин

3. Максимизация вероятности восстановления ребер

Идея: Вероятностная модель, объясняющая появление ребер графа.

Параметры модели (они же представления вершин) задают вид распределения.

Обучение модели = максимизация вероятности наблюдаемых ребер графа.

1) Bilinear Link Model (BLM) (2015)

$$\theta = \{In_u, Out_u\}_{u \in V} \quad p(v|u, \theta) = \frac{\exp(In_u^T Out_v)}{\sum_{\omega \in V} \exp(In_u^T Out_\omega)}$$

$$J(\theta) = \sum_{i=1}^{|E|} \log p(u_i, v_i | \theta) = \sum_{i=1}^{|E|} \log p(v_i | u_i, \theta) + \sum_{i=1}^{|E|} \log p(u_i) \rightarrow \max_{p(u), \theta}$$

Сложность обучения градиентным спуском $O(|E| |V| d)$, для ускорения используется контрастное оценивание

$$Z_u = \ln \sum_{\omega \in V} \exp(In_u^T Out_\omega) \quad \text{оценка параметров сводится к задаче классификации ребер:}$$

$$p_{NCE}(v|u, \alpha) = \exp(In_u^T Out_v - Z_u) \quad \alpha = \{(In_u, Out_u)\}_{u \in V}, Z_u$$

$$J_{NCE}(\alpha) = \frac{1}{|E|} \left(\sum_{i=1}^{|E|} L_m(u_i, v_i, \alpha) + \sum_{i=1}^{\nu|E|} L_n(\tilde{u}_i, \tilde{v}_i, \alpha) \right) \rightarrow \max_{\alpha}$$

$$L_m(u, v, \alpha) = \ln \frac{p_{NCE}(v|u, \alpha)}{p_{NCE}(v|u, \alpha) + \nu p_n(\tilde{v}_i)}$$

$$L_n(u, v, \alpha) = \ln \frac{\nu p_n(\tilde{v}_i)}{p_{NCE}(\tilde{v}_i|\tilde{u}_i, \alpha) + \nu p_n(\tilde{v}_i)}$$

итоговая сложность:

$O(|E| \nu d)$, ν - число негативных сэмплов (шумовых ребер на 1 реальное ребро)

3. Максимизация вероятности восстановления ребер

2) Large-scale information network embedding (LINE) (WWW, 2015)

Представление вершины состоит из 2 векторов: u_i (вершина сама по себе) и u'_i (в качестве контекста)

близость 1-го порядка = вес ребра

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-u_i^T u_j)} \quad \hat{p}_1(i, j) = \frac{\omega_{ij}}{\sum_{(i,j) \in E} \omega_{ij}} \quad O_1 = - \sum_{(i,j) \in E} \omega_{ij} \log p_1(v_i, v_j)$$

близость 2-го порядка = похожесть контекстов (соседей)

$$p_2(v_j|v_i) = \frac{\exp(u_j'^T u_i)}{\sum_{k=1}^{|V|} \exp(u_k'^T u_i)} \quad \hat{p}_2(v_j|v_i) = \frac{\omega_{ij}}{d_i}, d_i = \sum_{k \in N(i)} \omega_{ik} \quad O_2 = - \sum_{(i,j) \in E} \omega_{ij} \log p_2(v_j|v_i)$$

(минимизируется дивергенция Кульбака-Лейблера между модельным и наблюдаемым распределениями)

O_2 оптимизируется с помощью негативного сэмплирования

После независимой оптимизации O_1 и O_2 результаты представления конкатенируются

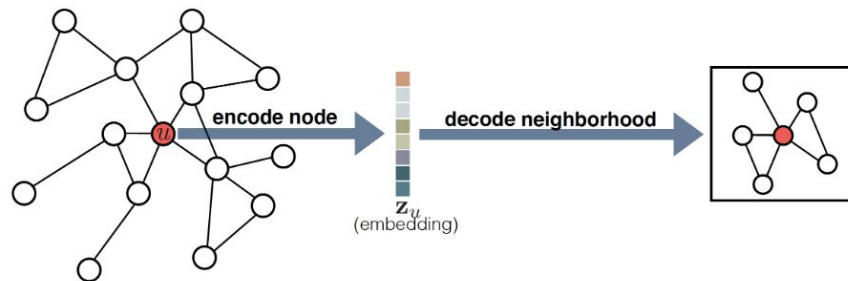
Преимущества: скорость обучения $O(|E|d)$, поэтому применим для больших графов (в т.ч. распределенно)

deepwalk, *node2vec*, *LINE* часто используются как бейзлайны

Выводы о shallow embedding

Где же здесь нейросети?

Три класса, что мы рассмотрели, это **shallow embedding**, т.е. когда представление вершины зависит от ее ID



Недостатки shallow embedding методов:

1. Представления вершин как параметры модели не пересекаются. В результате:
 - число параметров растёт как $O(|V|)$ – многовато для больших графов;
 - наличие общих параметров – потенциально выше эффективность модели + регуляризация;
2. Не используется информация о вершинах (атрибуты);
3. Представления не обобщаются на новые вершины, которые не встречались во время обучения.

Далее рассмотрим:

- глубокие автокодировщики (решают 1 и 3)
- графовые нейросети (решают 1, 2 и 3)

4. Глубокие автокодировщики

Идея: входные данные отображаются в пространство малой размерности, затем восстанавливаются

$$\text{Decoder}(\text{Encoder}(X_i)) = \text{Decoder}(Y_i) \approx X_i$$

1) Structural Deep Network Embedding (SDNE) (KDD, 2016)

близость 2-го порядка: строки матрицы смежности
восстанавливаются автокодировщиком

близость 1-го порядка: представления связанных вершин
близки (см. *Laplacian eigenmaps*)

Устранены недостатки:

- параметры модели – веса, а не представления;
- можно вычислять представление для новых вершин

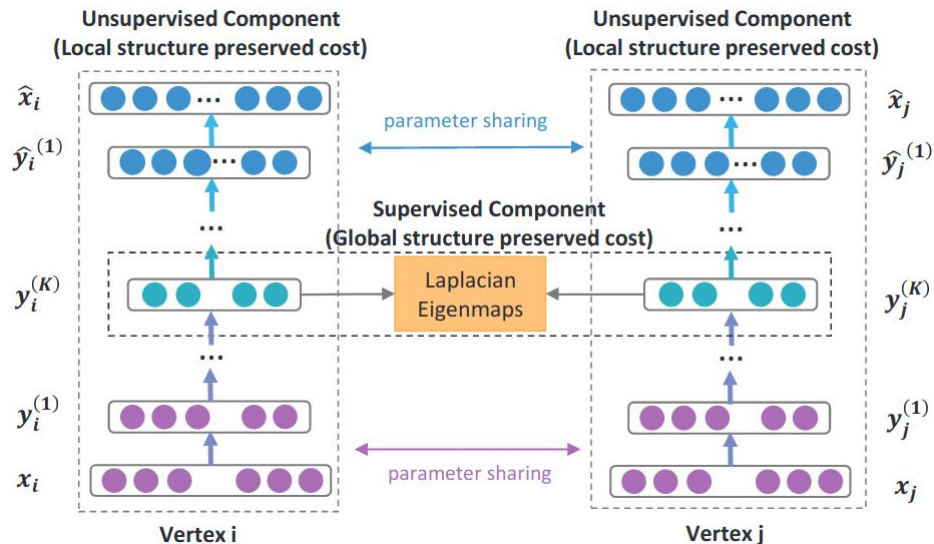


Table 5: precision@k on ARXIV GR-QC for link prediction

Algorithm	$P@2$	$P@10$	$P@100$	$P@200$	$P@300$	$P@500$	$P@800$	$P@1000$	$P@10000$
<i>SDNE</i>	1	1	1	1	1*	0.99**	0.97**	0.91**	0.257**
<i>LINE</i>	1	1	1	1	0.99	0.936	0.74	0.79	0.2196
<i>DeepWalk</i>	1	0.8	0.6	0.555	0.443	0.346	0.2988	0.293	0.1591
<i>GraRep</i>	1	0.2	0.04	0.035	0.033	0.038	0.035	0.035	0.019
<i>Common Neighbor</i>	1	1	1	0.96	0.9667	0.98	0.8775	0.798	0.192
<i>LE</i>	1	1	0.93	0.855	0.827	0.66	0.468	0.391	0.05

Significantly outperforms Line at the: ** 0.01 and * 0.05 level, paired t-test.

4. Глубокие автокодировщики

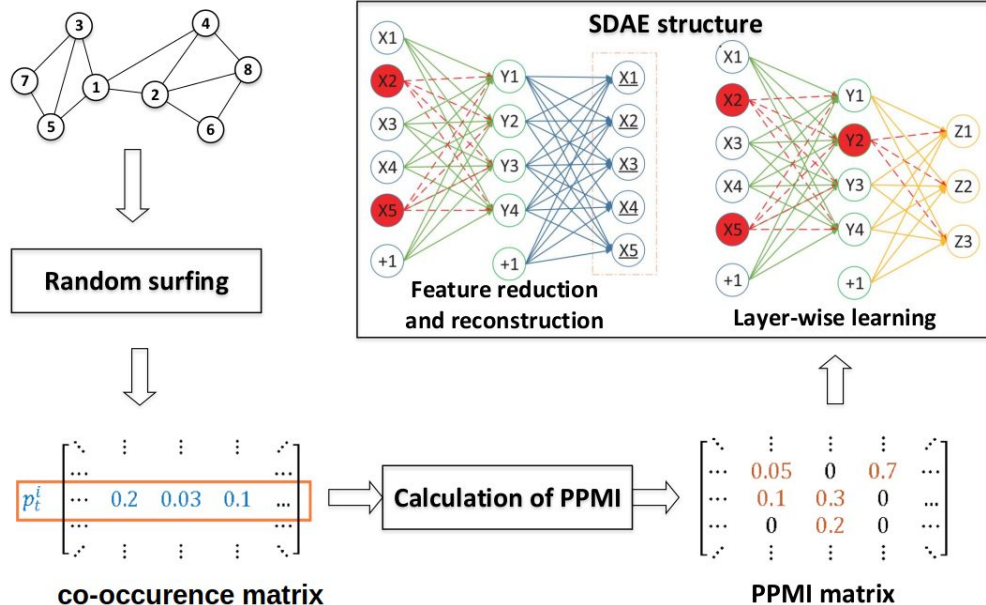
2) Deep Neural Networks for Learning Graph Representations (DNGR) (2016, AAAI)

Алгоритм состоит из 3 шагов:

1. случайные блуждания → матрица встречаемости вершин
2. вычисление матрицы поточечной взаимной информации (сглаживает эффект часто встречающихся вершин)

$$\text{PPMI}_{v_1, v_2} = \max(\log(\frac{\text{count}(v_1, v_2) \cdot |D|}{\text{count}(v_1)\text{count}(v_2)}), 0)$$

3. использование *stacked denoising autoencoder* (SDAE) для обучения нелинейных зависимостей
(*stacked*: несколько автокодировщиков друг за другом
denoising: произвольные элементы входа зануляются)



DNGR вычисляет полезные представления, однако настраивать нейросеть сложно

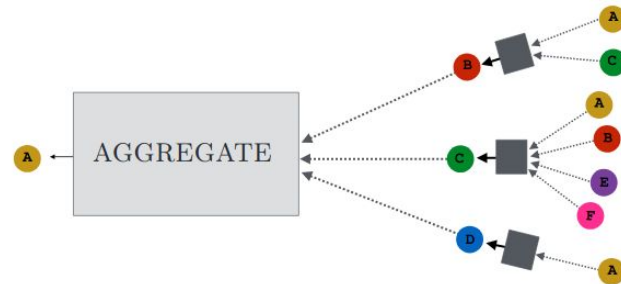
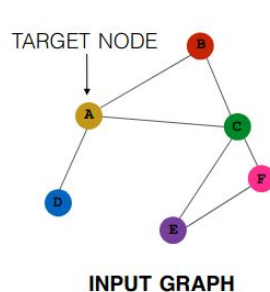
5. Графовые нейронные сети (GNN)

Идея: обучать представления на основе структуры и атрибутов вершин одновременно с помощью нейросети

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u$$

$$\begin{aligned}\mathbf{h}_u^{(k+1)} &= \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \right) \\ &= \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)} \right),\end{aligned}$$

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}$$



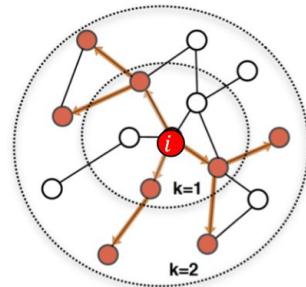
Обычно:

$$\mathbf{h}_u^{(k)} = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right) \quad \text{и другие вариации AGGREGATE и UPDATE}$$

Обучается под конкретную задачу (классификация, предсказание ребра и т.д.)

Особенности и отличия GNN от предыдущих методов:

- GNN это совместное обучение структуры с признаками вершин
- GNN как система распространения информации по графу; число слоев = глубина обхода
- GNN это обучение с учителем, под конкретную задачу
- размер модели GNN почти не зависит от размера графа (до 3 млрд вершин на 2019 год)

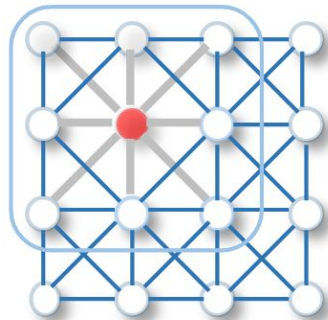


5. Графовые нейронные сети (GNN)

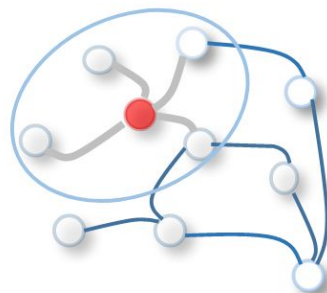
1) Graph Convolutional Networks (GCN) (ICLR, 2016)

симметрия (наравне с соседями) + нормализация (на степень)

$$\mathbf{h}_u^{(k)} = \sigma \left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)| |\mathcal{N}(v)|}} \right)$$

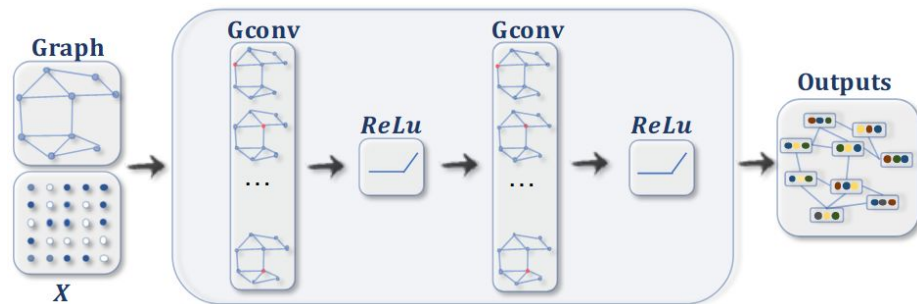


(a) 2D Convolution.

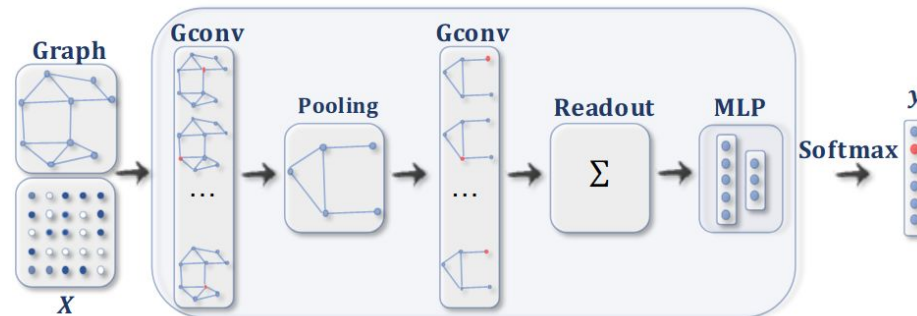


(b) Graph Convolution.

(а почему бы просто не обучать сверточную сеть на матрице смежности? зависимость от нумерации соседей)



Классификация вершин
5% размеченных достаточно



Классификация графов
добавляется *pooling* и *readout*

5. Графовые нейронные сети (GNN)

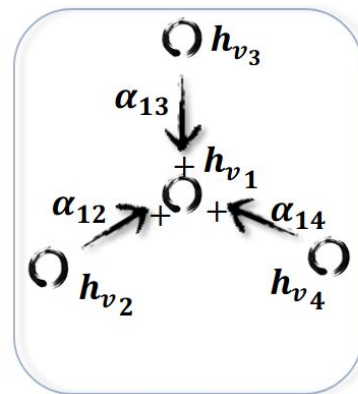
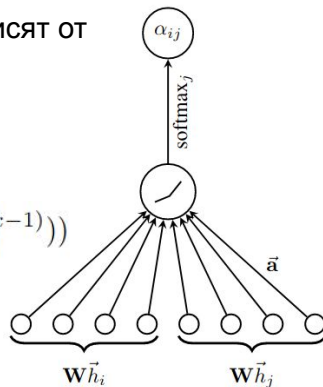
2) Graph Attention Network (GAT) (ICLR, 2017)

Идея: веса соседей при AGGREGATE зависят от параметров модели $\mathbf{a}$

$$\mathbf{h}_v^{(k)} = \sigma \left(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)} \right),$$

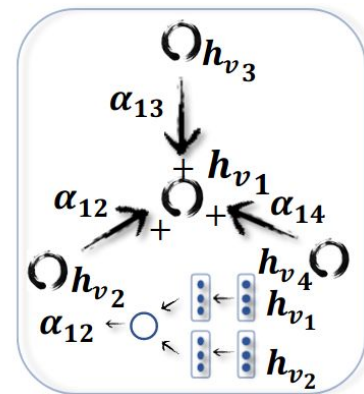
$$\alpha_{vu}^{(k)} = \text{softmax}(g(\mathbf{a}^T [\mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)} \parallel \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}]))$$

в качестве $g(\cdot)$ – LeakyReLU



GCN: веса фиксированные

$$a_{ij} = \frac{1}{\sqrt{\deg(v_i) \deg(v_j)}}$$



GAT: веса обучаются;
важные вершины
получают больший вес

Механизм *attention* значительно увеличивает выразительную мощность сети и качество классификации вершин

5. Графовые нейронные сети (GNN)

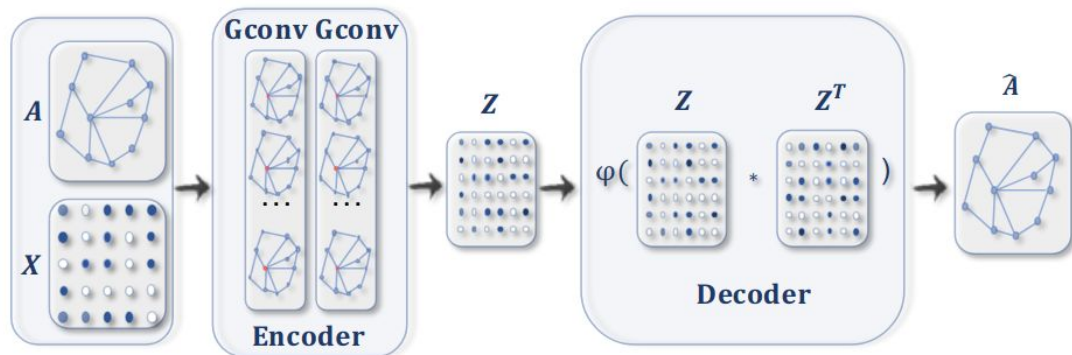
3) Graph autoencoders (GAE) (с 2016го)

Идея: автокодировщик для графа, с учетом еще и атрибутов вершин

энкодер (2 сверточных слоя GCN)

вычисляет представления вершин

декодер считает попарные произведения и реконструирует матрицу смежности

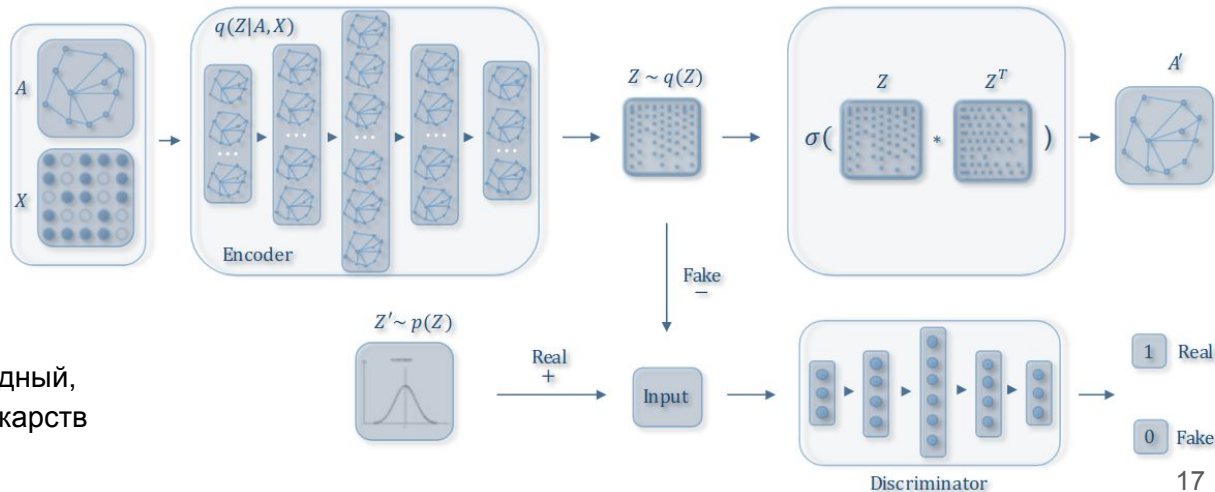


Решает задачу предсказания ребра

4) Adversarially Regularized Graph Autoencoder (ARGA) (2018)

Обучает представления для графа в стиле генеративно-сопоставительной сети

Генерирует новые графы, похожие на исходный, что полезно для поиска новых молекул лекарств



Полезные ссылки

- Книга 2020 года “Graph Representation Learning” https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf
- Детальный разбор процесса обучения word2vec <https://arxiv.org/pdf/1411.2738.pdf>
- Оригинальная статья про Noise contrastive estimation <http://proceedings.mlr.press/v9/gutmann10a/gutmann10a.pdf>
- DGL – библиотека, где реализованы многие графовые нейросети <https://www.dgl.ai/pages/about.html>