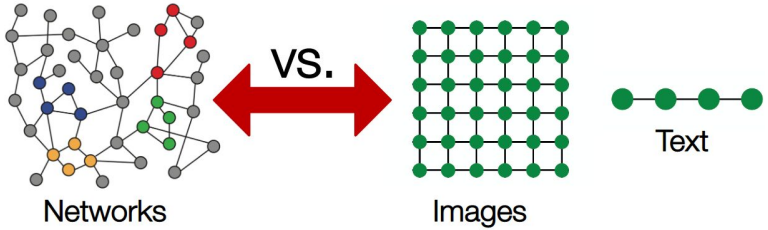
Лекция 7. Нейронные сети для анализа графов

Дробышевский Михаил
ИСП РАН
11 ноября 2020

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.



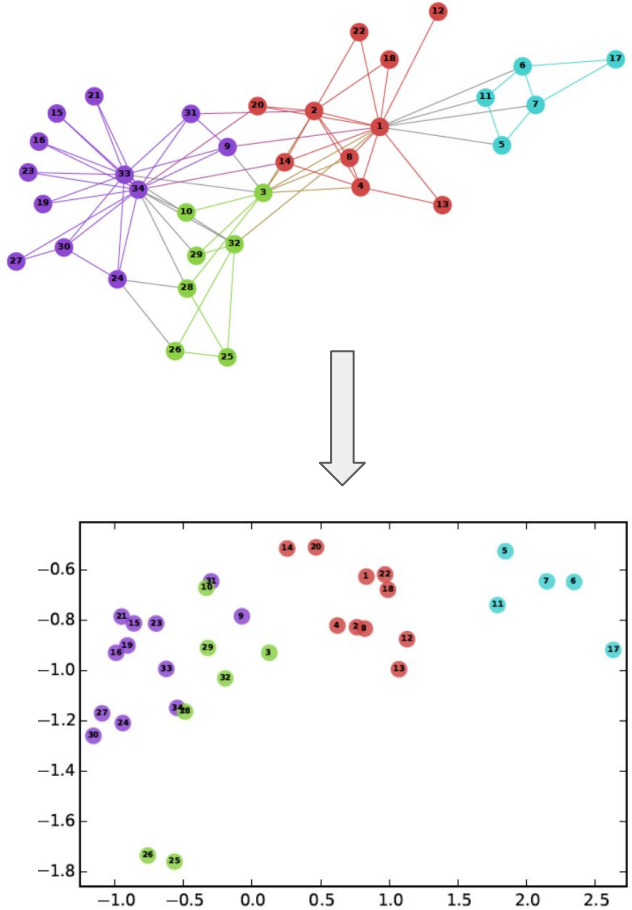
Задачи машинного обучения на графах и их приложения

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.

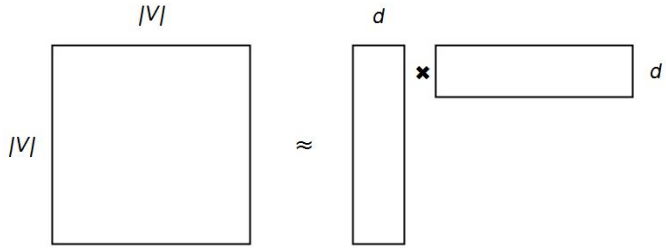


вложение графа ≈ **обучение представления**
(graph embedding) (representation learning)

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)

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

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

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

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 $distance = \dfrac{\left|Y_i - Y_j\right|^2}{\left|Y_i - Y_j\right|^2 + \sigma^2}$ $\qquad \varphi(Y) = \dfrac{1}{2} \sum_{i,j} \dfrac{W_{ij}}{\left|Y_i - Y_j\right|^2 + \sigma^2}$

Обозначения:

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

W – матрица смежности с весами,

D – диагональная матрица,

L = D - W – Лапласиан графа,

Y_i – векторное представление вершины *i* размерности *d*

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

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 $O(|V|^3)$.

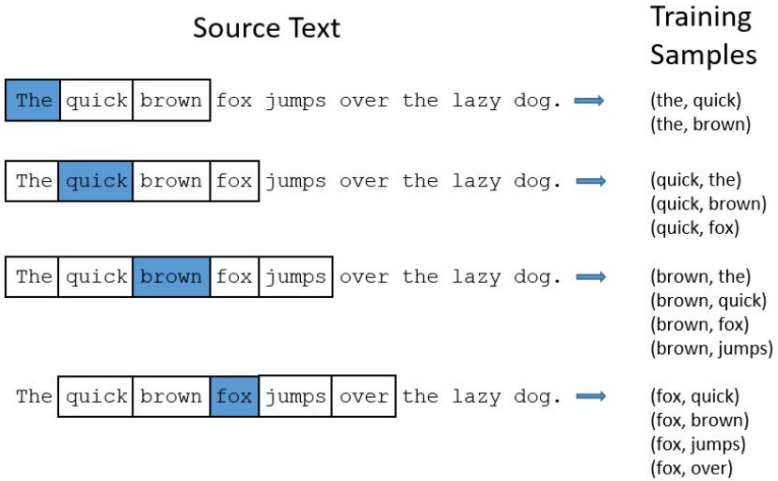
5) **High-Order Proximity preserved Embedding (HOPE)** (KDD, 2016)

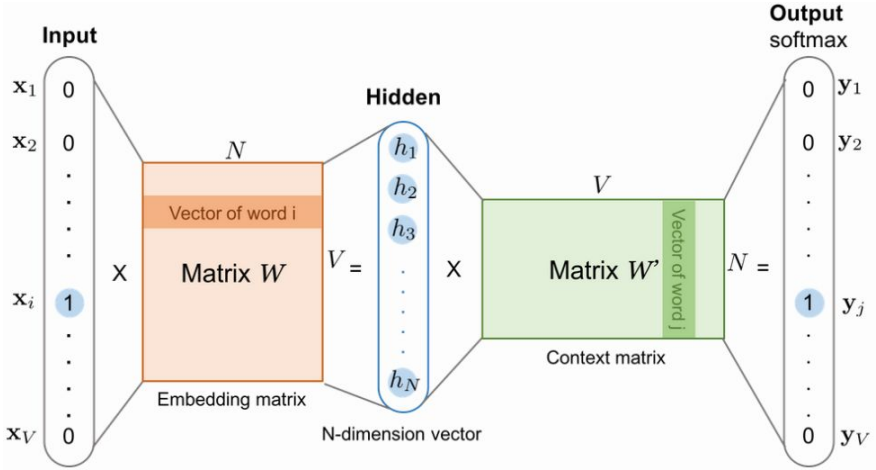
Вместо матрицы смежности взять матрицу близости *S* (Katz Index, Rooted Page Rank, Common Neighbors, Adamic-Adar score)

$\varphi(Y) = ||S - Y_s Y_t^T||_F^2$ (ориентированный граф – двойные представления Y_s и Y_t). Сложность алгоритма $O(|E|d^2)$

Основные недостатки алгоритмов матричного разложения:
сохранение близости только 1-го порядка и/или большая сложность алгоритма

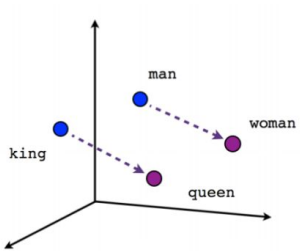
Отступление про word2vec



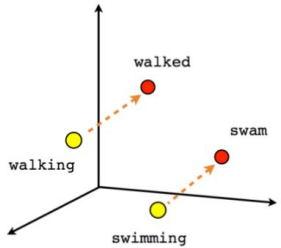


$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{i=1}^{V} \exp({v'_{w_i}}^\top v_{w_I})}$$

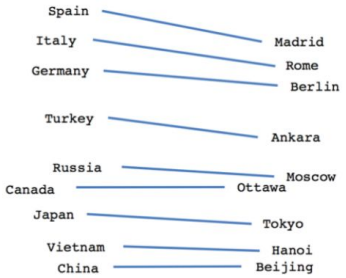
word2vec выучивает векторные представления слов, полезные в прикладных задачах.
Вектора показывают интересные семантические свойства



Male-Female



Verb tense

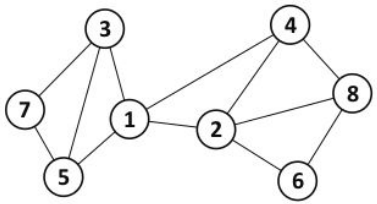


Country-Capital

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
...	...

$$\frac{1}{|V|}\sum_{i=1}^{|V|}\sum_{-c\le j\le c}\log p(v_{i+j}|Y_i)\to\max \qquad p(v_{i+j}|Y_i)=\frac{\exp(Y_{i+j}^T Y_i)}{\sum_{k=1}^{|V|}\exp(Y_k^T Y_i)}$$

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

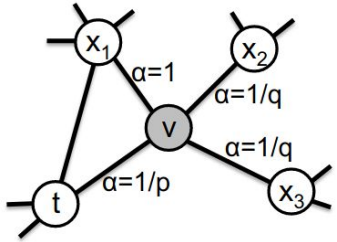
- Иерархический софтмакс (работает за *O(log|V|)*)
- **Негативное сэмплирование** (контрастное оценивание, Noise-contrastive estimation) – метод оценки параметров статистической модели $p_d(.) = p_m(., \alpha^*)$ $Z(\alpha) = \int p_m^0(u, \alpha)du$

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

2. Случайные блуждания (random walks)

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

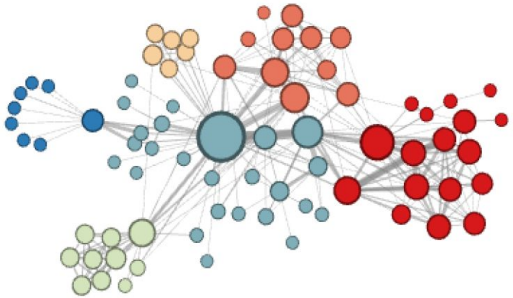
более гибкая версия deepwalk за счет параметризации блужданий



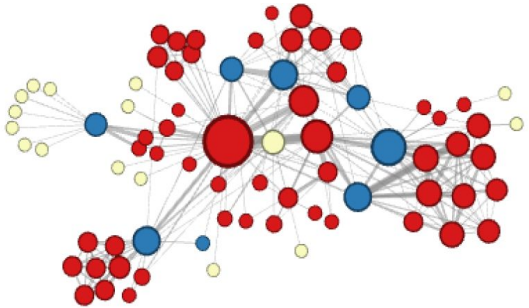
$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

$p \ll 1$ – 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)

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

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

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

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

$$J_{NCE}(\alpha) = \frac{1}{|E|}\left(\sum_{i=1}^{|E|} L_m(u_i, v_i, \alpha) + \sum_{i=1}^{\nu|E|} L_n(\tilde{u}_i, \tilde{v}_i, \alpha)\right) \to \max_{\alpha} \qquad \alpha = \{(In_u, Out_u)\}_{u \in V}, Z_u$$

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

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

итоговая сложность:
O(|E|vd), v - 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

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

близость 2-го порядка = похожесть контекстов (соседей)

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

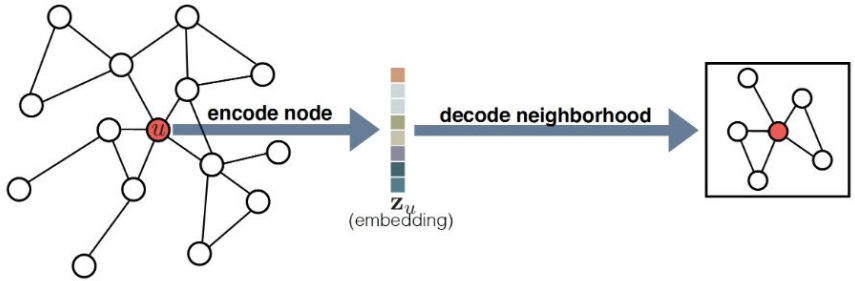
(минимизируется дивергенция Кульбака-Лейблера между модельным и наблюдаемым распределениями)

O_2 оптимизируется с помощью негативного сэмплирования

После независимой оптимизации O_1 и O_2 результаты представления конкатенируются

Выводы о shallow embedding

Где же здесь нейросети?
Почти все, 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 растет как *O(|V|)* – слишком много 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

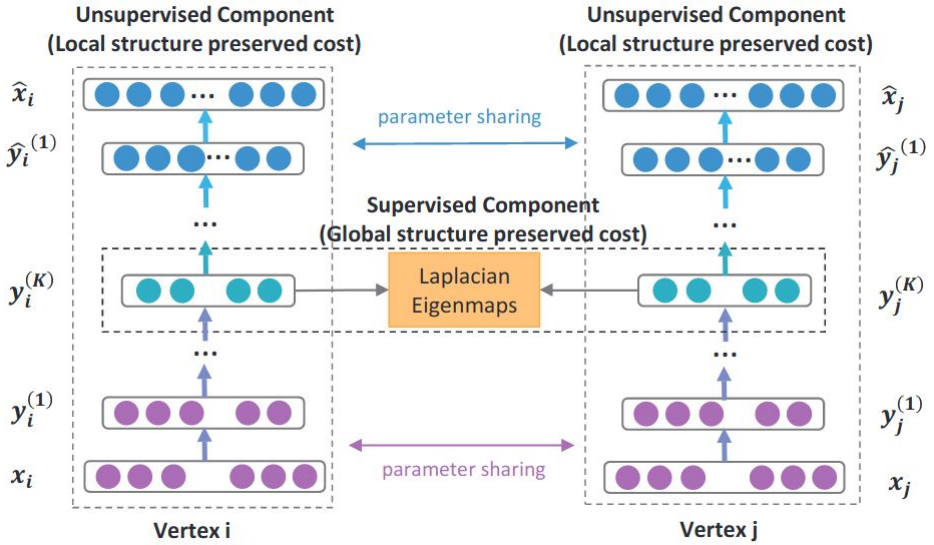
Идея: входные данные отображаются в пространство малой размерности, затем восстанавливаются

$$Decoder(Encoder(X_i)) = Decoder(Y_i) \approx X_i$$

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:

- есть общие параметры,
- можно вычислять представление для новых вершин

Table 4: MAP on ARXIV-GRQC and BLOGCATALOG on reconstruction task

Method	ARXIV-GRQC					BLOGCATALOG				
	SDNE	*GraRep*	*LINE*	*DeepWalk*	*LE*	*SDNE*	*GraRep*	*LINE*	*DeepWalk*	*LE*
MAP	**0.836****	0.05	0.69	0.58	0.23	**0.63****	0.42	0.58	0.28	0.12

Significantly outperforms GraRep at the: ** 0.01 level.

4. Глубокие автокодировщики

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

Алгоритм состоит из 3 шагов:

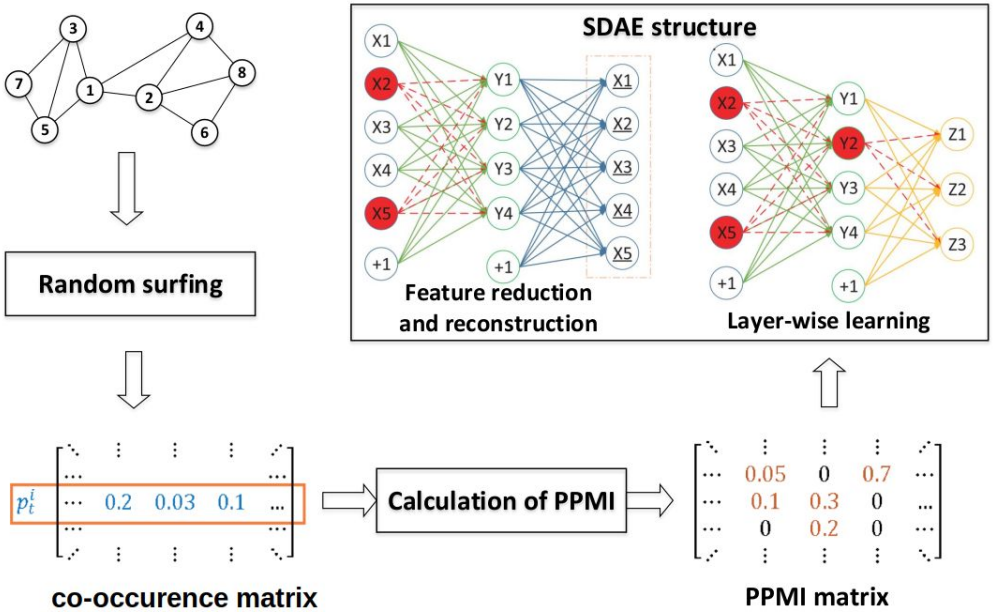
1. случайные блуждания → матрица встречаемости вершин
2. вычисление матрицы поточечной взаимной информации (сглаживает эффект часто встречающихся вершин)

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

3. использование stacked denoising autoencoder для обучения нелинейных зависимостей

(*stacked*: несколько автокодировщиков друг за другом

denoising: произвольные элементы входа зануляются)



5. Графовые нейронные сети (GNN)

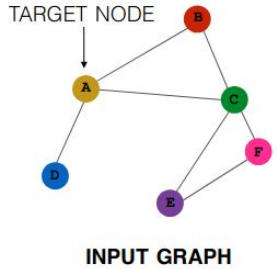
Идея: обучать представления на основе структуры и атрибутов вершин одновременно с помощью нейросети

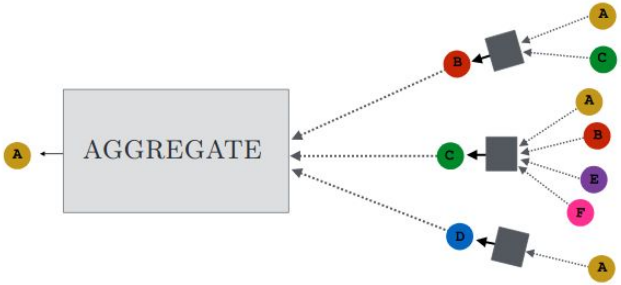
$$\mathbf{h}_u^{(0)} = \mathbf{x}_u$$

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

$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right),$$

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

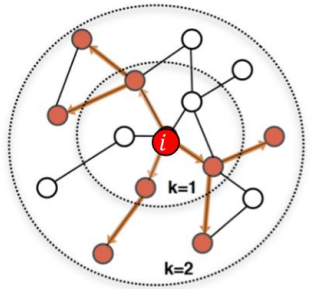




Обычно:

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

и другие вариации AGGREGATE и UPDATE



Особенности и отличия GNN от предыдущих методов:

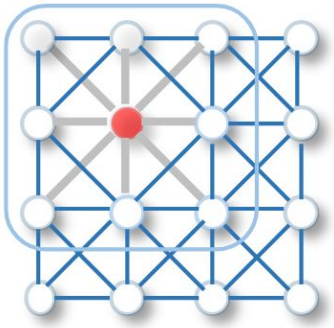
- GNN как система распространения информации по графу; число слоев = глубина обхода
- GNN это совместное обучение структуры с признаками вершин
- GNN это обучение с учителем, под конкретную задачу
- размер модели GNN почти не зависит от размера графа (до 3 млрд вершин на 2019 год)

5. Графовые нейронные сети (GNN)

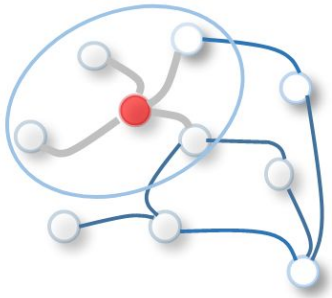
1) **Graph Convolutional Networks (GCN)** (ICLR, 2016)

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

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

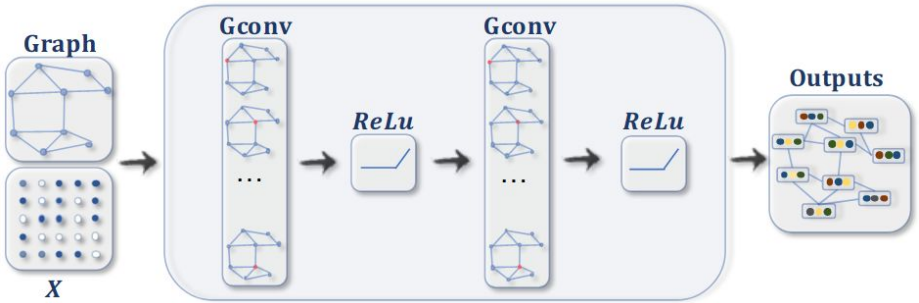


(a) 2D Convolution.

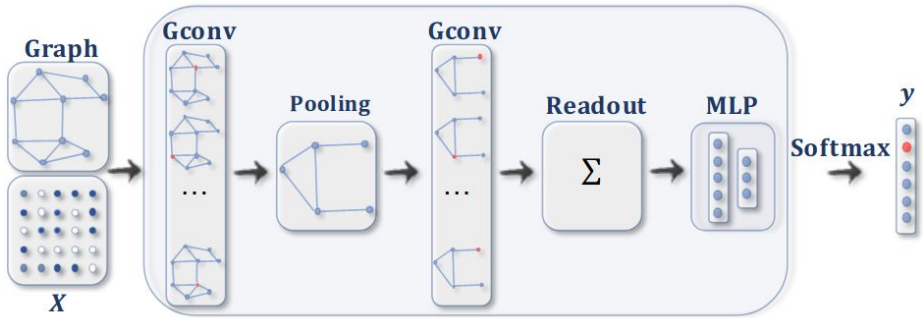


(b) Graph Convolution.

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



Классификация вершин



Классификация графов
добавляется *pooling* и *readout*

5. Графовые нейронные сети (GNN)

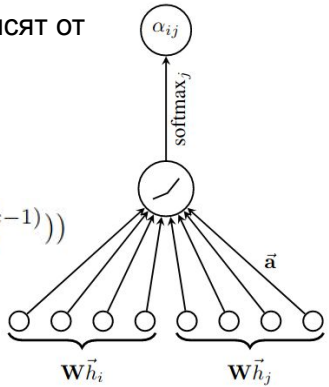
2) **Graph Attention Network (GAT)** (ICLR, 2017)

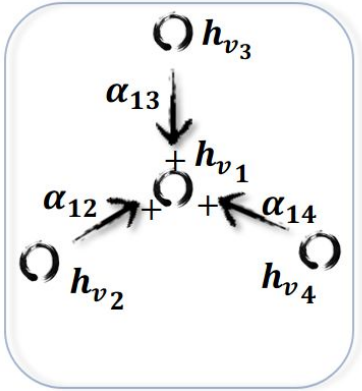
Идея: веса соседей при AGGREGATE зависят от параметров модели ***a***

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

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

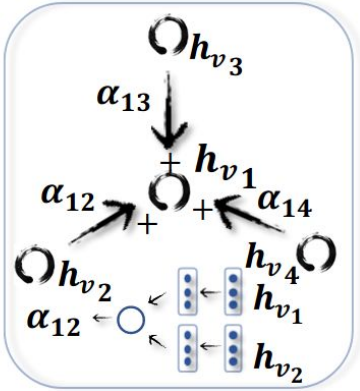
в качестве *g(·)* – LeakyReLU 





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

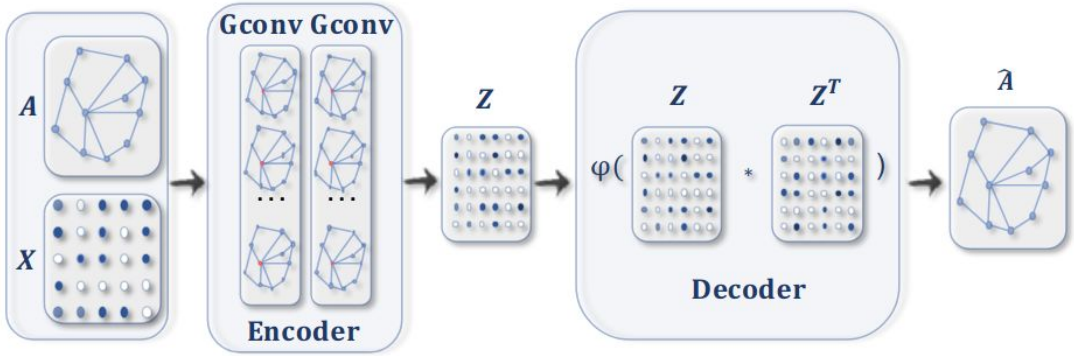
5. Графовые нейронные сети (GNN)

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

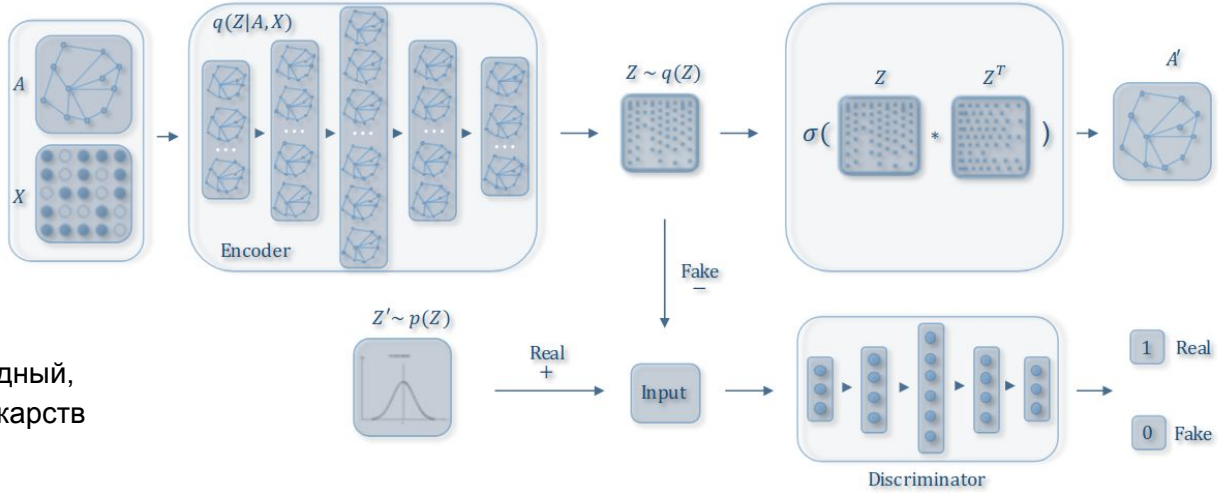


Решает задачу предсказания ребра

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

Обучает представления для графа в стиле генеративно-состязательной 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 Noice contrastive estimation http://proceedings.mlr.press/v9/gutmann10a/gutmann10a.pdf
- DGL – библиотека, где реализованы многие графовые нейросети https://www.dgl.ai/pages/about.html