

Deep Learning

Lecture 13
Graph Neural Network

Recap

Autoregressive models: PixelCNN, LMConv

Generative Adversarial Networks: theoretical results, vanishing gradients, mode collapse

Generative Adversarial Networks: Unrolled GAN, Energy-based GAN, Wasserstein GAN

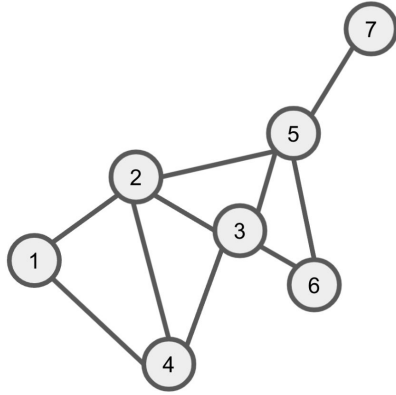
Evaluation metrics for GAN: Inception Score, Frechet Inception Distance

GAN's representative: Cycle GAN

Источники

- [Курс на ODS](#)
- [Курс cs224W](#)
- [Подборка статей на towardsdatascience.com](#)

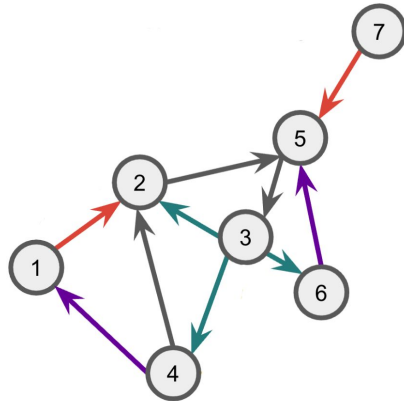
Графовые данные



Контакты людей

Связь молекул

Железнодорожные пути и дороги



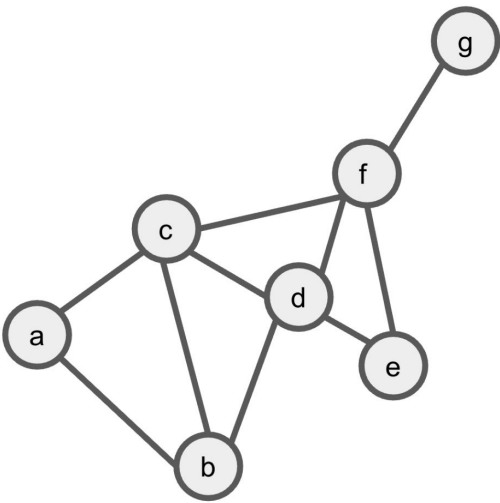
Цитирование

Подписки в соцсетях

Просмотренные видео на youtube

Граф – набор данных

Граф $G(V, E)$:



Матрица смежности

1	1	0	1	0	0	0
1	1	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	0	0	0
0	1	1	0	1	1	1
0	0	1	0	1	1	0
0	0	0	0	1	0	1

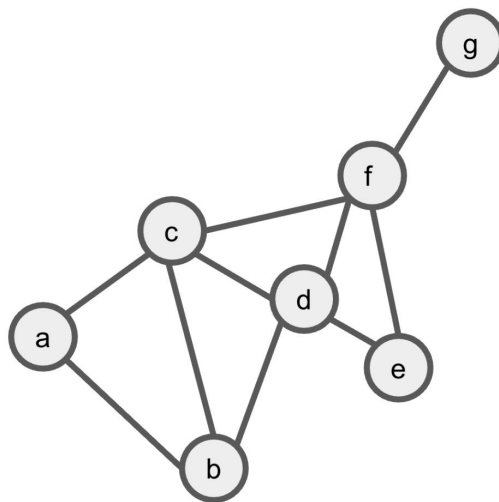
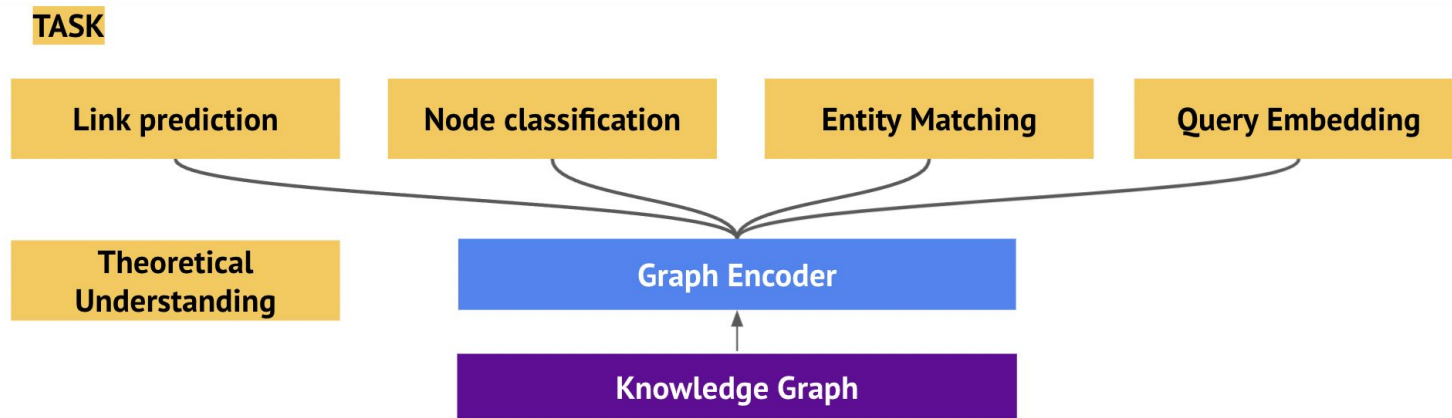
Edge index

$\begin{bmatrix} a, a, b, b, c, c, d, d, f, f, \\ b, c, c, d, d, f, f, e, e, g \end{bmatrix}$

Признаки вершин $VF \in \mathbb{R}^{|V| \times K}$

Признаки ребер $EF \in \mathbb{R}^{|E| \times M}$


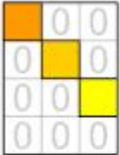
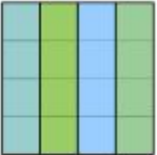

Задачи на графах



Подходы к решению: матрица смежности

1	1	0	1	0	0	0
1	1	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	0	0	0
0	1	1	0	1	1	1
0	0	1	0	1	1	0
0	0	0	0	1	0	1

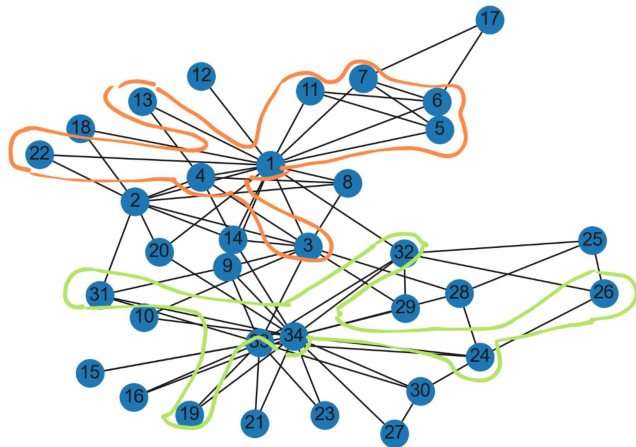
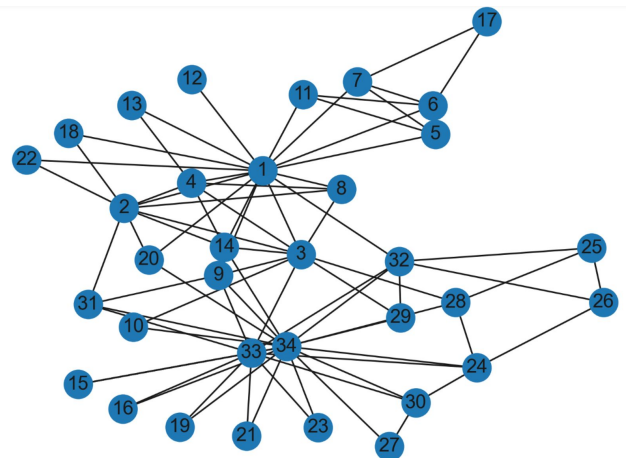
Подходы к решению: SVD



$$\begin{matrix} \mathbf{M} \\ m \times n \end{matrix} = \begin{matrix} \mathbf{U} \\ m \times m \end{matrix} \begin{matrix} \mathbf{\Sigma} \\ m \times n \end{matrix} \begin{matrix} \mathbf{V}^* \\ n \times n \end{matrix}$$

1	1	0	1	0	0	0
1	1	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	0	0	0
0	1	1	0	1	1	1
0	0	1	0	1	1	0
0	0	0	0	1	0	1

Подходы к решению: DeepWalk



Stefano is explaining deep walk on a Medium page

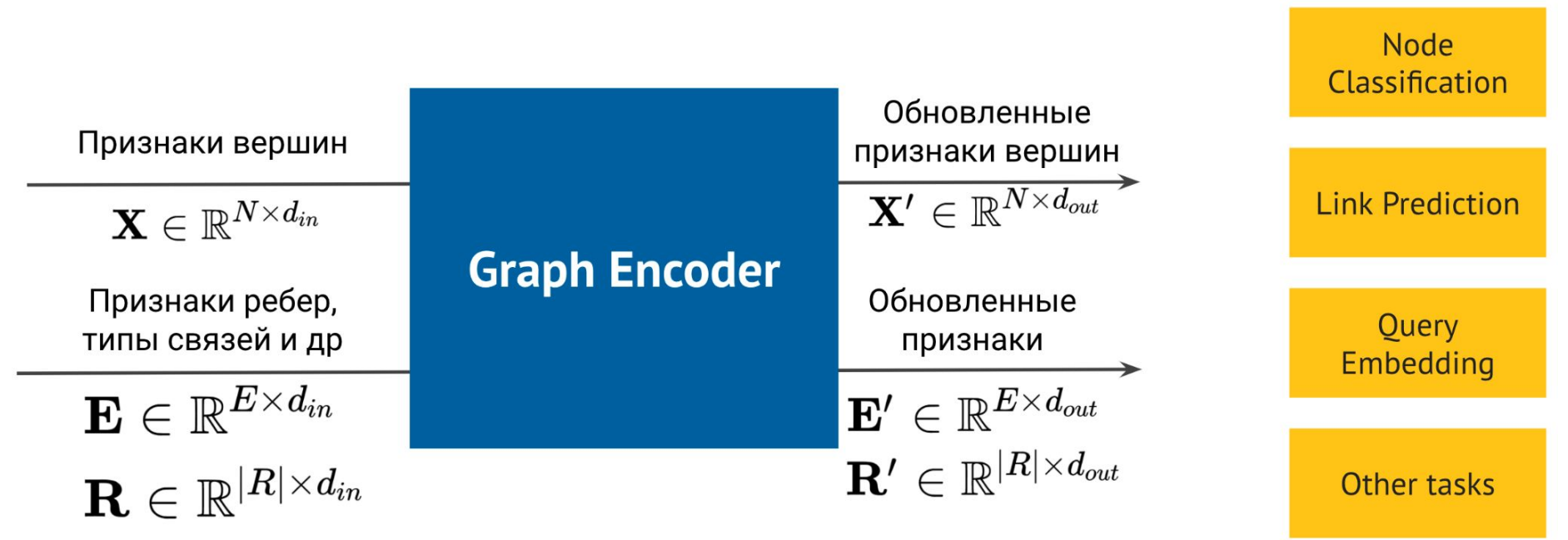
SkipGram Input

walk

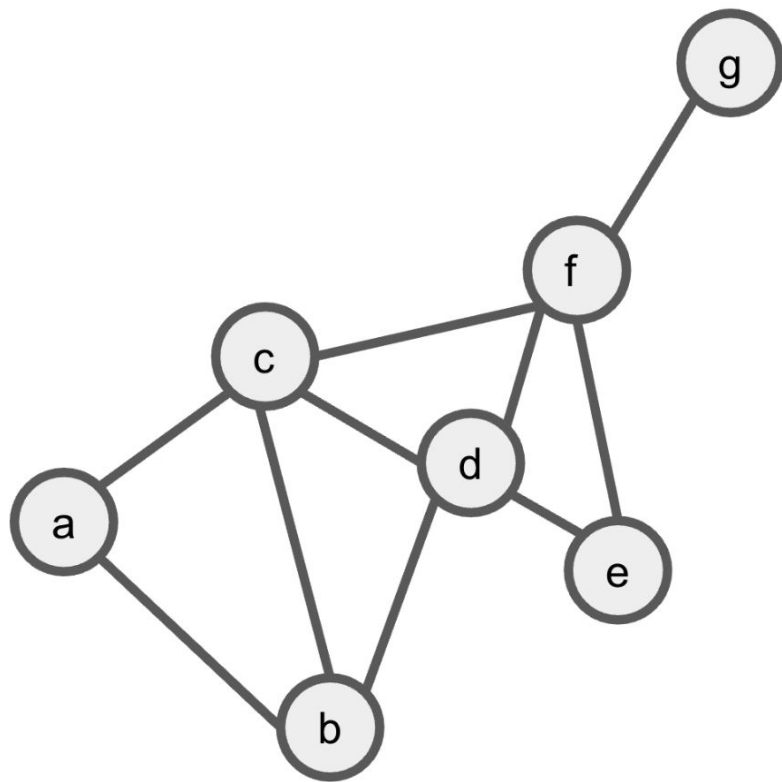
Expected SkipGram Output

explaining
deep
Medium
page

Обучаемые подходы решения задач на графах



Message Passing



Что у нас есть:

- **A** - матрица смежности (adjacency matrix)

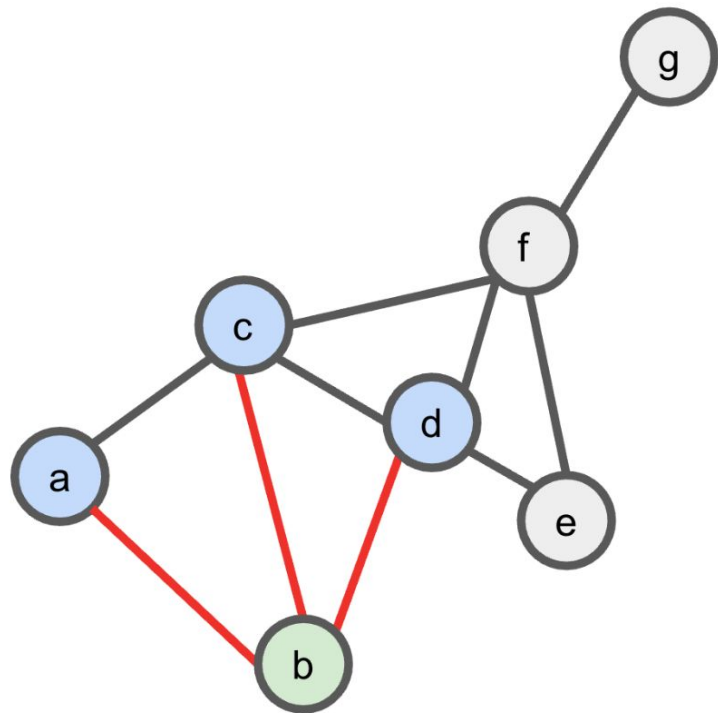
$$\mathbf{A} : N \times N$$

Для экономии памяти:

- Разреженная (sparse) A
- Лист граней (edge index)

```
[[ a, a, b, b, c, c, d, d, f, f],  
 [ b, c, c, d, d, f, f, e, e, g]]
```

Message Passing: Aggregate & Update



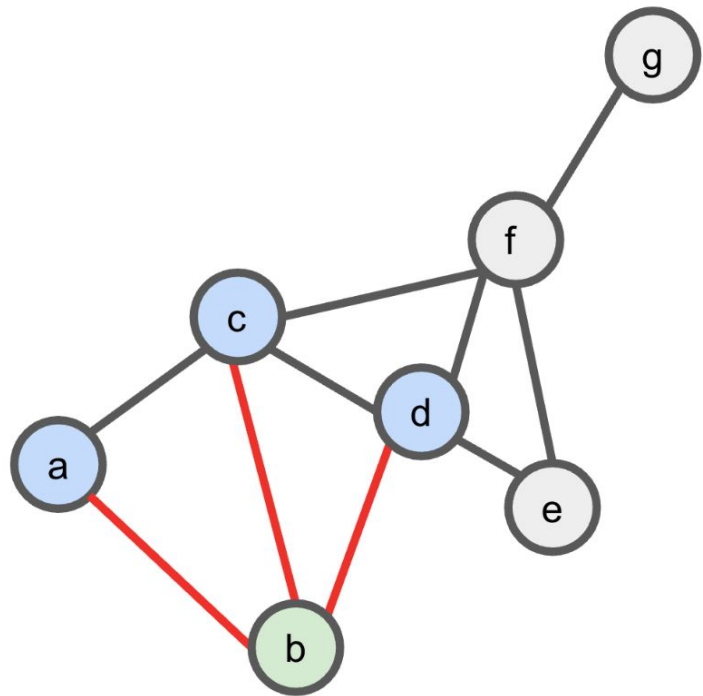
Новое представление узла b получается как **функция** от предыдущего представления $\mathbf{x}(b)$ и представления соседей $\mathbf{X}(N(b))$

$$\mathbf{h}_b = \phi(\mathbf{x}_b, \mathbf{X}_{\mathcal{N}_b})$$

Представление соседей получается путем **агрегации** их представлений

$$\mathbf{X}_{\mathcal{N}_b} = \psi(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d,)$$

Message Passing: Aggregate & Update



1. AGGREGATE

Построение сообщения **m**

$$\mathbf{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})$$

2. UPDATE

Обновление узла **u**

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})$$

Message Passing: **Aggregate**

На первом шаге для вершины u строится сообщение :

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}(\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\})$$

В графах нет простого понятия "местоположения" вершины, то есть мы не можем сказать, что вершина u находится "справа" или "сверху" от вершины v . У каждой вершины есть сообщество соседей, которое мы можем в общем случае перечислять в любом порядке. Поэтому функция агрегации должна быть инвариантна к перестановкам (**permutation invariance**) - то есть результат агрегации не зависит от порядка ее применения к вершинам-соседям.

Мы будем записывать permutation invariant функции как \oplus :

$$\mathbf{m}_{\mathcal{N}(u)} = \bigoplus_{v \in \mathcal{N}(u)} \psi(\mathbf{x}_v)$$

Например, часто используемое агрегирование через сумму представлений соседей и умножение с обучаемой весовой матрицей $\mathbf{W}_{\text{neigh}}$ будет записываться как:

$$\mathbf{m}_{\mathcal{N}(u)} = \mathbf{W}_{\text{neigh}} \sum_{v \in \mathcal{N}(u)} \mathbf{x}_v$$

Message Passing: **Update**

Новое представление вершины **u** на **k**-ом слое получается в результате функции UPDATE от предыдущего представления этой вершины **h_u** и сообщения **m**, полученного на шаге AGGREGATE:

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}(\mathbf{h}_u^k, \mathbf{m}_{\mathcal{N}(u)}^k)$$

Или с использованием нотации агрегирования:

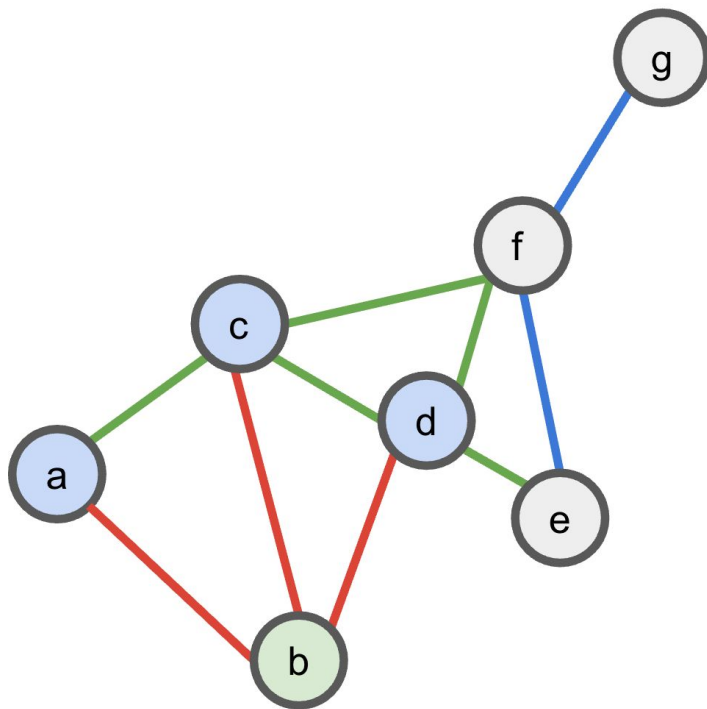
$$\mathbf{h}_u^{(k+1)} = \phi(\mathbf{h}_u^k, \bigoplus_{v \in \mathcal{N}(u)} \mathbf{h}_v^k)$$

В простейшем виде функция UPDATE может складывать преобразованные представления и пропускать результат через некоторую нелинейную функцию **σ** (sigmoid, tanh, ReLU, и т.д.):

$$\mathbf{h}_u^{(k+1)} = \sigma(\mathbf{W}_{\text{self}} \mathbf{h}_u^k + \mathbf{W}_{\text{neigh}} \mathbf{m}_{\mathcal{N}(u)}^k)$$

Глубина Message Passing сетей

Message Passing эквивалентен агрегации соседних представлений, поэтому для получения сообщений от k -hop соседей (k -hop neighborhood) нужно как минимум k слоев message passing.



1-hop (1 слой)

2-hop (2 слой)

3-hop (3 слой)

Матричная запись Message Passing сетей

Из линейной алгебры можно вспомнить, что произведение AX разреженной матрицы смежности A и матрицы признаков X содержит для каждой вершины сумму представлений ее соседей, что является частным случаем message passing с инвариантным агрегатором суммирования.

Для того, чтобы включить в результирующее произведение представление самой вершины, вводят простую аугментацию - self-loops, то есть виртуальные ребра-петли, соединяющие каждую вершину саму с собой. В матричной записи петли (self-loops) описываются как identity matrix I с единицами на главной диагонали. Тогда получается аугментированная матрица смежности:

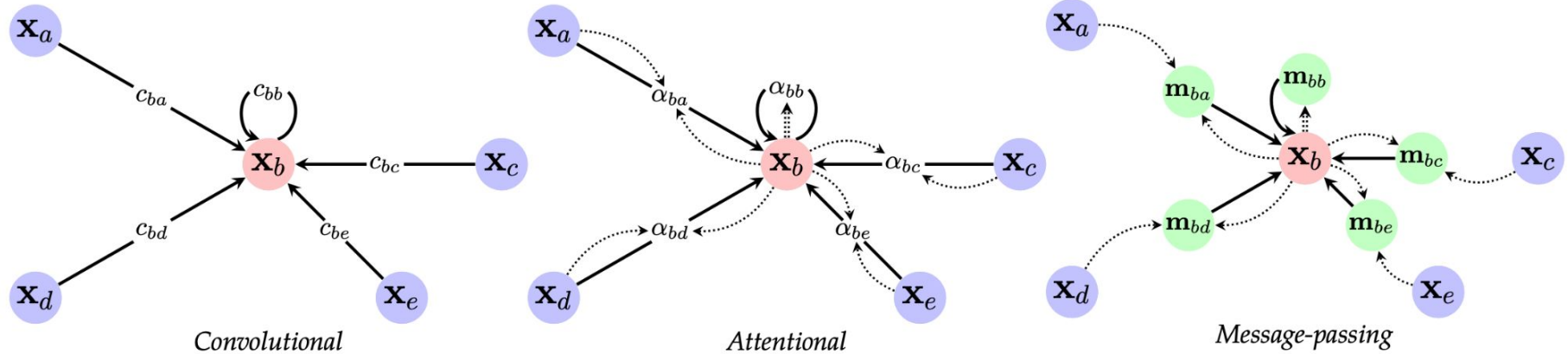
$$\tilde{A} = A + I$$

Полагая весовую матрицу агрегации соседей W_{neigh} равной весам агрегации предыдущего представления вершины W_{self} можно записать message passing на k -ом слое на уровне всего графа как уравнение:

$$\mathbf{H}^{(k+1)} = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}^{(k)})$$

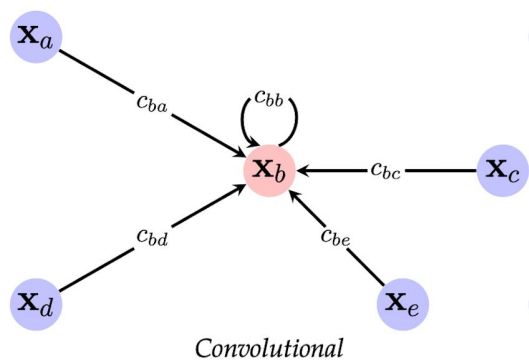
Различные GNN архитектуры усовершенствуют это уравнения дополнительной нормализацией (например, через степени вершин). Однако, не все архитектуры, например, такие как GAT, можно просто записать в матричной форме.

Message Passing Архитектуры



- Графовые конволюционные сети (Graph Convolutional Nets, GCNs), у которых веса ребер это скаляры-константы, получающиеся из степеней вершин;
- Графовые сети с вниманием (Graph Attention Nets, GATs), где веса - скаляры, но обучаемые в зависимости от представлений вершин;
- Message-Passing графовые сети (Message-Passing Neural Nets, MPNNs), где веса ребер - векторы, которые объединяются с представлениями вершин через нелинейные функции.

Graph Convolutional Nets (GCN)



В парадигме message passing способ получения представления вершины записывается как:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}(i)} c_{ij} \psi(\mathbf{x}_j)\right)$$

где c_{ij} является скалярным весом ребра, соединяющего вершины i и j .

Разные GCN модели по-разному определяют этот вес как некоторую нормализационную константу. Классическая работа [10] определяет c_{ij} как обратное от среднего геометрического степеней вершин i и j :

$$c_{ij} = \frac{1}{\sqrt{|\mathcal{N}(i)| |\mathcal{N}(j)|}}$$

Тогда UPDATE функция записывается как:

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

В матричном виде нормализация использует аугментированную диагональную матрицу степеней вершин D , полученную из аугментированной матрицы смежности $A = A + I$:

$$H = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} XW$$

Graph Attention Nets (GAT)

В парадигме message passing функцию UPDATE можно записать как:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \psi(\mathbf{x}_j)\right)$$

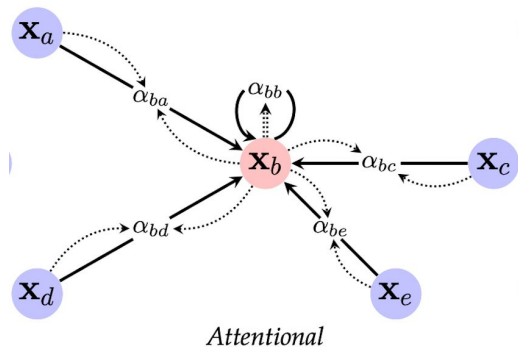
где $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ - обучаемый вес ребра. Классическая работа по GAT определяет этот весовой коэффициент через механизм внимания (attention) на основе представлений вершин и обучаемого вектора внимания \mathbf{a} (здесь \parallel – конкатенация, \mathbf{a}^T -транспонирование):

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_k]\right)\right)}$$

Простой attention коэффициент приводит к следующей формуле функции UPDATE:

$$\mathbf{h}_i = \left\| \sum_{k=1}^K \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j\right) \right\|$$

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \mathbf{h}_j\right)$$

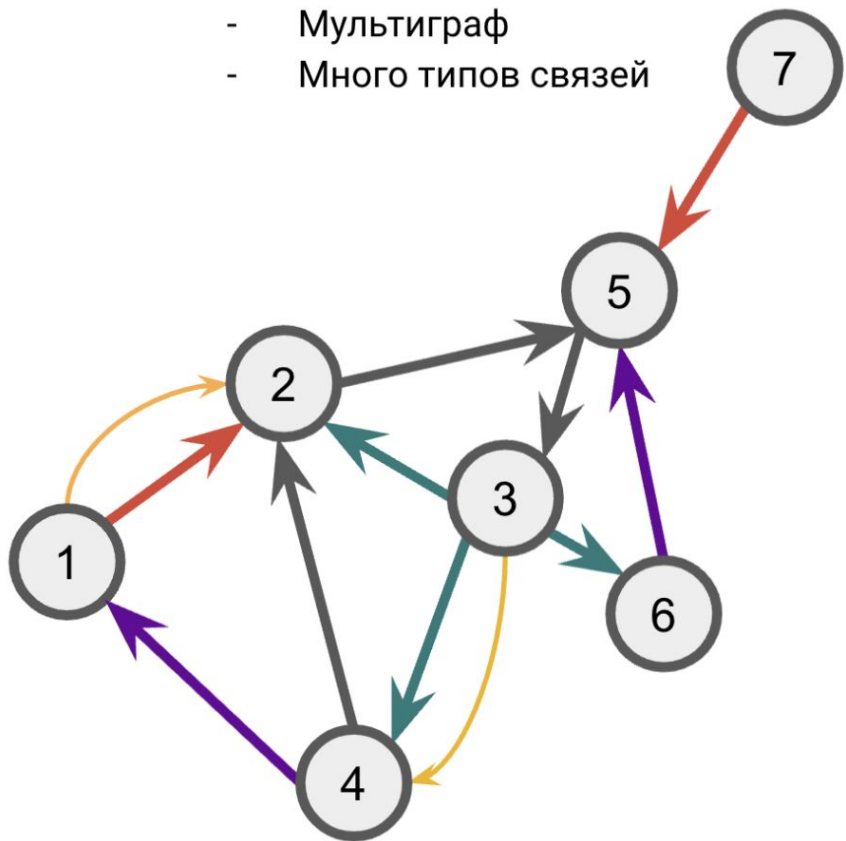


Вес ребер - по-прежнему скаляр, но обучаемый!

Можно обучать несколько коэффициентов (attention heads) на каждое ребро, тогда на последнем шаге при получении представлений нужно сконкатенерировать представления от attention heads:

Другие виды графов и подходы к их решению и другие проблемы (анонс)

- Направленный
- Мультиграф
- Много типов связей



Подходы к решению:

- Relational GCNs (R-GCN)
- Compositional GCNs (CompGCN)
- ***Ваши мысли?***

Что если граф меняется?

- Добавляются ребра
- Добавляются вершины
- Добавляются подграфы

Перейдем к семинару и посмотрит на код GCN.