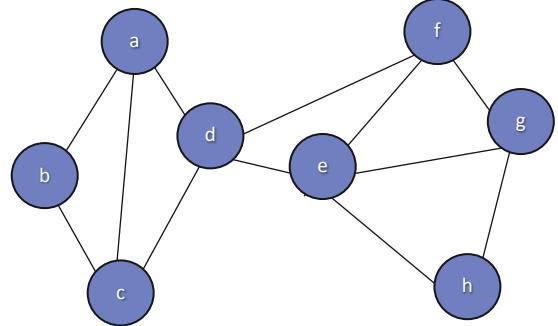


Графовые нейросети и их применение

Обозначения



$$G = (V, E), \text{ где}$$

G – структура, которая состоит из двух множеств

V – множество вершин и их признаков

E – множество ребер и их признаков

$|V|$ - количество вершин

$|E|$ - количество ребер

u, v - вершины

d_u - степень вершины u

e_{uv} - признаки ребра, соединяющие вершины u, v

Message passing. Общая формула

Формула для Message Passing:

$$h_v^{(k+1)} = \text{UPDATE} \left(h_v^{(k)}, \text{AGGREGATE} \left(\{ \text{MESSAGE} (h_u^k, e_{uv}) \mid u \in N(v) \} \right) \right),$$

где:

- $\text{MESSAGE} (h_u^k, e_{uv})$ – сообщение от узла u к узлу v , которое может включать признаки рёбра e_{uv} .
- AGGREGATE – агрегирует сообщения.
- UPDATE – обновляет представление узла

message passing описывает процесс распространения информации между узлами графа через передачу сообщений, агрегацию и обновление.

Этот метод используется на каждом слое GNN для создания локальных и глобальных представлений узлов.

Message passing. Общая формула

Каждая вершина получает сообщения от своих соседей:

$$m_v^{(t+1)} = \sum_{u \in \mathcal{N}(v)} M_t \left(h_v^{(t)}, h_u^{(t)}, e_{uv} \right)$$

где:

- $h_v^{(t)}$ — вектор признаков вершины v на шаге t ,
- e_{uv} — признаки ребра между u и v ,
- $M_t(\cdot)$ — обучаемая **message function** (например, MLP или attention).

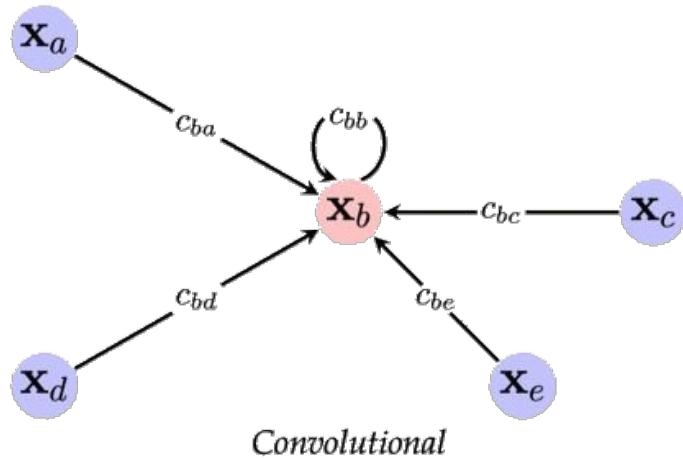
После получения сообщений вершина обновляет свой вектор состояния:

$$h_v^{(t+1)} = U_t \left(h_v^{(t)}, m_v^{(t+1)} \right)$$

где:

- $U_t(\cdot)$ — обучаемая **update function** (например, GRU или линейный слой),
- она решает, как использовать полученное сообщение для изменения внутреннего состояния узла.

Graph Convolutional Networks (GCN)



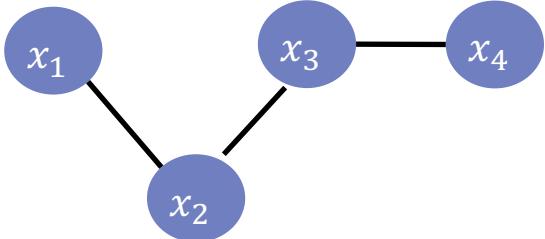
Архитектура multi-layer Graph Convolutional Network (GCN) впервые была представлена в 2017 году (<https://arxiv.org/pdf/1609.02907.pdf>), авторы рассматривали данную архитектуру для задачи Node Classification.

$$(1) \quad h_v^{(k+1)} = f(h_v^{(k)}, \text{AGGREGATE}(\{h_u^{(k)} \mid u \in N(v)\}))$$

Источник: <https://geometricdeeplearning.com>

Message passing to GCN

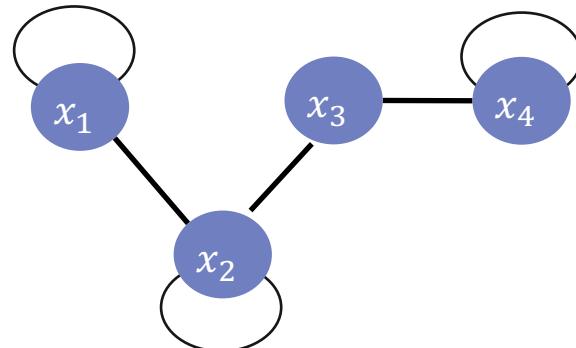
Матричное представление



$$Y_{n,1} = A_{n,n} \cdot X_{n,m} \cdot W_{m,1}$$



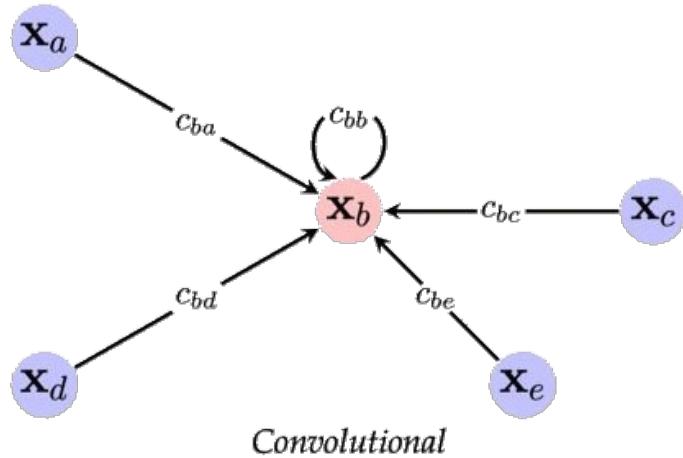
содержит для каждой вершины
сумму представлений соседей
(частный случай, где
агрегатор - суммирование)



$$\begin{aligned}\tilde{A} &= A + I \\ \hat{A} &= \hat{D}^{-1/2} \tilde{A} \hat{D}^{-1/2}\end{aligned}$$

Где \hat{D} - диагональная матрица
 A – матрица смежности
 I – единичная матрица

Graph Convolutional Networks (GCN)



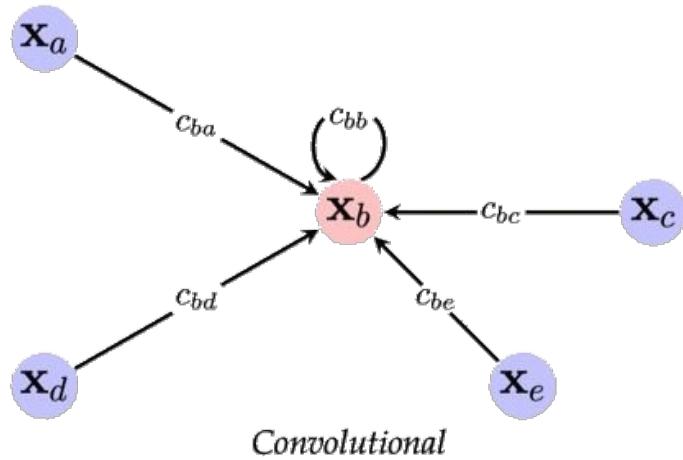
$$(1) \quad h_v^{(k+1)} = f(h_v^{(k)}, \text{AGGREGATE}(\{h_u^{(k)} \mid u \in N(v)\}))$$

$$(2) \quad H^{(k+1)} = \sigma(\hat{A}H^{(k)}W^{(k)}),$$

Где $H^{(k)}$ – матрица признаков узлов на слое k
 $W^{(k)}$ – обучаемая матрица весов
 σ – нелинейная функция активации
 $\hat{A} = \hat{D}^{-1/2} \tilde{A} \hat{D}^{-1/2}$

Источник: <https://geometricdeeplearning.com>

Graph Convolutional Networks (GCN)



Источник: <https://geometricdeeplearning.com>

$$(1) \quad h_v^{(k+1)} = f(h_v^{(k)}, \text{AGGREGATE}(\{h_u^{(k)} \mid u \in N(v)\}))$$

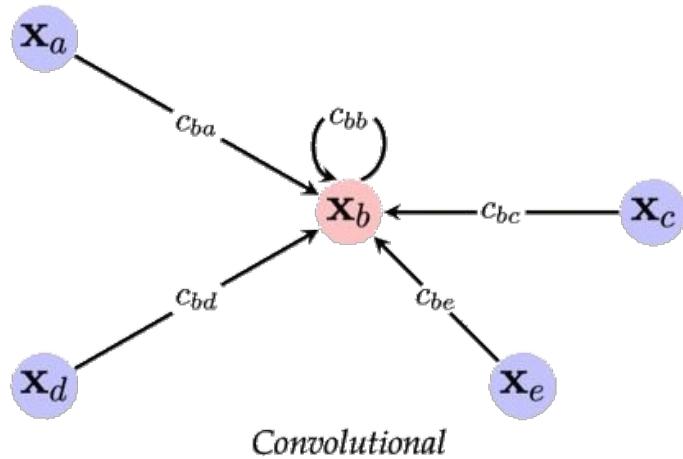
$$(2) \quad H^{(k+1)} = \sigma(\hat{A}H^{(k)}W^{(k)}),$$

Общая формула обновления признаков узлов на $k+1$ слое выглядит следующим образом:

$$(3) \quad H^{(k+1)} = \sigma(\hat{D}^{-1/2}(A + I)\hat{D}^{-1/2}H^{(k)}W^{(k)}),$$

Где $H^{(k)}$ - матрица признаков узлов на слое k ,
 $W^{(k)}$ – обучаемая матрица весов,
 σ – нелинейная функция активации
 \hat{D} - диагональная матрица
 A – матрица смежности
 I – единичная матрица

Graph Convolutional Networks (GCN)



Источник: <https://geometricdeeplearning.com>

- Message Function:

$$M_t(h_v, h_u, e_{uv}) = \frac{1}{\sqrt{d_v d_u}} W_t h_u$$

- Update Function:

$$U_t(h_v, m_v) = \sigma(m_v)$$

$$h_v^{(t+1)} = \sigma \left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_v d_u}} W_t h_u \right)$$

Это и есть классическая операция **свертки на графике** (аналог свертки CNN, но с нормировкой по соседям).

Graph Convolutional Networks (GCN)

Преимущества	Ограничения
Эффективно агрегирует локальную информацию соседей	Плохо масштабируется на большие графы (из-за использования всей матрицы смежности)
Лёгкая реализация благодаря линейным слоям и матрице смежности	Ограниченнная глубина (перфоманс может снижаться при увеличении числа слоёв из-за затухания сигналов)
Хорошо подходит для задач классификации узлов и предсказания связей	Требует предварительного вычисления нормализованной матрицы смежности

Over-smoothing

На каждом слое GCN выполняется агрегация соседей:

$$H^{(k+1)} = \sigma(\hat{A}H^{(k)}W^{(k)}),$$

где \hat{A} — нормализованная матрица смежности.

GCN использует нормализованную матрицу смежности \hat{A} , которая преобразует признаки на каждом слое. При увеличении числа слоёв K можно показать, что признаки сходятся к состоянию равновесия:

- С каждым слоем представления узлов усредняются, что приводит к размытию уникальных признаков узлов.
- После достаточного количества слоёв все узлы в одном связанным компоненте графа начинают иметь почти одинаковые представления.

Over-smoothing

В пределе — все узлы становятся неразличимыми.

$$\lim_{k \rightarrow \infty} (\hat{A})^k X = \pi X,$$

где π — стационарное распределение случайного блуждания на графе.

$$H^{(k+1)} = \sigma(\hat{A}H^{(k)}W^{(k)}),$$



$$H^{(K)} = \sigma(\hat{A}^K H^{(0)})$$

Over-smoothing – способы решить проблему

Ограничение числа слоёв	На практике GCN работают лучше с 1–3 слоями, так как это позволяет агрегировать локальную информацию без значительного затухания сигналов.
Residual Connections (остаточные связи)	Добавление остаточных связей (ResNet) помогает сохранить начальные признаки узлов на более глубоких слоях: $H^{(k+1)} = H^{(k)} + \sigma(\hat{A}H^{(k)}W^{(k)})$
DropEdge	Случайное удаление рёбер во время обучения уменьшает проблему затухания сигналов
BatchNorm/LayerNorm/PairNorm	Нормализация скрытых представлений на каждом слое помогает стабилизировать обучение
Другие архитектуры	Архитектуры, такие как Graph Attention Networks (GAT) и GraphSAGE, уменьшают затухание сигналов за счёт адаптивного внимания и индуктивных методов агрегации

Нормализация

- **BatchNorm / LayerNorm** — стабилизируют обучение, но не устраняют причину smoothing.
- **PairNorm** — нормализует попарные расстояния между узлами, сохраняя различимость:

$$H' = s \cdot \frac{H - \bar{H}}{\sqrt{\frac{1}{n} \sum_i \|H_i - \bar{H}\|_2^2}}.$$

- **NodeNorm / GraphNorm** — пер-узловая/пер-графовая центровка- масштабирование; полезно в батчинге.

Практика: комбинируют BatchNorm/LayerNorm (для оптимизации) с PairNorm/GraphNorm (для геометрии).

Residual Connections (остаточные связи)

Формула (pre-act вариант, стабильнее на практике)

$$\begin{aligned}\tilde{H}^{(l)} &= \text{Norm}(H^{(l)}) \\ Z^{(l)} &= \sigma(\hat{A} \tilde{H}^{(l)} W^{(l)}) \\ H^{(l+1)} &= H^{(l)} + Z^{(l)} \quad (\text{residual})\end{aligned}$$

- **Зачем:** прямой путь градиента, меньше «залипания» в усреднение.
- **Замечание:** «pre-activation» (норма/активация до свёртки) эмпирически облегчает оптимизацию (аналогично ResNet v2).

Residual Connections (остаточные связи)

Initial residual (возврат к исходным признакам)

В каждом слое подмешиваем $H^{(0)}$ (исходные фичи), чтобы **не терять первичную информацию**:

$$H^{(l+1)} = (1 - \alpha) \Phi^{(l)}(H^{(l)}) + \alpha H^{(0)}, \quad 0 < \alpha < 1,$$

где $\Phi^{(l)}$ — граф-свёртка + нелинейность.

Это идея APPNP/GCNII: персонализированное распространение (PageRank-подобный «телеport» к исходным фичам).

Jumping Knowledge (JK)

Комбинируем представления разных глубин:

$$H^{\text{JK}} = \text{concat}(H^{(1)}, H^{(2)}, \dots, H^{(L)}) \quad \text{или} \quad H^{\text{JK}} = \max_l H^{(l)}.$$

- **Зачем:** избегаем ситуации, когда «лучшие» признаки были на средних слоях.

DropEdge

На каждом обучающем шаге выбирается случайная подвыборка рёбер:

$$E' = \{(u, v) \in E : \xi_{uv} \sim \text{Bernoulli}(1 - p)\}$$

где p — вероятность удаления ребра (например, 0.2 или 0.3).

Далее GCN, GAT или любая другая GNN обучается **на усечённом графе** $G' = (V, E')$.

На следующем батче набор рёбер случайно обновляется — то есть сеть видит “чуть другой” граф.

Диффузионные архитектуры

APPNP (персонализированное распространение)

Итеративно распространяем предсказания, каждый раз возвращаясь к «источнику»:

$$H^{(k+1)} = (1 - \alpha) \hat{A} H^{(k)} + \alpha H^{(0)}.$$

— устойчиво против over-smoothing, можно делать десятки итераций распространения.

GCNII (DeepGCN-семейство)

Соединяет Initial Residual + Identity Mapping:

$$H^{(l+1)} = \sigma \left(\left((1 - \alpha_l) \hat{A} H^{(l)} + \alpha_l H^{(0)} \right) \left((1 - \beta_l) I + \beta_l W^{(l)} \right) \right),$$

где α_l, β_l зависят от глубины (например, уменьшаются).

— Позволяет строить **очень глубокие** GCN (20–100+ слоёв).

ResGCN + PairNorm + DropEdge

- **Кейс:** когда хотим глубину 10–40 слоёв на статичных средних графах.

Слой:

pre-norm → GCNConv → ReLU → residual add.

Блоки: каждые N слоёв — PairNorm; на обучении — DropEdge 10–30%.

- **Pre-norm** — нормализация до свёртки (LayerNorm / BatchNorm), стабилизирует обучение и улучшает обратное распространение градиента.
- **GCNConv** — стандартная операция свёртки по соседям.
- **ReLU** — активация для нелинейности.
- **Residual add** — добавляем исходные признаки слоя (skip connection).

MixHop – редизайн агрегации

- **MixHop**: одновременно агрегирует соседей на разных расстояниях $k = 1, 2, 3$
 $[\hat{A}X \parallel \hat{A}^2X \parallel \hat{A}^3X] \rightarrow \text{MLP}.$

$$H^{(l+1)} = \text{Concat}(\sigma(\hat{A}XW_1), \sigma(\hat{A}^2XW_2), \sigma(\hat{A}^3XW_3))$$

где:

- \hat{A} – нормализованная матрица смежности (1-hop связи),
- \hat{A}^2 – соседи на расстоянии 2,
- \hat{A}^3 – соседи на расстоянии 3,
- W_1, W_2, W_3 – обучаемые веса для каждой “глубины связи”.
- **Decoupled GCN**: разделяет «пропагатор» (\hat{A}) и «MLP-обработку», уменьшая «смешивание» градиента и диффузии.

Decoupled GCN – редизайн агрегации

- Decoupled GCN: разделяет «пропагатор» (\hat{A}) и «MLP-обработку», уменьшая «смешивание» градиента и диффузии.

В обычной GCN операция

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

одновременно выполняет два процесса:

1. **Диффузию (propagation):** распространение сигналов по графу через \hat{A} ,
2. **Обучение признаков:** линейное преобразование через $W^{(l)}$.

Decoupled GCN – редизайн агрегации

В обычной GCN операция

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

Разделить эти два процесса:

- сначала сделать всю диффузию (много шагов \hat{A}^k),
- потом отдельно обучить MLP на результирующих признаках.

$$H = \hat{A}^K X, \quad Y = \text{MLP}(H)$$

или более общий вариант:

$$H = \sum_{k=0}^K \theta_k \hat{A}^k X, \quad Y = \text{MLP}(H)$$

Decoupled GCN — Архитектуры

- **SGC**: предварительно считаем $\hat{A}^K X$ и обучаем один линейный классификатор — быстрый сильный baseline.

Она буквально берёт GCN, убирает все нелинейности и объединяет веса:

$$H = \hat{A}^K X W$$

- $\hat{A}^K X$ — многократная диффузия признаков,
- W — один линейный классификатор (MLP из одного слоя).

Decoupled GCN — Архитектуры

- **SIGN**: храним несколько степеней $\hat{A}^k X$ как каналы, обучаем MLP над конкатенацией.

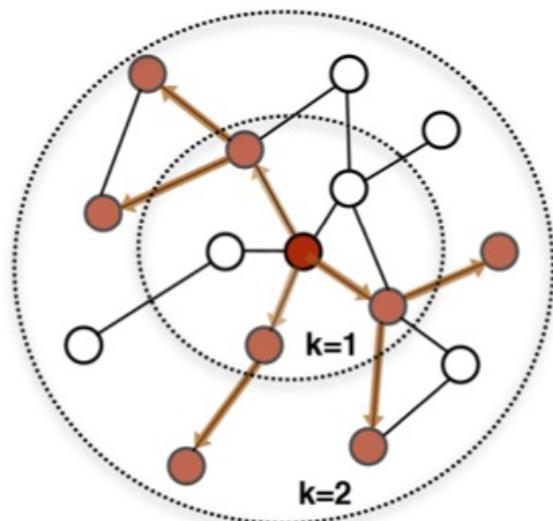
SIGN развивает ту же идею, но делает её **более гибкой и масштабируемой**:

$$H = \text{Concat}(\hat{A}^0 X, \hat{A}^1 X, \hat{A}^2 X, \dots, \hat{A}^K X)$$

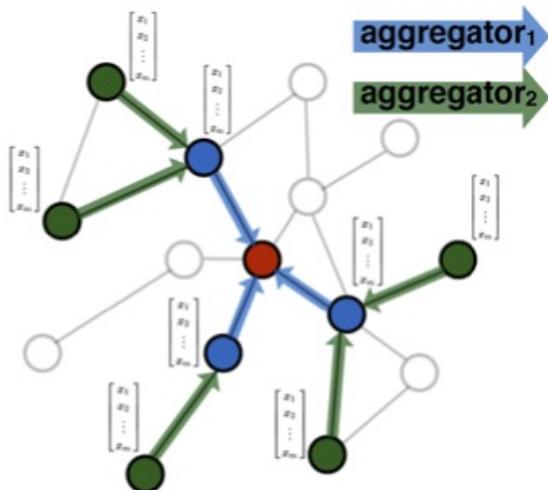
$$Y = \text{MLP}(H)$$

- SIGN не просто берёт одно $\hat{A}^K X$,
а **сохраняет несколько степеней диффузии** (multi-hop признаки).
- Затем объединяет их и подаёт в MLP.

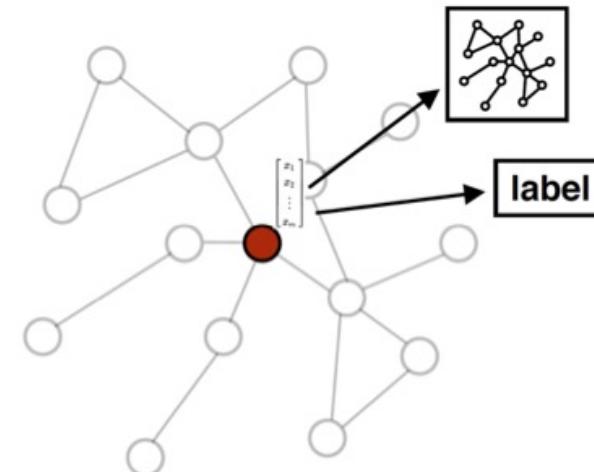
GraphSAGE



1. Sample neighborhood



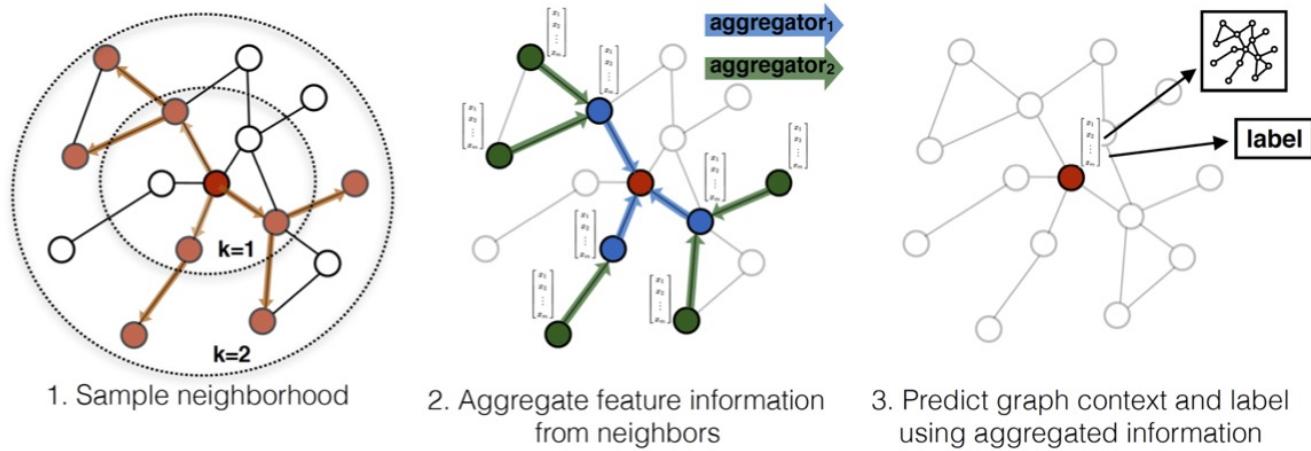
2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Источник. <https://arxiv.org/pdf/1706.02216>

GraphSAGE

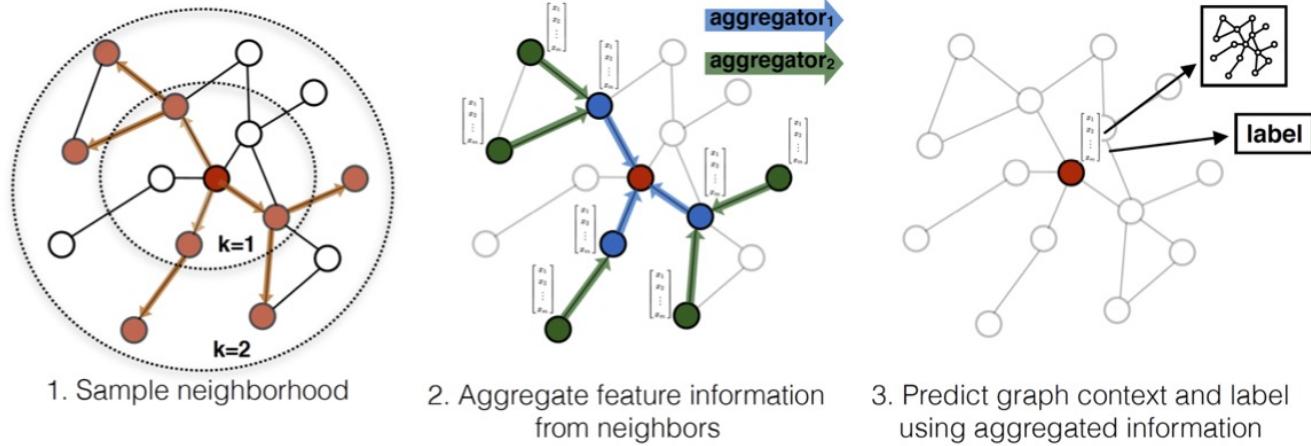


Источник. <https://arxiv.org/pdf/1706.02216>

Основные особенности GraphSAGE

- 1) Индуктивное обучение
- 2) Агрегация соседей
- 3) Масштабируемость

GraphSAGE

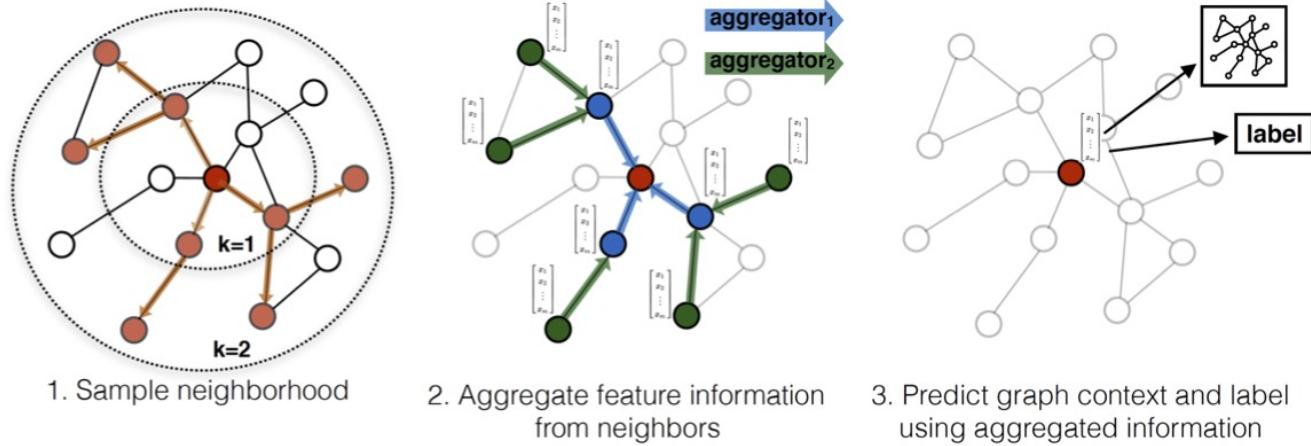


Источник. <https://arxiv.org/pdf/1706.02216>

Для узла v на слое k обновление его представления состоит из трёх этапов:

- 1) Выборка соседей $N(v)$:
- 2) Агрегация признаков соседей
- 3) Обновление признаков узла

GraphSAGE



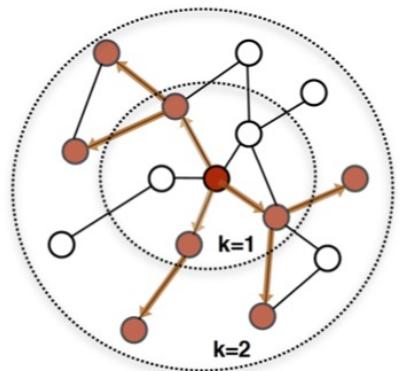
Источник. <https://arxiv.org/pdf/1706.02216>

Для узла v на слое k обновление его представления состоит из трёх этапов:

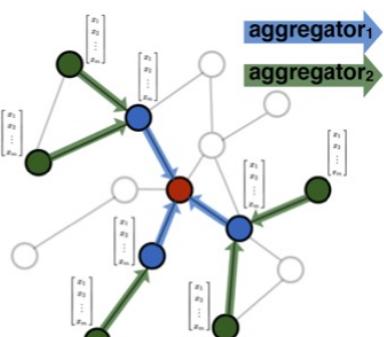
1) Формирование выборки соседей $N(v)$:

Выбирается фиксированное подмножество соседей узла v (например, случайная выборка l соседей). Это позволяет избежать обработки всех соседей одновременно, что особенно полезно для графов с высокой степенью узлов.

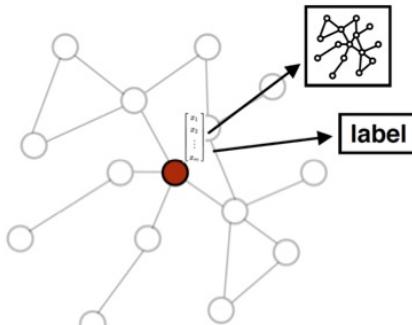
GraphSAGE



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Источник. <https://arxiv.org/pdf/1706.02216>

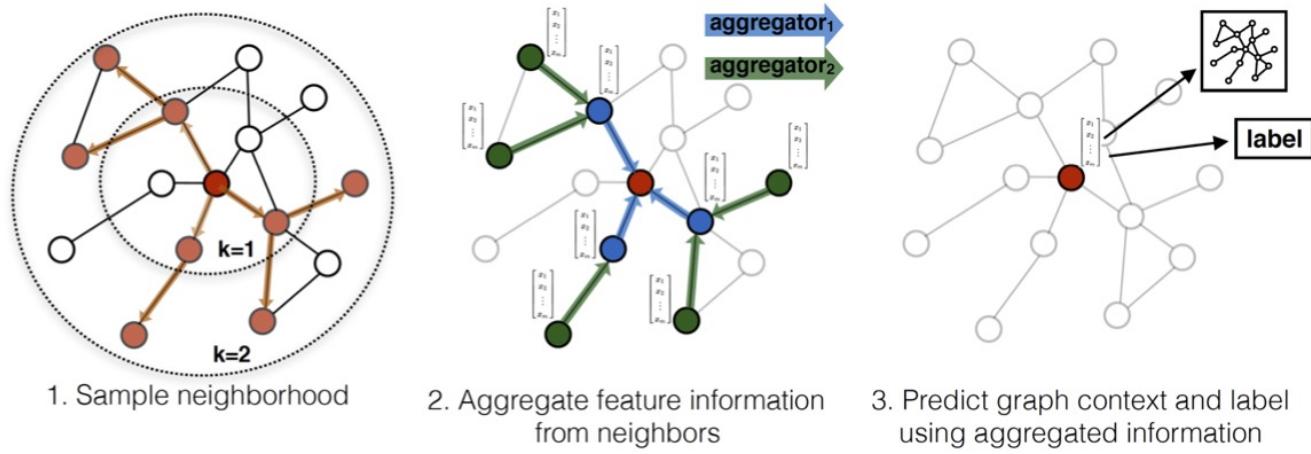
2) Агрегация признаков соседей

Признаки выбранных соседей агрегируются с использованием обучаемой функции *AGGREGATE* .

Примеры функций:

- Mean Aggregator: Усреднение признаков соседей.
- Max Pooling Aggregator: Максимизация преобразованных признаков соседей.
- LSTM Aggregator: Использование рекуррентной нейронной сети (LSTM) для агрегации признаков соседей.

GraphSAGE



Источник. <https://arxiv.org/pdf/1706.02216>

3) Обновление признаков узла

Агрегированные признаки соседей объединяются с признаками самого узла (например, через конкатенацию или сложение). Признаки обновляются с использованием обучаемой линейной трансформации и нелинейной активации.

GraphSAGE

Обновление признаков узла в формулах

На слое k для узла v :

1. Выборка соседей:

$$N(v) = \text{Sample}(N(v), l),$$

где l — фиксированное число соседей, выбираемых для узла v .

2. Агрегация признаков соседей:

$$h_{N(v)}^{(k)} = \text{AGGREGATE} \left(h_u^{(k)}, \forall u \in N(v) \right),$$

где $h_u^{(k)}$ — признаки соседа u на слое k ,

AGGREGATE — обучаемая или фиксированная функция агрегации.

GraphSAGE

GraphSAGE поддерживает несколько стратегий агрегации:

1. Mean Aggregator (усреднение):

а. Простая и эффективная стратегия.

$$h_{N(v)}^{(k)} = \frac{1}{N(v)} \sum_{u \in N(v)} h_u^{(k)}$$

2. Max Pooling Aggregator:

а. Использует нейросеть для обработки каждого соседа перед применением операции максимума.

$$h_{N(v)}^{(k)} = \max_{u \in N(v)} \text{ReLU}(W_{pool} h_u^{(k)} + b_{pool})$$

Где W_{pool} — обучаемая матрица весов для линейного преобразования признаков соседей

b_{pool} — это обучаемый вектор смещения для линейного преобразования. Размерность совпадает с количеством выходных признаков (Добавляется к линейно преобразованным признакам соседей, чтобы увеличить гибкость модели)

3. LSTM Aggregator:

Агрегация соседей через рекуррентную сеть (LSTM), которая может улавливать сложные зависимости между соседями.

GraphSAGE

3. Обновление признаков узла:

$$h_v^{(k+1)} = \sigma \left(W^k \cdot \text{CONCAT} \left(h_v^k, h_{N(v)}^k \right) \right),$$

где:

- W^k — обучаемая матрица весов,
- σ — нелинейная функция активации (например, ReLU),
- CONCAT — конкатенация или сложение.

GraphSAGE – L2-нормализация

После каждого слоя (или в конце) GraphSAGE выполняет:

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

То есть каждый эмбеддинг $h_v^{(l)}$ масштабируется так, чтобы его длина (норма) равнялась 1.

Зачем это нужно?

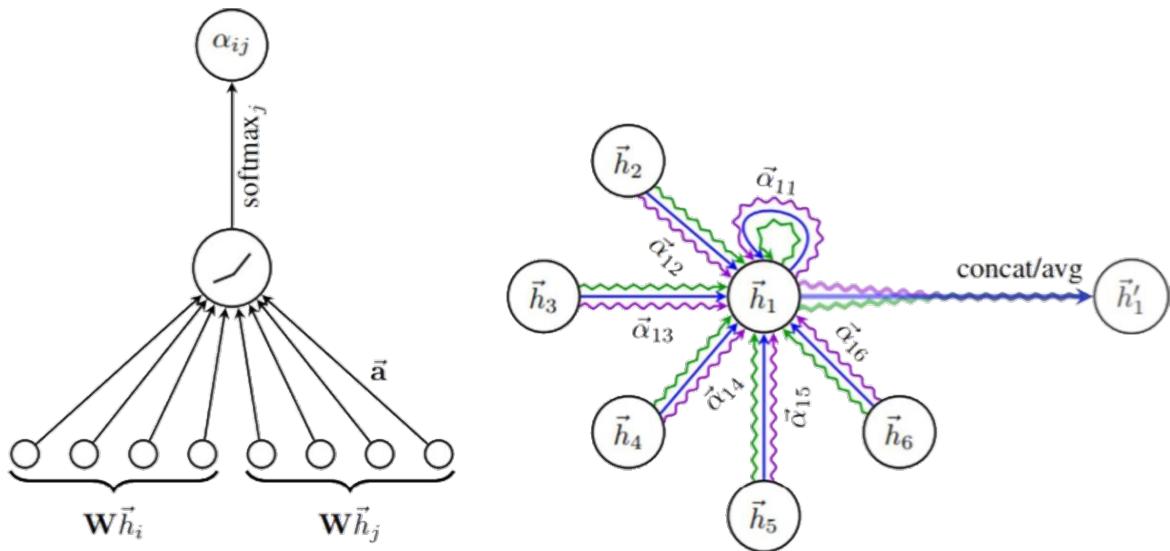
- Стабилизация обучения
- Снижение зависимости от числа соседей
- Улучшение качества в downstream-задачах
- Обеспечение консистентности при батч-обучении

GraphSAGE

Преимущества	Ограничения
Индуктивное обучение: обученная модель может обрабатывать узлы, которые не были видны во время обучения	Потеря информации: выборка фиксированного числа соседей может приводить к потере важных зависимостей в графе.
Масштабируемость: использование фиксированной выборки соседей позволяет работать с большими графами.	Зависимость от гиперпараметров: эффективность модели сильно зависит от таких параметров, как число соседей и выбор функции агрегации.
Гибкость: поддержка различных функций агрегации позволяет адаптировать модель к конкретной задаче.	

Graph Neural Networks (GAT)

Идея + историческая сводка

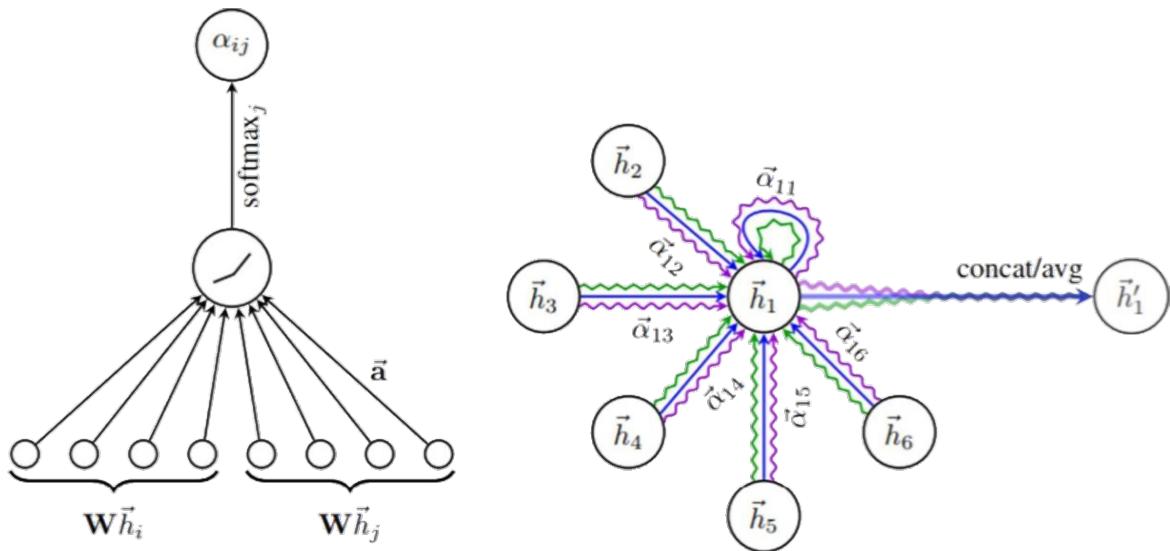


Graph Attention Networks (GAT) были представлены в 2018. Эта архитектура является улучшением Graph Convolutional Networks (GCN) и решает несколько её ограничений, включая проблему равного влияния всех соседей.

Основные идеи GAT:

1. Внимание (Attention)
2. Адаптивная агрегация
3. Локальное внимание

Graph Neural Networks (GAT)



Для каждого ребра (v, u) вычисляется "сырое" значение внимания e_{vu} , которое отражает важность узла u для обновления узла v .

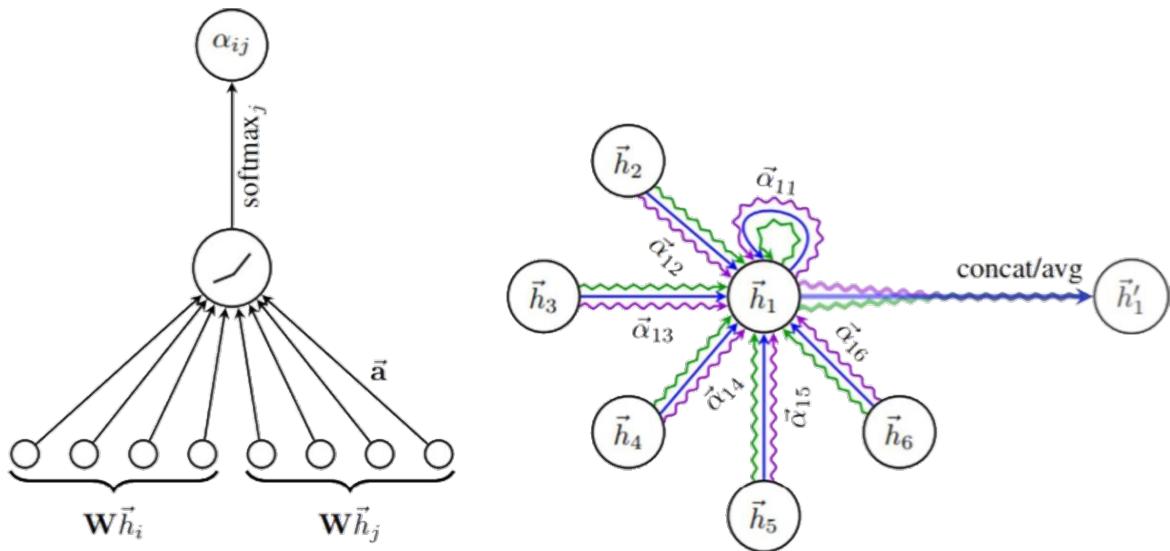
$$e_{vu} = \text{LeakyReLU}(\mathbf{a}^T [W h_v || W h_u]),$$

где:

- h_v, h_u — признаки узлов v и u ,
- W — обучаемая матрица весов, которая трансформирует признаки,
- \mathbf{a} — обучаемый вектор параметров для вычисления attention,
- $||$ — операция конкатенации,
- LeakyReLU — нелинейная функция активации.

e_{vu} измеряет, насколько "похожи" или "релевантны" h_v и h_u , и используется для определения значимости узла u для узла v .

Graph Neural Networks (GAT)



Нормализация внимания.

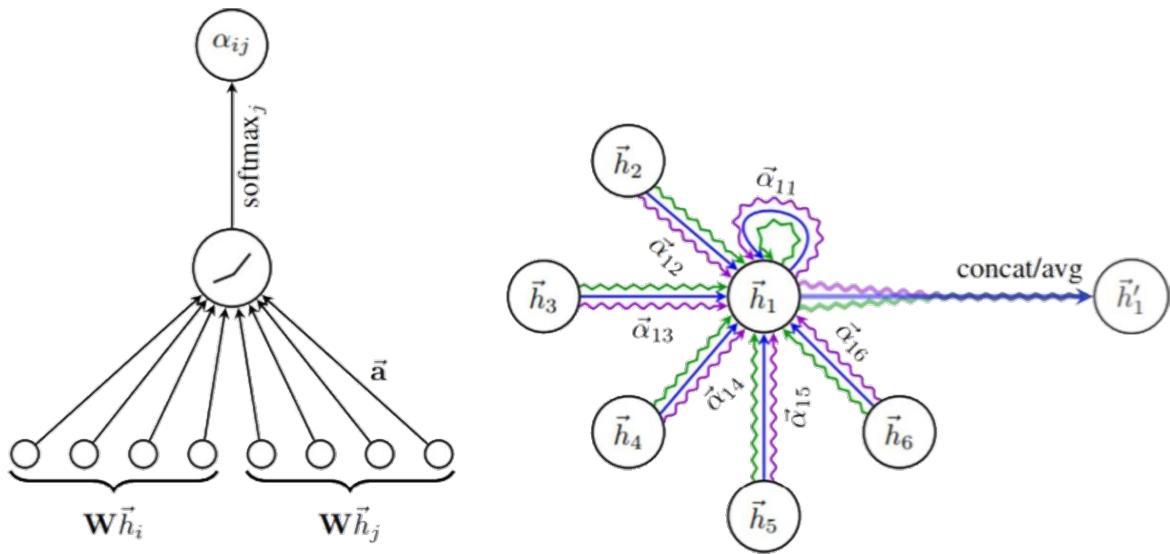
Для получения финального веса внимания α_{vu} , нормализуем e_{vu} через softmax:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{l \in N(v)} \exp(e_{vl})}$$

где:

- $N(v)$ — множество соседей узла v
- α_{vu} показывает относительный вклад узла u в обновление узла v .

Graph Neural Networks (GAT)



Агрегация соседей

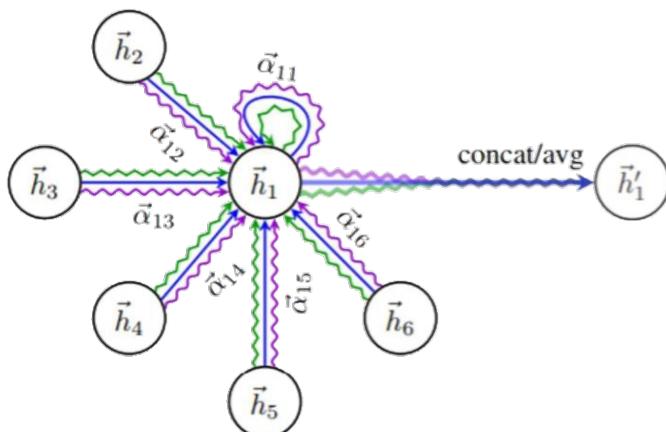
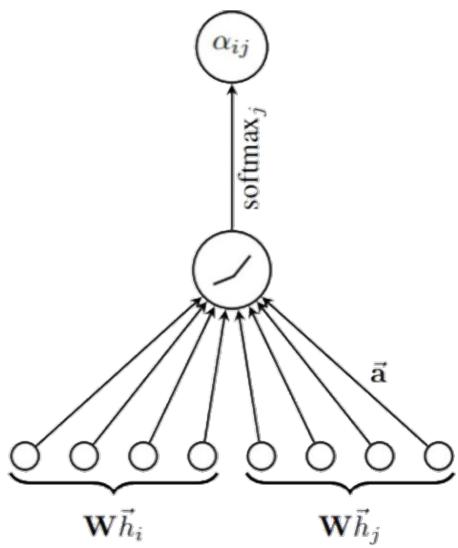
Признаки узла v обновляются через взвешенную сумму признаков его соседей:

$$h_v^{(k+1)} = \sigma \left(\sum_{u \in N(v)} \alpha_{vu} W h_u \right)$$

где:

- α_{vu} — нормализованный вес внимания,
- W — обучаемая матрица весов для преобразования признаков,
- σ — нелинейная функция активации (например, ReLU).

Graph Neural Networks (GAT2)



GATv1:

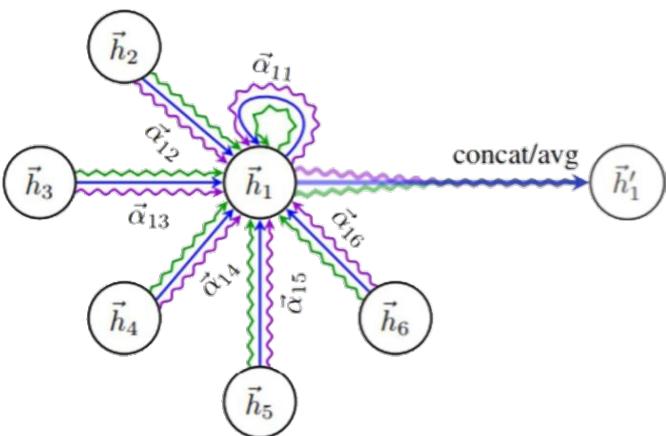
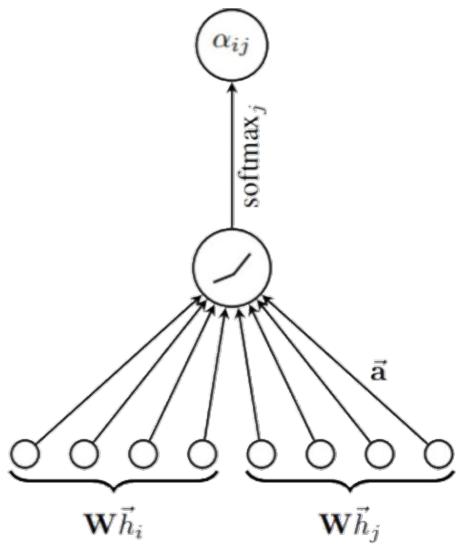
$$e_{vu} = \text{LeakyReLU}(\mathbf{a}^T [\mathbf{W} h_v || \mathbf{W} h_u]),$$

Конкатенация $[\mathbf{W} h_i || \mathbf{W} h_j]$ и скалярное произведение с a делают внимание **порядково-зависимым и несимметричным**.

Формально:

$$a^T [\mathbf{W} h_i || \mathbf{W} h_j] \neq a^T [\mathbf{W} h_j || \mathbf{W} h_i]$$

Graph Neural Networks (GAT2)



GATv1:

$$e_{vu} = \text{LeakyReLU}(\mathbf{a}^T [W h_v || W h_u]),$$

GATv2:

$$e_{vu} = \mathbf{a}^T \text{LeakyReLU}(W[h_v || h_u]),$$

Graph Neural Networks (GAT)

Multi-Head Attention

Реализация многоголового внимания:

Каждая голова вычисляет отдельные веса внимания и признаки.

1. На промежуточных слоях: конкатенация представлений от всех голов:

$$h_v^{(l+1)} = \left| \left| \sum_{u \in N(v)} \alpha_{vu}^l W^l h_u \right| \right|_1 \sigma \left(\sum_{u \in N(v)} \alpha_{vu}^l W^l h_u \right)$$

где L — число голов.

2. На последнем слое происходит усреднение представлений:

$$h_v^{(l+1)} = \frac{1}{L} \sum_{l=1}^L \sigma \left(\sum_{u \in N(v)} \alpha_{vu}^l W^l h_u \right)$$

Graph Neural Networks (GAT)

Преимущества	Ограничения
Динамическое взвешивание: каждое ребро оценивается индивидуально, что делает модель более гибкой.	Высокая вычислительная сложность: для каждого узла необходимо вычислить внимание для всех его соседей, что становится дорогим для больших графов
Локальная обработка: механизм внимания вычисляется только на уровне соседей, что снижает сложность для разреженных графов.	Чувствительность к гиперпараметрам: число голов, размер скрытых слоёв и коэффициенты регуляризации требуют тщательной настройки
Многоголовое внимание: улавливает более сложные зависимости между узлами.	Масштабируемость: GAT сложнее масштабировать на графы с миллионами узлов и рёбер.

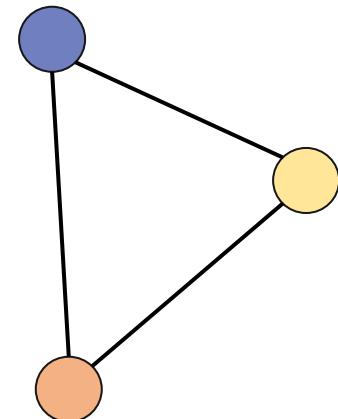
Изоморфизм графов

Пусть есть два графа:

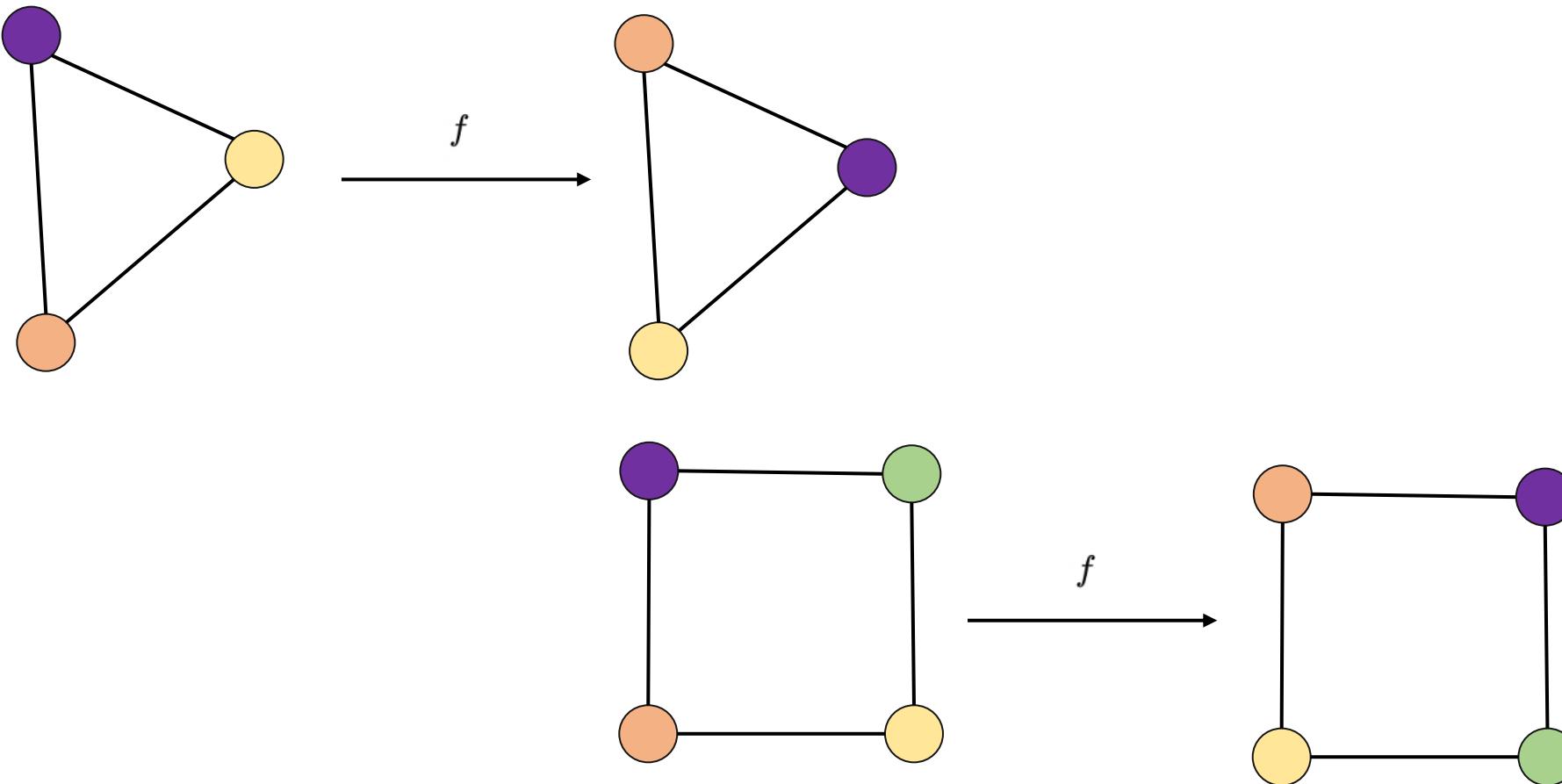
$$G_1 = (V_1, E_1), \quad G_2 = (V_2, E_2)$$

Они называются **изоморфными**,
если существует **биекция** $f : V_1 \rightarrow V_2$ такая, что
для любых $u, v \in V_1$:

$$(u, v) \in E_1 \iff (f(u), f(v)) \in E_2$$



Изоморфизм графов



Проблемы GAT и GCN

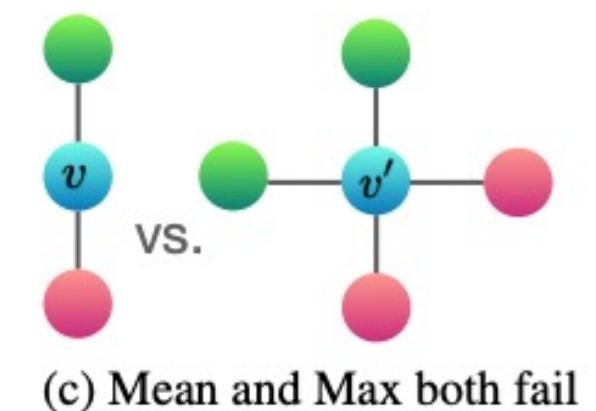
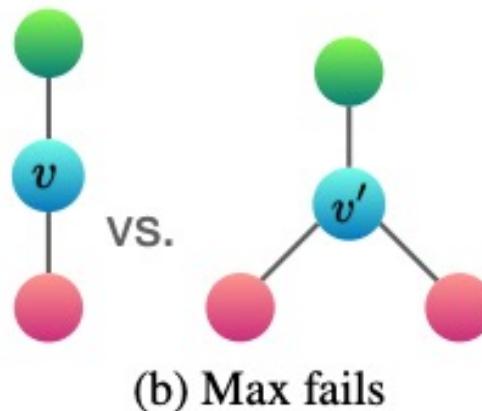
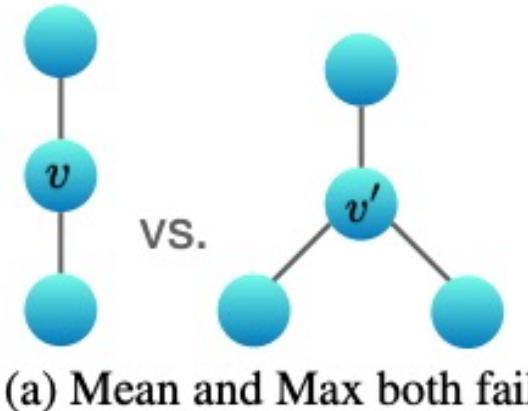
GCN и GAT агрегируют соседей через **симметричные функции** (сумма, среднее, softmax-усреднение).

Формально:

$$h'_i = f \left(h_i, \sum_{j \in N(i)} g(h_j) \right)$$

Поскольку сумма (или среднее) **инвариантны к перестановке соседей**,
эти модели **не различают мультисеты соседей**, если их элементы в сумме
совпадают.

Проблемы GAT и GCN



Источник: <https://arxiv.org/pdf/1810.00826>

Выразительность GNN

это способность модели различать **неизоморфные графы**,
то есть графы с разной структурой.

“Если два графа отличаются хотя бы локальной топологией,
но GNN выдаёт для них одинаковые эмбеддинги —
значит, она недостаточно выразительная.”

Формально:

$$f(G_1) = f(G_2) \Rightarrow G_1 \simeq G_2$$

где \simeq означает изоморфизм графов.

Тест Вайсфайлера–Лемана (1-WL)

- это классический алгоритм проверки изоморфизма графов;

Алгоритм:

1. На каждой итерации t :

- Для каждой вершины v_i собирается множество меток её соседей:

$$M_i^{(t)} = \{ c_j^{(t-1)} : j \in N(i) \}$$

- Формируется новая метка узла — комбинация его текущей метки и меток соседей:

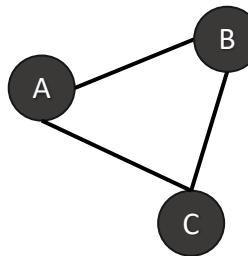
$$c_i^{(t)} = \text{Hash}(c_i^{(t-1)}, \text{Sort}(M_i^{(t)}))$$

- То есть мы “перекрашиваем” узлы, учитывая их окружение.

Тест Вайсфайлера–Лемана (1-WL)

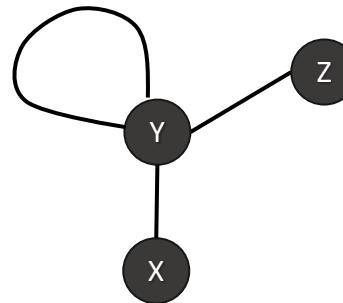
- это классический алгоритм проверки изоморфизма графов;
2. После нескольких итераций сравниваем **мультимножества меток** двух графов:
- Если на каком-то шаге они различаются → графы **не изоморфны**.
 - Если после всех шагов совпадают → графы **возможно изоморфны** (но не гарантировано).

Тест Вайсфайлера–Лемана (1-WL)



Граф 1

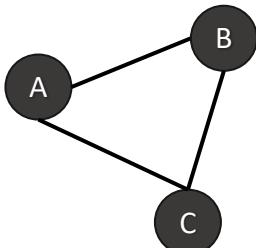
A(gray), B(gray), C(gray)



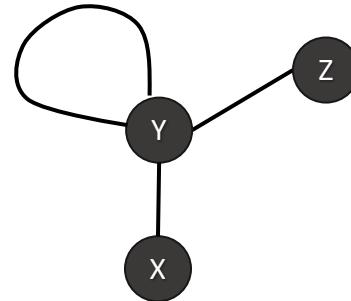
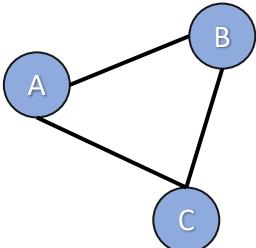
Граф 2

A(gray), B(gray), C(gray)

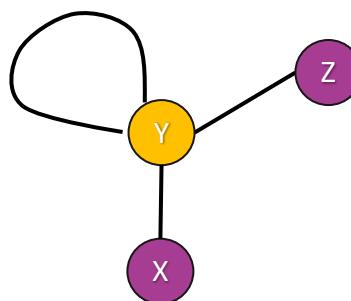
Тест Вайсфайлера–Лемана (1-WL)



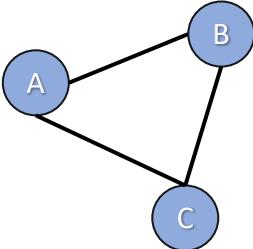
Узел	Соседи	Множество цветов соседей	Новый цвет
A	{B, C}	{gray, gray}	$\text{hash}(\text{gray}, \{\text{gray}, \text{gray}\}) = \text{blue}$
B	{A, C}	{gray, gray}	blue
C	{A, B}	{gray, gray}	blue



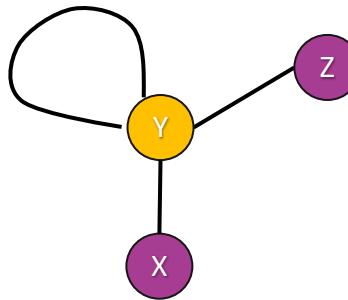
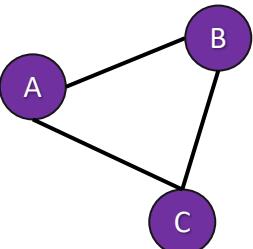
Узел	Соседи	Множество цветов соседей	Новый цвет
X	{Y}	{gray}	$\text{hash}(\text{gray}, \{\text{gray}\}) = \text{pink}$
Y	{X, Z, self}	{gray, gray, gray}	$\text{hash}(\text{gray}, \{\text{gray}, \text{gray}, \text{gray}\}) = \text{orange}$
Z	{Y}	{gray}	pink



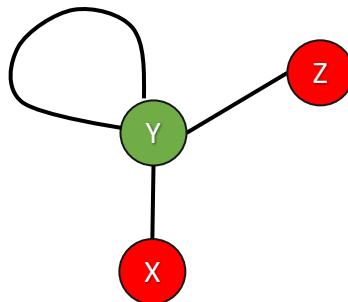
Тест Вайсфайлера–Лемана (1-WL)



Узел	Соседи	Множество цветов	Новый цвет
A	{B, C}	{blue, blue}	$\text{hash}(\text{blue}, \{\text{blue}, \text{blue}\}) = \text{violet}$
B	{A, C}	{blue, blue}	violet
C	{A, B}	{blue, blue}	violet

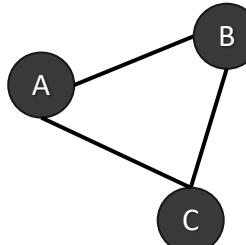


Узел	Соседи	Множество цветов	Новый цвет
X	{Y}	{orange}	$\text{hash}(\text{pink}, \{\text{orange}\}) = \text{red}$
Y	{X, Z, self}	{pink, pink, orange}	$\text{hash}(\text{orange}, \{\text{pink}, \text{pink}, \text{orange}\}) = \text{green}$
Z	{Y}	{orange}	red

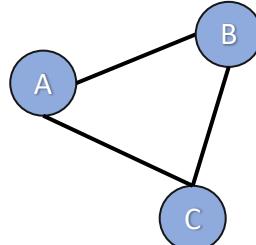


Тест Вайсфайлера–Лемана (1-WL)

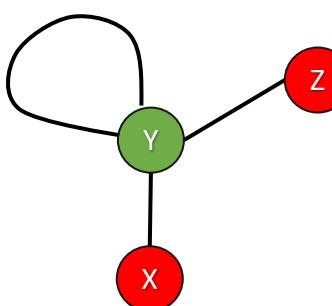
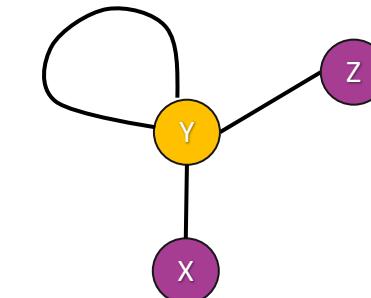
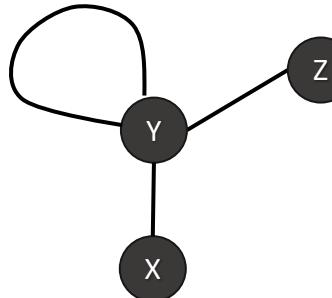
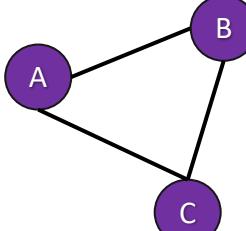
Шаг 0



Шаг 1

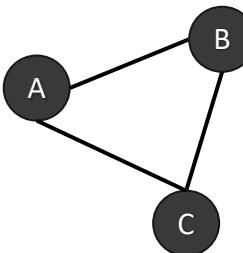


Шаг 2



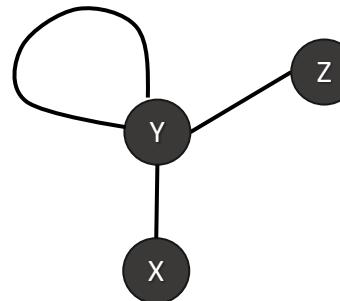
Тест Вайсфайлера–Лемана (1-WL) VS GCB

Один слой GCN обновляет признаки так: $H' = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H W$



У каждого узла 3 соседа (включая self-loop),
все имеют значение 1

$$h'_A = h'_B = h'_C = \frac{1 + 1 + 1}{3} = 1$$



Рассчитаем усреднение (также без весов, просто нормализованная сумма).

- Для X: соседи {X, Y} \rightarrow среднее $= (1 + 1)/2 = 1$
- Для Y: соседи {Y, X, Z} \rightarrow среднее $= (1 + 1 + 1)/3 = 1$
- Для Z: соседи {Z, Y} \rightarrow среднее $= (1 + 1)/2 = 1$

GIN - архитектура

1-WL:

$$c_i^{(t+1)} = \text{Hash}\left(c_i^{(t)}, \{c_j^{(t)} : j \in N(i)\}\right)$$

Интерпретация:

новый цвет вершины — это уникальный код,
зависящий от её текущего цвета и всех цветов соседей.

Переводим идею в GNN-форму:

Добавим обучаемую функцию (MLP), которая заменит "Hash" в WL-тесте:

$$h_i^{(k)} = \text{MLP}\left((1 + \epsilon)h_i^{(k-1)} + \sum_{j \in N(i)} h_j^{(k-1)}\right)$$

Примеры MPNN-функций для разных архитектур

Архитектура	Message $M_k(h_i, h_j)$	Update $U_k(h_i, m_i)$
GCN	$M(h_i, h_j) = \frac{1}{\sqrt{d_i d_j}} W h_j$	$U(h_i, m_i) = \sigma(m_i)$
GraphSAGE	$M(h_i, h_j) = h_j$	$U(h_i, m_i) = \sigma(W[h_i \ \text{AGG}(m_i)])$
GAT	$M(h_i, h_j) = \alpha_{ij} W h_j$	$U(h_i, m_i) = \sigma(m_i)$
GIN	$M(h_i, h_j) = h_j$	$U(h_i, m_i) = \text{MLP}((1 + \epsilon)h_i + m_i)$

Тест Вайсфайлера–Лемана (k-WL)

GIN:

$$h_v^{(k)} = \text{MLP} \left((1 + \epsilon) h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)} \right)$$

→ это агрегация по соседям вершины.

Чтобы перейти к k-WL:

- мы начинаем обновлять **представления не для отдельных вершин, а для k-кортежей вершин:**

$$h_{(v_1, \dots, v_k)}^{(t+1)} = f \left(h_{(v_1, \dots, v_k)}^{(t)}, \{h_{(v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_k)}^{(t)} : w \in V\} \right)$$

ИСТОЧНИКИ

- **GCN — Graph Convolutional Networks** Kipf, T. N., & Welling, M. (2017). *Semi-Supervised Classification with Graph Convolutional Networks*. ICLR.
 - Классическая модель GCN, основанная на спектральном приближении и нормализованной матрице смежности.
<https://arxiv.org/abs/1609.02907>
- **Улучшения GCN и проблема over-smoothing** Li, Q., Han, Z., & Wu, X.-M. (2018). *Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning*. AAAI.
 - Первая работа, формально показавшая эффект over-smoothing в GCN.
<https://arxiv.org/abs/1801.07606>
- **Rong, Y., Huang, W., Xu, T., & Huang, J.** (2020). *DropEdge: Towards Deep Graph Convolutional Networks on Node Classification*. ICLR.
 - Метод DropEdge для стабилизации глубоких GNN.
<https://arxiv.org/abs/1907.10903>
- **Zhao, L., Akoglu, L., & others.** (2020). *PairNorm: Tackling Oversmoothing in GNNs*.
 - Нормализация PairNorm для предотвращения схлопывания эмбеддингов.
<https://arxiv.org/abs/1909.12223>
- **GraphSAGE — Sample & Aggregate** Hamilton, W., Ying, Z., & Leskovec, J. (2017). *Inductive Representation Learning on Large Graphs*. NeurIPS.
 - Модель GraphSAGE: выборочные соседи, агрегаторы Mean/Pool/LSTM, L2-нормализация.
<https://arxiv.org/abs/1706.02216>

ИСТОЧНИКИ

- **GAT — Graph Attention Networks** Veličković, P., Cucurull, G., Casanova, A., Romero, A., et al. (2018). *Graph Attention Networks*. ICLR.
 - Первая GNN с механизмом внимания по рёбрам.
<https://arxiv.org/abs/1710.10903>
- **Официальный GitHub GAT**
 - Исходный код, примеры и визуализации.
<https://github.com/PetarV-/GAT>
- **Выразительность GNN (Weisfeiler-Leman)** Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). *How Powerful are Graph Neural Networks?* ICLR.
 - Показано соответствие GIN и тесту 1-WL.
<https://arxiv.org/abs/1810.00826>
- Morris, C., Ritzert, M., Fey, M., et al. (2019). *Weisfeiler and Leman Go Neural: Higher-order GNNs*. AAAI.
 - Построение k-WL-эквивалентных GNN.
<https://arxiv.org/abs/1810.02244>
- **Stanford CS224W: Machine Learning with Graphs**
 - Основной академический курс по GNN.
<http://snap.stanford.edu/class/cs224w>