

# Литературный обзор по теме «Разработка алгоритмов обучения без учителя для графовых структур данных»

Андреева Полина

## Содержание

<b>1</b>	<b>Введение</b>	<b>3</b>
<b>2</b>	<b>Обучение нейронных сетей</b>	<b>4</b>
<b>3</b>	<b>Варианты графовых нейронных сетей</b>	<b>5</b>
<b>4</b>	<b>Энкодер и функция потерь в графовой нейронной сети</b>	<b>6</b>
<b>5</b>	<b>Обозначения и постановка задачи</b>	<b>8</b>
<b>6</b>	<b>Наборы данных и метрики качества</b>	<b>9</b>
<b>7</b>	<b>Классификация и описание методов</b>	<b>13</b>
7.1	Виды функции потерь . . . . .	13
7.2	Обобщения . . . . .	24
7.3	Варианты энкодеров . . . . .	26
<b>8</b>	<b>Рекомендации по использованию методов</b>	<b>28</b>
<b>9</b>	<b>Заключение</b>	<b>29</b>
	<b>Список использованных источников</b>	<b>30</b>

# 1 Введение

Графовое представление данных это естественный, понятный человеку, в связи с чем, очень распространенный способ визуализации тех данных, которые могут взаимодействовать друг с другом. Графы используются повсеместно: от генетики до банковского дела. Как следствие, может возникать множество важных задач с использованием информации из графов: рекомендации товаров/фильмов/друзей в социальных сетях, классификация протеинов в молекулах и т.д. Эти задачи можно разделить на 4 основных типа:

- Предсказание связи между вершинами
- Классификация вершин
- Классификация ребер
- Классификация графов

Из-за своей распространенности, а также типов возникающих задач, графы могут играть важную роль в машинном обучении.

Но задачи машинного и глубокого обучения оперируют как правило с матричными структурами. Поэтому основной проблемой графов является включение информации из графовой структуры, являющейся намного более сложной, чем матричная, в модель машинного обучения. Традиционные подходы используют для этой цели какие-либо количественные параметры графа (матрица смежности, степени вершин, самая короткая длина пути между вершинами и т.д.) [1, 2] что является довольно ограниченным способом: во-первых, невозможно изменять эти значения во время обучения, так как они считаются один раз во время препроцессинга, во-вторых, данные подходы могут быть очень неэффективными по времени из-за высокой размерности графа.

Поэтому, более эффективным способом оказалось независимое от задачи обучение представлений вершин графов в низкоразмерном пространстве. Что является уже не препроцессингом, а задачей машинного обучения само по себе [4, 24]. Данные представления вершин называются эмбедингами и получают из совмещения информации, полученной из матрицы признаков и от соседей. Эмбединги строятся во время обучения, как и в любой нейронной сети в ходе минимизации некоторой функции, называемой функцией потерь и означающей насколько далекий результат получился от желаемого. Варианты функций потерь для графовых нейронных сетей и их влияние на качество построенных эмбедингов будут рассмотрены в данном обзоре.

## 2 Обучение нейронных сетей

В самом общем случае, нейронная сеть состоит из трех частей: входной слой; скрытые слои, в которых происходят линейные преобразования с матрицей весов  $\mathbf{W}$  входных данных в выходные; и выходной слой с нелинейным преобразованием и получением результата. А обучение нейронной сети включает в себя два этапа, которые проведенные друг за другом называются эпохой.

Первый этап — это прямое преобразование входных данных в выходные, называющийся прямым проходом. Матрица весов чаще всего инициализируется случайно, но по итогу обучения веса подбираются таким образом, чтобы результат выходного слоя был достаточно удовлетворительным. Для определения того, насколько хорош этот результат после каждой эпохи, необходимо ввести некоторую функцию, называемую функцией потерь  $\mathcal{L}$ . Аргументами данной функции являются: полученный результат в ходе прямого прохода, а также, в некоторых случаях, результат, к которому надо стремиться. Значением этой функции является потеря, показывающая насколько далеко результат от желаемого. То есть, веса настраиваются таким образом, чтобы минимизировать функцию потерь.

Необходимым условием экстремума для любой дифференцируемой функции является равенство нулю производной по всем переменным. В случае функции потерь нейронной сети, нулю приравниваются производные по весам. Но так как вид функции потерь в общем случае произвольный, а количество весов, как правило, более трех, то решать систему уравнений для поиска оптимума функции потерь оказывается невозможным. Поэтому в машинном обучении используют для этой цели метод градиентного спуска. Сначала берут производную по весам от функции потерь. Более того, если сеть глубокая, то есть, если скрытых слоев, идущих последовательно друг за другом, больше, чем один, то и производные по весам более ранних слоев, надо брать последовательно, выражая через производные по весам более поздних слоев (производная сложной функции). Это и есть второй этап, называемый обратным распространением ошибки. После окончания этого этапа получают производные по всем весам. Вектор, состоящий из этих производных, — градиент функции потерь, указывающий в сторону ее возрастания в пространстве весов. Так как необходимо получить именно минимум функции, то каждый вес "исправляется" в сторону, обратную градиенту. В общем случае:

$$\mathbf{W} = \mathbf{W} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \quad (1)$$

где  $\lambda$  — параметр, означающий длину шага изменения веса в сторону, обратную градиенту. Называется скоростью обучения.

В задачах, где есть размеченные данные, функция потерь показывает насколько расходитсся результат с настоящими отметками. Но таких размеченных данных бывает мало, или для какой-то конкретной задачи может не быть вовсе. Для решения таких задач была придумана парадигма неконтролируемого обучения. Самой распространенными вариациями задач такого рода являются кластеризация (минимизируются расстояния между

данными), а самым распространенным решением являются автоэнкодеры (сначала входная информация сжимается до небольшого размера, а затем восстанавливается, минимизируется ошибка реконструкции).

Еще одним важным подходом в обучении нейронных сетей является построение представления (эмбединга) в более низкоразмерном пространстве для каждого входного элемента данных. Этот подход является наиболее общим, потому что теперь имеется возможность перевести данные в низкоразмерные представления, и уже после обучения отдельно на эмбедингах решать задачи классификации, предсказания и тд. Например, данный способ очень помогает в случаях, если стоит задача классификации, а размеченных данных слишком мало. В таком случае можно сначала обучить представления, а затем обучить классификатор на представлениях имеющихся данных. После этого, для новых вершин построить представление и в пространстве эмбедингов найти вершины какого класса ближе к рассматриваемой. Функции потерь в случае обучения представлений также делятся на контролируемые (например кросс-энтропия, сравнивающая результат с имеющимися данными) и неконтролируемые (ошибка реконструкции).

Именно такое построение представлений активно используется в графовых нейронных сетях. Такая популярность достигнута благодаря независимости данного подхода от конкретной задачи и удобству отображения графово-структурированных данных в матрично-структурированные.

### 3 Варианты графовых нейронных сетей

Как видно из предыдущего пункта, на текущий момент для решения вопроса графовых нейронных сетей наиболее популярный и эффективный способ - построение эмбедингов. Такие методы разделяются по двум критериям. Во-первых, во время построения эмбедингов, то есть в процессе прямого прохода - по наличию или отсутствию возможности распространения задач на невидимые до тренировки данные. Во-вторых, по формату функций потерь - контролируемые или нет.

Transductive манера построения функции потерь предполагает изменение самого вектора - эмбединга в процессе минимизации функции потерь; все последующие задачи можно решать только на уже имеющихся эмбедингах [13,17,18]. Inductive манера позволяет в процессе решения последующих задач вводить новые вершины и строить эмбединги для них. Для этого настраиваются во время обучения сети не сами эмбединги, а параметры некоторой функции  $f : G \rightarrow \mathbb{R}^d$ , отображающей граф  $G(V, E)$ , где  $V$  - множество вершин графа, а  $E$  - множество рёбер графа, в евклидово пространство размерностью  $d$ . Последний способ вдохновлен сверточными нейронными сетями [21,22,32].

Важную роль в обучении представлений играет функция потерь, ведь именно в процессе ее минимизации выводится более оптимальный эмбединг. В задачах на графах можно также как и в обычном машинном обучении, рассматривать как контролируемые функции потерь (сравнение предсказаний классов с реальными значениями [32]), так и

неконтролируемые (ошибка реконструкции [19, 26], новые варианты функций потерь, появившиеся благодаря графовой структуре данных, позволяющей измерять некоторого рода "близость" вершин в графе [11, 13, 17, 25]).

Варианты неконтролируемых, индуктивных задач предоставляют большие возможности. Поэтому основной причиной рассмотрения именно этих вариаций является их универсальность и гибкость для решения множества последующих задач.

## 4 Энкодер и функция потерь в графовой нейронной сети

### Энкодер

Основная идея построения энкодера в том, чтобы на каждом слое сети каким-либо образом агрегировать информацию от соседей, затем соединить новый эмбединг соседства с эмбедингом рассматриваемой вершины на предыдущем слое и воспользоваться этим эмбедингом для построения представления следующей вершины (см. Рис. 1).

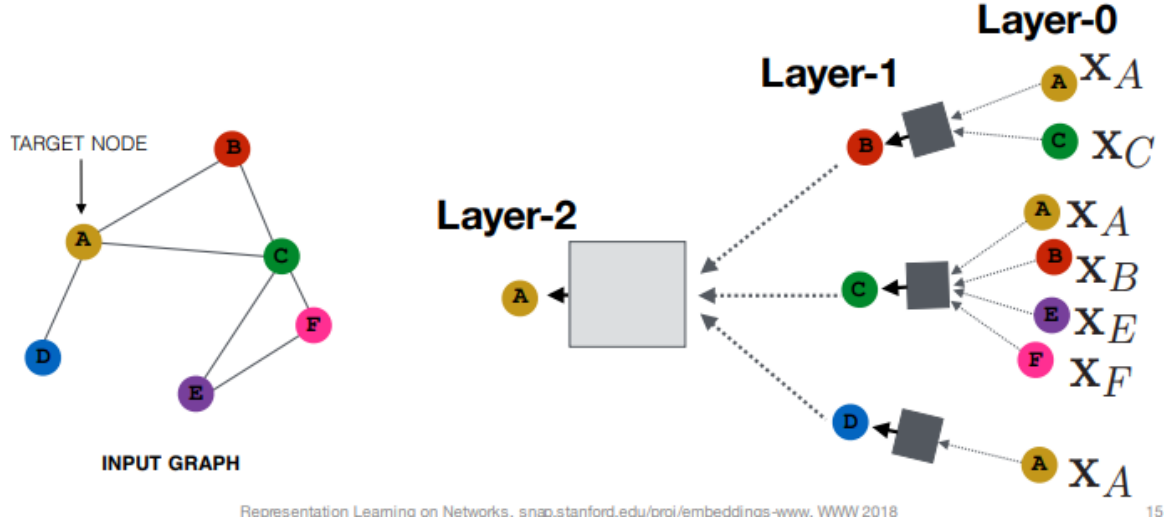


Рисунок 1 [32]

Более детально: для построения эмбединга  $\Phi_v$  вершины  $v$  за  $K$  шагов, необходимо инициализировать начальные векторы для каждой вершины. Как правило в качестве инициализации выбирается вектор признаков вершины, если такой имеется, если нет - то случайно.

$$\mathbf{h}_v^0 = \mathbf{x}_v \quad (2)$$

Далее, с помощью некоторой функции - агрегатора, собирается информация от соседей рассматриваемой вершины в один новый эмбединг соседства. Эта информация

соединяется с эмбедингом самой вершины на предыдущем слое нейронной сети, и на эту агрегированную информацию применяется слой нейронной сети. Как итог - получается эмбединг вершины на новом слое.

$$\mathbf{h}_v^k = \sigma(\mathbf{W}_k \cdot AGG(\{\mathbf{h}_u^{k-1} : u \in \mathcal{N}(v) \cup \{v\}\})) \quad (3)$$

В данном случае,  $\mathbf{h}_v^k$  – представление вершины  $v$  на  $k$ -ом слое.

Наиболее простым вариантом функции  $AGG$  является средняя сумма [32]. Эмбединги, полученные на последнем слое  $K$ , являются конечным вариантом  $\Phi_v$ . В ходе обучения нейронной сети, которое подразумевает минимизацию функции потерь, происходит "настраивание" весов  $\mathbf{W}$ .

### Функции потерь в графовых нейронных сетях

В случае неконтролируемых задач графовых нейронных сетей, в качестве функции потерь может выступать ошибка реконструкции. Для этого, помимо энкодера, строится и декодер, и минимизируется разница между реальным и декодированным значением:  $\|X - \hat{X}\|$  или  $\|A - \hat{A}\|$  [19, 26]. Кроме того, есть случай, когда вместо декодера аналогичным образом строят дискриминатор. Но все же более интересным вариантом будет рассмотреть функцию потерь, построенную таким образом, чтоб учитывать графовую структуру данных без настройки дополнительных слоев. В таком случае, функция потерь будет стремиться к тому, чтобы ближайшие соседи в графе были также близко и в низкоразмерном пространстве. Такие функции потерь используют при построении и transductive эмбедингов в методах [10, 13, 17, 18] и др., но их достаточно просто распространить и на графовые сверточные нейронные сети. В общем случае, при рассмотрении последнего варианта функций потерь встает вопрос, как обозначить "близость" вершин и эмбедингов. Про близость в низкоразмерном пространстве говорить достаточно просто - это, например, косинусное расстояние  $\Phi_u^T \cdot \Phi_v$  или разница между векторами. Осталось определить близость вершин в графе. Обозначим  $similarity(v, u)$  - какая-то абстрактная схожесть вершины  $v$  и  $u$ .

Вариантов для вида  $similarity$  множество:

- непосредственная близость по соединениям ребрами (тогда  $similarity(u, v)$  - это элемент матрицы смежности) [5, 9, 17]
- вероятность появления в ходе случайного блуждания ( $similarity(u, v)$  – вероятность появления данной пары в ходе случайного блуждания) [13, 16, 18, 26]
- какая - то особая метрика близости, функция, которая для каждой пары вершин выдает значение, типа Personalized PageRank. [10, 11, 25]
- и тд.

Итого, идея функция потерь для неконтролируемой задачи на графах выражена в следующем соотношении:

$$similarity(v, u) \approx \Phi_u^T \cdot \Phi \quad (4)$$

Наконец, можно подвести итог и перейти к постановке задачи

## 5 Обозначения и постановка задачи

### Обозначения

Граф представляется в виде упорядоченной пары  $G = (V, E)$ , где  $V = \{v_i\}$  – непустое множество вершин, а  $E = \{e_{i,j}\}$  – множество пар вершин, называемых ребрами ( $e_{i,j}$  – это ребро между вершинами  $v_i, u_i \in V$ ). Порядок графа  $|V|$  – число вершин в графе, для краткости обозначается  $n$ . Аналогично, размер графа  $|E|$  – число рёбер, обозначим  $m$ .  $\mathcal{N}(v)$  – множество, состоящее только из тех вершин, которые являются соседями для вершины  $v$ .  $\mathbf{A}$  – матрица смежности, где каждое число  $a_{ij}$  означает наличие ( $a_{i,j} = 1$ ) или отсутствие ( $a_{i,j} = 0$ ) ребра между вершинами  $v_i$  и  $v_j$  в случае невзвешенного графа и вес ребра в случае взвешенного. Диагональная матрица  $\mathbf{D}$  является матрицей степеней, у которой на диагонали стоят степени соответствующей вершины:  $d_{ii} = \sum_{j=1}^n a_{ij}$ .  $\mathbf{W} = \{w_{ij}\}$  – матрица настраиваемых весов в нейронной сети. Матрица  $\mathbf{X}$  – матрица признаков вершин в графе. Каждая колонка  $X_i$  – это вектор признаков соответствующей вершины  $v_i$ . Размерность пространства признаков равна  $d$ .  $\mathcal{L}$  – функция потерь.  $\Phi_i, \Theta_i$  – векторы в низкоразмерном пространстве, т. н. эмбединги вершины  $v_i$ . Первый вектор означает представление вершины как исходной, а второе – представление вершины как контекстной для другой вершины.

### Постановка задачи

Используя граф  $G(V, E)$ , матрицу смежности графа  $A$ , матрицу признаков  $X$ , необходимо построить функцию  $f$ , которая отображает каждую вершину графа в вектор в низкоразмерном пространстве (см. Рисунок 2)  $f : G \rightarrow R^n$  таким образом, чтобы данные векторы (называемые эмбедингами) похожих вершин графа лежали ближе друг к другу  $similarity(u, v) \approx \Phi_u^T \cdot \Phi$ . Соответственно для этого необходимо также определить и меру схожести эмбедингов.

В данном обзоре будут рассмотрены различные функции потерь и проведен их сравнительный анализ на нескольких вариантах сверточных нейронных сетей.

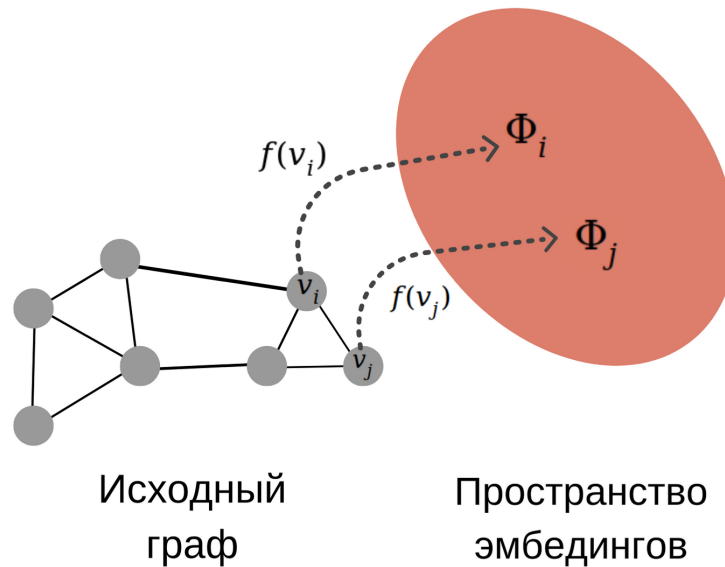


Рисунок 2

## 6 Наборы данных и метрики качества

### Задачи

Качество построенных эмбедингов проверяется на последующих задачах. Чаще всего решаются задачи классификации вершин и предсказания рёбер между вершинами, реже - кластеризации вершин. Также встречаются случаи, когда авторы метода проверяют качество, визуализируя получившиеся эмбединги или реконструируя граф и считая разницу с изначальным.

В данном разделе описаны наиболее популярные наборы данных и метрики для проверки качества методов, которые будут изложены в следующем разделе.

### Наборы данных

Наборы данных по смыслу делятся на социальные сети, сети цитирования, сети смежных слов и тд, так что представим в данном разделе соответствующую классификацию. В таблице 1 представлена краткая справка по наиболее часто встречаемым наборам данных: размер, направленность, а описание методов представлено ниже:

1. Социальные сети: SN-Twitter, Flickr, YouTube, Reddit, Epinions, BlogCatalog.

Вершины представляют собой пользователей, а ребра - дружбу между ними. Некоторые сети имеют атрибуты узлов, как например у сети авторов BlogCatalog атрибуты - темы, на которые пишет данный автор. У сети YouTube, метки вершин представляют предпочитаемые жанры.

2. Сети цитирования: Cora, Citeseer, CoCit, DBLP, PubMed.

Каждая вершина представляет собой "мешок слов" статьи, а ребро между двумя вершинами - существование цитирования. Метка вершины - это сфера тематики данной статьи.

3. Сети со-авторств: Arxiv.

Вершины - авторы. Ребро между двумя вершинами существует, если авторы участвовали в написании одной статьи.

4. Сети слов: Wikipedia.

Вершины - слова. Если между двумя вершинами стоит ребро, значит эти два слова появляются в окне из 5 слов не менее 5 раз. Метки означают части речи.

5. Другое:

Zachary's karate network - известная и часто используемая для визуализации сеть университетского клуба карате.

PPI Protein-Protein Interaction. Каждый граф соответствует отдельной ткани человека. Вершины - протеины с набором атрибутов. Метка - роль белка с точки зрения его клеточных функций.

Таблица 1: Справка по наборам данных.

Категория	Название датасета	Характеристика	Размер
Социальная сеть	SN-Twitter	Направленный	$ V  = 465K$ $ E  = 834K$
	Flickr	Ненаправленный	$ V  = 80K$ $ E  = 5,90M$ $ L  = 195$
	YouTube	Ненаправленный	$ V  = 1,13M$ $ E  = 2,99M$ $ L  = 47$
	Reddit	Ненаправленный	$ V  = 231K$ $ E  = 11,6M$ $ L  = 41$
	Epinion	Направленный	$ V  = 75K$ $ E  = 508K$
	BlogCatalog	Ненаправленный	$ V  = 10K$ $ E  = 33K$ $ L  = 39$
Сеть цитирования	Cora	Ненаправленный	$ V  = 23K$ $ E  = 91K$ $ L  = 70$
	Citeseer	Ненаправленный	$ V  = 3K$ $ E  = 4K$ $ L  = 6$
	CoCit	Направленный	$ V  = 44K$ $ E  = 195K$ $ L  = 15$
	DBLP-Ci	Направленный	$ V  = 12.5K$ $ E  = 49K$
	PubMed	Направленный	$ V  = 19K$ $ E  = 44K$ $ L  = 3$
Сеть соавторств	Arxiv GR-QC i	Ненаправленный	$ V  = 5K$ $ E  = 28K$
Сеть смежности слов	Wikipedia	Ненаправленный	$ V  = 4K$ $ E  = 184K$ $ L  = 40$
Другое	Zachary's karate network	Ненаправленный	$ V  = 34$ $ E  = 78$ $ L  = 4$
	PPI	Ненаправленный	$ V  = 3K$ $ E  = 38K$ $ L  = 50$

Наиболее часто используемые **метрики** для проверки качества метода на конкретных задачах собраны в таблице 2:

Таблица 2: Метрики качества.

Метрика	Формула, описание	Для каких задач
Micro-F1 score <sup>1</sup>	$\frac{2 \cdot P \cdot R}{P + R}$	Классификация вершин
Macro-F1 score	$\frac{\sum_{l \in L} F1(l)}{ L }$ , где $F1(l)$ – $F1$ – score для метки $l$	Классификация вершин
AUC (Area Under Curve)	Площадь под кривой ошибок (кривая в осях $TP/(TP + FN)$ к $FP/(TN + FP)$ , где точки относятся к различным значениям порога отсечения, при котором считается, что ребро существует)	Предсказание рёбер
Precision	$\frac{N^k \cap N(v)}{k}$ , где $N^k$ – $k$ ближайших соседей вершины судя по построенным эмбедингам, $N(v)$ – истинные соседи вершины $v$ .	Реконструкция графа
NMI (Normalized Mutual Information)	$\frac{2 \cdot I(Y; C)}{H(Y) + H(C)}$ , где $Y$ – истинные метки классов, $C$ – метки кластеризации, $H(., .)$ – энтропия $I(., .)$ – взаимная информация	Кластеризация вершин

---

<sup>1</sup>Данная метрика считается глобально, подсчетом  $TP$  = true Positives,  $FP$  = False Postives,  $FN$  = False Negative по всем меткам  $l \in L$ .

$$P = \frac{\sum_{l \in L} TP(l)}{\sum_{l \in L} (TP(l) + FP(l))}, R = \frac{\sum_{l \in L} TP(l)}{\sum_{l \in L} (TP(l) + FN(l))}$$

## 7 Классификация и описание методов

Во-первых, функции потерь можно в общем случае разделить на несколько основных видов:

- Максимизация вероятности появления одной вершины в соседстве другой
- Задание некоторой матрицы, каждый элемент которой означает меру схожести соответствующих вершин

Более подробно каждый вариант из рассматриваемых функций потерь будет описан в пункте 7.1

Во-вторых, реализация этих функций потерь будет рассмотрена на нескольких разных энкодерах, взятых из методов GCN (graph convolutional network), GAN (graph attention network), GraphSAGE (Sample and Aggregate). Различия между этими энкодерами описано в пункте 7.3

### 7.1 Виды функции потерь

В данном разделе кратко описаны идеи всех рассматриваемых в обзоре функций потерь и обобщены в таблице 4.

В таблице 3 собраны функции потерь по отдельности. В первом столбце — название метода, где появилась данная функция потерь. Во втором — рассматривается ли для вершины только одно представление или два (целевое, которое является основным и контекстное — являющееся вспомогательным для построения эмбедингов других вершин), в третьем — сам вид функции потерь, а в последнем — способ оптимизации.

- Метод Laplacian Eigenmaps [5] стремясь сохранить два эмбединга ближе в случае, когда в графе между соответствующими вершинами больше вес (в случае взвешенного графа), минимизирует следующую функцию потерь:

$$\begin{aligned}\mathcal{L}(\Phi) &= \frac{1}{2} \sum_{i,j} |\Phi_i - \Phi_j|^2 a_{i,j} = \frac{1}{2} \sum_{i,j} |\Phi_i - \Phi_j|^2 a_{i,j} = \frac{1}{2} \sum_{i,j} (\Phi_i^2 + \Phi_j^2 - 2\Phi_i\Phi_j) a_{i,j} = \\ &= \frac{1}{2} \sum_i \Phi_i^2 d_{ii} + \frac{1}{2} \sum_j \Phi_j^2 d_{jj} - \sum_{i,j} \Phi_i\Phi_j a_{i,j} = \text{tr}(\Phi^T \mathbf{L} \Phi)\end{aligned}\tag{5}$$

где  $\mathbf{L} = \mathbf{D} - \mathbf{A}$  — лапласиан графа,  $\Phi$  — матрица эмбедингов, в которой отдельный ряд  $\Phi_v$  соответствует эмбеддингу вершины  $v$ .

Решение этой задачи — это собственные вектора, соответствующие  $d$  наименьшим собственным числам нормализованного Лапласиана графа.

- Graph Factorization [9] предполагает, что степень схожести между вершинами определяет матрица смежности:

$$\mathcal{L}(\Phi) = \frac{1}{2} \sum_{i,j} (a_{i,j} - \Phi_i \cdot \Phi_j)^2 = \|\mathbf{A} - \Phi \cdot \Phi^T\|^2\tag{6}$$

- В методе HOPE [10], также как и в предыдущем минимизируется  $\|\mathbf{C} - \Phi \cdot \Theta\|^2$ , где  $\mathbf{C}$  — матрица, каждый элемент которой отражает схожесть между двумя соответствующими вершинами. Для минимизации этой функции потерь используется SVD - разложение:

1. В случае Katz Index, матрица  $\mathbf{C}$  представляет собой взвешенную сумму всех возможных путей между двумя вершинами.

$$\mathbf{C}^{Katz} = \sum_{l=1}^{\infty} \beta \cdot \mathbf{A}^l = \beta \cdot \mathbf{A} \cdot \mathbf{C}^{Katz} + \beta \cdot \mathbf{A}, \quad (7)$$

где  $\beta$  — коэффициент затухания определяет как быстро затухает вес пути с возрастанием длины этого пути. В пределе получится:

$$\mathbf{C}^{Katz} = (\mathbf{I} - \beta \cdot \mathbf{A})^{-1} \cdot \beta \cdot \mathbf{A} \quad (8)$$

2. Rooted PageRank: элемент матрицы  $\mathbf{C}$  в данном случае означает вероятность для конкретной вершины, оказаться в ходе бесконечного случайного блуждания с рестартом в другой вершине.

$$\mathbf{C}^{RPR}(t) = \alpha \cdot \mathbf{C}^{RPR}(t-1) \cdot (\mathbf{D}^{-1} \mathbf{A}) + (1 - \alpha) \cdot \mathbf{I} \quad (9)$$

где  $\alpha$  — вероятность случайного перехода к соседу. Если для краткости  $\mathbf{D}^{-1} \mathbf{A}$  обозначить одной матрицей  $\mathbf{P}$ , а также рассмотреть  $\mathbf{C}^{RPR}$  при стремлении  $t$  к бесконечности, то получится однозначно-заданный вид:

$$\begin{aligned} \mathbf{C}^{PPR}(t) &= \alpha^2 \cdot \mathbf{C}^{PPR}(t-2) \mathbf{P}^2 + \alpha(1 - \alpha) \mathbf{P} + (1 - \alpha) \mathbf{I} = \\ &= \alpha^t \mathbf{C}^{PPR}(0) \mathbf{P}^t + (1 - \alpha) \sum_{i=0}^{t-1} \alpha^i \mathbf{P}^i \end{aligned} \quad (10)$$

$$\mathbf{C}^{PPR} = \lim_{t \rightarrow \infty} \mathbf{C}^{PPR}(t) = (1 - \alpha)(\mathbf{I} - \alpha \mathbf{P})^{-1} \quad (11)$$

3. Common Neighbors отображает количество вершин, являющихся соседями одновременно для обоих рассматриваемых вершин:

$$\mathbf{C}^{CN} = \mathbf{A}^2 \quad (12)$$

4. Adamic-Adar score аналогично Common Neighbours считает количество одних и тех же соседей для двух вершин, но учитывает некоторый "вес" каждой вершины, который обратен ее степени:

$$\mathbf{C}^{AA} = \mathbf{A} \cdot \mathbf{D} \cdot \mathbf{A} \quad (13)$$

Авторы статьи показывают, что любую матрицу  $C$ , отражающую схожесть вершин, можно представить в виде  $C = \mathbf{M}_g^{-1} \mathbf{M}_l$ , а в связи с разреженностью  $\mathbf{M}_g, \mathbf{M}_l$  можно эффективно использовать разреженное обобщенное сингулярное разложение (SVD) для построения эмбедингов графа: SVD применяется к  $\mathbf{M}_g, \mathbf{M}_l$ . Оптимальные значения эмбедингов - считаются как корни произведения сингулярного числа на соответствующий сингулярный вектор.

- Авторы статьи [27] показывают, что каждая из четырех моделей: DeepWalk, LINE, PTE, Node2vec, выполняют неявную матричную факторизацию (алгоритм NetMF). Для каждой модели выведены матричные формы. Например, алгоритм DeepWalk эквивалентен факторизации следующей матрицы:

$$\log\left(\frac{\text{vol}(G)}{T} \left(\sum_{r=1}^T \mathbf{P}^r\right) \mathbf{D}^{-1}\right) - \log(b) \quad (14)$$

в данном случае  $b$  негативных примеров для Negative Sampling,  $\text{vol}(G)$  - объем взвешенного графа (сумма степеней всех вершин),  $T$  - длина случайного блуждания,  $r$  - размер окна в DeepWalk.

Если для краткости положить  $\mathbf{M} = \left(\frac{\text{vol}(G)}{bT} \left(\sum_{r=1}^T \mathbf{P}^r\right) \mathbf{D}^{-1}\right)$ . Тогда, предлагаемый алгоритм NetMF применяет SVD к матрице  $\log \mathbf{M}$ , а оптимальные значения эмбедингов - это, также как и в предыдущем пункте, корни произведения сингулярного числа на соответствующий сингулярный вектор.

- Функция потерь в SDNE [26] состоит из двух частей, каждая из которых отвечает за близость между вершинами 1 и 2 порядка соответственно.

Близость первого порядка - локальная близость между двумя вершинами. Чем больше вес между вершинами, тем ближе вершины.

Близость второго порядка - схожесть между структурой соседства двух вершин. Математически:  $p_u = (a_{u,1}, \dots, a_{u,|V|})$  - определяет близость первого порядка для вершины  $u$  ко всем остальным. Тогда близость второго порядка между вершинами  $u, v$  определяется схожестью между  $p_v, p_u$

Во-первых, в данном методе строится автоэнкодер и в качестве учета близости второго порядка используют ошибку реконструкции матрицы смежности. В оригинальной статье используются большие штрафы для не нулевых элементов, чем для нулевых добавлением умножения на некоторые смещения  $\mathbf{b}_i$  (т.е. ненулевые элементы должны иметь меньшую ошибку реконструкции, чем нулевые):

$$\mathcal{L}_{2nd} = \|(\hat{A} - A) \odot B\|_F^2 \quad (15)$$

где  $\odot$  - произведение Адамара,  $B$ -матрицы смещений (biases).

Таблица 3: Методы неконтролируемого обучения

Метод	Контекстный эмбединг	Функция потерь	Подходы к оптимизации
Laplacian EigenMaps	×	$\frac{1}{2} \sum_{i,j}  \Phi_i - \Phi_j ^2 A_{i,j}$ $= tr(\Phi^T L \Phi)$	Matrix Factorization
Graph Factorization	×	$\frac{1}{2} \sum_{i,j} (A_{i,j} - \Phi_i \cdot \Phi_j)^2$ $+ \frac{\lambda}{2} \sum_i \ \Phi_i\ ^2$	SGD with asynchronous optimization
HOPE	✓	$\ \mathbf{C} - \Phi \cdot \Theta\ _F^2$ , $\mathbf{C}$ – матрица схожести вершин, см. уравнения 7, ??, ??	Matrix Factorization
NetMF	✓	$\ \log(\frac{vol(G)}{bT} (\sum_{r=1}^T \mathbf{P}^r) \mathbf{D}^{-1}) - \Phi \cdot \Theta\ _F^2$ (значение параметров см. в описании к ур-ию 44)	Matrix Factorization + Negative Sampling
DeepWalk	✓	$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \frac{\exp(\Theta_i \cdot \Phi_j)}{\sum_{v_k \in V} \exp(\Theta_k \cdot \Phi_j)}$ , где $N(v)$ – см. таблицу ??. В данном случае случайные блуждания фиксированной длины	Hierarchical softmax
Node2Vec	✓	$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \frac{\exp(\Theta_i \cdot \Phi_j)}{\sum_{v_k \in V} \exp(\Theta_k \cdot \Phi_j)}$ Случайные блуждания имеют два параметра - вероятность перехода к вершинам, отвечающим за исследования локальной и глобальной структур	Negative Sampling
Struc2Vec	✓	$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \frac{\exp(\Theta_i \cdot \Phi_j)}{\sum_{v_k \in V} \exp(\Theta_k \cdot \Phi_j)}$ Случайные блуждания строятся по контекстному графу (см ур.-я 24,25), учитываемому структурную схожесть вершин	Hierarchical Softmax
Seed	×	$\ X - \hat{X}\ _2^2$	Deep autoencoders

App	✓	$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \frac{1}{1+\exp(-\Theta_i \cdot \Phi_j)}$ для построения случайных последовательностей используется метод Monte-Carlo End-Point	Skip-Gram with Negative Sampling
VERSE	×	$-\sum_{i,j=1}^n \text{sim}_G(v_i, v_j) \log \frac{\exp(\Phi_i \cdot \Phi_j)}{\sum_{k=1}^n \exp(\Phi_i \cdot \Phi_k)}$ $\text{sim}_G$ – распределение схожести вершин графа. В случае PPR, $\text{sim}_G(v, \cdot)$ – последняя вершина в одном случайном блуждании начатого с вершины $v$ , в случаях других мер схожести, определяется уравнениями 29, 30	Noise Constrictive Estimations
GraphSAGE	×	$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \frac{1}{1+\exp(-\Phi_i \cdot \Phi_j)}$	Negative Sampling
SDNE	×	$\ (\hat{A} - A) \odot B\ _F^2 + \alpha \sum_{v_i, v_j \in V} a_{i,j} \ \Phi_i - \Phi_j\ _2^2$	Deep autoencoders
DGI	×	$\frac{1}{N+M} (\sum_{i=1}^N \mathbb{E}_{(X,A)} [\log \mathcal{D}(\Phi_i, s)] + \sum_{j=1}^M \mathbb{E}_{(\tilde{X}, \tilde{A})} [\log(1 - \mathcal{D}(\tilde{\Phi}_j, s))])$ где $s$ – суммарный вектор графа, $\mathcal{D}$ – дискриминатор	Noise Constrictive Estimations
LINE-1	×	$-\sum_{v_i, v_j \in V} a_{ij} \log \frac{1}{1+\exp(-\Phi_i \cdot \Phi_j)}$	Negative Sampling
LINE-2	✓	$-\sum_{v_i, v_j \in V} a_{ij} \log \frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{v_k \in V} \exp(\Phi_i \cdot \Theta_k)}$	Negative Sampling

Для того, чтобы учитывать локальную структуру (близости 1 порядка) строится функция потерь на основе Laplacian Eigenmaps – похожие вершины в графе, в пространстве эмбедингов также должны быть ближе:

$$\mathcal{L}_{1st} = \sum_{i,j=1}^n a_{i,j} \|\Phi_i - \Phi_j\|_2^2 \quad (16)$$

Общая функция потерь выглядит следующим образом:

$$\begin{aligned} \mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st} = \\ &= \|(\hat{A} - A) \odot B\|_F^2 + \alpha \sum_{i,j=1}^n a_{i,j} \|\Phi_i - \Phi_j\|_2^2 \end{aligned} \quad (17)$$

$\alpha$  - параметр, контролирующий отношение степеней важности локальной структуры к глобальной.

- В качестве еще одного метода, использующего автоэнкодер рассмотрим метод *SEED* [19], ориентированного на построения эмбедингов графов, а не вершин. Обучает автоэнкодер на случайных блужданиях типа WEAVE (каждый подграф представляется в виде матрицы  $X$ , где каждый столбец  $p$  представляет одну вершину в виде конкатенации атрибута этой вершины и наиболее раннего визита в ходе  $p$ -го блуждания). Обучение автоэнкодера происходит минимизируя ошибку реконструкции  $\hat{X}$ :

$$\mathcal{L} = \|X - \hat{X}\|_2^2 \quad (18)$$

На последнем шаге усредняются представления подграфов, или, если обобщать, то отображаются полученные представления  $\Phi$  в некоторое новое пространство и усредняют полученные представления:

$$\hat{\mu}_G = \frac{1}{s} \sum_{i=1}^s \phi(\Phi_i) \quad (19)$$

- Еще один метод, который требует дополнительного обучения некоторой функции помимо энкодеров - DGI [28]. Сначала с помощью графовой свертки получают эмбединги  $\Phi$  вершин. Затем строится функция  $\mathcal{R} : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^d$ , которая полученные эмбединги вершин переводит в один единственный суммарный вектор  $s = \mathcal{R}(\Phi)$ , относящийся к графу. В случае, когда в датасете один граф, в качестве функции потерь выступает noise-contrastive функция потерь, которая настраивает дискриминатор  $\mathcal{D}$ , различающий относится ли поступающий в нее эмбединг данному графу или не относится. Для этого, соответственно необходимо построить и эмбединги

негативных примеров  $\tilde{\Phi}_i$ . Авторами статьи утверждается, что такая функция потерь максимизирует совместную информацию между  $\Phi_i$  и  $s$ , основанную на дивергенции Дженсена - Шеннона.

Таким образом, окончательная функция потерь выглядит следующим образом:

$$\mathcal{L} = \frac{1}{N + M} \left( \sum_{i=1}^N \mathbb{E}_{(X,A)} [\log \mathcal{D}(\Phi_i, s)] + \sum_{j=1}^M \mathbb{E}_{(\tilde{X}, \tilde{A})} [\log(1 - \mathcal{D}(\tilde{\Phi}_j, s))] \right) \quad (20)$$

- Теперь можно приступить к рассмотрению методов, основанных на случайных блужданиях. Эти методы считают в качестве меры схожести двух вершин - вероятность появления их во время одного случайного блуждания.

Итак, сначала запускается  $\gamma$  случайных блужданий, каждое длиной  $t$ . Выбирается размер окна  $w$ . Для каждой вершины соседями считаются другие вершины, попавшие в окно размера  $w$ . Таким образом строится множество соседей для каждой вершины. И затем минимизируется:

$$\mathcal{L} = \sum_{v_j \in V} \sum_{v_i \in N(v_j)} -\log(P(v_i | \Phi_j)) \quad (21)$$

Самый основной и первый метод подонного рода в графовых нейронных сетях, DeepWalk, [13] использует случайные блуждания фиксированной длины.

- Node2vec [18] в отличие от DeepWalk исследует и локальную и глобальную структуру графа и с помощью двух гиперпараметров может учитывать влияние каждой из сторон: один гиперпараметр отвечает за вероятность агента во время случайных блужданий вернуться на шаг назад, т.е. за учет локальной структуры графа, а другой гиперпараметр, отвечает за вероятность агента сделать шаг в сторону от вершины, с которой началось блуждание, таким образом, учитывая глобальную структуру графа. Таким образом, в Node2Vec  $N(v)$  отличается от DeepWalk из-за самого характера случайного блуждания.
- Struc2vec [16] рассчитывает расстояние между вершинами по структурной схожести, а не топологической и является немного более сложным чем DeepWalk и Node2Vec за счет построения нового графа, называемого контекстным.

1. Мера структурной близости между двумя вершинами, учитывающими соседство размера  $k$  (соседи, расстояние до которых менее или равно  $k$  рёбер) – это разница в степени вершины и степеней ее соседей:

$$f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u), s(R_k(v)))) \quad (22)$$

где  $R_k(u)$  – множество соседей вершины  $u$  строго на расстоянии  $k$ ,  $s(R)$  – отсортированная последовательность степеней каждой вершины из  $R$ .  $g(R_1, R_2)$  – расстояние между двумя отсортированными последовательностями чисел.

2. На данном этапе строится новый, взвешенный, многослойный, так называемый контекстный, граф. Каждый слой  $k$  отвечает за соседство вершин в радиусе  $k$ . В каждом слое  $k$  находятся все вершины, а ребра между вершинами внутри слоя имеют вес, обратно пропорциональный структурной схожести между этими вершинами:

$$w_k(u, v) = \exp^{-f_k(u, v)} \quad (23)$$

Между слоями ребра соединяют лишь одни и те же вершины. Вес между ребром от  $u_k$  к  $u_{k+1}$  равен логарифму суммы ребер, инцидентных  $u_k$  и имеющих вес больший, чем средний по данному слою.

3. Далее по этому контекстному графу строится несколько случайных блужданий фиксированной длины, начиная с нулевого слоя. Вероятность перехода к вершине внутри слоя равна:

$$p_k(u, v) = \frac{\exp^{-f_k(u, v)}}{\sum_{l \neq u, l \in V} \exp^{-f_k(u, l)}} \quad (24)$$

А вероятность перейти на следующий слой:

$$\begin{aligned} p_k(u_k, u_{k+1}) &= \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})}, \\ p_k(u_k, u_{k-1}) &= 1 - p_k(u_k, u_{k+1}) \end{aligned} \quad (25)$$

4. Наконец, как и в DeepWalk, Node2vec, для построения представлений каждой вершины используется Skip-Gram, который максимизирует вероятность появления контекста вершины (контекст вершины задается окном размера  $w$ , центрированного на вершине в последовательности случайного блуждания, построенного на предыдущем шаге)
- VERSE [11] вводит распределения схожести между вершинами и минимизируют расхождение Кульбака - Лейблера (KL) между распределением схожести вершин в графе  $sim_G$  и распределением схожести эмбедингов  $sim_E$ :

$$\sum_{v \in V} KL(sim_G(v, \cdot) || sim_E(v, \cdot)) \quad (26)$$

В качестве распределения схожести вершин в пространстве эмбедингов используется нормализованное с помощью softmax скалярное произведение эмбедингов:

$$sim_E(v, \cdot) = \frac{\exp \Phi_v \Phi^T}{\sum_{i=1}^n \exp(\Phi_v \Phi_i)} \quad (27)$$

Также как и HOPE использует различные меры для распределения схожести в графе (PPR, SimRank, Adjacency similarity):

1. Personalized PageRank. По определению:

$$ppr_v(t+1) = \alpha \cdot ppr_v(t)\mathbf{P} + (1-\alpha)s, \quad (28)$$

где  $s$  — начальное распределение. В случае рассмотрения вершины  $v$ ,  $s$  — это вектор длины  $n$ , со всеми элементами равными нулю кроме одного, равного 1 на месте, соответствующем вершине  $v$ . Тогда, в пределе  $\lim_{t \rightarrow \infty} ppr_v(t) = sim_G(v, \cdot)$ , и как уже было посчитано в пункте про метод HOPE, один экземпляр  $sim_G(v, \cdot)$  это последняя вершина в одном бесконечном случайном блуждании с вероятностью рестарта  $1-\alpha$ , начатом из вершины  $v$ .

2. SimRank: мера структурной взаимосвязи двух вершин, основана на предположении, что схожие вершины связаны с другими схожими вершинами. Определяется рекурсивно:

$$sim_G^{SR} = \frac{C}{|I(u)||I(v)|} \sum_{i=1}^{|I(u)|} \sum_{j=1}^{|I(v)|} sim_G^{SR}(I_i(u), I_j(v)) \quad (29)$$

$I(v)$  — множество соседей вершины  $v$ , с ребрами, входящими в  $v$ ,  $C$  — число между 0 и 1, геометрически обесценивает важность дальних узлов.

3. Adjacency similarity: если  $Out(u)$  — степень выходящих ребер из вершины  $u$ , то

$$sim_G^{ADJ}(u, v) = \begin{cases} 1/Out(u) & \text{если } (u, v) \in E \\ 0 & \text{иначе} \end{cases} \quad (30)$$

Рассмотрим подробнее функцию потерь:

По определению расстояния Кульбака-Лейбнера для дискретных распределений:

$$KL(P||Q) = \sum_{i=1}^n p_i \log \frac{p_i}{q_i} \quad (31)$$

В случае с распределениями схожести в графе и пространстве эмбедингов, получаем:

$$\begin{aligned} KL(sim_G(v, \cdot) || sim_E(v, \cdot)) &= \sum_{i=1}^n sim_G(v, i) \log \frac{sim_G(v, i)}{sim_E(v, i)} = \\ &= \sum_{i=1}^n sim_G(v, i) \log sim_G(v, i) - \sum_{i=1}^n sim_G(v, i) \log sim_E(v, i) \end{aligned} \quad (32)$$

Но первое слагаемое, включает в себя только распределение схожести в графе, что не влияет на минимизацию, так как задается изначально. Поэтому, в качестве функции потерь остается только:

$$\mathcal{L} = - \sum_{v \in V} \text{sim}_G(v, \cdot) \log \frac{\exp \Phi_v \Phi^T}{\sum_{i=1}^n \exp(\Phi_v \Phi_i)} \quad (33)$$

Для оптимизации используется Noise Contrastive Estimation (NCE): алгоритм строит классификатор (логистическую регрессию), который разделяет вершины из настоящего распределения  $\text{sim}_G(v, \cdot)$  и распределения шума  $Q$ . Производная от NCE с увеличением негативных примеров, сходится градиенту функции (33).

В отличие от HOPE требует на вход не обязательно весь граф из-за чего может быть использован на больших графах.

- LINE [17] Явно задает две функции, отвечающие за близость 1 и 2 порядка, а затем минимизирует функцию - комбинацию из двух.

Так же, как и в VERSE, задаются два распределения совместных вероятностей двух вершин и эмбедингов: одно,  $\hat{p}$  - используя матрицу смежности, другое,  $p$  - для эмбедингов. Например для случая близости 1 порядка:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\Phi_i \cdot \Phi_j)} \quad (34)$$

$$\hat{p}_1(v_i, v_j) = \frac{a_{ij}}{\sum_{i,j \in V} a_{ij}} \quad (35)$$

А затем минимизируют расхождение Кульбака - Лейблера (KL) между распределениями:

$$\mathcal{L} = KL(\hat{p}_1, p_1) = \sum_{i,j} \hat{p}_{1ij} \log \frac{\hat{p}_{1ij}}{p_{1ij}} = \sum_{i,j} \hat{p}_{1ij} \log \hat{p}_{1ij} - \sum_{i,j} \hat{p}_{1ij} \log p_{1ij} \quad (36)$$

так как на результат минимизации функции не влияют константы, то окончательно минимизируется следующая функция:

$$\mathcal{L} = - \sum_{i,j \in V} a_{ij} \log \frac{1}{1 + \exp(-\Phi_i \cdot \Phi_j)} \quad (37)$$

В случае сохранения близости второго порядка:

$$p_2(v_j | v_i) = \frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \Theta_k)} \quad (38)$$

$$\hat{p}_2(v_j|v_i) = \frac{a_{ij}}{d_i}, \quad (39)$$

где  $d_i = \sum_{k \in N(i)} a_{ik}$  — степень вершины

Минимизируется следующая функция:

$$\sum_{i \in V} \lambda_i KL(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)) \quad (40)$$

$\lambda$  вводится для того чтоб показать важность каждой вершины в графе, которая может быть определена как степень вершины или по алгоритмам PageRank. Для простоты в оригинальной статье  $\lambda_i$  был положен как  $d_i$ . Тогда, как и в случае LINE-1, не учитывая константы, функция минимизации выглядит следующим образом:

$$\mathcal{L} = - \sum_{i,j \in V} a_{ij} \log \frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \Theta_k)} \quad (41)$$

- APP [25] в качестве меры схожести тоже считает вероятность оказаться в одной вершине, в ходе случайного блуждания бесконечной длины с вероятностью рестарта  $\alpha$ , начатого из другой. Но используется некоторая аппроксимация данного варианта с помощью метода сэмплирования Монте-Карло End-Point. Рандомно выбираются пути, начинающиеся в одной вершине с вероятностью остановки  $\alpha$  и заканчивающийся в другой. По аналогии с DeepWalk, Node2Vec, каждый путь рассматривается как направленная последовательность, но в которой пары вершин рассматриваются только по прямому направлению. Таким образом и учитывается несимметричность. Затем оптимизируется следующая функция потерь, учитывающая Negative Sampling:

$$\mathcal{L} = \sum_j \sum_i \#Sampled_j(i) \cdot (\log \sigma(\Phi_j \cdot \Theta_i) + b \cdot E_{t_n \sim P_D} [\log \sigma(-\Phi_j \cdot \Theta_n)]) \quad (42)$$

где  $\#Sampled_j(i)$  — количество путей  $(j, i)$ ,  $b$  — количество негативных примеров, берущихся из распределения  $P_D$ ,  $\sigma$  в анном случае сигмоида.

- Концепция self-supervised задач придумана как решение проблемы классификации для малого количества размеченных данных. Для этого параллельно к основной задаче решается другая, либо предобучается вначале pretext задача. Это происходит путем минимизации функции потерь  $\mathcal{L}_{self}$ , одновременно или перед тем, как минимизировать основную  $\mathcal{L}_{task}$ . То есть сначала тренируются сверточные сети на основе какой-то структурной информации графа, либо на информации атрибутов. И уже потом дооптимизируются на размеченных данных. Благодаря этому в self-supervised задачах можно найти некоторые способы использования структурной информации для неконтролируемой функции потерь для построения представлений

вершин. Например, как и в VERSE [11], можно воспользоваться понятием схожести, представленным матрицей  $C$ .

1. в статье [29] в качестве такой схожести  $C$  выбрали наиболее короткие расстояния между вершинами.  $C_{ij} = p(v_i, v_j)$ , где  $p(v_i, v_j)$  – длина самого короткого пути между вершиной  $v_i$  и  $v_j$
2. кроме того, для выбора близости между представлениями вершин в векторном пространстве можно тренировать еще один слой  $f(|\Phi_i - \Phi_j|)$ , тогда в качестве функции потерь поставить  $\mathcal{L} = ||f(|\Phi_i - \Phi_j|) - (i, j)||$  [31]
3. В [30] предлагают тренировать помимо энкодера, представляющего вершины в низкоразмерное пространство еще и классификатор, предсказывающий, для каждой вершины классы остальных, контекстных к ней вершин. Каждый класс означает длины самого короткого пути.

Все вышеперечисленные функции потерь можно разделить на некоторые основные виды, эти обобщения представлены в таблице 4

Таблица 4: Общий вид функций потерь

id	Общий вид	Методы
1	$-\sum_{i=1}^n \sum_{j=1}^n c_{i,j} \log \frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{k=1}^n \exp(\Phi_i \cdot \Theta_k)}$	VERSE, LINE
2	$-\sum_{i=1}^n \sum_{j: v_j \in N(v_i)} c_{i,j} \log \frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{k=1}^n \exp(\Phi_i \cdot \Theta_k)}$	DeepWalk, Node2Vec, Struc2Vec, APP, GraphSage
3	$  C - \Phi \cdot \Theta  ^2$	HOPE, NetMF, Graph Factorization
4	$tr(\Phi^T L \Phi) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n  \Phi_i - \Phi_j ^2 a_{ij}$	LaplacianEigenMaps, SDNE-1
5	Ошибки реконструкции (O.P.) $  \hat{M} - M  $	Seed (O.P. матрицы фичей $M = X$ ), SDNE-2 (O.P. матрицы смежности $M = A$ )

## 7.2 Обобщения

В таблице 4 представлен общий вид различных функций потерь. Можно заметить

несколько схожестей:

- В методе NetMF рассказано как можно перевести функции потерь вида 2) в вид 3)
- Кроме этого, с помощью матрицы перехода, уже встречавшейся в обзоре,  $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$  можно представить функцию потерь вида 2) в виде 1)

Рассмотрим эти два утверждения подробнее:

- Во-первых, описание метода NeMF общими словами:
  1. В изначальной функции потерь делаются несколько замен обозначений
  2. Так как функция потерь минимизируется по  $\Phi^T \Theta$ , то берется производная от  $\mathcal{L}$  по  $\Phi^T \Theta$  и приравнивается к нулю
  3. Ищутся корни. Итого находят выражение для  $\Phi^T \Theta$ . Обозначают для краткости  $\Phi^T \Theta = \log(\mathbf{M})$
  4. Но теперь стоит задача найти сами эмбединги, если известно их произведение
  5. В явном виде сложно найти, тогда минимизируется  $\|\log(\mathbf{M}) - \Phi^T \Theta\|$
  6. А решение данной минимизации будет SVD на  $\log(\mathbf{M})$ .
  7.  $\log(\mathbf{M}) = U_d \Sigma_d V_d^T$ .
  8. Оптимальные значения эмбедингов  $\Phi = U_d \sqrt{\Sigma_d}$

Например алгоритм LINE эквивалентен факторизации матрицы:

$$\log(\text{vol}(G) D^{-1} A D) - \log(b) \quad (43)$$

А алгоритм DeepWalk эквивалентен факторизации следующей матрицы:

$$\log\left(\frac{\text{vol}(G)}{T} \left(\sum_{r=1}^T \mathbf{P}^r\right) \mathbf{D}^{-1}\right) - \log(b) \quad (44)$$

в данном случае  $b$  негативных примеров для Negative Sampling,  $\text{vol}(G)$  - объем взвешенного графа (сумма степеней всех вершин),  $T$  - длина случайного блуждания,  $r$  - размер окна в DeepWalk.

- Во-вторых, в методе DeepWalk важно количество пар  $(i, j)$  ( $i$ -целевая,  $j$ -контекст) внутри окон всех запущенных случайных блужданий. Рассмотрим упрощенный вариант, когда длина случайного блуждания равна длине окна  $2w+1$ . Если окно имеет длину  $w$ , то вероятность появления пары  $(i, j)$  считается как объединение событий:  $A_1 \cup \dots \cup A_w \cup A_{-1} \dots \cup A_{-w}$ , где  $A_l$  — событие того, что случайным блужданием из

$i$  в  $j$  попали за 1 шагов. Тогда, вероятность внутри одного блуждания появления пары  $(i, j)$  равна

$$Pr(i, j) = Pr(A_{-w} \cup \dots \cup A_{-1} \cup A_1 \dots \cup A_w) = \sum_{r=1}^w (\mathbf{P}^r(i, j) + \mathbf{P}^r(j, i)) \quad (45)$$

То есть, если от вершины  $i$  было запущено  $\gamma$  случайных блужданий, то пар  $(i, j)$  будет  $\gamma \sum_{r=1}^w (\mathbf{P}^r(i, j) + \mathbf{P}^r(j, i))$  штук, или в матричном виде:  $\gamma \sum_{r=1}^w (\mathbf{P}^r + \mathbf{P}^{\mathbf{T}^r})$ . А т.к. каждый элемент матрицы  $\mathbf{P}$  больше нуля и меньше единицы, то применим формулу убывающей геометрической прогрессии:

$$\mathbf{C} = \gamma \sum_{r=1}^w (\mathbf{P}^r + \mathbf{P}^{\mathbf{T}^r}) = \gamma (\mathbf{P}(\mathbf{I} - \mathbf{P}^w)(\mathbf{I} - \mathbf{P})^{-1} + \mathbf{P}^{\mathbf{T}}(\mathbf{I} - \mathbf{P}^{\mathbf{T}^w})(\mathbf{I} - \mathbf{P}^{\mathbf{T}})^{-1}) \quad (46)$$

За  $\mathbf{C}$  мы обозначили так называемую котекстную матрицу. Теперь,

$$\sum_{i,j} c_{ij} \log\left(\frac{\exp(\Phi_i \cdot \Theta_j)}{\sum_{k \in V} \exp(\Phi_i \cdot \Theta_k)}\right) \quad (47)$$

Эквивалентно функции потерь DeepWalk для случая, когда длина окна равна длине случайных блужданий.

### 7.3 Варианты энкодеров

Довольно распространены и популярны среди методов глубокого обучения на графах методы GCN (Graph Convolutional Network) [32] и GAN (Graph Attention Network) [22]. Однако в оригинальном виде они используют в качестве функции потерь cross-entropy, которая считается для режима обучения с учителем (необходимы размеченные данные). Тем не менее можно рассмотреть только части, касающиеся неконтролируемого этапа - построения отоюражающей функции.

- GCN [32]: в качестве агрегирующей функции выбирается взвешенная сумма по соседям, и эмбединг самой вершины на прошлом слое тоже прибавляет:

$$\begin{aligned} \mathbf{h}_v^0 &= \mathbf{x}_v \\ \mathbf{h}_v^k &= \sigma(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}), \forall k \in \{1, \dots, K\} \\ \Phi_v &= \mathbf{h}_v^K \end{aligned} \quad (48)$$

- GAN [22] - Идея в том, что соседи оказывают разное влияние на данную вершину и необходимо это учесть:

$$\mathbf{h}_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}_k \mathbf{h}_u^{k-1}\right) \quad (49)$$

- GraphSAGE [21] В отличие от предыдущего метода, может использовать разные варианты агрегирования информации от соседей кроме среднего значения, например, выбор максимального значения или применение LSTM. Кроме того, еще одно отличие от базового варианта это конкатенирование представления самой вершины с прошлого слоя к агрегированной информации от соседних вершин, вместо суммирования с ней. Преимущество над предыдущим методом - это обобщение:

$$\mathbf{h}_v^k = \sigma([\mathbf{W}_k \cdot AGG(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}), \mathbf{B}_k \mathbf{h}_v^{k-1}]) \quad (50)$$

#### Выводы:

- Традиционные методы построения эмбедингов (основанные на факторизации) используют плотные матрицы, которые характеризуют граф, из-за чего эффективны лишь на небольших графах, а так же учитывают схожесть только по расстоянию между вершинами, а не атрибутам.
- Методы, основанные на случайных блужданиях учитывают не только ближайших соседей, но и глобальную структуру графа. Работают быстрее, так как используют для вычислений не все пары вершин.
- Сверточные методы учитывают только локальную структуру, но зато более вычислительно эффективны и учитывают атрибуты вершин.
- Существует ряд методов (VERSE, LINE, HOPE), которые учитывают понятие мер схожести.

## 8 Рекомендации по использованию методов

(предполагается, что эти рекомендации мы получим сами как итог - при каких энкодерах какие функции потерь более эффективны для каких задач, а пока тут информация из других статей) В работах [24] и [4] был проведен подробный сравнительный анализ рассматриваемых в данном обзоре методов.

Основные результаты следующие:

1. Методы, учитывающие роль вершины как источника и контекста при изучении представлений, рекомендуются для прогнозирования связей в ориентированных графах.
2. Простой классификатор, основанный на непосредственном соседстве, предлагает лучшую или сопоставимую производительность для ряда наборов данных.
3. Для задач предсказания связей на неориентированных графах методы на основе PPR ( APP и VERSE) - являются наиболее эффективными методами во всех наборах данных.
4. В задачах предсказания связей, метод LINE, который непосредственно использует матрицу смежности в качестве матрицы близости, превосходит методы случайного блуждания для неориентированных графов.
5. В задачах предсказания связей, для ориентированных графов с низкой взаимностью повторное представление контекста узла играет большую роль, и для задач предсказания направленных связей следует использовать методы кодирования и использования двух пространств внедрения для исходной и целевой контекстных представлений узлов.
6. В задачах предсказания связей более глубокие модели не имеют значительного преимущества перед поверхностными.
7. Для направленных графов, для задачи реконструкции графов HOPE предпочитается APP
8. Для задач классификации узлов, степень гомофилии графа должна быть подсчитана прежде чем выбирать подход. Подходы глубокого обучения, основанные на агрегации подходят для высокой степени гомофилии, а для низкой - DeepWalk.
9. Node2Vec более предпочтителен для классификации вершин, а для предсказания рёбер - методы, учитывающие разные степени схожести (HOPE,SDNE)

## 9 Заключение

Было рассмотрено множество различных методов, которые разделены на общие группы по подходу к обучению представления графов. Методы различаются также и учетом различной информации графов (локальная или глобальная структуры, схожесть по атрибутам вершин). Основным наиболее эффективным подходом остается построение эмбедингов на основе нейронных сверточных сетей, хотя и другие методы показывают преимущества на конкретных задачах. При решении задач и выборе метода, необходимо учитывать структурные свойства графов, размер графа, тип задач.

Также был сделан акцент на экспериментальную часть всех методов и основная информация по наиболее часто использованным датасетам и метрикам собрана в таблицах 1, 2

Основное будущее развитие всех методов это масштабирование на большие графы, а также улучшения методов, направленные на улучшение качества и скорости работы алгоритмов.

## Список используемых источников

- [1] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social Network Data Analytics*, pages 115–148. 2011.
- [2] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98:404–409, 2001.
- [3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Yu Philip. 2019. A comprehensive survey on graph neural networks. *arXiv:1901.00596*.
- [4] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” *Knowledge-Based Syst.*, vol. 151, 2018, doi: 10.1016/j.knosys.2018.03.022.
- [5] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*. 585–591
- [6] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.
- [7] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319– 2323
- [8] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. 2007. Graph embedding and extensions: A general framework for dimensionality reduction. *TPAMI* 29, 1 (2007)
- [9] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *WWW. ACM*, 37–48.
- [10] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*. 1105–1114
- [11] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, “VERSE: Versatile graph embeddings from similarity measures,” *Web Conf. 2018 - Proc. World Wide Web Conf. WWW 2018*, pp. 539–548, 2018, doi: 10.1145/3178876.3186120.
- [12] G. Rizos, S. Papadopoulos, and Y. Kompatsiaris, Multilabel user classification using the community structure of online networks, vol. 12, no. 3. 2017.
- [13] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: online learning of social representations,” in *KDD*, 2014, pp. 701–710.

- [14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In NIPS. 3111–3119
- [15] Shaosheng Cao, Wei Lu, and Qionghai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In CIKM. 891–900
- [16] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In KDD. ACM, 385–394
- [17] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “LINE: large-scale information network embedding,” in WWW, 2015, pp. 1067–1077.
- [18] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in KDD, 2016, pp. 855–864.
- [19] L. Wang et al., “INDUCTIVE AND UNSUPERVISED REPRESENTATION LEARNING ON GRAPH STRUCTURED OBJECTS,” 2016.
- [20] A. Bordes, N. Usunier, A. Garcia-dur, J. Weston, and O. Yakhnenko, “Translating Embeddings for Modeling Multi-relational Data,” pp. 1–9.
- [21] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” Adv. Neural Inf. Process. Syst., vol. 2017-Decem, no. Nips, pp. 1025–1035, 2017.
- [22] P. Velickovic, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, “Graph attention networks,” 6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc., pp. 1–12, 2018.
- [23] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1225–1234.
- [24] M. Khosla, V. Setty, and A. Anand, “A Comparative Study for Unsupervised Network Representation Learning,” IEEE Trans. Knowl. Data Eng., pp. 1–1, 2019, doi: 10.1109/tkde.2019.2951398.
- [25] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao. Scalable graph embedding for asymmetric proximity. In AAAI, pages 2942–2948, 2017.
- [26] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In SIGKDD, pages 1225–1234. ACM, 2016.
- [27] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In WSDM, pages 459–467, 2018.

- [28] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. In International Conference on Learning Representations (ICLR).
- [29] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhan Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. arXiv preprint arXiv:2006.10141, 2020.
- [30] Zhen Peng, Yixiang Dong, Minnan Luo, Xiao-Ming Wu, and Qinghua Zheng. Self-supervised graph representation learning via global context prediction. arXiv preprint arXiv:2003.01604, 2020.
- [31] Jin Wei, Ma Yao, Derr Tyler, Liu Zitao, Wang Yiqi, Tang Jiliang. Node similarity preserving graph convolutional networks arXiv preprint arXiv: 2011.09643 21. 56.
- [32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. CoRR, abs/1609.02907, 2016. URL <http://arxiv.org/abs/1609.02907>.