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

Андреева Полина

Содержание

1 Введение **3**

2 Задачи на графах **4**

3 Постановка задачи и необходимые обозначения **5**

4 Функции потерь **7**

5 Наборы данных и метрики качества **8**

6 Классификация и описание методов **11**
6.1 Методы, основанные на факторизации 11
6.2 Методы, основанные на случайных блужданиях 15
6.3 Методы, осованные на глубоком обучении 19
6.4 Другие методы 22

7 Рекомендации по использованию методов **24**

8 Заключение **25**

Список использованных источников **26**

1 Введение

Графовое представление данных это естественный, понятный человеку с одной стороны, и удобный, подходящий для машинной обработки с другой, способ визуализации данных из различных областей: от генетики до банковского дела. И хотя большинство самых распространенных архитектур нейронных сетей (например CNN, RNN) ориентировано на векторные структуры, в последнее время появились новые подходы глубокого обучения для представления и моделирования графово-структурированных данных.

Большинство успехов в глубоком убучении достигнуто с применением одной из двух парадигм - обучения с учителем (контролируемое обучение) или обучение с подкреплением. В первом случае модель делает предположение, а результат сравнивается с истинным значением, после чего производится обратное распространение ошибки. Во втором подходе, агент взаимодействует со средой. Он делает действие, а среда подает сигналы подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или её модель. Но в обоих случаях пределы обучения определяются людьми, внешними надзирателями. Для истинного интеллекта необходимы более независимые стретегии обучения. В связи с чем возникает еще одна парадигма - неконтролируемое обучение, когда агент обучается с целью обучиться (например, младенцы изучают мир из любопытсва, через наблюдение, без подсказок). Более того, углубляясь в понимание работы человеческого интеллекта, можно заметить, что во взрослой жизни, люди не обучаются постоянно с нуля, а приспосабливаются к текущей ситуации, используя имеющиеся навыки и знания. По аналогии с чем в глубоком обучении придумана следующая парадигма, развивающая работу искусственного интеллекта - обучение с переносом. С точки зрения затраченных ресурсов, такой способ решения задач намного эффективней, так как не требует большого объема новых данных.

В данном обзоре сначала будет рассмотрен основной подход алгоритмов на графах. Затем - кратко расшифрованы основные обозначения, необходимые при описании методов, сформулирована задача, представлены наиболее часто используемые наборы данных и метрики проверки качества в экспериментах. В следующем параграфе описаны сами методы, разделенные на классы. В конце даны рекомендации по использованию конкретных методов на конкретных задачах.

2 Задачи на графах

Графы – структуры, состоящие из уникальных сущностей (узлов или вершин графа) и связей между ними (ребер графа), являются основной формой представления и хранения знаний и удобным иснтрументом для работы с большинством видов данных. Среди основных преимуществ графового представления данных это логическая строгость и масштабируемость на большие объемы информации. Последнее время было сделано много попыток расширить сверточные нейронные сети на графы. Проблема графовых структур данных в самом не-векторном представлении. В связи с чем, основным подходом в задачах на графах стало представление элементов графа в некотором высокоразмерном пространстве, в виде векторов, так называемых эмбедингов [1]. А далее такое представление в виде таблицы подается на вход классификатору. В таком виде задача решается как для любой другой обычной нейронной сети. Методы, использующие эмбединги, включают в себя Graph Neural Networks (GNNs) [1] и были применены к различных задачам относящимся к графам.

Задачи на графах можно разделить на следующие наиболее общие группы:

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

Очень много существующих методов работают в режиме обучения без учителя – то есть, без привязки к решаемой задаче, получают векторное представление узла в графе. А качество таких эмбедингов проверяют на классических задачах, чаще всего - классификация вершин.

3 Постановка задачи и необходимые обозначения

Постановка задачи

Используя граф, матрицу смежности, матрицу признаков, необхожимо построить функцию, которая отображает каждую вершину графа в вектор в низкоразмерном пространстве (см. Рисунок 1) таким образом, чтобы данные векторы (называемые эмбеддингами) похожих вершин графа лежали ближе друг к другу. Соответственно для этого необходимо также определить и меру похожести эмбеддингов.

На основании этой постановки, перечисленные в данном обзоре методы различаются а) выбором меры схожести двух вершин - структурной, по признакам, или одновременно б) способом отображения ии отображающей функцией.

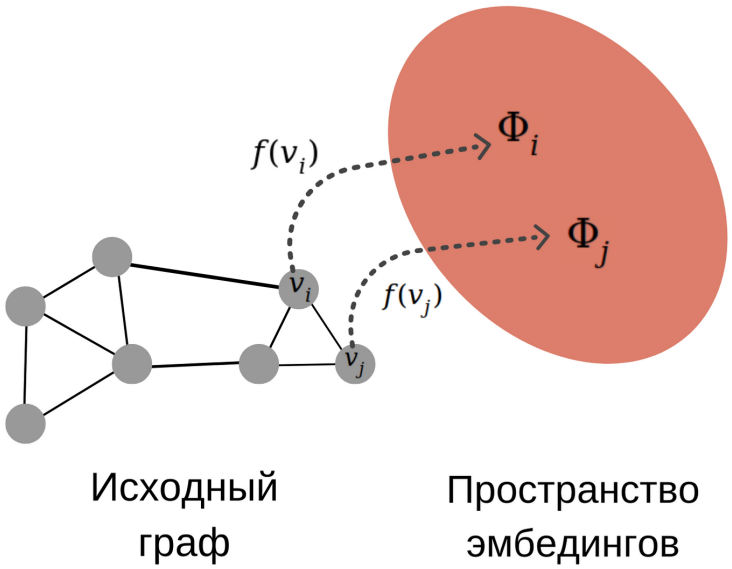

Рисунок 1

Обозначения

Граф представляется в виде упорядоченной пары $G = (V, E)$, где $V = \{v_i\}-$ непустое множество вершин, а $E = \{e_{ij}\}-$ множество пар веришин, называемых ребрами ($e_{ij}-$ это реберо между вершиными v_i и v_j). Порядок графа $|V|-$ число вершин в графе обозначается для краткости n. Аналогично, размер графа $|E|-$ число рёбер обозначим m. $A-$ матрица смежности, где каждое число $(a_{i,j})$ означает наличие $(a_{i,j} = 1)$ или отсутвие $(a_{i,j} = 0)$ ребра между вершинами v_i и v_j в случае не взвешенного графа и вес ребра в случае взвешенного. $W = \{w_{ij}\}-$ матрица весов в нейронной сети. Матрица $X-$ 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.

Φ_i, Θ_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. Вышеперечисленные обозначения собраны в таблице 1

Таблица 1: Необходимые обозначения

Обозначение	Расшифровка
$\|V\| = n$	Количество вершин в графе
$\|E\| = m$	Количество рёбер в графе
$\mathbf{A} = \{a_{ij}\}$	Матрица смежности
$\mathbf{I}$	Единичная матрица
$\mathbf{D}$	Матрица степеней вершин графа
$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$	Матрица переходов
$\mathbf{X} = \{x_{ij}\}, i \in \{1, .., n\}, j \in \{1, .., r\}$	Матрица признаков вершин
r	Размерность признакового пространства
$\mathbf{W}$	Матрица весов
$\mathcal{L}$	Функция потерь
Φ_i, Θ_i	Исходный и контекстный эмбединги вершины v_i
d	Размерность эмбедингов
$N(v)$	Соседи вершины v внутри окна определенного размера последовательности случайного блуждания

4 Функции потерь

5 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 проверят 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 2 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 предствляют 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 одно 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 наобором атрибутов. Метка - роль белка с точки зрения его клеточных функций.

Таблица 2: 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 соавторовств	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 3:

Таблица 3: Метрики качества.

Метрика	Формула, описание	Для каких задач
Micro-F1 score [1]	$\frac{2 \cdot P \cdot R}{P+R}$	Классификация вершин
Macro-F1 score	$\frac{\sum_{l \in L} F1(l)}{\lvert L \rvert}$, 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 $N^k - k$ ближаших 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 инфорация	Кластеризация вершин

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

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

6 Классификация и описание методов

Разделение на классы, использованное в данном обзоре взято из [2] и добавлены дополнительные методы. Данная классификация разделяет методы по подходу отображения вершин графа в эмбединги: методы, основанные на а) факторизации, б) случайных блужданиях, в) глубоком обучении и г) другие.

В данном разделе кратко описаны идеи всех рассматриваемых методов, а в таблице 4 собрана короткая справка по методам: на чем основаны методы, функции потерь, выбранные авторами оригинальных статей подходы к оптимизации и наличие/отсутствие разделения эбедингов на контекстный и целевой.

6.1 Методы, основанные на факторизации

Методы, основанные на факторизации рассматривают матрицы, отражающие соединения между вершинами графа: матрица смежности, Лапласиан графа, матрицу вероятностей перехода между вершинами. Затем эта матрицы факторизуется для получения эмбедингов.

- Метод Laplacian Eigenmaps [3] стремясь сохранить два эмбединга ближе в случае, когда в графе между соответствующими вершинами больше вес (в случае взвешенного графа), минимизирует следующую функцию потерь:

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

  где $\mathbf{L} = \mathbf{D} - \mathbf{A}$ – лапласиан графа.

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

- Graph Factorization [7] для быстрого разложения матрицы смежности на собственные векторы, использует стохастический градиентный спуск для минимизации следующей функции потерь:

$$\mathcal{L}(\Phi, \lambda) = \frac{1}{2}\sum_{i,j}(A_{i,j} - \Phi_i \cdot \Phi_j)^2 + \frac{\lambda}{2}\sum_i ||\Phi_i||^2 \tag{2}$$

  Преимущество - скорость.

- В методе HOPE [8] минимизируется $||\mathbf{C} - \Phi \cdot \Theta||_F^2$. В данном случае элементы матрицы c_{ij} отражают близость вершин v_i, v_j. C можно представить в виде $C = \mathbf{M}_g^{-1}\mathbf{M}_l$, а матрицы $\mathbf{M}_g, \mathbf{M}_l$ отличаются для каждой из мер схожести:

1. Katz Index:
$$\begin{aligned} \mathbf{M}_g &= \mathbf{I} - \beta \cdot \mathbf{A}, \\ \mathbf{M}_l &= \beta \cdot \mathbf{A}, \end{aligned} \tag{3}$$
где β – коэффициент затухания определяет как быстро затухает вес пути с возрастанием длины этого пути

2. Rooted PageRank
$$\begin{aligned} \mathbf{M}_g &= \mathbf{I} - \alpha \cdot \mathbf{P}, \\ \mathbf{M}_l &= (1 - \alpha) \cdot \mathbf{I}, \end{aligned} \tag{4}$$
где α – вероятность рандомного перехода к соседу.

3. Common Neighbors
$$\begin{aligned} \mathbf{M}_g &= \mathbf{I}, \\ \mathbf{M}_l &= \mathbf{A}^2, \end{aligned} \tag{5}$$

4. Adamic-Adar score
$$\begin{aligned} \mathbf{M}_g &= \mathbf{I}, \\ \mathbf{M}_l &= \mathbf{A} \cdot \mathbf{D} \cdot \mathbf{A}, \end{aligned} \tag{6}$$

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 синуглярного числа на соответвующий сингулярный вектор.

- Авторы статьи [25] 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 экивалентен факторизации следующей матрицы:
$$\log(\frac{vol(G)}{T}(\sum_{r=1}^{T}\mathbf{P}^r)\mathbf{D}^{-1}) - \log(b) \tag{7}$$
в данном случае b негативных примеров для Negative Sampling, vol(G) - объем взвешенного графа (сумма степеней всех вершин), T - длина случайного блуждания, r - размер окна в DeepWalk.

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

Таблица 4: Методы неконтролируемого обучения

Основан на	Метод	Контекстный эмбединг	Функция потерь	Подходы к оптимизации
факторизации	Laplacian EigenMaps	×	$\frac{1}{2}\sum_{i,j}\|\Phi_i - \Phi_j\|^2 A_{i,j}$ $= tr(\Phi^T L \Phi)$	Matrix Factorization
факторизации	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
факторизации	HOPE	✓	$\|\|\mathbf{C} - \Phi \cdot \Theta\|\|_F^2$, $C-$ матрица схожести вершин, см. уравнения 3, 5, 6	Matrix Factorization
факторизации	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 7)	Matrix Factorization + Negative Sampling
случайных блужданиях	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 1. В данном случае случайные блуждания фиксированной длины	Hierarchical softmax
случайных блужданиях	Node2Vec	✓	$-\sum_{v_j \in V}\sum_{v_i \in N(v_j)} log\frac{\exp(\Theta_i \cdot \Phi_j)}{\sum_{v_k \in V}\exp(\Theta_k \cdot \Phi_j)}$ Случаыные блуждания меют два параметра - вероятноность перехода к вершинам, отвечающим за исследования локальной и глобальной структур	Negative Sampling
случайных блужданиях	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 12,13), учитываемому структурную схожесть вершин	Hierarchical Softmax
случайных блужданиях	Seed	×	$\|\|X - \hat{X}\|\|_2^2$	Deep autoencoders

случайных блужданиях	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
случайных блужданиях	VERSE	×	$-\sum_{i,j=1}^{n} sim_G(v_i, v_j) \log \frac{\exp(\Phi_i \cdot \Phi_j)}{\sum_{k=1}^{n} \exp(\Phi_i \cdot \Phi_k)}$ sim_G – распределение схожести вершин графа. В случае PPR, $sim_G(v, \cdot)$ – последняя вершина в одном случайном блуждании начатого с вершины v, в случаях других мер схожести, определяется уравнениями 21, 22	Noise Constractive Estimations. Negative Sampling
глубоком обучении	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
глубоком обучении	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
моделировании соседства	LINE-1	×	$-\sum_{v_i, v_j \in V} a_{ij} \log \frac{1}{1+exp(-\Phi_i \cdot \Phi_j)}$	Negative Sampling
моделирование соседства	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

6.2 Методы, основанные на случайных блужданиях

Коротко, все последующие методы можно описать тремя шагами: а) использовать некоторую функцию f, для представления вершин графа в низко-размерном пространстве в виде векторов. Например, Φ_i, Φ_j для вершин v_i, v_j, б) Определить функцию схожести двух вершин $Similarity(v_i, v_j)$ в) Оптимизировать параметры функции f из первого шага, таким образом, чтобы $Similarity(v_i, v_j) \approx \Phi_i^T \Phi_j$.

Собственно, методы основанные на RandomWalk считают в качестве данной меры схожести - вероятность появления этих двух вершин во время случайного блуждания.

Итак, сначала запускаются случайные блуждания, строится множество соседей для каждой вершины и затем минимизируется:

$$\mathcal{L} = \sum_{v_j \in V} \sum_{v_i \in N(v_j)} -log(P(v_i|\Phi_j)) \tag{8}$$

- DeepWalk [11] использует случайные блуждания фиксированной длины.

- Node2vec [16] в отличие от DeepWalk исследует и локальную и глобальную структуру графа и с помощью двух гиперапараметров 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 [14] рассчитывает расстояние между вершинами по структурной схожести, а не топологической.

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

    $$f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u), s(R_k(v))) \tag{9}$$

    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. В оригинальной статье, описывающей данный метод, используется Dynamic Time Wraping (DTW), которое минимизирует расстояние между каждой парой элементов двух последоватльностей. Последнее считается следующим образом:

    $$d(a, b) = \frac{max(a, b)}{min(a, b)} - 1 \tag{10}$$

Хотя в принципе в данном методе можно использовать и другие способы нахождения расстояния между последовательностями степеней вершин.

2. После чего 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:
$$w_k(u, v) = \exp^{-f_k(u,v)} \tag{11}$$
Между слоями ребра соединяют лишь соотевтвующие 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:
$$p_k(u, v) = \frac{\exp^{-f_k(u,v)}}{\sum_{l \neq u, l \in V} \exp^{-f_k(u,l)}} \tag{12}$$
А вероятность перейти на следующий слой:
$$\begin{aligned} p_k(u_k, u_{k+1}) &= \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})}, \\ p_k(u_k, u_{k-1}) &= 1 - p_k(u_k, u_{k+1}) \end{aligned} \tag{13}$$
4. Далее, 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 последовталеьности случайного блуждания, построенного на предыдущем шаге)

- SEED [17] ориентирован 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 атрибуата этой вершины и наиболее раннего визита в ходе p-го блуждания). Обучение автоэнкодера происходит минимизируя ошибку реконструкции $\hat{X}$:
$$\mathcal{L} = ||X - \hat{X}||_2^2 \tag{14}$$
На последнем шаге усредненяются представления подграфов, или, если обобщать, то отображаются полученные представления Φ в некоторое новое пространство и

усредняют полученные представления:

$$\hat{\mu}_G = \frac{1}{s}\sum_{i=1}^{s}\phi(\Phi_i) \tag{15}$$

- APP [23] - сохраняет несимметричность, таким образом, что приписывает каждой вершине две роли - исходную и целевую (Φ_i, Θ_i). Используя метод сэмплировния Монте-Карло End-Point, рандомно выбирются пути, начинающиеся в одной вершине с вероятностью остановки α 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:

$$\mathcal{L} = \sum_j \sum_i \#Sampled_j(i) \cdot (\log\sigma(\Phi_j \cdot \Theta_i) + b \cdot E_{t_n \sim P_D}[\log\sigma(-\Phi_j \cdot \Theta_n)]) \tag{16}$$

  где $\#Sampled_j(i)-$ количество путей (j, i), b – количество негативных примеров, берущихся из распределения P_D, σ в анном слуачае сигмоида.

- TransE [18] приспособлен для графов знаний, у которых ребра также могут быть разных типов. Частая задача на таких графах - предсказание пропущенных связей. В TransE отношения между сущностями представлены в виде триплета:

$$h(\text{head entity}), l(\text{relation}), t(\text{tail entity}) \rightarrow (h, l, t) \tag{17}$$

  Затем, сущности переводятся одним из вышеперечисленных методов в простнаство эмбедингов.

  Далее отношения предсталяются как переносы (translations):

$$\begin{gathered}(h + l) \approx t, \text{данный факт - правда} \\ (h + l) \neq t, \text{иначе}\end{gathered} \tag{18}$$

  И в конце происходит оптимизация на основе Negative Sampling.

- VERSE [9] так же использует понятия схожести между вершинами и минимизируют расхождение Кульбака - Лейблера (KL) между распределением схожести вершин в графе sim_G и распределением схожести эмбедингов sim_E:

$$\sum_{v \in V} KL(sim_G(v, \cdot)||sim_E(v, \cdot)) \tag{19}$$

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

$$sim_E(v, \cdot) = \frac{\exp \Phi_v \Phi^T}{\sum_{i=1}^{n} \exp(\Phi_v \Phi_i)} \tag{20}$$

В данном выше уравнении $\Phi-$ матрица эмбеддингов, в которой отдельный ряд Φ_v соответствует эмбеддингу вершины v.

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

1. Personalized PageRank: один экзамепляр $sim_G(v, \cdot)$ это последняя вершина в одом случайном блуждании, начатом с вершины v.
2. SimRank: мера структорной взаимосвязи двух вершин, основана на предположении, что схожие вершины связаны с дргуими схожими вершинами. Определяется рекурсивно:

$$sim_G^{SR} = \frac{C}{|I(u)||I(v)|} \sum_{i=1}^{|I(u)|} \sum_{j=1}^{|I(v)|} sim_G^{SR}(I_i(u), I_j(v)) \tag{21}$$

   $I(v)$ – 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

$$sim_G^{ADJ}(u, v) = \begin{cases} 1/Out(u) & \text{если}(u, v) \in E \\ 0 & \text{иначе} \end{cases} \tag{22}$$

Рассмотрим подробнее функцию потерь:

По определению расстояния Кульбака-Лейбнера для дискретных распределений:

$$KL(P||Q) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} \tag{23}$$

В случае с распределениями схожести в графе и прстранстве эмбеддингов, получаем:

$$KL(sim_G(v, \cdot)||sim_E(v, \cdot)) = \sum_{i=1}^{n} sim_G(v, \cdot) \log \frac{sim_G(v, i)}{sim_E(v, i)} =$$
$$= \sum_{i=1}^{n} sim_G(v, i) \log sim_G(v, i) - \sum_{i=1}^{n} sim_G(v, i) \log sim_E(v, i) \tag{24}$$

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:

$$\mathcal{L} = -\sum_{v \in V} sim_G(v, \cdot) \log \frac{\exp \Phi_v \Phi^T}{\sum_{i=1}^{n} \exp(\Phi_v \Phi_i)} \tag{25}$$

Для оптимизации используется Noise Contrastive Estimation (NCE): алгоритм строит классификатор (логистическую регрессию), который разделяет вершины из настоящего распределения $sim_G(v, \cdot)$ и распределения шума Q. Производная от NCE с увеличением негативных примеров, сходится градиенту функции (25).

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

Преимуществом данных методов являются: а) вычислительная эффективность, так как они не требуют рассматрения абсолютно всех пар вершин, а только те, что появляются в одном случайном блуждании; б) возможность учитывают как локальную так и глобальную структуру графа.

6.3 Методы, осованные на глубоком обучении

Общая идея данных методов - агрегировать информацию от соседних вершин и передавать на следующий слой, выстраивая таким образом полноценную нейронную сеть. (см. Рисунок 2). Довольно распространены и популярны среди методов глубокого обучения на графах методы GCN (Graph Convolutional Network) [26] и GAT (Graph Attention Network) [20]. 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 их можно модифицировать, беря только части, касающиеся неконтролируемого этапа:

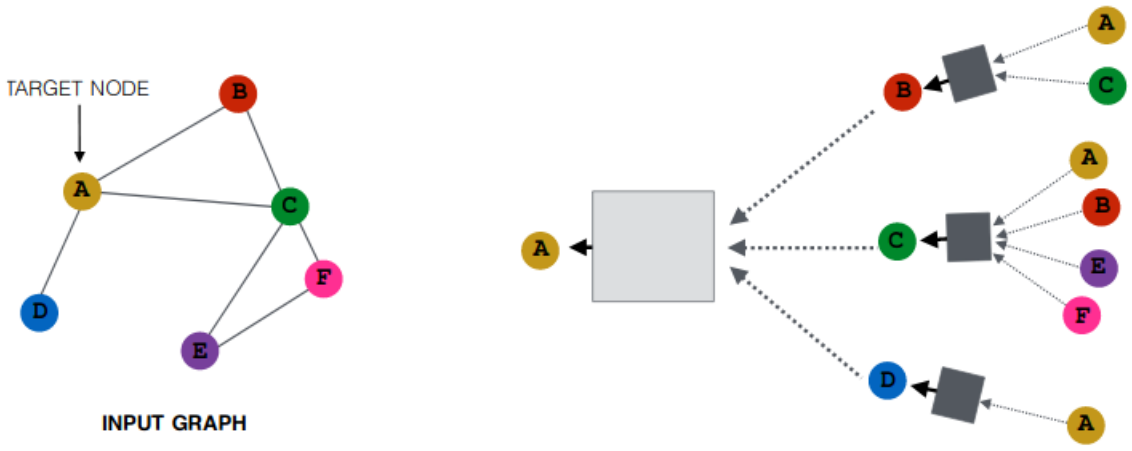

Рисунок 2

- GCN [26] - на каждом новом слое строится представление вершины путем агрегирования информации от соседей на прошлом слое и суммируется с представлением самой вершины на прошлом слое. Веса тренируются с помощью SGD:

$$\begin{aligned} \mathbf{h}_v^0 &= \mathbf{x}_v \\ \mathbf{h}_v^k &= \sigma(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}), \forall k \in \{1, ..., K\} \\ \Phi_v &= \mathbf{h}_v^K \end{aligned} \tag{26}$$

  Где $\mathbf{h}_v^k-$ представление вершины v на $k-$ом слое.

- GAT [20] - Идея в том, что соседи оказывают разное влияние на данную вершину и необходимо это учесть:

$$\mathbf{h}_v^k = \sigma(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}_k \mathbf{h}_u^{k-1}) \tag{27}$$

  Существует эффективная модификация GCN, метод GraphSAGE, который работает полностью в неконтролируемом режиме и подходит для больших графов.

- CraphSAGE [19] В отличие от предыдущего метода, может использовать разные варианты аггрегирования информации от соседей кроме среднего значения, например, выбор максимального значения или применение LSTM. Кроме того, еще одно

отличие от базового варианта это конкатеринирование представления самой вершины с прошлого слоя к агрегированной информации от соседних вершин, вместо суммирования с ней. Преимущество над предыдущим методом - это обобщение:

$$\mathbf{h}_v^k = \sigma([\mathbf{W}_k \cdot AGG(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}), \mathbf{B}_k \mathbf{h}_v^{k-1}]) \tag{28}$$

Для настройки весов минимизируется функция потерь, которая может иметь как и контролируемый вид (cross-entropy), так и неконтролирумый, аналогичный DeepWalk:

$$-\sum_{v_j \in V} \sum_{v_i \in N(v_j)} \log \sigma(\Phi_i \cdot \Phi_j) \tag{29}$$

- SDNE [24] состоит из двух шагов: 1) Неконтролируемая часть, данный автоэнкодер отвечает за построение эмбедингов, учитывая близость 2 порядка. 2) Контролируемая часть, строится такая функция потерь, чтоб накладывать штрафы, когда похожие вершины в пространстве эмбедингов находятся далеко друг от друга, т.е. учитывать близость вершин 1 порядка с помощью матрицы смежности.

  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: $p_u = (a_{u,1}, ..., a_{u,|V|})-$ определяет близость первого порядка для вершины u ко всем остальным. Тогда близость второго порядка между вершинами u, v определяется сходестью между p_v, p_u

  Глубокий автоэнкодер коротко можно описать следующим образом. Это неконтролируемая модель, состоящая из двух частей - энкодер и декодер g. Энкодер состоит из нескольких нелинейных функций, которые отображают входные данные (вершины) в скрытые представления (в конечном счете - в эмбединги). Декодер тоже состоит из нескольких нелинейных функций, и наоборот принимая на вход эмбединги, реконструирует граф.

  Имея в качестве входа $\mathbf{a}_i$ (колонки матрицы смежности) скрытые представления каждого слоя записываются следующим образом:

$$\begin{aligned} \mathbf{y}_i^{(1)} &= \sigma(W^{(1)}\mathbf{a}_i + \mathbf{b}^{(1)}), \\ \mathbf{y}_i^{(k)} &= \sigma(W^{(k)}\mathbf{y}_i^{(k-1)} + \mathbf{b}^{(k)}), k = 2, ..., K \end{aligned} \tag{30}$$

  $y_i^{(K)} = \Phi_i$ – и есть эмбединг вершины v_i. После получения всех Φ_i , можно получить реконструкцию $\hat{\mathbf{a}}_i$ обращая процесс энкодера.

  В качестве ошибки реконструкции в оригинальной статье используются б**о**льшие штрафы для не нулевых элементов, чем для нулевых добавлением умножения на

смещения $\mathbf{b}_i$ (т.е. ненулевые элементы должны иметь меньшую ошибку реконструкции, чем нулевые):

$$\mathcal{L}_{2nd} = ||(\hat{A} - A) \odot B||_F^2 \tag{31}$$

где $\odot$ – произведение Адамара, B–матрицы смещений (biases). g– декодер.

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

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} a_{i,j} ||\Phi_i - \Phi_j||_2^2 \tag{32}$$

Общая функция потерь выглядит следующим образом:

$$\mathcal{L}_{mix} = \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st} = \\ = ||(\hat{A} - A) \odot B||_F^2 + \alpha \sum_{i,j=1}^{n} a_{i,j} ||\Phi_i - \Phi_j||_2^2 \tag{33}$$

α - параметр, контролирующий отношение степеней важности локальной структуры к глобальной.

6.4 Другие методы

- LINE [15] 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/эмбедингов, одно - используя матрицу смежности, другое - для эмбедингов. Например для случая близости 1 порядка:

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

$$\hat{p_1}(v_i, v_j) = \frac{a_{ij}}{\sum_{i,j \in V} a_{ij}} \tag{35}$$

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

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

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

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

В случае сохранения близости второго порядка:

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

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

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

Минимизируется следующая функция:

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

λ 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 λ_i был положен как d_i. Тогда, как и в случае LINE-1, не учитывая константы, фукнция минимизации выглядит следующим образом:

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

Выводы:

- Традиционные методы построения эмбедингов (снованные 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 быстрей, так как используют для вычислений не все пары вешин.
- 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.

7 Рекомендации по использованию методов

В работах [22] и [2] 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 поверхностыми.

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 аггрегации подходят для высокой степени гомофилии, а для низкой - DeepWalk.

9. Node2Vec более предпочтителен для класссификации вершин, а для предсказания рёбер - методы, учитывающие разные степени схожести (HOPE,SDNE)

8 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 2, 3

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] Z. Wu, S. Pan, F. Chen, G. Long,C. Zhang, S. Yu Philip. 2019. A comprehensive survey on graph neural networks. arXiv:1901.00596.

[2] 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.

[3] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In NIPS. 585–591

[4] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. science 290, 5500 (2000), 2323–2326.

[5] 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

[6] 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)

[7] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In WWW. ACM, 37–48.

[8] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In KDD. 1105–1114

[9] 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.

[10] G. Rizos, S. Papadopoulos, and Y. Kompatsiaris, Multilabel user classification using the community structure of online networks, vol. 12, no. 3. 2017.

[11] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in KDD, 2014, pp. 701–710.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations ofWords and Phrases and their Composition- ality. In NIPS. 3111–3119

[13] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In CIKM. 891–900

[14] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In KDD. ACM, 385–394

[15] 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.

[16] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in KDD, 2016, pp. 855–864.

[17] L. Wang et al., "INDUCTIVE AND UNSUPERVISED REPRESENTATION LEARNING ON GRAPH STRUCTURED OBJECTS," 2016.

[18] A. Bordes, N. Usunier, A. Garcia-dur, J. Weston, and O. Yakhnenko, "Translating Embeddings for Modeling Multi-relational Data," pp. 1–9.

[19] 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.

[20] 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.

[21] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Pro-ceedings of the 22nd International Conference on Knowledge Discoveryand Data Mining, ACM, 2016, pp. 1225–1234.

[22] 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.

[23] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao. Scalable graph embedding for asymmetric proximity. In AAAI, pages 2942–2948, 2017.

[24] D.Wang, P. Cui, andW. Zhu. Structural deep network embedding. In SIGKDD, pages 1225–1234. ACM, 2016.

[25] 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. 21. 56.

[26] 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.