3. ЛЕКЦИЯ – Встраивание узлов

3.1. Методы реконструкции окрестностей

Рассмотрим методы обучения встраивания узлов, которые могут зашифровать узлы как маломерные вектора. При этом суммируется их позиция и локальных соседей согласно структуре графа. Другими словами, необходимо спроецировать узлы в скрытое пространство, где геометрические отношения в этом скрытом пространстве соответствуют отношениям (например, ребрам) в исходном графе или сети (рис. 3.1) .

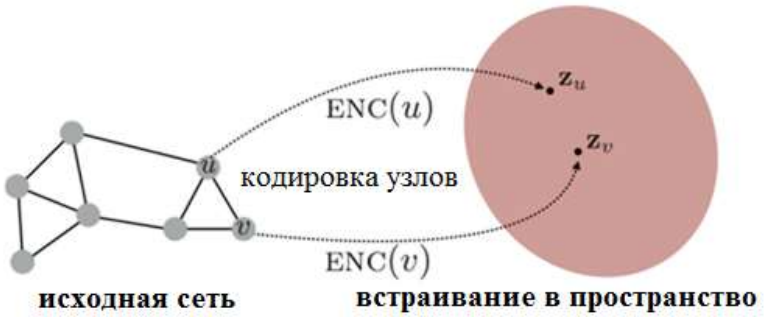

Рис. 3.1. Иллюстрация проблемы встраивания узлов.

Таким образом, необходимо рассмотреть кодировщик, который отображает узлы в маломерном пространстве встраивания. Данное встраивание в пространство должно оптимизировано таким образом, чтобы расстояния в нем отражали относительное положение узлов в исходном графе.

Перспектива кодер-декодер.

В рамках кодера-декодера рассматривается проблема обучения представления графа, включающую две ключевые операции. Во-первых, модель кодера отображает каждый узел в графе в маломерный вектор или вложение. Далее, декодер модель берет маломерные вложения узлов и использует их для восстановления информации об окрестности каждого узла в исходном графе. Эта идея кратко изложена на рисунке 3.2.

Кодер. Формально кодировщик – это функция, которая сопоставляет узлы $v \in V$ с векторными вложениями $z_v \in \mathbb{R}^d$ (где z_v соответствует вложению для узла $v \in V$). В простейшем случае кодер имеет следующую сигнатуру (3.1) :

$$ENC: V \to \mathbb{R}^d \qquad (3.1)$$

это означает, что кодер принимает идентификаторы узлов в качестве входных данных для генерации вложений узлов. В большинстве работ по встраиванию узлов кодер определяется подходом неглубокого встраивания, где эта функция кодировщика представляет собой просто поиск встраивания на основе идентификатора узла. Другими словами, мы имеем, что (3.2).

$$ENC(v) = Z_{[v]} \quad (3.2)$$

где $Z \in \mathbb{R}^{|V| \times d} R$ матрица, содержащая векторы вложения для всех узлов, а $Z_{[v]}$ обозначает строку Z, соответствующую узлу v.

Рис. 3.2: Обзор подхода кодер-декодер

Кодер (рис. 3.2) сопоставляет узел u с маломерным вложением z_u. Затем декодер использует z_u для восстановления информации о локальном окружении u.

Однако необходимо отметить, что кодировщик также может быть обобщен за пределы подхода неглубокого вложения. Например, кодировщик может использовать объекты узла или локальную структуру графа вокруг каждого узла в качестве входных данных для создания встраивания. Эти обобщенные архитектуры кодировщиков, часто называемые графовыми нейронными сетями (graph neural network – GNN).

Декодер. Роль декодера заключается в восстановлении определенной статистики графа из вложений узлов, которые генерируются кодером. Например, учитывая встраивание z_u узла u, декодер может попытаться предсказать набор соседей $N(u)$ или его строку $A[u]$ в матрице смежности графа.

Хотя возможно множество декодеров, стандартной практикой является определение попарных декодеров, которые имеют следующую сигнатуру (3.3):

$$DEC: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+ \quad (3.3)$$

Попарные декодеры могут быть интерпретированы как предсказывающие взаимосвязь или сходство между парами узлов. Например, простой попарный декодер может предсказать, являются ли два узла соседями в графе.

Применение попарного декодера к паре вложений (z_u, z_v) приводит к восстановлению взаимосвязи между узлами u и v. Цель состоит в том, чтобы оптимизировать кодер и декодер для минимизации потерь при восстановлении, чтобы (3.4).

$$DEC(ENC(u), ENC(v)) = DEC(z_u, z_v) \approx S[u, v] \qquad (3.4)$$

Здесь предполагается, что $S[u, v]$ является мерой сходства между узлами на основе графа. Например, простая задача реконструкции предсказания того, являются ли два узла соседями, будет соответствовать S [u, v], $A[u, v]$. Однако можно определить $S[u, v]$ и более общими способами, например, используя любую статистику попарного перекрытия окрестностей, рассмотренную в разделе 2.2.

Оптимизация модели кодер-декодер. Для достижения цели реконструкции (уравнение 3.4) стандартной практикой является минимизировать эмпирическую потерю реконструкции L по набору пар обучающих узлов D (3.5):

$$L = \sum_{(u,v)\in D} l(DEC(z_u, z_v), S[u, v]) \qquad (3.5)$$

где $L: R \times R \to R$ – функция потерь, измеряющая расхождение между декодированными (т.е. оцененными) значениями подобия $DEC(z_u, z_v)$ и истинными значениями $S[u, v]$. В зависимости от определения декодера (DEC) и функции подобия (S), функция потерь может быть среднеквадратичной ошибкой или даже потерей классификации, такой как перекрестная энтропия. Таким образом, общая цель состоит в том, чтобы обучить кодер и декодер, так что попарные узловые связи могут быть эффективно восстановлены на обучающем наборе D. Большинство подходов сводят к минимуму потери в уравнении 3.5 с использованием стохастического градиентного спуска, но есть определенные случаи, когда можно использовать более специализированные методы оптимизации (например, основанные на факторизации матрицы).

Обзор подходов кодер-декодер. Рассмотрим перспективу методов кодирования-декодирования, которые применяются для обобщения нескольких хорошо известных методов встраивания узлов, все из которых используют подход неглубокого кодирования (табл. 3.1). Ключевое преимущество фреймворка кодер-декодер заключается в том, что он позволяет кратко определять и сравнивать различные

методы встраивания на основе их функции декодера, их меры подобия на основе графов и их функции потерь.

Таблица 3.1. Краткое описание некоторых хорошо известных алгоритмов неглубокого встраивания.

Модель	Декодер	Мера подобия	Функция потерь
Lap. Eigenmaps	$\|z_u, z_v\|_2^2$	общая	$DEC(z_u, z_v) * S[u, v]$
Graph Fact.	${z_u}^T z_v$	$A[u, v]$	$\|DEC(z_u, z_v) - S[u, v]\|_2^2$
GraRep	${z_u}^T z_v$	$A[u, v], \dots, A^k[u, v]$	$\|DEC(z_u, z_v) - S[u, v]\|_2^2$
HOPE	${z_u}^T z_v$	общая	$\|DEC(z_u, z_v) - S[u, v]\|_2^2$
DeepWalk	$\frac{e^{{z_u}^T z_v}}{\sum_{k \in V} e^{{z_u}^T z_k}}$	$P_G(v\|u)$	$-S[u, v] \log\big(DEC(z_u, z_v)\big)$
node2vec	$\frac{e^{{z_u}^T z_v}}{\sum_{k \in V} e^{{z_u}^T z_k}}$	$P_G(v\|u)$ необъективно	$-S[u, v] \log\big(DEC(z_u, z_v)\big)$

Примечание что декодеры и функции подобия для методов, основанных на случайном блуждании, являются асимметричными, причем функция подобия $P_G(v|u)$ соответствует вероятности посещения v при случайном блуждании фиксированной длины, начиная с u.

3.2. Подходы, основанные на факторизации

Один из способов рассмотрения идеи кодера-декодера – это с точки зрения факторизации матрицы. Действительно, задача декодирования локальной структуры окрестностей из вложения узла тесно связана с восстановлением записей в матрице смежности графа. В более общем плане можно рассматривать эту задачу как использование факторизации матрицы для изучения маломерной аппроксимации подобия узел-узел матрица S, где обобщается матрица смежности и фиксируется некоторое определенное пользователем понятие сходства узлов.

Собственные карты Лапласа. Одним из самых ранних и наиболее влиятельных подходов, основанных на факторизации, является метод собственных карт Лапласа (Laplacian Eigenmaps), который основан на идеях спектральной кластеризации, рассмотренных в главе 2. В этом подходе определяется декодер на основе L_2-расстояния между вложениями узлов. Затем функция потерь взвешивает пары узлов в соответствии с их сходством на графике (3.6) [3]:

$$DEC(z_u, z_v) = \|z_u - z_v\|_2^2 \qquad L = \sum_{(u,v) \in D} DEC(z_u, z_v) * S[u, v] \qquad (3.6)$$

Структура, лежащая в основе этого подхода, заключается в том, что уравнение (3.6) наказывает модель, когда очень похожие узлы имеют вложения, которые находятся далеко друг от друга.

Если S построено так, что оно удовлетворяет свойствам матрицы Лапласа, то вложения узлов, которые минимизируют потери в уравнении (3.6), идентичны решению для спектральной кластеризации, которое мы обсуждали в разделе 2.3. В частности, если предположим, что вложения z_u являются d-мерными, то оптимальное решение, минимизирующее уравнение (3.6), задается d наименьшими собственными векторами лапласиана (исключая собственный вектор всех единиц).

Методы внутреннего произведения. Они основаны на методе собственных карт Лапласа, в более поздних работах обычно используется декодер на основе внутреннего продукта, определяемый следующим образом (3.7):

$$DEC(z_u, z_v) = {z_u}^T z_v \tag{3.7}$$

Здесь предполагается, что сходство между двумя узлами – например, перекрытие между их локальными окрестностями – пропорционально точечному произведению их вложений. Некоторые примеры этого стиля алгоритмов встраивания узлов включают: факторизацию графа (Graph Factorization – GF), GraRep и HOPE. Все эти три метода объединяет внутренний декодер (уравнение 3.7) со следующей среднеквадратичной ошибкой (3.8):

$$L = \sum_{(u,v)\in D} \|DEC(z_u, z_v) - S[u, v]\|_2^2 \tag{3.8}$$

Методы отличаются прежде всего тем, как они определяют $S[u, v]$, т.е. понятием сходства между узлами или перекрытием окрестностей, которые они используют. В то время как подход факторизации графа (GF) непосредственно использует матрицу смежности и устанавливает S, A, в подходах GraRep и HOPE, которые используют более общие стратегии. В частности, GraRep определяет S на основе степеней матрицы смежности, в то время как алгоритм HOPE поддерживает общие меры перекрытия окрестностей (например, любую меру перекрытия окрестностей из раздела 2.2).

Эти методы называются подходами матричной факторизации, поскольку их функции потерь могут быть минимизированы с помощью алгоритмов факторизации, таких как разложение по сингулярным значениям (SVD). Действительно, пу-

тем укладки вложений узлов $z_u \in \mathbb{R}^d$ в матрицу $Z \in \mathbb{R}^{|V|*d}$ цель реконструкции для этих подходов может быть записана как (3.9):

$$L \approx \|\mathrm{ZZ}^T - \mathrm{S}\|_2^2 \tag{3.9}$$

что соответствует маломерной факторизации матрицы подобия узел-узел S. Интуитивно понятно, что целью этих методов является изучение вложений для каждый узел таков, что внутреннее произведение между изученными векторами вложения аппроксимирует некоторую детерминированную меру сходства узлов.

3.3. Вложения случайного блуждания

Из предыдущего раздела очевидно, все методы внутреннего произведения используют детерминированные меры сходства узлов. Они часто определяют S как некоторую полиномиальную функцию матрицы смежности, а вложения узлов оптимизируются таким образом, чтобы ${z_u}^T z_v \approx S[u, v]$. Основываясь на этих успехах, в последние годы наблюдается всплеск успешных методов, которые адаптируют подход внутреннего продукта (произведение) к использованию стохастических меры перекрытия окрестностей. Ключевым нововведением в этих подходах является то, что вложения узлов оптимизированы таким образом, чтобы два узла имели похожие вложения, если они имеют тенденцию встречаться при коротких случайных блужданиях по графу.

DeepWalk и node2vec. Подобно подходам к факторизации матриц, DeepWalk и node2vec используют подход неглубокого встраивания и декодер внутреннего продукта. Ключевое различие в этих методах заключается в том, как они определяют понятия подобия узлов и реконструкции окрестностей. Вместо прямого восстановления матрицы смежности A (или некоторой детерминированной функции A) эти подходы оптимизируют вложения для кодирования статистики случайных блужданий. Математически цель состоит в том, чтобы изучить вложения так, чтобы выполнялось следующее (приблизительно) (3.10) :

$$DEC(z_u, z_v) \triangleq \frac{e^{{z_u}^T z_v}}{\sum_{k \in V} e^{{z_u}^T z_k}} \approx P_{G,T}(v|u) \tag{3.10}$$

где $P_{G,T}(v|u)$ – вероятность посещения v при случайном блуждании длиной T, начинающемся с u, при этом T обычно определяется как находящийся в диапа-

зоне $T \in \{2, \ldots, 10\}$. Опять же, ключевое различие между уравнением (3.10) и подходами, основанными на факторизации (например, уравнение 3.8), заключается в том, что мера подобия в уравнении (3.10) является как стохастический и асимметричный.

Для обучения вложений случайного блуждания общая стратегия заключается в использовании декодера из уравнения (3.10) и минимизации следующих потерь перекрестной энтропии (3.11):

$$L = \sum_{(u,v)\in D} -\log\big(DEC(z_u, z_v)\big) \tag{3.11}$$

Здесь используется D для обозначения обучающего набора случайных блужданий, который генерируется путем выборки случайных блужданий, стартом которых является каждая вершина. Например, можно предположить, что, при N пар совпадающих вершин, для каждой вершины U выбираются из распределения $(u, v) \sim P_{G,T}(v|u)$

Однако, к сожалению, неточная оценка потерь в уравнении (3.11) может быть вычислительно затратным. Действительно, знаменатель в уравнении (3.10) имеет временную сложность $O(|V|)$, что делает общую временную сложность вычисления функции потерь равной $O(|D||V|)$. Существуют разные стратегии чтобы преодолеть эту вычислительную проблему, и это одно из существенных различий между оригинальными алгоритмами DeepWalk и node2vec. DeepWalk использует иерархический softmax (hierarchical softmax) для аппроксимации уравнения (3.10), которое включает использование структуры бинарного дерева для ускорения вычислений. С другой стороны, node2vec использует метод сравнения с шумом (noise contrastive) для аппроксимации уравнения (3.11), где нормализующий коэффициент аппроксимируется с помощью отрицательных образцов (negative samples) (3.12):

$$L = \sum_{(u,v)\in D} -\log\big(\sigma({z_u}^T, z_v)\big) - \gamma \mathbb{E}_{v_n \sim P_n(V)}\big[\log\big(-\sigma({z_u}^T, z_v)\big)\big] \tag{3.12}$$

Здесь используется σ для обозначения логистической функции, $P_n(V)$ для обозначения распределения по множеству вершин V, и предполагается, что $\gamma > 0$ является гиперпараметром. На практике $P_n(V)$ часто определяется как равномерное распределение, и ожидание аппроксимируется с использованием выборки Монте-Карло.

Подход node2vec также отличается от более раннего DeepWalk. алгоритм,

позволяя более точно определять случайные блуждания. В частности, тогда как DeepWalk просто использует равномерно случайные блуждания для определения $P_{G,T}(v|u)$, подход node2vec вводит гиперпараметры, которые позволяют вероятности случайных блужданий плавнее интерполироваться между блужданиями, тогда они похожи на поиск в ширину или поиск в глубину по графу.

Крупномасштабные вложения информационных сетей (LINE – Large-scale information network embeddings). В дополнение к DeepWalk и node2vec, алгоритм LINE часто обсуждается как один из подходов случайного блуждания. Подход LINE явно использует не случайные блуждания, но разделяет концептуальные мотивы с DeepWalk и node2vec. Основная идея LINE состоит в том, чтобы объединить два объекта кодера-декодера. Первый объект направлен на кодирование смежности первого порядка. информации и использует следующий декодер (3.13):

$$EC(z_u, z_v) = \frac{1}{1+e^{-z_u{}^T z_v}} \tag{3.13}$$

с мерой сходства на основе смежности (т. е. $S[u, v] = A[u, v]$). Второй объект больше с подходом случайного блуждания. Это тот же декодер (3.10), но он обучается с использованием KL-дивергенции для кодирования информация о смежности (т. е. информация в A^2). Таким образом, LINE концептуально связан с node2vec и DeepWalk. Он использует вероятностный декодер и вероятностную функцию потерь (на основе KL-дивергенции). Однако вместо выборки случайных блужданий он явно восстанавливает информацию о окрестностях первого и второго порядка.

Дополнительные варианты идеи случайного блуждания. Одно из преимуществ случайного блуждания заключается в том, что его можно расширить и изменить, сместив или изменив случайные блуждания. Например, описываются случайные блуждания, которые «пропускают» вершины, такой подход похож на GraRep и определяет случайные блуждания на основе структурных отношений между вершинами (опуская информацию о соседстве) генерируют вложения (embeddings) вершин, кодируют структурные роли в графе.

Методы случайных блужданий и матричная факторизация. Легко заметить, что методы случайных блужданий на самом деле тесно связаны с подходами матричной факторизации. Предположим, что мы определяем матрицу значений сходства между вершинами (3.14):

$$S_{DW} = \log\left(\frac{vol(v)}{T}(\sum_{t=1}^{T} P^t)D^{-1}\right) - \log(b) \quad (3.14)$$

где b – константа и $P = D^{-1}A$. В этом случае вложения Z, изученные DeepWalk, удовлетворяют (3.15):

$$Z^T Z \approx S_{DW} \quad (3.15)$$

Интересно, что мы также можем разложить внутреннюю часть уравнения (3.16) как:

$$(\sum_{t=1}^{T} P^t)D^{-1} = D^{-\frac{1}{2}}(U(\sum_{t=1}^{T} \Lambda^t)U^T)D^{-\frac{1}{2}} \quad (3.16)$$

где $U\Lambda U^T = L_{sym}$ — собственное разложение симметричного нормированного Лапласиан. Это показывает, что вложения, изученные DeepWalk, на самом деле тесно связаны с вложениями спектральной кластеризации. Ключевое отличие состоит в том, что вложения DeepWalk контролируют влияние различных собственных значений через T, т. е. длину случайного блуждания. Проводятся аналогичные связи с матричной факторизацией и для node2vec и обсуждаются другие родственные подходы на основе факторизации, вдохновленные этой связью.

3.4. Ограничения неглубоких вложений

В подходах неглубокого вложения модель кодировщика, которая сопоставляет вершины с вложениями, представляет собой просто поиск вложений. Уравнение 3.2 обучает уникальное вложение для каждой вершины в графе. Этот подход добился многих успехов за последнее время. Однако необходимо отметить, что подходы к неглубокому вложению имеют ряд существенных недостатков:

1. Первая проблема заключается в том, что методы неглубокого вложения не передают какие-либо параметры между вершинами в кодировщике, поскольку кодировщик напрямую оптимизирует уникальный вектор вложения для каждой вершины. Такое отсутствие совместного использования параметров неэффективно как статистически, так и вычислительно. Со статистической точки зрения совместное использование параметров может повысить эффективность обучения, а также выступать в качестве мощной формы регуляризации. С вычислительной точки зрения отсутствие совместного использования параметров означает, что количество параметров в методах неглубокого вложения неизбежно растет как временную сложность O(|V|). Это может быть неразрешимы в массивных графах.

2. Вторая ключевая проблема неглубокого вложения заключается в том, что

оно не использует функции вершины в кодировщике. Многие наборы графических данных содержат обширную информацию о функциях, которая потенциально может быть информативной в процессе кодирования.

3. И, наконец, возможно, самое главное, методы поверхностного встраивания по своей сути являются трансдуктивными. Эти методы могут только генерировать вложения для вершин, которые присутствовали на этапе обучения. Генерация вложений для новых вершин, наблюдаемых после этапа обучения, невозможна, если не будут выполнены дополнительные оптимизации для изучения вложений для этих вершин. Это ограничение предотвращает использование неглубоких методов встраивания в индуктивных приложениях, которые включают обобщение на невидимые вершины после обучения.

Чтобы снять эти ограничения, неглубокие кодировщики можно заменить более сложными кодировщиками, которые в большей степени зависят от структуры и атрибутов графа.