8. ЛЕКЦИЯ - Традиционные подходы к созданию графов

И так было рассмотрено множество методов изучения представлений графов. Теперб обсудим отдельную, но тесно связанную с ней задачу: проблему генерации графов.

Целью генерации графов является создание моделей, которые могут генерировать реалистичные структуры графов. В некотором смысле можно рассматривать эту проблему как зеркальное отражение проблемы вложения графов. Вместо того чтобы предполагать, что в качестве входных данных для модели задана структура графа $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, при генерации графа желательно, чтобы на выходе модели был граф $\mathcal{G}$. Конечно, простое создание произвольного графа не обязательно сложно. Например, легко сгенерировать полносвязный граф или граф без ребер. Однако ключевой проблемой при создании графов является создание графов, обладающих определенными желаемыми свойствами. Способ, который определяет «желательные свойства» – и то, как выполняется генерация графа – сильно различается в зависимости от разных подходов.

Поэтому начнем с обсуждения традиционных подходов к построению графов. Эти подходы предшествуют большинству исследований в области обучения представления графов и даже исследованиям машинного обучения в целом. Таким образом, методы, которые будут обсуждаться в этой главе, обеспечивают основу для мотивации подходов, основанных на глубоком обучении.

8.1 Обзор традиционных подходов

Традиционные подходы к созданию графов обычно включают указание некоторого генеративного процесса, который определяет, как создаются ребра в графе. В большинстве случаев можно сформулировать этот порождающий процесс как способ определения вероятности или правдоподобия $P(A[u, v] = 1)$ ребра, существующего между двумя вершинами u и v. Задача состоит в том, чтобы разработать какой-то генеративный процесс, который одновременно податлив и способен генерировать графы с нетривиальными свойствами или характеристиками. Управляемость важна, потому что хочется иметь возможность выбирать или анализировать сгенерированные графы. Однако также хотчется, чтобы эти графы обладали некоторыми свойствами, делающими их эффективными моделями для графов, которые видим в реальном мире.

Три подхода, которые будут рассматриваться, представляют собой небольшое, но репрезентативное подмножество традиционных подходов к построению графов, которые существуют в литературе.

8.2 Модель Эрдёша-Реньи

Возможно, самой простой и наиболее известной генеративной моделью графов является модель Эрдёша-Реньи (Erdos-Renyi – ER). В этой модели определяется вероятность возникновения ребра между любой парой вершин как,

$$P(A[u,v]=1)=r,,\forall u,v\in V,u\neq V \qquad (8.1)$$

где $r\in[0,1]$ – параметр, управляющий плотностью графа. Другими словами, модель ER просто предполагает, что вероятность возникновения ребра между любыми парами вершин равна r.

Модель ER привлекательна своей простотой. Чтобы сгенерировать случайный граф ER, просто выбираем (или подбираем), сколько вершин мы хотим, устанавливаем параметр плотности r, а затем используем уравнение (8.1) для создания матрицы смежности. Поскольку все вероятности ребер независимы, временная сложность построения графа составляет $O(|V|^2)$, т. е. линейна по размеру матрицы смежности.

Однако недостатком модели ER является то, что она не создает очень реалистичных графов. В частности, единственным свойством, которым можно управлять в ER-модели, является плотность графа, поскольку параметр r равен средней степени в графе. Другие свойства графа, такие как распределение степеней, наличие структур сообщества, коэффициенты кластеризации вершин и наличие структурных мотивов, не учитываются моделью ER. Хорошо известно, что графы, сгенерированные ER-моделью, не отражают распределения этих более сложных свойств, которые, как известно, важны для структуры и функционирования графов реального мира.

8.3 Стохастические блочные модели

Многие традиционные подходы к созданию графов стремятся улучшить модель ER за счет лучшего учета дополнительных свойств графов реального мира, которые модель ER игнорирует. Одним из ярких примеров является класс стохастических блочных моделей (Stochastic Block Models – SBM), которые стремятся генерировать графы со структурой сообщества.

В базовой модели SBM задается γ различных блоков: $C_1,\ldots,C_\gamma$. Тогда каждая вершина $u\in V$ имеет вероятность p_i принадлежности к блоку i, т. е. $p_i=P(u\in C_i),\forall u\in V,i=1,\ldots,\gamma$, где $\sum_{i=1}^{\gamma}p_i=1$. Затем вероятности ребер задаются матрицей вероятностей между блоками $C\in[0,1]^{\gamma\times\gamma}$, где $C[i,j]$ создает

вероятность возникновения ребра между вершиной в блоке C_i и вершиной в блоке C_j. Генеративный процесс для базовой модели SBM выглядит следующим образом:

1. Для каждой вершины $u \in V$ сопоставляется u блоку C_i путем выборки из категориального распределения, определяемого $(p_i, \ldots, p_\gamma)$.

2. Для каждой пары вершин $u \in C_i$ и $v \in C_j$ выбирается ребро в соответствии с (8.2)

$$P(A[u,v] = 1) = C[i,j] \tag{8.2}$$

Ключевое нововведение в SBM заключается в том, что можно контролировать вероятности ребер внутри и между различными блоками, и это позволяет нам генерировать графы, демонстрирующие структуру сообщества. Например, обычная практика SBM состоит в том, чтобы установить постоянное значение α на диагонали матрицы C, т. е. $C[i,i] = \alpha,\ i = 1,\ldots,\gamma$, и отдельную константу $\beta < \alpha$ на недиагональных элементах, т. е. $C[i,j] = \beta, i,j = 1,..\gamma, i \neq j$. В этом случае вершины имеют вероятность α наличия ребра с другой вершины, назначенным тому же сообществу, и меньшую вероятность $\beta < \alpha$ наличия ребра с другой вершиной, назначенной другому сообществу.

Описанная выше модель SBM представляет собой лишь самую базовую вариацию общей структуры SBM. Существует множество вариаций структуры SBM, включая подходы для двудольных графов, графов с вершинами, а также подходы для вывода параметров SBM из данных. Однако ключевым моментом, общим для всех этих подходов, является идея создания модели генеративного графа, которая может отразить понятие сообществ в рамках графа.

8.4 Преимущественное присоединение

Модель SBM может генерировать графы со структурами сообщества. Однако, как и простая модель ER, подход SBM ограничен тем, что он не может зафиксировать структурные характеристики отдельных вершин, которые присутствуют в большинстве реальных графов.

Например, в модели SBM все вершины в сообществе имеют одинаковое распределение степеней. Это означает, что структура отдельных сообществ относительно однородна в том смысле, что все вершины имеют схожие структурные свойства (например, одинаковые степени и коэффициенты кластеризации). Однако, к сожалению, такая однородность совершенно нереалистична в реальном мире. В реальных графах часто видно гораздо более

разнородные и разнообразные распределения степеней, например, с множеством вершин низкой степени и небольшим количеством «центр» вершин высокой степени.

Третья генеративная модель, которую мы представим, названная моделью преимущественного присоединения (Preferential Attachment – PA), представляет собой попытку уловить это характерное свойство реальных распределений степеней. Модель PA построена на предположении, что многие графы реального мира демонстрируют распределение степеней по степенному закону, а это означает, что вероятность того, что вершина u имеет степень d_u, примерно определяется следующим уравнением (8.3):

$$P(d_u = k)\alpha k^{-\alpha} \tag{8.3}$$

где $\alpha > 1$ – параметр.

Распределения по степенному закону и другие родственные распределения обладают тем свойством, что у них тяжелые хвосты. Формально наличие тяжелых хвостов означает, что распределение вероятностей стремится к нулю для крайних значений медленнее, чем экспоненциальное распределение. Это означает, что распределения с тяжелыми хвостами присваивают нетривиальную массу вероятности событиям, которые по существу «невозможны» при стандартном экспоненциальном распределении. В случае распределений степеней свойство тяжелого хвоста означает, что существует ненулевая вероятность встретить небольшое количество вершин очень высокой степени. Интуитивно степенное распределение степеней отражает тот факт, что графы реального мира имеют большое количество вершин с малыми степенями, а также имеют небольшое количество вершин с очень большими степенями (существует много споров относительно распространенности фактических распределений по степенному закону в реальных данных).

Модель PA генерирует графы, демонстрирующие распределение степеней по степенному закону, с помощью простого генеративного процесса:

1. Сначала инициализируется полносвязный граф с m_0 вершинами.
2. Далее итеративно добавляем к этому графу $n - m_0$ вершин. Для каждой новой вершины u, которая добавляется на итерации t, соединяем ее с $m < m_0$ существующими вершинами в графе и выбираем m соседей этой вершины путем выборки без замены согласно следующему распределению вероятностей (8.4):

$$P(A[u,v]) = \frac{d_v^{(t)}}{\sum_{v' \in V^{(t)}} d_{v'}^{(t)}} \quad (8.4)$$

где $d_v^{(t)}$ обозначает степень вершины v на итерации t, а $V^{(t)}$ обозначает множество вершин, которые были добавлены в граф до итерации t.

Основная идея заключается в том, что модель PA соединяет новые вершины с существующими вершинами с вероятностью, пропорциональной степеням существующих вершин. Это означает, что вершины с высокой степенью будут иметь тенденцию накапливать все больше и больше соседей в феномене «богатые становятся еще богаче» по мере роста графа. Можно сказать, что описанная выше модель PA генерирует связные графы, которые имеют степенное распределение степеней с $\alpha = 3$.

Важным аспектом модели PA, который отличает ее от моделей ER и SBM, является то, что процесс генерации является авторегрессивным. Вместо указания вероятностей ребер для всего графа за один шаг модель PA опирается на итеративный подход, в котором вероятности ребер на шаге t зависят от ребер, которые были добавлены на шаге $t-1$. Очевидно, что понятие авторегрессионная генерация будет повторяться в контексте подходов глубокого обучения к генерации графов.

8.5 Традиционные приложения

И так были описаны три традиционных подхода к построению графов: модель Эрдёша-Реньи (ER), стохастическая блочная модель (SBM) и модель преимущественного присоединения (PA). Суть этих моделей заключается в том, что указывается процесс генерации вероятностной модели, которая позволяет зафиксировать некоторые полезные свойства реальных графов, оставаясь при этом управляемыми и простыми для анализа. Исторически сложилось так, что эти традиционные модели поколения использовались в двух ключевых приложениях:

Генерация синтетических данных для задач сравнительного анализа и анализа. Первое полезное применение генеративных моделей заключается в том, что их можно использовать для создания синтетических графов для задач сравнительного анализа. Например, предположим, что разработан алгоритм обнаружения сообщества. Было бы разумно ожидать, что подход должен быть в состоянии сделать вывод о базовых сообществах в графе, сгенерированном моделью SBM. Точно так же, если разработан механизм сетевого анализа, который должен масштабироваться до очень больших графов, рекомендуется протестировать его на синтетических графах, сгенерированных моделью PA,

чтобы убедиться, что механизм анализа может работать с тяжелыми хвостами распределения степеней.

Создание нулевых моделей. Второй ключевой прикладной задачей для традиционных подходов к построению графов является создание нулевых моделей. Предположим, исследуется набор данных социальной сети. После анализа и вычисления различных статистических данных, таких как распределение степеней и коэффициенты кластеризации, можно задать следующий вопрос: насколько неожиданны характеристики этой сети? Модели генеративных графов дают нам точный ответ на этот вопрос. В частности, можно исследовать, в какой степени вероятны (или неожиданны) различные характеристики графов в разных генеративных моделях. Например, наличие распределений степеней с тяжелыми хвостами в социальной сети на первый взгляд может показаться удивительным, но это свойство на самом деле ожидаемо, если предположим, что данные генерируются в соответствии с процессом приемущественного присоединения. В общем, традиционные генеративные модели графов дают нам возможность исследовать, какие виды характеристик можно легко объяснить с помощью простых генеративных процессов. В статистическом смысле они предоставляют нулевые модели, которые можно использовать в качестве ориентира для нашего понимания графов реального мира.

Глубокие генеративные модели

Рассмотренные традиционные подходы к построению графов полезны во многих случаях. Их можно использовать для эффективного создания синтетических графов с определенными свойствами, а также для понимания того, как определенные структуры графов могут возникать в реальном мире. Однако ключевым ограничением этих подходов является то, что они полагаются на фиксированный процесс ручной генерации. Короче говоря, традиционные подходы могут генерировать графы, но у них нет возможности изучать генеративную модель на основе данных.

Теперь рассмотрим различные подходы, решающие именно эту проблему. Эти подходы будут направлены на изучение генеративной модели графов на основе набора обучающих графов. Такие подходы позволяют избежать ручного кодирования определенных свойств, таких как структура сообщества или распределение степеней, в генеративную модель. Вместо этого целью этих подходов является разработка моделей, которые могут наблюдать за набором графов $\{G_1, \ldots, G_n\}$ и учиться генерировать графы с характеристиками, аналогичными обучающему набору.

Представим ряд базовых глубоких генеративных моделей для графов. Эти модели будут адаптировать три наиболее популярных подхода к построению общих глубоких генеративных моделей: вариационные автоэнкодеры (Variational Autoencoder – VAE), генеративно-состязательные сети (Generative Adversarial Networks – GAN) и авторегрессионные модели. Сосредоточимся на простых и общих вариантах этих моделей, уделяя особое внимание деталям высокого уровня. Более того, хотя эти генеративные методы в принципе можно комбинировать друг с другом – например, VAE часто комбинируют с авторегрессионными подходами – не будем подробно обсуждать такие комбинации. Вместо этого начнем с обсуждения базовых моделей VAE для графов, где есть стремление сгенерировать весь граф одновременно в стиле автоэнкодера. Также обсудим, как можно использовать объекты на основе GAN вместо вариационных потерь, но в условиях, когда графы генерируются одновременно. Эти генеративные модели «все сразу» аналогичны генеративным моделям ER и SBM в том, что одновременно отбираются все ребра в графе. Также обсуждим авторегрессионные подходы, которые позволяют генерировать граф постепенно, а не сразу (например, генерировать граф от вершины к вершине). Авторегрессионные подходы имеют сходство с моделью преимущественного присоединения в том, что вероятность добавления ребра на каждом этапе во время генерации зависит от того, какие ребра были ранее добавлены в граф (рис. 8.1).

Для простоты все методы будут сосредоточены только на создании структур графа (т. е. матриц смежности), а не на создании свойств вершин или ребер. Предполагается базовое знакомство с вероятностными автокодировщиками, состязательными и авторегрессионными генеративными моделями, такими как языковые модели на основе долгой краткосрочной памяти.

Глубокие генеративные модели графов являются самыми технически сложными и при этом они находятся на ранних этапах своего зарождения. Таким образом, наша цель – представить ключевые методологические основы, которые вдохновили ранние исследования в этой области, а также выделить несколько влиятельных моделей. Как следствие, часто будем избегать деталей низкого уровня в пользу более подробного ознакомления с этой зарождающейся областью.

8.6 Подходы к вероятностному автокодированию

Вероятностные автокодировщики (VAEs) являются одним из наиболее популярных подходов к разработке глубоких генеративных моделей. Их теория и мотивация глубоко укоренены в статистической области вероятностного вывода.

Однако, ключевую идею, лежащую в основе применения вероятностных автокодировщиков к графам, можно суммировать следующим образом (рис. 9.1): наша цель – обучить модель вероятностного декодера $p_\theta(A|Z)$, из которой можно выбирать реалистичные графы (т. е. матрицы смежности) $\hat{A} \sim p_\theta(A|Z)$ путем обусловливания скрытой переменной Z. В вероятностном смысле стремимся изучить условное распределение по матрицам смежности (причем распределение обусловлено некоторой скрытой переменной).

Нейронная сеть кодировщика отображает входной граф $G = (A, X)$ в апостериорное распределение $q_\varphi(Z|G)$ по скрытым переменным Z. Учитывая выборку из этого апостериорного распределения, модель декодера $p_\theta(A|Z)$ пытается реконструировать матрица смежности.

Чтобы обучить автокодировщик, объединяем вероятностный декодер с моделью вероятностного кодировщика $q_\theta(Z|\mathcal{G})$. Эта модель отображает входной граф $\mathcal{G}$ на апостериорное распределение по скрытой переменной Z. Идея заключается в том, что совместно обучаем кодировщик и декодер так, чтобы декодер мог восстанавливать обучающие графы с учетом скрытой переменной $Z \sim q_\theta(Z|\mathcal{G})$, взятой из кодировщика. Затем, после обучения, можно отказаться от кодировщика и сгенерировать новые графы путем выборки скрытых переменных $Z \sim p(Z)$ из некоторого предшествующего (безусловного) распределения и подачи этих значений в декодер.

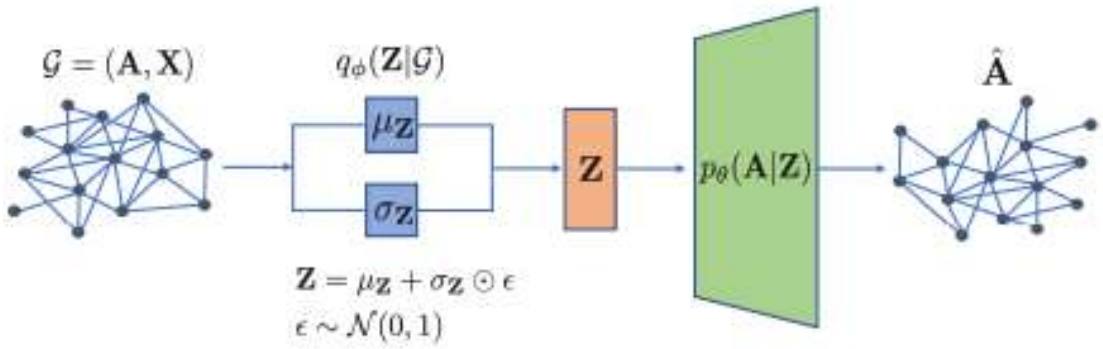

Рис. 8.1: Иллюстрация стандартной модели VAE, примененной к настройке графа.

В более формальных и математических деталях, чтобы построить вероятностный автокодировщик для графов, необходимо указать следующие ключевые компоненты [30]:

1. Вероятностная модель кодировщика q_φ. В случае графов – вероятностная модель кодировщика принимает граф $\mathcal{G}$ в качестве входных данных. Ис-

ходя из них, q_φ находит распределение $q_\varphi(Z|\mathcal{G})$ по скрытым представлениям. Как правило, в вероятностном автокодировщике трюк с изменением параметров с гауссовскими случайными величинами используется для получения вероятностной функции q_φ. То есть определяется скрытое условное распределение как $Z \sim \mathcal{N}\,(\mu_\varphi(\mathcal{G}), \sigma(\varphi(\mathcal{G}))$, где μ_φ и σ_φ – нейронные сети, которые генерируют параметры среднего значения и дисперсии для нормального распределения, из которых выбираются скрытые вложения Z.

2. Модель вероятностного декодера p_θ. Декодер принимает скрытое представление Z в качестве входных данных и использует их для установления условного распределения по графам. Предположим, что p_θ определяет условное распределение по элементам матрицы смежности, т. е. можно вычислить $p_\theta\,(A[u,v]\;=\;1|Z)$.

3. Предварительное распределение p(Z) по скрытому пространству. Примем стандартный гауссовский априор $\mathbf{Z}\sim\mathcal{N}(0,1)$, который обычно используется для вероятностных автокодировщиков.

Учитывая эти компоненты и набор обучающих графов $\{\mathcal{G}_1,\ldots,\mathcal{G}_n\}$, можно обучить модель автокодировщика, минимизировав нижнюю границу вероятности доказательства (evidence likelihood lower bound – ELBO) (8.5):

$$L = \sum_{\mathcal{G}_i\in\{\mathcal{G}_1,\ldots,\mathcal{G}_n\}} \mathbb{E}_{q\theta(\boldsymbol{Z}|\mathcal{G}_i)}[p_\theta(\mathcal{G}_i|Z)] - \mathrm{KL}(q_\theta(Z|\mathcal{G}_i)\parallel p(Z)) \qquad (8.5)$$

Основная идея заключается в том, что стремяться максимизировать способность декодера к восстановлению – т. е. коэффициент вероятности $\mathbb{E}_{q\theta(\boldsymbol{Z}|\mathcal{G}_i)}[p_\theta(\mathcal{G}_i|Z)]$ – при минимизации KL-расхождения между скрытым апостериорным распределением $q_\theta(Z|\mathcal{G}_i)$ и априорным $p(Z)$.

Мотивация, лежащая в основе функции потерь ELBO, лежит в теории вероятностного вывода. Однако, главная идея в том, что необходимо сгенерировать распределение по скрытым представлениям так, чтобы достичь двух (конфликтующих) целей [4]:

1. Отобранные скрытые представления кодируют достаточно информации, чтобы позволить нашему декодеру реконструировать входные данные.

2. Скрытое распределение максимально близко к априорному.

Первая цель гарантирует что, есть обучающие графы в качестве входных данных, тогда научим декодировать значимые графы из закодированных скрытых представлений. Вторая цель действует как регуляризатор и гарантирует, что можно декодировать значимые графы, даже когда отбираем скрытые представления из априорного $p(Z)$. Вторая цель критически важна, если необходимо генерировать новые графы после обучения: можно генерировать новые графы путем вы-

борки из предыдущих и передачи этих скрытых вложений в декодер, и этот процесс будет работать только в том случае, если эта вторая цель удовлетворена.

Рассмотрим два различных способа, с помощью которых идея вероятностных автокодировщиков может быть реализована для графов. Подходы различаются тем, как они определяют кодировщик, декодер и скрытые представления. Однако они разделяют общую идею адаптации модели к графам.

Скрытые значения на уровне вершин

Первый подход, который рассмотрим, тесно связан с идеей кодирования и декодирования графов на основе вложений вершин. Ключевая идея этого подхода заключается в том, что кодировщик генерирует скрытые представления для каждой вершины в графе. Затем декодер принимает пары вложений в качестве входных данных и использует эти вложения для прогнозирования вероятности возникновения ребра между двумя вершинами. Эта идея названа автокодировщиком вероятностного графа (Variational Graph Autoencoder – VGAE).

Модель кодировщика. Модель кодировщика в этой конфигурации может быть основана на любой из архитектур графовых нейронных сетей. В частности, учитывая матрицу смежности A и свойства вершины X в качестве входных данных, используются две отдельные сети для генерации параметров среднего значения и дисперсии, соответственно, обусловленные этими входными данными (8.6) [30]:

$$\mu_Z = GNN_\mu(A, X) \quad log\sigma_Z GNN_\sigma(A, X) \tag{8.6}$$

Здесь μ_Z – это $|\mathcal{V}| \times$ d-мерная матрица, которая задает среднее значение вложения для каждой вершины во входном графе. Логарифмическая матрица $\log \sigma_{\mathbf{Z}} \in \mathbb{R}^{|V| \times d}$ аналогично определяет логарифмическую дисперсию для скрытого вложения каждой вершины.

Учитывая закодированные параметры μ_Z и $\log \sigma_{\mathrm{Z}}$, можно выбрать набор вложений скрытых вершин путем вычисления (8.7),

$$Z = \epsilon \circ exp\big(log(\sigma_{\mathrm{Z}})\big) + \mu_Z \tag{8.7}$$

где $\varepsilon \sim \mathcal{N}(0, 1)$ – размерная матрица $|\mathcal{V}| \times$ d с независимо выбранными единичными нормальными элементами.

Модель декодера. Учитывая матрицу выборочных вложений вершин $Z \in \mathbb{R}^{|V| \times d}$, целью модели декодера является предсказание вероятности для всех ребер в графе. Формально декодер должен указать $p_\theta(A|Z)$ – апостериорную вероят-

ность матрицы смежности с учетом вложений вершин. Опять же, здесь могут быть использованы многие методы такие как различные средства декодирования ребре. Автокодировщик вероятностного графа (VGAE) используют простой декодер точечного произведения, определяемый следующим образом (8.8):

$$p_\theta(A[u,v]=1|z_u,z_v)=\sigma(z_u^{\mathrm{T}}z_v) \tag{8.8}$$

где σ используется для обозначения сигмовидной функции. Обратите внимание, что можно было бы использовать различные реберные декодеры, при условии, что они генерируют действительные значения вероятности.

Чтобы вычислить потери на реконструкцию в уравнении (9.1), предполагается независимость между ребрами и определяем апостериорное $p_\theta(\mathcal{G}|Z)$ по полному графу следующим образом (8.9):

$$p_\theta(\mathcal{G}|Z)=\prod_{(u,v)\in V^2} p_\theta(A[u,v]=1|z_u,z_v) \tag{8.9}$$

что соответствует двоичной потере кросс-энтропии по краевым вероятностям. Чтобы сгенерировать дискретные графы после обучения, можно выбрать ребра на основе апостериорных распределений Бернулли в уравнении (9.4).

Ограничения. Базовая модель VGAE определяет допустимую генеративную модель для графов. После обучения этой модели для восстановления набора обучающих графов можно выбрать вложения вершин Z из стандартного нормального распределения и использовать декодер для генерации графа. Однако генеративная способность этого базового подхода крайне ограничена, особенно когда используется простой декодер точечного произведения. Основная проблема заключается в том, что декодер не имеет параметров, поэтому модель не способна генерировать нетривиальные графовые структуры при отсутствии обучающего графа в качестве входных данных. На самом деле, предложенная модель VGAE в качестве подхода к генерации вложений вершин, не рассматривается как генеративная модель для выборки новых графов.

В некоторых работах предлагалось устранить ограничения VGAE как общей модели, сделав декодер более мощным. Например, предлагают дополнить декодер «итеративным» декодером на основе графовых нейронных сетей. Тем не менее, простой подход с использованием вероятностного автокодировщика на уровне вершин не стал успешным подходом для генерации графов. Он достиг высоких результатов в задачах реконструкции и в качестве фреймворка автокодирования, но в качестве генеративной модели этот простой подход сильно ограничен.

Скрытые значения на уровне графа

В качестве альтернативы подходу VGAE на уровне вершин можно также определить вероятностные автокодировщики на основе скрытых представлений на уровне графа. В этом подходе снова используется функцию потерь ELBO (уравнение 9.1) для обучения модели автокодировщика. Однако модифицируем функции кодировщика и декодера для работы со скрытыми представлениями $z_{\mathcal{G}}$ на уровне графа. Вероятностный автокодировщик на уровне графа, предложен под названием GraphVAE.

Модель кодировщика. Модель кодировщика на уровне графа может быть произвольной моделью графовой нейронной сети, дополненной объединяющим слоем. В частности, позволим сети: $Z^{|V|\times|V|} \times \mathbb{R}^{|V|\times m} \rightarrow \mathbb{R}^{||V|\times d}$ обозначать любую сеть k-уровня, которая выводит матрицу вложений вершин, и будем использовать POOL: $\mathbb{R}^{||V|\times d} \rightarrow \mathbb{R}^{d}$ для обозначения функции объединения, которая накладывает матрицу вложений вершин $\boldsymbol{Z} \in \mathbb{R}^{||V|\times d}$ на вектор вложений на уровне графа $z_{\mathcal{G}} \in \mathbb{R}^{d}$. Используя это обозначение, можно определить кодировщик этой модели на уровне графа с помощью следующих уравнений (8.10):

$$\mu_{Z_{\mathcal{G}}} = POOL_{\mu}\left(GNN_{\mu}(A,X)\right) \;\; log\sigma_{Z_{\mathcal{G}}} = POOL_{\sigma}\left(GNN_{\sigma}(A,X)\right) \qquad (8.10)$$

где снова используем две отдельных сети для параметризации среднего значения и дисперсии апостериорного нормального распределения по скрытым переменным. Обратите внимание на критическое различие между этим кодировщиком на уровне графа и кодировщиком на уровне вершин из предыдущего раздела: здесь генерируется среднее значение $\mu_{z\mathcal{G}} \in \mathbb{R}^{d}$ и логарифм параметров дисперсии $\log \sigma_{z\mathcal{G}} \in \mathbb{R}^{d}$ для одного вложения на уровне графа $z_{\mathcal{G}} \sim \mathcal{N}(\mu_{z\mathcal{G}}, \sigma_{z\mathcal{G}})$, тогда как определили апостериорные распределения для каждой отдельной вершины.

Модель декодера. Цель модели декодера в VAE на уровне графа состоит в том, чтобы определить $p_{\theta}(\mathcal{G}|z_{\mathcal{G}})$, апостериорное распределение конкретной структуры графа с учетом скрытого вложения на уровне графа. Оригинальная модель GraphVAE, предложенная для решения этой проблемы решает задачу путем объединения базового многослойного персептрона (MLP) с предположением о распределении Бернулли. В этом подходе используется MLP для наложения скрытого вектора $z_{\mathcal{G}}$ на матрицу $\tilde{A} \in [0,1]^{|V|\times|V|}$ реберных вероятностей (8.11):

$$\tilde{A} = \sigma\left(\mathrm{MLP}\left(Z_{\mathcal{G}}\right)\right) \qquad (8.11)$$

где сигмоидальная функция σ используется для гарантии вхождения в [0, 1]. В принципе, после этого можно определить апостериорное распределение аналогично случаю на уровне вершин (8.12):

$$p_\theta(\mathcal{G}|Z) = \prod_{(u,v)\in V} \tilde{A}[u,v]A\,[u,v] + \big(1 - \tilde{A}[u,v]\big)(1 - A[u,v]) \qquad (8.12)$$

где A обозначает истинную матрицу смежности графа $\mathcal{G}$, а $\tilde{A}$ –предсказанния матрица вероятностей ребер. Другими словами, предполагается независимые распределения Бернулли для каждого ребра, и общая цель логарифмической вероятности эквивалентна набору независимых двоичных функций потерь кросс-энтропии на каждом ребре. Однако при реализации уравнения (9.8) на практике возникают две ключевые проблемы:

1. Во-первых, если используется MLP в качестве декодера, то нужно предопределить фиксированное количество вершин. Как правило, эта проблема решается путем предположения о максимальном количестве вершин и использования маскирующего подхода. В частности, можно предугадать максимальное количество вершин n_{max} , что ограничит размерность вывода декодера MLP матрицами размера $n_{max} \times n_{max}$. Чтобы декодировать граф с вершинами $|\mathcal{V}| < n_{max}$ во время обучения, маскируется (то есть игнорируются) все записи в $\tilde{A}$ с индексами строк или столбцов, превышающими $|\mathcal{V}|$. Чтобы после обучения модели генерировать графы различных размеров, можно указать распределение $p(n)$ по размерам графов с поддержкой $\{2, \ldots, n_{max}\}$ и выполнить выборку из этого распределения для того, чтобы определить размер сгенерированных графов. Простая стратегия для определения $p(n)$ заключается в использовании эмпирического распределения размеров графа в обучающих данных.

2. Вторая ключевая проблема при применении уравнения (9.8) на практике заключается в том, что не известен правильный порядк строк и столбцов в $\tilde{A}$, когда вычисляем потери при восстановлении. Матрица $\tilde{A}$ просто генерируется MLP, и когда хотим использовать её для вычисления вероятности обучающего графа, нам нужно неявно предположить некоторый порядок вершин (т. е. строк и столбцов $\tilde{A}$). Формально, потеря в уравнении (9.8) требует, чтобы мы указали порядок расположения вершин $\pi \in \Pi$, чтобы упорядочить строки и столбцы в $\tilde{A}$.

Это важно, потому что если просто игнорируется эта проблема, то декодер может перенастроиться на произвольный порядок вершин, используемый во время обучения. Существуют две популярные стратегии решения этой проблемы.

Первый подход заключается в применении эвристики сопоставления графов, с целью попытаться найти такой порядок вершин $\tilde{A}$ для каждого обучающего графа, который даст наибольшую вероятность, это изменит потерю на (8.13)

$$p_\theta(\mathcal{G}|Z) = \max_{\pi\in\Pi} \prod_{(u,v)\in V} \tilde{A}^\pi[u,v]A\,[u,v] + \big(1-\tilde{A}^\pi[u,v]\big)(1-A[u,v]) \quad (8.13)$$

где используется $\tilde{\boldsymbol{A}}^{\boldsymbol{\pi}}$ для обозначения предсказанной матрицы смежности при определенном порядке вершин π. Однако, к сожалению, вычисление максимума в уравнении (9.9) – даже с использованием эвристических приближений – требует больших вычислительных затрат, и модели, основанные на сопоставлении графов, имеют придел в масштабировании на уровне нескольких сотен вершин. В последнее время, используются эвристические методы для упорядочивания вершин. Например, можно располагать вершины на основе поиска по глубине или ширине, начиная с вершины наивысшей степени. В этом подходе просто указывается конкретная функция упорядочения π и вычисляем потери: $p_\theta(\mathcal{G}|Z) \approx \prod_{(u,v)\in V} \tilde{A}^\pi[u,v]A\,[u,v] + \big(1-\tilde{A}^\pi[u,v]\big)(1-A[u,v])$ или рассматривается небольшой набор эвристических методов в упорядочивании $\pi_1, \ldots, \pi_n$ и усредним их результаты: $p_\theta(\mathcal{G}|Z) \approx \sum_{\pi_i\in\{\pi_1,\ldots,\pi_n\}} \prod_{(u,v)\in V} \tilde{A}^\pi[u,v]A\,[u,v] + \big(1-\tilde{A}^\pi[u,v]\big)(1-A[u,v])$. Эти методы не решают проблему сопоставления графов, но, похоже, они хорошо работают на практике.

Ограничения. Как и в случае подхода VAE базовая структура на уровне графа имеет ряд ограничений на уровне вершин. Наиболее заметно, что использование скрытых представлений на уровне графа вводит проблему указания порядка вершин. Эта проблема, вместе с использованием декодеров MLP, в настоящее время ограничивает применение базового VAE графового уровня для небольших графов с сотнями вершин или меньше. Однако, фреймворк VAE на уровне графа можно комбинировать с более эффективными декодерами, включая некоторые методы авторегрессии и это может привести к созданию более надежных моделей. Также рассмотрим один яркий пример такого подхода, когда выделим конкретную задачу генерации структур молекулярного графа.

8.7 Состязательные подходы

Вероятностные автокодировщики (VAEs) являются популярной платформой для глубоких генеративных моделей – не только для графов, но и для изображений, текста и широкого спектра других областей. VAE имеют четко определенную вероятностную мотивацию, и существует множество работ, в которых ис-

пользуется и анализируется структура скрытых пространств, изучаемых с помощью моделей VAE. Однако известно, что они также страдают от серьезных ограничений, таких как тенденция VAE выдавать размытые выходные данные в сфере изображений. Многие современные генеративные модели используют альтернативные фреймворки, причем генеративные состязательные сети (Generative Adversarial Networks – GAN) являются одними из самых популярных.

Основная идея, лежащая в основе общих генеративных моделей на основе GAN, заключается в следующем: сначала определяется обучаемая генеративная сеть $g_\theta: \mathbb{R}^d \to \mathcal{X}$, она будет обучена делать реалистичные (но поддельные) выборки данных $\tilde{x} \in \mathcal{X}$, принимая случайное начальное значение $\mathbf{z} \in \mathbb{R}^d$ в качестве входных данных (например, выборку из нормального распределения); в то же время определяется сеть дискриминатор $d_\varphi: \mathcal{X} \to [0,1]$, ее цель различать реальные выборки данных $x \in \mathcal{X}$ и выборки, сгенерированные генератором $\tilde{x} \in \mathcal{X}$. Здесь можно предположить, что дискриминатор выводит вероятность того, что данный входной сигнал является поддельным.

Чтобы обучить GAN, генератор и дискриминатор совместно оптимизируются в состязательной игре (8.14):

$$\min_\theta \max_\phi \mathbb{E}_{x\sim p_{data}(x)}\left[log\big(1-d_\phi(x)\big)\right] + \mathbb{E}_{z\sim p_{seed}(z)}\left[log\big(d_\phi(g_\theta(z))\big)\right] \quad (8.14)$$

где $p_{\text{data}}(x)$ обозначает эмпирическое распределение реальных выборок данных (например, однородная выборка по обучающему набору), а $p_{\text{seed}}(x)$ обозначает случайное начальное распределение (например, стандартное многомерное нормальное распределение). Уравнение (9.10) представляет собой задачу минимаксной оптимизации. Генератор пытается минимизировать дискриминационную мощность дискриминатора, в то время как он пытается максимизировать свою способность обнаруживать поддельные образцы. Оптимизация минимаксной цели GAN, а также более поздних вариаций, является сложной задачей.

Базовый подход GAN к генерации графов

В контексте генерации графов подход, основанный на GAN, аналогичен VAE на уровне графа. Например, для генератора можно использовать простой многослойный персептрон (MLP), чтобы создать матрицу граничных вероятностей, заданную начальным вектором z (8.15):

$$\tilde{A} = \sigma\big(\text{MLP}(z)\big) \quad (8.15)$$

Учитывая эту матрицу вероятностей ребер, можно затем сгенерировать дискретную матрицу смежности $\widehat{\boldsymbol{A}} \in \mathbb{Z}^{|V|\times|V|}$ путем выборки независимых переменных Бернулли для каждого ребра, с вероятностями, заданными элементами $\widetilde{\boldsymbol{A}}$; т. е. $\widehat{\boldsymbol{A}}[\mathrm{u},\mathrm{v}] \sim$ Бернулли ($\widetilde{\boldsymbol{A}}$ $[\mathrm{u},\mathrm{v}]$). В качестве дискриминатора можно использовать любую модель классификации графов на основе графовой нейронной сети. Затем, модель генератора и модель дискриминатора могут быть обучены в соответствии с уравнением (9.10) с использованием стандартных инструментов для оптимизации GAN.

Преимущества и ограничения подхода GAN

Как и в случае с подходами VAE, фреймворк GAN может быть расширен различными способами. Можно использовать более мощные модели генератора – например, используя методы авторегрессии, – и можно даже включить свойства вершин в модели генератора и дискриминатора.

Одним из важных преимуществ фреймворка на основе GAN является то, что он устраняет сложность указания порядка вершин при вычислении потерь. До тех пор, пока модель дискриминатора инвариантна к перестановкам, что имеет место почти для каждой графовой нейронной сети, подход GAN не требует указания какого-либо порядка вершин. Порядок матрицы смежности, создаваемый генератором, не имеет значения, если дискриминатор инвариантен к перестановкам. Однако, несмотря на это важное преимущество, подходы к генерации графов основанные на GAN до сих пор привлекали меньше внимания и пользовались меньшим успехом, чем их вероятностные аналоги. Возможно, это связано с трудностями, связанными с минимаксной оптимизацией, которая требует подходы, основанные на GAN, а исследование пределов такой генерации в настоящее время является открытой проблемой.

8.8 Методы авторегрессии

Таким образом, было подробно рассмотрено, как идеи вероятностного автокодирования (VAE) и генеративных состязательных сетей (GAN) могут быть применены к генерации графов. Однако, для генерации матриц смежности могут использоваться простые многослойные персептроны (MLP). Теперь представим более сложные методы авторегрессии, которые могут декодировать структуры графов из скрытых представлений. Они могут быть объединены с фреймворками GAN и VAE, но их также можно обучать как автономные генеративные модели.

Моделирование зависимостей ребер

Простые генеративные модели предполагали, что ребра генерируются независимо. Было определено вероятность графа, заданного скрытым представлением

z, разложив общую вероятность на набор независимых вероятностей ребер следующим образом (8.16):

$$P(\mathcal{G}|z) = \prod_{(u,v)\in V^2} P(A[u,v]|z) \tag{8.16}$$

Предполагать независимость между ребрами удобно, поскольку это упрощает модель вероятности и позволяет проводить эффективные вычисления. Однако это не только сильное, но и ограничивающее предположение, поскольку графы реального мира демонстрируют множество сложных зависимостей между ребрами. Например, тенденцию к тому, что графы имеют высокие коэффициенты кластеризации, трудно уловить в модели, не зависящей от ребер. Чтобы устранить эту проблему — сохраняя при этом управляемость, авторегрессионная модель ослабляет предположение о независимости ребер.

Вместо этого в авторегрессионном подходе предполагается, что ребра генерируются последовательно и что вероятность создания каждого из них может быть обусловлена ребрами, которые были сгенерированы ранее. Чтобы уточнить эту идею, будем использовать L для обозначения нижней треугольной части матрицы смежности A. Предполагая, что работая с простыми графами, A и L будут содержать одинаковую информацию, но при этом в следующих уравнениях будет удобнее работать с L. Затем будем использовать обозначение $L[v_1, :]$ для строки L, соответствующей вершине v_1, и будем считать, что строки L индексируются вершинами $v_1, \ldots, v_{|V|}$. Обратите внимание, что из-за нижней треугольной природы $L[\, v_i, v_j\,] = 0, \forall_j > i$, что означает, что нужно беспокоиться только о создании первых i записей для любой строки $L\,[v_i, :]$; остальные могут быть просто дополнены нулями. Учитывая это, авторегрессионный подход сводится к следующей декомпозиции общей вероятности графа (8.17):

$$P(\mathcal{G}|z) = \prod_{i=1}^{|V|} P(L[v_i, :]|L[v_1, :], \ldots, L[v_{i-1}, :], z) \tag{8.17}$$

Другими словами, когда генерируется строка $L\,[v_i, :]$, тогда ставим условие для всех предыдущих сгенерированных строк $L\,[v_j, :]$ с $j < i$.

Рекуррентные модели для генерации графов

Теперь обсудим два конкретных примера авторегрессионной генерации. Они основаны на идеях, и в целом указывают на стратегии, которые можно было бы использовать для решения этой задачи. В первой модели, которую рассмотрим, называемой GraphRNN, моделируются авторегрессионные зависимости с ис-

пользованием рекуррентной нейронной сети (RNN). Во втором подходе – называемом графом рекуррентной сети внимания (GRAN) –генерируются графы, используя графовые нейронные сети для определения матрицы смежности.

GraphRNN

Первой моделью, в которой использовался этот подход к генерации авторегрессии, была GraphRNN. Основная идея подхода заключается в использовании иерархической рекуррентной сети для моделирования зависимостей ребер в уравнении (9.13).

Первый RNN в иерархической модели, называемый RNN уровня графа, используется для моделирования состояния графа. Формально RNN на уровне графа поддерживает скрытое состояние h_i, которое обновляется после генерации каждой строки матрицы смежности $L\,[v_i,:]$ (8.18):

$$h_{i+1} = RNN_{graph}(h_i, L\,[v_i, L]) \tag{8.18}$$

где используencz RNN_{graph} для обозначения общего обновления состояния RNN с h_i, соответствующего скрытому состоянию, и $\boldsymbol{L}\,[v_i, L]$ для наблюдения. В оригинальной формулировке фиксированное начальное скрытое состояние $h_0 = 0$ используется для инициализации RNN на уровне графа, но в принципе это состояние также может быть изучено с помощью модели кодировщика графа или выбрано из скрытого пространства в подход с использованием VAE.

Второй RNN – называемый RNN уровня вершины или RNN_{node} – генерирует записи $L\,[v_i,:]$ авторегрессионным способом. RNN_{node} принимает скрытое состояние h_i на уровне графа в качестве начального ввода, а затем последовательно генерирует двоичные значения $L\,[v_i,;]$, предполагая условное распределение Бернулли для каждой записи. Общий подход GraphRNN называется иерархическим, потому что RNN уровня вершины инициализируется на каждом временном шаге текущего скрытого состояния RNN уровня графа.

Как RNN_{graph} на уровне графа, так и RNN_{node} на уровне вершины могут быть оптимизированы для максимизации вероятности обучающих графов (уравнение 9.13) с использованием стратегии принудительного обучения, что означает, что основные значения истинности L всегда используются для обновления RNN во время обучения. Чтобы управлять размером генерируемых графов, RNN также обучаются выводить маркеры конца последовательности, которые используются для обозначения окончания процесса генерации. Обратите внимание, что — как и в случае с подходами VAE на уровне графа вычисление вероятности в уравнении

(9.13) требует, чтобы предполагалось определенный порядок сгенерированных вершин.

После обучения, чтобы максимизировать вероятность обучающих графов (уравнение 9.13), модель GraphRNN можно использовать для генерации во время тестирования (просто запустив иерархический RNN) начиная с фиксированного начального скрытого состояния h_0. Поскольку RNN на уровне ребер включает в себя процесс стохастической выборки для генерации дискретных ребер, модель GraphRNN способна генерировать разнообразные выборки графов даже при использовании фиксированного начального вложения. Однако модель GraphRNN, в принципе, может использоваться в качестве декодера или генератора в рамках VAE или GAN, соответственно.

Граф рекуррентных сетей внимания (GRAN)

Ключевое преимущество подхода GraphRNN заключается в том, что он моделирует зависимости между ребрами. Используя допущение авторегрессионного моделирования (уравнение 8.18), GraphRNN может обусловливать создание ребер на i шаге генерации, на основе состояния графа, которое уже было сгенерировано на этапах $1, \ldots i - 1$. Обусловливание таким образом значительно упрощает генерацию сложных мотивов и регулярных структур графа, например сетки. На рисунке 9.3 можно видеть, что GraphRNN способен лучше генерировать сеткообразные структуры, по сравнению с базовым VAE графового уровня. Однако подход GraphRNN по-прежнему имеет серьезные ограничения. Как можно видеть на рис. 9.3, при обучении на образцах сеток модель GraphRNN по-прежнему генерирует нереалистичные артефакты (например, длинные цепочки). Более того, GraphRNN может быть трудно обучить и масштабировать до больших графов из-за необходимости обратного распространения через множество этапов повторения RNN.

Чтобы устранить некоторые ограничения подхода GraphRNN, была предложена модель GRAN. GRAN, что расшифровывается как graph recurrent attention networks (реккуретные графовые сети внмания), поддерживает авторегрессионную декомпозицию процесса генерации. Однако вместо использования RNN в моделировании процесса генерации авторегрессии GRAN использует графовые нейронные сети. Ключевая идея GRAN заключается в том, что мы можем смоделировать условное распределение каждой строки матрицы смежности, запустив GNN на сгенерированном к настоящему времени графе (рис. 8.2) (8.19):

$$P(L[v_i, :]|L[v_1, :], \ldots, L[v_{i-1}, :], z) \approx GNN\left(L[v_1, : v_{i-1}, :], \tilde{X}\right) \qquad (8.19)$$

Здесь используется $L[v_1 : v_{i-1}, :]$ для обозначения матрицы смежности нижнего треугольника графа, которая была сгенерирована до этапа генерации i.

GNN в уравнении (9.15) может быть создана многими способами, но важнейшим требованием является то, что она должна генерировать вектор вероятностей ребер $L[v_i, :]$, из которого можно выбирать дискретные реализации ребер во время генерации. Например, используя вариацию модели графическая сеть внимания (graph attention network – GAT) для определения этой GNN. Наконец, поскольку нет атрибутов вершин, связанных со сгенерированными вершинами, входная матрица признаков $\tilde{X}$ для GNN может содержать случайно выбранные векторы (которые полезны для различия вершин).

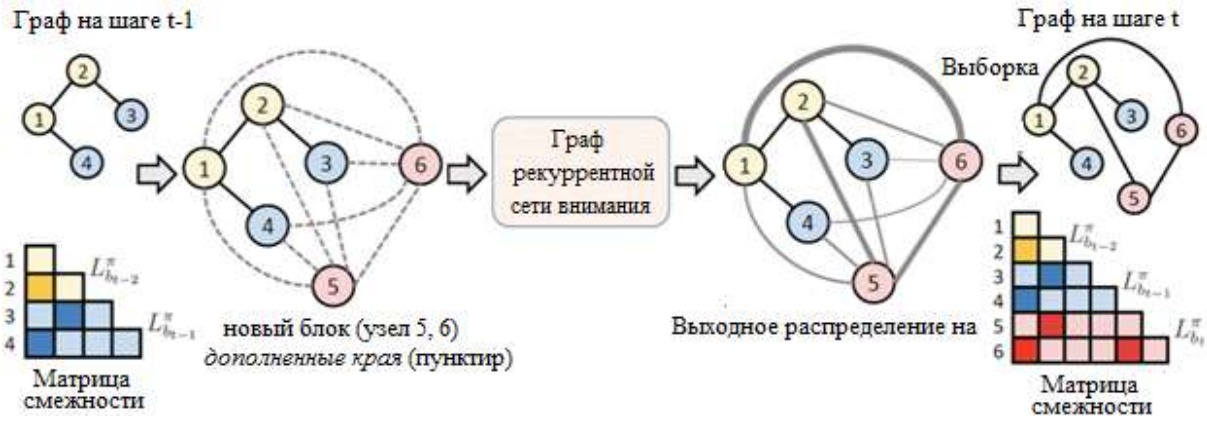

Рис. 8.2: Иллюстрация подхода генерации GRAN.

Модель GRAN может быть обучена аналогично GraphRNN, путем максимизации вероятности обучения графов (уравнение 8.18) с использованием принуждения учителя. Как и в модели GraphRNN, также должен указан порядок вершин для вычисления вероятности на обучающих графах. Наконец, как и GraphRNN, можно использовать GRAN в качестве генеративной модели после обучения, просто запустив процесс стохастической генерации (например, из фиксированного начального состояния), она также может быть интегрирована в фреймворки на основе VAE или GAN.

Ключевое преимущество модели GRAN, по сравнению с GraphRNN, заключается в том, что ей не нужно поддерживать длинную и сложную историю в RNN на уровне графа. Вместо этого модель GRAN явно определяет условия на уже сгенерированном графе, используя графовую нейронную сеть на каждом шаге генерации. Также подробно обсуждается, как модель GRAN может быть оптимизирована для облегчения генерации больших графов с сотнями тысяч вершин. Например, одним из ключевых улучшений производительности является идея о том, что несколько вершин могут быть добавлены одновременно в одном блоке, вместо добавления их по одной за раз. Эта идея проиллюстрирована на рисунке 8.2.

8.9 Оценка генерации графа

Таким образом, были представлен ряд все более сложных подходов к генерации графов, основанных на VAE, GAN и авторегрессионных моделях. Представляя эти подходы, было отмечено превосходство одних над другими. Также приведено несколько примеров сгенерированных графов на рисунке 9.3, которые намекают на возможности этих подходов. Однако, как на самом деле количественно сравниваются эти модели? Как можно сказать, что один подход к генерации графов лучше другого? Оценка генеративных моделей является сложной задачей, поскольку не существует естественного понятия точности или ошибки. Например, можно сравнить потери при реконструкции или вероятности моделирования на развернутых графах, но это осложняется отсутствием единого определения вероятности в различных подходах к генерации.

В общем случае генерация практики заключается в анализе и сравнении распределения статистических данных для сгенерированных графов по тестовым наборам. Формально можно предположить, что есть набор статистических данных графа $S = (s_1,\ s_2,\ \ldots,\ s_n)$, где предполагается, что каждая переменная из них $s_{i,\mathcal{G}}\colon \mathbb{R} \to [0, 1]$ определяет одномерное распределение по $\mathbb{R}$ графа, а также моделей GraphRNN и GRAN (рис. 8.3).

Каждая строка соответствует разным наборам данных. В первом столбце показан пример реального графа, в то время как другие представляют собой случайно выбранные образцы, сгенерированные соответствующей моделью.

Каждая строка соответствует разным наборам данных. В первом столбце показан пример реального графа, в то время как другие представляют собой случайно выбранные образцы, сгенерированные соответствующей моделью для заданного графа $\mathcal{G}$. Например, для полученого графа $\mathcal{G}$ можно вычислить распределение степеней, распределение коэффициентов кластеризации и распределение различных мотивов или графлетов. Учитывая конкретную статистику s_i – вычисленную как на тестовом графе $s_i, \mathcal{G}_{test}$, так и на сгенерированном $s_i, \mathcal{G}_{gen}$ –можно узнать расстояние между распределением статистики на тестовом графе и сгенерированном, используя меру распределения, такую как общее расстояние изменения (8.19):

$$d\big(s_i, \mathcal{G}_{test}, s_i, \mathcal{G}_{gen}\ \big) = sup_{x\in\mathbb{R}}\ \big|s_i, \mathcal{G}_{test}(x) - s_i, \mathcal{G}_{gen}(x)\big| \qquad (8.19)$$

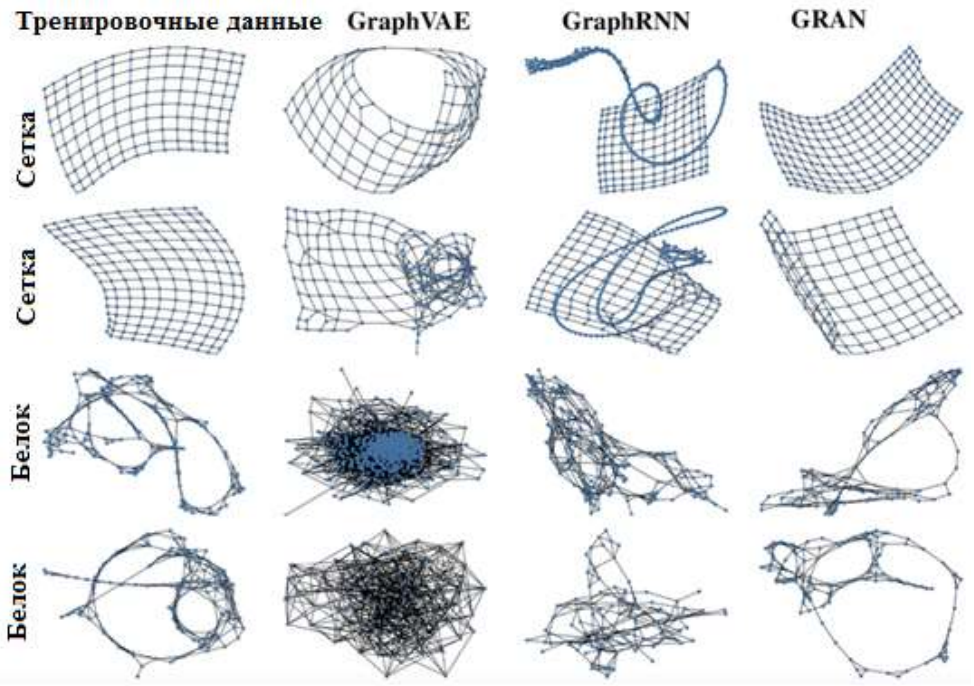

Рисунок 8.3: Примеры графов, сгенерированных с помощью базового значения уровня

Чтобы получить оценку производительности, можно вычислить среднее расстояние попарного распределения между наборами сгенерированных и тестовых графов.

8.10 Генерация молекул

Все подходы к генерации графов, которые были представлены, полезны для общих случаев. Однако не предполагалась конкретная область данных, и целью было просто сгенерировать реалистичные графовые структуры (т. е. матрицы смежности) на основе заданного обучающего набора графов. Однако стоит отметить, что многие сосредоточены конкретно на задаче генерации молекул.

Целью генерации молекул является создание структур молекулярного графа, которые одновременно являются достоверными (например, химически стабильными) и в идеале обладают некоторыми желаемыми свойствами (например, лекарственными или растворимостью). В отличие от общей проблемы генерации графов, исследования по генерации молекул могут существенно выиграть от знаний, относящихся к конкретной предметной области, как для разработки моделей, так и для стратегий оценки. Например, предлагается усовершенствованный вариант подхода VAE на уровне графа, который использует знания об известных молекулярных мотивах. Учитывая сильную зависимость от знаний о предметной области и уникальные проблемы генерации молекул по сравнению с обычными графами, не рассматривались эти подходы подробно. Тем не менее, важно выделить эту область как одну из самых быстрорастущих областей генерации графов.