5. ЛЕКЦИЯ – Графовая нейронная сеть

Теперь обратим внимание на более сложные модели кодировщиков. Представим формализацию графовой нейронной сети, которая является общей основой для определения глубоких нейронных сетей на графовых данных. Ключевая идея заключается в том, что необходимо генерировать представления вершин, которые на самом деле зависят от структуры графа, а также от любой информации о функциях, которая может быть.

Основная проблема при разработке сложных кодеров для данных, структурированных по графам, заключается в том, что обычный набор инструментов глубокого обучения неприменим. Например, свёрточные нейронные сети (convolutional neural networks – CNN) четко определены только для входных данных со структурой сетки (например, изображений), в то время как рекуррентные нейронные сети (recurrent neural networks – RNN) четко определены только для последовательностей (например, текста). Чтобы определить глубокую нейронную сеть поверх общих графов, нам нужно определить новый вид архитектуры глубокого обучения.

Инвариантность перестановок и эквивариантность. Одной из разумных идей для определения глубокой нейронной сети над графами было бы просто использовать матрицу смежности в качестве входных данных для глубокой нейронной сети. Например, чтобы сгенерировать вложение всего графа, можно бы просто сгладить матрицу смежности и передать результат в многослойный персептрон (multi-layer perceptron – MLP) :

$$z_G = MLP(A[1] \oplus A[2] \oplus \ldots \oplus A[|V|]) \qquad (5.1)$$

где $A[i] \in R|V|$ обозначает строку матрицы смежности, и используется знак $\oplus$ для обозначения векторной конкатенации (операция склеивания объектов линейной структуры).

Проблема с этим подходом заключается в том, что он зависит от произвольного порядка вершин, который использовался в матрице смежности. Другими словами, такая модель не является инвариантной к перестановкам, и ключевым требованием для проектирования нейронных сетей над графами является то, что они должны быть инвариантными к перестановкам (или эквивариантными). В математических терминах любая функция f, которая принимает матрицу смежности A в качестве входных данных, в идеале должна удовлетворять одному из двух следующих свойств:

$$f(PAP^T) = f(A) \text{ (Инвариантность перестановки)} \quad (5.2)$$

$$f(PAP^T) = Pf(A) \text{ (Перестановочная эквивалентность)} \quad (5.3)$$

где P – матрица перестановок. Инвариантность перестановки означает, что функция не зависит от произвольного порядка строк / столбцов в матрице смежности, в то время как эквивариантность перестановки означает, что выходные данные f переставляются согласованным образом, когда мы переставляем матрицу смежности. (Неглубокие кодеры, которые мы обсуждали, являются примером эквивариантных функций перестановки.) Обеспечение инвариантности или эквивалентности является ключевой задачей, когда изучаются графы, и в последующем еще будем возвращаться к вопросам, связанным с эквивалентностью и инвариантностью перестановок.

5.1. Передача нейронных сообщений

Модель базовой графовой нейронной сети (graph neural network – GNN) может быть мотивирована различными способами. Та же фундаментальная модель GNN была получена как обобщение сверток на неевклидовых данные, и как дифференцируемый вариант распространения убеждений, а также по аналогии с классическими тестами на изоморфизм графов. Независимо от мотивации, определяющей особенностью GNN является то, что он использует форму передачи нейронных сообщений, при которой векторные сообщения обмениваются между вершинами и обновляются с помощью нейронных сетей.

Необходимо подробно рассмотреть основы системы передачи нейронных сообщений, сосредоточимся на самой структуре передачи сообщений. Тогда можно взять входной граф $G = (V, E)$ вместе с набором объектов вершины $X \in R^{d \times |V|}$ и использовать эту информацию для генерации вложений вершин $z_u, \forall u \in V$.

Обзор структуры передачи сообщений.

Во время каждой итерации передачи сообщений в GNN скрытое вложение h_u^k, соответствующее каждому вершине $u \in V$, обновляется в соответствии с информацией, агрегированной из окрестности графа $N(u)$ (рис. 5.1). Это обновление для передачи сообщений может быть выражено следующим образом:

$$h_u^{k+1} = UPDATE^{(k)}\left(h_u^{(k)}, AGGREGATE^{(k)}\left(\left\{h_v^{(k)}, \forall v \in N(u)\right\}\right)\right) \quad (5.4)$$

$$= UPDATE^{(k)}\left(h_{\mathrm{u}}^{(\mathrm{k})},\, m_{N(u)}^{(k)}\right) \tag{5.5}$$

где UPDATE и AGGREGATE - произвольные дифференцируемые функции (т.е. нейронные сети), а $m_{N(u)}$ - это «сообщение», которое агрегируется из окрестности графа $N(u)$. Используются надстрочные индексы, чтобы различать вложения и функции на разных итерациях передачи сообщений.

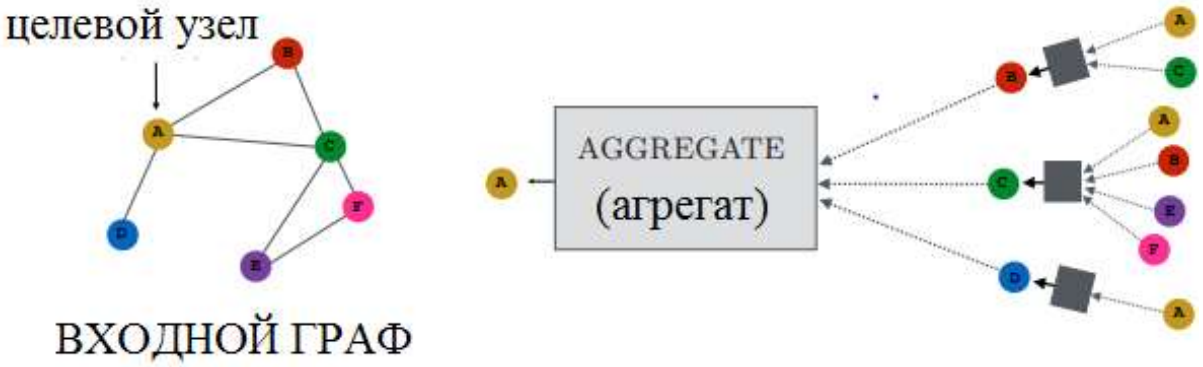

Рис. 5.1: Обзор того, как отдельная вершина агрегирует сообщения из своего локального окружения.

На рисунке 5.1 модель агрегирует сообщения от локальных соседей графа A (т. е. B, C и D), и, в свою очередь, сообщения, поступающие от этих соседей, основаны на информации, собранной (агрегированной) из их соответствующих соседей, и так далее. Эта визуализация показывает двухуровневую версию модели передачи сообщений. Обратите внимание, что граф вычислений GNN формирует древовидную структуру, разворачивая окрестности вокруг целевой вершины.

На каждой итерации k GNN агрегатная функция принимает в качестве входных данных набор вложений вершин в окрестности графа $N(u)$ и генерирует сообщение $m_{\mathrm{N(u)}}^{(k)}$ на основе этой агрегированной информации о соседстве. Функция обновления UPDATE затем объединяет сообщение $m_{\mathrm{N(u)}}^{(k)}$ с предыдущим встраивание $h_u^{(k-1)}$ вершины для генерации обновленного встраивания $h_u^{(k)}$.

Начальные вложения при $k = 0$ устанавливаются на входные объекты для всех вершин, т. е. $h_u^{(0)} = x_u\,, \forall u\ \in V$. После выполнения K итераций передачи сообщения GUN можно использовать выходные данные конечного слоя для определения вложений для каждого вершины, т. е.,

$$z_u = h_u^{(K)} = x_u\,, \forall u\ \in V \tag{5.6}$$

Поскольку агрегатная функция принимает набор в качестве входных данных, GNNs, определенные таким образом, по замыслу являются эквивариантные перестановкам.

Функции вершины. В отличие от методов неглубокого встраивания, платформа GNN требует, чтобы были функции вершины $x_u, \forall u \in V$ в качестве входных данных для модели. На многих графах есть возможности вершин для использования (например, функции экспрессии генов в биологических сетях или текстовые функции в социальных сетях).

Мотивации и интуиции.

Основная интуиция, лежащая в основе платформы передачи сообщений GNN, проста: на каждой итерации каждый вершина агрегирует информацию из своего локального окружения, и по мере выполнения этих итераций каждое вложение вершины содержит все больше и больше информации из других областей графа. Чтобы быть точным:

- после первой итерации ($k = 1$) каждое вложение вершины содержит информацию из его окрестности с 1-шагового соседства, т.е. каждое вложение вершины содержит информацию об особенностях его непосредственных соседей по графу, которые могут быть достигнуты с помощью путь длиной 1 в графе;
- после второй итерации ($k = 2$) каждое вложение вершины содержит информацию из его окрестности с 2-шагового соседства;
- после k итераций каждое вложение вершины содержит информацию о его окрестности с k-шагового соседства.

Но какую «информацию» на самом деле кодируют эти вложения вершин? Как правило, эта информация поступает в двух формах. С одной стороны, существует структурная информация о графе. Например, после k итераций передачи сообщения GNN встраивание $h_u^{(k)}$ вершины может кодировать информацию о степени всех вершин в окрестности k- шаговой окрестности . Эта структурная информация может быть полезна для многих задач. Например, при анализе молекулярных графов можно использовать информацию о степени для определения типов атомов и различных структурных мотивов, таких как бензольные кольца.

В дополнение к структурной информации, другой ключевой вид информации, получаемой при внедрении вершины GNN, основан на характеристиках. После k итераций передачи сообщений GNN вложения для каждого вершины также кодируют информацию обо всех объектах в их окрестности k-шага. Эта локальная функция-агрегации поведения GNN аналогично поведению сверточных ядер в сверточных нейронных сетях. Однако, в то время как CNNs агрегирует информа-

цию об объектах из пространственных определенных участков изображения, и так GNNs агрегирует информацию на основе локальных окрестностей графа.

Базовая графовая нейронная сеть.

Структура GNN пока обсуждалась относительно абстрактно как серия итераций передачи сообщений с использованием функций обновления и агрегирования (уравнение 5.4). Чтобы преобразовать абстрактную структуру GNN, определенную уравнением 5.4, во что-то, что можно реализовать, необходимо предоставить конкретные экземпляры этим функциям обновления и агрегирования. Начнем рассмотрение самой базовой структуры GNN, которая представляет собой упрощение оригинальных моделей GNN, предложенных Мерквиртом, Ленгауэром и Скарселли.

Базовая передача сообщений GNN определяется как (5.7) [20].

$$h_u^{(k)} = \sigma\left(W_{self}^{(k)} h_u^{(k-1)} + W_{neigh}^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)}\right) \qquad (5.7)$$

где $\boldsymbol{W}_{self}^{(k)}, \boldsymbol{W}_{neigh}^{(k)}$(k) $\in \mathrm{R}^{d(k) \times d(k-1)}$ – обучаемые матрицы параметров, а σ обозначает поэлементную нелинейность. Термин смещения $b^{(k)} \in R^{d(k)}$ часто опускается для простоты обозначения, но включение термина смещения может быть важным для достижения высокой производительности. В этом уравнении используются надстрочные индексы для различения параметров, вложений и размерностей в разных слоях GNN.

Передача сообщений в базовой платформе GNN аналогична стандартному многослойному персептрону (MLP) или рекуррентной нейронной сети, например в стиле Элмана (Elman RNN), поскольку она основана на линейных операциях, за которыми следует одноэлементная нелинейность. В начале суммируются сообщения, поступающие от соседей; затем объединяется информация о соседстве с предыдущим внедрением вершины, используя линейную комбинацию; и, наконец, применяется поэлементная нелинейность.

Можно эквивалентно определить базовый GNN с помощью функций UPDATE и AGGREGATE (5.8, 5.9):

$$m_{N(u)} = \sum_{v \in N(u)} h_v, \qquad (5.8)$$

$$UPDATE\left(h_v, m_{N(u)}\right) = \sigma\left(W_{self} h_u + W_{neigh} m_{N(u)}\right) \qquad (5.9)$$

Необходимо вспомнить, что используется (5.10)

$$m_{N(u)} = AGGREGATE^{(k)}\left(\left\{h_v^{(k)}, \forall v \in N(u)\right\}\right) \quad (5.10)$$

в качестве сокращения для обозначения сообщения, которое было агрегировано из окрестности графа u. Здесь также обратим внимание, что отсутствует верхний индекс, обозначающий итерацию в приведенных выше уравнениях, что мы часто будем делать для краткости обозначения.

Уравнения на уровне вершины и графа. В описании базовой модели GNN было определено, что основные операции передачи сообщений на уровне вершины. Теперь будем использовать это соглашение. Однако важно отметить, что многие GNN также могут быть кратко определены с помощью уравнений на уровне графа. В случае базовой GNN можно записать определение модели на уровне графа следующим образом (5.11):

$$H^{(t)} = \sigma\left(AH^{(k-1)}W_{neigh}^{(k)} + H^{(k-1)}W_{self}^{(k)}\right) \quad (5.11)$$

где $H^{(k)} \in R^{|V|\times d}$ обозначает матрицу представлений вершин на уровне t в GNN (причем каждый вершина соответствует строке в матрице), A –матрица смежности графа, при этом опущен термин смещения для простоты обозначения. Хотя это представление на уровне графа нелегко применить ко всем моделям GNN, оно часто бывает более кратким, а также подчеркивает, сколько функций может быть эффективно реализовано с использованием небольшого числа разреженных матричных операций.

Передача сообщений с помощью самоциклов.

В качестве упрощения подхода к передаче нейронных сообщений обычно добавляют самоциклы к входному графу и опускают явный шаг обновления. В этом подходе определим передачу сообщения просто как (5.12),

$$h_u^{(k)} = AGGREGATE\left(\left\{h_v^{(k-1)}, \forall v \in N(u) \cup \{u\}\right\}\right) \quad (5.12)$$

где теперь агрегация выполняется по множеству $N(u) \cup \{u\}$, т. е. по соседям вершины, а также по самоq вершине. Преимущество этого подхода заключается в том, что больше не нужно определять явную функцию обновления, поскольку обновление неявно определяется с помощью метода агрегации. Упрощение передачи сообщений таким образом часто может облегчить перенастройку, но это также серьезно ограничивает выразительность GNN, поскольку информация, поступа-

ющая от соседей вершины, не может быть отличена от информации самой вершины.

В случае базового GNN добавление самоциклов эквивалентно разделению параметров между матрицами W_{self} и W_{neigh}, что дает следующее обновление на уровне графа (5.13):

$$H^{(t)} = \sigma\left((A + I)H^{(t-1)}W^{(t)}\right) \tag{5.13}$$

Это называется подходом GNN с самоциклом.

5.2. Обобщенная агрегация окрестностей

Базовая модель GNN, описанная в уравнении (5.7), может обеспечить высокую производительность, и ее теоретические возможности хорошо изучены. Однако, точно так же, как простой MLP или Elman RNN, базовый GNN может быть улучшен и обобщен многими способами. Рассмотрим, как AGGREGATE оператор может быть обобщен и улучшен, аналогичное обсуждение операции UPDATE.

Нормализация окрестностей.

Самая простая операция агрегирования окрестностей (уравнение 5.8) просто берет сумму вложений соседей. Одна из проблем этого подхода заключается в том, что он может быть нестабильным и очень чувствительным к степеням вершины. Например, предположим, что вершина u имеет в 100 раз больше соседей, чем вершина $u0$ (т. е. гораздо более высокую степень), тогда можно разумно ожидать такого рода $\left\|\sum_{v\in N(u)} h_v\right\| \gg \left\|\sum_{v'\in N(u')} h_{v'}\right\|$ (для любой разумной векторной нормы $\|\cdot\|$). Эта резкая разница в величине может привести к численной нестабильности, а также к трудностям оптимизации.

Одним из решений этой проблемы является простая нормализация операции агрегирования на основе степеней вовлеченных вершин. Самый простой подход – просто взять среднее значение, а не сумму (5.14):

$$m_{N(u)} = \frac{\sum_{v\in N(u)} h_v}{|N(u)|} \tag{5.14}$$

но исследователи добились успеха с другими факторами нормализации, такими как следующая симметричная нормализация, используемая Кипфом и Веллингом (5.15).

$$m_{N(u)} = \sum_{v \in N(u)} \frac{h_v}{\sqrt{|N(u)||N(u)|}} \quad (5.15)$$

Например, в графе цитирования – виде данных, которые проанализировали Кипф и Веллинг – информация из вершин очень высокой степени (т. е. статей, которые цитируются много раз) может быть не очень полезной для определения членства сообщества в графе, поскольку эти статьи могут цитироваться тысячи раз в различных подполях. Симметричная нормализация также может быть мотивирована на основе теории спектральных графов. В частности, объединение симметрично-нормализованной агрегации (уравнение 5.15) вместе с базовой функцией обновления GNN (уравнение 5.9) приводит к аппроксимации первого порядка свертки спектрального графа.

Сверточные сети графов (GCNs).

Одна из самых популярных моделей базовой графовой нейронной сети, сверточная сеть графа (graph convolutional network GCN), использует симметрично-нормализованную агрегацию, а также подход обновления с автоматическим циклом. Таким образом, модель GCN определяет функцию передачи сообщений как (5.16).

$$h_u^{(k)} = \sigma\left(W^{(k)} \sum_{v \in N(u) \cup \{u\}} \frac{h_v}{\sqrt{|N(u)||N(u)|}}\right) \quad (5.16)$$

Этот подход был впервые описан Кипфом и Веллингом и зарекомендовал себя как одна из самых популярных и эффективных базовых архитектур GNN.

Нормализовать или не нормализовать? Правильная нормализация может иметь важное значение для достижения стабильной и надежной работы при использовании GNN. Однако важно отметить, что нормализация также может привести к потере информации. Например, после нормализации может быть трудно (или даже невозможно) использовать изученные вложения для различения вершин разной степени, а различные другие структурные особенности графа могут быть скрыты нормализацией. Фактически, базовый GNN, использующий оператор нормализованного агрегирования в уравнении (5.14), доказуемо менее мощный, чем базовый агрегатор сумм в уравнении (5.8). Таким образом, использование нормализации является вопросом, относящимся к конкретному приложению. Обычно нормализация наиболее полезна в задачах, где информация о характеристиках вершины гораздо полезнее, чем структурная информация, или где существует очень широкий диапазон степеней вершины, который может привести к нестабильности во время оптимизации.

Установка агрегаторов.

Нормализация окрестностей может быть полезным инструментом для повышения производительности GNN. Но что можно сделать больше для улучшения оператора AGGREGATE? Возможно, есть что-то более сложное, чем просто суммирование по соседним вложениям?

Операция агрегирования окрестностей по сути является заданной функцией. Пусть дан набор соседних вложений $\{h_v, \forall\, v \in N(u)\}$, и необходимо сопоставить этот набор с одним вектором $m_{N(u)}$. Тот факт, что $\{h_v, \forall\, v \in N(u)\}$ является множеством, на самом деле довольно важен: не существует естественного порядка соседей вершин, и любая функция агрегации, которую определяют, должна быть инвариантной к перестановкам.

Установка объединения. Один из принципиальных подходов к определению агрегационной функции основан на теории нейронных сетей, инвариантных к перестановкам. Например, уравнение 5.17 показывают, что функция агрегации со следующей формой является универсальным аппроксиматором функции множества (5.17):

$$m_{N(u)} = MLP_\theta\left(\sum_{v \in N(u)} MLP_{\boldsymbol{\varphi}}(h_v)\right) \qquad (5.17)$$

где, используем MLP_θ для обозначения произвольно глубокого многослойного персептрона, параметризованного некоторыми обучаемыми параметрами θ. Теоретические результаты показывают, что любая инвариантная к перестановкам функция, которая отображает набор вложений в одно вложение, может быть аппроксимирована с произвольной точностью моделью, следующей уравнению (5.17).

Необходимо обратить внимание, что данная теория использует сумму вложений после применения первого MLP (как в уравнении 5.17). Тем не менее, возможно заменить сумму альтернативной функцией уменьшения, такой как элементный максимум или минимум, а также обычно комбинируют модели, основанные на уравнении (5.17), с подходами нормализации.

Подходы к объединению множеств, основанные на уравнении (5.17), часто приводят к небольшому увеличению производительности, хотя они также вносят повышенный риск перенастройки, в зависимости от глубины используемых MLP. Если используется объединение наборов, обычно используют MLP, которые имеют только один скрытый слой, поскольку этих моделей достаточно для удовлетворения теории, но они не настолько перепараметризованы, чтобы рисковать катастрофическим перенастройкой.

Объединение Яносси.

Установленные подходы к объединению в пул для агрегации окрестностей, по сути, просто добавляют дополнительные уровни MLP поверх более базовых архитектур агрегации. Эта идея проста, но, как известно, увеличивает теоретическую емкость GNNS. Однако существует другой альтернативный подход, называемый Объединение Яносси, которое также реализуемо более эффективно, чем простое вычисление суммы или среднего значения вложений соседей.

Напомним, что задача агрегирования окрестностей заключается в том, что необходимо использовать функцию, инвариантную к перестановкам, поскольку не существует естественного порядка соседей вершины. В подходе объединения множеств (уравнение 5.17) была достигнута эта инвариантность перестановок, полагаясь на сумму, среднее значение или поэлементный максимум, чтобы свести множество вложений к одному вектору. Эта модель более мощна, объединив это сокращение с нейронными сетями прямой связи (т. е. MLP). Объединение Яносси использует совершенно другой подход: вместо использования перестановочно-инвариантного сокращения (например, суммы или среднего значения) применяется функцию чувствительную к перестановкам и усреднение результата по множеству возможных перестановок.

Пусть $\pi_i \in \Pi$ обозначает функцию перестановки, которая отображает множество $\{h_v, \forall \mathrm{v} \in \mathrm{N(u)}\}$ к определенной последовательности $(h_{v_1}, h_{v_2}, \ldots, h_{v_{|N(u)|}})_{\pi_i}$. Другими словами, π_i берет неупорядоченный набор соседних вложений и размещает эти вложения в последовательности, основанной на некотором произвольном порядке. Затем подход к объединению в пул Яносси выполняет агрегацию окрестностей с помощью (5.18),

$$m_{N(u)} = MLP_\theta \left(\frac{\mathbf{1}}{|\Pi|} \sum_{\boldsymbol{\pi} \in \Pi} \rho_\varphi \left(h_{v_1}, h_{v_2}, \ldots, h_{v_{|N(u)|}} \right)_{\pi_i} \right) \qquad (5.18)$$

где Π обозначает набор перестановок, а ρ_φ – чувствительная к перестановкам функция, например, нейронная сеть, которая работает с последовательностями. На практике ρ_φ обычно определяется как долгая краткосрочная память (Long short-term memory – LSTM), поскольку известно, что LSTM представляют собой мощную архитектуру нейронной сети для последовательностей.

Если множество перестановок Π в уравнении (5.18) равно всем возможным перестановкам, то агрегатор в уравнении (5.18) также является универсальной функцией аппроксиматор для множеств, подобных уравнению (5.17). Однако суммирование по всем возможным перестановкам, как правило, неразрешимо. Таким образом, на практике объединение Яносси использует один из двух подходов:

1. Выборка случайного подмножества возможных перестановок во время каждого применения агрегатора и суммирование только по этому случайному подмножеству.

2. Используйте канонический порядок вершин в наборе окрестностей; например, упорядочите вершины в порядке убывания в соответствии с их степенью, при этом связи будут разорваны случайным образом.

Исследователи включают подробное обсуждение и эмпирическое сравнение этих двух подходов, а также других методов аппроксимации (например, усечение длины последовательности), и их результаты указывают на то, что объединение в стиле Янossи может улучшить объединение наборов в ряде установок синтетической оценки.

Внимание соседей.

В дополнение к более общим формам агрегации множеств, популярной стратегией улучшения уровня агрегации в GNNs является привлечение внимания. Основная идея заключается в присвоении веса внимания или важности каждому соседу, который используется для оценки влияния этого соседа на этапе агрегирования. Первой моделью GNN, применившей этот стиль была графическая сеть внимания (Graph Attention Network – GAT), которая использует веса внимания для определения взвешенной суммы соседей (5.19) [21]:

$$m_{N(u)} = \sum_{v \in N(u)} \alpha_{u,v} h_v \tag{5.19}$$

где $a_{u,v}$ обозначает внимание к соседу $v \in N(u)$, когда агрегируется информация в вершине u. В оригинале GAT веса внимания определяются как (5.20) [21],

$$\alpha_{u,v} = \frac{exp(a^{\mathrm{T}}[Wh_u \oplus Wh_v])}{\sum_{v' \in N(u)} exp(a^{\mathrm{T}}[Wh_u \oplus Wh_{v'}])} \tag{5.20}$$

где a - обучаемый вектор внимания, W - треугольная матрица, а знак $\oplus$ обозначает операцию объединения.

Известно, что вычисление внимания в стиле GAT хорошо работает с графическими данными. Однако, в принципе, может быть использована любая стандартная модель внимания из литературы по глубокому обучению в целом. Популярные варианты внимания включают билинейную модель внимания (5.21),

$$\alpha_{u,v} = \frac{exp(h_u^{\mathrm{T}} W h_v)}{\sum_{v' \in N(u)} exp(h_u^{\mathrm{T}} W h_{v'})} \quad (5.21)$$

а также вариации уровней внимания с использованием MLP, например (5.22),

$$\alpha_{u,v} = \frac{exp\big(MLP(h_u, h_v)\big)}{\sum_{v' \in N(u)} exp\Big(MLP\big(h_u, h_{v'}\big)\Big)} \quad (5.22)$$

где MLP ограничен скалярным выводом.

Кроме того, хотя это менее распространено в литературе по GNN, также возможно добавить несколько «голов» внимания в стиле популярного преобразования архитектуры. В этом подходе вычисляются K различных весов внимания $a_{u,v,k}$, используя независимо параметризованные слои внимания. Сообщения, агрегированные с использованием различных весов внимания, затем преобразуются и объединяются на этапе агрегирования, обычно с линейной проекцией, за которой следует операция объединения, например, (5.23 и 5.24),

$$m_{N(u)} = [a_1 \oplus a_2 \oplus \ldots \oplus a_k] \quad (5.23)$$

$$a_k = W_k \sum_{v \in N(u)} \alpha_{u,v,k} h_v \quad (5.24)$$

где веса внимания $a_{u,v,k}$ для каждой из K головок внимания могут быть вычислены с использованием любого из вышеперечисленных механизмов внимания.

Графическое внимание и трансформаторы. Модели GNN с многоголовым вниманием (уравнение 5.23) тесно связаны с архитектурой трансформатора. Трансформеры – популярная архитектура как для обработки естественного языка (natural language processing – NLP), так и для компьютерного зрения, и — в случае NLP - они были важной движущей силой крупных современных системах, такие как BERT и XLNet. Основная идея трансформера заключается в определении слоев нейронной сети, полностью основанных на операции внимания. На каждом слое в трансформере, новое скрытое представление генерируется для каждой позиции во входных данных (например, для каждого слова в предложении) с использованием нескольких заголовков внимания для вычисления весов внимания между всеми парами позиций во входных данных, которые затем агрегируются с взвешенными суммами на основе этих весов внимания (способом, аналогичным уравнению 5.23). Фактически, базовый уровень трансформатора в точности

эквивалентен уровню GNN, использующему многоголовое внимание (т. е. уравнение 5.23), если мы предположим, что GNN получает полностью связанный граф в качестве входных данных.

Эта связь между GNNs и трансформаторами использовалась в многочисленных работах. Например, одна стратегия внедрения для проектирования GNNs заключается в том, чтобы просто начать с модели трансформатора, а затем применить двоичную маску смежности на уровне внимания, чтобы гарантировать, что информация агрегируется только между вершинами, которые действительно связаны на графе. Этот стиль реализации GNN может извлечь выгоду из многочисленных существующих хорошо спроектированных библиотек для архитектур трансформеров [20].

Однако недостатком этого подхода является то, что преобразователи должны вычислять попарное внимание между всеми позициями/узлами во входных данных, что приводит к квадратичной временной сложности $O(|V|^2)$ для агрегирования сообщений для всех узлов в графе по сравнению с $O(|V||E|)$ временная сложность для более стандартной реализации GNN.

Добавление внимания является полезной стратегией для увеличения репрезентативности модели GNN, особенно в тех случаях, когда есть предварительные знания, указывающие на то, что некоторые соседи могут быть более информативными, чем другие. Например, рассмотрим случай классификации статей по тематическим категориям на основе в сетях цитирования. Часто встречаются статьи, которые охватывают тематические границы или которые высоко цитируются в различных областях. В идеале, основанный на внимании GNN научился бы игнорировать эти документы при передаче нейронных сообщений, поскольку такие беспорядочные соседи, скорее всего, были бы неинформативны при попытке определить тематическую категорию конкретного вершины.

5.3. Обобщенные методы обновления

Агрегатный оператор в моделях GNN, привлекает наибольшее внимание исследователей — с точки зрения предложения новых архитектур и вариаций. Это было особенно актуально после введения фреймворка GraphSAGE, который ввел идею обобщенной агрегации окрестностей. GraphSAGE реализован в TensorFlow и может быть легко интегрирован в другие конвейеры машинного обучения. Однако передача сообщений GNN включает в себя два ключевых этапа: агрегирование и обновление, и во многих отношениях оператор обновления

играет одинаково важную роль в определении мощности и индуктивного смещения модели GNN.

До сих пор рассматривался базовый подход GNN, где операция обновления включает линейную комбинацию текущего вложения вершины с сообщением от его соседей, а также подход с самоциклом, который просто включает добавление самоцикла в граф перед выполнением агрегации окрестностей. Теперь рассмотрим разнообразные обобщения оператора обновления.

Чрезмерное сглаживание и влияние соседства. Одна общая проблема с GNN, которую могут помочь решить обобщенные методы обновления, известна как чрезмерное сглаживание. Основная идея чрезмерного сглаживания заключается в том, что после нескольких итераций передачи сообщений GNN представления для всех вершин графа могут стать очень похожими друг на друга. Эта тенденция особенно распространена в базовых моделях GNN и моделях, использующих подход самообновления. Чрезмерное сглаживание проблематично, поскольку оно делает невозможным построение более глубоких моделей GNN, которые используют долгосрочные зависимости на графе — так как эти глубокие модели GNN, как правило, просто генерируют чрезмерно сглаженные вложения.

Эта проблема чрезмерного сглаживания в GNNs может быть формализована путем определения влияния входного признака каждого вершины $h_u^{(0)} = x_u$ на встраивание конечного слоя всех других вершин в графе, т. е. $h_u^{(K)} \forall\, v \in V$. В частности, для любой пары вершин u и v можно количественно оценить влияние вершины u на вершина v в GNN, изучив величину соответствующей матрицы Якоби (5.25) [21]:

$$I_K(u, v) = 1^T \left(\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}}\right) 1 \qquad (5.25)$$

где 1 - вектор всех единиц. Значение I_K (u,v) использует сумму записи в матрице Якоби $\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}}$ как мера того, насколько начальное встраивание вершины u влияет на окончательное встраивание вершины v в GNN. Учитывая это определение, доказывается следующая теорема:. Для любой модели GNN, использующей подход к самоцикловому обновлению и функцию агрегирования вида (5.26),

$$AGGREGATE\left(\{h_v, \forall\, v \in N(u) \cup \{u\}\}\right) = \frac{1}{f_n(|N(u)\cup\{u\}|)} \sum_{v \in N(u)\cup\{u\}} h_v \qquad (5.26)$$

где $f: \mathbb{R}^+ \to \mathbb{R}^+$ – произвольная дифференцируемая функция нормализации, мы имеем, что (5.27) [21],

$$I_K(u, v) \; \alpha \; p_{g,K}(u|v) \tag{5.27}$$

где $p_{g,K}(u|v)$ обозначает вероятность посещения вершины v при случайном блуждании длиной K, начинающемся с вершины u.

Таким образом, спользуя K-уровневую модель в стиле GCN (Graph Convolution Neural Network – графовые свёрточные нейронные сети), влияние вершины u и вершины v пропорционально вероятности достижения вершины v при K-шаговом случайном блуждании, начинающемся с вершины u. Важным следствием этого, однако, является то, что при $K \to \infty$ влияние каждого вершины приближается к стационарному распределению случайных блужданий по графу, это означает, что информация о местном районе теряется. Более того, во многих графах реального мира, которые содержат вершины высокой степени и напоминают так называемые графы «расширения», требуется всего $k = Olog(|V|)$ шагов для случайного блуждания, начинающегося с любого вершины, чтобы сходиться к почти равномерному распределению.

Теорема применима непосредственно к моделям, использующим подход самообновления цикла, но результат также может быть расширен в асимптотическом смысле для базового обновления GNN (т.е. уравнения 5.9) до тех пор, пока $\left\|\boldsymbol{W}_{self}^{(k)}\right\| < \left\|\boldsymbol{W}_{neigh}^{(k)}\right\|$ на каждом уровне k. Таким образом, при использовании простых моделей GNN – и особенно тех, которые основаны на подходе к самоцикловому обновлению и построение более глубоких моделей может фактически снизить производительность. По мере добавления новых слоев теряется информация о локальных структурах окрестностей, и наши изученные вложения становятся чрезмерно сглаженными, приближаясь к почти равномерному распределению.

Конкатенация и пропуск соединений.

Чрезмерное сглаживание является основной проблемой GNN. Чрезмерное сглаживание происходит, когда информация, относящаяся к вершине, становится «размытой» или «теряется» после нескольких итераций передачи сообщений GNN. Интуитивно можно ожидать чрезмерного сглаживания в тех случаях, когда информация, собираемая от соседних вершин во время передачи сообщений, начинает доминировать в обновленных представлениях вершин. В этих случаях обновленные представления вершин (т. е. $m_{N(u)}$ векторы) будут слишком сильно

зависеть от входящего сообщения, агрегированного от соседей (т. е. $h_u^{(k)}$ векторов). Очевидный способ решить эту проблему – использовать векторные конкатенации или пропускать соединения, которые пытаются напрямую сохранить информацию из предыдущих циклов передачи сообщений на этапе обновления.

Эти методы конкатенации и пропуска соединения можно использовать в сочетании с большинством других подходов к обновлению GNN. Таким образом, ради общности надо использовать $UPDATE_{base}$ для обозначения базовой функции обновления, которая ляжет в основу (например, мы можем предположить, что $UPDATE_{base}$ задается уравнением 5.9), и определяем обновления с пропуском соединения поверх этой базовой функции. Одно из самых простых обновлений соединения с пропуском использует конкатенацию для сохранения большего количества информации на уровне вершины во время передачи сообщения (5.28):

$$UPDATE_{concat}\left(h_u, m_{N(u)}\right) = \left[UPDATE_{base}\left(h_u, m_{N(u)}\right) \oplus h_u\right] \qquad (5.28)$$

где объединяется вывод базовой функции обновления с представлением вершины предыдущего уровня. Опять же, ключевая идея здесь в том, что поощряется модель распутывания информации во время передачи сообщения, отделяя информацию, поступающую от соседей (т. е. $m_{N(u)}$) от текущего представления каждой вершины (т. е $h_u^{(k)}$).

Пропуск соединения на основе конкатенации был предложен в структуре GraphSAGE, которая была одной из первых работ, подчеркнувших возможные преимущества такого рода модификаций функции обновления. Однако в дополнение к конкатенации также можно использовать другие формы пропуска соединений, такие как метод линейной интерполяции (5.29):

$$UPDATE_{interpolate}\left(h_u, m_{N(u)}\right) = \\ \propto_1 \circ UPDATE_{base}\left(h_u, m_{N(u)}\right) + \propto_2 \odot h_u \qquad (5.29)$$

где $\alpha_1, \alpha_2 \in [0,1]^d$ – стробирующие векторы, где $\alpha_2 = 1 - \alpha_1$, а $\circ$ обозначает поэлементное умножение. В этом подходе окончательное обновленное представление является линейной интерполяцией между стробирования α_1 могут быть изучены вместе с моделью различными способами. Например, генерируется α_1 как результат отдельного однослойного GNN, который принимает текущие представления скрытого слоя в качестве признаков. Однако могут быть использованы и другие более простые подходы, такие как непосредственное изучение параметров α_1 для каждого уровня передачи сообщений или

использование MLP (многослойный перцептрон Румельхарта) на представлениях текущей вершины для генерации параметров стробирования.

В общем, конкатенация и остаточные соединения представляют собой простые стратегии, которые могут помочь решить проблему чрезмерного сглаживания в GNN, а также улучшить численную стабильность оптимизации. Действительно, подобно пользе остаточных соединений в сверточных нейронных сетях (CNN), применение этих подходов к GNN может облегчить обучение гораздо более глубоких моделей. На практике эти методы, как правило, наиболее полезны для задач классификации вершин с умеренно глубокими GNN (например, 2–5 слоев) и превосходно подходят для задач, демонстрирующих гомофилию, т. е. когда прогноз для каждой вершины тесно связан с особенностями её окружения.

Закрытые обновления.

Таким образом, были расмотрены подходы с пропуском соединения и остаточным соединением, которые сильно аналогичны методам, используемым в компьютерном зрении для построения более глубоких архитектур CNN. Исследователи также черпали вдохновение в методах стробирования, используемых для повышения стабильности и способности к обучению рекуррентных нейронных сетей (RNN). В частности, один из способов просмотра алгоритма передачи сообщений GNN заключается в том, что функция агрегирования получает наблюдение от соседей, которое затем используется для обновления скрытого состояния каждой вершины. С этой точки зрения можно напрямую применять методы, используемые для обновления скрытого состояния архитектур RNN на основе наблюдений. Например, в одной из первых архитектур GNN функция обновления определяется как (5.30),

$$h_u^{(k)} = GRU\left(h_u^{(k-1)}, m_{N(u)}^{(k)}\right) \tag{5.30}$$

где GRU обозначает уравнение обновления ячейки со стробируемой рекуррентной единицей (GRU) В других подходах использовались обновления на основе архитектуры LSTM (сети долгой краткосрочной памяти).

В общем, любая функция обновления, определенная для RNN (рекуррентные нейронные сети), может использоваться в контексте GNN. Тогда заменяется аргумент скрытого состояния функции обновления RNN (обычно обозначаемый $h^{(\mathrm{t})}$) скрытым состоянием вершины, и заменяется вектор наблюдения (обычно обозначаемый $x^{(\mathrm{t})}$) сообщением, агрегированным из локальной окрестности. Важно отметить, что параметры этого обновления RNN

всегда совместно используются вершинами (т. е. используencz одну и ту же ячейку LSTM или GRU для обновления каждой вершины). На практике исследователи обычно совместно используют параметры функции обновления и между уровнями передачи сообщений GNN.

Эти закрытые обновления очень эффективны для облегчения глубоких архитектур GNN (например, более 10 слоев) и предотвращения чрезмерного сглаживания. Как правило, они наиболее полезны для приложений GNN, где задача прогнозирования требует сложных рассуждений о глобальной структуре графа, таких как приложения для анализа программ или комбинаторной оптимизации.

Связи со знаниями.

Таким образом предполагается, что используются выходные данные последнего слоя GNN. Другими словами, предполагается, что представления вершин z_{u}, которые используется в нисходящей задаче, равны вложениям вершин конечного уровня в GNN (5.31):

$$z_{\mathrm{u}} = h_u^{(K)}, \forall\, v \in V \tag{5.31}$$

Это предположение делается во многих подходах GNN, и ограничения этой стратегии во многом мотивировали потребность в остаточных и закрытых обновлениях для ограничения чрезмерного сглаживания.

Однако дополнительная стратегия улучшения качества представлений конечной вершины состоит в том, чтобы просто использовать представления на каждом уровне передачи сообщений, а не использовать только выходные данные конечного уровня. В этом подходе мы определяем окончательные представления вершины z_{u} как (5.32),

$$z_{\mathrm{u}} = f_{JK}(h_u^{(0)} \oplus h_u^{(1)} \oplus \ldots \oplus h_u^{(K)}) \tag{5.32}$$

где f_{JK} – произвольная дифференцируемая функция. Эта стратегия известна как добавление связей прыжковых знаний (JK – jumping knowledge). Во многих приложениях функция f_{JK}, может быть, просто определена как функция идентичности, что означает, что просто объединяем вложения вершин из каждого слоя, но также изучаются и другие варианты, такие как подходы с максимальным объединением и уровни внимания LSTM. Этот подход часто приводит к

последовательным улучшениям в широком спектре задач и в целом является полезной стратегией.

5.4. Многореляционные графовые нейронные сети

До сих пор обсуждение GNN и передачи нейронных сообщений неявно предполагало, что есть простые графы. Однако существует множество приложений, в которых рассматриваемые графы являются мультиреляционными или иным образом неоднородными (например, графы знаний).

Нейронные сети реляционного графа.

Первый подход, предложенный для решения этой проблемы, широко известен как подход сверточной сети реляционных графов (Relational Graph Convolutional Network – RGCN). В этом подходе необходимо расширить функцию агрегирования, чтобы учесть несколько типов отношений, указав отдельную матрицу преобразования для каждого типа отношения (5.33):

$$m_{N(u)} = \sum_{r \in R} \sum_{v \in N_\tau(u)} \frac{W_\tau h_v}{f_n(N(u),N(v))} \tag{5.33}$$

где f_n – функция нормализации, которая может зависеть как от окрестности вершины u, так и от соседа v, по которому производится агрегирование. Можно обсудить несколько стратегий нормализации для определения f_n. Таким образом, мультиреляционная агрегация в RGCN аналогична базовому подходу GNN с нормализацией, где отдельно агрегируется информация по разным типам ребер.

Совместное использование параметров. Одним из недостатков подхода RGCN является резкое увеличение количества параметров, поскольку теперь у нас есть одна обучаемая матрица для каждого типа отношения. В некоторых приложениях, таких как приложения на графах знаний с множеством различных отношений, такое увеличение параметров может привести к переоснащению и медленному обучению. Предлагается схема для решения этой проблемы путем совместного использования параметров с базисными матрицами, где (5.34):

$$W_\tau = \sum_{i=1}^{b} \alpha_{i,\tau} B_i \tag{5.34}$$

В этом базисно-матричном подходе все матрицы отношений определяются как линейные комбинации b базисных матриц $B_1, \ldots, B_b$, а единственными параметрами, характерными для отношений, являются веса b комбинаций $\alpha_{1,\tau}, \ldots, \alpha_{b,\tau}$ для каждого отношения τ . Таким образом, при таком подходе к

совместному использованию базиса можуj переписать полную функцию агрегирования как (5.35):

$$m_{N(u)} = \sum_{r \in R} \sum_{v \in N_\tau(u)} \frac{\alpha_\tau \times_1 B \times_2 h_v}{f_n(N(u), N(v))} \quad (5.35)$$

где $B = (B_1, \ldots, B_b)$ – тензор, образованный суммированием базисных матриц, $\alpha_\tau = (\alpha_{1,\tau}, \ldots, \alpha_{b,\tau})$ – вектор, содержащий веса базисных комбинаций для отношения τ, и $\times i$, обозначающий тензорное произведение по моде i. Таким образом, альтернативный взгляд на подход RGCN с совместным использованием параметров заключается в том, что изучается вложение для каждого отношения, а также тензор, который является общим для всех отношений.

Расширения и варианты. Архитектура RGCN может быть расширена многими способами, и в целом можно называть подходы, которые определяют отдельные матрицы агрегации для каждого отношения, нейронными сетями реляционного графа. Например, может использоваться вариант этого подхода без совместного использования параметров для моделирования мультиреляционного набора данных, относящегося к лекарствам, болезням и белкам, и аналогичная стратегия используется для анализа графов лингвистических зависимостей.

Объединение внимания и функций.

Реляционный подход GNN, в котором определяется отдельный параметр агрегации для каждого отношения, применим для мультиреляционных графов и случаев, когда есть дискретные граничные функции. Чтобы приспособиться к случаям, когда есть более общие формы граничных функций, можно использовать эти функции внимания или путем объединения этой информации с соседними вложениями во время передачи сообщения. Например, при любом базовом подходе к агрегации $AGGREGATEbase$ одной простой стратегией использования пограничных функций является определение новой функции агрегации как (5.36):

$$m_{N(u)} = AGGREGATEbase\big(\{h_v \oplus e_{(u,\tau,v)}, \forall\, v \in N(u)\}\big) \quad (5.36)$$

где $e_{(u,\tau,v)}$ обозначает произвольный векторный признак для ребра (u, τ, v). Этот подход прост и универсален, и в последнее время он показал успех в подходах, основанных на внимании, в качестве базовой функции агрегации.

5.5 Объединение графов

Нейронные передачи сообщений создают набор вложений вершин, однако, что, делать если нужны прогнозы на уровне графа? Другими словами, предполагается, что цель состоит в том, чтобы изучить представления вершин z_u, $\forall u \in V$, но что, если изучено вложение z_G для всего графа G? Эту задачу часто называют объединением графов, поскольку наша цель – объединить вложения вершин, чтобы изучить вложение всего графа.

Установка подходов к объединению. Подобно оператору $AGGREGATE$, задачу объединения графов можно рассматривать как проблему обучения множествам. Необходимо разработать функцию объединения f_p, которая отображает набор вложений вершин $\{z_1, \ldots, z_{|V|}\}$ во вложение z_G, представляющее полный граф. В самом деле, любой из подходов, обсуждаемых в разделе для изучения наборов вложений соседей, также может быть использован для объединения на уровне графа. На практике есть два подхода, которые чаще всего применяются для изучения вложений на уровне графа с помощью объединения наборов. Первый подход состоит в том, чтобы просто взять сумму (или среднее значение) вложений вершин (5.37):

$$z_G = \frac{\sum_{v \in V} z_u}{f_n(|V|)} \tag{5.37}$$

где f_n – некоторая нормирующая функция (например, тождественная функция). Несмотря на простоту, объединение, основанное на сумме или среднем значении вложений вершин, часто бывает достаточным для приложений, использующих небольшие графы.

Второй популярный подход, основанный на наборах, использует комбинацию LSTM (сети долгой краткосрочной памяти) и внимание к объединению вложений вершин. В этом подходе объединения итерируем серию агрегаций на основе внимания, определяемых следующим набором уравнений, которые повторяются для $t = 1, \ldots, T$ шагов (5.38 – 5.41):

$$g_t = LSTM(o_{t-1}, g_{t-1}) \tag{5.38}$$

$$e_{v,t} = f_a(z_u, g_t), \forall v \in V \tag{5.39}$$

$$a_{v,t} = \frac{exp(e_{v,i})}{\sum_{u \in V} exp(e_{v,t})}, \forall v \in V \tag{5.40}$$

$$o_t = \sum_{v \in V} a_{v,t} z_u \tag{5.41}$$

В приведенных выше уравнениях вектор g_t представляет собой вектор запроса внимания на каждой итерации t. В уравнении (5.39) вектор запроса используется для вычисления показателя внимания по каждой вершине с использованием функции внимания $f_a: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ (например, скалярное произведение), и затем этот показатель внимания нормализуется в уравнении (5.40). Наконец, в уравнении (5.41) взвешенная сумма вложений вершин вычисляется на основе весов внимания, и эта взвешенная сумма используется для обновления вектора запроса с использованием обновления LSTM (уравнение 5.38). Обычно векторы q_0 и o_0 инициализируются нулевыми значениями, и после повторения уравнений (5.38 – 5.41) для T итераций вложение для полного графа вычисляется как (5.42):

$$z_G = o_1 \oplus o_2 \oplus \ldots \oplus o_T \tag{5.42}$$

Этот подход представляет собой сложную архитектуру объединения множества на основе внимания и стал популярным методом объединения во многих задачах классификации на уровне графа.

Подходы к укрупнению графов. Одним из ограничений подходов к объединению наборов является то, что они не используют структуру графа. Хотя разумно рассматривать задачу объединения графов просто как задачу обучения набора. Также могут быть преимущества от использования топологии графа на этапе объединения. Одной из популярных стратегий для достижения этого является выполнение кластеризации или огрубления графа как средства объединения представлений вершин.

В данных подходах предполагается, что есть некоторая функция кластеризации (5.43),

$$f_c \to G \times \mathbb{R}^{|V| \times d} \to \mathbb{R}^{+|V| \times c} \tag{5.43}$$

который сопоставляет все вершины в графе с назначением по C кластерам. В частности, предполагается, что эта функция выводит матрицу назначений $S = f_c(G, Z)$, где $S[u, i] \in R +$, обозначающую силу связи между вершиной u и кластером i. Одним из простых примеров функции f_c может быть метод спектральной кластеризации, где назначение кластера основано на спектральном разложении матрицы смежности графа. В более сложном определении f_c можно использовать другую GNN для прогнозирования кластерных назначений].

Независимо от подхода, используемого для создания матрицы назначения кластеров S, ключевая идея данного подхода заключается в том, что используется эта матрица для огрубления графа. В частности, можно использовать матрицу назначений S для вычисления новой укрупненной матрицы смежности (5.44)

$$A^{new} = S^T AS \in \mathbb{R}^{+c \times c} \tag{5.44}$$

и новый набор функций вершины (5.45) [22].

$$X^{new} = S^T X \in \mathbb{R}^{c \times d} \tag{5.45}$$

Таким образом, новая матрица смежности теперь представляет силу связи (то есть ребер) между кластерами в графе, а новая матрица признаков представляет агрегированные вложения для всех вершин, назначенных каждому кластеру. Затем можно запустить GNN на этом огрубленном графе и повторить весь процесс огрубления для нескольких итераций, где размер графа уменьшается на каждом шаге. Затем окончательное представление графа вычисляется путем объединения наборов вложений вершин в достаточно огрубленном графе.

Этот подход, основанный на угрублении, вдохновлен подходами объединения, используемыми в сверточных нейронных сетях (CNN), и он опирается на интуицию, и можно построить иерархические GNN, которые работают с различной степенью детализации входного графа. На практике эти подходы к огрублению могут привести к высокой производительности, но они также могут быть нестабильными и трудными для обучения. Например, чтобы весь процесс обучения был дифференцируемым от начала до конца, функции кластеризации f_c должны быть дифференцируемыми, что исключает большинство готовых алгоритмов кластеризации, таких как спектральная кластеризация. Существуют также подходы, которые огрубляют граф, выбирая набор вершин для удаления, а не объединяя все вершины в кластеры, что может привести к преимуществам с точки зрения вычислительной сложности и скорости.

5.6 Обобщенная передача сообщений

До сих пор был рассмотрен наиболее популярный стиль передачи сообщений GNN, который работает в основном на уровне вершины. Однако подход передачи сообщений GNN также можно обобщить, чтобы использовать информацию на уровне границ и графа на каждом этапе передачи сообщений. Например, в

более общем подходе, определяется каждая итерация передачи сообщения в соответствии со следующими уравнениями (5.46 – 5.49) [23]:

$$h_{(u,v)}^{(k)} = UPDATE_{edge}\left(h_{(u,v)}^{(k-1)}, h_u^{(k-1)}, h_v^{(k-1)}, h_G^{(k-1)}\right) \quad (5.46)$$

$$m_{N(u)} = AGGREGATE_{node}\left(\left\{h_{(u,v)}^{(k)} \forall v \in N(u)\right\}\right) \quad (5.47)$$

$$h_u^{(k)} = UPDATE_{node}\left(h_u^{(k-1)}, m_{N(u)}, h_G^{(k-1)}\right) \quad (5.48)$$

$$h_G^{(k)} = UPDATE_{graph}\left(h_G^{(k-1)}, \left\{h_u^{(k)}, \forall v \in V\right\}, \left\{h_{(u,v)}^{(k)}, \forall (\mathrm{u}, v) \in \mathcal{E}\right\}\right) \quad (5.49)$$

Важным нововведением в этой обобщенной структуре передачи сообщений является то, что во время передачи сообщений генерируются скрытые вложения $h_{(u,v)}^{(k)}$ для каждого ребра в графе, а также вложение $h_G^{(k)}$, соответствующее всему графу. Это позволяет модели передачи сообщений легко интегрировать функции на уровне границ и графа, и недавнее исследование также показало, что этот обобщенный подход к передаче сообщений имеет преимущества по сравнению со стандартной GNN с точки зрения логической выразительности. Создание вложений для ребер и всего графа во время передачи сообщений также упрощает определение функций потерь на основе задач классификации на уровне графа или ребра.

С точки зрения операций передачи сообщений в этой обобщенной структуре мы сначала обновляем вложения ребер на основе вложений их инцидентных вершин (уравнение 5.46). Затем обновляется вложения вершин, объединяя вложения ребер для всех их инцидентных ребер (уравнения 5.47 и 5.48). Вложение графа используется в уравнении обновления для представлений вершин и ребер, а само вложение на уровне графа обновляется путем агрегирования всех вложений вершин и ребер в конце каждой итерации (уравнение 5.49).