Графовые нейронные сети

Граф - структура данных, которая представляется наборов вершин (nodes) и их отношений (edges). Исследователи уделяет все больше внимания анализу графов с помощью методов машинного обучения, потому что они обладают большой выразительность во многих задачах, таких как: анализ социальных сетей, анализ физических систем, QSAR задачах, графах знаний и тд.

Проблема анализа графов заключается в том, что это не евклидовая структура, т.е для нее нет готового векторного представления в некотором линейном пространстве. Это значит, что в исходном виде методы машинного обучения к ним неприменимы. Чтобы решить эту проблему, исследователям приходилось самим придумывать для графов некоторые векторные представления на основе отдельных признаков и структурных особенностей. Однако с достаточной полнотой и выразительностью придумать описание такой сложной структуры как граф очень сложно. С таким подходом нет никаких гарантий, что предложенное описание и используемые в нем признаки будут оптимальными для каждой конкретной задачи.

После прорыва в области компьютерного зрения, который совершили сверточные нейронные сети, исследователи решили придумать обобщение этого подхода на структуру графа. Это позволило бы нейронной сети самой извлекать нужные признаки из графа и составлять для нее лучшее векторное представление для каждой отдельной задачи. Таким образом появились графовые нейронные сети[By, 2019].

Графовые нейронные сети (GNN[Скарселли, 2009]) - методы глубокого обучение, которые работают непосредственно со структурой графа. Они получилили широкое распространение в последнее время.

Рассмотрим общий *процесс разработки* GNN[Жоу, 2018] для некоторой задачи:

1) Необходимо определить структуру графа. Обычно существует два сценария: структурные сценарии и неструктурные сценарии. В структурных сценариях структура графа явно указана в задаче. Это относится к молекулам, физическим системам, графам знаний и так далее. В неструктурных сценариях графы не указаны, поэтому мы должны сначала построить граф на основе задачи, например, построить «словесный» графа для текста или построить граф сцены для изображения.
2) Необходимо определить тип графа и его размер.

   Графы обычно классифицируют на ориентированные и неориентированные, гомогенные и гетерогенные, статические и динамические. В зависимости от типа графа может менятся выбор архитектуры сети и подходы к обучению.

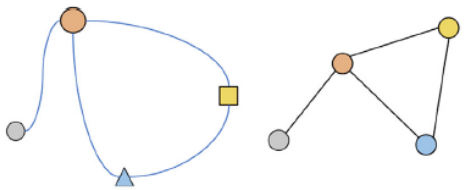
Для размера графа нет четкой классификации на “маленькие” и “большие”. Это зависит от имеющихся вычислительных ресурсов (например, объема памяти GPU). Будем считать, что граф “большой”, если его нельзя обработать на 1 GPU.

3) Выбор функции потерь. Он зависит от типа задачи и подхода к обучению.

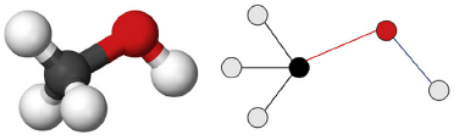
Задачи на графах обычно делятся на:

- Node-level задачи, которые фокусируются на вершинах. Например, задача классификации вершины.
- Edge-level задачи, которые включают, например, классификацию ребер или предсказание связи.
- Graph-level задачи включают классификацию графов, регрессионные задачи и тд.

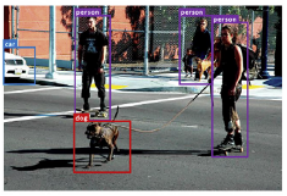
Подходы к обучению разделяются на классические подходы машинного обучения: supervised, unsupervised и semi-supervised. Примеры различных графов можно увидеть на рисунке ниже.

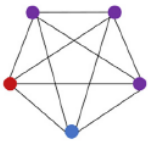


(a) Physics

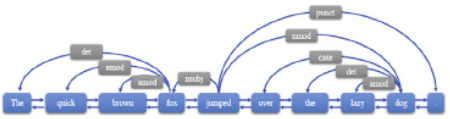


(b) Molecule

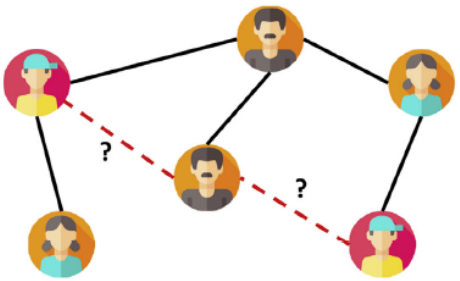




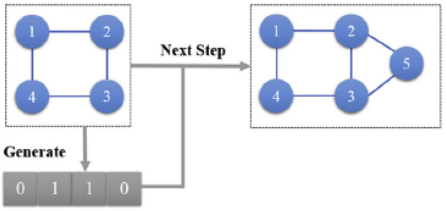
(c) Image



(d) Text



(e) Social Network



(f) Generation

4) Последний шаг - выбор архитектуры. Он происходит на основе 3х основных вычислительных компонент из которых состоят слои графовой нейронной сети.

- Propagation Module
- Sampling Module
- Pooling Module

Подробнее про архитектуры GNN мы поговорим ниже.

Введем некоторые обозначения:

$G = (V, E)$ - граф, где $|V| = N$ это число вершин в граф, а $|E| = N_e$ - число ребер.

$A \in \Re^{N \times N}$ - матрица смежности графа.

h_v - векторное представление вершины v, скрытое состояние вершины.

h^t_v - скрытое состояние вершины в момент t (на t-м слое).

e_{vw} - векторное представление ребра vw

$\odot$ - поэлементное умножение

I_N - единичная матрица размерности N

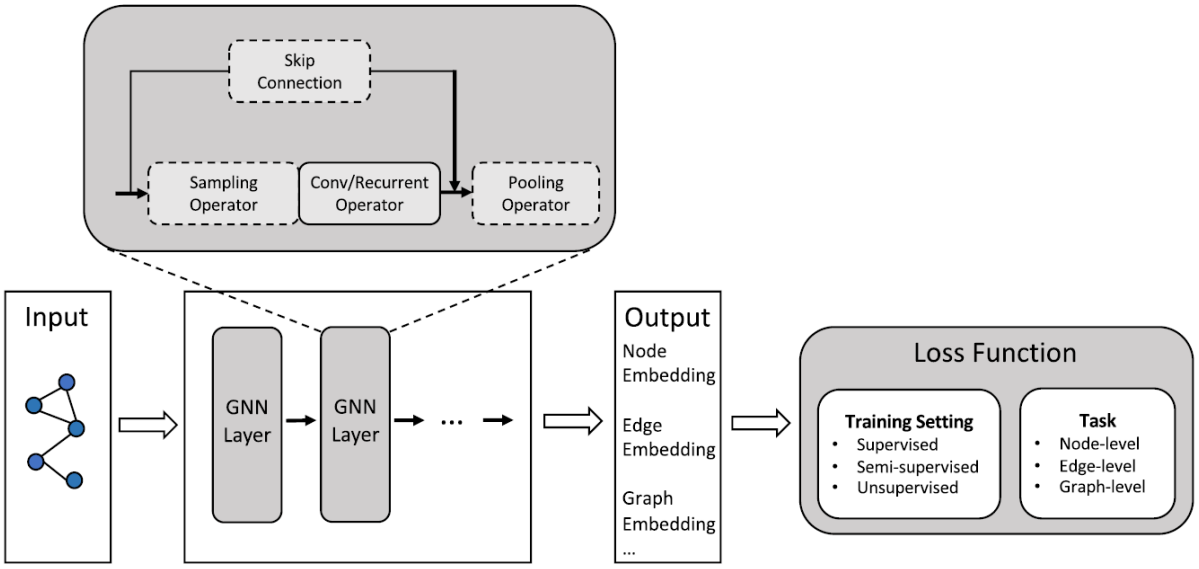
σ - функция сигмоиды

$\aleph_v$ - множество вершин, смежных v

$|x|$ - мощность множества x

|| - операция конкатенации

Общая архитектура GNN представляет из себя некоторое количество GNN слоев (GNN Layer). Эти слои принимают на вход признаки вершин графа (матрицу X размера $N \times M$, где N - число вершин, M - размерность описания каждой вершины), признаки ребер графа (матрицу E размера $N^e \times M^e$, где N^e - число ребер, M^e - размерность описания каждого ребра) и матрицу смежности A размера $N \times N$. Каждый слой, как показано на картинке ниже, состоит из 3х блоков, указанных в 4 пункте процесса разработки.



Propagation Module - модуль распространения, который используется для распространения информации между узлами, чтобы агрегировать информацию как из признаков, так и топологическую информацию графа. В этом модуле
обычно используется оператор свертки (convolution operator) или рекуррентный оператор (recurrent operator) для агрегирования информации от соседей.

Skip Connection используется для учета более ранней информации из вершин и решения проблемы затухания градиента.

Sampling Module отвечает за поведение **propagation module** в случае, если граф большой. Например, это может быть семплирование некоторого подграфа, к которому будет применяться оператор свертки.

Pooling Module используется когда нам нужны представления высокого уровня: подграфа или графа. Модули объединения необходимы для извлечения информации
из вершин и построения итогового представления. Например, это может быть суммирования векторов вершин, которое будет представлять нам описание всего графа.

После $t-$ ого GNN слоя мы получаем матрицу H^t , которая является обновленным вариантом матрицы X, т.е новым описанием наших вершин. Ее строки мы будем называть скрытыми состояниями вершин. После прохода последнего слоя мы получим итоговые векторные представления для вершин, ребер и при необходимости для всего графа (с помощью pooling module на последнем слое). Эти вектора мы используем для решения исходной задачи. Например, можно отправить такие вектора в MLP, который выдаст нам класс графа или вещественное число. Предсказанное значение отправляется в функцию потерь (подобранную под задачу) и методом градиентного спуска происходит обучение внутренних слоев GNN сети.

Теперь подробнее рассмотрим варианты **propagation module.** Имеется 2 типа модулей: сверточные и рекуррентные. Мы в основном будем говорить про сверточные модули, т.к они чаще всего используются в GNN сетях. Сверточные модули подразделяются на спектральные (spectral) и пространственные (spatial) подходы.

Спектральные подходы работают со спектральным представлением графов. Эти методы теоретически основаны на графовой обработке сигналов[Шуман, 2013] и определяют оператор свертки в спектральном пространстве.

В спектральных методах сигнал графа x сначала преобразуется в спектральное пространство преобразованием Фурье F на графе, затем выполняется свертка. После свертки результирующий сигнал преобразуется обратно с помощью обратного преобразования Фурье F^{-1} на графе. Преобразование Фурье графа имеет вид: $F(x) = U^T x$, $F^{-1}(x) = Ux$, где U - матрица собственных векторов нормализованного Лапласиана графа $L = I_N - D^{-1/2} A D^{-1/2}$. D - диагональная матрица степеней вершин.

В итоге оператор свертки матрицы g с сигналом x будет иметь вид:

$$g \star x = F^{-1}(F(g) \odot F(x)) = U(U^T g \odot U^T x),$$

что в случае диагональной матрицы g_w имеет вид $g_w \star x = U g U^T x$. g_w - это и есть фильтр или веса, которые будут обучаться в процессе обучения самой сети.

Рассмотрим несколько конкретных архитектур:

1) **ChebNet**[Хаммонд, 2009]. Идея этой архитектуры в том, чтобы аппроксимировать операцию свертки многочленами Чебышева до K-й степени включительно. Операция свертки принимает вид: $g_w \star x \approx \Sigma_{k=0}^{K} w_k T_k(\bar{L}) x$, где $\bar{L} = \frac{2}{\lambda_{max}} L - I_N$

   (λ_{max} - максимально собственное значение L.)

2) **GCN**[Кипф, 2017]. (Graph Convolution Network). Архитектура совпадает с предыдущей при выборе параметра $K = 1$. Это позволяет бороться с переобучение и ускорить процесс обучения. Операция свертки принимает вид $g_w \star x \approx w(I_N - D^{-1/2} A D^{-1/2}) x$. Для решения проблемы затухающих / взрывающихся градиентов был придуман трюк перенормировки. В предыдущей формуле делается замена $I_N - D^{-1/2} A D^{-1/2} \to \bar{D}^{-1/2} \bar{A} \bar{D}^{-1/2}$, где $\bar{A} = A + I_N$ и $\bar{D}_{ij} = \Sigma_j \bar{A}_{ij}$.

В итоге формула обновления скрытых состояний принимает вид:

$H = \overline{D}^{-1/2}\overline{A}\,\overline{D}^{-1/2}$ X W,

где $X \in \Re^{N\times M}$ - входная матрица признаков вершин (т.е N - векторов размерности M), $W \in \Re^{M\times M'}$ - матрица обучаемых параметров, $H \in \Re^{N\times M'}$ - новая матрица скрытых состояний.

Пространственные подходы (spatial) определяют свертки непосредственно на графе и основываются на его топологии. Основная проблема пространственных подходов - определение операции свертки с окрестностями разного размера (с переменным числом соседей у вершины) и поддержание локальной инвариантности как у CNN (т.е чтобы выход свертки не зависел от расположения вершины, к которой применяется свертка).

Рассмотрим несколько примеров конкретных архитектур:

1) **Neural FPs**[Дувенод, 2015]**.** Эта архитектура используем различные матрицы весов для вершин с различным числом соседей. Формула обновления скрытых состояний принимает вид:

$t = h_v^{\,t} + \Sigma_{u\in \aleph_v} h_u^{\,t}$

$h_v^{\,t+1} = \sigma(t\, W^{t+1}_{|\aleph_v|})$,

где $W^{t+1}_{|\aleph_v|}$ -матрица весов для вершин степени $|\aleph_v|$.

2) **GraphSAGE**[Хамильтон, 2017]. Общих подход применяющий семплирование и агрегацию информации из соседей для данной вершины. Формулы обновления имеют вид:

$h_{\aleph_v}^{\,t+1} = AGG_{t+1}(\{h_u^{\,t},\ \vee\ u\ \in \aleph_v\})$,

$h_v^{\,t+1} = \sigma(W^{t+1} * [h_v^{\,t} \,||\, h_{\aleph_v}^{\,t+1}])$.

Этот метод используется не все множество соседей вершины, а семплирует некоторое фиксированное количество из равномерного распределения. AGG_{t+1} - функция агрегации, которая может быть, например, средним или LSTM.

3) **GAT. (Graph Attention Network)**[Великовик, 2018]**.** Метод добавляет механизм внимания в propagation module. Т.е сначала определяется “важность” соседей, а затем обновляются скрытые состояния на основе взвешивании соседей:

$h_v^{\,t+1} = \rho(\Sigma_{u\in \aleph_v} \alpha_{vu} W h_u^{\,t})$

$$\alpha_{vu} = \frac{exp(LeakyRely(a^T[Wh_v||Wh_u]))}{\Sigma_{k\in\aleph_v} exp(LeakyRely(a^T[Wh_v||Wh_k]))}$$

где W - это матрица весов, a - вектор весов.

Это некоторые конкретные методы, которые относятся к spatial подходу. Однако их всех объединяет одна и та же идея: обновление информации внутри вершины, происходит на основе информации, которая содержится в соседних вершинах. Поэтому был предложен общий фреймворк для описания подобных архитектур, который называется MPNN - Message Passing Neural Network[Гилмер, 2017].

Такие модели содержат 2 фазы: фаза передачи сообщения и фаза считывания. В первую фазу используется функция M_t, которая агрегирует информацию от соседей и делает из него "сообщение" m_v^t. Затем используется функция U_t, которая обновляет скрытое состояние вершины h_v^t:

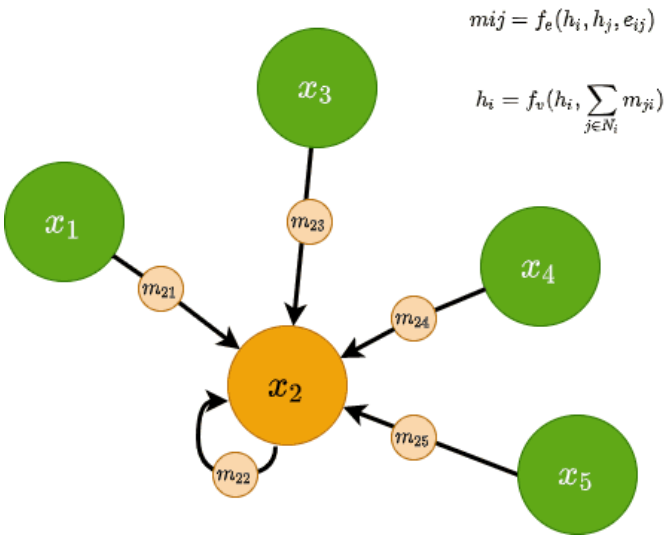
$$m_v^{t+1} = \Sigma_{u\in\aleph_v} M_t(h_v^t, h_u^t, e_{vu})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}).$$

На фазе считывания применяется функция R, которая вычисляет итоговое значение, требующееся в задаче (это может быть класс самого графа или, например, вещественное число связанное с каждой отдельной вершиной):

$$\hat{y} = R(\{h_v^t \mid v \in G\}).$$

Для определения архитектуры разработчику необходимо лишь определить функции M_t, U_t, R. Общая идея spatial подходов изображена на рисунке ниже.



Таким образом мы определили общий подход к построению архитектуры графовой сети, рассмотрели различные варианты построения ее внутренних слоев. Обучение таких сетей происходит за счет классического алгоритма обучения нейронных сетей - градиентного спуска. Графовые нейронные сети - это множество алгоритмов, которые появились сравнительно недавно. Они активно набирают популярность среди исследователей, которые работают с графовыми структурами. За последнее время с помощью них было получено много интересных результатов[1], но при этом все равно остается множество направлений для исследования в этой теме.

[1] Список статей по этой теме можно найти по ссылке: https://github.com/thunlp/GNNPapers

Список использованной литературы:

1) *Velickovic P., Cucurull G., Casanova A., Romero A., Lio P., Bengio Y.* Graph attention networks, 2018. URL: https://arxiv.org/abs/1710.10903. (Дата обращения: 05.12.2021).
2) *Duvenaud D.K., Maclaurin D., Aguileraiparraguirre J., Gomezbombarelli R., Hirzel T.D., Aspuruguzik A., Adams R.P.* Convolutional networks on graphs for learning molecular fingerprints, 2015. URL: https://arxiv.org/abs/1509.09292. (Дата обращения: 05.12.2021).
3) *Gilmer J., Schoenholz S.S., Riley P.F., Vinyals O., Dahl G.E.* Neural message passing for quantum chemistry // International conference on machine learning, 2017. Сс. 1263–1272. URL: https://arxiv.org/abs/1704.01212 (Дата обращения: 05.12.2021).
4) *Hamilton W.L., Ying Z., Leskovec J.* Inductive representation learning on large graphs, 2017. Сс. 1024–1034. URL: https://arxiv.org/pdf/1706.02216.pdf. (Дата обращения: 05.12.2021).
5) *Hammond D.K., Vandergheynst P., Gribonval R.* Wavelets on graphs via spectral graph theory, 2009. Сс. 129–150. URL: https://arxiv.org/pdf/0912.3848.pdf. (Дата обращения: 05.12.2021).
6) *Kipf T.N., Welling M.* Semi-supervised classification with graph convolutional networks, 2017. URL: https://arxiv.org/abs/1609.02907. (Дата обращения: 05.12.2021).
7) *Scarselli F., Gori M., Tsoi A., Hagenbuchner M., Monfardini G.* The graph neural network model // IEEE Transactions on Neural Networks, т. 20, № 1, 2009. Сс. 61-80. URL: https://persagen.com/files/misc/scarselli2009graph.pdf. (Дата обращения: 05.12.2021).
8) *Shuman D.I., Narang S.K., Frossard P., Ortega A., Vandergheynst P.* The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains // IEEE SPM, 30, 2013. Сс. 83–98. URL: https://arxiv.org/abs/1211.0053. (Дата обращения: 05.12.2021).
9) *Wu Z., Pan S., Chen F., Long G., Zhang C., Yu P.S.* A Comprehensive Survey on Graph Neural Networks, 2019. URL: https://arxiv.org/abs/1901.00596. (Дата обращения: 05.12.2021).
10) *Zhou J., Cui G., Zhang Z., Yang C., Liu Z., Sun M.* Graph Neural Networks: A Review of Methods and Applications, 2018. URL: https://arxiv.org/abs/1812.08434. (Дата обращения: 05.12.2021).