Нейронные сети на графах

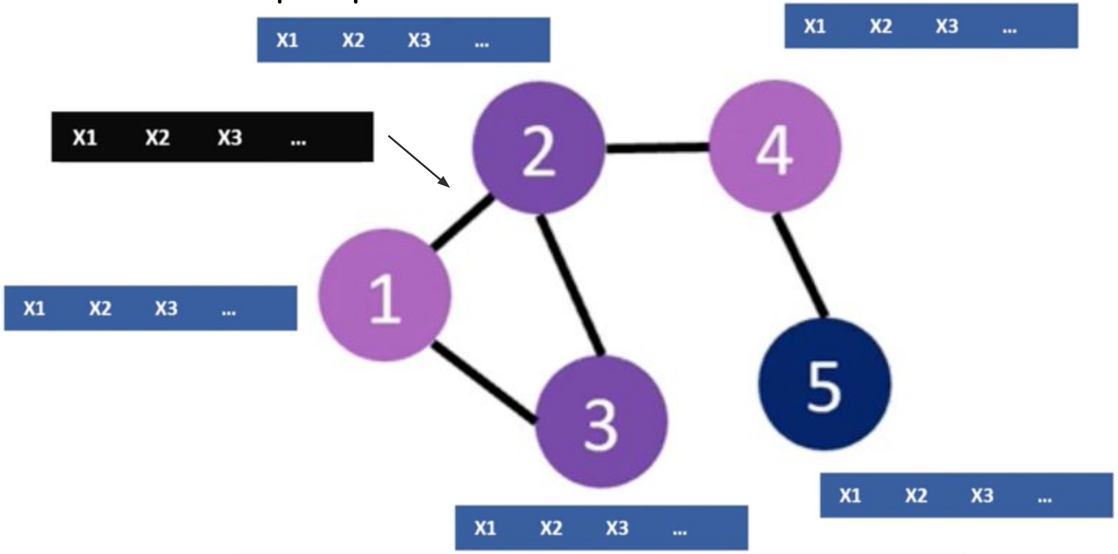
Алекберов Артём, Седашов Данила, 181

Часть 1.

Определения и задачи

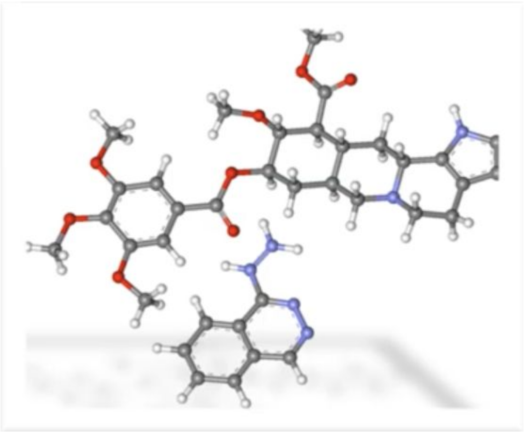
Что такое граф?

- Набор вершин и рёбер $G = (V, E)$
- В памяти представляется матрицей смежности размера $V \times V$
- У каждой вершины и ребра есть набор признаков



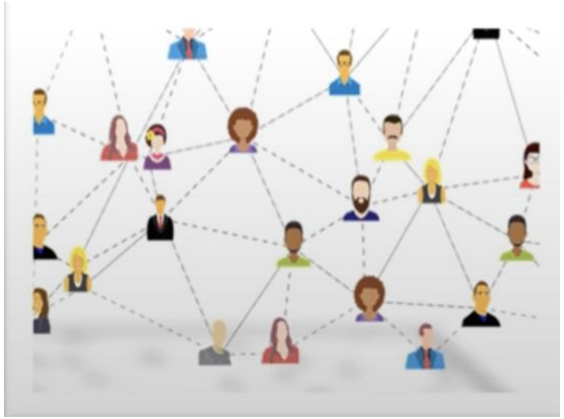
	V1	V2	...
V1	0	1	...
V2	1	0	...
V3	1	1	...
...	...	...	...

Примеры графовых данных



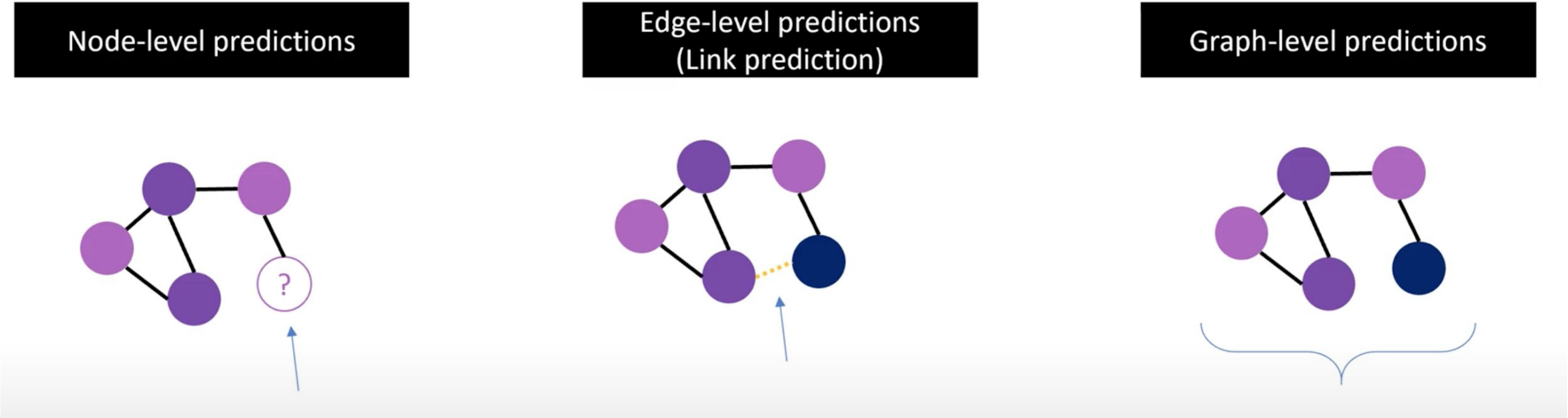
Химия / медицина

Рекомендательные системы



Социальные сети

Обзор задач



Курит ли этот человек?

Следующее видео на YouTube?

Подходит ли эта молекула для лекарства?

Часть 2.

Рекуррентные нейронные сети на графах

Проблемы графовых данных

- Размерности входов могут меняться (количество вершин)
- Существуют изоморфные графы => модель должна быть инвариантна к ним
- Графы не лежат в евклидовом пространстве

Проблемы графовых данных

- Размерности входов могут меняться (количество вершин)
- Существуют изоморфные графы => модель должна быть инвариантна к ним
- Графы не лежат в евклидовом пространстве

Решение: будем обучать эмбеддинги для каждой вершины и потом при необходимости агрегировать их в эмбеддинг графа

Общий алгоритм

Пусть $x_v \in \mathbb{R}^d$ — вектор признаков вершины v,

$x^e_{(v,u)} \in \mathbb{R}^c$ — вектор признаков ребра, соединяющего вершины u, v.

Также пусть $N(v)$ — множество соседей вершины v.

Общий алгоритм

Пусть $x_v \in \mathbb{R}^d$ — вектор признаков вершины v,

$x^e_{(v,u)} \in \mathbb{R}^c$ — вектор признаков ребра, соединяющего вершины u, v.

Также пусть $N(v)$ — множество соседей вершины v.
Будем обновлять эмбеддинги вершин итеративно.
Пусть $h_v^{(t)}$ — эмбеддинг вершины v на t-ой итерации.

Общий алгоритм обновления эмбеддингов: для каждой из вершин делаем преобразование:

$$\mathbf{h}_v^{(t)} = \sum_{u \in N(v)} f(\mathbf{x}_v, \mathbf{x^e}_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^{(t-1)})$$

пока не достигнем точки равновесия ($f(\cdot)$ — параметрическая ф-ия)

Теорема Банаха о неподвижной точке

Пусть $(\mathbb{X}, d)$ — непустое метрическое пространство,

$\boldsymbol{T : \mathbb{X} \to \mathbb{X}}$ — сжимающее отображение
(то есть $\exists\, 0 \leq \alpha < 1 : d(Tx, Ty) \leq \alpha d(x, y) \;\forall\, x, y \in \mathbb{X}$)

Теорема Банаха о неподвижной точке

Пусть $(\mathbb{X}, d)$ — непустое метрическое пространство,

$T : \mathbb{X} \to \mathbb{X}$ — сжимающее отображение
(то есть $\exists\, 0 \leq \alpha < 1 : d(Tx, Ty) \leq \alpha d(x, y) \;\forall\, x, y \in \mathbb{X}$)

Тогда:

1) существует и единственна неподвижная точка $x^* \in \mathbb{X} : Tx^* = x^*$
2) итерационная последовательность $x, Tx, T^2x, \ldots$

сходится к этой точке с экспоненциальной скоростью

Теорема Банаха о неподвижной точке

Пусть $(\mathbb{X}, d)$ — непустое метрическое пространство,

$T : \mathbb{X} \to \mathbb{X}$ — сжимающее отображение
(то есть $\exists\, 0 \le \alpha < 1 : d(Tx, Ty) \le \alpha d(x, y)\ \forall\, x, y \in \mathbb{X}$)

Тогда:
1) существует и единственна неподвижная точка $x^* \in \mathbb{X} : Tx^* = x^*$
2) итерационная последовательность $x, Tx, T^2x, \ldots$
сходится к этой точке с экспоненциальной скоростью

Пусть F_w — глобальная функция преобразования (которая переводит множество всех эмбеддингов в себя)
Таким образом, при условии, что F_w — сжимающее отображение, алгоритм обновления эмбеддингов сойдётся

Общий алгоритм: продолжение

- Введём также параметрическую функцию $g(\cdot;)$ которая принимает на вход эмбеддинги и формирует ответ
- Через эту функцию можем определить функционал ошибки $\mathcal{L}$ (например, MSE для регрессии или кросс-энтропию для классификации)

Общий алгоритм: продолжение

- Введём также параметрическую функцию $g(\cdot;)$ которая принимает на вход эмбеддинги и формирует ответ
- Через эту функцию можем определить функционал ошибки $\mathcal{L}$ (например, MSE для регрессии или кросс-энтропию для классификации)
- Таким образом, алгоритм обучения следующий:
  - итеративно обновляем $h_v^{(t)}$ для каждой вершины, пока не достигнем точки равновесия ($h_v^{(0)}$ инициализируем случайно)
  - считаем градиент функционала по параметрам f и g: $\frac{\partial \mathcal{L}}{\partial w}$
  - делаем шаг градиентного спуска

Реализации f и g

- На g не накладывается никаких ограничений => g = MLP
- F_w должна быть сжимающим отображением

Linear GNN

- Будем считать $f(x_v, x^e_{(v,u)}, x_u, h_u^{(t-1)}) = A_{n,u} h_u^{(t-1)} + b_n$

где $A_{n,u} \in \mathbb{R}^{s \times s}, b_n \in \mathbb{R}^s$(s — размер эмбеддинга)

Linear GNN

- Будем считать $f(x_v, x^e_{(v,u)}, x_u, h_u^{(t-1)}) = A_{n,u} h_u^{(t-1)} + b_n$

где $A_{n,u} \in \mathbb{R}^{s \times s}, b_n \in \mathbb{R}^s$(s — размер эмбеддинга)

- $A_{n,u}, b_n$ генерируются при помощи двух MLP
- $A_{n,u} = \frac{\mu}{s \cdot |N(u)|} \cdot resize(MLP_1(x_v, x^e_{(v,u)}), x_u)$, $\mu \in (0, 1)$
- $b_n = MLP_2(x_v)$

Linear GNN

- Будем считать $f(x_v, x^e_{(v,u)}, x_u, h_u^{(t-1)}) = A_{n,u} h_u^{(t-1)} + b_n$

где $A_{n,u} \in \mathbb{R}^{s \times s}, b_n \in \mathbb{R}^s$(s — размер эмбеддинга)

- $A_{n,u}, b_n$ генерируются при помощи двух MLP
- $A_{n,u} = \frac{\mu}{s \cdot |N(u)|} \cdot resize(MLP_1(x_v, x^e_{(v,u)}), x_u),$ $\mu \in (0, 1)$
- $b_n = MLP_2(x_v)$

При условии, что используются подходящие функции активации (например, tanh), несложно доказывается, что итоговая F_w — сжимающее отображение

Non-linear GNN

- $f(\cdot)$ — просто MLP
- Необходимо, чтобы выполнялось свойство сжимающего отображения

Non-linear GNN

- $f(\cdot)$ — просто MLP
- Необходимо, чтобы выполнялось свойство сжимающего отображения
- Это решается добавлением к функционалу ошибки регуляризации Якобиана: $\mathcal{L} = \ldots + \beta L(||\frac{\partial F_w}{\partial x}||)$

$$L(y) = \begin{cases} (y-\mu)^2, y > \mu \\ 0, otherwise \end{cases} \qquad \mu \in (0,1)$$

Non-linear GNN

- $f(\cdot)$ — просто MLP
- Необходимо, чтобы выполнялось свойство сжимающего отображения
- Это решается добавлением к функционалу ошибки регуляризации Якобиана: $\mathcal{L} = \ldots + \beta L(||\frac{\partial F_w}{\partial x}||)$

$$L(y) = \begin{cases} (y-\mu)^2, y > \mu \\ 0, otherwise \end{cases} \qquad \mu \in (0,1)$$

В общем случае регуляризатор может быть любым выражением, дифференцируемым по **w** и монотонно возрастающим по норме Якобиана

В обеих моделях для оптимизации вычислений используется **Almeida–Pineda recurrent backpropagation**

Результаты

- Mutagenesis dataset (классификация молекул)
- Молекула = граф, в каждом графе одна вершина является помеченной и по ней формируется ответ

Method	Knowledge	Reference	Accuracy
non–linear GNN	AB+C+PS		90.5%
$1nn(d_m)$	AB	[87]	81%
$1nn(d_m)$	AB+C	[87]	88%
TILDE	AB	[92]	77%
TILDE	AB+C	[92]	82%
RDBC	AB	[88]	83%
RDBC	AB+C	[88]	82%

Gated graph neural network

- Предыдущие модели могут не очень хорошо работать на больших графах
- Применим GRU

$$\mathbf{h}_v^{(1)} = [\boldsymbol{x}_v^\top, \mathbf{0}]^\top \tag{1}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \tag{2}$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \tag{3}$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \tag{4}$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \tag{5}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \tag{6}$$

Часть 3.

Сверточные нейронные сети на графах

Spectral convolutions

Лапласиан. Определение

Рассмотрим неориентированный граф
его матрицу смежности A и матрицу степеней D

Тогда лапласиан графа L определяется так:

$$L = D - A \in \mathbb{R}^{n \times n}$$

$$L_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

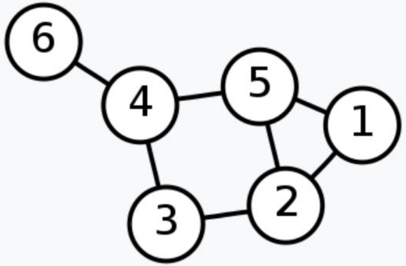
Лапласиан. Определение

Нормализованный Лапласиан

$$L^{\text{sym}} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

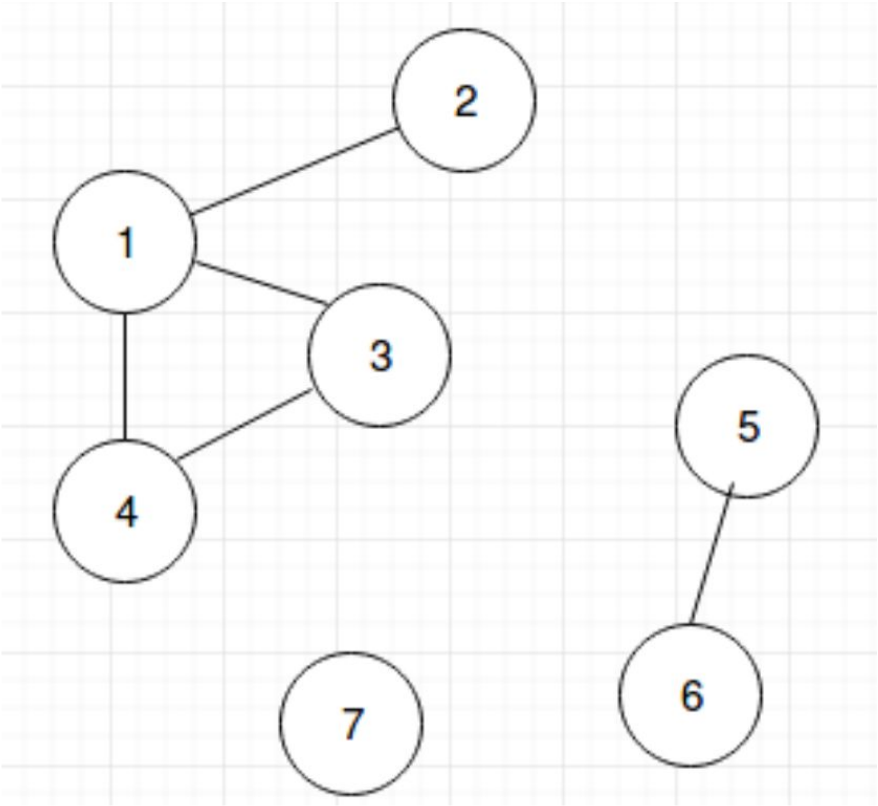
$$L^{\text{sym}}_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

Пример

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Свойства

- Лапласиан - симметричная матрица
- Лапласиан - неотрицательно определенная матрица
- Сумма каждой строки/столбца = 1
- Собственное значение λ_0 = 0 т.к. v = (1, 1, ..., 1) - собственный вектор
- число собственных значений, равных 0, = числу компонент графа




```
[ 3, -1, -1, -1,  0,  0,  0]
[-1,  1,  0,  0,  0,  0,  0]
[-1,  0,  2, -1,  0,  0,  0]
[-1,  0, -1,  2,  0,  0,  0]
[ 0,  0,  0,  0,  1, -1,  0]
[ 0,  0,  0,  0, -1,  1,  0]
[ 0,  0,  0,  0,  0,  0,  0]
```

Преобразование Фурье графа

- Определим сигнал на графе как $f : V \to \mathbb{R}$
- Преобразование фурье от функции f станет функцией от спектра
- Пусть λ_l - l-ое собственное значение Лапласиана, μ_l - l-ый собственный вектор Лапласиана
  Тогда

$$\mathcal{GF}[f](\lambda_l) = \hat{f}\,(\lambda_l) = \langle f, \mu_l \rangle = \sum_{i=1}^{N} f(i)\mu_l^*(i),$$

where $\mu_l^* = \mu_l^{\mathrm{T}}$.

Обратное преобразование Фурье графа

$$\mathcal{IGF}[\hat{f}\,](i) = f(i) = \sum_{l=0}^{N-1} \hat{f}\,(\lambda_l)\,\mu_l(i)$$

Преобразование Фурье графа

- Т.к. Лапласиан раскладывается в следующем виде

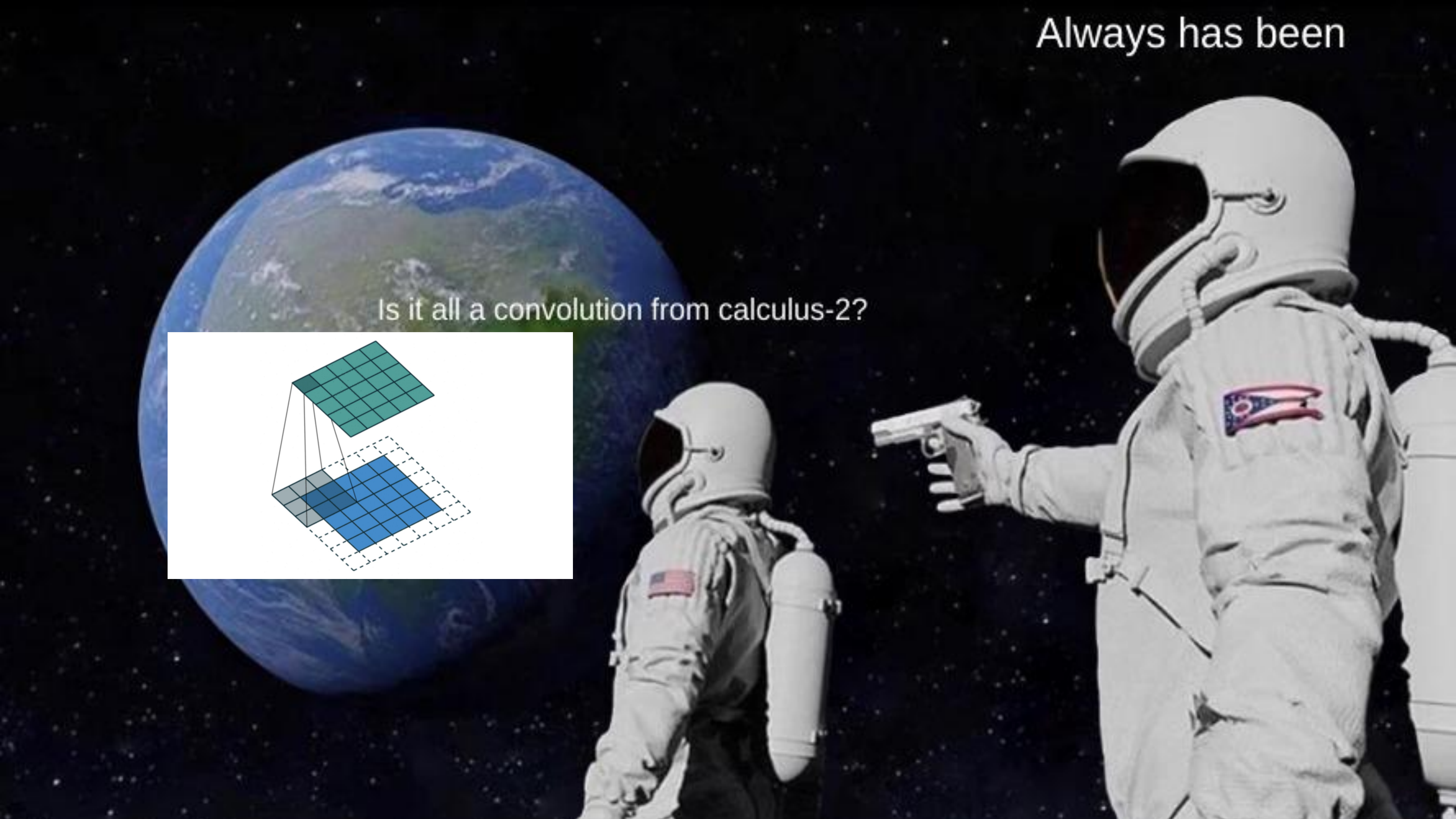
$$\mathcal{L} = U \Lambda U^{-1} = U \Lambda U^T$$

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} = diag([\lambda_1, ..., \lambda_n]) \in \mathbb{R}^{n \times n}$$

- То преобразование Фурье можно определить как $\hat{f} = U^T f$

Свертка в мат. анализе

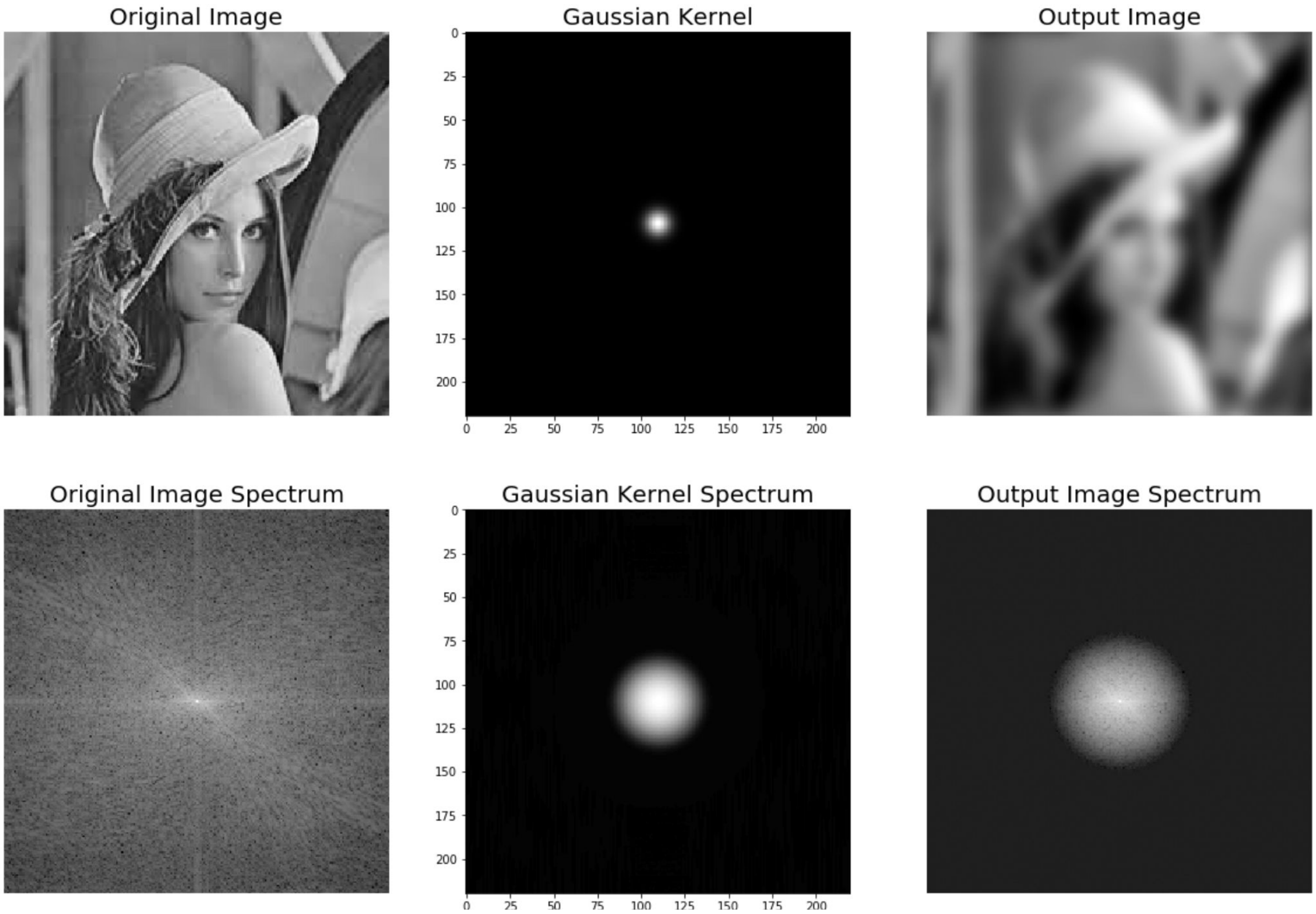
$$(f * g)(x) \overset{\text{def}}{=} \int\limits_{\mathbb{R}^n} f(y)\, g(x-y)\, dy = \int\limits_{\mathbb{R}^n} f(x-y)\, g(y)\, dy.$$


Always has been
Is it all a convolution from calculus-2?

Теорема о свертке

Пусть F - преобразование Фурье, тогда

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$



Original Image
Gaussian Kernel
Output Image
Original Image Spectrum
Gaussian Kernel Spectrum
Output Image Spectrum
0
25
50
75
100
125
150
175
200

Spectral graph convolution

Пользуясь теоремой, если f - сигнал, g - ядро свертки, то определим свертку как

$$g(\cdot_G)f = \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(f)) = U(U^T g \odot U^T f)$$

$$= U g_\theta(\Lambda) U^T f = g_\theta(\mathcal{L}) f$$

где $g_\theta(\Lambda) = diag(U^T g)$ (делаем из вектора матрицу с этим вектором на диагонали)

Spectral CNN

Теперь введем сигнал как функцию $\boldsymbol{f : V} \rightarrow \mathbb{R}^{\boldsymbol{C_k}}$

И определим сверточный слой нейросети так:

$$f_j^{(k+1)} = \sigma(U \sum_{i=1}^{C_k} g_{\theta}{}_{i,j}^{(k)} U^T f_i^{(k)}) = \sigma(U \sum_{i=1}^{C_k} g_{\theta}{}_{i,j}^{(k)} \hat{f}_i^{(k)})$$

$$g_{\theta}{}_{i,j}^{(k)} = \begin{bmatrix} \theta_1^{(k)} & & & \\ & \theta_2^{(k)} & & \\ & & \ddots & \\ & & & \theta_n^{(k)} \end{bmatrix}$$

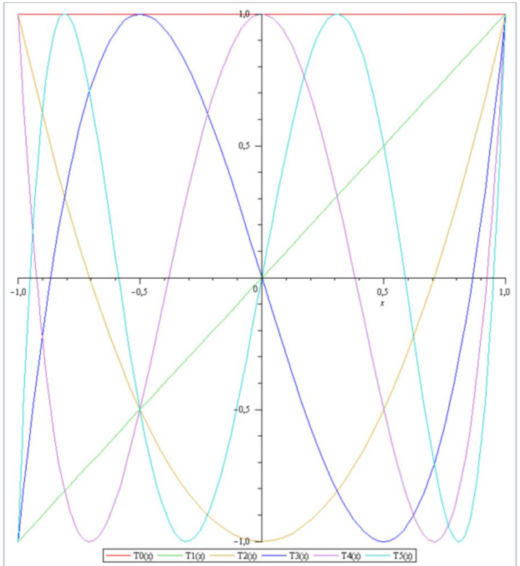
Проблемы

- число параметров зависит от числа вершин
- смена индексации вершин приводит к смене Лапласиана, а значит и базиса
- разложение Лапласиана - $O(n^3)$
- фильтр не является локальным
(каждый раз у свертки n параметров)

Решение - ChebNet!

Многочлены Чебышева

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$



Свойства многочленов Чебышева

Свойств довольно много, но самое главное, что

- ортогональность, скалярное произведение - $(f, g) = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x) g(x) dx$
- используются для приближения функций (перед этим надо перевести функцию в интервал ортогональности многочленов - [-1, 1])

Spectral graph convolution. Напоминание

Если f - сигнал, g - ядро свертки, то определим свертку как

$$g(\cdot_G)f = \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(f)) = U(U^T g \odot U^T f)$$

$$= U g_\theta(\Lambda) U^T f = g_\theta(\mathcal{L}) f$$

где $g_\theta(\Lambda) = diag(U^T g)$ (делаем из вектора матрицу с этим вектором на диагонали)

Приближение многочленами Чебышева

Диагональной формы:

1. $\widetilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_n \in [-1, 1]$
2. $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\Lambda})$

Лапласиана:

1. $\tilde{\mathcal{L}} = 2\mathcal{L}/\lambda_{max} - I_n$
2. $g_\theta(\mathcal{L})f = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathcal{L}})f$

Приближение многочленами Чебышева

$$g_\theta(\Lambda) = \begin{bmatrix} \hat{g}(\lambda_1) & & & \\ & \hat{g}(\lambda_2) & & \\ & & \ddots & \\ & & & \hat{g}(\lambda_n) \end{bmatrix} =$$

$$\begin{bmatrix} \sum_{k=0}^{K-1} \theta_k T_k(\hat{\lambda_1}) & & & \\ & \sum_{k=0}^{K-1} \theta_k T_k(\hat{\lambda_2}) & & \\ & & \ddots & \\ & & & \sum_{k=0}^{K-1} \theta_k T_k(\hat{\lambda_n}) \end{bmatrix}$$

where θ_k is a vector of Chebyshev coefficients, which is trainable parameter.

Приближение многочленами Чебышева

$$f(\cdot_G)g_\theta = g_\theta(U\Lambda U^T)f = U\sum_{k=0}^{K-1}\theta_k T_k(\widetilde{\Lambda})U^T f = \sum_{k=0}^{K-1}U\theta_k T_k(\widetilde{\Lambda})U^T f$$

$$= \sum_{k=0}^{K-1}U\theta_k(\sum_{c=0}^{k}\alpha_{kc}\widetilde{\Lambda}^k)U^T f = \sum_{k=0}^{K-1}\theta_k(\sum_{c=0}^{k}\alpha_{kc}U\widetilde{\Lambda}^k U^T)f$$

$$= \sum_{k=0}^{K}\theta_k(\sum_{c=0}^{k}\alpha_{kc}(U\widetilde{\Lambda}U^T)^k)f = \sum_{k=0}^{K-1}\theta_k T_k(U\widetilde{\Lambda}U^T)f = \sum_{k=0}^{K-1}\theta_k T_k(\widetilde{\mathcal{L}})f$$

Какие преимущества мы получили?

- не надо искать собственные значения Лапласиана
- количество параметров не зависит от размера графа
- фильтры стали K-локализованными

Попробуем сделать еще меньше параметров...

Graph Convolutional Network

- зафиксируем K = 1 в ChebNet
- будем считать, что λ_{max} ~ 2

Тогда

$$f(\cdot_G)g = \sum_{k=0}^{1} \theta_k T_k(\widetilde{\mathcal{L}})f = \theta_0 T_0(\widetilde{\mathcal{L}})f + \theta_1 T_1(\widetilde{\mathcal{L}})f$$

Graph Convolutional Network

- чтобы еще больше уменьшить количество параметров, предположим, что $\theta_0 = -\theta_1 = \theta$
- Тогда

$$f(\cdot_G)g \approx (\theta_0 + \theta_1(\mathcal{L} - I_n))f = \theta_0 - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = (\theta(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} + I_n))f$$

θ это просто одно число

Улучшаем стабильность

- добавляем петли: $\widetilde{A} = A + I_n$
- пересчитываем матрицу степеней: $\widetilde{D}_{i,i} = \sum_{j=1}^{n} \widetilde{A}_{i,j}$

- в итоге:

$$f(\cdot_G)g = \theta \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} f$$

Пример

- semi-supervised классификация вершин с помощью двух слоев GCN:

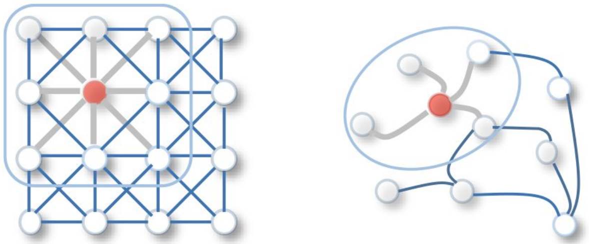
$$Z = softmax(\hat{A}ReLU(\hat{A}fW^{<0>})W^{<1>}) \tag{48}$$

$W^{<0>} \in \mathbb{R}^{C \times H}$ is an input-to-hidden weight matrix for a hidden layer with H feature maps. $W^{<1>} \in \mathbb{R}^{H \times F}$ is a hidden-to-output weight matrix, F is the dimension of feature maps in the output layer.

Spatial convolutions

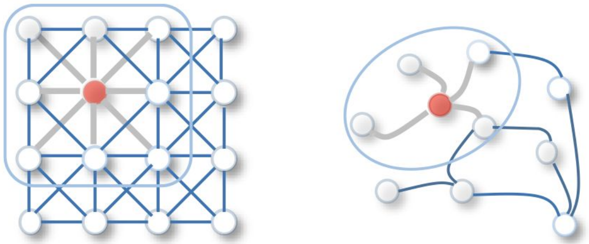
NN4G

- Основная идея везде одна: определим свёрточный слой по аналогии с изображениями



NN4G

- Основная идея везде одна: определим свёрточный слой по аналогии с изображениями
- Выполним свёртку, просуммировав информацию со всех соседей и добавив skip connections
- Получаем формулу обновления эмбеддинга на k-ом шаге:



$$\mathbf{h}_v^{(k)} = f\left(\mathbf{W}^{(k)^T}\mathbf{x}_v + \sum_{i=1}^{k-1}\sum_{u\in N(v)} \mathbf{\Theta}^{(k)^T}\mathbf{h}_u^{(k-1)}\right)$$

Здесь f — функция активации, эмбеддинг на первой итерации нулевой

NN4G

- Выражение с предыдущего слайда можно переписать в матричной форме:

$$\mathbf{H}^{(k)} = f(\mathbf{X}\mathbf{W}^{(k)} + \sum_{i=1}^{k-1} \mathbf{A}\mathbf{H}^{(k-1)}\mathbf{\Theta}^{(k)})$$

Матрица смежности не нормирована, поэтому могут возникнуть проблемы с разными порядками эмбеддингов разных вершин

Diffusion convolutional neural network

- Предполагается, что информация передаётся от одной вершины к другой с некоторой вероятностью, так что за какое-то количество итераций система достигнет равновесия:

$$\mathbf{H}^{(k)} = f(\mathbf{W}^{(k)} \odot \mathbf{P}^k \mathbf{X})$$

Здесь $P = D^{-1}A \in \mathbb{R}^{n \times n}$ — матрица вероятностей передачи информации

Diffusion convolutional neural network

- Предполагается, что информация передаётся от одной вершины к другой с некоторой вероятностью, так что за какое-то количество итераций система достигнет равновесия:

$$\mathbf{H}^{(k)} = f(\mathbf{W}^{(k)} \odot \mathbf{P}^k\mathbf{X})$$

Здесь $P = D^{-1}A \in \mathbb{R}^{n\times n}$ тностей передачи информации

Отметим, что на каждом шаге эмбеддинг того же размера, что и вектор признаков, и не зависит от предыдущего эмбеддинга. Поэтому в конце матрицы эмбеддингов со всех слоёв стакаются

GraphSage

Количество соседей вершины может быть огромным, поэтому не эффективно рассматривать их всех. Предлагается выбрать случайное подмножество фиксированного размера:

$$\mathbf{h}_v^{(k)} = \sigma(\mathbf{W}^{(k)} \cdot f_k(\mathbf{h}_v^{(k-1)}, \{\mathbf{h}_u^{(k-1)}, \forall u \in S_{\mathcal{N}(v)}\})), \quad (24)$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $f_k(\cdot)$ is an aggregation function, $S_{\mathcal{N}(v)}$ is a random sample of the node v's neighbors. The aggregation function should be invariant to the permutations of node orderings such as a mean, sum or max function.

Graph attention network

До этого мы полагали, что у всех соседей одинаковый вес, хотя это может быть не так. Попробуем применить attention:

$$\mathbf{h}_v^{(k)} = \sigma\Big(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}\Big),$$

Веса вычисляются как:

$$\alpha_{vu}^{(k)} = softmax(g(\mathbf{a}^T [\mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)} || \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}))$$

где $g(\cdot) - LeakyReLU, a^T$ — обучаемый вектор параметров,
|| обозначает конкатенацию векторов

Spectral vs Spatial

- Спектральные модели имеют хорошее теоретическое обоснование
- Спектральные модели менее эффективны: им нужно либо вычислять собственные векторы, либо обрабатывать весь граф единовременно
- Пространственные модели более применимы к большим графам, так как вычисления могут производиться батчами
- Спектральные модели имеют плохую обобщающую способность на новые графы
- Спектральным моделям нужны неориентированные графы, пространственные могут работать с любыми

Часть 3.

Graph Pooling

Что мы умеем?

Что мы умеем?

Получать векторные представления вершин

Что мы умеем?

Получать векторные представления вершин

Но как тогда делать классификацию графов?...

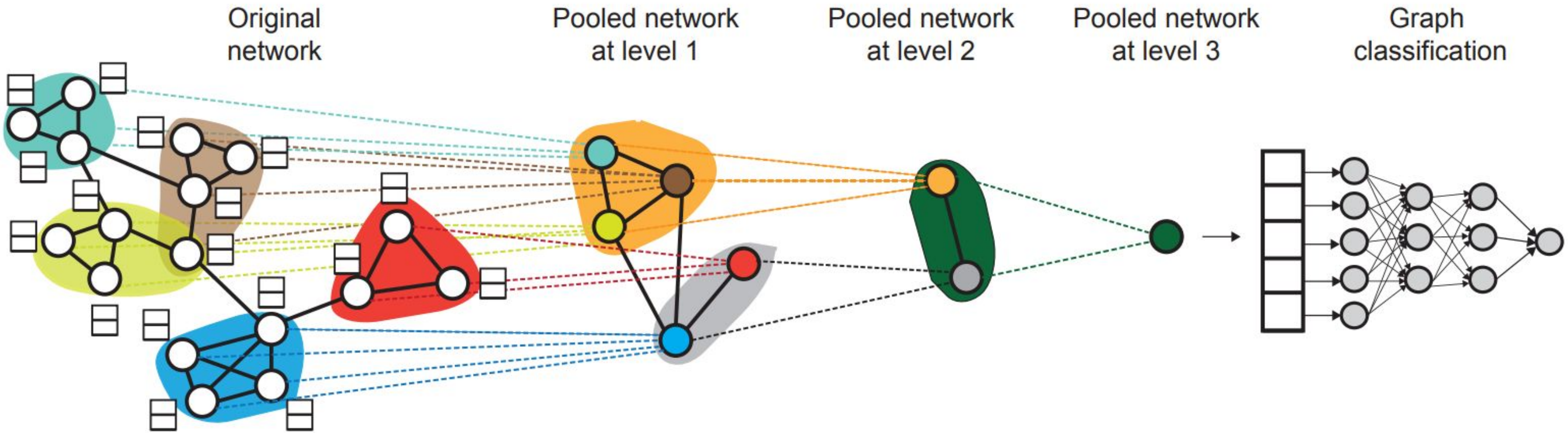
Что мы умеем?

Получать векторные представления вершин

Но как тогда делать классификацию графов?...

Надо научиться агрегировать информацию с вершин

Pipeline



Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

Самое простое

$$\mathbf{h}_G = mean/max/sum(\mathbf{h}_1^{(K)}, \mathbf{h}_2^{(K)}, ..., \mathbf{h}_n^{(K)})$$

GRACLUS

- быстрая кластеризация без подсчета собственных значений
- основано на расширении kernel k-means на графы

Pooling with GRACLUS

- с помощью GRACLUS получаем матрицу кластеризации

$\boldsymbol{S}^{(l)} \in \{0,1\}^{n_{l-1} \times n_l}$ mapping each node to its cluster index in $\{1, \ldots, n_l\}$, with $n_l < n_{l-1}$ clusters.

- делаем простой max/average пуллинг
- пересчитываем матрицу смежности

$$\boldsymbol{A}^{(l)} = \boldsymbol{S}^{(l)\top} \boldsymbol{A}^{(l-1)} \boldsymbol{S}^{(l)}$$

Pooling with GRACLUS

- не может выучить специфичные для данного набора графов структура из-за фиксированного алгоритм (хочется выучить кластеризацию в end-to-end подходе)
- выдает бинарную матрицу (хочется понимать влияние каждой вершины на каждую)
- на большим графах все равно долго

DIFFPOOL

- попробуем выдавать кластеризацию (вероятности) с помощью еще одной GNN:

$$\mathbf{S}^{(l)} = \mathrm{softmax}\left(\mathrm{GNN}_1^{(l)}(\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)})\right)$$

- пересчитываем векторные представления

$$\boldsymbol{X}^{(l)} = \boldsymbol{S}^{(l)\top}\mathrm{GNN}_2^{(l)}(\boldsymbol{A}^{(l-1)}, \boldsymbol{X}^{(l-1)}) \qquad \text{and} \qquad \boldsymbol{A}^{(l)} = \boldsymbol{S}^{(l)\top}\boldsymbol{A}^{(l-1)}\boldsymbol{S}^{(l)}.$$

DIFFPOOL

Какие лоссы использовать в качестве мотивации хорошо кластеризовать?

- $\sum_l \|A^{(l)} - S^{(l)} S^{(l)\mathsf{T}}\|$ (вершины рядом должны принадлежать одному кластеру)

DIFFPOOL

Какие лоссы использовать в качестве мотивации хорошо кластеризовать?

- $\sum_l \|\boldsymbol{A}^{(l)} - \boldsymbol{S}^{(l)} \boldsymbol{S}^{(l)\mathsf{T}}\|$ (вершины рядом должны принадлежать одному кластеру)
- $L_{\mathrm{E}} = \frac{1}{n} \sum_{i=1}^{n} H(S_i)$ (тянем кластеризацию к более вырожденному one-hot вектору)

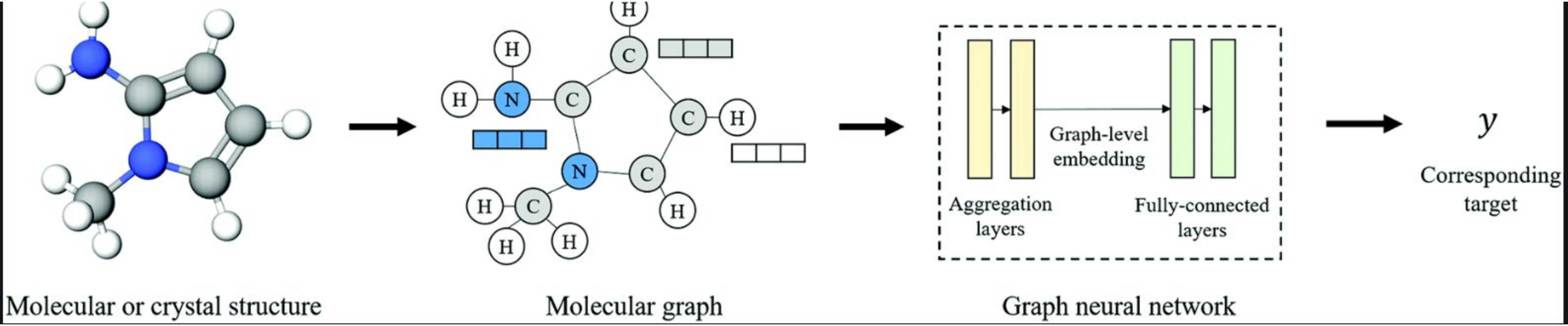
	Method	Data Set					
		ENZYMES	D&D	REDDIT-MULTI-12K	COLLAB	PROTEINS	Gain
Kernel	GRAPHLET	41.03	74.85	21.73	64.66	72.91	
	SHORTEST-PATH	42.32	78.86	36.93	59.10	76.43	
	1-WL	53.43	74.02	39.03	78.61	73.76	
	WL-OA	60.13	79.04	44.38	80.74	75.26	
GNN	PATCHYSAN	–	76.27	41.32	72.60	75.00	4.17
	GRAPHSAGE	54.25	75.42	42.24	68.25	70.48	–
	ECC	53.50	74.10	41.73	67.79	72.65	0.11
	SET2SET	60.15	78.12	43.49	71.75	74.29	3.32
	SORTPOOL	57.12	79.37	41.82	73.76	75.54	3.39
	DIFFPOOL-DET	58.33	75.47	46.18	**82.13**	75.62	5.42
	DIFFPOOL-NOLP	61.95	79.98	46.65	75.58	76.22	5.95
	DIFFPOOL	**62.53**	**80.64**	**47.08**	75.48	**76.25**	**6.27**

Часть 4.

Applications

Химия

- предсказания какие-то свойства молекул
- генерация новых молекул

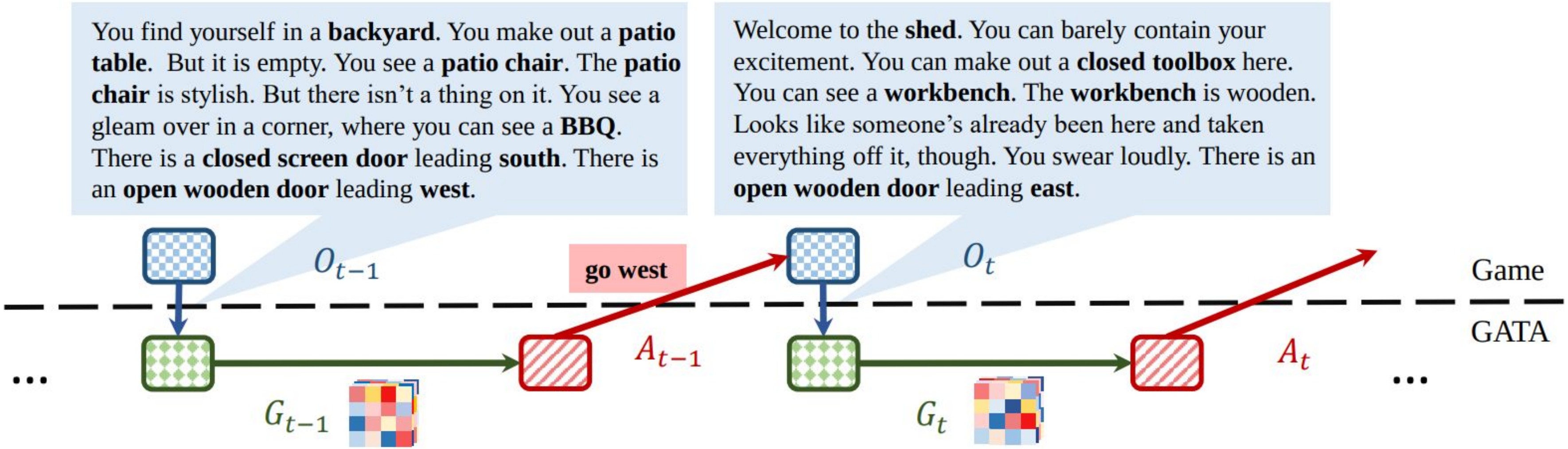


“A deep learning approach to antibiotic discovery”
2020

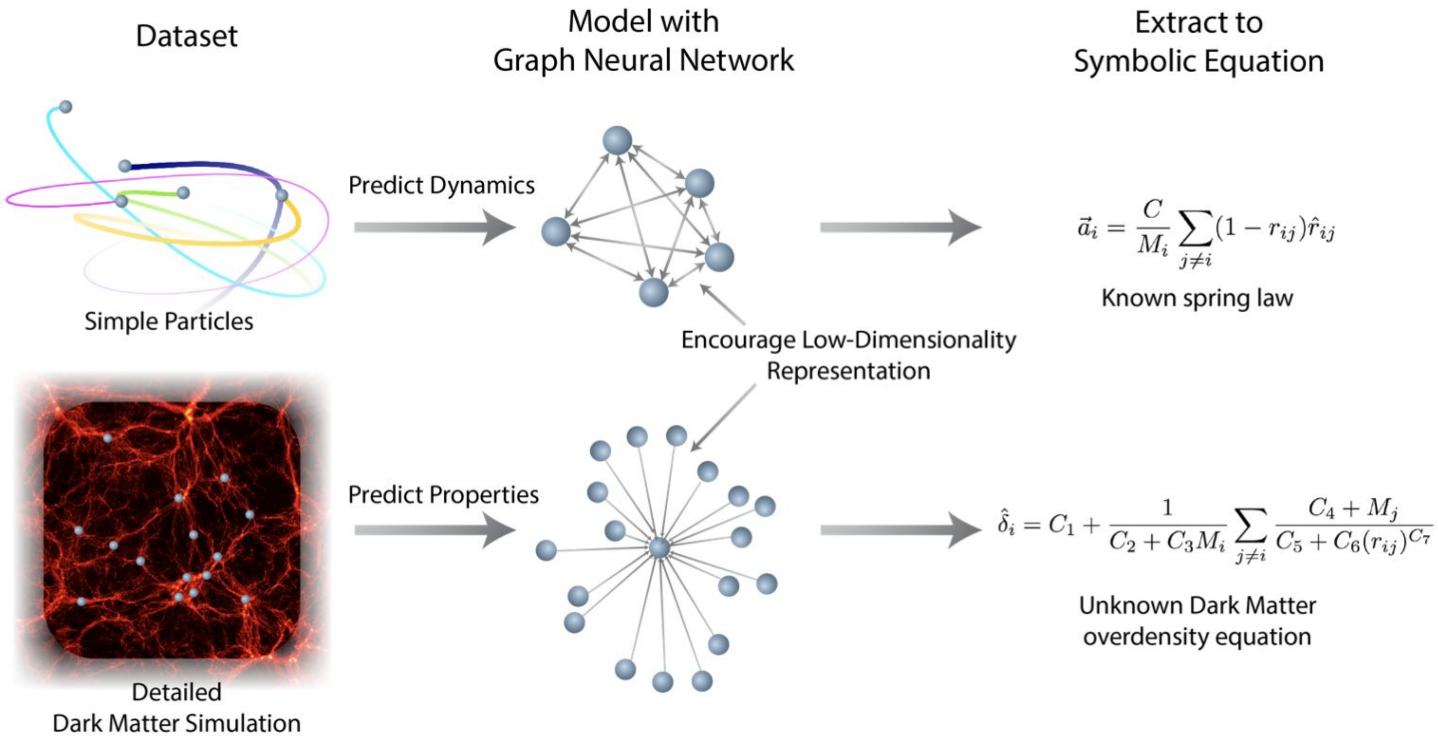
- классификация на то, проявляет ли молекула свойства антибиотика
- найдена новая молекула - Halicin (уже проверили на мышах)

“Learning Dynamic Belief Graphs to Generalize on Text-Based Games”

2020



Открытие законов Вселенной?



[Discovering Symbolic Models from Deep Learning with Inductive Biases]

Список источников

- A Comprehensive Survey On Graph Neural Networks
- The Graph Neural Network Model
- Gated Graph Sequence Neural Networks
- Graph Neural Networks: A Review of Methods and Applications
- Understanding Graph Neural Networks (video)
- Graph Laplacian and its application in Machine learning
- Anisotropic, Dynamic, Spectral and Multiscale Filters Defined on Graphs
- Discovering Symbolic Models from Deep Learning with Inductive Biases
- Understanding Spectral Graph Neural Network