Deep Learning on Graphs

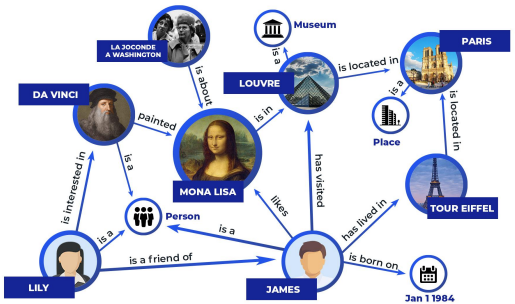
Садртдинов Ильдус, Семавина Юлия

Problem Statement

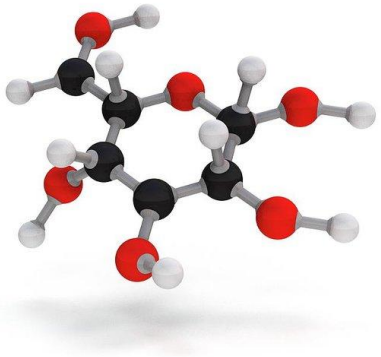
$$\tau(G, n) \in \mathbb{R}^m$$

Graph Data

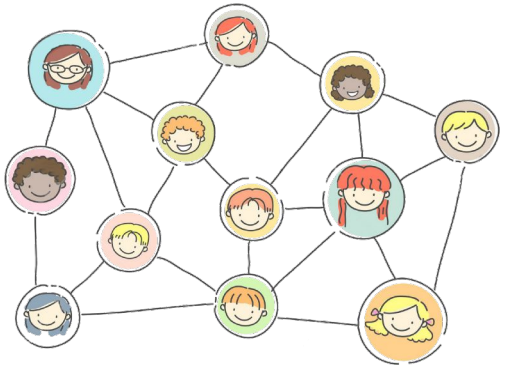
Knowledge Graphs



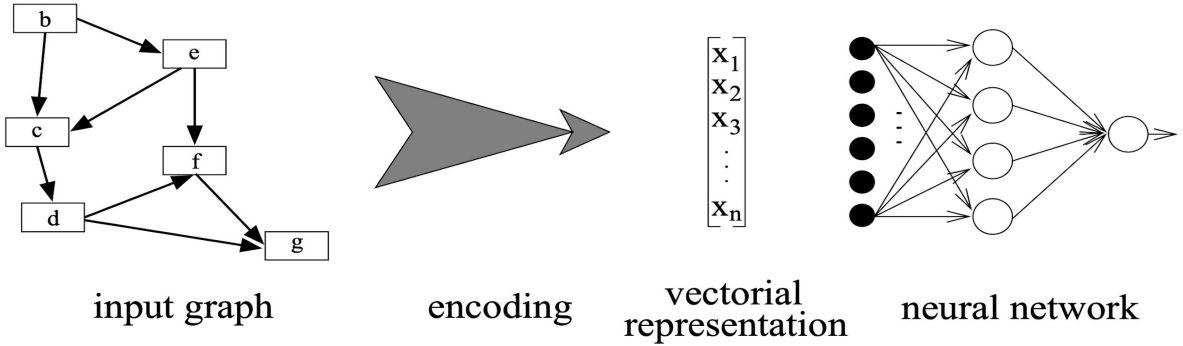
Molecules



Social Networks



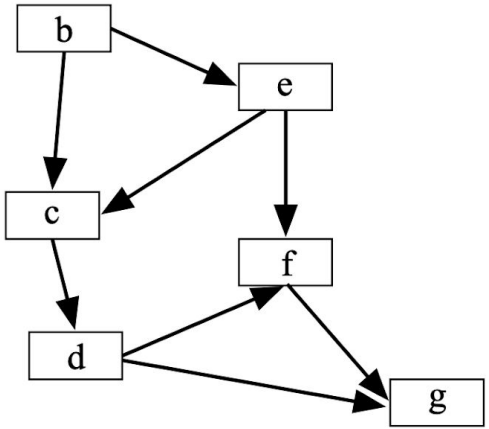
Approach - 1 (Graph preprocessing, feature extraction)



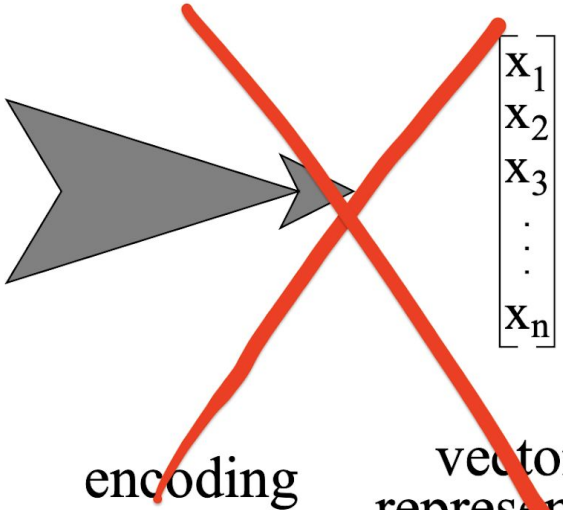
- node lists
- node-edge-node triples
- adjacency matrix
- PageRank, ...
- embeddings

- the relevance of different features of the graphs may change dramatically for different learning tasks
- since the encoding process has to work for any classification task, each graph must get a different representation; this may result in vectorial representations which are very difficult to classify
- Information loss

Approach - 2 (Graph neural networks)

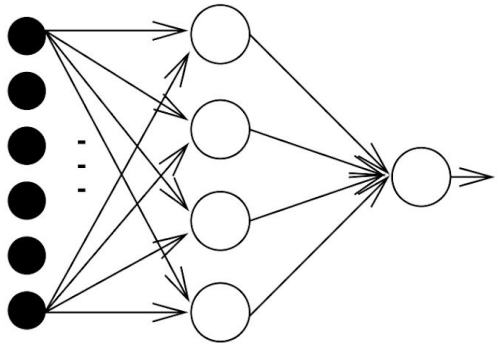


input graph



encoding

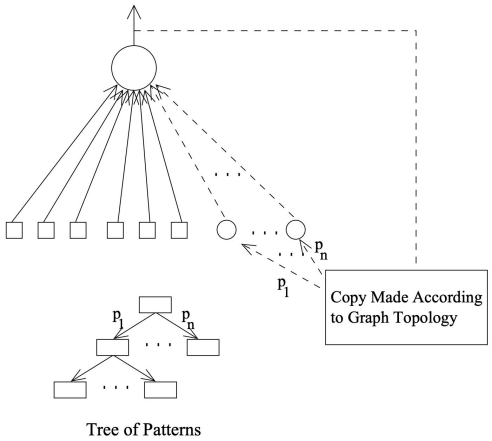
vectorial representation

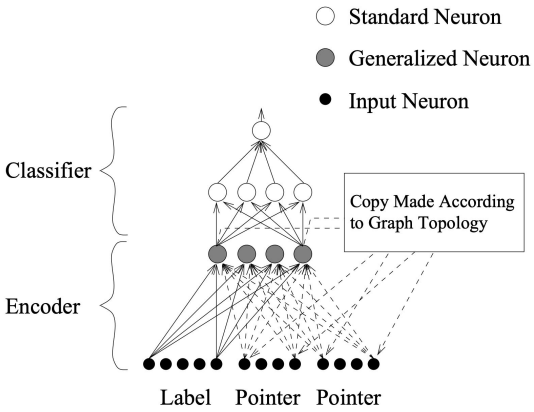


neural network

Graph Neural Networks

1997 - NN was applied to directed acyclic graphs, which motivated early studies on GNNs. Generalized Recursive Neuron approach.

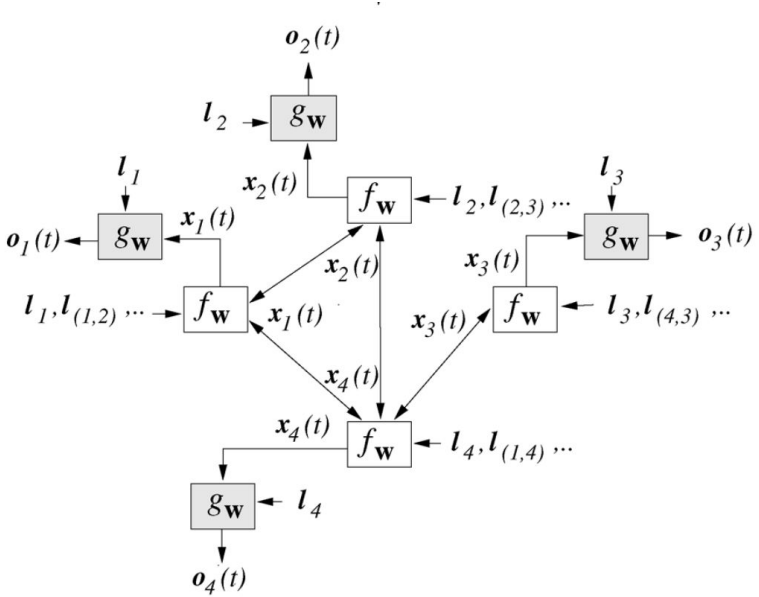




$$o^{(g)}(x) = f(\sum_{i=1}^{N_L} w_i l_i + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_j o^{(g)}(out_X(x,j))),$$

Graph Neural Networks

2005, 2009 - GNN



Graph Neural Networks

And now...

Graph Neural Networks

And now...

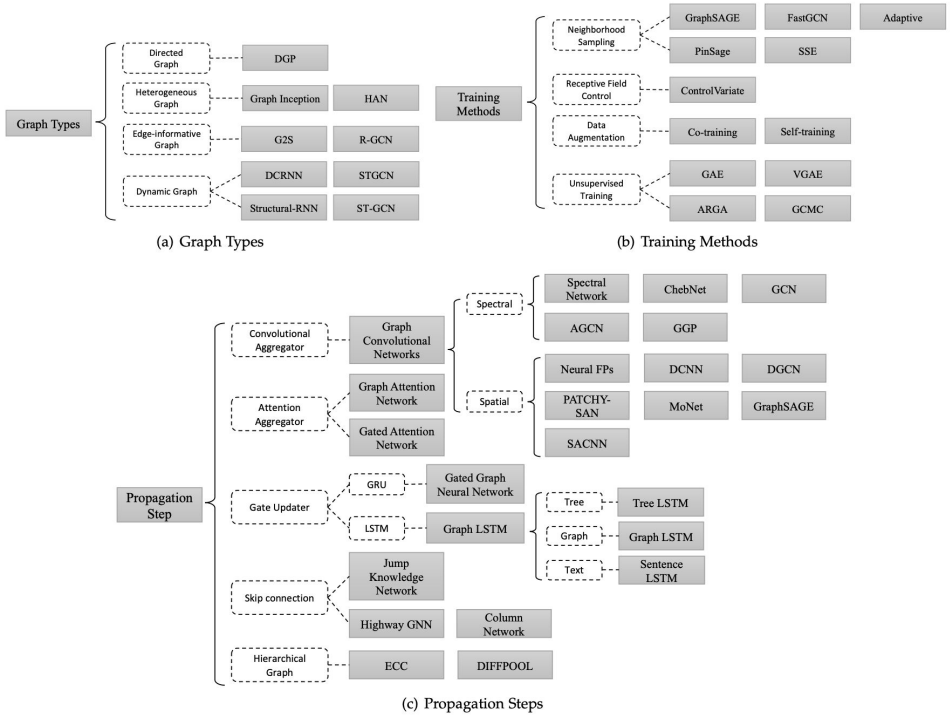


Fig. 2. An overview of variants of graph neural networks.

Task types

Node-focused tasks

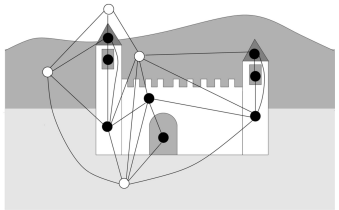
Tasks are associated with individual nodes in the graph.

Examples: node classification, link prediction, node recommendation.

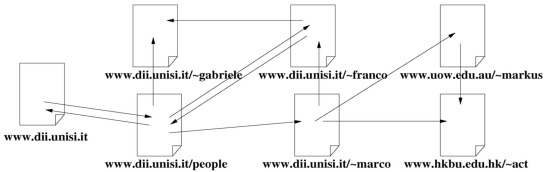
Graph-focus tasks

Tasks are associated with the whole graph.

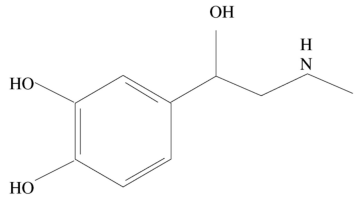
Examples: graph classification, estimating certain properties of the graph.



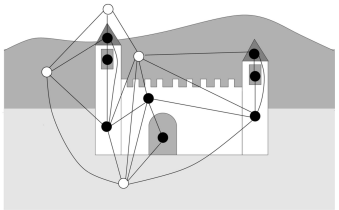
object detection



node classification



Estimating certain properties

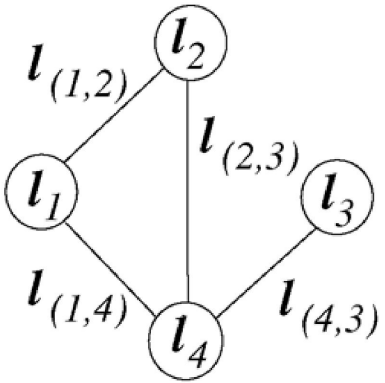


Graph classification

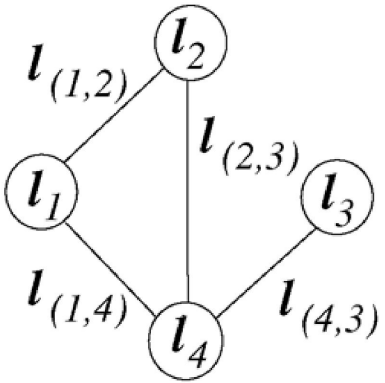
Graph Neural Network (GNN)

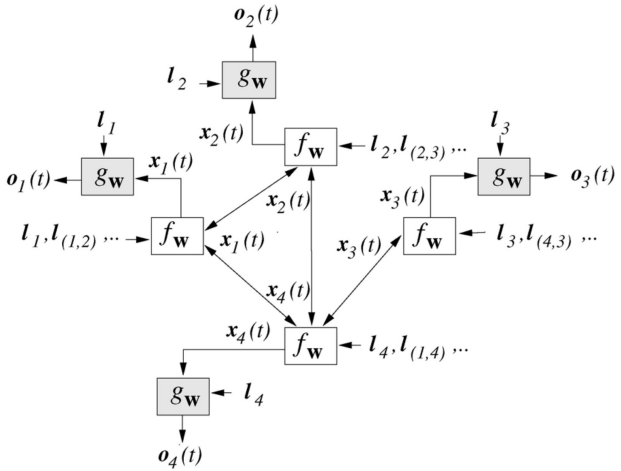
- suitable for both graph and node-focused applications
- can process a more general class of graphs including cyclic, directed, and undirected graphs
- no preprocessing
- universal approach
- vertex order invariance

Graph Neural Network (GNN)

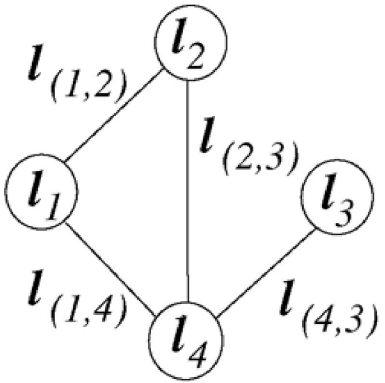


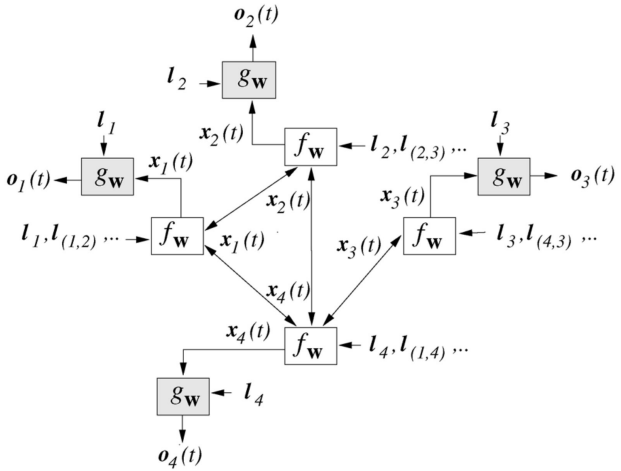
Graph Neural Network (GNN)

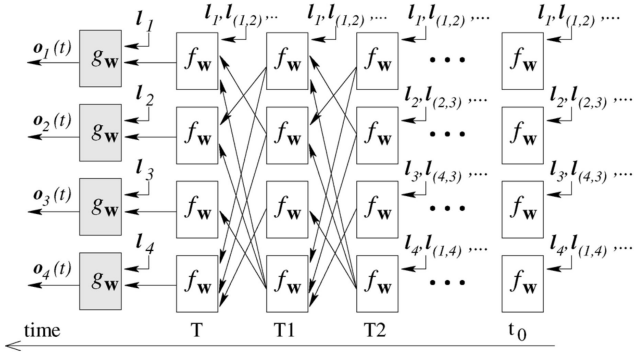




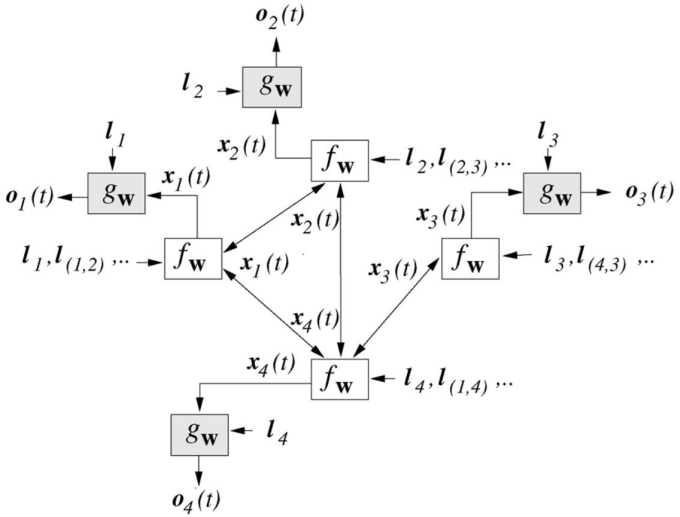
Graph Neural Network (GNN)



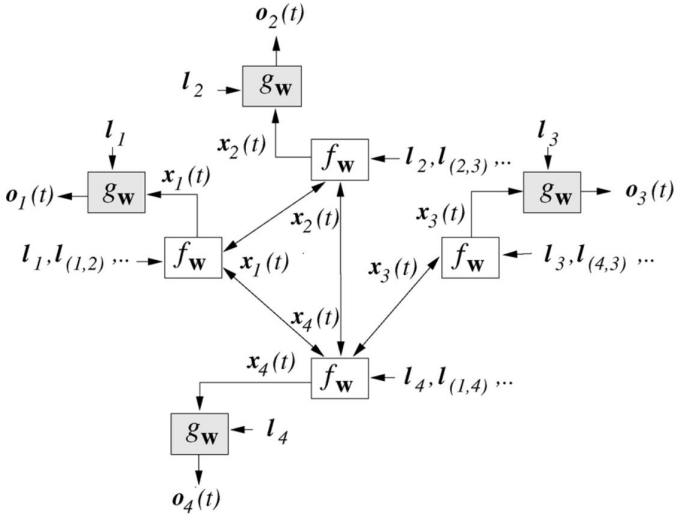


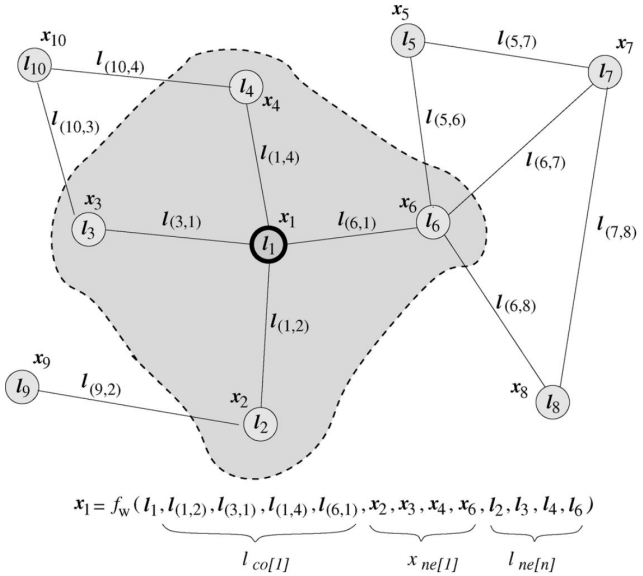


Graph Neural Network (GNN)



Graph Neural Network (GNN)

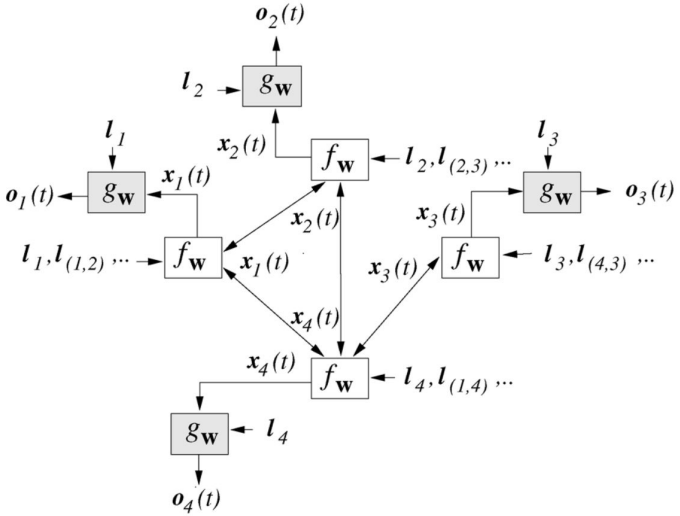


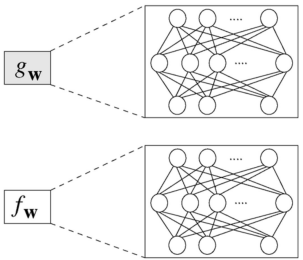


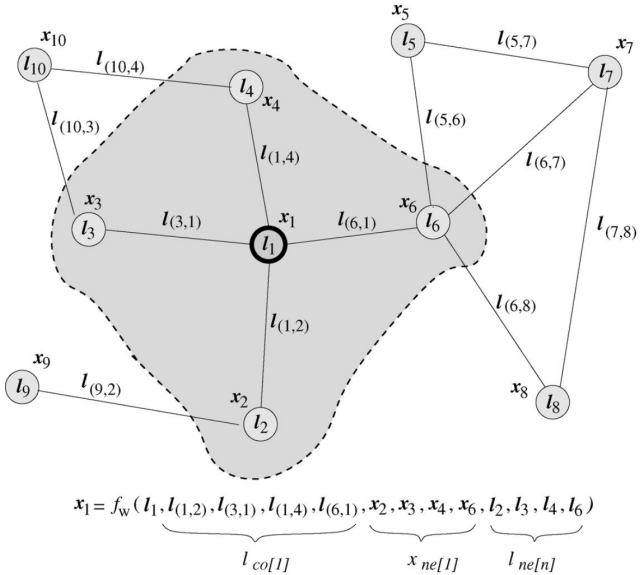
local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

Graph Neural Network (GNN)







local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

Graph Neural Network (GNN)

local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

global transition function: $\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$

global output function: $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$

Graph Neural Network (GNN)

local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

global transition function: $\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$

global output function: $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$

When are x, o uniquely defined?

Graph Neural Network (GNN)

local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

global transition function: $\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$

global output function: $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$

When are x, o uniquely defined?

F_w - contraction map - $\exists \mu \, \forall x, y : \|F_w(x, l) - F_w(y, l)\| < \mu \|x - y\|$

Graph Neural Network (GNN)

local transition function: $\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$

local output function: $\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$

global transition function: $\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$

global output function: $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$

When are x, o uniquely defined?

F_w - contraction map - $\exists \mu \, \forall x, y : \|F_w(x, l) - F_w(y, l)\| < \mu \|x - y\|$

Jacobi method: $x(t+1) = F_w(x(t), l)$

Graph Neural Network (GNN)

- learning parameter: w
- learning set:

$$\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j})|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G};$$
$$n_{i,j} \in \boldsymbol{N}_i; \boldsymbol{t}_{i,j} \in I\!R^m, 1 \leq i \leq p, 1 \leq j \leq q_i\}$$

- cost function:

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2$$

- learning algorithm

1. The states are iteratively updated by $x(t+1) = F_w(x(t), l)$ until at time they approach the fixed point solution of $x = F_w(x, l)$.
2. The gradient is computed $\frac{de_w(T)}{dw}$.
3. The weights are updated according to the gradient.

```
MAIN
    initialize w;
    x=Forward(w);
    repeat
        ∂e_w/∂w=BACKWARD(x, w);
        w=w − λ · ∂e_w/∂w;
        x=FORWARD(w);
    until (a stopping criterion);
    return w;
end

FORWARD(w)
    initialize x(0), t = 0;
    repeat
        x(t + 1) = F_w(x(t), l);
        t=t + 1;
    until ‖x(t) − x(t − 1)‖ ≤ ε_f
    return x(t);
end

BACKWARD(x,w)
    o = G_w(x, l_N);
    A = ∂F_w/∂x (x, l);
    b = ∂e_w/∂o · ∂G_w/∂x (x, l_N);
    initialize z(0), t=0;
    repeat
        z(t) = z(t + 1) · A + b;
        t=t − 1;
    until ‖z(t − 1) − z(t)‖ ≤ ε_b;
    c = ∂e_w/∂o · ∂G_w/∂w (x, l_N);
    d = z(t) · ∂F_w/∂w (x, l);
    ∂e_w/∂w = c + d;
    return ∂e_w/∂w;
end
```

Graph Neural Network (GNN)

Theorem 2 (Backpropagation): Let $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ be the transition and the output functions of a GNN, respectively, and assume that $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$ and $G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N})$ are continuously differentiable w.r.t. $\boldsymbol{x}$ and $\boldsymbol{w}$. Let $\boldsymbol{z}(t)$ be defined by

$$\boldsymbol{z}(t) = \boldsymbol{z}(t+1) \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l}) + \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l_N}) \,. \qquad (7)$$

Then, the sequence $\boldsymbol{z}(T), \boldsymbol{z}(T-1), \ldots$ converges to a vector $\boldsymbol{z} = \lim_{t \to -\infty} \boldsymbol{z}(t)$ and the convergence is exponential and independent of the initial state $\boldsymbol{z}(T)$. Moreover

$$\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N}) + \boldsymbol{z} \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}) \qquad (8)$$

holds, where $\boldsymbol{x}$ is the stable state of the GNN.

MAIN
  initialize $\boldsymbol{w}$;
  $\boldsymbol{x}$=**Forward**($\boldsymbol{w}$);
  repeat
    $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$=BACKWARD($\boldsymbol{x}, \boldsymbol{w}$);
    $\boldsymbol{w}=\boldsymbol{w} - \lambda \cdot \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
    $\boldsymbol{x}$=FORWARD($\boldsymbol{w}$);
  until (a stopping criterion);
  return $\boldsymbol{w}$;
end

FORWARD($\boldsymbol{w}$)
  initialize $\boldsymbol{x}(0)$, $t = 0$;
  repeat
    $\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l})$;
    t=$t+1$;
  until $\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\| \leq \varepsilon_f$
  return $\boldsymbol{x}(t)$;
end

BACKWARD($\boldsymbol{x}$,$\boldsymbol{w}$)
  $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N})$;
  $\boldsymbol{A} = \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l})$;
  $\boldsymbol{b} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l_N})$;
  initialize $\boldsymbol{z}(0)$, t=0;
  repeat
    $\boldsymbol{z}(t) = \boldsymbol{z}(t+1) \cdot \boldsymbol{A} + \boldsymbol{b}$;
    t=$t-1$;
  until $\|\boldsymbol{z}(t-1) - \boldsymbol{z}(t)\| \leq \varepsilon_b$;
  $\boldsymbol{c} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N})$;
  $\boldsymbol{d} = \boldsymbol{z}(t) \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$;
  $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}} = \boldsymbol{c} + \boldsymbol{d}$;
  return $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
end

Graph Neural Network (GNN)

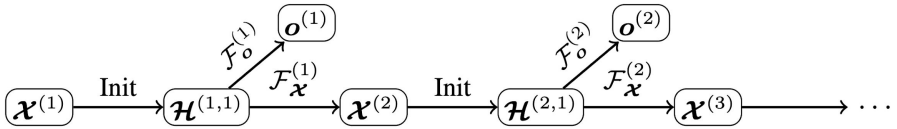
Limitations:

- global transition function must be a contraction map
- it is inefficient to update the hidden states of nodes iteratively for the fixed point + the number of updates is unknown
- uses the same parameters in the iteration while most popular neural networks use different parameters in different layers, which serve as a hierarchical feature extraction method
- there are also some informative features on the edges which cannot be effectively modeled in the original GNN. For example, the edges in the knowledge graph have the type of relations and the message propagation through different edges should be different according to their types

GNN Improvements/Modifications

2016 - Gated Graph Sequence Neural Networks (GGS-NNs):

- unroll the recurrence for a fixed number of steps T
- remove the need to constrain parameters to ensure convergence (contraction map)
- add node annotations to add initialization information as an input
- use GRU
- extend to output sequences



Task	RNN	LSTM	GG-NN
bAbI Task 4	97.3±1.9 (250)	97.4±2.0 (250)	100.0±0.0 (50)
bAbI Task 15	48.6±1.9 (950)	50.3±1.3 (950)	100.0±0.0 (50)
bAbI Task 16	33.0±1.9 (950)	37.5±0.9 (950)	100.0±0.0 (50)
bAbI Task 18	88.9±0.9 (950)	88.9±0.8 (950)	100.0±0.0 (50)

GNN Improvements/Modifications

2018 - Stochastic Steady-state Embedding (SSE)

- algorithm which can run in a large graph
- stochastic fixed-point gradient descent to accelerate training

2018 - Dynamic Graph Neural Network (DGNN)

- can model the dynamic information as the graph evolving

Applications

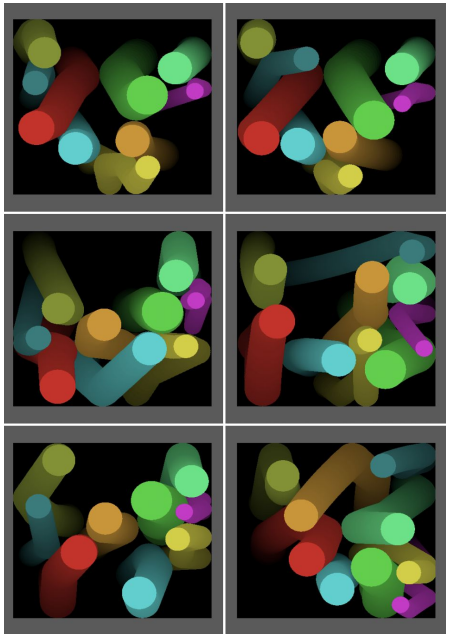
Interaction network

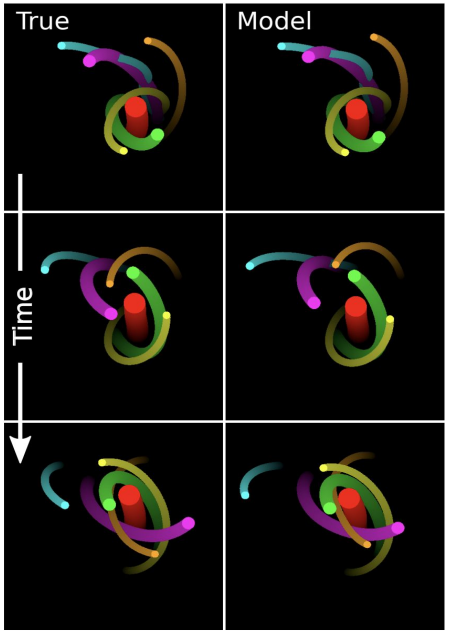
- capture the complex interactions that can be used to predict future states and abstract physical properties, such as energy

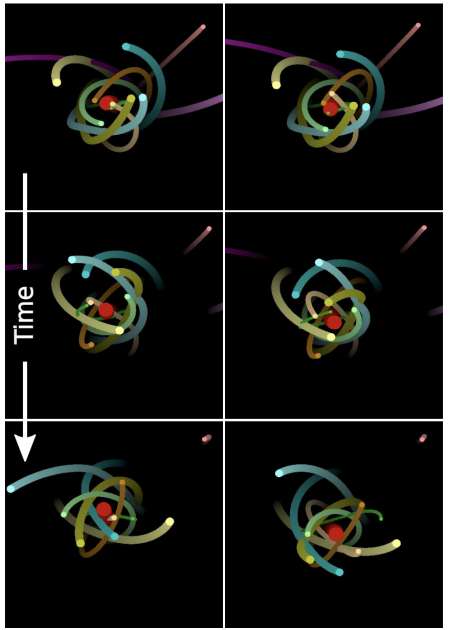
Applications

Interaction network

- capture the complex interactions that can be used to predict future states and abstract physical properties, such as energy



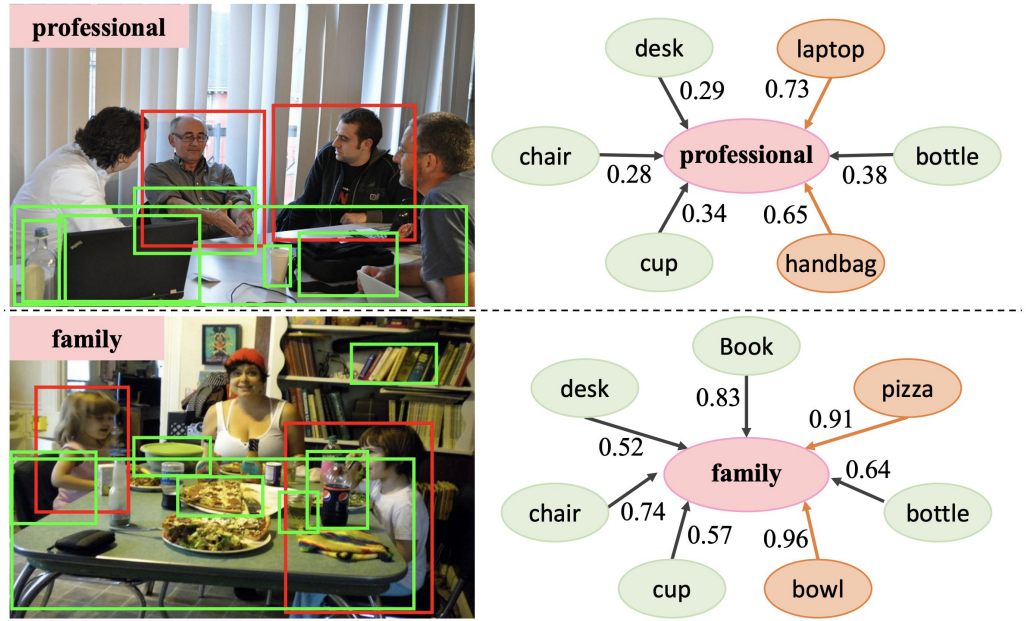




Applications

Graph Reasoning Model (GRM)

- recognize social relationships
- Uses GGNN



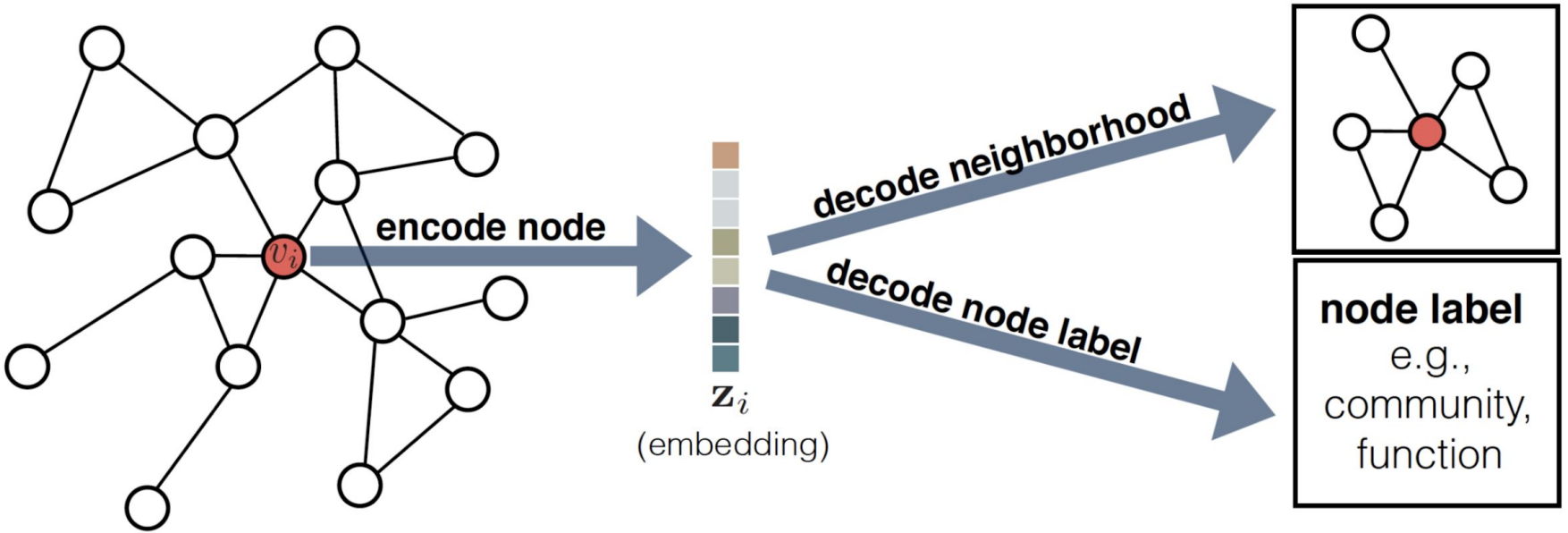
Вопросы

1. Чем отличаются graph-focused и node-focused задачи? Приведите один пример каждой из них.
2. Опишите архитектуру Graph Neural Network.
3. Недостатки модели Graph Neural Network.

Источники

- Graph Neural Networks: http://persagen.com/files/misc/scarselli2009graph.pdf
- Supervised Neural Networks for the Classication of Structures 1997: https://pdfs.semanticscholar.org/3e33/eca03933caaec671e20692e79d1acc9527e1.pdf?_ga=2.38869348.693624108.1579823060-1153278333.1576771929
- Surveys on GNN: https://arxiv.org/pdf/1901.00596.pdf , https://arxiv.org/pdf/1812.08434.pdf, https://arxiv.org/pdf/1812.04202.pdf
- Gated Graph Sequence NN: https://arxiv.org/pdf/1511.05493.pdf
- GNN Applications: https://arxiv.org/pdf/1511.05493.pdf , https://www.ijcai.org/Proceedings/2018/0142.pdf

Эмбеддинги


v_i
encode node
$\mathbf{z}_i$
(embedding)
decode neighborhood
decode node label
node label
e.g.,
community,
function

Encoder-decoder approach

Для задания эмбеддинга нам нужно определить следующие функции:

- $s_{\mathcal{G}} : \mathcal{V} \times \mathcal{V} :\to \mathbb{R}^+$ - функция расстояния между вершинами (proximity measure)
- $\mathrm{ENC} : \mathcal{V} :\to \mathbb{R}^d$ - encoder
- $\mathrm{DEC} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ - decoder
- $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ - функция потерь (loss function)

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$v_i \in \mathcal{V}$ - node
$z_i \in \mathbb{R}^d$ - embedding

$$\mathrm{DEC}(\mathrm{ENC}(v_i), \mathrm{ENC}(v_j)) = \mathrm{DEC}(z_i, z_j) \approx s_{\mathcal{G}}(v_i, v_j)$$

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{V} \times \mathcal{V}} \ell(\mathrm{DEC}(z_i, z_j), s_{\mathcal{G}}(v_i, v_j))$$

Direct encoding

В подходе direct encoding каждая вершина кодируется отдельным вектором. Варьируя decoder и функцию расстояния мы можем выбирать, какие свойства вершины хотим зашить в эмбеддинг.

$$\mathrm{ENC}(v_i) = Zv_i$$

$$Z \in \mathbb{R}^{d \times |\mathcal{V}|}, v_i \in \mathbb{I}_{\mathcal{V}}$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$v_i \in \mathcal{V}$ - node
$z_i \in \mathbb{R}^d$ - embedding
$\mathbb{I}_{\mathcal{V}}$ - graph OHE

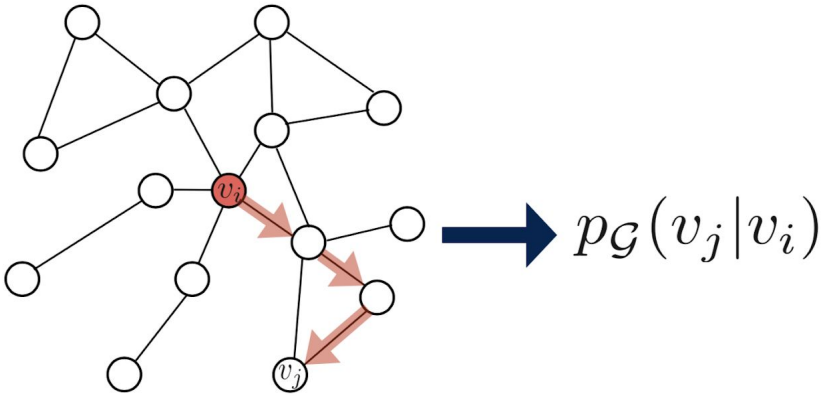
Direct encoding

Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

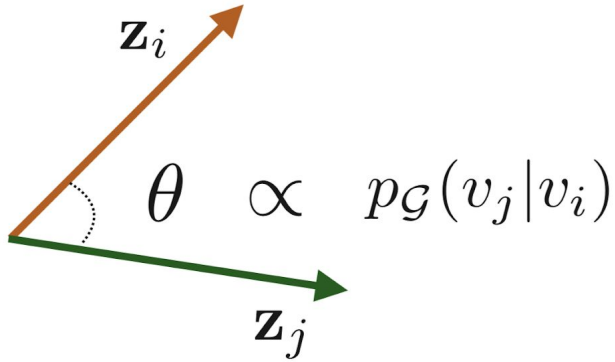
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
A - adjacency matrix
$v_i \in \mathcal{V}$ - node
$z_i \in \mathbb{R}^d$ - embedding
$s_{\mathcal{G}}$ - proximity measure
$\rho_{\mathcal{G}}$ - random walk probability

Node2Vec

- Выборочная стратегия (sampling strategy) - генерируем случайные пути на графе для оценки вероятности перехода между вершинами
- Skip-gram модель (по аналогии с Word2Vec) - обучаем эмбеддинги на основе собранной статистики

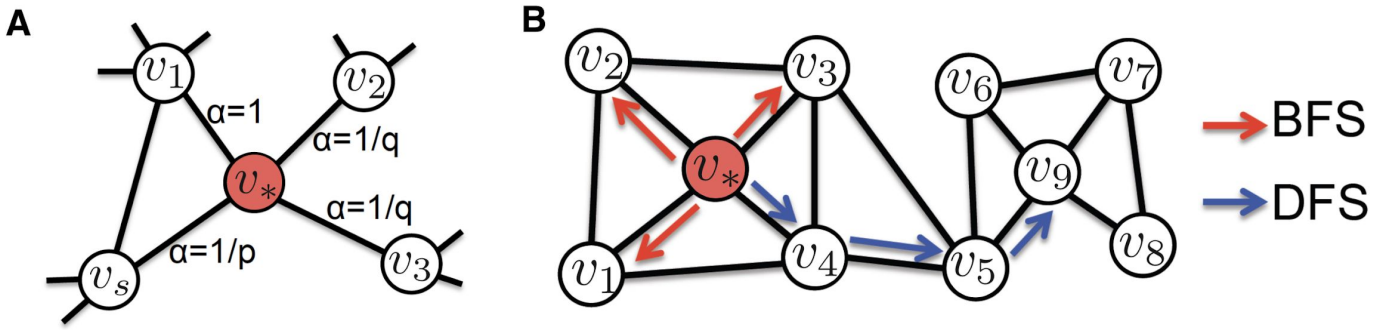


1. Run random walks to obtain co-occurrence statistics.



2. Optimize embeddings based on co-occurrence statistics.

Node2Vec: Sampling strategy

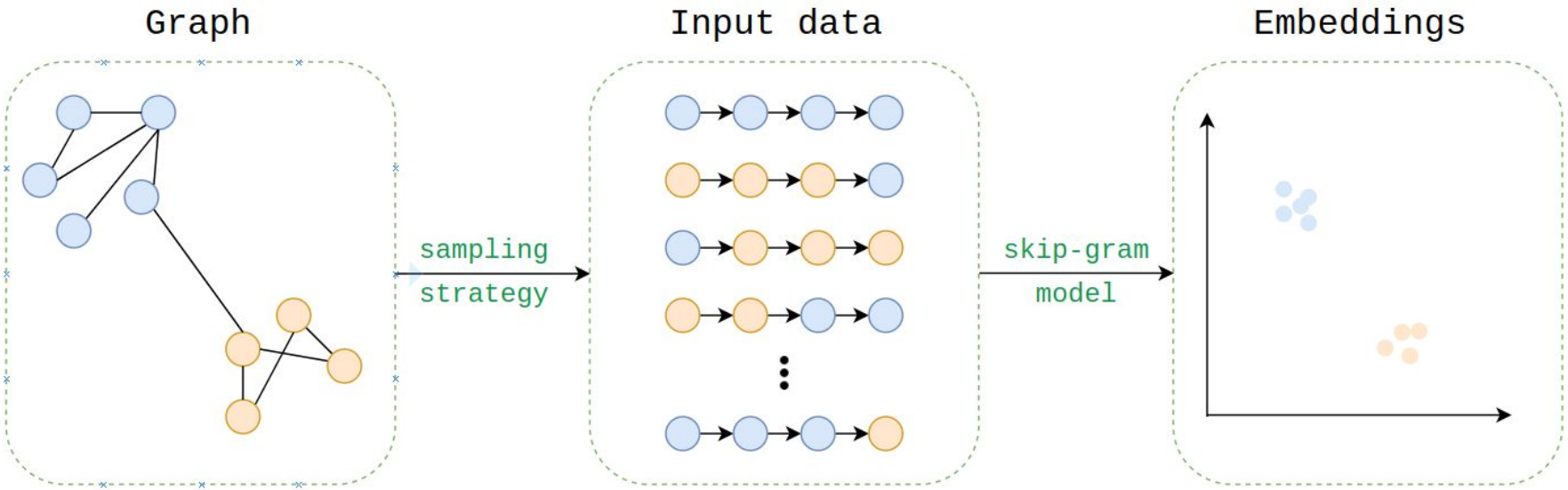


$$\alpha_{pq}(v_s, v_x) = \begin{cases} \frac{1}{p}, & d(v_s, v_x) = 0 \\ 1, & d(v_s, v_x) = 1 \\ \frac{1}{q}, & d(v_s, v_x) = 2 \end{cases}$$

$$\pi(v_*, v_x) \sim \alpha_{pq}(v_s, v_x) w(v_*, v_x)$$

v_* - current node
v_s - previous node
$d(v_s, v_x)$ - shortest path between v_s and v_x
$\pi(v_*, v_x)$ - passage probability from v_* to v_x
$w(v_*, v_x)$ - weight of edge between v_* and v_x

Node2Vec: Skip-gram

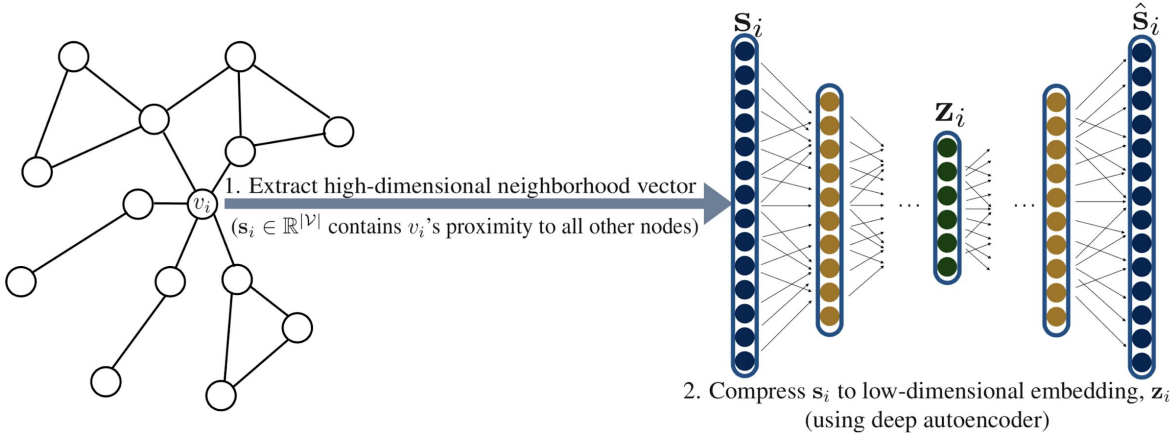


Direct encoding

+ Выбором функции расстояния и decoder мы можем варьировать информацию, которую закладываем в эмбеддинг.
+ Быстро и легко обучается

- Эмбеддинги получаются независимыми: нет общих параметров для разных вершин, так что вектора обучаются отдельно
- При обучении не учитывается информация о самих вершинах (например, данные профиля в социальных сетях)
- Нельзя создавать эмбеддинги для новых вершин

Neighborhood autoencoders



$$\mathrm{DEC}(\mathrm{ENC}(s_i)) = \mathrm{DEC}(z_i) \approx s_i$$

$$s_i \in \mathbb{R}^{|\mathcal{V}|} : s_{ij} = s_{\mathcal{G}}(v_i, v_j)$$

$$\mathcal{L} = \sum_{v_i \in \mathcal{V}} ||\mathrm{DEC}(z_i) - s_i||_2^2$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$v_i \in \mathcal{V}$ - node
$s_i \in \mathbb{R}^{|\mathcal{V}|}$ - neighborhood vector
$z_i \in \mathbb{R}^d$ - embedding
$s_{\mathcal{G}}$ - proximity measure

Neighborhood autoencoders

Примеры энкодеров:

- Deep Neural Graph Representations (DNGR) - основан на RandomWalk
- Structural Deep Network Embeddings (SDNE) - матрица смежности + комбинация MSE и Laplacian Eigenmaps как функция потерь

Neighborhood autoencoders

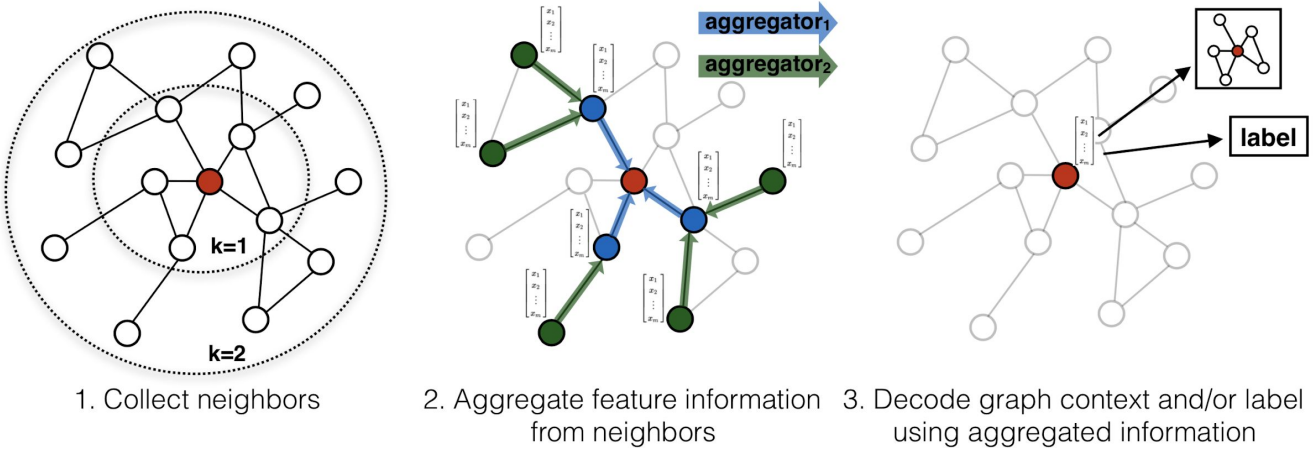
+ Параметры сети одни и те же для всех вершин, эмбеддинги обучаются согласованно ⇒ бесплатная регуляризация

- Все еще не использована внутренняя информация вершин
- Все еще нельзя создавать эмбеддинги для новых вершин

Neighborhood aggregation & convolutional encoders

Идея:

- Использовать внутреннюю информацию вершины
- Собирать и усреднять внутреннюю информацию соседних вершин



1. Collect neighbors

2. Aggregate feature information from neighbors

3. Decode graph context and/or label using aggregated information

Neighborhood aggregation & convolutional encoders

Algorithm 1: Neighborhood-aggregation encoder algorithm. Adapted from [28].

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

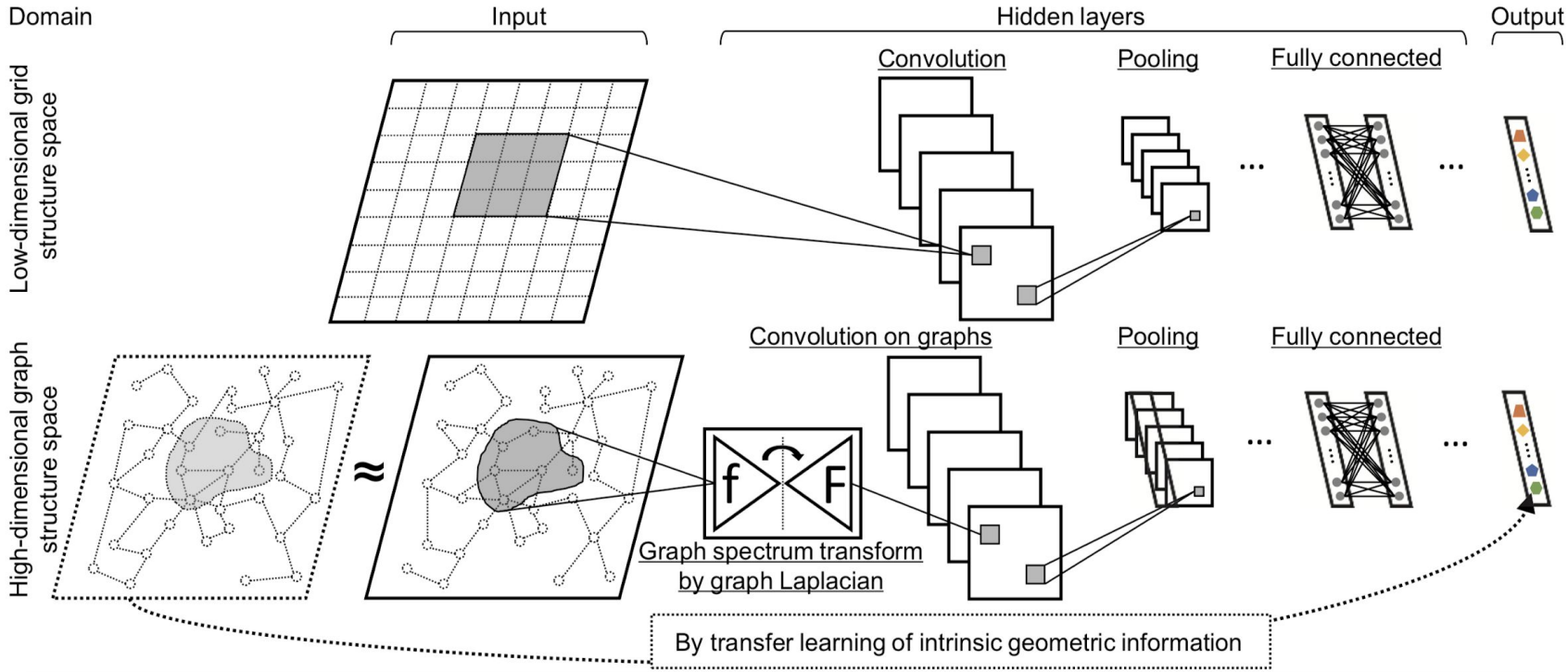
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Neighborhood aggregation & convolutional encoders

Примеры:

- Некоторые архитектуры Graph Convolutional Networks (GCN)
- GraphSAGE: конкатенация в качестве COMBINE, любой AGGREGATE (например, поэлементное среднее или LSTM).

Graph Convolutional Networks (GCN)



Спектральная свертка

Рассмотрим матрицу Лапласа для графа:

$$L = D - A$$

D - диагональная матрица со степенями вершин графа

A - матрица смежности графа

Спектральная свертка

Рассмотрим матрицу Лапласа для графа:

$$L = D - A$$

D - диагональная матрица со степенями вершин графа

A - матрица смежности графа

Эта матрица симметрическая, поэтому ее можно диагонализовать:

$$L = U \begin{bmatrix} \lambda_0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{N-1} \end{bmatrix} U^T$$

Спектральная свертка

Рассмотрим некоторую функцию на вершинах графа. Если мы пронумеруем вершины, можно представить ее как вектор:

$$f_G : \mathcal{V} :\to \mathbb{R}, f_G \in \mathbb{R}^N$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda_{N-1}$ - eigenvalues

Спектральная свертка

Рассмотрим некоторую функцию на вершинах графа. Если мы пронумеруем вершины, можно представить ее как вектор:

$$f_G : \mathcal{V} :\to \mathbb{R}, f_G \in \mathbb{R}^N$$

Оказывается, что этот вектор можно представить в таком виде (по аналогии с преобразованием Фурье):

$$f_G = \sum_{l=0}^{N-1} \hat{f}_G(\lambda_l) u_l$$

$\hat{f}_G : \mathbb{R} \to \mathbb{R}$ - амплитуды

u_l - собственные векторы матрицы Лапласа

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda_{N-1}$ - eigenvalues

Спектральная свертка

Для нахождения амплитуд существует обратное преобразование:

$$\hat{f}_G(\lambda_l) = \sum_{n=1}^{N} f_G(n) u_l(n) = \langle f_G, u_l \rangle$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda$ - eigenvalues
$u_0, ..., u_{N-1}$ - eigenvectors
$f_G \in \mathbb{R}^N$ - graph function
$\hat{f}_G : \mathbb{R} \to \mathbb{R}$ - amplitudes

Спектральная свертка

Для нахождения амплитуд существует обратное преобразование:

$$\hat{f}_G(\lambda_l) = \sum_{n=1}^{N} f_G(n) u_l(n) = \langle f_G, u_l \rangle$$

Теперь можно определить свертку двух векторов:

$$f *_G g = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\hat{g}(\lambda_l)u_l$$

$$f *_G g = \hat{g}(L)f = U \begin{bmatrix} \hat{g}(\lambda_0) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \hat{g}(\lambda_{N-1}) \end{bmatrix} U^T f$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda$ - eigenvalues
$u_0, ..., u_{N-1}$ - eigenvectors
$f_G \in \mathbb{R}^N$ - graph function
$\hat{f}_G : \mathbb{R} \to \mathbb{R}$ - amplitudes

Сверточные слои

$$x_{k+1,j} = \sigma\left(\sum_{i=1}^{\phi_k} f_{k,i,j} * x_{k,i}\right) = \sigma\left(\sum_{i=1}^{\phi_k} \hat{f}_{k,i,j}(L)x_{k,i}\right) =$$

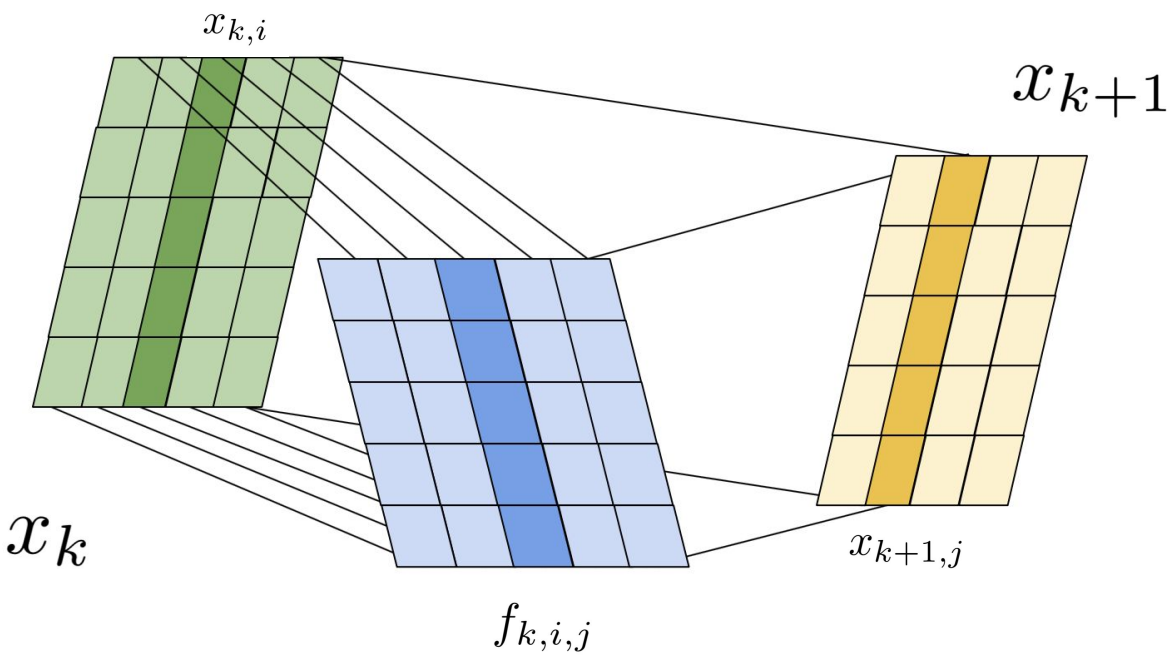
$$= \sigma\left(U\sum_{i=1}^{\phi_k} F_{k,i,j}U^T x_{k,i}\right), j = 1, \ldots, \phi_{k+1}$$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda$ - eigenvalues
$x_k \in \mathbb{R}^{N \times \phi_k}$ - k-th layer output
$x_{k,i} \in \mathbb{R}^N$ - i-th feature map of x_k
$f_{k,i,j} \in \mathbb{R}^N$ - filter vector
$F_{i,j,k} \in \mathbb{R}^{N \times N}$ - diagonal filter matrix
$\alpha_0, ..., \alpha_{N-1}$ - trainable parameters
σ - non-linear function

$$\hat{f}_{k,i,j}(L) = UF_{i,j,k}U^T \quad F_{i,j,k} = \begin{bmatrix} \alpha_0\lambda_0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \alpha_{N-1}\lambda_{N-1} \end{bmatrix}$$

Всего обучаемых параметров в слое: $N\phi_k\phi_{k+1}$

Сверточные слои



$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ - graph
$N = |\mathcal{V}|$ - number of nodes
L - Laplacian matrix
U - matrix of eigenvectors
$\lambda_0, ..., \lambda$ - eigenvalues
$x_k \in \mathbb{R}^{N \times \phi_k}$ - k-th layer output
$x_{k,i} \in \mathbb{R}^N$ - i-th feature map of x_k
$f_{k,i,j} \in \mathbb{R}^N$ - filter vector
$F_{i,j,k} \in \mathbb{R}^{N \times N}$ - diagonal filter matrix
$\alpha_0, ..., \alpha_{N-1}$ - trainable parameters
σ - non-linear function

Спектральные свертки

+ Агрегация информацию со всего графа с учетом его структурных особенностей ⇒ выделение локальных и глобальных закономерностей

- Большое число параметров: ***O(N)*** на каждом слое
- Необходимость работать с собственными векторами: временные затраты $O(N^3)$ на разложение матрицы и $O(N^2)$ при forward- и backpropagation
- Параметры специфичны для графа, поэтому под каждый граф нужна отдельная нейросеть

Предложенные улучшения

- ChebNet
  - Уменьшение числа параметров за счет использования фильтров специального вида
  - Уход от вычисления собственных векторов за счет использования многочленов Чебышева
- Возможность использовать сеть на разных графах: neighborhood aggregation + другие архитектуры CNN
  - Neural FPs - разные фильтры для вершин разных степеней
  - PATCHY-SCAN - 1-D свертки для соседей вершины
  - GraphSAGE

Вопросы

1. ~~Посчитайте сдвиги для градиентного бустинга.~~
2. Опишите любую модель direct encoding для эмбеддингов. В чем плюсы и минусы direct encoding?
3. В чем преимущество метода Neighborhood aggregation над другими эмбеддингами? Опишите вкратце архитектуру его энкодера.
4. Напишите формулу для спектральной свертки двух векторов (функций на вершинах графа) в любой из предложенных вариаций, объясните входящие в нее компоненты.

Источники

- Обзор: https://arxiv.org/pdf/1812.04202.pdf
- Эмбеддинги: https://www-cs.stanford.edu/people/jure/pubs/graphrepresentation-ieee17.pdf
- Node2Vec: https://arxiv.org/pdf/1607.00653.pdf
- Спектральные свертки https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14803