Graph Neural Networks:

A Review of Methods and Applications

Jie Zhou*, Ganqu Cui*, Zhengyan Zhang*, Cheng Yang, Zhiyuan Liu, Maosong Sun

Reporter: Tianxiang Sun (孙天祥，19110240005)

txsun19@fudan.edu.cn

https://arxiv.org/pdf/1812.08434.pdf

Outline

1. Introduction
2. Models
    - Graph Neural Networks
    - Variants of Graph Neural Networks
    - General Frameworks
3. Applications
    - Structural Scenarios
    - Non-Structural Scenarios
    - Other Scenarios
4. Open Problems
5. Conclusion

Introduction

- Neural networks has achieved great success on regular Euclidean data (images, text, etc.)
- Social network, molecular fingerprints, protein interface prediction…
- Generalize neural networks from Euclidean domain to non-Euclidean domain.

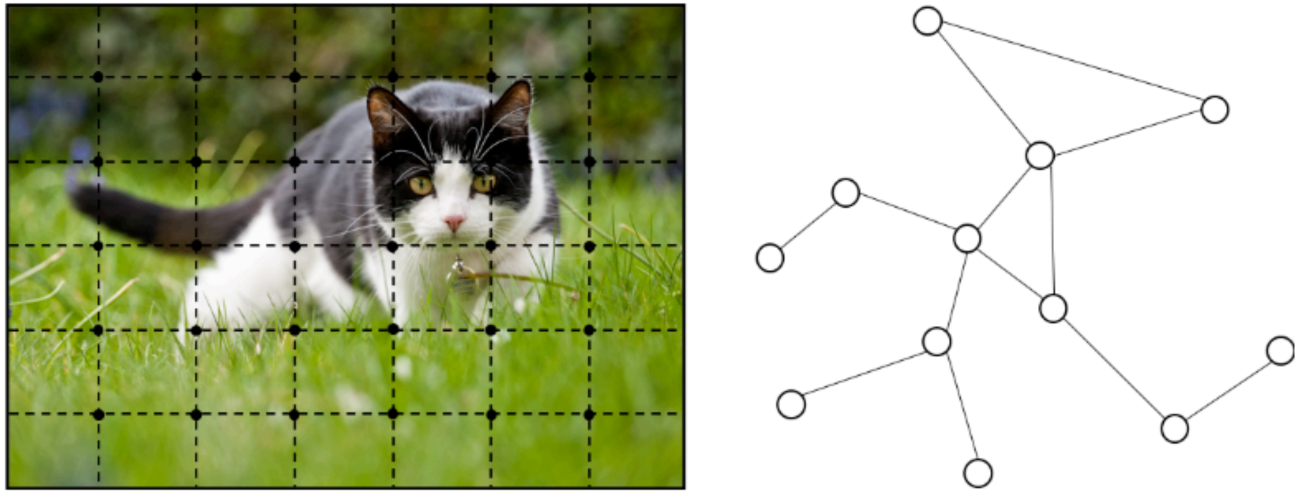


Fig. 1. Left: image in Euclidean space. Right: graph in non-Euclidean space

Models – Graph Neural Networks

- The first paper - The Graph Neural Network Model (Scarselli et al.)
- Target: learn a state embedding $\mathbf{h}_v \in \mathbb{R}^s$, each node can produce an output $\mathbf{o}_v$

$$\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]})$$

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v)$$

- Formulate above into matrix-like:

$$\mathbf{H} = F(\mathbf{H}, \mathbf{X})$$

$$\mathbf{O} = G(\mathbf{H}, \mathbf{X}_N)$$

- Note that *F* is a contraction map. According to Banach's fixed point theorem, the fixed point can be iteratively obtained by

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X})$$

- so that $\mathbf{H}(T) \approx \mathbf{H}$

Models – Graph Neural Networks

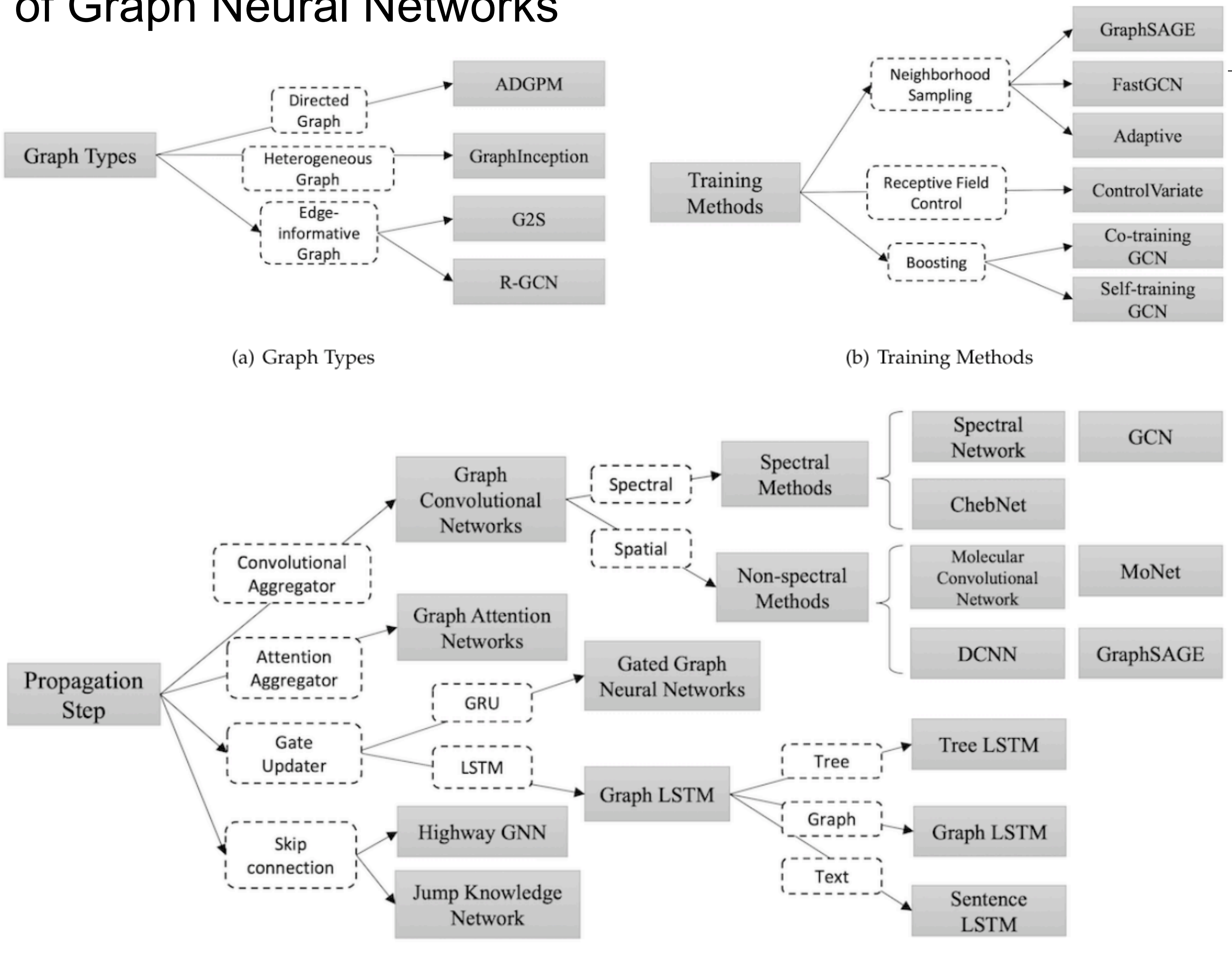
- The gradient of weights is computed from the loss

$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$

- The weights are updated according to the gradient
- Limitations:
  - Inefficient to update the hidden states of nodes iteratively for the fixed point
  - Different from hierarchical feature extraction methods due to parameter sharing
  - Not consider informative features on the edges
  - The fixed points make representation less informative for distinguishing each node

Models – Variants of Graph Neural Networks

- Categories



(a) Graph Types

(b) Training Methods

(c) Propagation Steps

Models – Variants of Graph Neural Networks

- Graph Types
    - Directed Graphs
        - ADGPM (Kampffmeyer et al.)

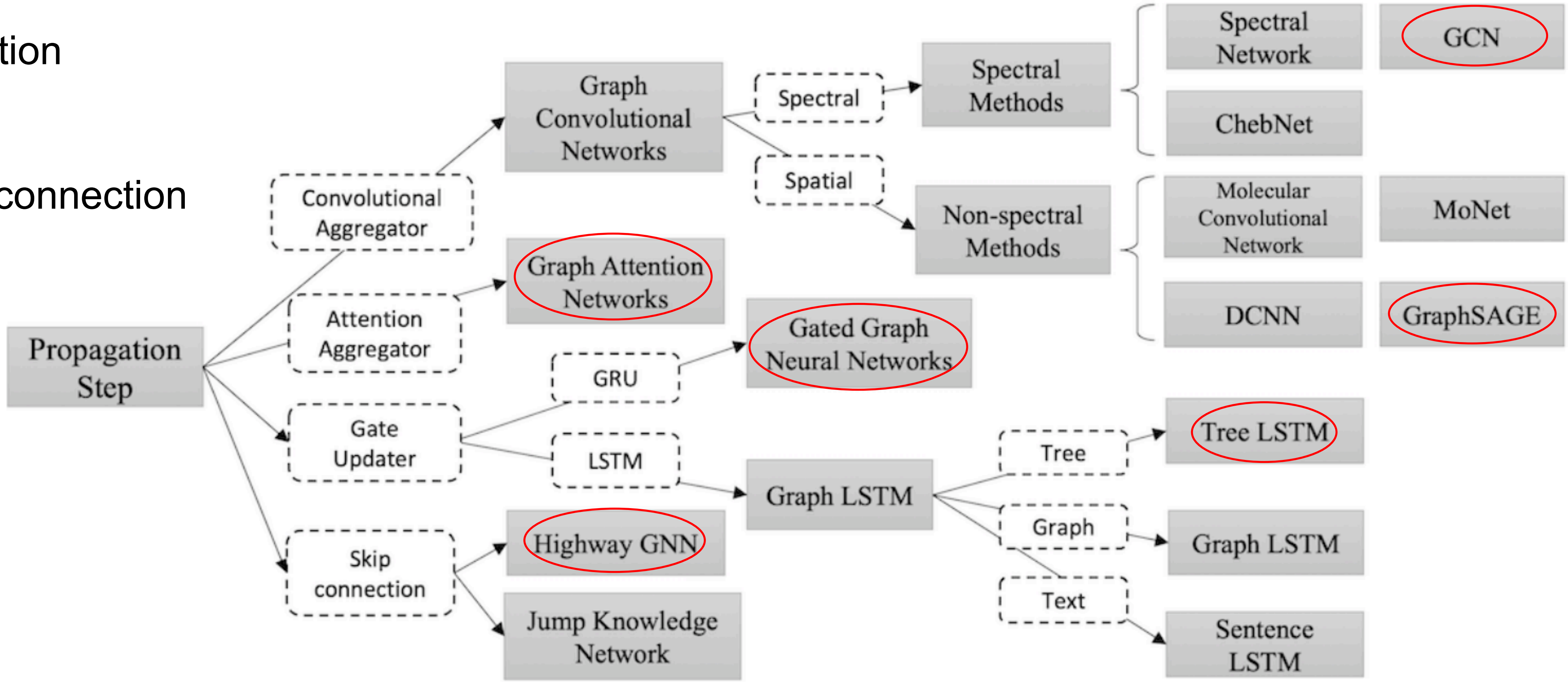
$$\mathbf{H}^t = \sigma(\mathbf{D}_p^{-1}\mathbf{A}_p\sigma(\mathbf{D}_c^{-1}\mathbf{A}_c\mathbf{H}^{t-1}\mathbf{W}_c)\mathbf{W}_p)$$

    - Heterogeneous Graphs
        - GraphInception (Zhang et al.)
            - Group the neighbors according to their node types and distances.
    - Graphs with Edge Information
        - G2S (Beck et al.)

$$\mathbf{h}_v^t = \rho(\frac{1}{|\mathcal{N}_v|}\sum_{u\in\mathcal{N}_v}\mathbf{W}_r(\mathbf{r}_v^t \odot \mathbf{h}_u^{t-1}) + \mathbf{b}_r)$$

Models – Variants of Graph Neural Networks

- Propagation Steps
  - Convolution
  - Attention
  - Gate
  - Skip connection

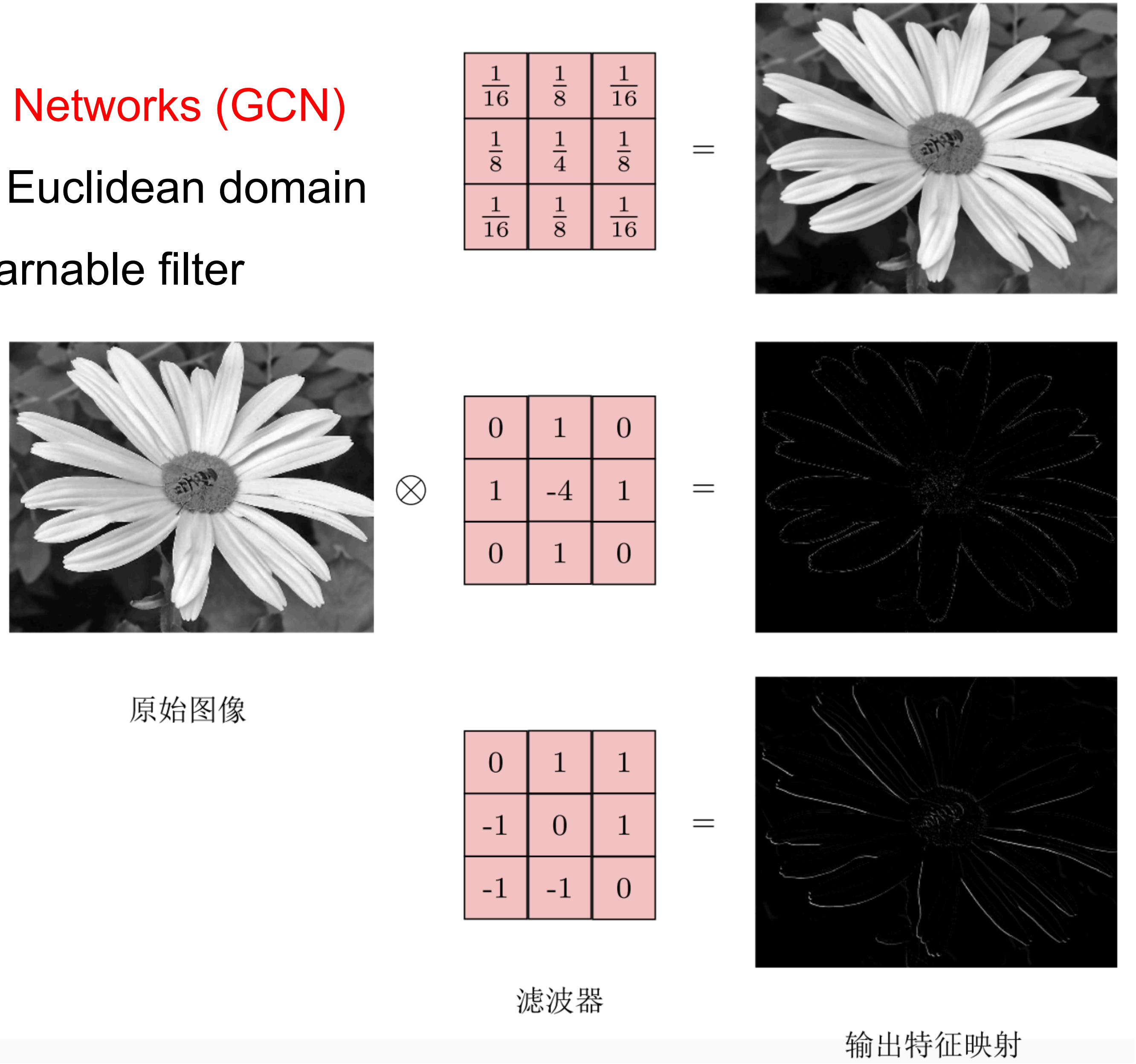


Models – Variants of Graph Neural Networks

- Propagation Steps
  - Convolution
    - Spectral: *GCN*
    - Spatial: *GraphSAGE*
  - Attention: *GAT*
  - Gate: *GGNN*
  - Skip connection: *Highway GCN*

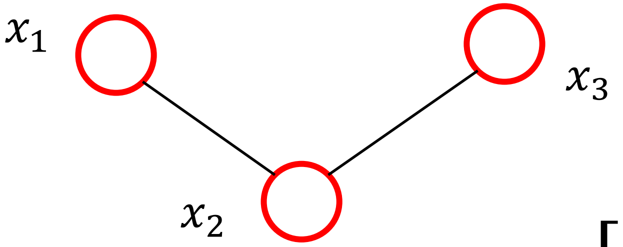
Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - Convolutions on Euclidean domain
  - Fixed filter -> Learnable filter



Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - Convolutions on non-Euclidean domain



$$A \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} x_2 \\ x_1 + x_3 \\ x_2 \end{bmatrix}$$

☹

$$L = D - A \qquad \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + x_3 \end{bmatrix}$$

☺

$$L^{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \qquad \begin{bmatrix} 1 & -\frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & 1 & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & 1 \end{bmatrix} \qquad \begin{bmatrix} x_1 - \frac{x_2}{\sqrt{2}} \\ -\frac{x_1}{\sqrt{2}} + x_2 - \frac{x_3}{\sqrt{2}} \\ -\frac{x_2}{\sqrt{2}} + x_3 \end{bmatrix}$$

☺

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - Convolution theorem: The Fourier transform of a convolution of two signals is the pointwise product of their Fourier transforms.

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

  - By applying inverse Fourier transform, we can write:

$$f * g = \mathcal{F}^{-1}\big\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\big\}$$

  - Fourier transform on signals:

$$F(\omega) = \mathcal{F}[f(t)] = \int f(t) e^{-i\omega t} dt$$

  - In contrast, Fourier transform on graph is

$$F(\lambda_l) = \hat{f}(\lambda_l) = \sum_{i=1}^{N} f(i) u_l^*(i)$$

  - Write the above in matrix-like: $\hat{f} = U^T f$

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - An original version from [(Bruna et al.)]
  - The convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian

Fourier transform

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{U}\mathbf{g}_\theta(\mathbf{\Lambda})\boxed{\mathbf{U}^T\mathbf{x}}$$

  - Where U is the matrix of eigenvectors the normalized graph Laplacian

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

  - Problems: potentially intense computation, non-spatially localized filters…

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - An original version from [(Bruna et al.)]
  - The convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian

Convolution in Fourier domain

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{U}\boxed{\mathbf{g}_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x}}$$

  - Where U is the matrix of eigenvectors the normalized graph Laplacian

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

  - Problems: potentially intense computation, non-spatially localized filters…

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - An original version from (Bruna et al.)
  - The convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian

Inverse Fourier transform

$$\mathbf{g}_\theta \star \mathbf{x} = \underbrace{\mathbf{U}\mathbf{g}_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x}}$$

  - Where U is the matrix of eigenvectors the normalized graph Laplacian

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

  - Problems: potentially intense computation, non-spatially localized filters…

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - A modified version from (Defferrard et al.)
  - The $\mathbf{g}_\theta(\mathbf{\Lambda})$ can be approximated by a truncated expansion in terms of Chebyshev polynomials $\mathbf{T}_k(x)$ up to Kth order:

$$\mathbf{g}_\theta \star \mathbf{x} \approx \sum_{k=0}^{K} \theta_k \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{x}$$

$$\tilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}_N$$

  - The convolution operation is K-localized
  - No need to compute the eigenvectors of the Laplacian

Models – Variants of Graph Neural Networks

- Graph Convolutional Networks (GCN)
  - Maybe the most popular version in today (Kipf et al.)
  - Let $\lambda_{max} \approx 2$

$$\mathbf{g}_{\theta'} \star \mathbf{x} \approx \theta'_0 \mathbf{x} + \theta'_1 \left(\mathbf{L} - \mathbf{I}_N\right) \mathbf{x} = \theta'_0 \mathbf{x} - \theta'_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{x}$$

  - Let $\theta = \theta'_0 = -\theta'_1$

$$\mathbf{g}_\theta \star \mathbf{x} \approx \theta \left(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}\right) \mathbf{x}$$

  - Use a renormalization trick $\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$

$$\mathbf{Z} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \mathbf{\Theta}$$

Models – Variants of Graph Neural Networks

- GraphSAGE (Hamilton and Ying et al.)
  - Non-spectral approaches define convolutions directly on the graph.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

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**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Models – Variants of Graph Neural Networks

- GraphSAGE (Hamilton and Ying et al.)
  - Aggregators:
    - Mean aggregator

$$\mathbf{h}_v^t = \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{t-1}\} \cup \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$$

    - LSTM aggregator
      - Larger expressive capability
      - Not permutation invariant
    - Pooling aggregator

$$\mathbf{h}_{\mathcal{N}_v}^t = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_u^{t-1} + \mathbf{b}\right), \forall u \in \mathcal{N}_v\})$$

Models – Variants of Graph Neural Networks

- Graph Attention Network (GAT) (Velickovic et al.)
  - Compute hidden states of each node by attending over its neighbors
  - Graph attentional layer computes the coefficients in the attention mechanism of the node pair *(i, j)* by:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]\right)\right)}$$

  - The final output of each node can be obtained by:

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\mathbf{h}_j\right)$$

  - Widely used in NLP. (Transformer)

Models – Variants of Graph Neural Networks

- Gated Graph Neural Network (GGNN) (Li et al.)
  - Use Gated Recurrent Unit (GRU) in the propagation step, unroll time steps to a fixed number
  - Use back propagation through time to compute gradients

$$
\begin{aligned}
\mathbf{a}_v^t &= \mathbf{A}_v^T [\mathbf{h}_1^{t-1} \dots \mathbf{h}_N^{t-1}]^T + \mathbf{b} \\
\mathbf{z}_v^t &= \sigma\left(\mathbf{W}^z \mathbf{a}_v^t + \mathbf{U}^z \mathbf{h}_v^{t-1}\right) \\
\mathbf{r}_v^t &= \sigma\left(\mathbf{W}^r \mathbf{a}_v^t + \mathbf{U}^r \mathbf{h}_v^{t-1}\right) \\
\widetilde{\mathbf{h}_v^t} &= \tanh\left(\mathbf{W}\mathbf{a}_v^t + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}\right)\right) \\
\mathbf{h}_v^t &= \left(1 - \mathbf{z}_v^t\right) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^t}
\end{aligned}
$$

  - In other works, LSTM are also used in a similar way as GRU, such as *Tree-LSTM*, *Graph LSTM*, *Sentence LSTM*…

Models – Variants of Graph Neural Networks

- Highway GCN
  - Experimentally, deeper models could not improve the performance, they could even perform worse!
  - A straightforward solution: Residual/Skip Connection
  - (Rahimi et al.) use layer-wise gate:

$$\mathbf{T}(\mathbf{h}^t) = \sigma\left(\mathbf{W}^t\mathbf{h}^t + \mathbf{b}^t\right)$$
$$\mathbf{h}^{t+1} = \mathbf{h}^{t+1} \odot \mathbf{T}(\mathbf{h}^t) + \mathbf{h}^t \odot (1 - \mathbf{T}(\mathbf{h}^t))$$

Models – General Frameworks

- Message Passing Neural Network (MPNN)
  - Unifies GNNs and GCNs
- Non-Local Neural Network (NLNN)
  - Unifies “self-attention”-style methods
- Graph Network (GN)
  - A more general framework, unifying MPNN, NLNN, Interaction Networks, CommNet…

Models – General Frameworks

- Message Passing Neural Network (MPNN)
  - Proposed by (Gilmer et al.)
  - Contains 2 phases:
    - Message passing

aggregate $$\mathbf{m}_v^{t+1} = \sum_{w \in \mathcal{N}_v} M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right)$$

update $$\mathbf{h}_v^{t+1} = U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right)$$

    - Readout

$$\hat{\mathbf{y}} = R(\{\mathbf{h}_v^T | v \in G\})$$

Models – General Frameworks

- Non-Local Neural Network (NLNN) (Wang et al.)
  - A non-local operation computes the response at a position as a weighted sum of the features at *all positions*
  - The generic non-local operation:

$$\mathbf{h}_i' = \frac{1}{\mathcal{C}(\mathbf{h})} \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j) g(\mathbf{h}_j)$$

  - Particular implementations:
    - Gaussian $f(\mathbf{h}_i, \mathbf{h}_j) = e^{\mathbf{h}_i^T \mathbf{h}_j}$
    - Embedded Gaussian $f(\mathbf{h}_i, \mathbf{h}_j) = e^{\theta(\mathbf{h}_i)^T \phi(\mathbf{h}_j)}$
    - Dot product $f(\mathbf{h}_i, \mathbf{h}_j) = \theta(\mathbf{h}_i)^T \phi(\mathbf{h}_j)$
    - Concatenation $f(\mathbf{h}_i, \mathbf{h}_j) = \mathrm{ReLU}(\mathbf{w}_f^T [\theta(\mathbf{h}_i) \| \phi(\mathbf{h}_j)])$

Models – General Frameworks

- Graph Network (GN)
  - Proposed by [(Battaglia et al.)]
  - Flexible representations
  - Configurable within-block structure
  - Composable multi-block architectures

Graph definition. In [27], a graph is defined as a 3-tuple $G = (\mathbf{u}, H, E)$ (here we use H instead of V for notation's consistency). $\mathbf{u}$ is a global attribute, $H = \{\mathbf{h}_i\}_{i=1:N^v}$ is the set of nodes (of cardinality N^v), where each $\mathbf{h}_i$ is a node's attribute. $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$ is the set of edges (of cardinality N^e), where each $\mathbf{e}_k$ is the edge's attribute, r_k is the index of the receiver node and s_k is the index of the sender node.

1) ϕ^e is applied per edge, with arguments $(\mathbf{e}_k, \mathbf{h}_{r_k}, \mathbf{h}_{s_k}, \mathbf{u})$, and returns $\mathbf{e}'_k$. The set of resulting per-edge outputs for each node i is, $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$. And $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of all per-edge outputs.
2) $\rho^{e\to h}$ is applied to E'_i, and aggregates the edge updates for edges that project to vertex i, into $\bar{\mathbf{e}}'_i$, which will be used in the next step's node update.
3) ϕ^h is applied to each node i, to compute an updated node attribute, $\mathbf{h}'_i$. The set of resulting per-node outputs is, $H' = \{\mathbf{h}'_i\}_{i=1:N^v}$.
4) $\rho^{e\to u}$ is applied to E', and aggregates all edge updates, into $\bar{\mathbf{e}}'$, which will then be used in the next step's global update.
5) $\rho^{h\to u}$ is applied to H', and aggregates all node updates, into $\bar{\mathbf{h}}'$, which will then be used in the next step's global update.
6) ϕ^u is applied once per graph and computes an update for the global attribute, $\mathbf{u}'$.

Applications

- Structural
  - Physics
  - Chemistry and biology
  - Knowledge graph
- Non-structural
  - Image
  - Text
- Others
  - Generative models
  - Combinatorial optimization
  - …

Open Problems

- GNN models are not good enough to offer satisfying solutions for any graph in any condition.
- There exists some problems:
  - Shallow structure
    - Most of GNNs are no more than 3 layers (over-smoothing)
  - Dynamic graphs
    - GNN can not change adaptively
  - Non-structural scenarios
    - How to model image/text as a graph?
  - Scalability
    - Hard to scale (social networks, recommendation systems…)

Conclusion

- The world is compositional, or at least, we understand it in compositional terms.

> *...[a human mental model] has a similar relation-structure to that of the process it imitates. By 'relation-structure' I do not mean some obscure non-physical entity which attends the model, but the fact that it is a working physical model which works in the same way as the process it parallels... physical reality is built up, apparently, from a few fundamental types of units whose properties determine many of the properties of the most complicated phenomena, and this seems to afford a sufficient explanation of the emergence of analogies between mechanisms and similarities of relation-structure among these combinations without the necessity of any theory of objective universals.* (Craik, 1943, page 51-55)

– "The Nature of Explanation" (Kenneth Craik, 1943)

Thanks!

Reporter: Tianxiang Sun (孙天祥，19110240005)

txsun19@fudan.edu.cn