Graph Neural Network

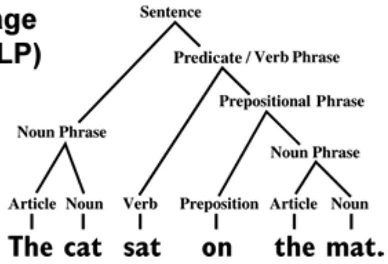
Jiahui Chen
Department of Mathematical Sciences
University of Arkansas
Reference: Kipf's slides

Deep Learning

IMAGENET

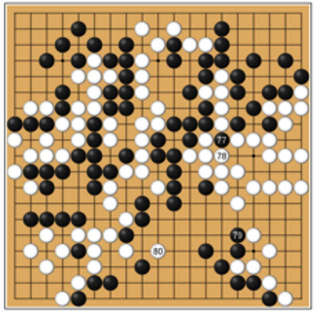
Speech data

Natural language processing (NLP)



...

Grid games



Deep neural nets that exploit:

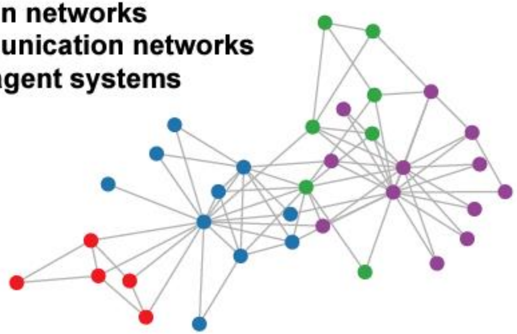
- translation equivariance (weight sharing)
- hierarchical compositionality

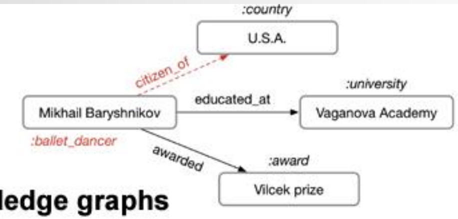
[Thomas Kipf, University of Amsterdam]

Graph Structured Data

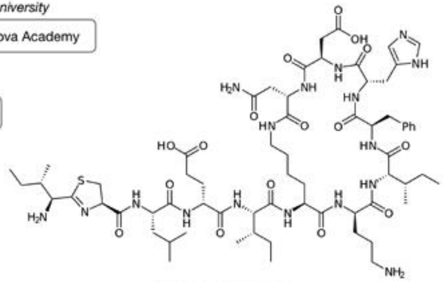
A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems



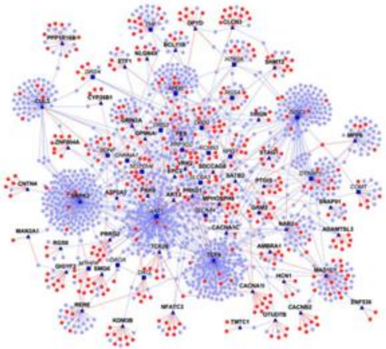


Knowledge graphs



Molecules

Protein interaction networks



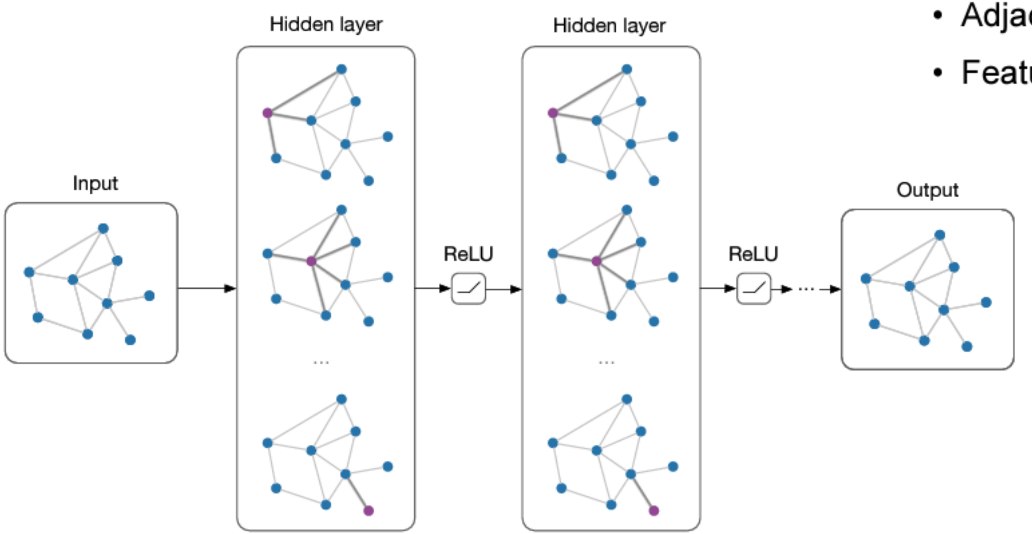
Road maps

How to apply the standard deep learning architectures?

[Thomas Kipf, University of Amsterdam]

Graph Neural Networks (GNNs)

The bigger picture:

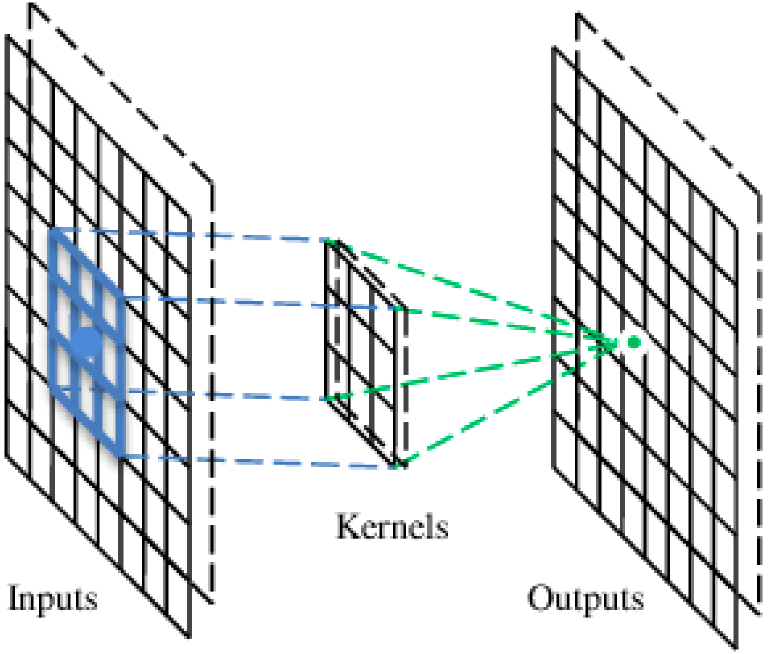


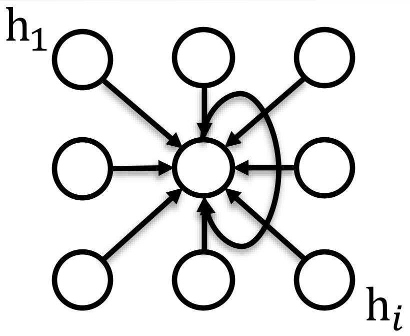
Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

Main idea: Pass messages between pairs of nodes & agglomerate

Graph Convolutional Network





Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

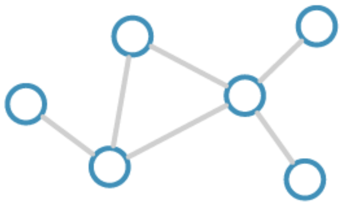
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

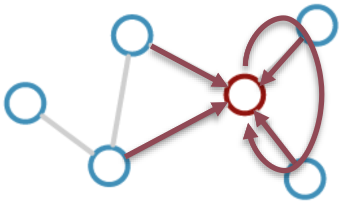
$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Graph Convolutional Network

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transudative and inductive settings

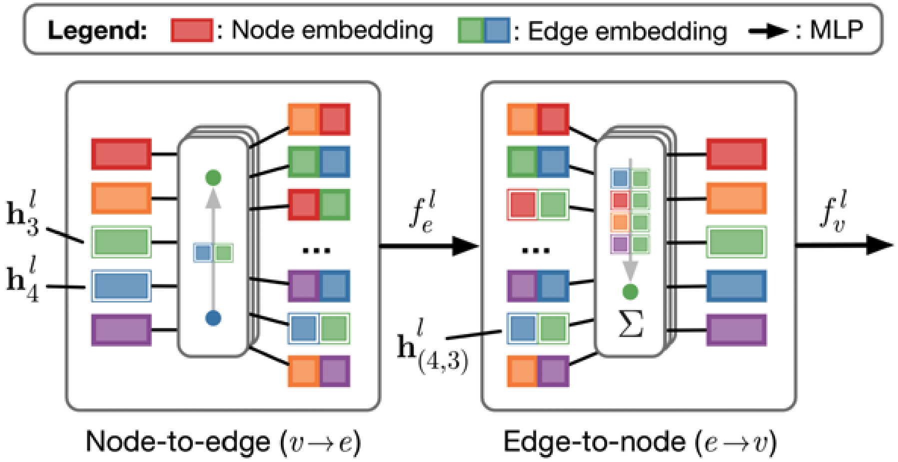
Limitations:

- Requires gating mechanism/residual connections for depth
- Only indirect support for edge features

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

GCNs with Edge Embeddings



Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

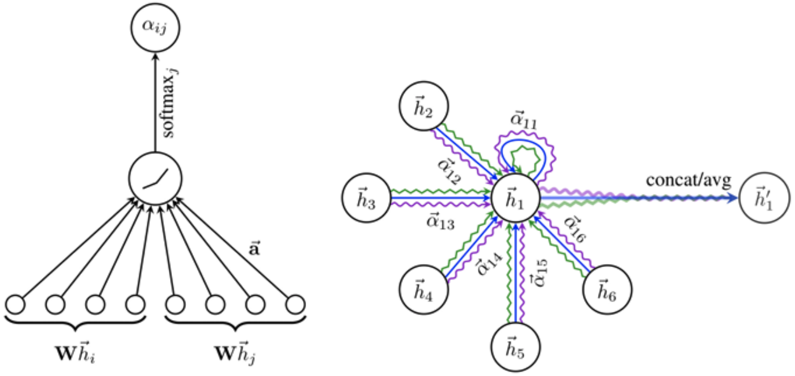
Pros:

- Supports edge features
- More expressive than GNN
- Support sparse matrix

Cons:

- Need to store intermediate edge-based activations
- Difficult to implement with subsampling
- In practice limited to small graph

Graph Neural Network with Attention



[Figure from Veličković et al. (ICLR 2018)]

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

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

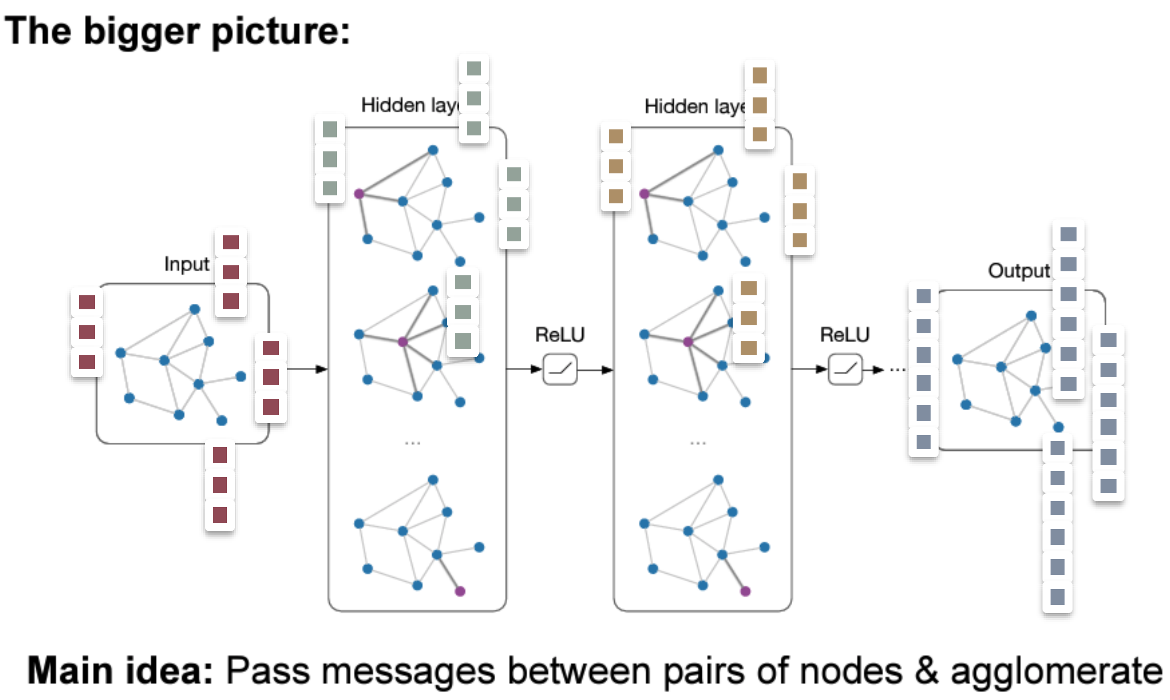
Pros:

- No need to store intermediate edge-based activation vectors (when using dot-product attn.)
- Slower than GCNs but faster then GNNs with edge embeddings

Cons:

- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

Graph Pooling



Graph Pooling:

- Max-pooling
- Min-pooling
- Mean-pooling