CS 190I
Deep Learning
Graph Neural Networks

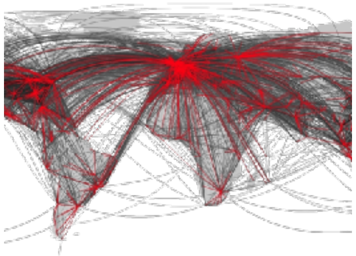
Lei Li (leili@cs)
UCSB

Recap

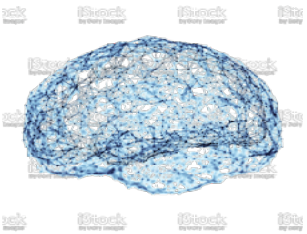
	training objective	backbone	size(#params)	training data (#tokens)
ELMo	next token prediction	two separate LSTM	94M	5.5 billion
BERT	masked token prediction + next sentence prediction	Transformer Encoder	110M 340M	3.3 billion
GPT-3	next token prediction	Transformer Decoder	175B	500 billion

Graph Data is everywhere

Social Graphs

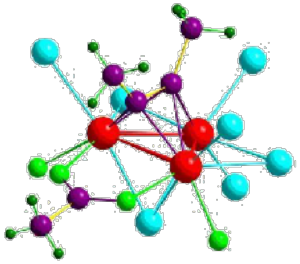


Transportation Graphs

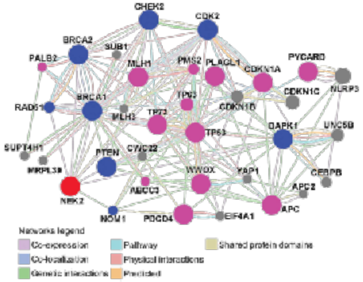


Brain Graphs

Web Graphs



Molecular Graphs

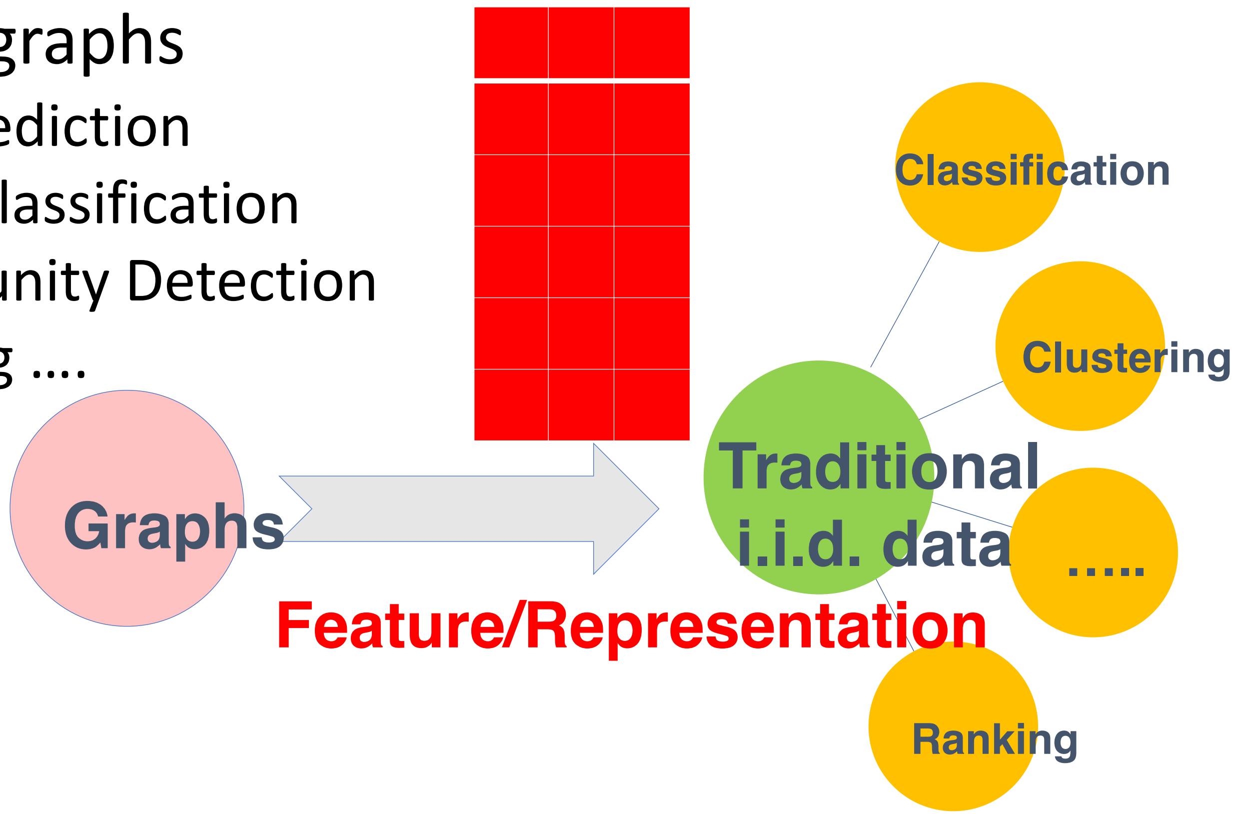


Gene Graphs

ML on Graphs

Numerous real-world problems can be summarized as a set of tasks on graphs

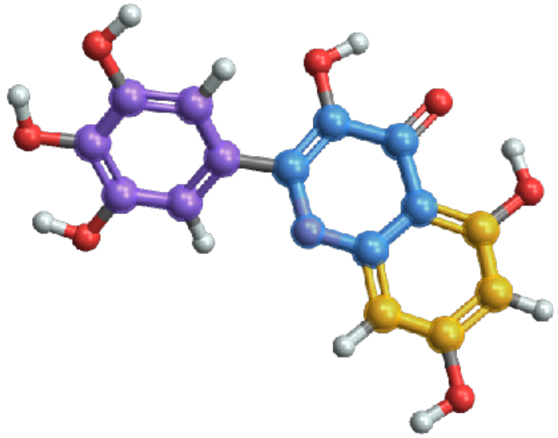
- Link prediction
- Node Classification
- Community Detection
- Ranking ....

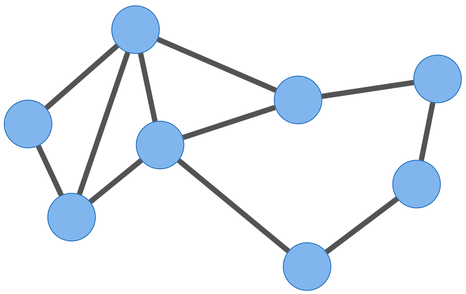


Slides adapted from Yao Ma & Jiliang Tang@MSU

Example: predict toxicity of a drug

Toxic?

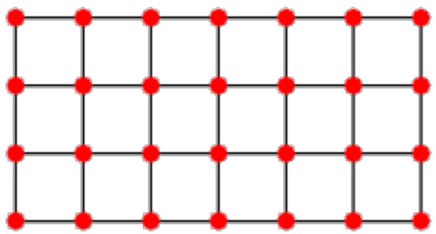


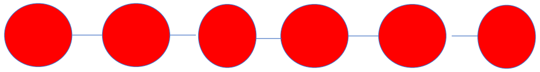


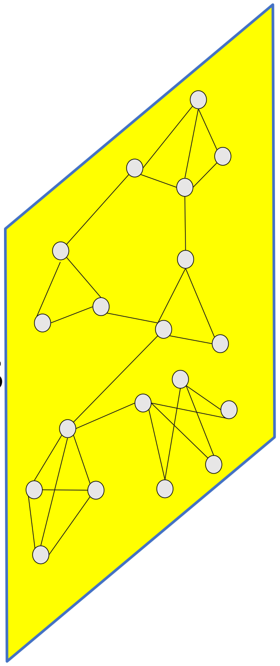
Deep Learning Meets Graphs: Challenges

Traditional DL is designed for simple grids or sequences

- CNNs for fixed-size images/grids
- RNNs for text/sequences



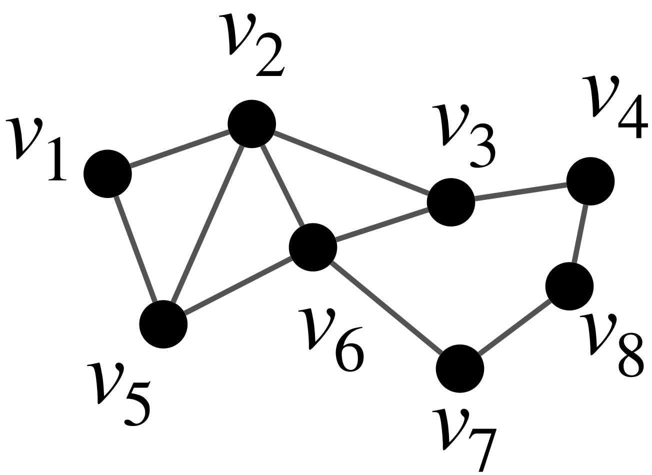




But nodes on graphs have different connections

- Arbitrary neighbor size
- Complex topological structure
- No fixed node ordering

Graph Representation



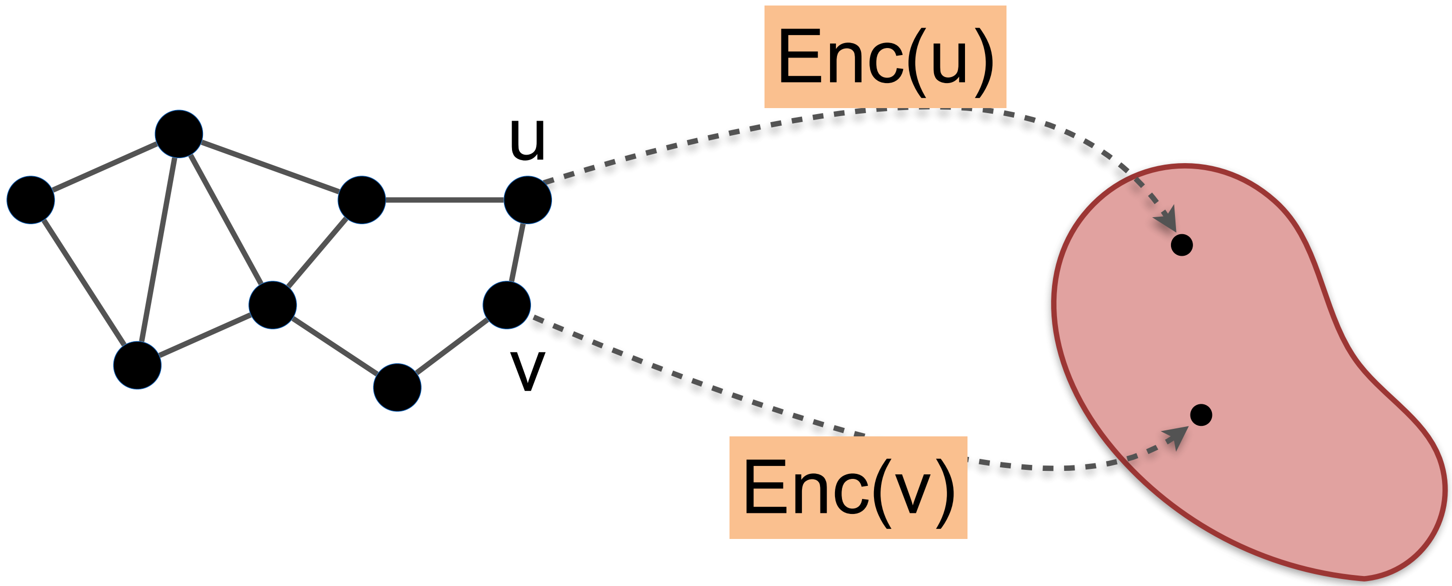
Graph: $G = \{V, E\}$

Nodes: $V = \{v_1, v_2, \ldots, v_N\}$

Edges: $E = \{e_1, e_2, \ldots, e_M\} \subset V \times V$

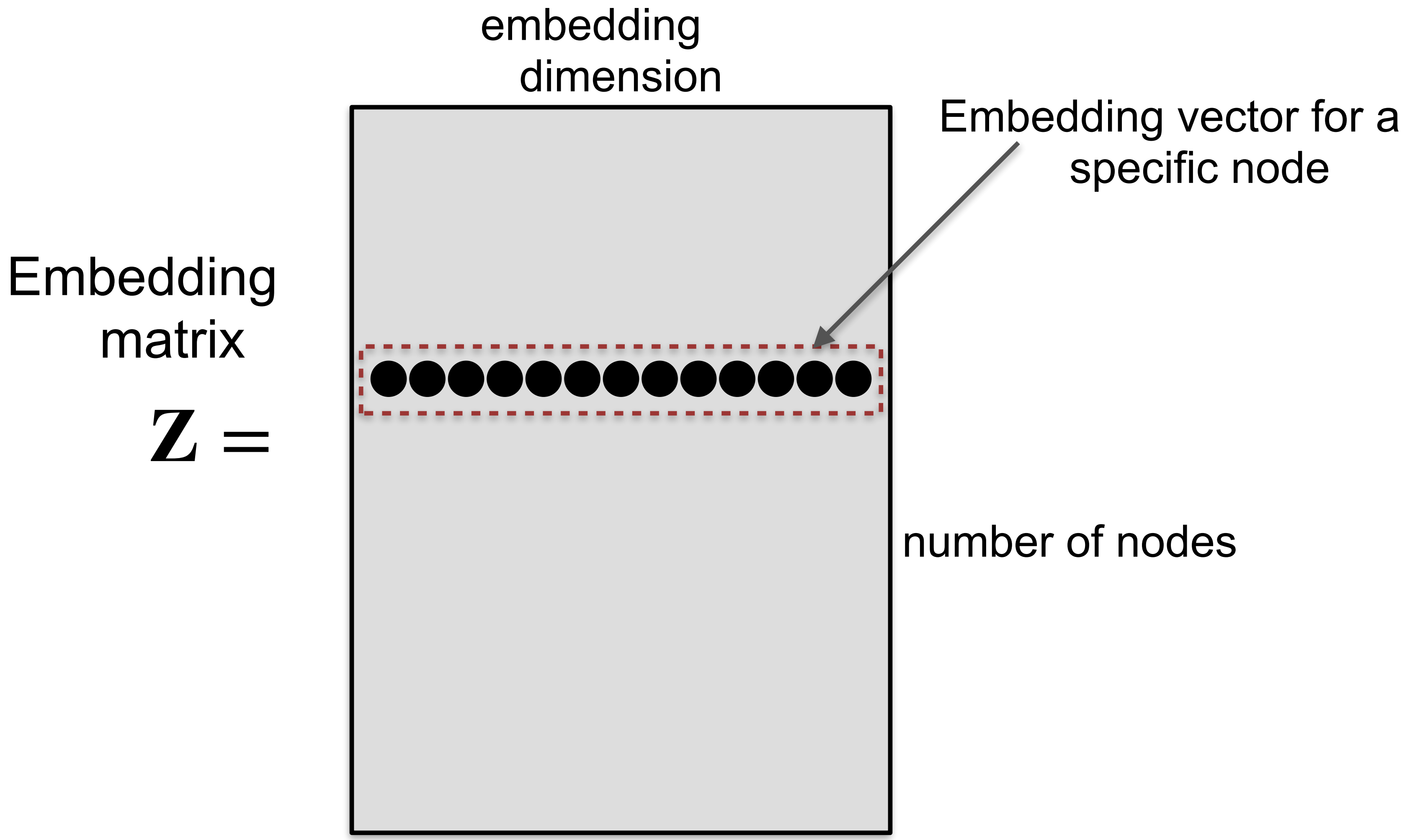
Node Embedding

$$Enc(\,\cdot\,) : V \to \mathbb{R}^d$$

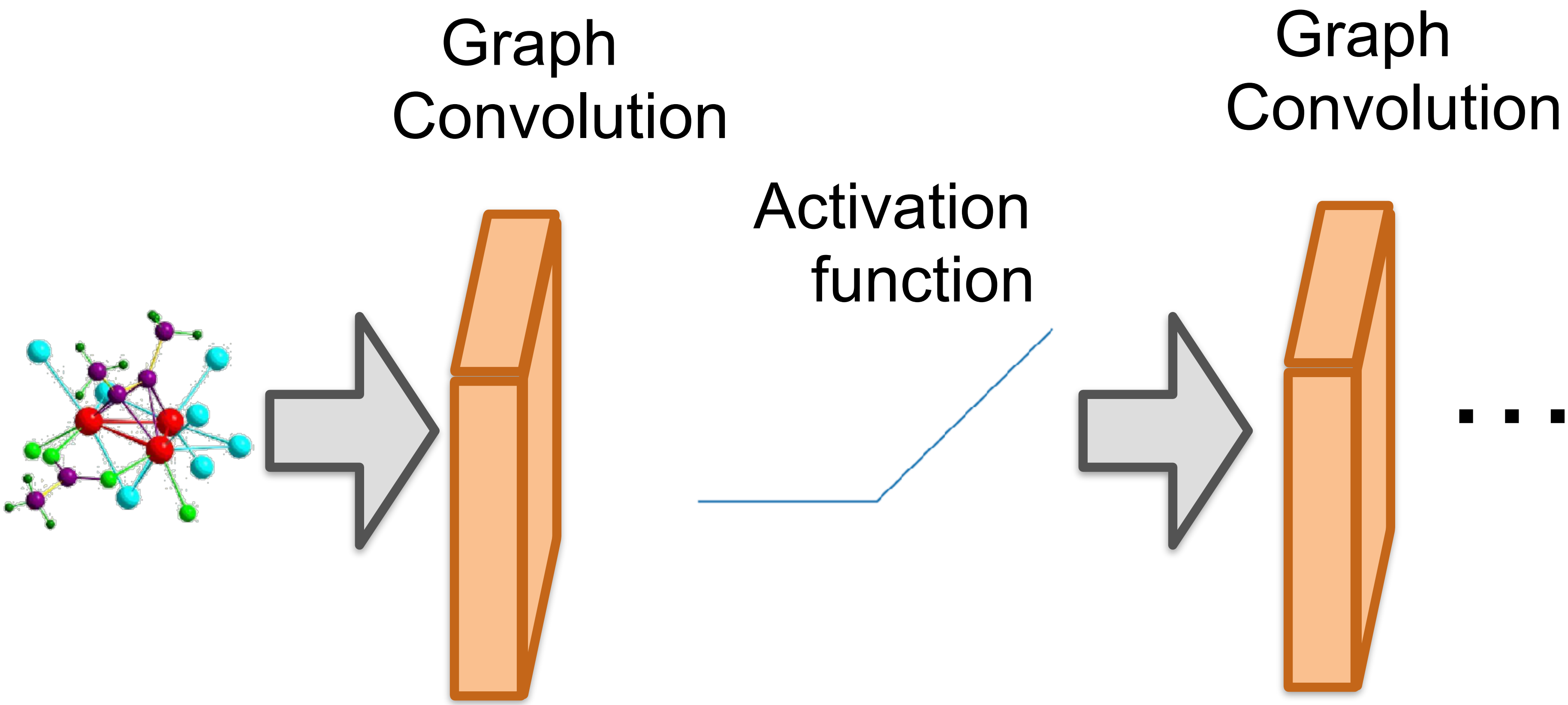


“Shallow” Node Embedding

- is just a lookup-table



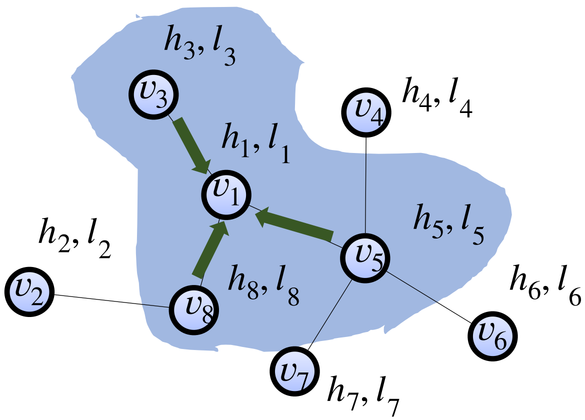
Deep Graph Neural Network



Output is embedding matrix for nodes
for further downstream tasks: e.g. node classification

Graph Neural Network

Every node's neighbor defines a convolutional kernel



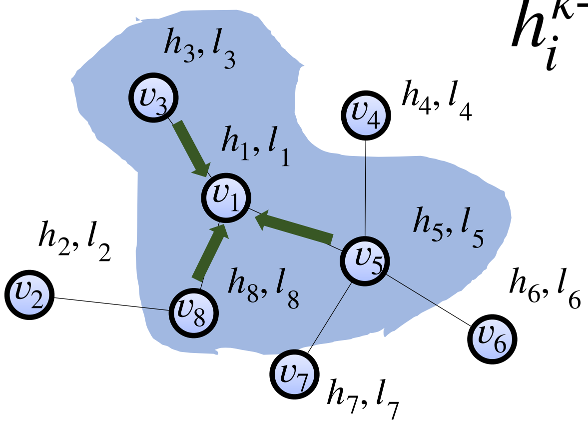
aggregate information
from its neighbors

Aggregate Neighbors

h_i: node (hidden) embedding vector

aggregate information from its neighbors

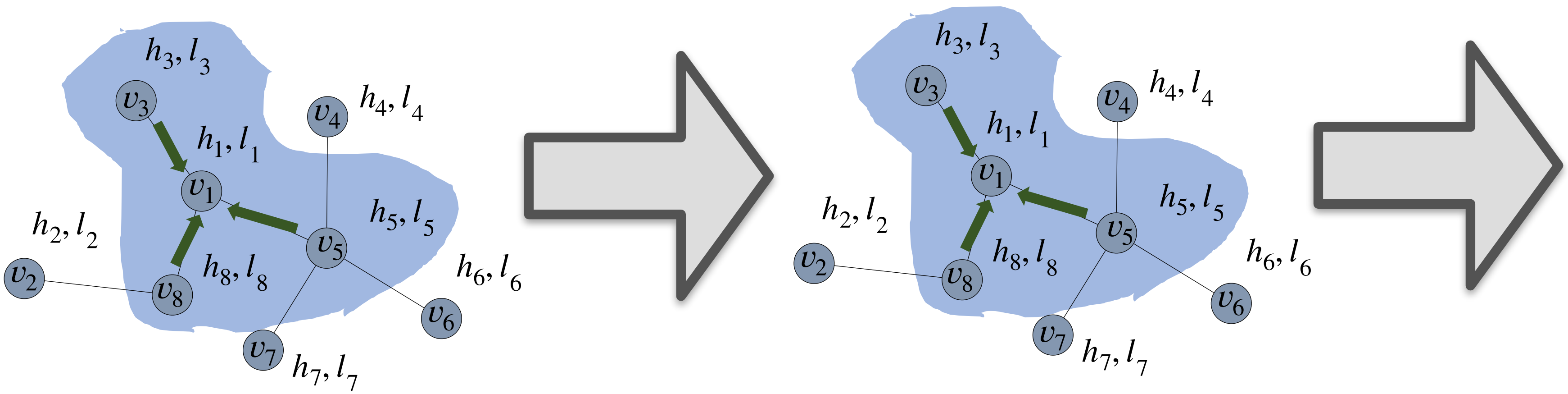
$$h_i^{k+1} = \text{Aggregate}_{v_j \in N(v_i)} f(h_i^k, h_j^k), \forall v_i \in V$$



$N(v_i)$: Neighbors of the node v_i.

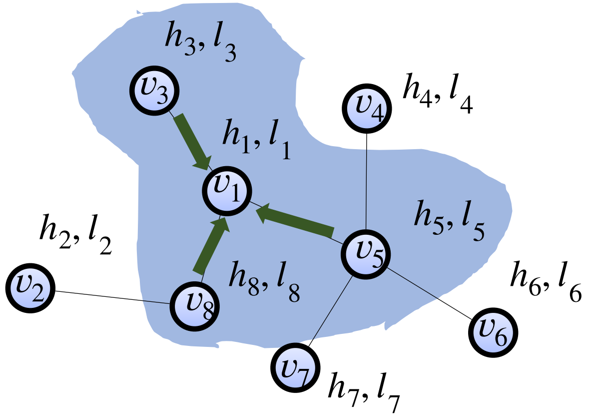
$f(\cdot)$: Feedforward network.

Multiple Computation Layers



A Simple Graph Convolution Layer

- Simple approach: averaging neighbor's message and apply nonlinear transformation



initial embedding: $h_i^0 = x_i$

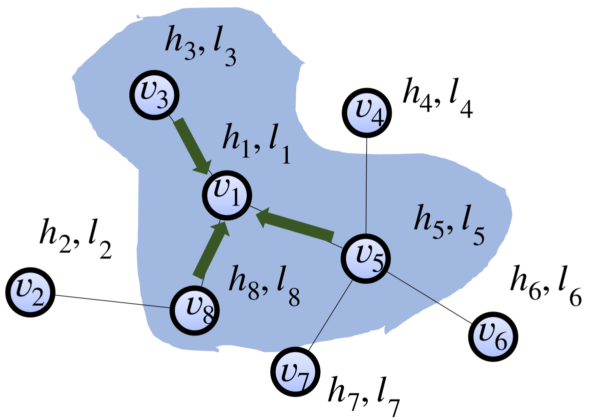
computing next layer:

$$h_i^{k+1} = \sigma(W_k \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} h_j^k + B_k h_i^k)$$

$$h_1^2 = tanh\left(W_1 \cdot \frac{1}{3}(h_3^1 + h_5^1 + h_8^1) + B_1 h_1^1 \right)$$

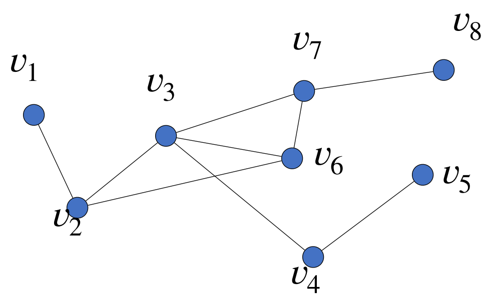
A Simple Graph Convolution Layer

- More layers:



$$h_1^{(3)} = tanh\left(W_2 \cdot \frac{1}{3}(h_3^{(2)} + h_5^{(2)} + h_8^{(2)}) + B_2 h_1^{(2)} \right)$$

Matrix Representations of Graphs



Adjacency Matrix: $A[i,j] = 1$ if v_i is adjacent to v_j

$A[i,j] = 0$, otherwise

Adjacency Matrix $\boldsymbol{A}$

$$
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
$$

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representation of GCN

- Neighbor Aggregation can be performed efficiently using matrix operations

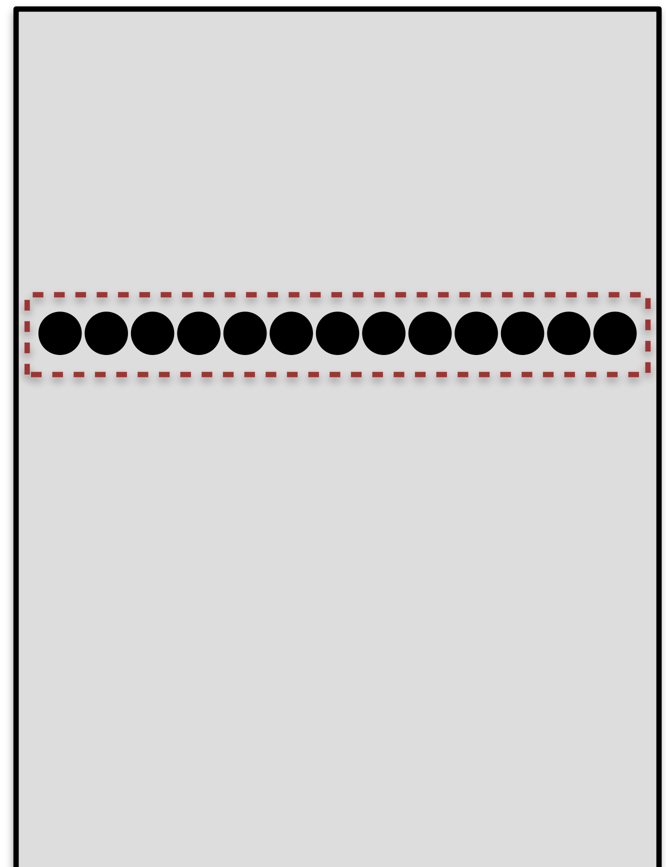
$$H^k = [h_1^k, \ldots, h_{|V|}^k]^T$$

Then $\displaystyle\sum_{v_j \in N(v_i)} h_j^k = A_{i,:} H^k$

Let D be diagonal matrix

$$D_{i,i} = \text{Degree}(v_i) = \sum_j A_{i,j}$$

Then $\displaystyle\frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} h_j^k = D^{-1} A H^k$



Matrix Representation of GCN

- Neighbor Aggregation can be performed efficiently using matrix operations

$$H^k = [h_1^k, \ldots, h_{|V|}^k]^T$$

$$\tilde{A} = D^{-1}A$$

$$H^{k+1} = \sigma(\tilde{A}H^k \cdot W_k^T + H^k B_k^T)$$

Graph Convolution Network

- Neighbor Aggregation can be performed efficiently using matrix operations
- To make $\tilde{A}$ symmetric

$$H^k = [h_1^k, \ldots, h_{|V|}^k]^T$$

$$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

$$H^{k+1} = \sigma(\tilde{A} H^k \cdot W_k^T + H^k B_k^T)$$

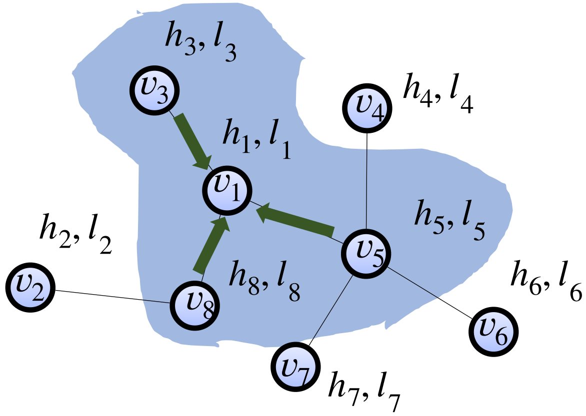
Prediction Layer

- For node classification:

$$o_i = \mathrm{Softmax}(h_i^{(m)})$$

- For graph classification:

$$o = \mathrm{Softmax}(\frac{1}{N}\sum_i h_i^{(m)})$$

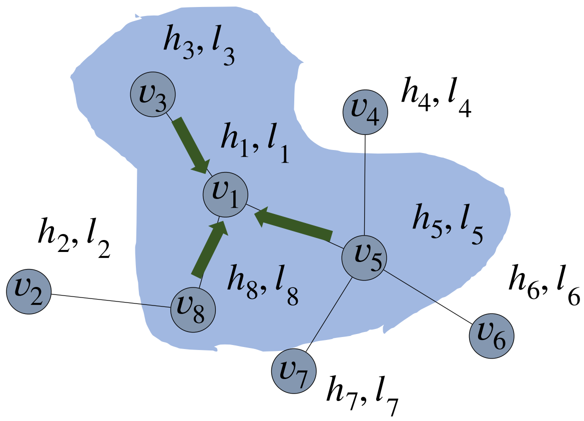


Property: Equivariant

- the embeddings computed from graph convolution layers is invariant to node permutation

$$h_i^0 = x_i$$

$$h_i^{k+1} = \sigma\left(W_k \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} h_j^k + B_k h_i^k\right)$$



Model Training

- Parameters: weight matrix for each layer

$$h_i^{k+1} = \sigma(\boxed{W_k} \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} h_j^k + \boxed{B_k} h_i^k)$$

- Supervised training: e.g. Node classification
  - Linked nodes have similar embedding

$$L = \sum_i CE(y_i, f(h_i^K)) \qquad f_i = \text{Softmax}(h_i^{(K)})$$

  - y_i is node label

Model Training

- Parameters: weight matrix for each layer

$$h_i^{k+1} = \sigma\left(\boxed{W_k}\frac{1}{|N(v_i)|}\sum_{v_j \in N(v_i)} h_j^k + \boxed{B_k} h_i^k\right)$$

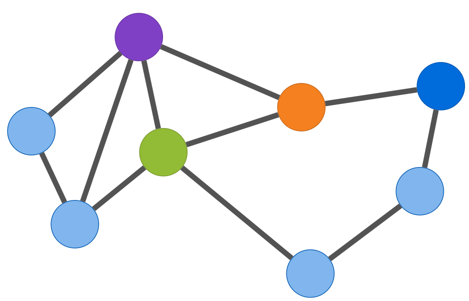
- Unsupervised training:
  - Linked nodes have similar embedding

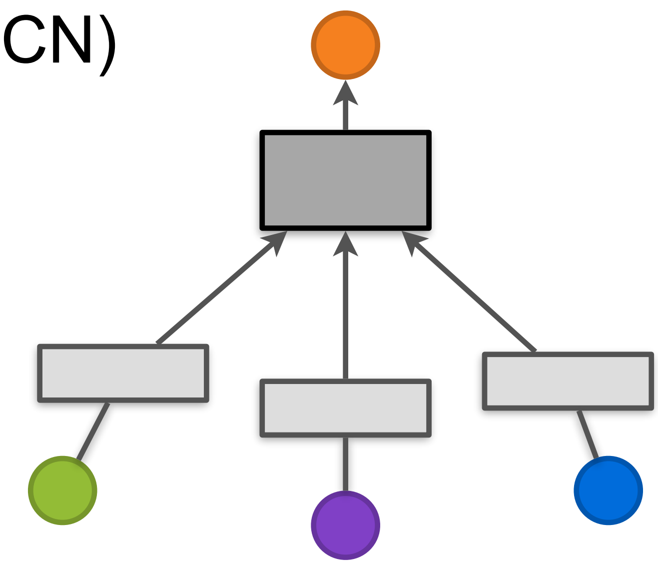
$$L = \sum_{i,j} CE(y_{i,j}, Sim(h_i^K, h_j^K))$$

  - $y_{i,j} = 1$ if there is edge from v_i to v_j
  - Similarity can be defined in many ways: e.g. inner product $h_i \cdot h_j$

Generic GNN framework

- GNN layer = message passing + Aggregation
  - different design choices under this framework
  - Graph convolutional network (GCN)
  - GraphSAGE
  - GAT

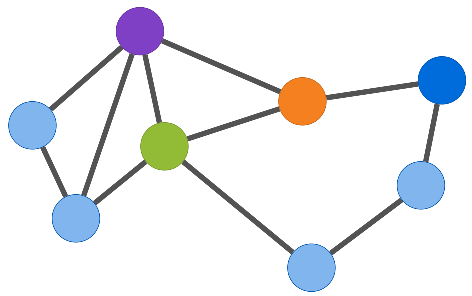


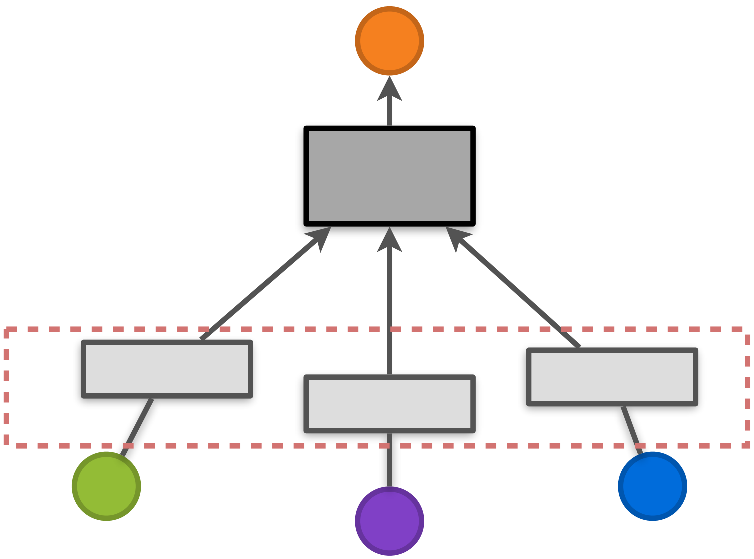


Message Computation

- Each node will create a message
- e.g. Linear projection

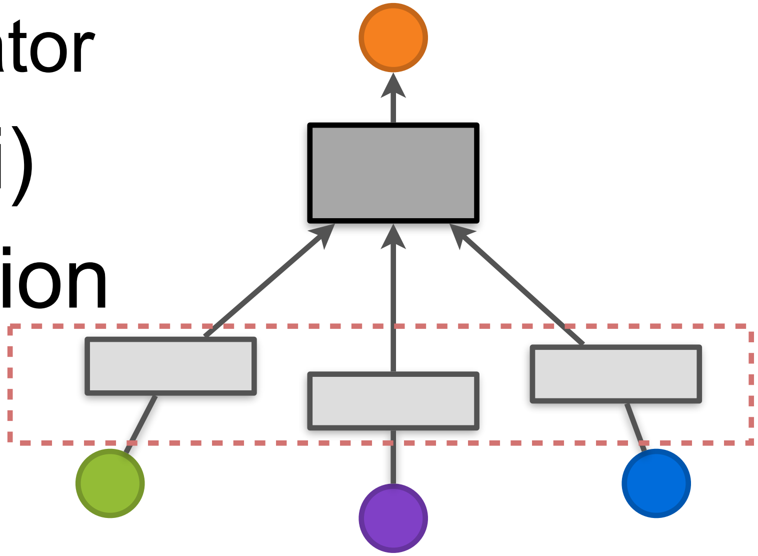
$$m_i^k = W_k \cdot h_i^{(k)}$$

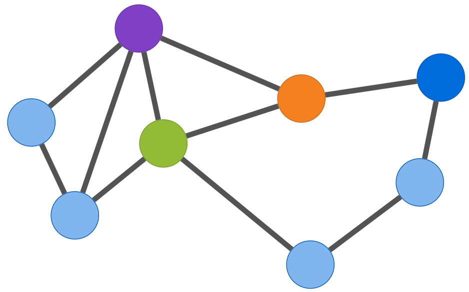




Aggregation/Pooling

- Each node will aggregate messages from its neighbors
- e.g.
  - Sum, Mean, Max operator
- Concat(AGG{m_j}, m_i)
- Apply nonlinear activation

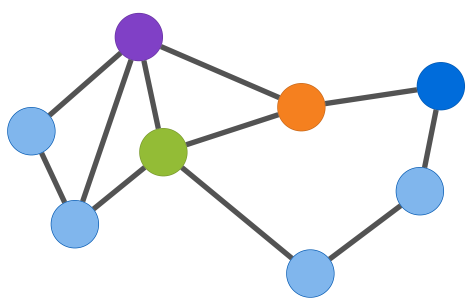


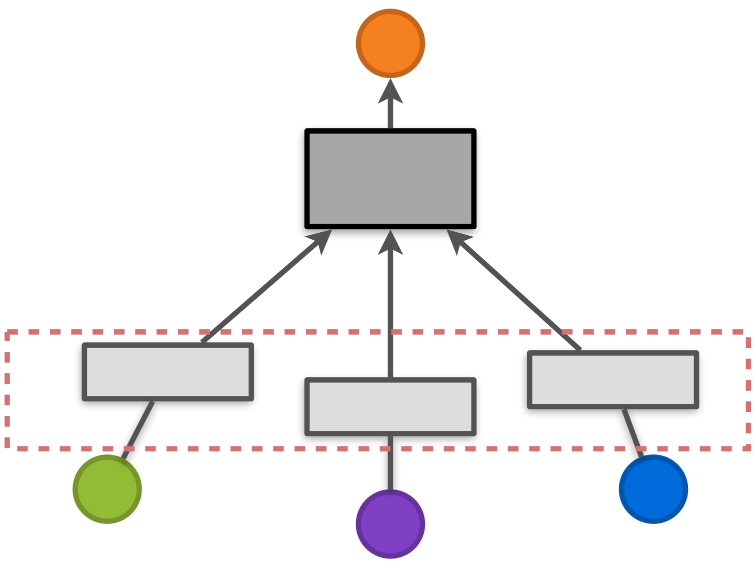


GraphSAGE

$$h_i^{k+1} = \sigma\left(W_k \cdot \mathrm{CONCAT}\left(h_i^k, \mathrm{AGG}(\{h_j^k, \forall v_j \in N(v_i)\}) \right) \right)$$

AGG can be designed in multiple ways, like pooling (sum, avg, max)



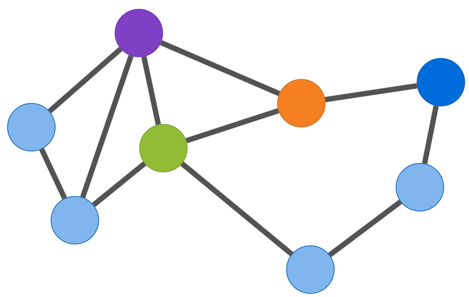


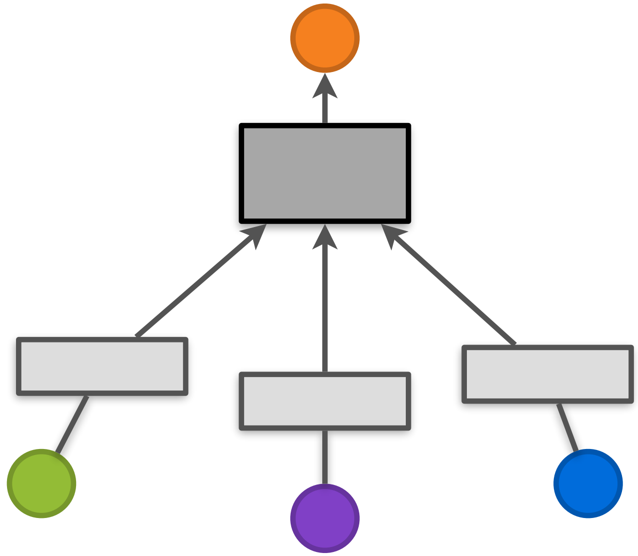
Graph Attention Network (GAT)

$$h_i^{k+1} = \sigma\left(\sum_{v_j \in N(v_i)} \alpha_{ij} W_k h_{v_j}^k\right)$$

attention weight

$$\alpha_{ij} = \text{Attention}(W_k h_i, W_k h_j) = \frac{\exp(W_k h_i)^T W_k h_j}{\sum_{j'} \exp(W_k h_i)^T W_k h_{j'}}$$

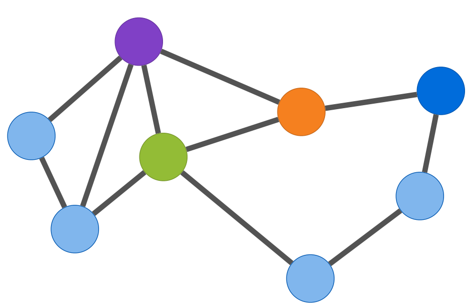


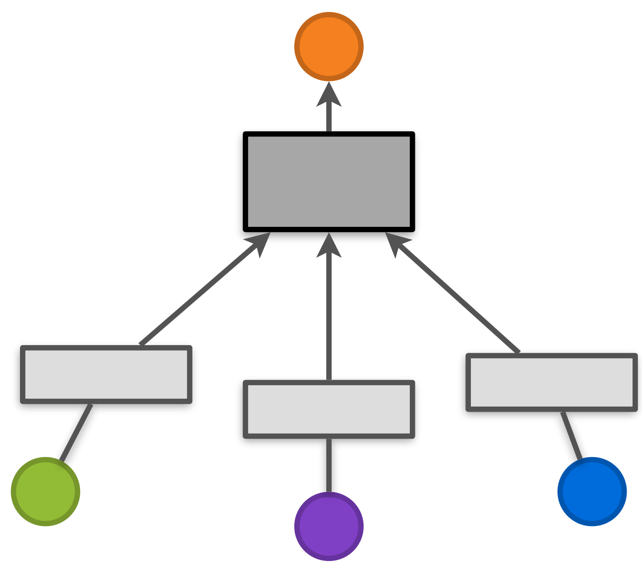


Multi-head Attention for GAT? Yes

$$h_i^{k+1} = \sigma\Big(\sum_{v_j \in N(v_i)} \alpha_{ij} W_k h_{v_j}^k\Big)$$

$$\alpha_{ij} = \mathrm{Attention}(W_k h_i, W_k h_j) = \frac{\exp(W_k h_i)^T W_k h_j}{\sum_{j'} \exp(W_k h_i)^T W_k h_{j'}}$$

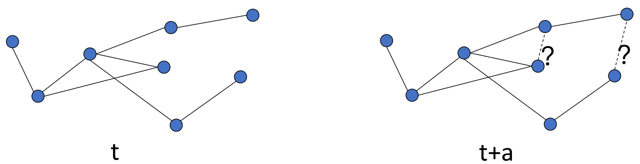




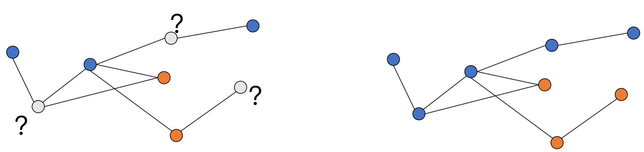
Tasks on Graph-Structured Data

Node-level

Link Prediction

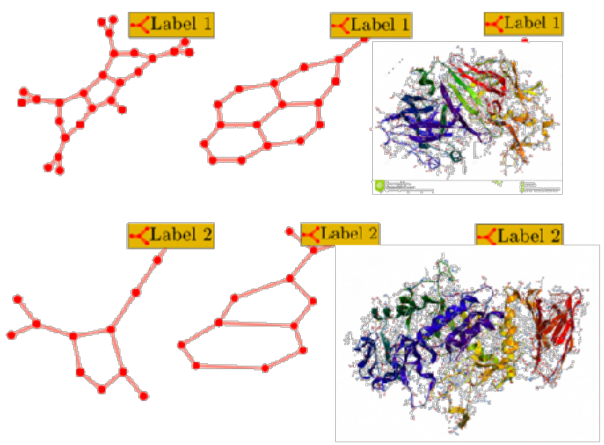


Node Classification

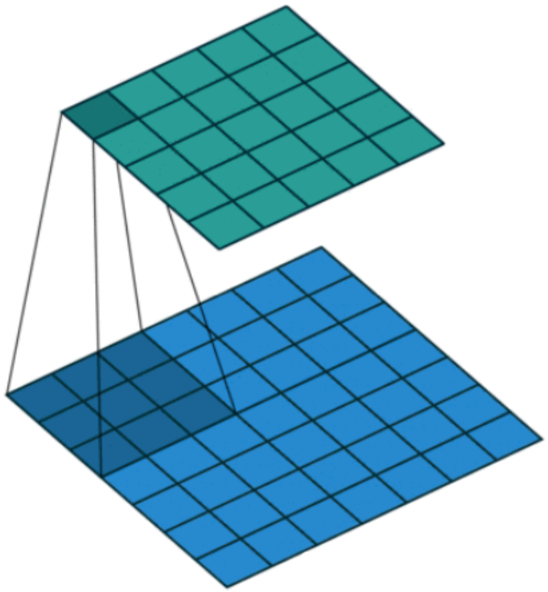


Graph-level

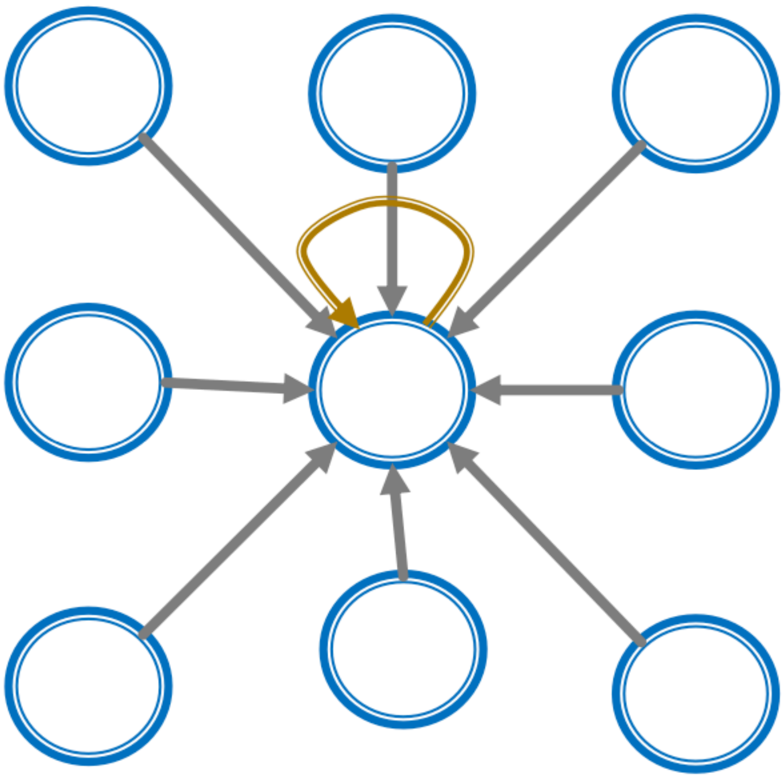
Graph Classification



Relation between GNN and CNN



Image



Graph

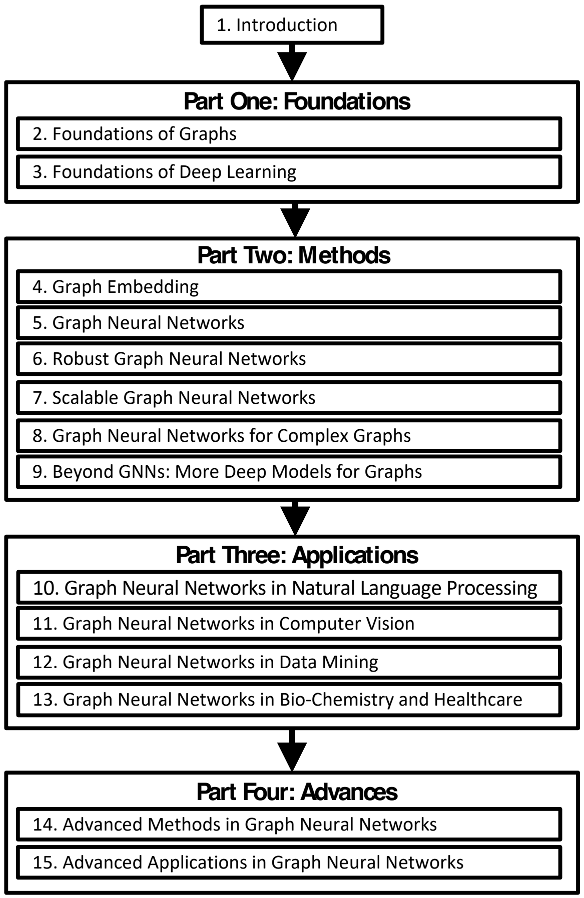
CNN can be viewed as a special GNN on grid graph

GNN vs. Transformer

- Transformer is special GNN on a full-connected graph

Book: Deep Learning on Graphs

https://cse.msu.edu/~mayao4/dlg_book/



Summary

- Graph neural network
  - message passed along graph edges
  - aggregate message/embedding by FFN
  - many variants

Next Up

- Variational Auto-Encoder