# Graph Neural Networks (GAT, Pin SAGE)

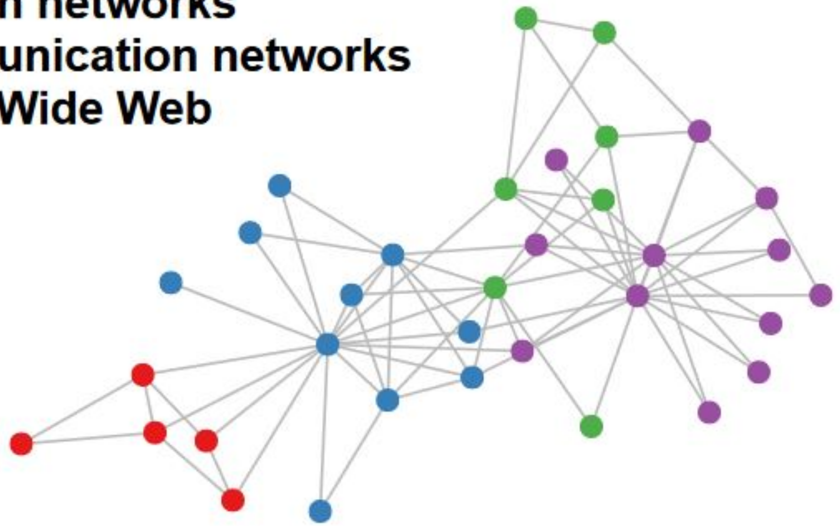
1/29/2021

Mahya MK

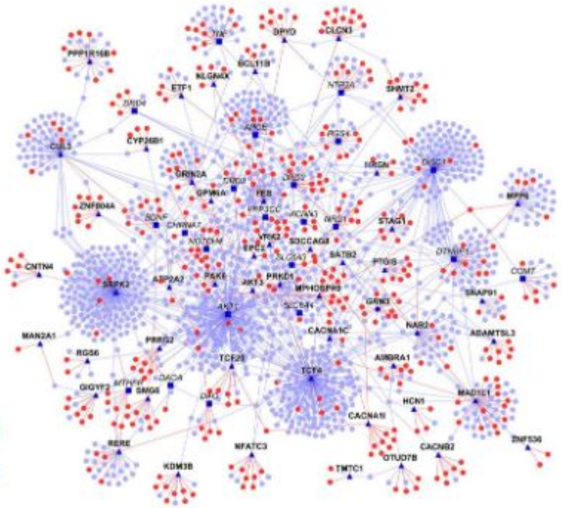
# Introduction

- **Data can be represented in the form of graphs**

**Social networks**
**Citation networks**
**Communication networks**
**World Wide Web**

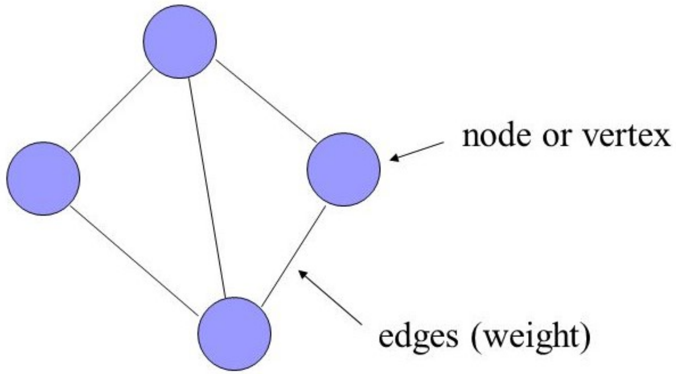


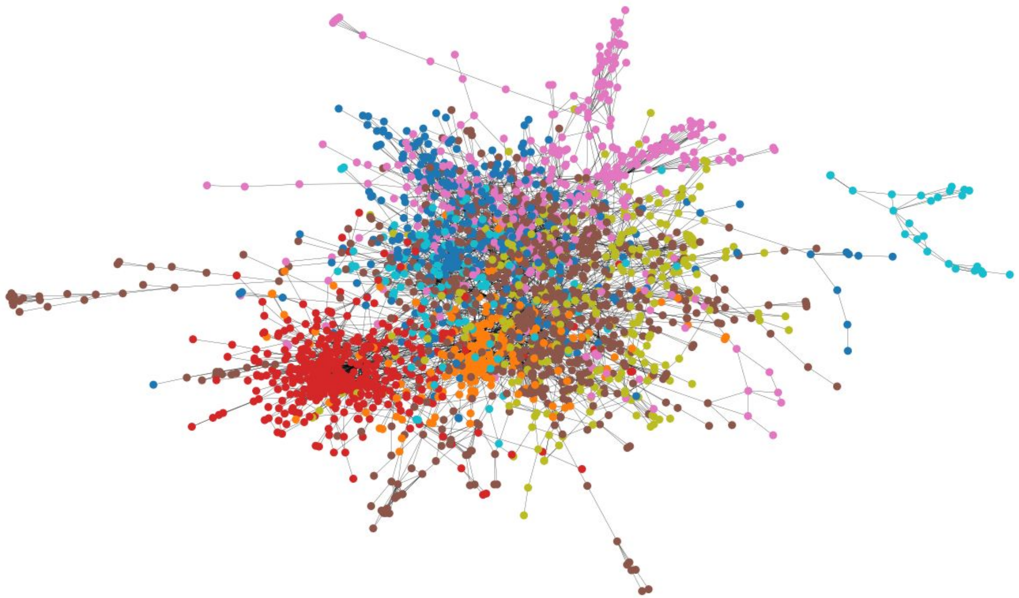
**Protein interaction networks**



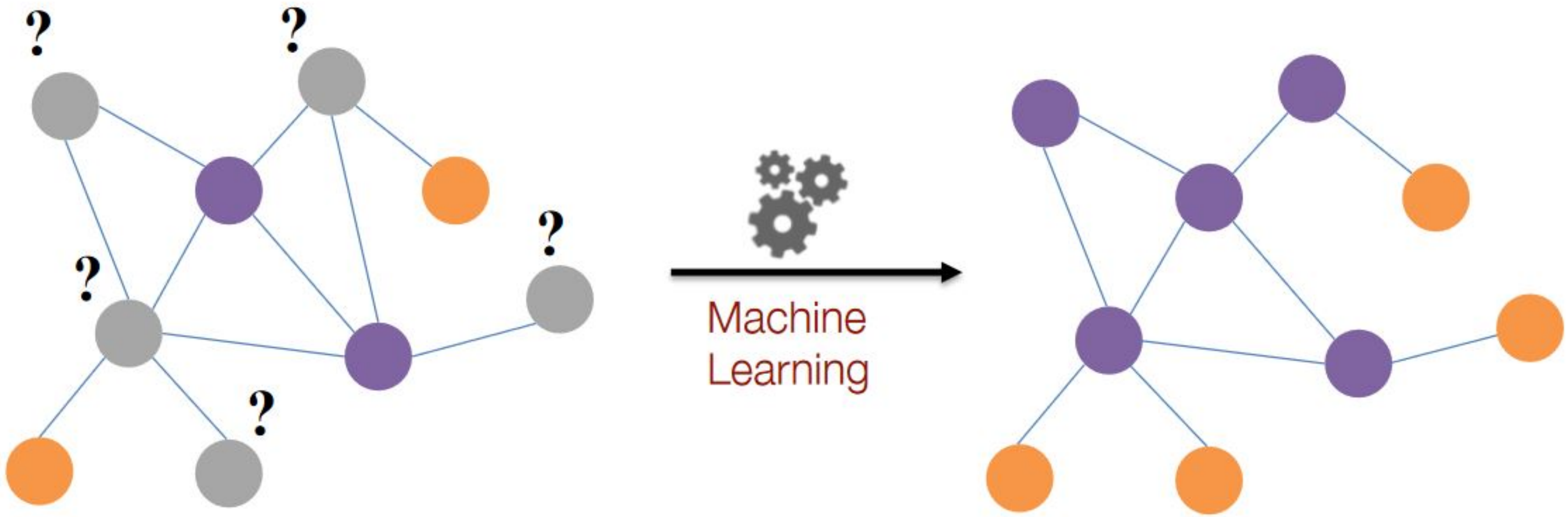
# Introduction

- **We can use the representation of nodes to perform several tasks**
  - **Classification**
  - **Recommendation**
  - **Link Prediction**

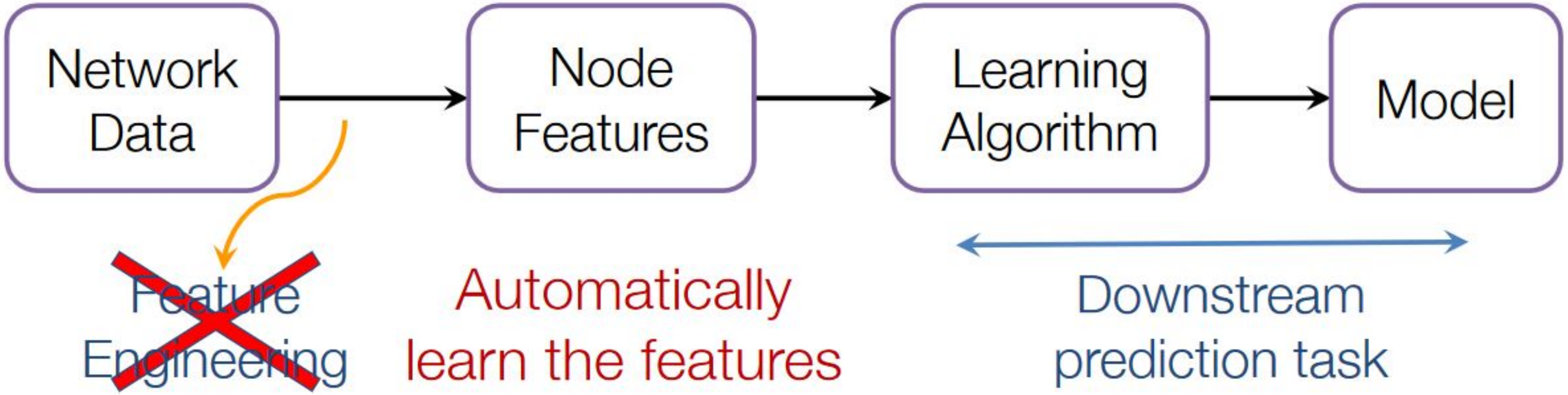


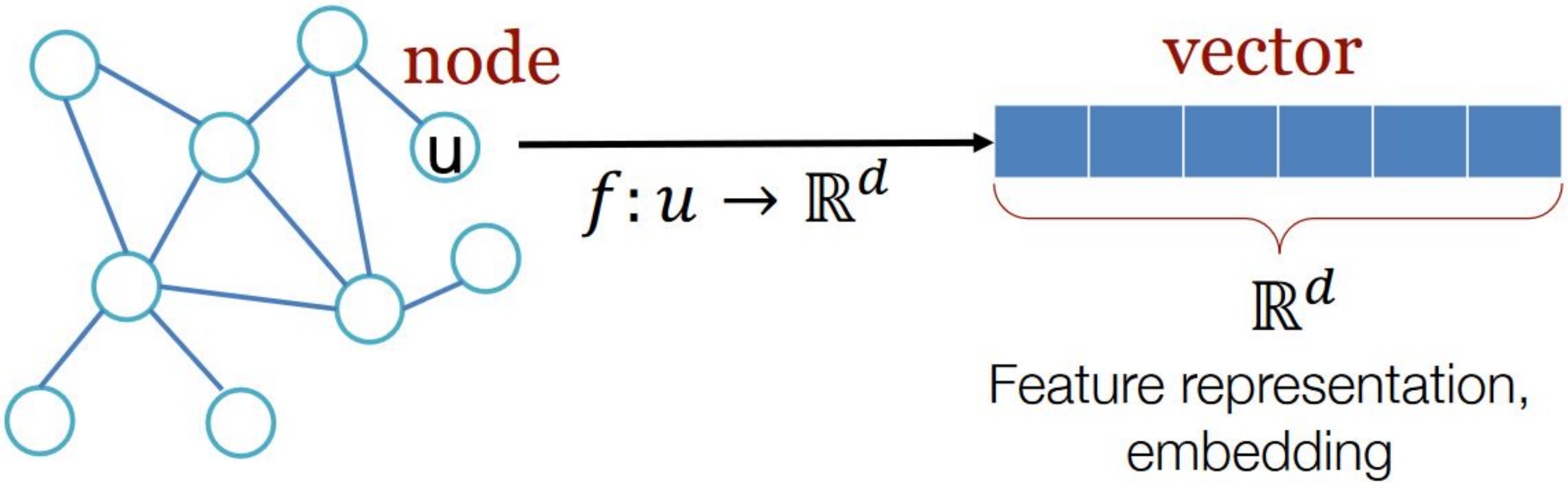
# Node Classification



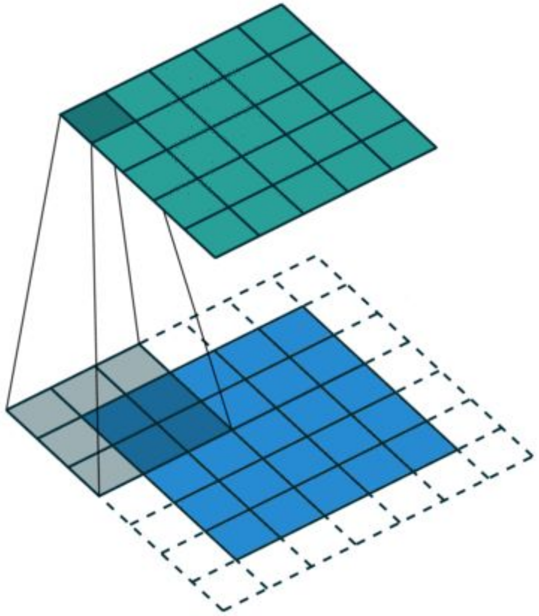
# Graph Representation



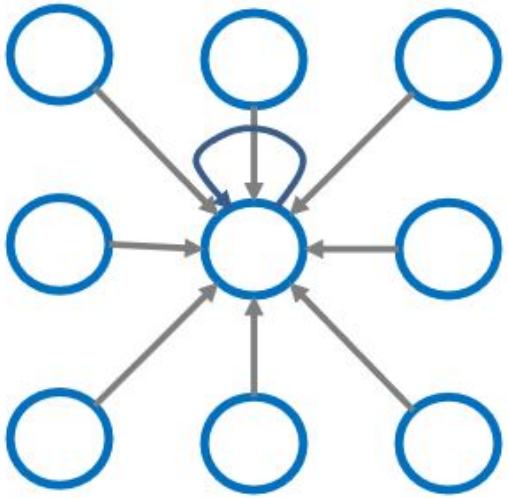
# Feature Learning in Graph



# From Images to Networks

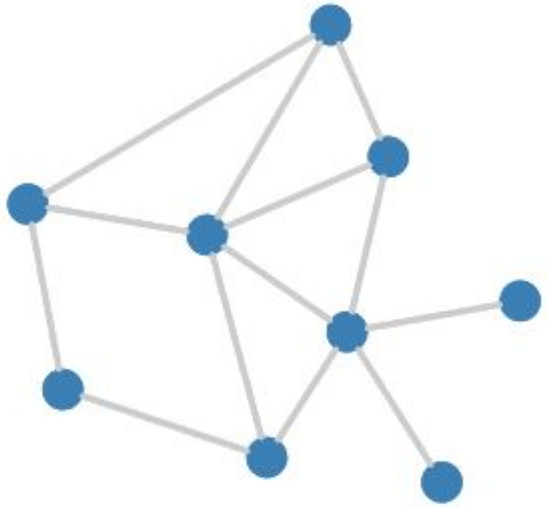


Image

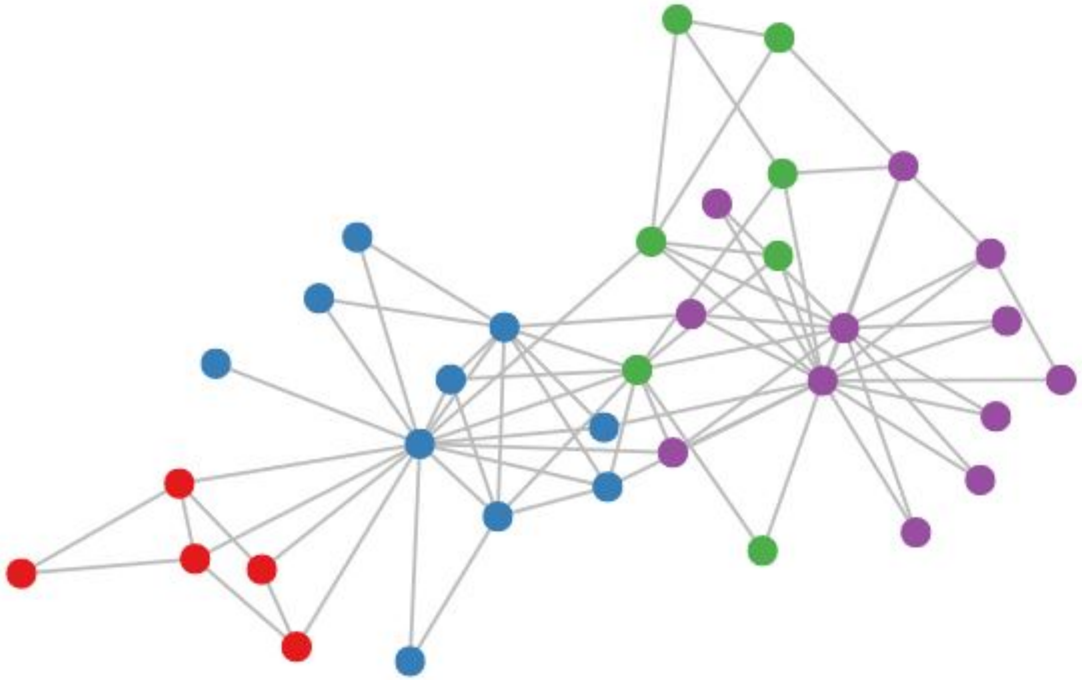


Graph

# Real-World Graphs

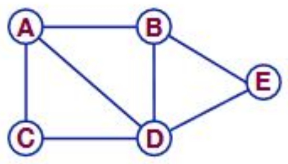


**OR**

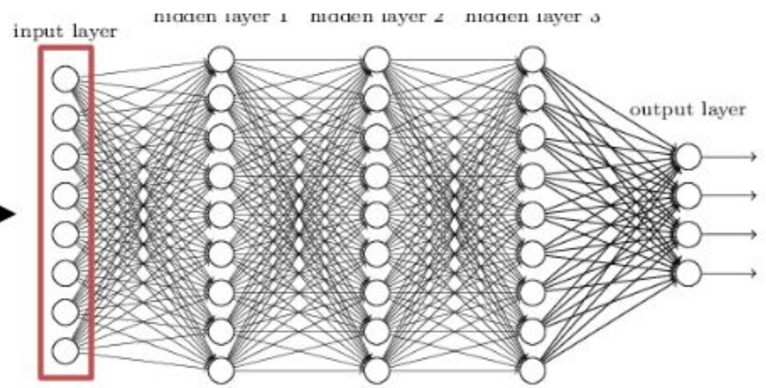


# Naïve Graph Neural Network

- Join adjacency matrix and features
- Feed them into a deep neural net




$$
\begin{array}{c}
\\ A \\ B \\ C \\ D \\ E
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & \multicolumn{2}{c}{\text{Feat}} \end{array} \\
\left(\begin{array}{ccccc|cc}
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}\right)
\end{array}
\longrightarrow
$$

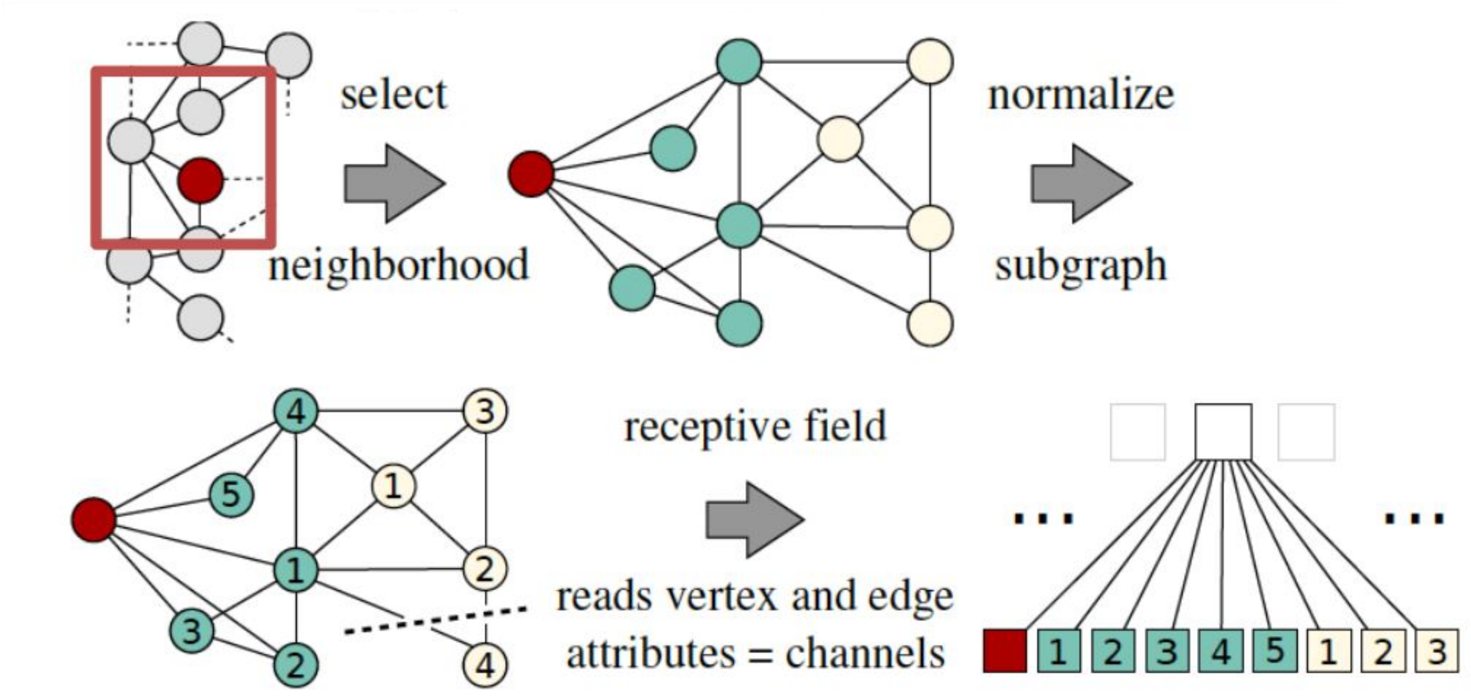



?

- Issues:
  1. #parameters
  2. Just using for fixed-size graph
  3. Variant to Node ordering

**Challenging for standard deep neural net architectures (CNNs, RNNs)**

# Graph Convolutional Network (GCN)

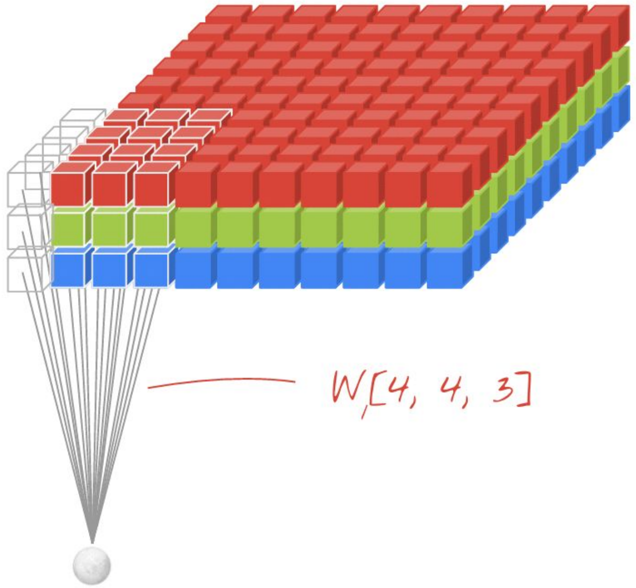


# Optimal Convolutional Neural Network for Graph

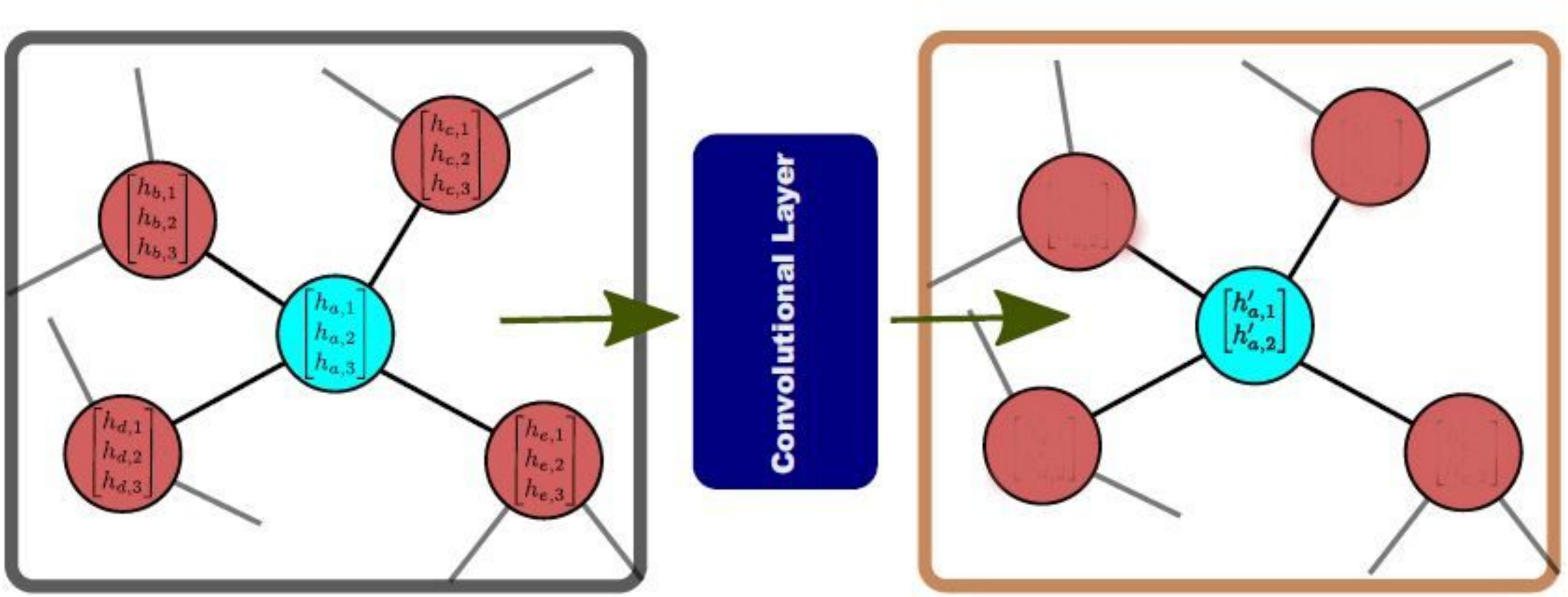
- Invariant to node ordering
- Locality
- Model parameters will be independent from graph size (Applicable to unseen data)
- Independent of graph structure

# Difference between Convolving in Image & Graph

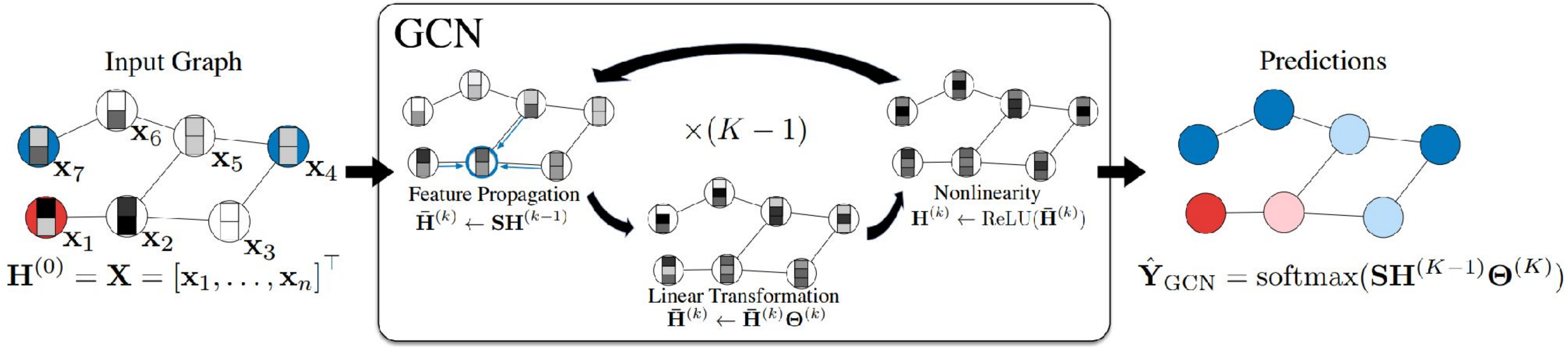
- Building Block of GCNs
- For each node in the graph, a convolutional operator consists of two main steps:
  - Aggregation of neighboring node features
  - Applying a nonlinear function to generate the output features



# Graph Convolutional Layer



# GCN Structure




Wu, Felix, et al. "Simplifying Graph Convolutional Networks."(2019)

# Towards the GAT convolution layer

1. Naïve convolution:

- Uniformly average neighboring node features and apply a non-linear function

2. Less-naïve convolution:

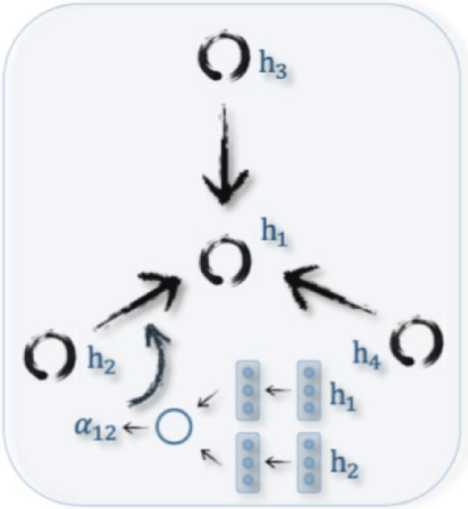
- Multiply features by a weight matrix then uniformly average features and apply a non-linear function

3. Kipf & Welling:

- Weights in average depends on degree of neighboring nodes

4. Velickovic et al. (2018) (Graph Attention Networks):

- Weights computed by a self-attention mechanism based on node features



# Graph Attention Layer -Aggregation

- **Input Features**

$$h=\{\overrightarrow{h_1}, \overrightarrow{h_2},\ldots, \overrightarrow{h_N}\}, \overrightarrow{h_i} \in \mathbb{R}^F$$

- **Output Features**

$$h'=\{\overrightarrow{h'_1}, \overrightarrow{h'_2},\ldots, \overrightarrow{h'_N}\}, \overrightarrow{h'_i} \in \mathbb{R}^{F'}$$

- **Aggregation**

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

# Graph Attention Layer (Weights)

- **Self-Attention head**

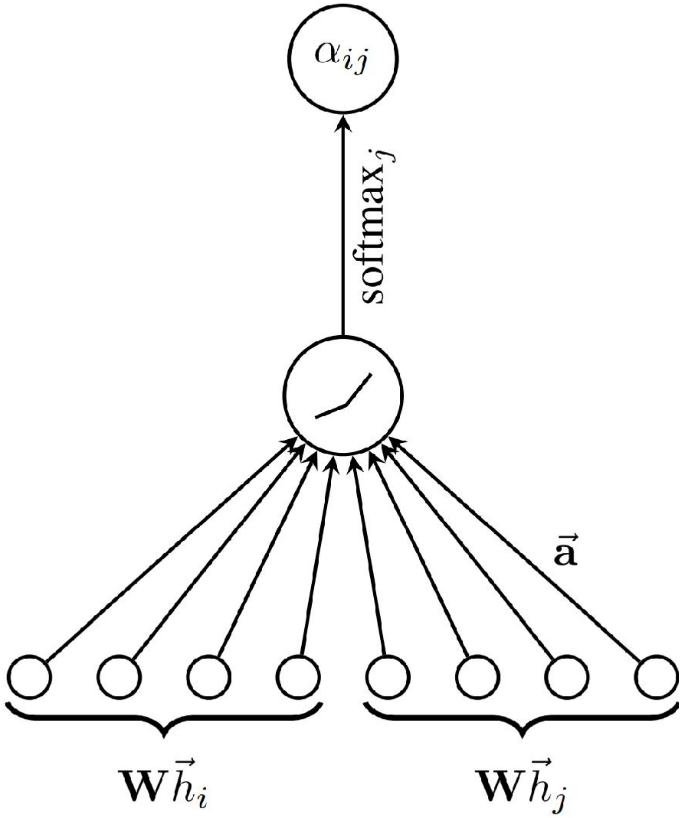
$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

- **Output of self-attention head**


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

- **Weights can then be used to compute output**

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



# Graph Attention Layer (Multiple Heads)

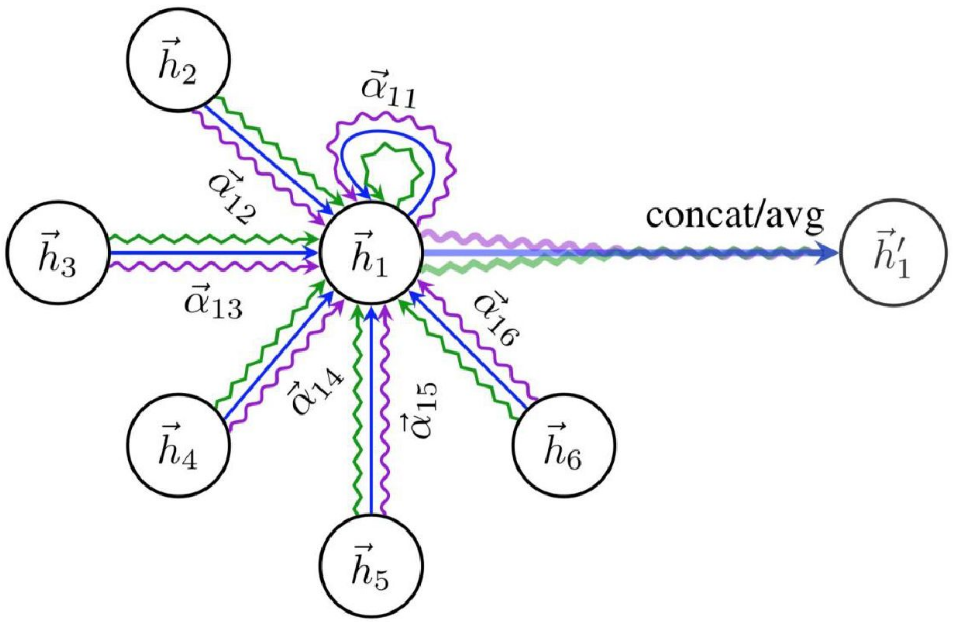
- **Aggregation with single head**

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

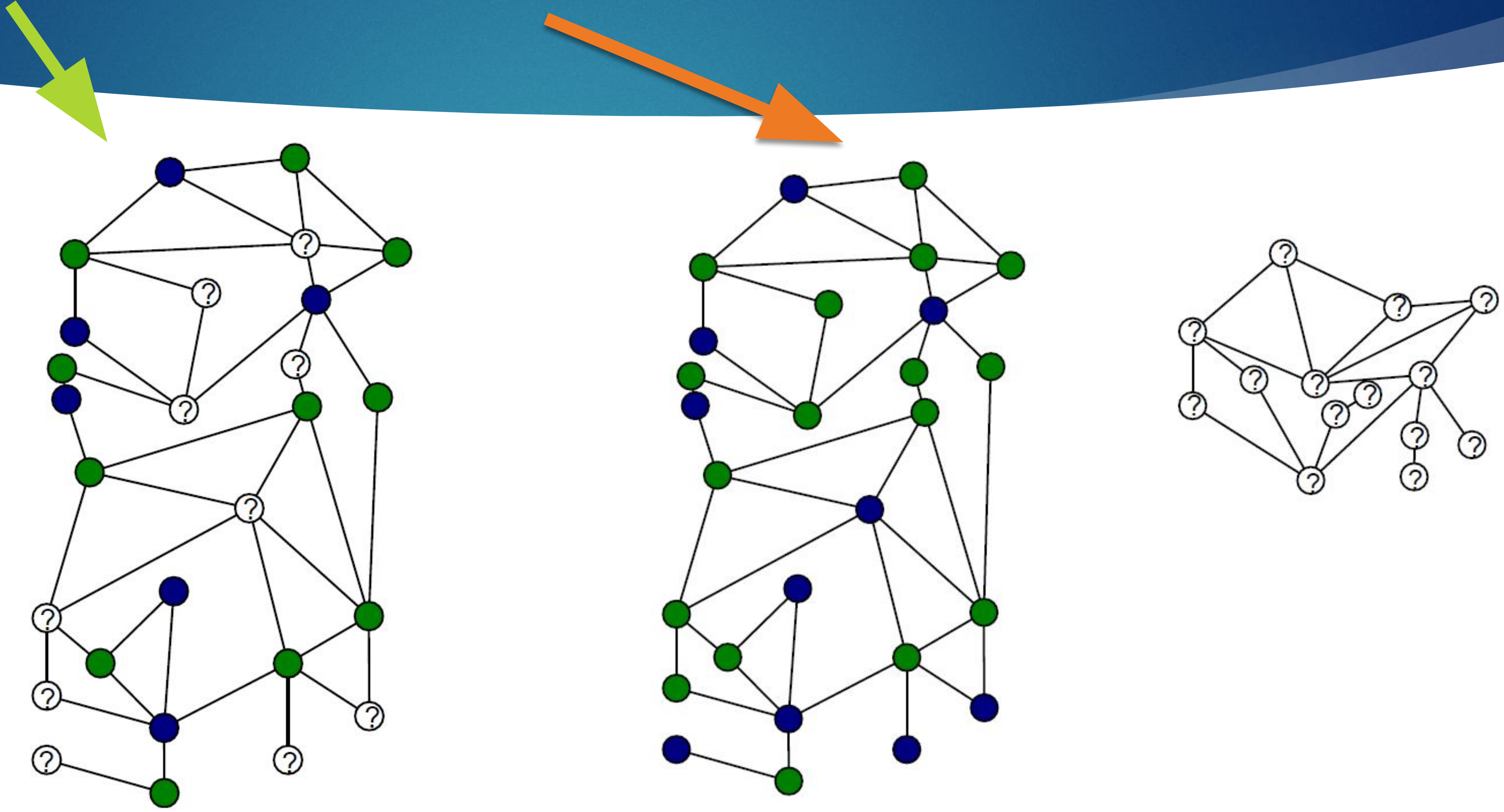
- **Aggregation with multiple heads**

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

- **Final Layer uses average instead of concatenation**



# Transductive & Inductive



# Experiments/ Transductive

- Citation Networks: Cora, Citeseer and Pubmed
- Each node in the graph belongs to a one of C classes.
  - Cora dataset

GRAPH ATTENTION NETWORKS

**Petar Veličković***
Department of Computer Science and Technology
University of Cambridge
petar.velickovic@cst.cam.ac.uk

**Guillem Cucurull***
Centre de Visió per Computador, UAB
gcucurull@gmail.com

**Arantxa Casanova***
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

**Adriana Romero**
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

**Pietro Liò**
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

**Yoshua Bengio**
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

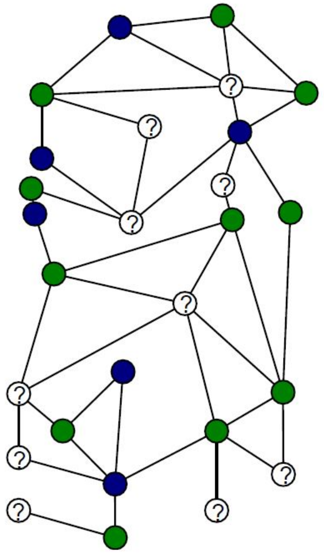
ABSTRACT

We present graph attention networks (GATs), novel neural network architectures that operate on graph-structured data, leveraging masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations. By stacking layers in which nodes are able to attend over their neighborhoods' features, we enable (implicitly) specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure upfront. In this way, we address several key challenges of spectral-based graph neural networks simultaneously, and make our model readily applicable to inductive as well as transductive problems. Our GAT models have achieved or matched state-of-the-art results across four established transductive and inductive graph benchmarks: the *Cora*, *Citeseer* and *Pubmed* citation network datasets, as well as a *protein-protein interaction* dataset (wherein test graphs remain unseen during training).

500

256

- 20 nodes per class is used for training, 500 nodes are used for validation and 1000 for testing.
- Architecture:
  - 2 Layers of convolutions.
  - First layer has 8 attention heads, each computing 8 features.
  - Second layer has1attention head computing C features, followed by a Soft Max layer.
  - Dropout is applied to layer inputs as well as to attention coefficients.



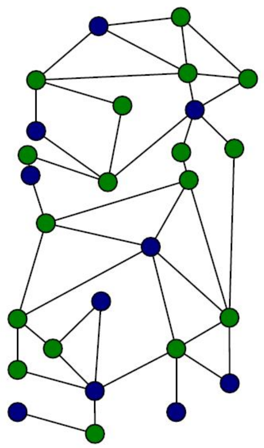
# Experiments/ Inductive

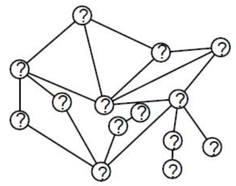
- Protein-Protein interaction dataset
- Training: 20graphs,Validation: 2graphs, Testing: 2graphs
- Testing graphs are completely unobserved during training
- Eachnodehas50features.
- 121classes (each node can have multiple labels)
- Architecture:

  Three-Layer GAT model.

  Layers#1&#2:four attention heads computing 256 features each.

  Layer#3: six attention heads computing 121 features each, that are averaged and followed by a logistic sigmoid activation.



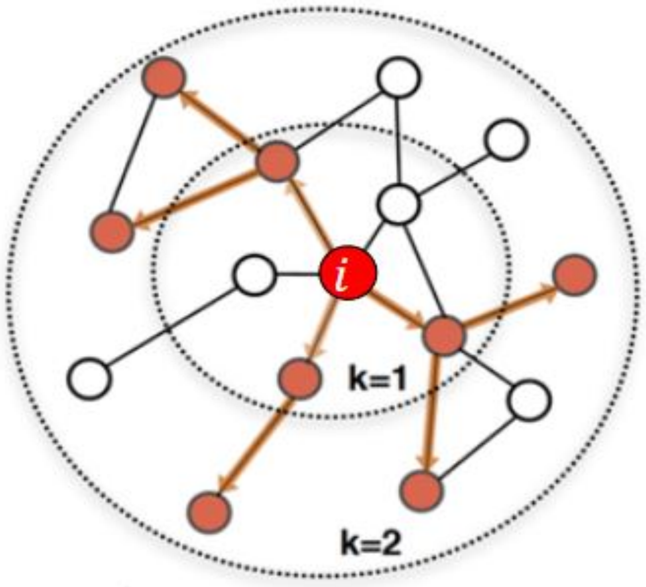


# Graph SAGE

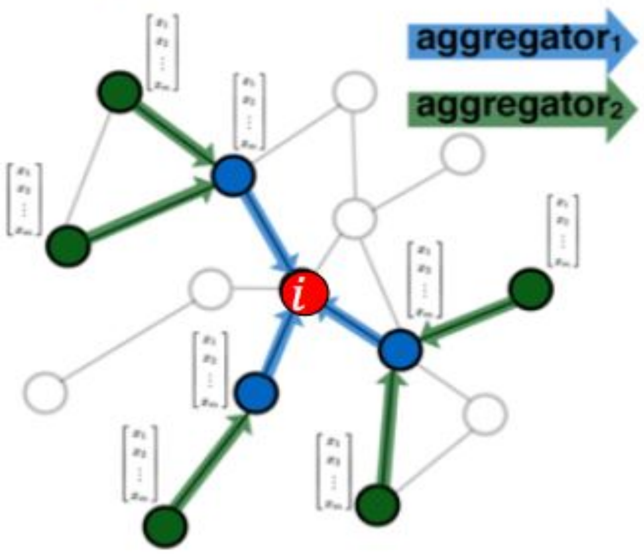
- Adapt the GCN idea to inductive node embedding
- Generalize beyond simple convolutions
- Demonstrate that this generalization
  - Leads to significant performance gains
  - Allows the model to learn about local structures

# Idea SAGE

- Idea: Node's neighborhood defines a computation graph
- Learn how to propagate information across the graph to compute node features



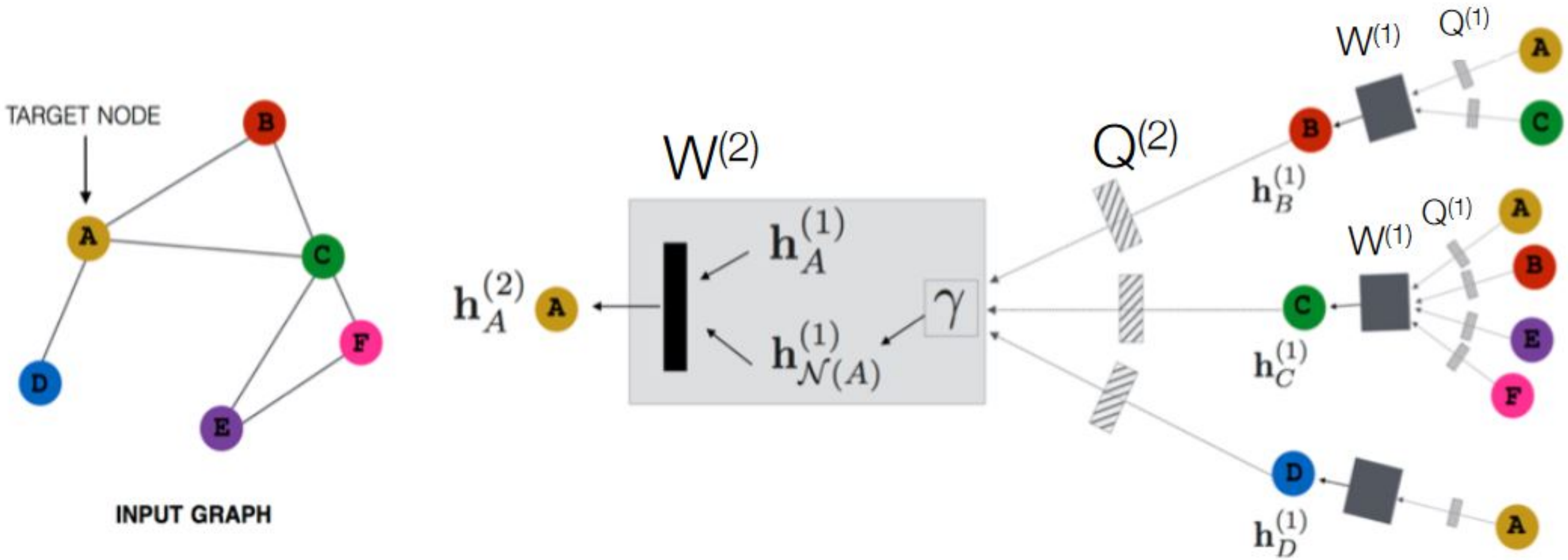

Determine node computation graph



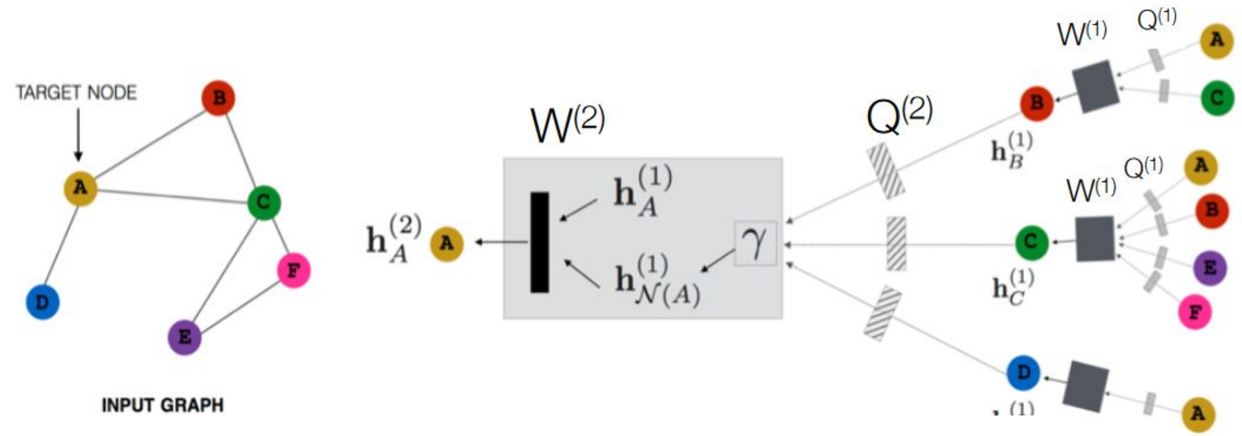

Propagate and transform information

# Graph SAGE

- Each node defines its own computational graph
  - Each edge in this graph is a transformation/aggregation function



# Graph SAGE




**Update for node *A*:**

$$\underbrace{h_A^{(k+1)}}_{\substack{k+1^{th}\text{ level} \\ \text{features of node } A}} = ReLU\left(\underbrace{W^{(k)}h_A^{(k)}}_{\substack{\text{Transform } A\text{'s own} \\ \text{features from level } k}}, \sum_{n\in\mathcal{N}(A)} \underbrace{\left(ReLU(Q^{(k)}h_n^{(k)})\right)}_{\substack{\text{Transform and aggregate} \\ \text{features of neighbors } n}}\right)$$

- $h_A^{(0)}$ = attributes of node $A$
- $\Sigma(\cdot)$: Aggregator function (e.g., avg., LSTM, max-pooling)

# SAGE Algorithm

initialize representations as features

K = "search depth"

aggregate information from neighbors

$$
\begin{aligned}
&\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V};\\
&\textbf{for } k = 1...K \textbf{ do}\\
&\quad \textbf{for } v \in \mathcal{V} \textbf{ do}\\
&\quad\quad \mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});\\
&\quad\quad \mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)\\
&\quad \textbf{end}\\
&\quad \mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}\\
&\textbf{end}\\
&\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}
\end{aligned}
$$

concatenate neighborhood info with current representation and propagate

$$J = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - \frac{1}{|Q|} \cdot \sum_{q=1}^{Q} \mathbb{E}_{v_n \sim P_n(v)} \log\left(-\sigma(\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

classification (cross-entropy) loss

# Weisfeiler-Lehman graph

- The classic Weisfeiler-Lehman graph isomorphism test is a special case of Graph SAGE
- We replace the hash function with trainable neural nets

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

**for** $k = 1 \ldots K$ **do**

  **for** $v \in \mathcal{V}$ **do**    HASH

    $\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \cancel{\text{AGGREGATE}}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \cancel{\sigma}\left(\cancel{\mathbf{W}}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}^k_{\mathcal{N}(v)})\right)$

  **end**

  $\cancel{\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}}$

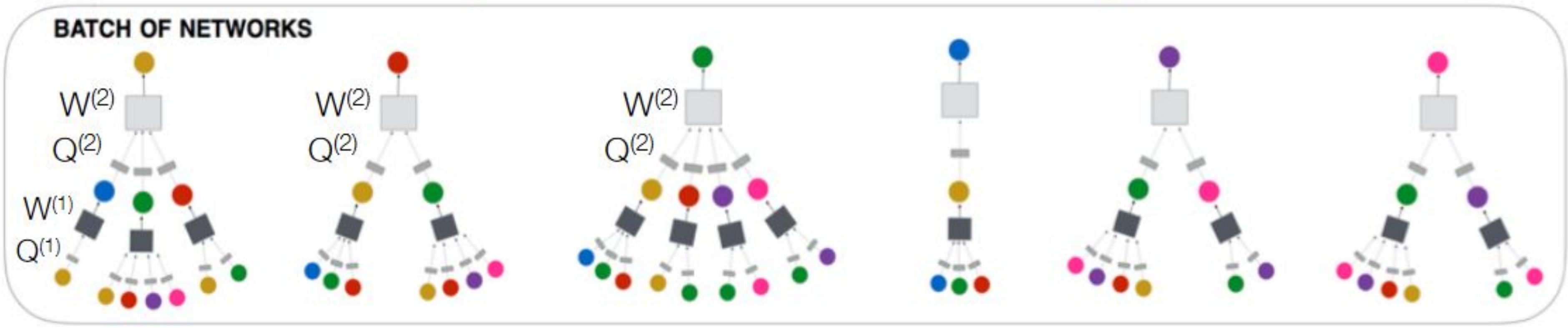
**end**

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Shervashidze, Nino, et al. "Weisfeiler-Lehman graph kernels." Journal of Machine Learning Research (2011)

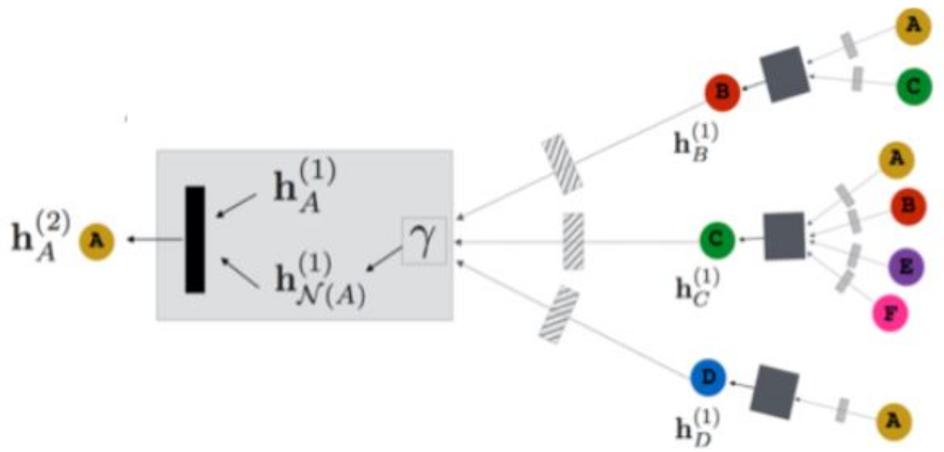
# Parameter Sharing

- Two types of parameters
  - Aggregate function can have parameters.
  - Matrix W(k)
- Adapt to inductive setting (e.g., unsupervised loss, neighborhood sampling, mini batch optimization)
- Generalized notion of "aggregating neighborhood"



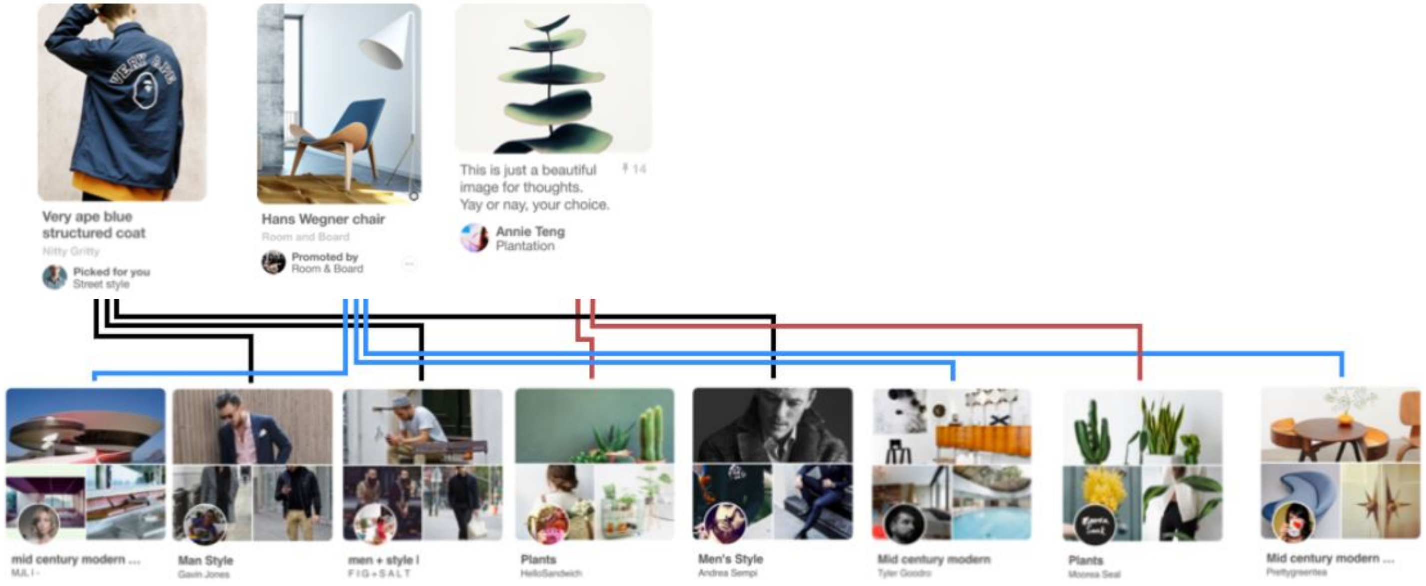
# Benefits of the algorithm

- Can use different aggregators
- Can use different loss functions
- Model has a constant number of parameters
- Fast scalable inference
- Can be applied to any node in any network



# Application: Pinterest

- **Human curated collection of pins**
- Pin: a visual bookmark someone has saved from the internet to a board they've created.
- Board: A greater collection of ideas (pins having s.th. in common)

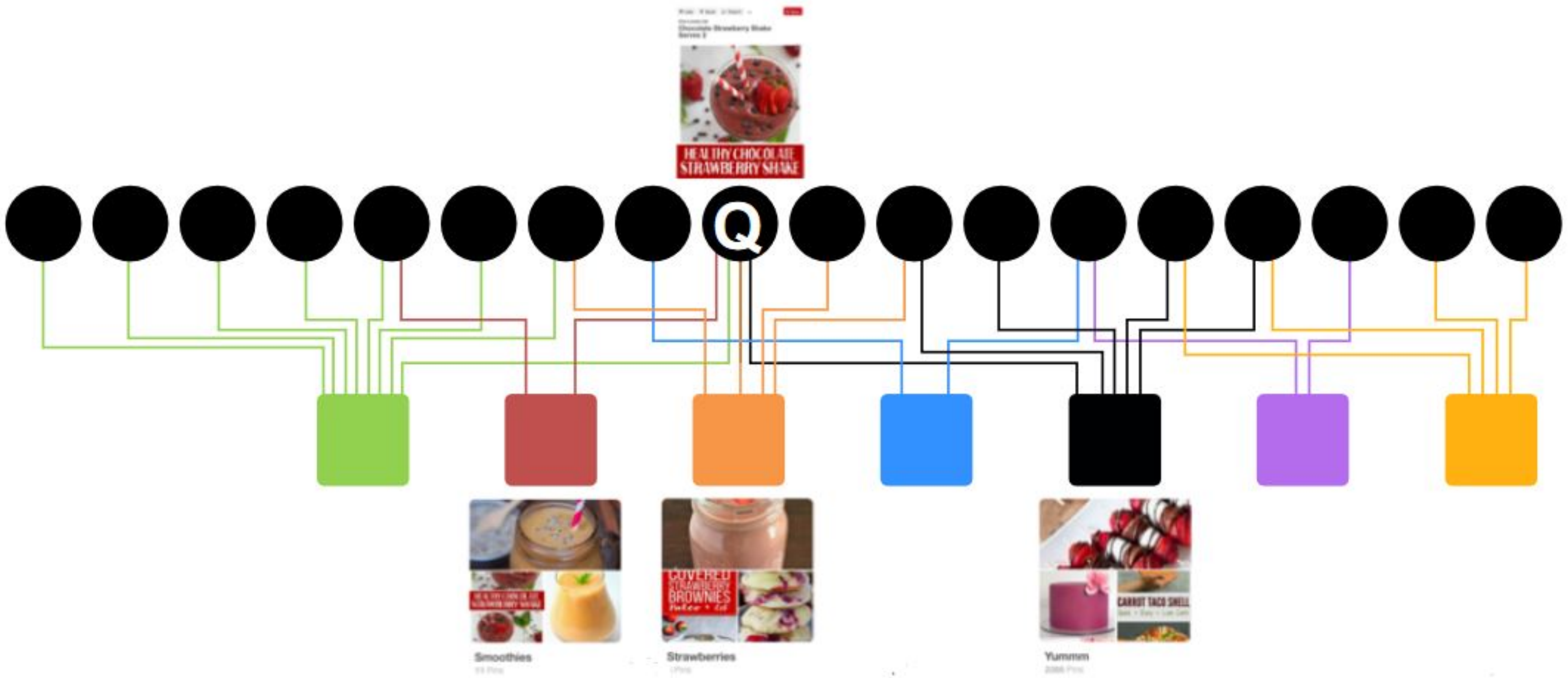


# Pin SAGE

- Semi-Supervised node embedding for graph-based recommendations
- Graph:2B pins, 1B boards, 20B edges

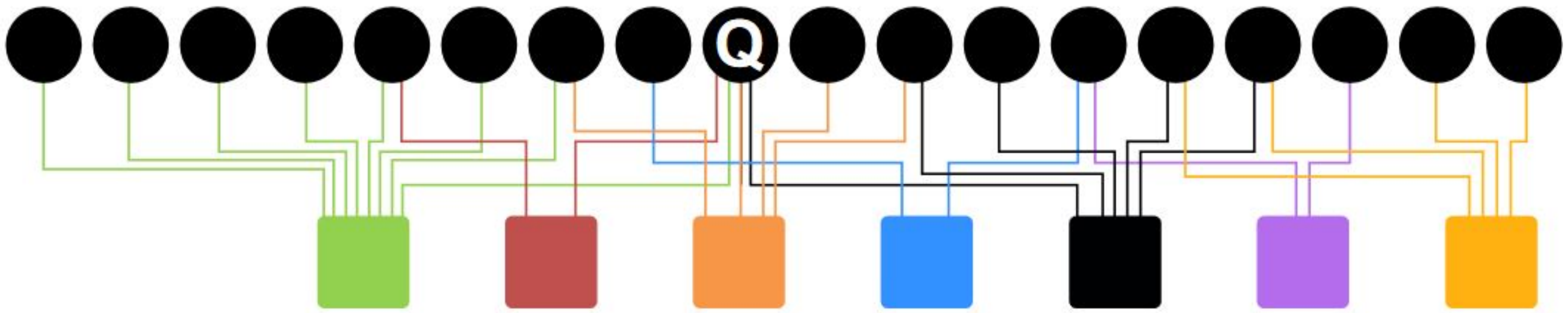
pins

boards



# Pinterest Graph

- Pinterest Graph
- Graph is dynamic : need to apply to new nodes without model retraining
- Rich node features : content, image



# Task: item-item recommendation

## Related Pin recommendations

- Given user is looking at pin **Q**, what pin **X** are they going to save next.


Query

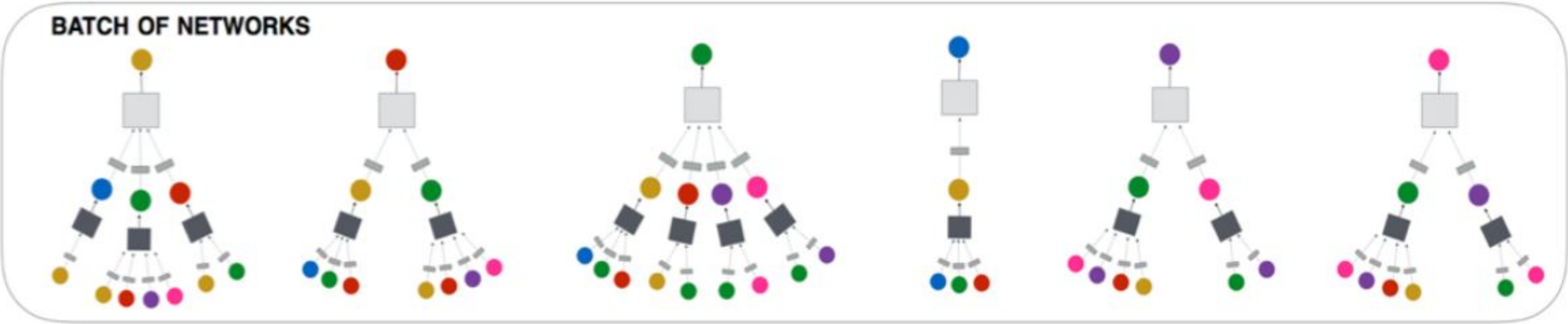

Positive

Rnd. negative


Hard negative

# GraphSAGE Training

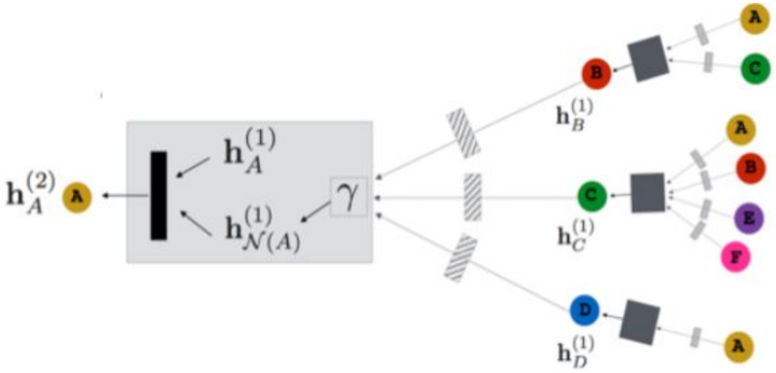
- Leverage inductive capability, and train on individual sub graphs
  - 300 million nodes, 1 billion edges, 1.2 billion pin pairs **(Q, X)**

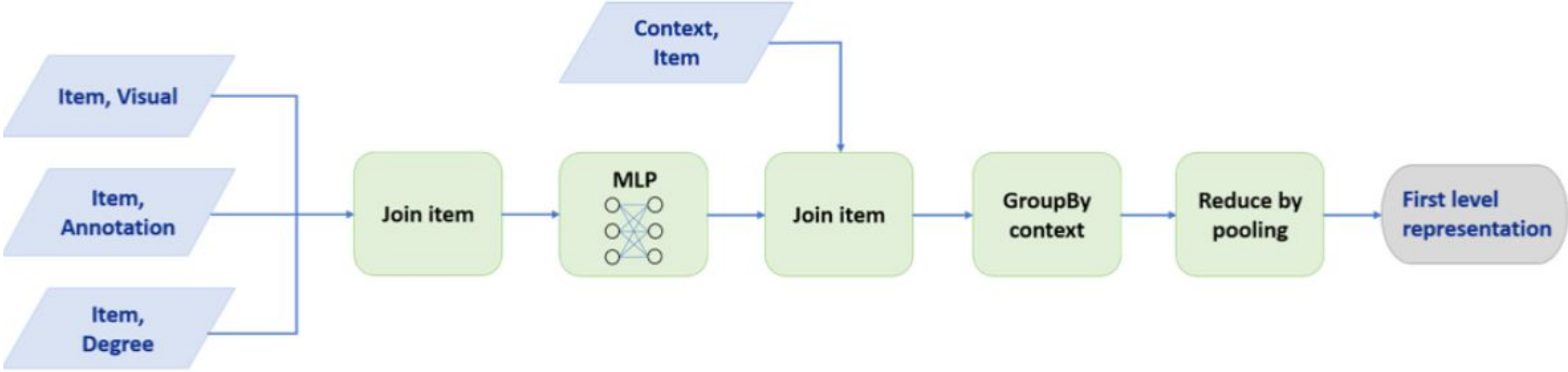



- Large batch size: 2048 per mini batch

# Graph SAGE: Inference

- Use Map Reduce for model inference



# Related Pin recommendations

- Given user is looking at pin Q, predict what pin X are they going to save next
- Baselines for comparison
  - Visual:VGG-16 visual features
  - Annotation:Word2Vec model
  - Combined: combine visual and annotation
  - RW: Random-walk based algorithm Graph SAGE
- Setup: Embed 2B pins, perform nearest neighbor to generate recommendations

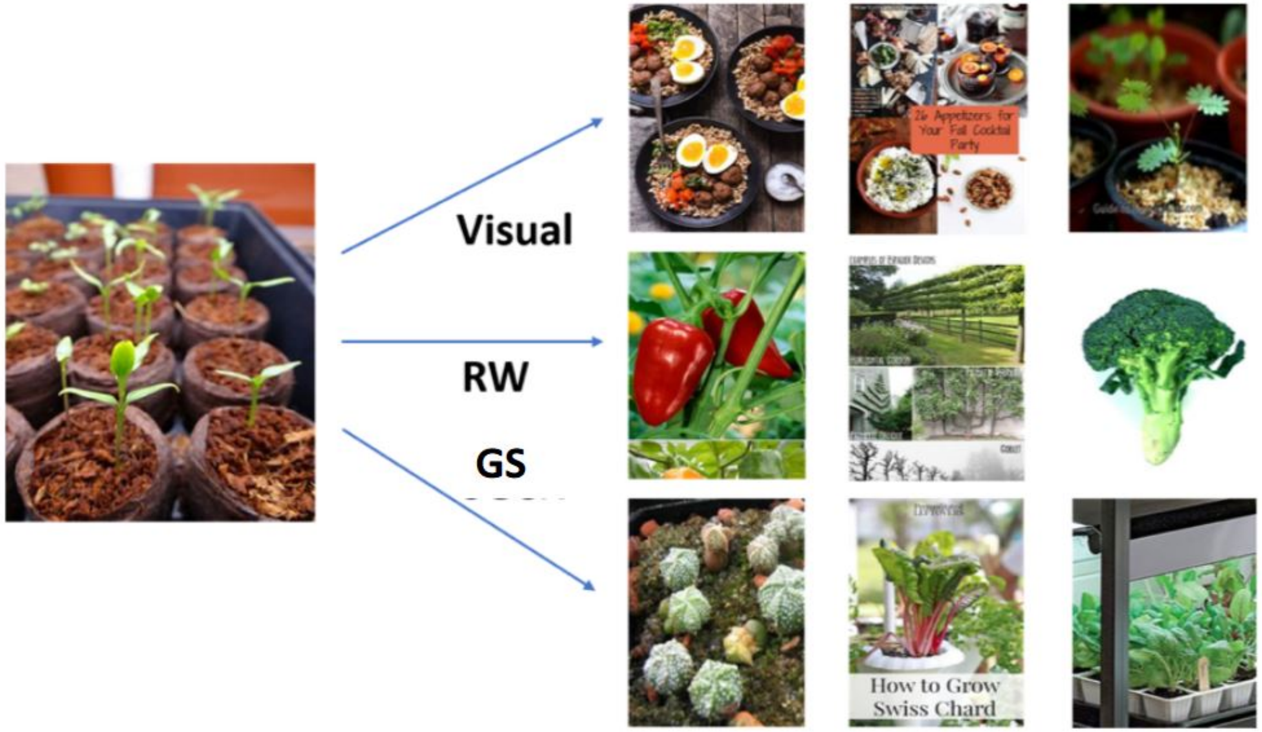
# Ranking

Task :Given Q, rank X as high as possible among 2B pins

- Hit-rate: Pct. P was among top-k
- MRR: Mean reciprocal rank

| Method | Hit-rate | MRR |
|---|---|---|
| Visual | 17% | 0.23 |
| Annotation | 14% | 0.19 |
| Combined | 27% | 0.37 |
| GraphSAGE | 46% | 0.56 |

# Example Recommendations



# Summary

- Graph Convolution Networks

  Generalize beyond simple convolutions
- Fuses node features & graph info
- State-of-the-art accuracy for node classification and link prediction.
- Model size independent of graph size;
- can scale to billions of nodes
- Largest embedding to date (3B nodes, 20B edges)
- Leads to significant performance gains

# References


- Graph Attention Networks,  PETER VELIČKOVIĆ& YOSHUABENGIO, et al., 2018.
- “Graph Attention Networks”, Presenter: Karim Khayrat; Facilitators: Matthew Chang-Kit  & Parham Hamouni
- Graph Convolutional Neural Networks for Web-Scale Recommender Systems, Ying, et al., 2018.
- “Graph Representation Learning with Graph Convolutional Networks”, Presenter: Jure Leskovec, 2018
- [IPAM2019] Thomas Kipf "Unsupervised Learning with Graph Neural Networks"