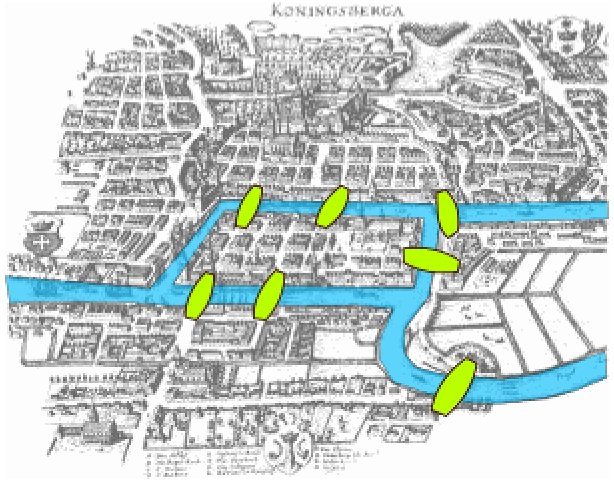
Graph

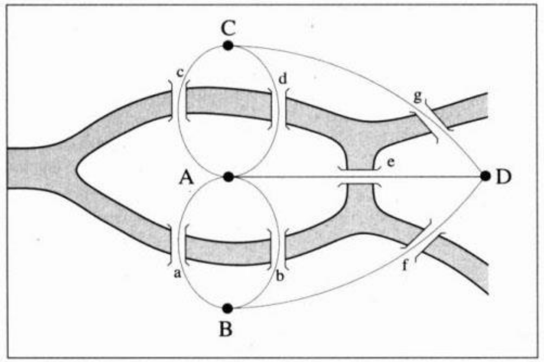
Prof. Seungchul Lee
Industrial AI Lab.

Graph (or Network)

- Abstract relations, topology, or connectivity

- Graphs $G = (V, E)$
  - V: a set of vertices (nodes)
  - E: a set of edges (links, relations)
  - weight (edge property)
    - distance in a road network
    - strength of connection in a personal network





Graph (or Network)

- Graphs can be *directed* or *undirected*

- Graphs model any situation where you have objects and pairwise relationships (symmetric or asymmetric) between the objects

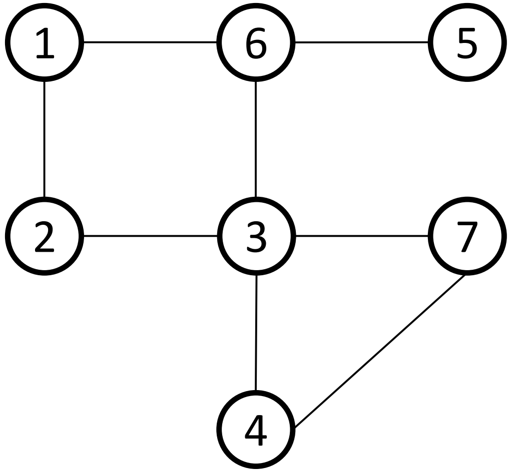
Vertex	Edge	
People	like each other	undirected
People	is the boss of	directed
Tasks	cannot be processed at the same time	undirected
Computers	have a direct network connection	undirected
Airports	planes flies between them	directed
City	can travel between them	directed

Graph Representation

- Question
  - How to represent a graph for a computer to understand it

- Any guess?
  - Adjacent matrix

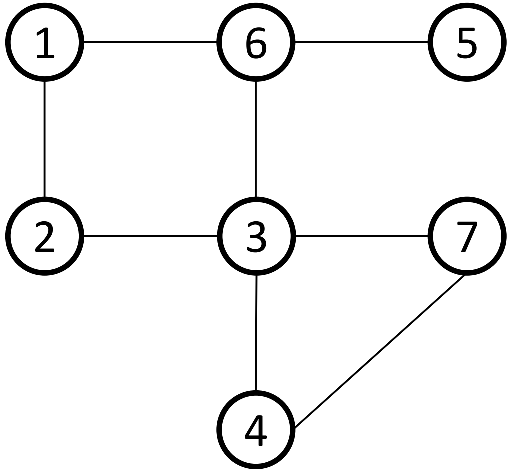
- Graph can be represented as adjacency matrix A
  - Adjacency matrix A indicates adjacent nodes for each node



POSTECH

Adjacent Matrix

- Undirected graph $G = (V, E)$
- Let computers to understand a structure of graph



$$V = \{1, 2, \cdots, 7\}$$
$$E = \{\{1,2\}, \{1,6\}, \{2,3\}, \{3,4\}, \{3,6\}, \{3,7\}, \{4,7\}, \{5,6\}\}$$

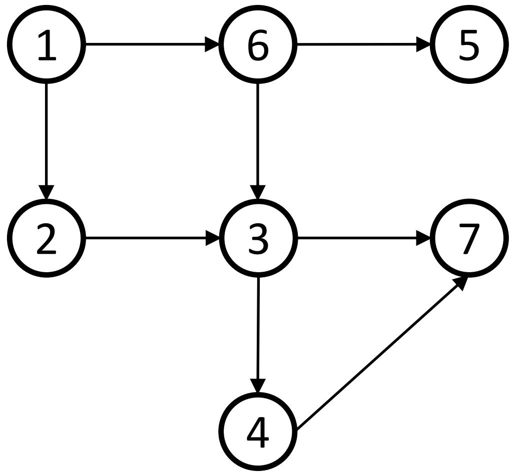
$$\text{Adjacency list} = \begin{cases} \text{adj}(1) = \{2,6\} \\ \text{adj}(2) = \{1,3\} \\ \text{adj}(3) = \{2,4,6,7\} \\ \text{adj}(4) = \{3,7\} \\ \text{adj}(5) = \{6\} \\ \text{adj}(6) = \{1,3,5\} \\ \text{adj}(7) = \{3,4\} \end{cases}$$

$$\text{Adjacency matrix (symmetric) } A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Adjacent Matrix

- Directed graph $G = (V, E)$

- Let computers to understand a structure of graph



$$V = \{1, 2, \cdots, 7\}$$
$$E = \{\{1,2\}, \{1,6\}, \{2,3\}, \{3,4\}, \{3,7\}, \{4,7\}, \{6,3\}, \{6,5\}\}$$

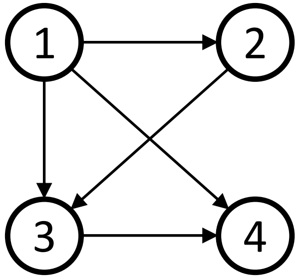
$$\text{Adjacency list} = \begin{cases} \text{adj}(1) & = \{2, 6\} \\ \text{adj}(2) & = \{3\} \\ \text{adj}(3) & = \{4, 7\} \\ \text{adj}(4) & = \{7\} \\ \text{adj}(5) & = \phi \\ \text{adj}(6) & = \{3, 5\} \\ \text{adj}(7) & = \phi \end{cases}$$

$$\text{Adjacency matrix (symmetric) } A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

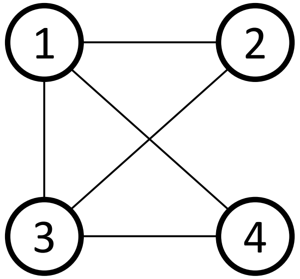
sparse

Quiz 1

- Directed graph

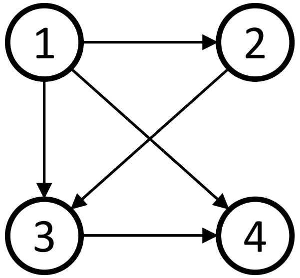


- Undirected graph



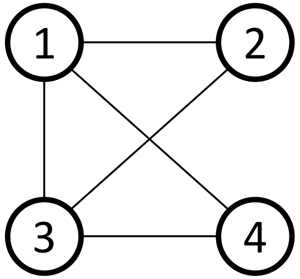
Quiz 1

- Directed graph



	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

- Undirected graph



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Quiz 2

- Directed graph G = (V,E)
  - V = {0,1,2,3,4,5}
  - Adjacency list

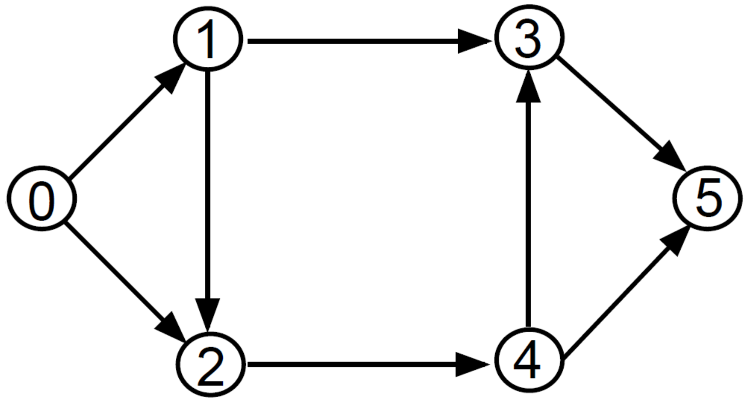
$$
\begin{aligned}
Adj(0) &= \{1,2\} \\
Adj(1) &= \{2,3\} \\
Adj(2) &= \{4\} \\
Adj(3) &= \{5\} \\
Adj(4) &= \{3,5\} \\
Adj(5) &= \emptyset
\end{aligned}
$$

- Q: draw the corresponding directed graph

Quiz 2

- Directed graph G = (V,E)
  - V = {0,1,2,3,4,5}
  - Adjacency list

$$\begin{aligned} Adj(0) &= \{1,2\} \\ Adj(1) &= \{2,3\} \\ Adj(2) &= \{4\} \\ Adj(3) &= \{5\} \\ Adj(4) &= \{3,5\} \\ Adj(5) &= \emptyset \end{aligned}$$



- Q: draw the corresponding directed graph

Degree

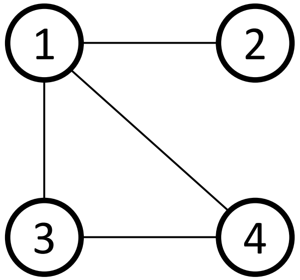
- Degree of undirected graph
  - the degree of vertex in a graph is the number of edges connected to it
  - denote the degree of vertex i by d_i
  - for an undirected graph of n vertices

$$d_i = \sum_{j=1}^{n} A_{ij}$$

- Degree matrix D of adjacent matrix A

$$D = \mathrm{diag}\{d_1, d_2, \cdots\}$$

- Example

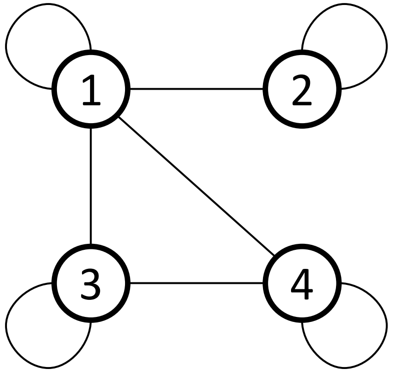


$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \Rightarrow \quad D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Self-Connecting Edges

- Adding I is to add self-connecting edges

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \Rightarrow \quad A + I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \Rightarrow \quad \tilde{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$




```
A = np.array([[0,1,1,1],
              [1,0,0,0],
              [1,0,0,1],
              [1,0,1,0]])

A_self = A + np.eye(4)

print(A_self)
```

```
[[1. 1. 1. 1.]
 [1. 1. 0. 0.]
 [1. 0. 1. 1.]
 [1. 0. 1. 1.]]
```

```
D = np.array(A_self.sum(1)).flatten()
D = np.diag(D)

print(D)
```

```
[[4. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 3.]]
```

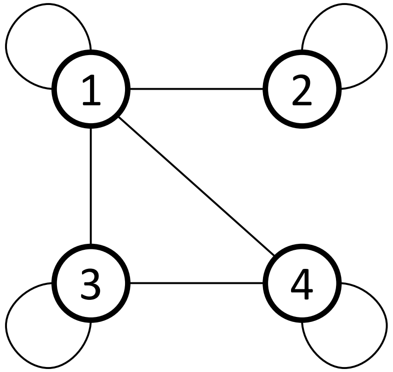
Neighborhood Normalization

- Some nodes have many edges, but some don't
  - Adding I is to add self-connecting edges
  - Considering neighboring nodes in the normalized weights
  - To prevent numerical instabilities and vanishing/exploding gradients in order for the model to converge

- (First attempt) Normalized $\tilde{A}$
  - It is not symmetric.

$$\tilde{A} = \tilde{D}^{-1}(A+I)$$

$$A + I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \tilde{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$




```
A_norm = np.linalg.inv(D).dot(A_self)
```

```
[[0.25       0.25       0.25       0.25      ]
 [0.5        0.5        0.         0.        ]
 [0.33333333 0.         0.33333333 0.33333333]
 [0.33333333 0.         0.33333333 0.33333333]]
```

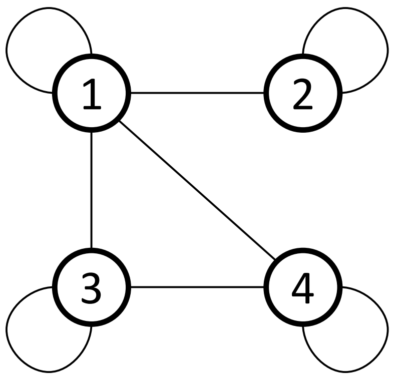
Neighborhood Normalization

- Some nodes have many edges, but some don't
  - Adding I is to add self-connecting edges
  - Considering neighboring nodes in the normalized weights
  - To prevent numerical instabilities and vanishing/exploding gradients in order for the model to converge

- Normalized $\tilde{A}$
  - Now it is symmetric.
  - (Skip the details)

$$\tilde{A} = \tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}$$

$$A + I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \tilde{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$




```
from scipy.linalg import fractional_matrix_power

D_half_norm = fractional_matrix_power(D, -0.5)
```

```
[[0.5        0.         0.         0.        ]
 [0.         0.70710678 0.         0.        ]
 [0.         0.         0.57735027 0.        ]
 [0.         0.         0.         0.57735027]]
```

```
A_self = np.asmatrix(A_self)
D_half_norm = np.asmatrix(D_half_norm)

A_half_norm = D_half_norm*A_self*D_half_norm
```

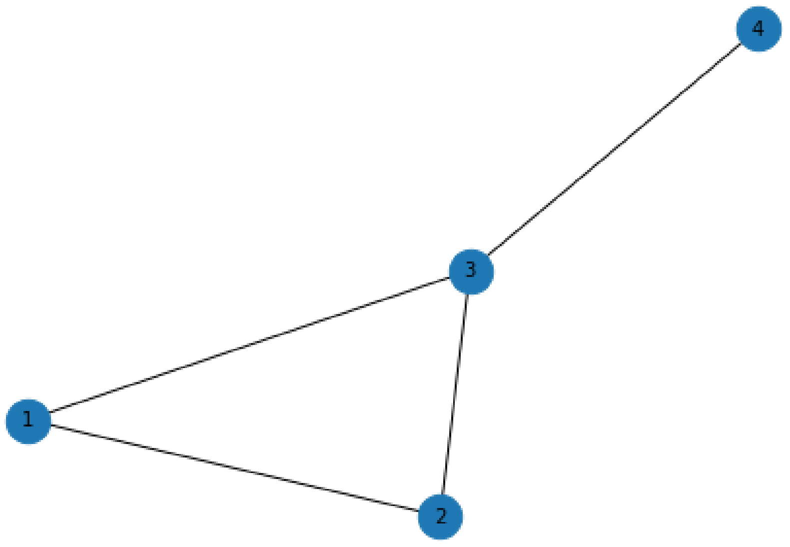
```
[[0.25       0.35355339 0.28867513 0.28867513]
 [0.35355339 0.5        0.         0.        ]
 [0.28867513 0.         0.33333333 0.33333333]
 [0.28867513 0.         0.33333333 0.33333333]]
```

NetworkX

- https://networkx.org/
- Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks

```
import networkx as nx
```

```
G = nx.Graph()

G.add_nodes_from([1, 2, 3, 4])
G.add_edges_from([(1,2), (1,3), (2,3), (3,4)])

# plot a graph
pos = nx.spring_layout(G)

nx.draw(G, pos, node_size = 500)
nx.draw_networkx_labels(G, pos, font_size = 10)
plt.show()
```

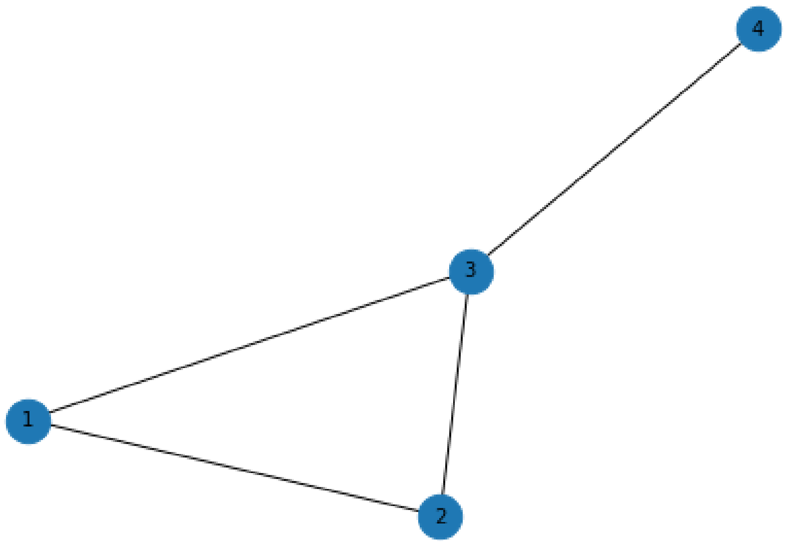


NetworkX

- Adjacency matrix

```
A = nx.adjacency_matrix(G)

print(A)
print(A.todense())
```

```
  (0, 1)	1
  (0, 2)	1
  (1, 0)	1
  (1, 2)	1
  (2, 0)	1
  (2, 1)	1
  (2, 3)	1
  (3, 2)	1
[[0 1 1 0]
 [1 0 1 0]
 [1 1 0 1]
 [0 0 1 0]]
```



Graph Neural Networks

Prof. Seungchul Lee
Industrial AI Lab.

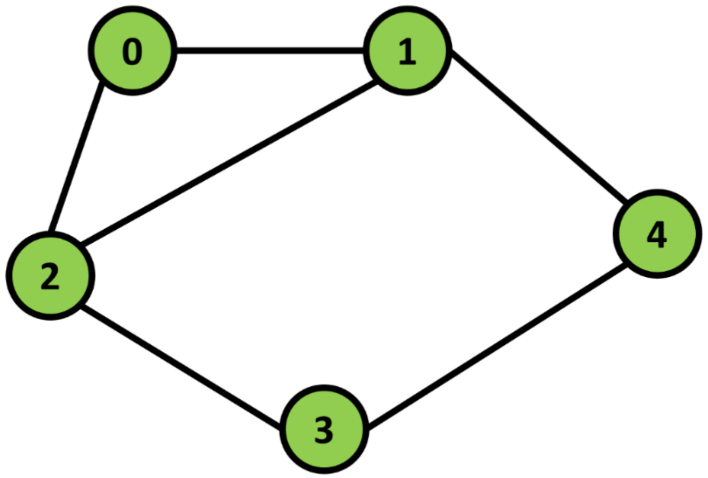
Graph Data

- Characteristic of graph data
  - A graph contains relationships between data

Person 1
- Age
- Height
- Weight
- College
- etc.

Person 2
- Age
- Height
- Weight
- College
- etc.

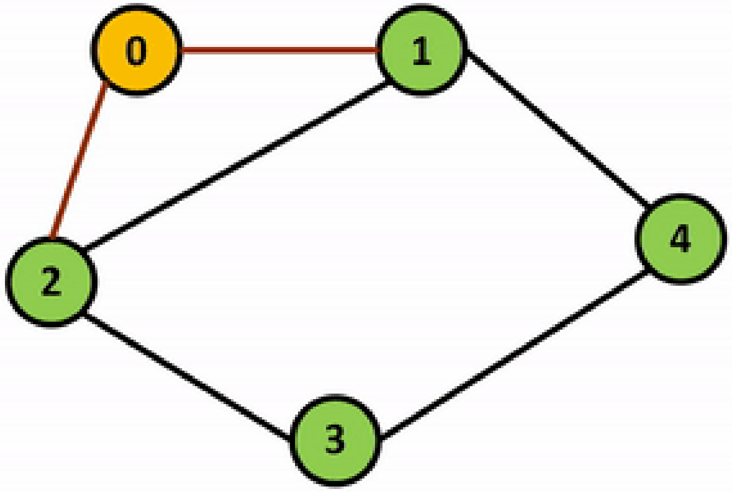
Independent



Dependent

Dependency of Graph Data

- Adjacency Matrix
  - Graph data represent this dependency by adjacency matrix
    - $A_{ij} = 1$ if there is a link from node i to node j
    - $A_{ij} = 0$ otherwise

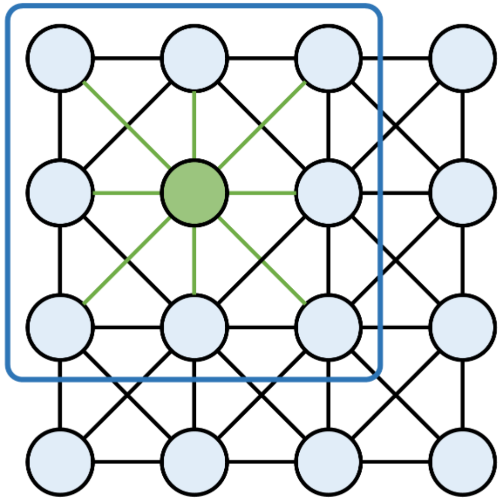


	0	1	2	3	4
0	0	1	1	0	0
1					
2					
3					
4					

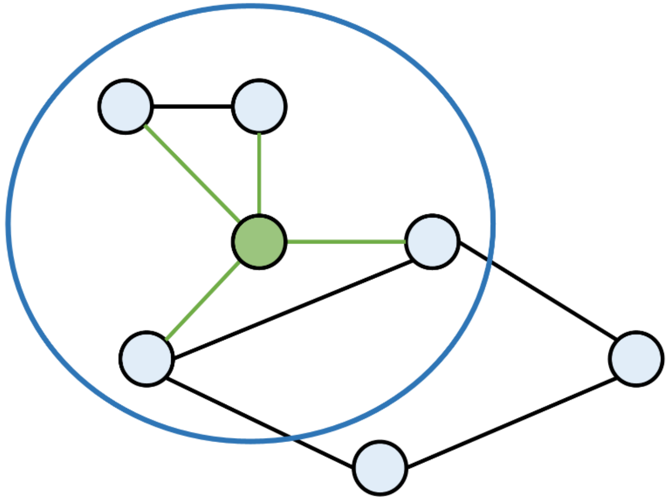
Connection between CNN and GCN

- GCNs perform similar operations where the model learns the features by inspecting neighboring nodes
- The major difference
  - CNNs are specially built to operate on regular structured data
  - GCNs operate for the graph data that the number of nodes connections vary and the nodes are unordered

Kernel

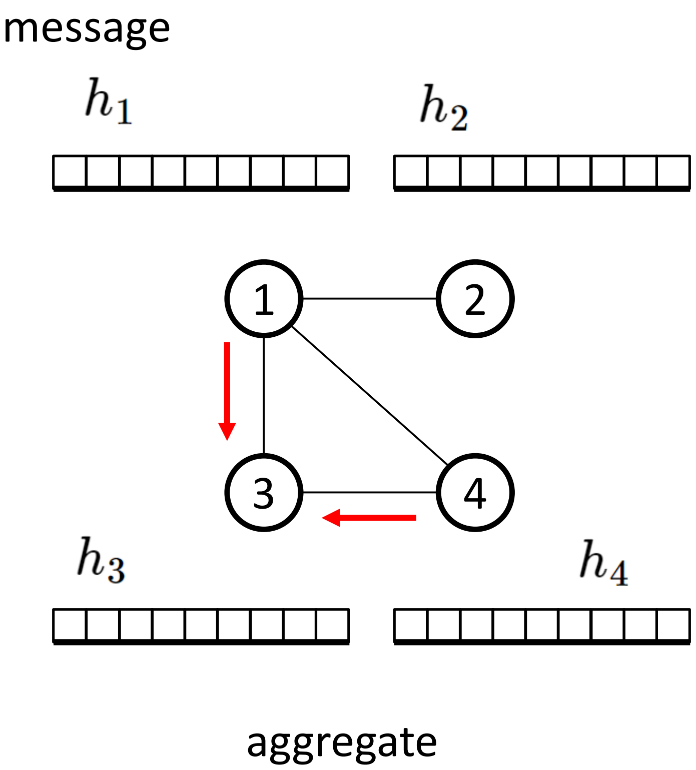


Adjacent matrix
and Kernel

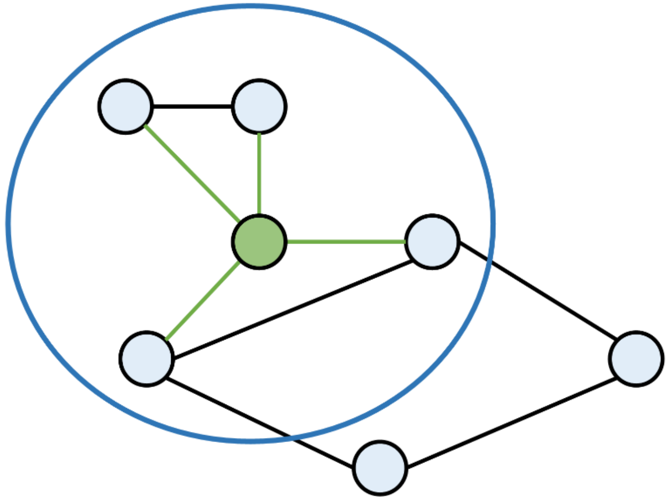


Basics of GCN

- Similar to CNN, GCN updates each node with their adjacent nodes
- Unlike CNN, each node of GCN has different number of adjacent nodes
  - Indicate adjacent nodes of each node by adjacency matrix A



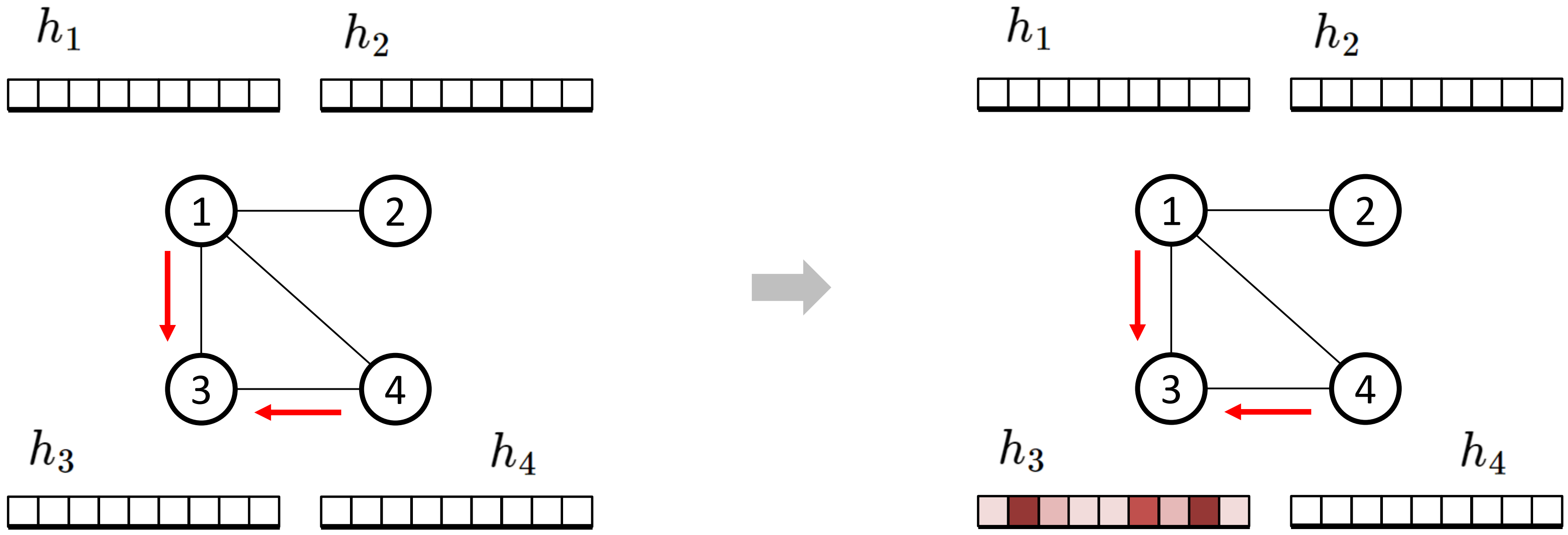
Adjacent matrix and Kernel



Basics of GCN

1) Message: information passed by neighboring nodes to the central node
2) Aggregate: collect information from neighboring nodes
3) Update: embedding update by combining information from neighboring nodes and from itself

1) message



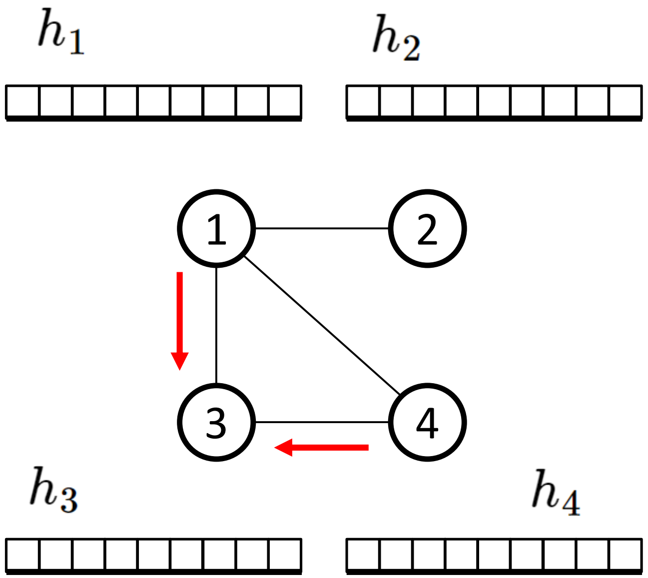
aggregate

update

2) Message Aggregation from Local Neighborhood

$$h_u^{(k+1)} = \quad \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)$$

$$H^{(k+1)} = \quad AH^{(k)}$$



aggregate

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \text{—} h_1 \text{—} \\ \text{—} h_2 \text{—} \\ \text{—} h_3 \text{—} \\ \text{—} h_4 \text{—} \end{bmatrix}$$

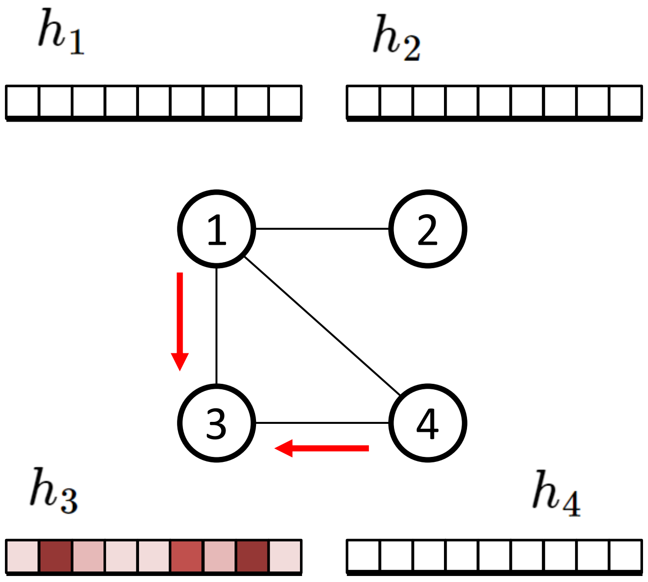
aggregate

$$\Rightarrow \begin{bmatrix} \text{———} \\ \text{———} \\ \text{———} \\ \text{———} \end{bmatrix}$$

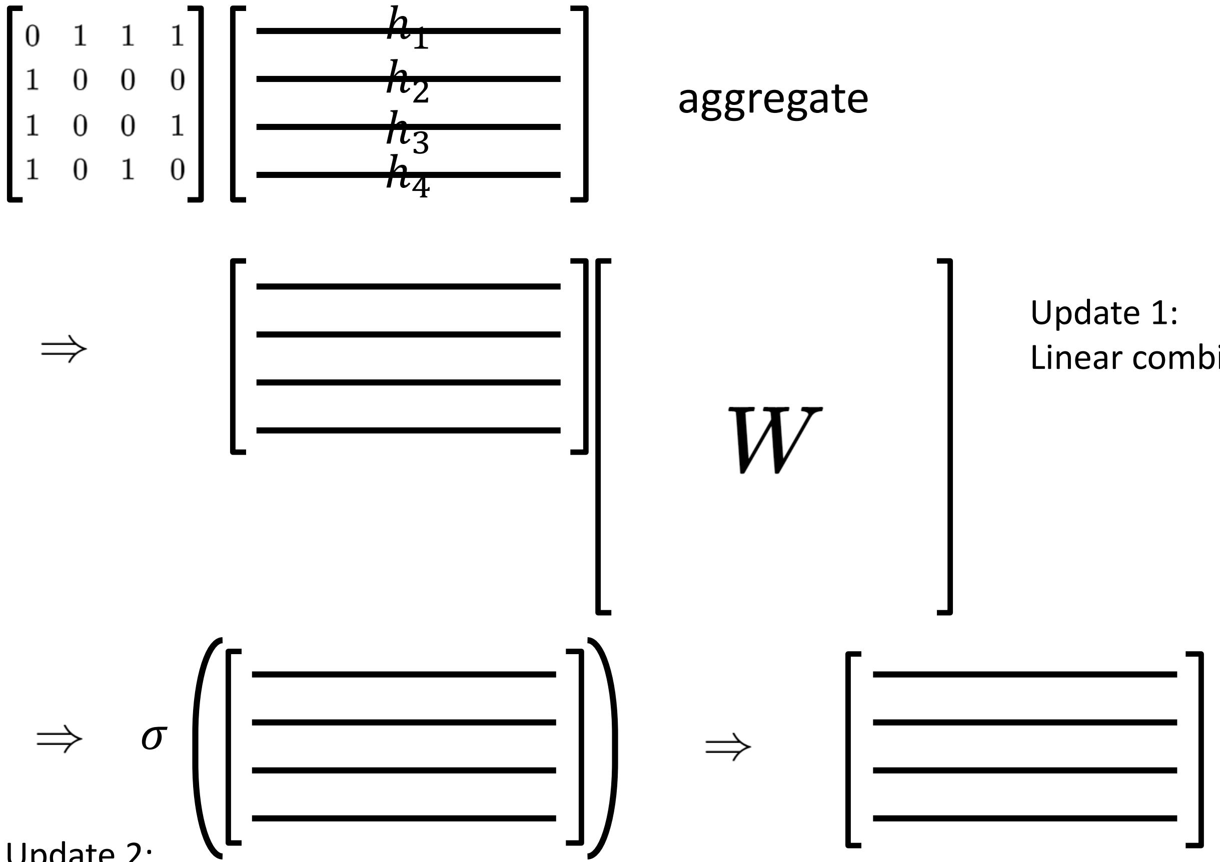
3) Update

$$h_u^{(k+1)} = \text{UPDATE}\left(\quad \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

$$H^{(k+1)} = \sigma\left(\quad AH^{(k)} W_{\text{neigh}}^{(k)}\right)$$



update



Update 2:
Non-linear function

Further Improvements

$$h_u^{(k+1)} = \text{UPDATE}\left(\quad \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

$$H^{(k+1)} = \sigma\left(\qquad AH^{(k)}\, W_{\text{neigh}}^{(k)}\right)$$

- Message Passing with Self-Loops
  - As a simplification of the neural message passing approach, it is common to add self-loops to the input graph and omit the explicit update step

$$\begin{aligned} h_u^{(k+1)} &= \text{UPDATE}\left(h_u^{(k)}, \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right) \\ &= \text{UPDATE}\left(\text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u) \cup \{u\}\right\}\right)\right) \end{aligned}$$

$$H^{(k+1)} = \sigma\left((A+I)\, H^{(k)}\, W^{(k)}\right)$$

Further Improvements

$$h_u^{(k+1)} = \text{UPDATE}\left(\boxed{}\ \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

$$H^{(k+1)} = \sigma\left(\boxed{}\ A H^{(k)} W_{\text{neigh}}^{(k)}\right)$$

- Message Passing with Self-Loops
  - As a simplification of the neural message passing approach, it is common to add self-loops to the input graph and omit the explicit update step

$$\begin{aligned} h_u^{(k+1)} &= \text{UPDATE}\left(h_u^{(k)}, \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right) \\ &= \text{UPDATE}\left(\text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u) \cup \{u\}\right\}\right)\right) \end{aligned}$$

$$H^{(k+1)} = \sigma\left((A + I)\, H^{(k)}\, W^{(k)}\right)$$

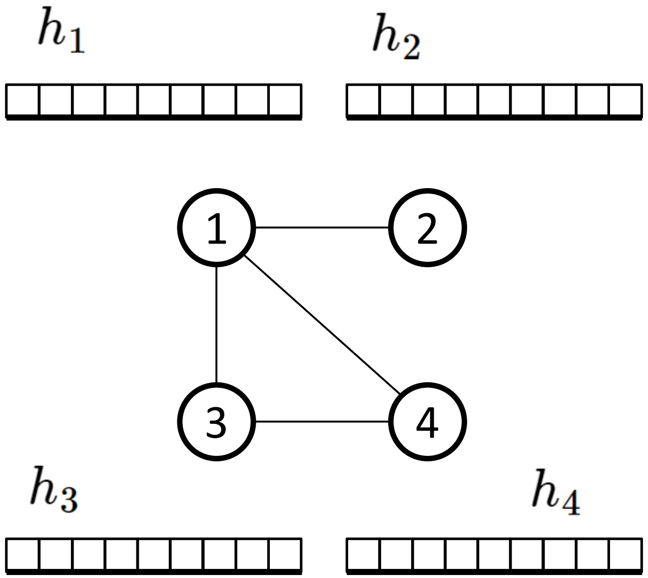
- Neighborhood Normalization
  - The most basic neighborhood aggregation operation simply takes the sum of the neighbor embedding.
  - One issue with this approach is that it can be unstable and highly sensitive to node degrees.
  - One solution to this problem is to simply normalize the aggregation operation based upon the degrees of the nodes involved.
  - The simplest approach is to just take a weighted average rather than sum.

$$\begin{aligned} \tilde{A} &= D^{-1/2} A D^{-1/2} + I \\ &\approx \tilde{D}^{-1/2} (A + I) \tilde{D}^{-1/2} \qquad \text{where } \tilde{D} \text{ is the degree matrix of } A + I \end{aligned}$$

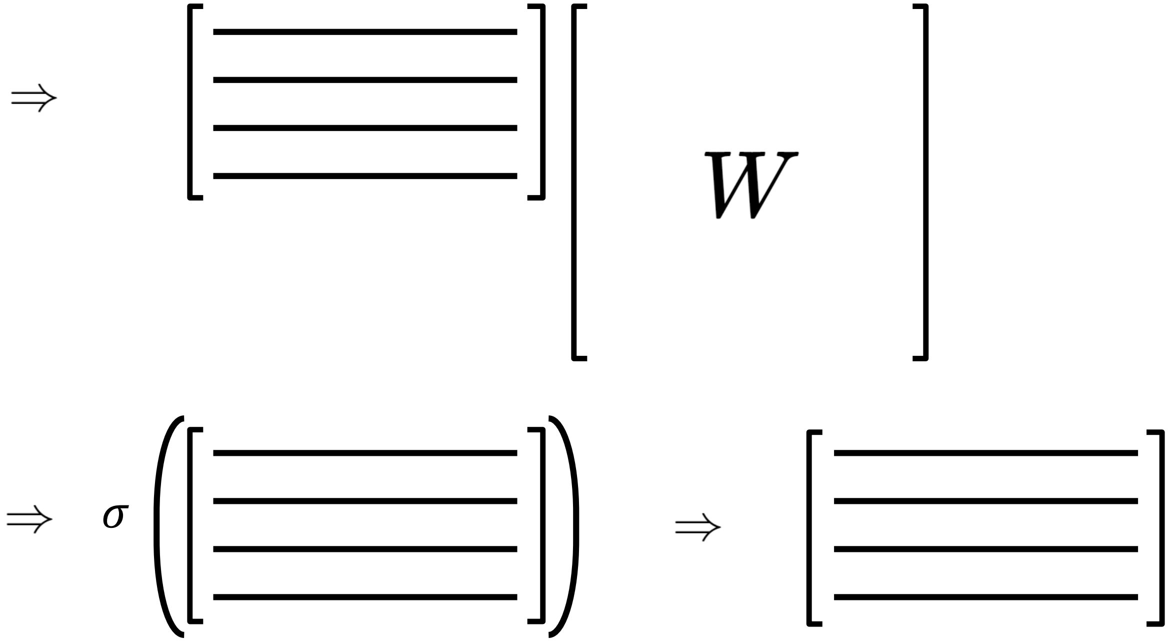
Message Passing

$$h_u^{(k+1)} = \text{UPDATE}\left(\boxed{}\ \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

$$H^{(k+1)} = \sigma\left(\boxed{}\ AH^{(k)} W_{\text{neigh}}^{(k)}\right)$$



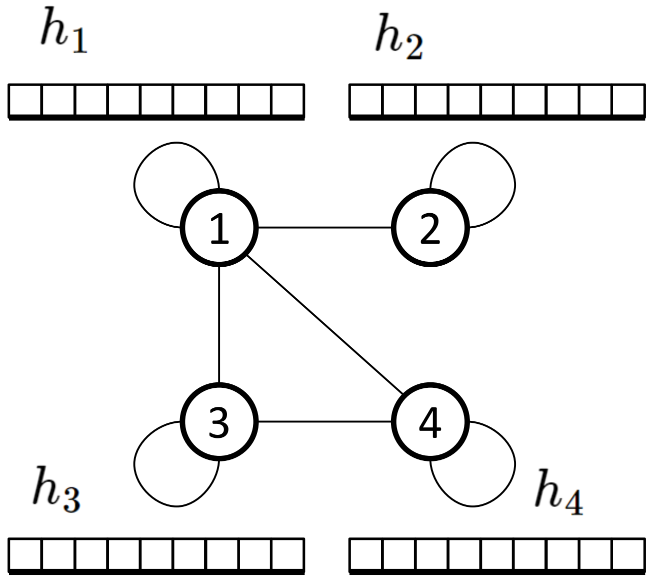
$$\overset{A}{\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$



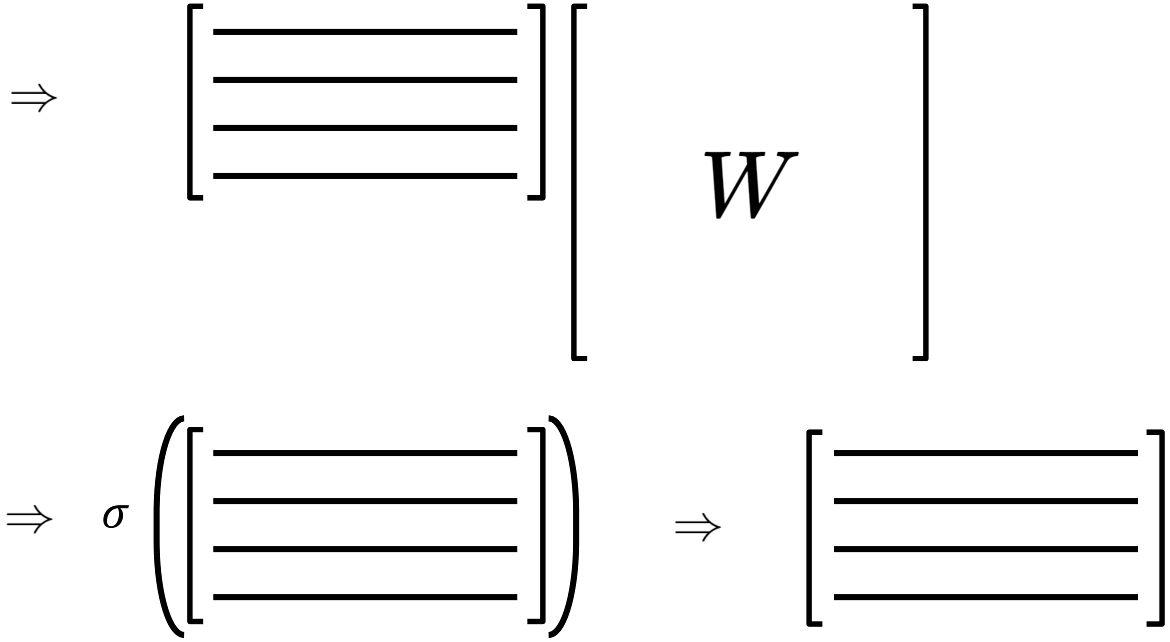
1) Message Passing with Self-Loops

$$h_u^{(k+1)} = \text{UPDATE}\left(h_u^{(k)}, \text{AGGREGATE}\left(\left\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right)$$

$$H^{(k+1)} = \sigma\left(AH^{(k)}W_{\text{self}}^{(k)} + AH^{(k)}W_{\text{neigh}}^{(k)}\right)$$



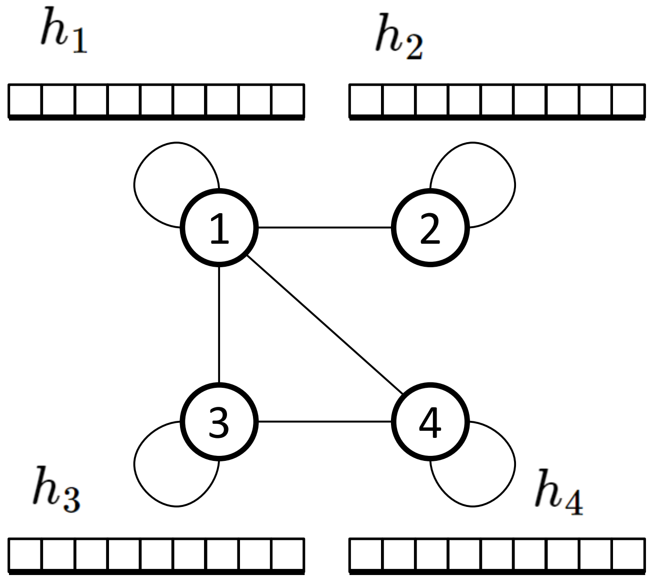
$$\overset{A+I}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$

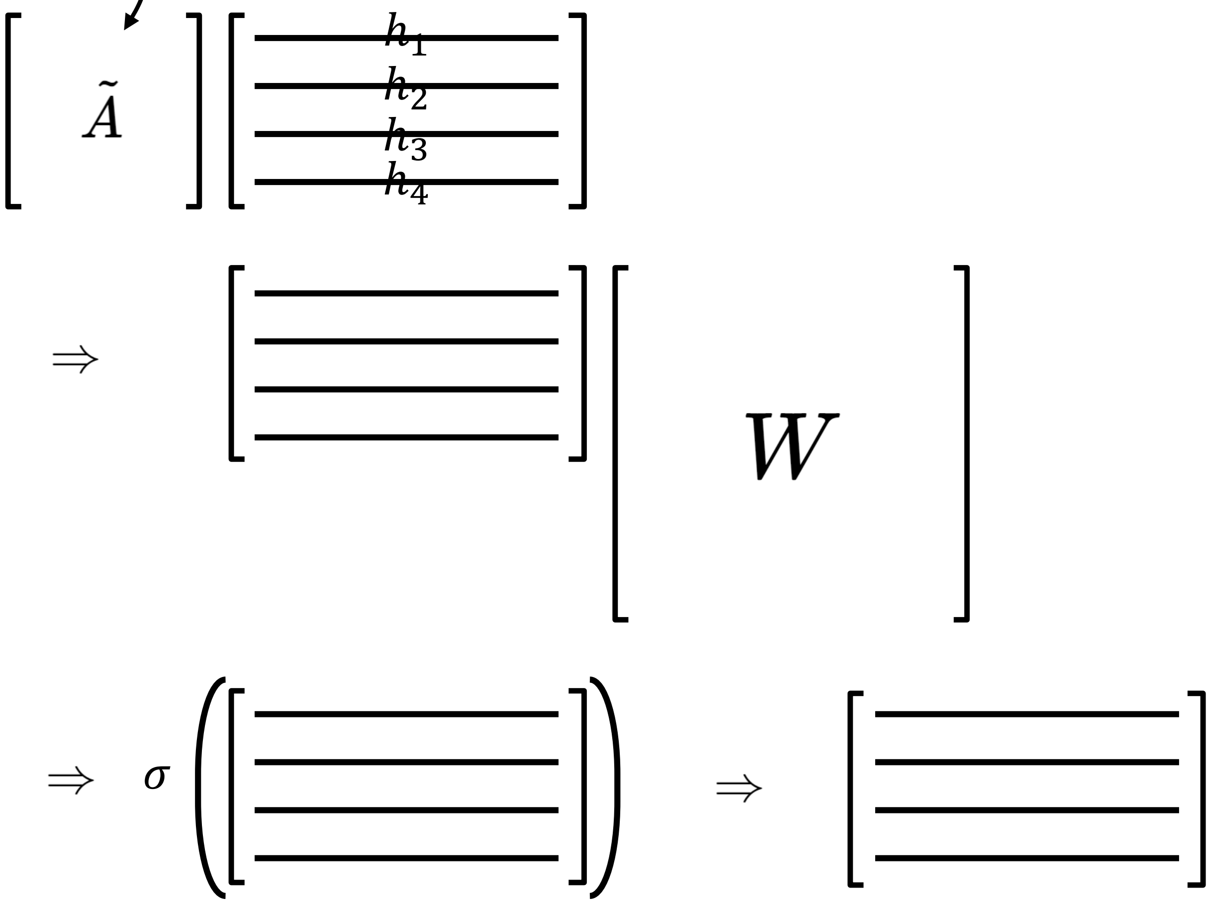


2) Neighborhood Normalization

```
[[0.25      , 0.35355339, 0.28867513, 0.28867513],
 [0.35355339, 0.5       , 0.        , 0.        ],
 [0.28867513, 0.        , 0.33333333, 0.33333333],
 [0.28867513, 0.        , 0.33333333, 0.33333333]]
```

$$\leftarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

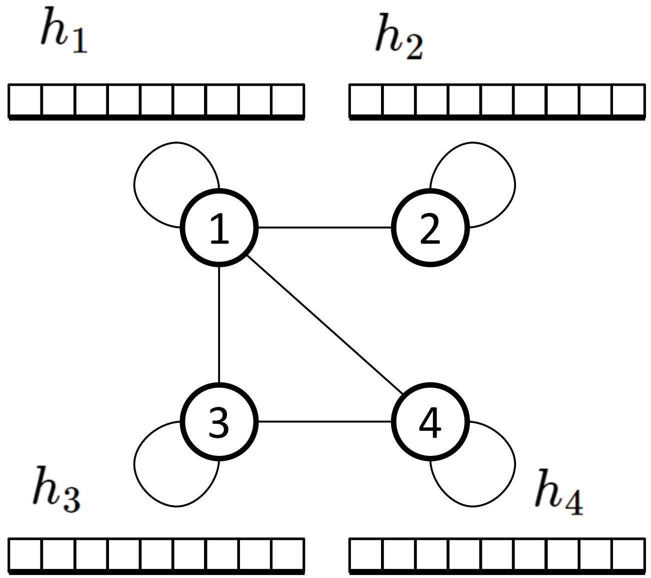


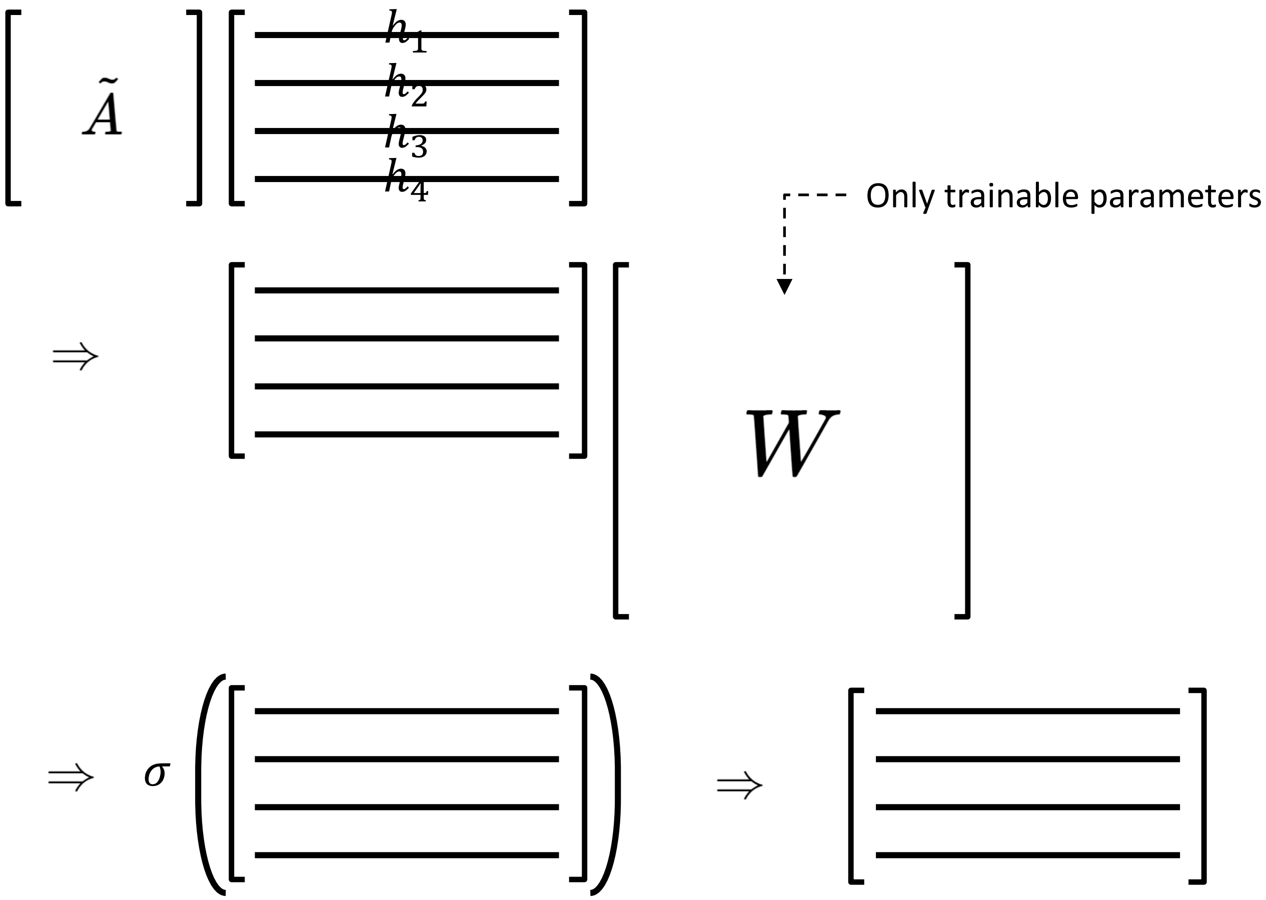


$$H^{(k+1)} = \sigma\left(\left(\tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}\right) H^{(k)} W^{(k)}\right)$$

$$= \sigma\left(\tilde{A} H^{(k)} W^{(k)}\right)$$

Q: Which One Is Trainable?





$$H^{(k+1)} = \sigma\left(\left(\tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}\right) H^{(k)}\, W^{(k)}\right)$$

$$= \sigma\left(\tilde{A}H^{(k)}\, W^{(k)}\right)$$

Finally Graph Convolutional Networks

- Multi-layer Graph Convolutional Network (GCN)

$$H^{(k+1)} = \sigma\left((A+I)\, H^{(k)}\, W^{(k)}\right)$$

← Self-loops

$$\Downarrow$$

$$H^{(k+1)} = \sigma\left(\left(\tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}\right) H^{(k)}\, W^{(k)}\right)$$

$$= \sigma\left(\tilde{A} H^{(k)}\, W^{(k)}\right)$$

← Neighborhood Normalization

$$\begin{aligned}\tilde{A} &= D^{-1/2} A D^{-1/2} + I \\ &\approx \tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}\end{aligned}$$

Finally Graph Convolutional Networks

- Multi-layer Graph Convolutional Network (GCN)

$$H^{(k+1)} = \sigma\left((A+I)\, H^{(k)}\, W^{(k)}\right)$$

$$\Downarrow$$

$$H^{(k+1)} = \sigma\left(\left(\tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}\right) H^{(k)}\, W^{(k)}\right)$$

$$= \sigma\left(\tilde{A} H^{(k)}\, W^{(k)}\right)$$

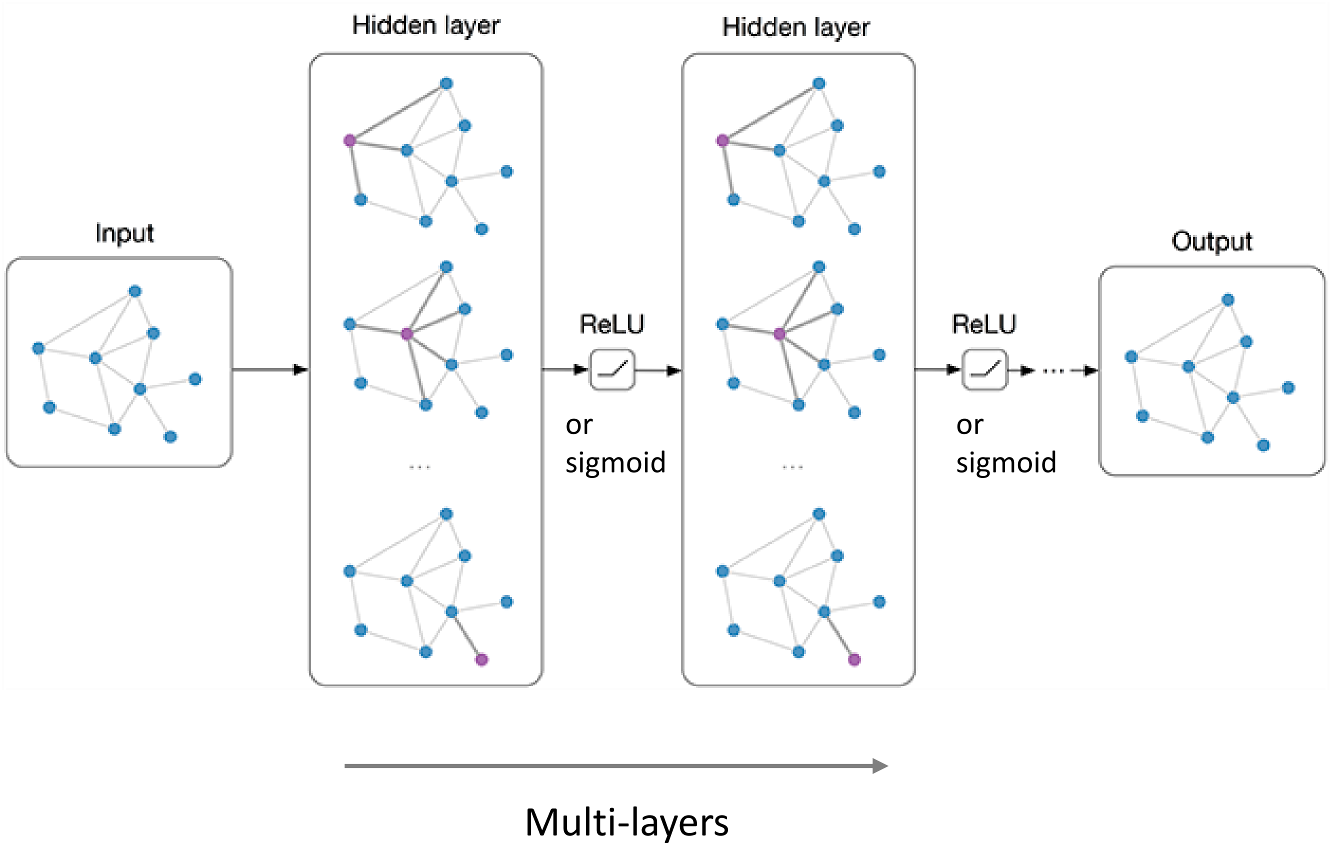


Image from https://tkipf.github.io/graph-convolutional-networks/

Feature Vector Updates

```python
A = nx.adjacency_matrix(G).todense()
```

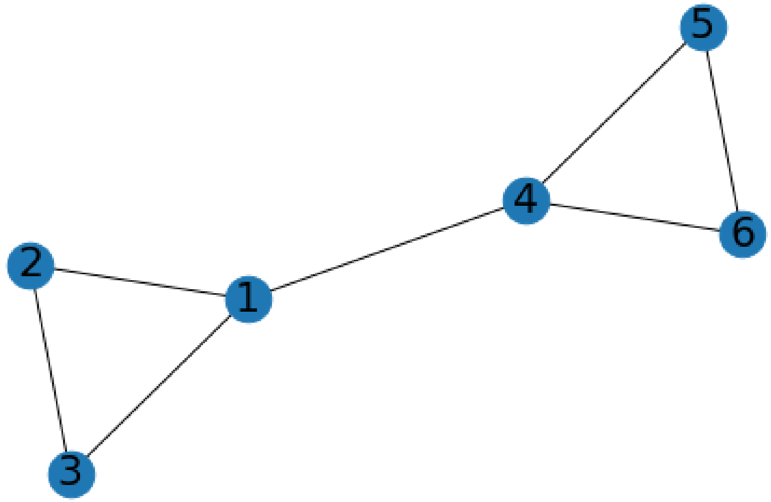
```
[[0 1 1 1 0 0]
 [1 0 1 0 0 0]
 [1 1 0 0 0 0]
 [1 0 0 0 1 1]
 [0 0 0 1 0 1]
 [0 0 0 1 1 0]]
```

```python
H = np.matrix([1,0,0,-1,0,0]).T
```

```
[[ 1]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
```

```python
A*H
```

```
matrix([[-1],
        [ 1],
        [ 1],
        [ 1],
        [-1],
        [-1]])
```



Feature Vector Updates

```
A = nx.adjacency_matrix(G).todense()
```

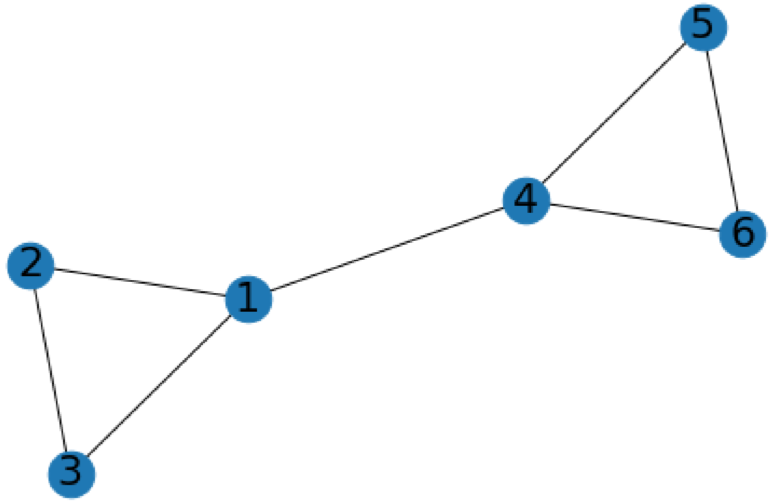
```
[[0 1 1 1 0 0]
 [1 0 1 0 0 0]
 [1 1 0 0 0 0]
 [1 0 0 0 1 1]
 [0 0 0 1 0 1]
 [0 0 0 1 1 0]]
```

```
H = np.matrix([1,0,0,-1,0,0]).T
```

```
[[ 1]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
```

```
A_self = A + np.eye(6)

A_self*H
```

```
matrix([[ 0.],
        [ 1.],
        [ 1.],
        [ 0.],
        [-1.],
        [-1.]])
```



POSTECH

Feature Vector Updates

```
A = nx.adjacency_matrix(G).todense()
```

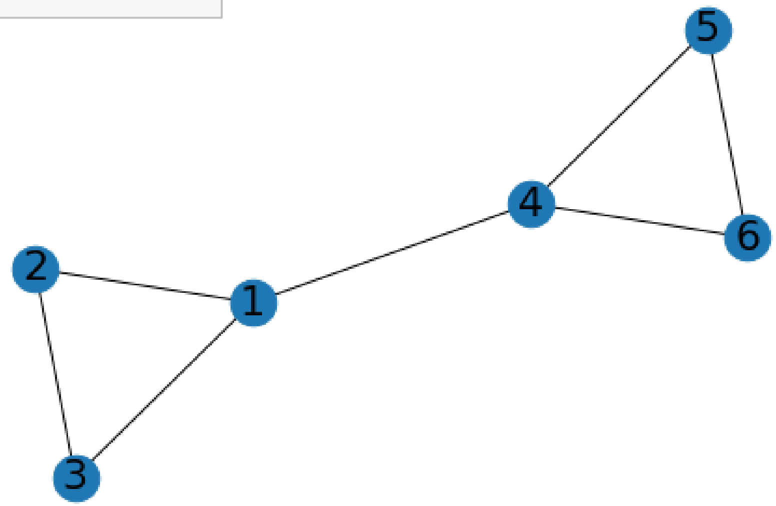
```
[[0 1 1 1 0 0]
 [1 0 1 0 0 0]
 [1 1 0 0 0 0]
 [1 0 0 0 1 1]
 [0 0 0 1 0 1]
 [0 0 0 1 1 0]]
```

```
H = np.matrix([1,0,0,-1,0,0]).T
```

```
[[ 1]
 [ 0]
 [ 0]
 [-1]
 [ 0]
 [ 0]]
```

```
D = np.array(A_self.sum(1)).flatten()
D = np.diag(D)

D_half_norm = fractional_matrix_power(D, -0.5)

A_self = np.asmatrix(A_self)
D_half_norm = np.asmatrix(D_half_norm)

A_half_norm = D_half_norm*A_self*D_half_norm

A_half_norm*H
```

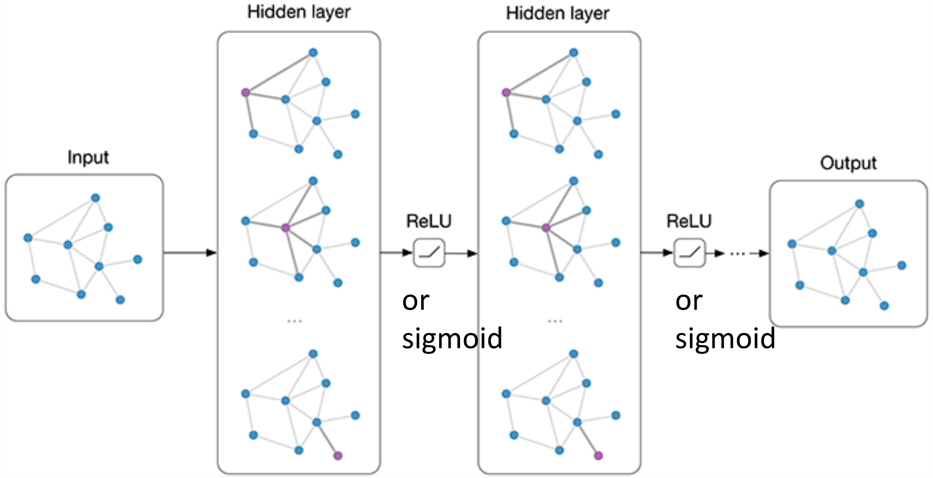
```
matrix([[ 0.        ],
        [ 0.28867513],
        [ 0.28867513],
        [ 0.        ],
        [-0.28867513],
        [-0.28867513]])
```



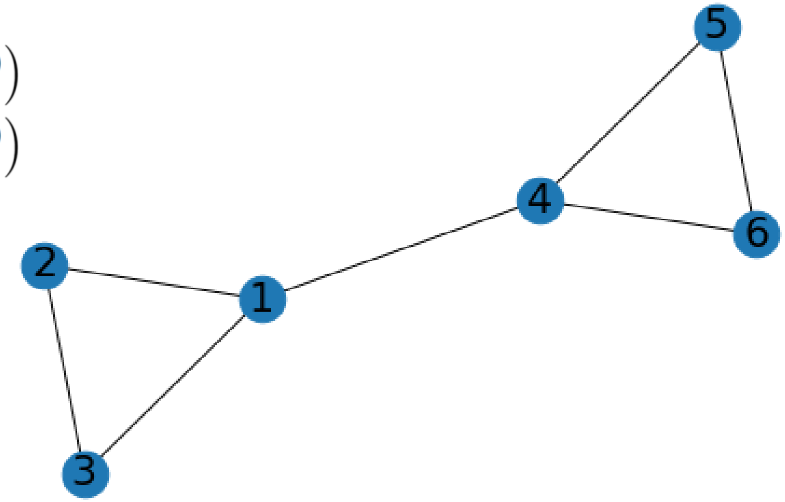
Build 2-layer GCN using ReLU as the Activation Function

```
W1 = np.random.randn(1, 4) # input: 1 -> hidden: 4
W2 = np.random.randn(4, 2) # hidden: 4 -> output: 2

def relu(x):
    return np.maximum(0, x)

def gcn(A, H, W):
    D = np.diag(np.array(A_self.sum(1)).flatten())
    D_half_norm = fractional_matrix_power(D, -0.5)
    H_new = D_half_norm*A_self*D_half_norm*H*W
    return relu(H_new)

H1 = H
H2 = gcn(A, H1, W1)
H3 = gcn(A, H2, W2)

print(H3)
```

```
[[0.         0.07472825]
 [0.         0.08628875]
 [0.         0.08628875]
 [0.12632564 0.        ]
 [0.14586829 0.        ]
 [0.14586829 0.        ]]
```

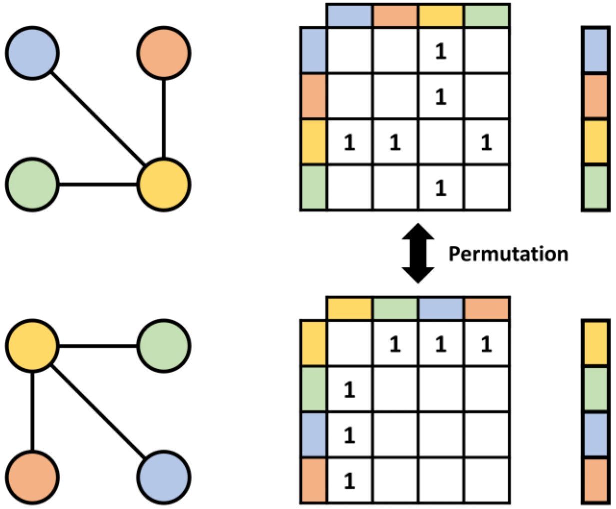


$$H^{(2)} = \mathrm{ReLU}\left(\tilde{A}H^{(1)}W^{(1)}\right)$$
$$H^{(3)} = \mathrm{ReLU}\left(\tilde{A}H^{(2)}W^{(2)}\right)$$



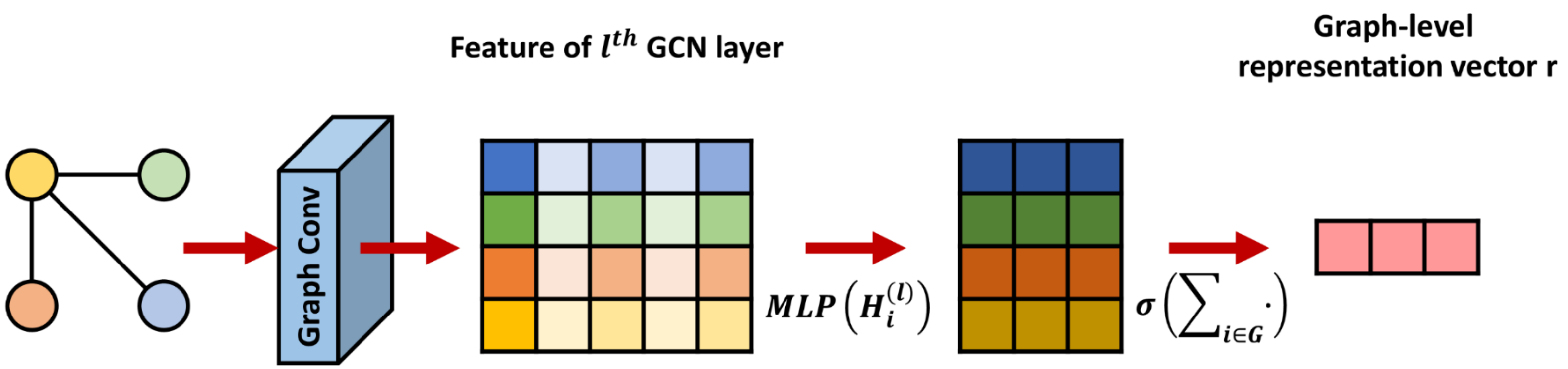
Readout: Permutation Invariance

- Adjacency matrix can be different even though two graph has the same network structure
  - Even if the edge information between all nodes is the same, the order of values in the matrix may be different due to rotation and symmetry
- Readout layer makes this permutation invariant by multiplying MLP



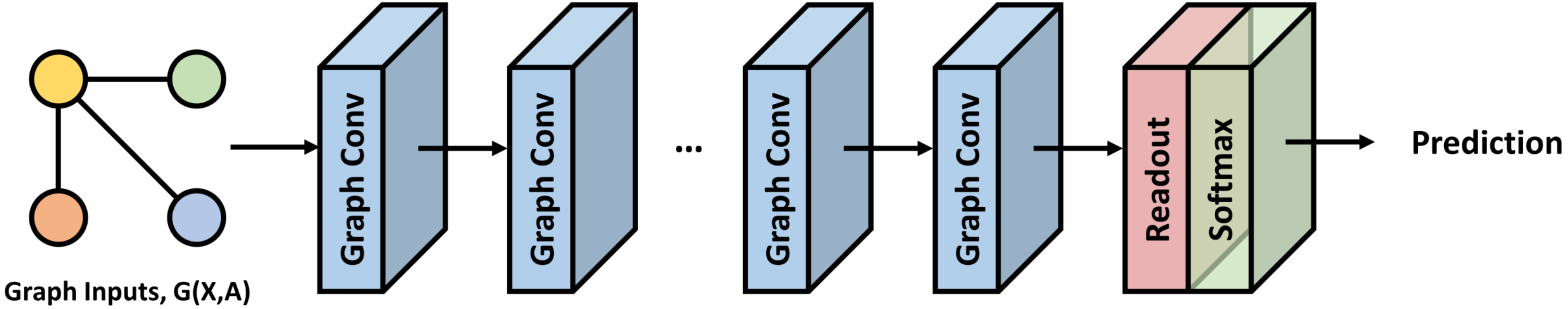
- Node-wise summation

$$Z_G = \tau\left(\sum_{i \in G} MLP\left(H_i^{(L)}\right)\right)$$

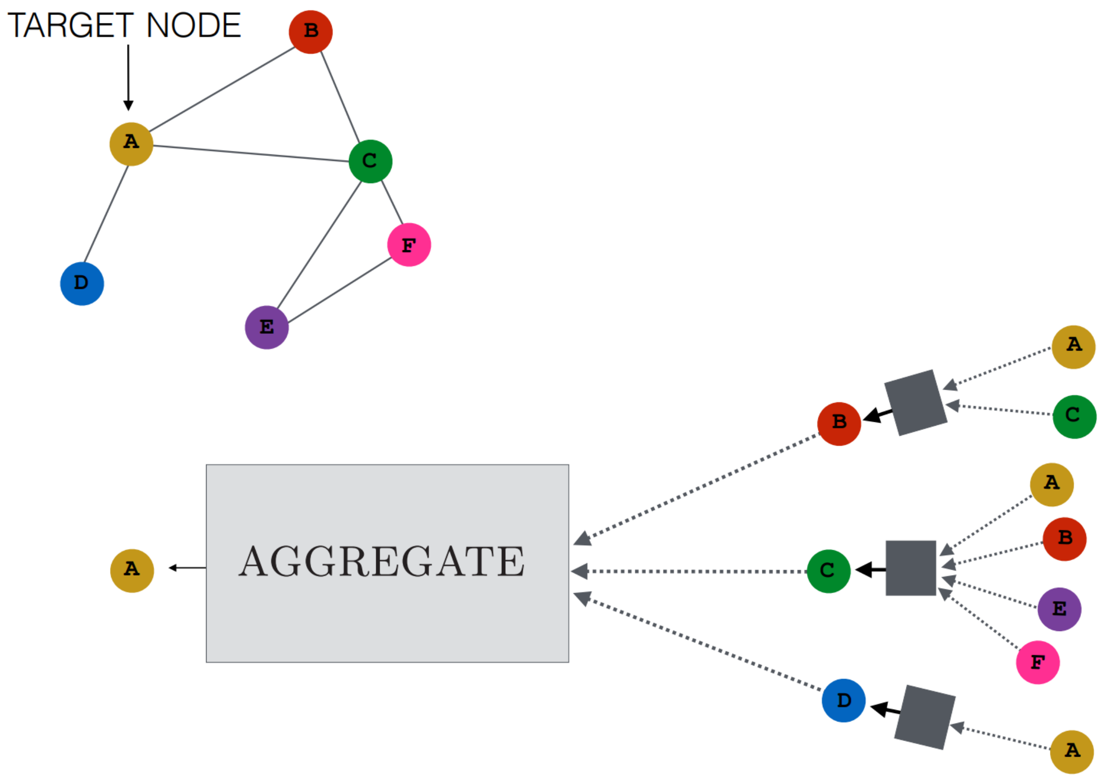


Finally Graph Convolutional Networks

- Similar to convolutional neural network
- Multiple graph convolution layers



GCN as Message Passing Framework



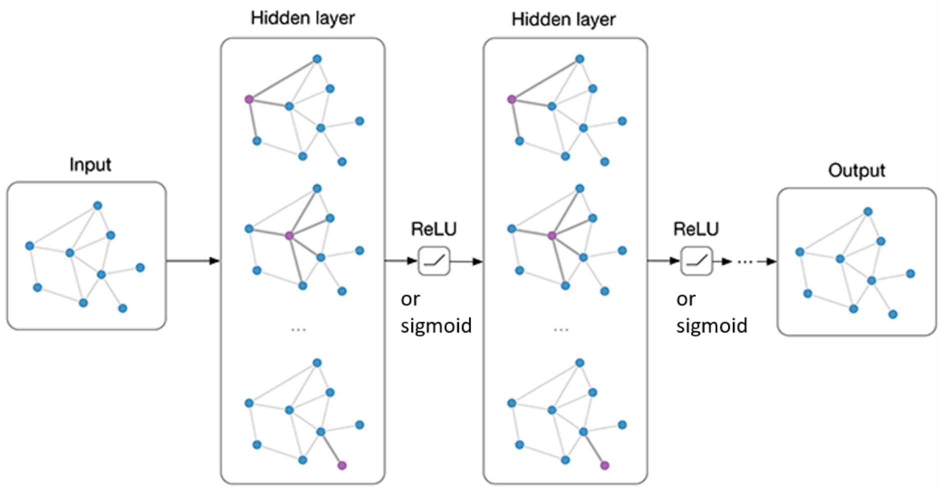
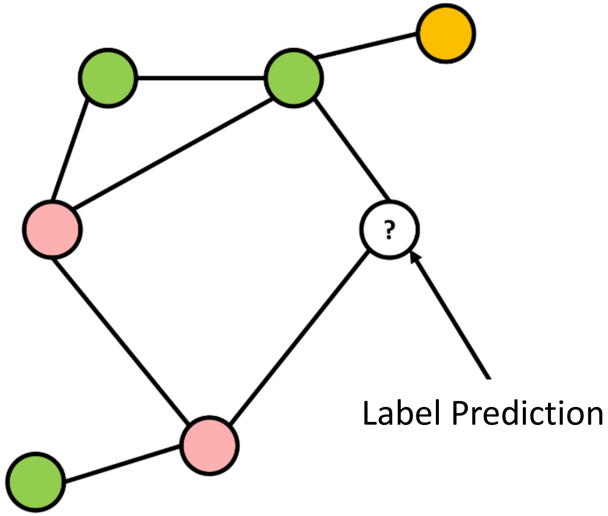


Image from Graph Representation Learning by William L. Hamilton

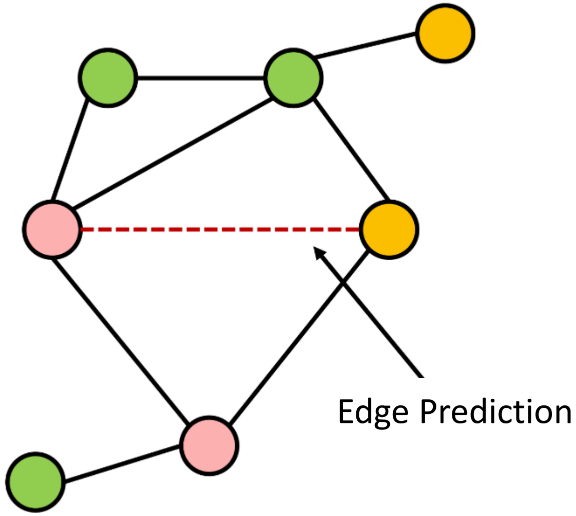
Tasks for Graph Neural Network

- 3 GNN applications

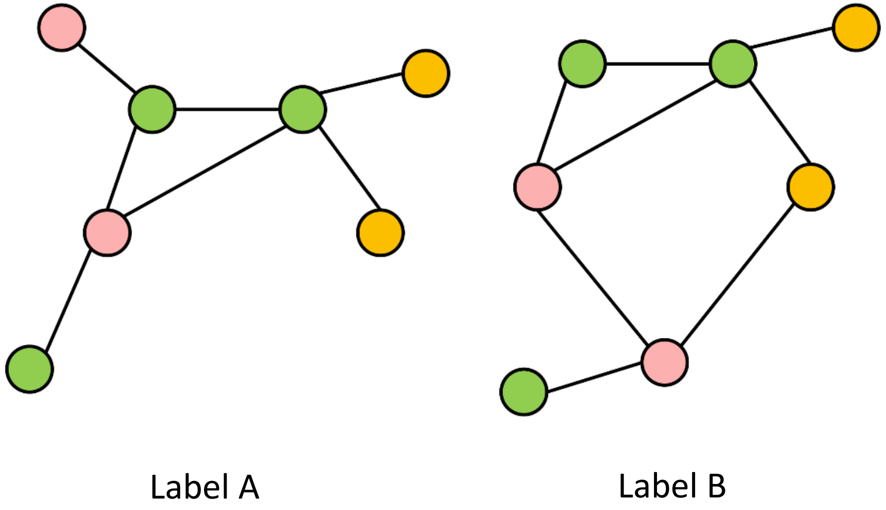
Task 1: node classification



Task 2: edges prediction



Task 3: graph classification



List of GNN Python Libraries

- PyTorch Geometric
  - PyG
  - Built upon PyTorch
- Deep Graph Library (DGL)
  - Based on PyTorch, TensorFlow or Apache MXNet.
- Graph Nets
  - DeepMind's library for building graph networks in Tensorflow and Sonnet
- Spektral
  - Based on the Keras API and TensorFlow 2

From https://neptune.ai/blog/graph-neural-networks-libraries-tools-learning-resources

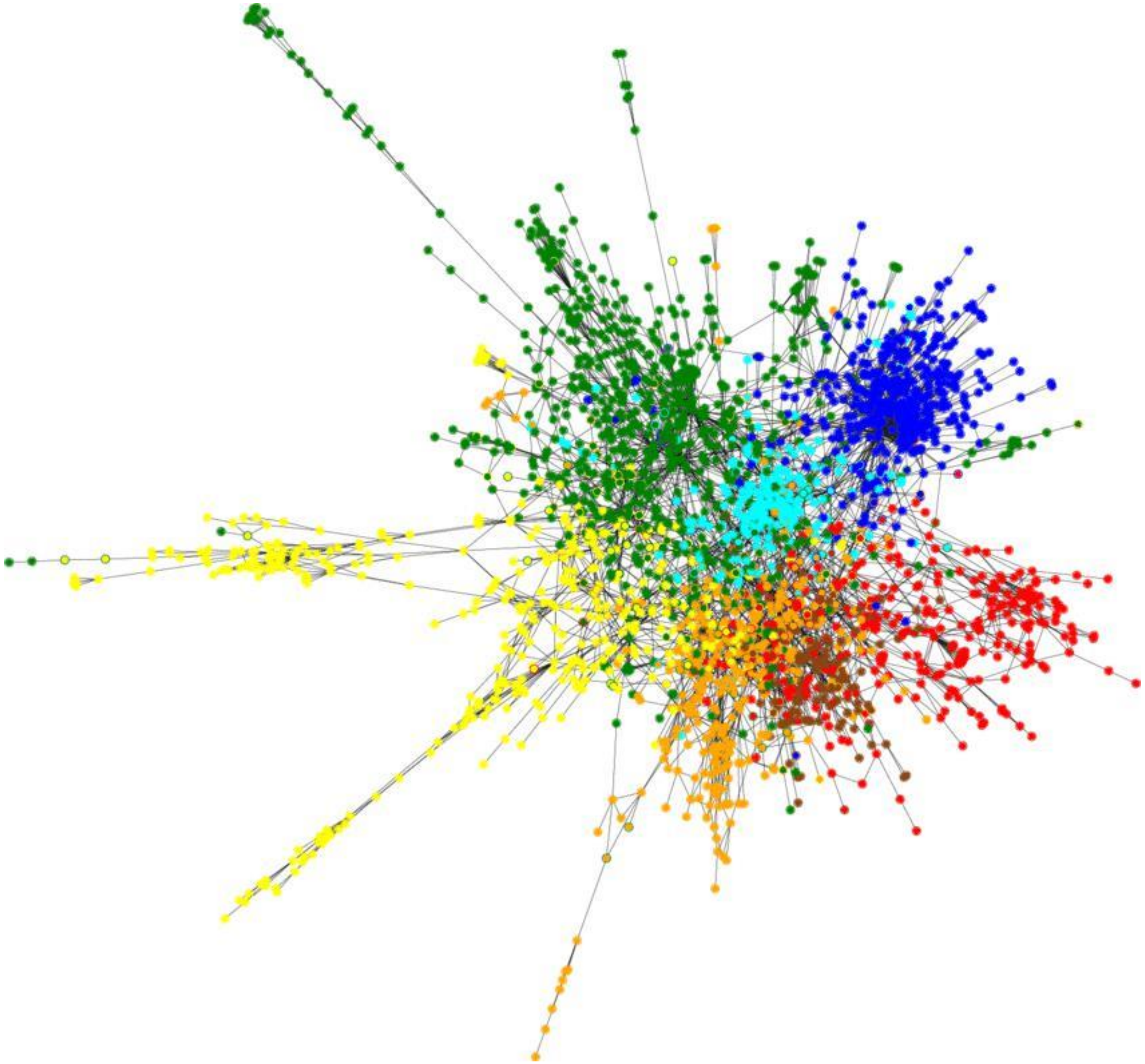
Lab 1: Node Classification using Graph Convolutional Networks

- CORA dataset
  - This dataset is the MNIST equivalent in graph learning
- The CORA dataset consists of 2708 scientific publications classified into one of seven classes.
  - Case_Based: 298
  - Genetic_Algorithms: 418
  - Neural_Networks: 818
  - Probabilistic_Methods: 426
  - Reinforcement_Learning: 217
  - Rule_Learning: 180
  - Theory: 351
- The citation network consists of 5429 links.
- Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.
- The dictionary consists of 1433 unique words.

```
H shape:  (2708, 1433)
The number of nodes (N):  2708
The number of features (F) of each node:  1433
The number of classes:  7
```

CORA dataset

- 2708 nodes of papers
- 5429 edges by citation
- 7 classes



Graph G and Normalized Adjacency Matrix A

```
G = nx.Graph(name = 'Cora')
G.add_nodes_from(nodes)
G.add_edges_from(edge_list)
```

```
A = nx.adjacency_matrix(G)

I = np.eye(A.shape[-1])
A_self = A + I

D = np.diag(np.array(A_self.sum(1)).flatten())
D_half_norm = fractional_matrix_power(D, -0.5)

A_half_norm = D_half_norm * A_self * D_half_norm

A_half_norm = np.array(A_half_norm)
H = np.array(H)
```

GCN Model

```
H_in = tf.keras.layers.Input(shape = (F, ))
A_in = tf.keras.layers.Input(shape = (N, ))

graph_conv_1 = spektral.layers.GraphConv(channels = 16,
                                         activation = 'relu')([H_in, A_in])

graph_conv_2 = spektral.layers.GraphConv(channels = 7,
                                         activation = 'softmax')([graph_conv_1, A_in])

model = tf.keras.models.Model(inputs = [H_in, A_in], outputs = graph_conv_2)

model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-2),
              loss = 'categorical_crossentropy',
              weighted_metrics = ['acc'])

model.summary()
```

Train and Evaluation

- Train

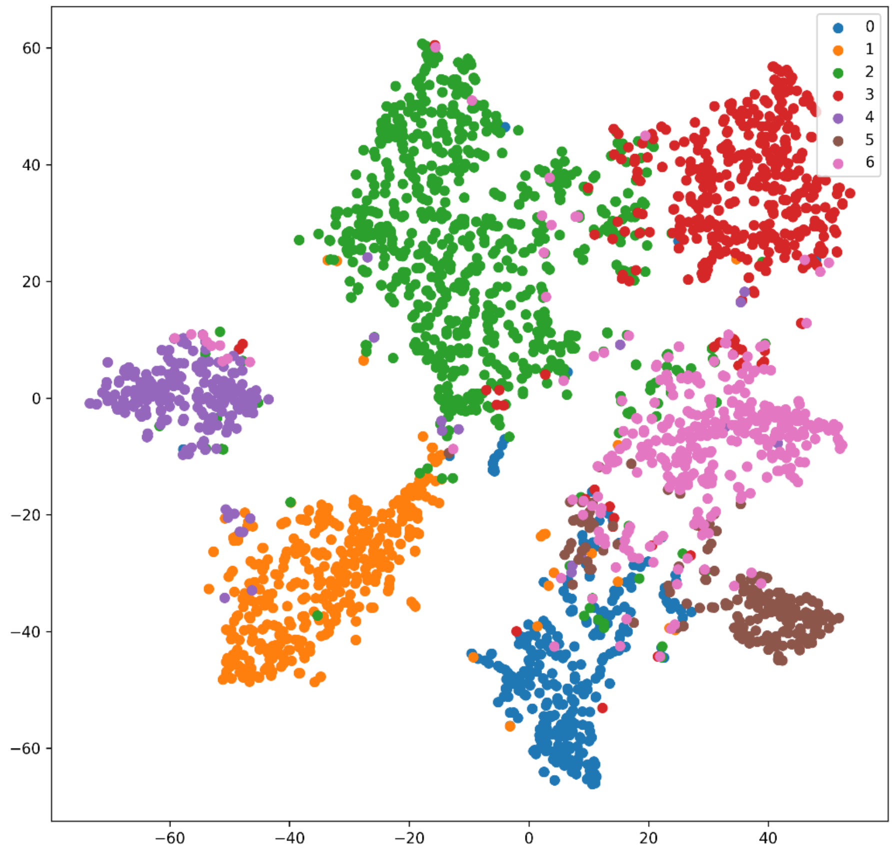
```
model.fit([H, A_half_norm],
          labels_encoded,
          sample_weight = train_mask,
          epochs = 30,
          batch_size = N,
          shuffle = False)
```

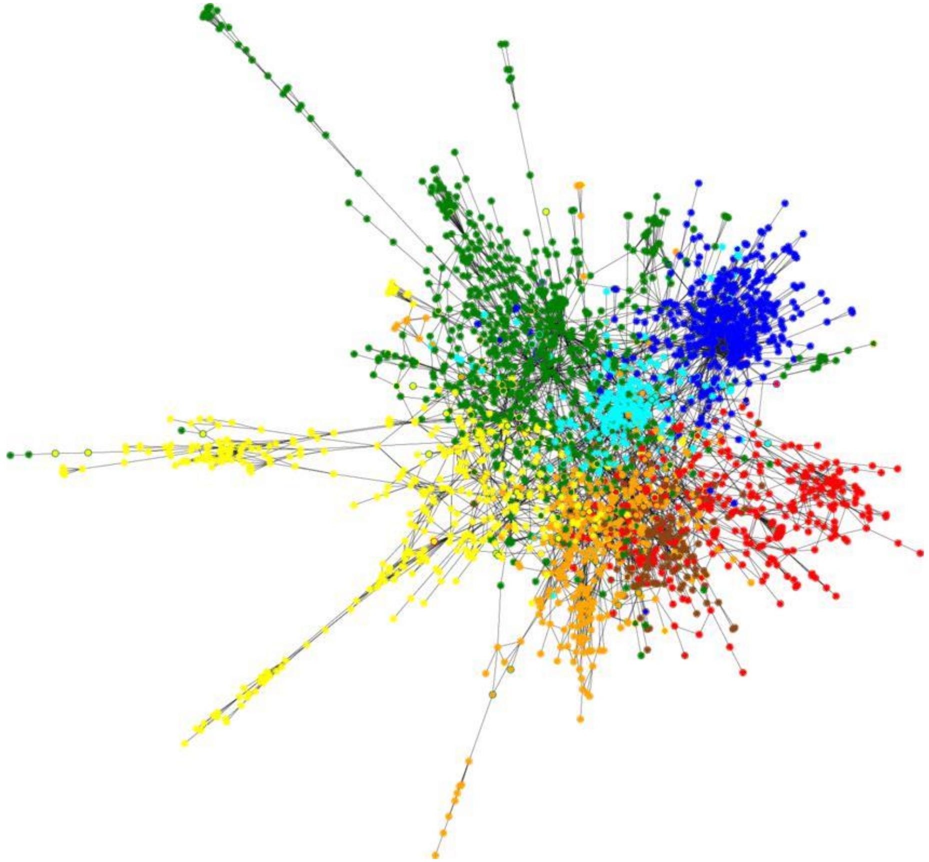
- Evaluation

```
y_pred = model.evaluate([H, A_half_norm],
                        labels_encoded,
                        sample_weight = test_mask,
                        batch_size = N)
```

Low Dimensional Mapping

- T-SNE





Learning Resources for Graph Neural Networks

- Stanford Course: CS224W Machine Learning with Graphs by Prof. Jurij Leskovec
- Github Repository: Collection of Recent GNN Papers
- Graph Neural Network Papers With Code

- Books
  - Network Science by Albert-László Barabási
  - Graph Representation Learning Book by William L. Hamilton

From https://neptune.ai/blog/graph-neural-networks-libraries-tools-learning-resources