STOR566: Introduction to Deep Learning

Lecture 16: Graph Convolutional Network

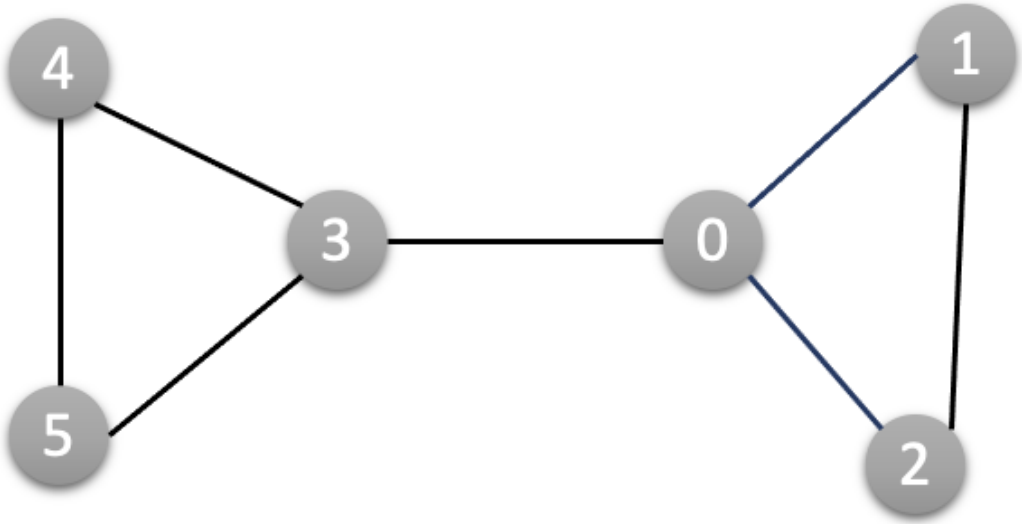
Yao Li
UNC Chapel Hill

Oct 18, 2022

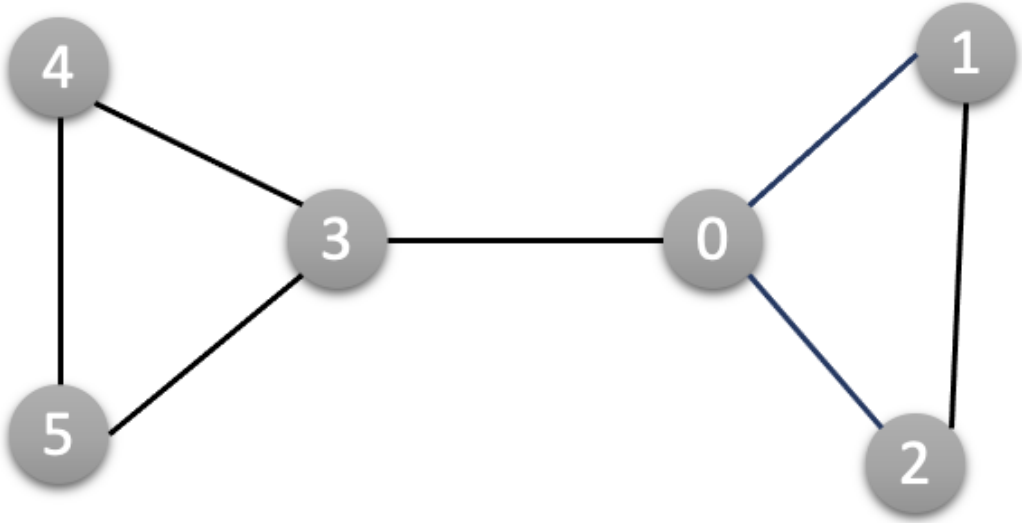
Materials are from *Deep Learning (UCLA)*

Graph Basics

Adjacency Matrix



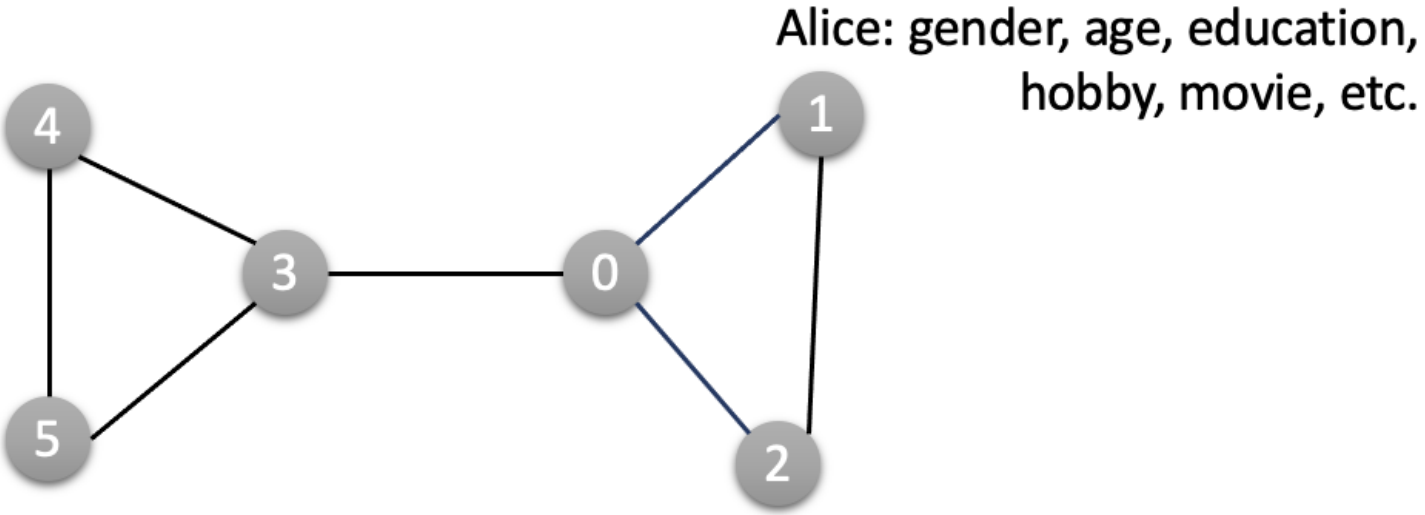
Adjacency Matrix



- Given a graph of N nodes, the adjacency matrix: $A \in \mathbb{R}^{N \times N}$
- A of the example graph:

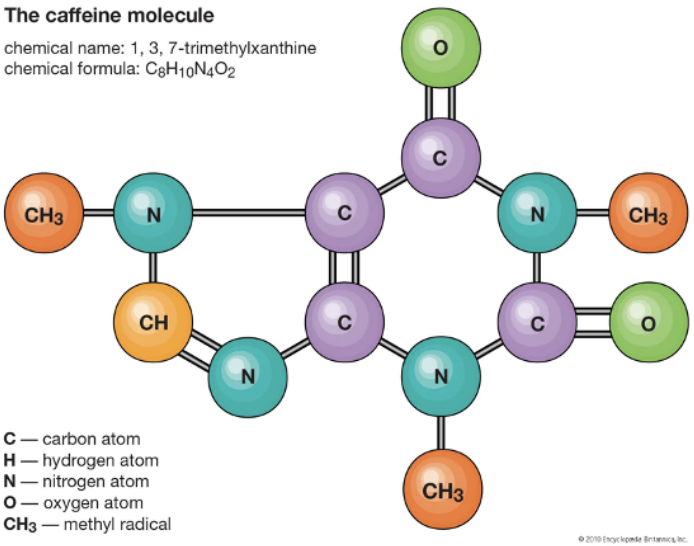
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Node Attribute Matrix

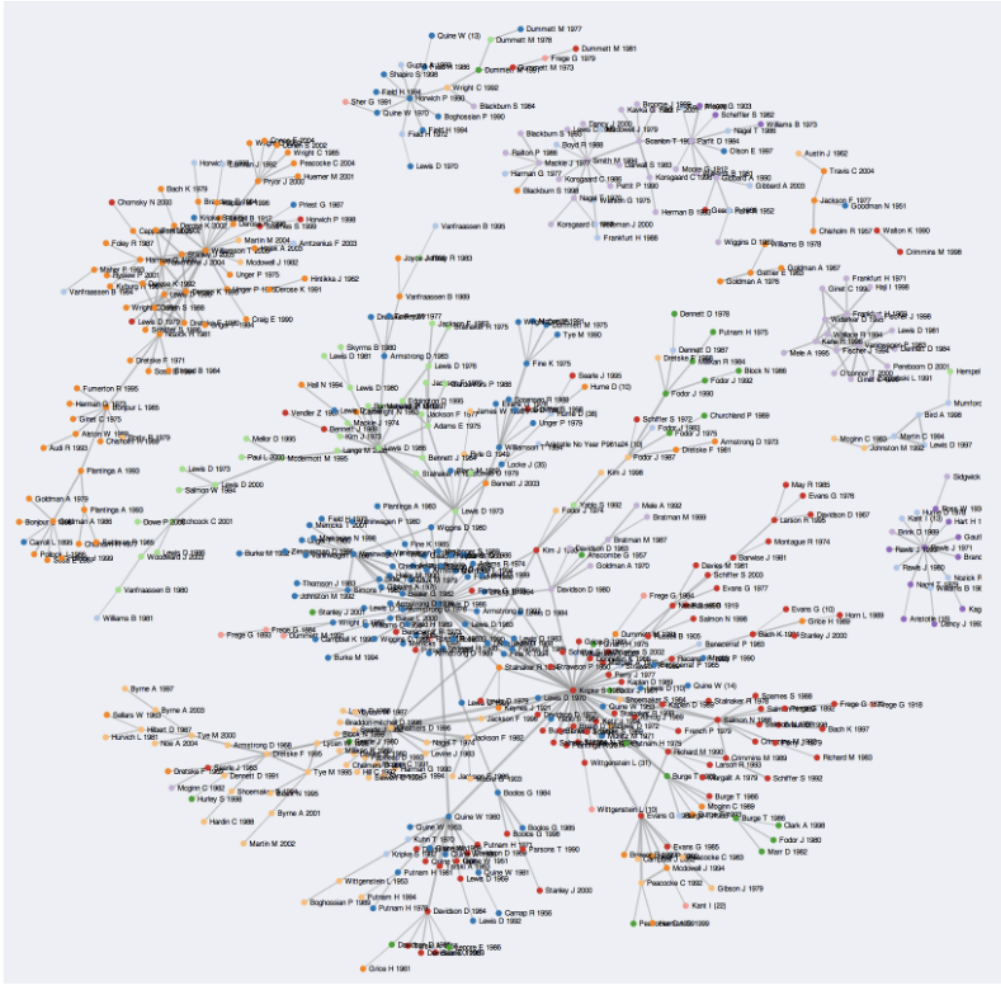


- Each node is associated with a D-dimensional feature vector.
- The node attribute matrix: $X \in \mathbb{R}^{N \times D}$
- Example: $D = 1, X = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$

Data Mode



https://www.britannica.com/science/molecule



https://kieranhealy.org/blog/archives/2013/06/18/a-co-citation-network-for-philosophy/

- Each observation is a graph: classify chemical molecules (Batch mode)
- All observations make one graph: classify documents within a document citation network (Single mode)

Graph Convolutional Neural Network

Node classification problem

- Given a graph of N nodes, with adjacency matrix $A \in \mathbb{R}^{N \times N}$
- Each node is associated with a D-dimensional feature vector.
- $X \in \mathbb{R}^{N \times D}$: each row corresponds to the feature vector of a node
- Observe labels for a subset of nodes: $Y \in \mathbb{R}^{N \times L}$, only observe a subset of rows, denoted by Y_S
- Goal: Predict labels for unlabeled nodes (transductive setting)

Reminder of NN

- Given hidden input $H^{(l)}$, the hidden output is

$$H^{(l+1)} = \sigma\left(H^{(l)} W^{(l)} + \boldsymbol{b}^{(l)}\right)$$

- $W^{(l)}$: weight matrix of layer l
- $\boldsymbol{b}^{(l)}$: bias at layer l
- $\sigma(\cdot)$: activation function

Reminder of NN

- Given hidden input $H^{(l)}$, the hidden output is

$$H^{(l+1)} = \sigma\left(H^{(l)}W^{(l)} + \boldsymbol{b}^{(l)}\right)$$

- $W^{(l)}$: weight matrix of layer l
- $\boldsymbol{b}^{(l)}$: bias at layer l
- $\sigma(\cdot)$: activation function

- Replace input $H^{(l)}$ with X (the node feature matrix)

$$H^{(1)} = \sigma\left(XW^{(0)} + \boldsymbol{b}^{(0)}\right)$$

- Problem: Graph information not used!

Graph Convolution

- Given hidden input $H^{(l)}$, the hidden output is

$$H^{(l+1)} = \sigma\left(PH^{(l)}W^{(l)} + \boldsymbol{b}^{(l)}\right)$$

- P: normalized from the Adjacency Matrix A

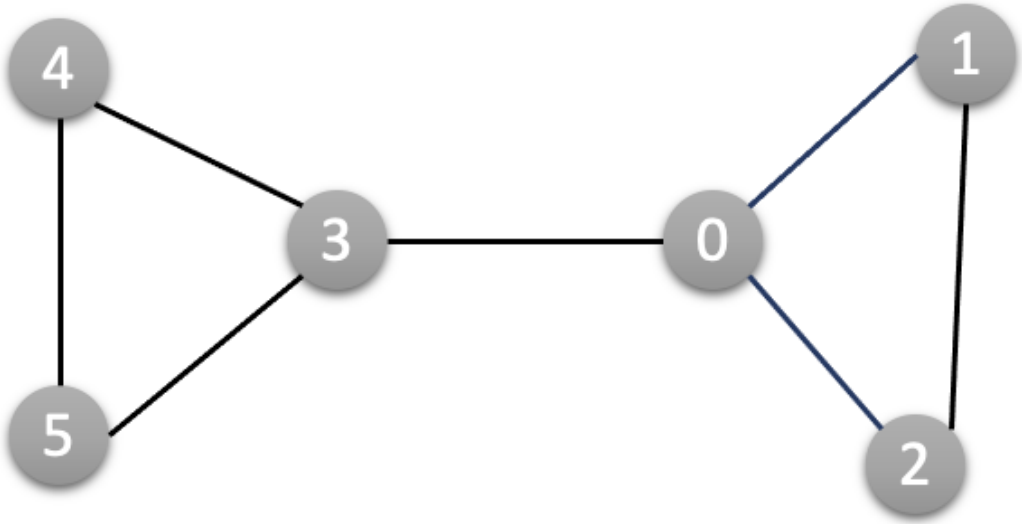
Graph Convolution

- Given hidden input $H^{(l)}$, the hidden output is

$$H^{(l+1)} = \sigma\left(PH^{(l)}W^{(l)} + \boldsymbol{b}^{(l)}\right)$$

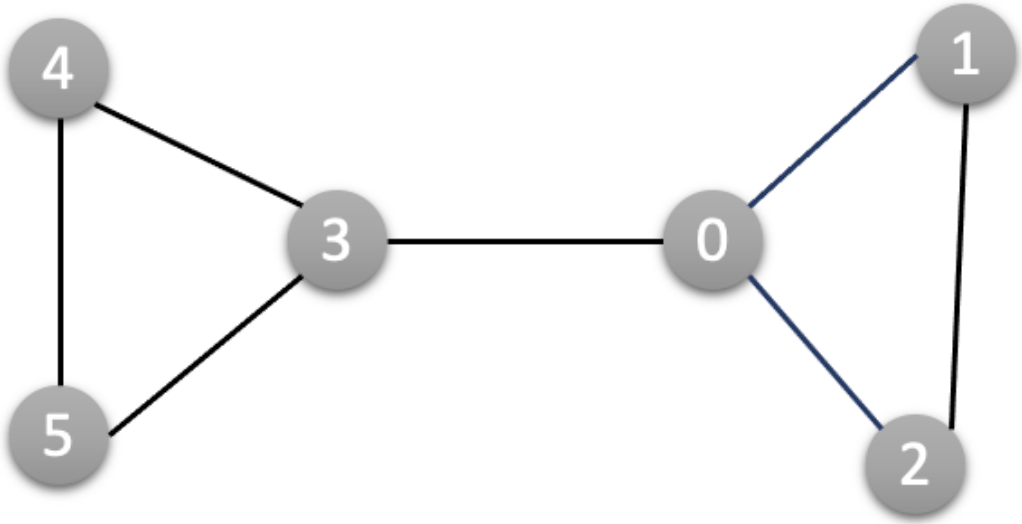
- P: normalized from the Adjacency Matrix A
- $P \in \mathbb{R}^{N \times N}$
- In the first layer: $H^{(0)} = X \in \mathbb{R}^{N \times D}$
- $W^{(0)} \in \mathbb{R}^{D \times d_0}$
- Ignore the bias term for simplicity in the following part

Example



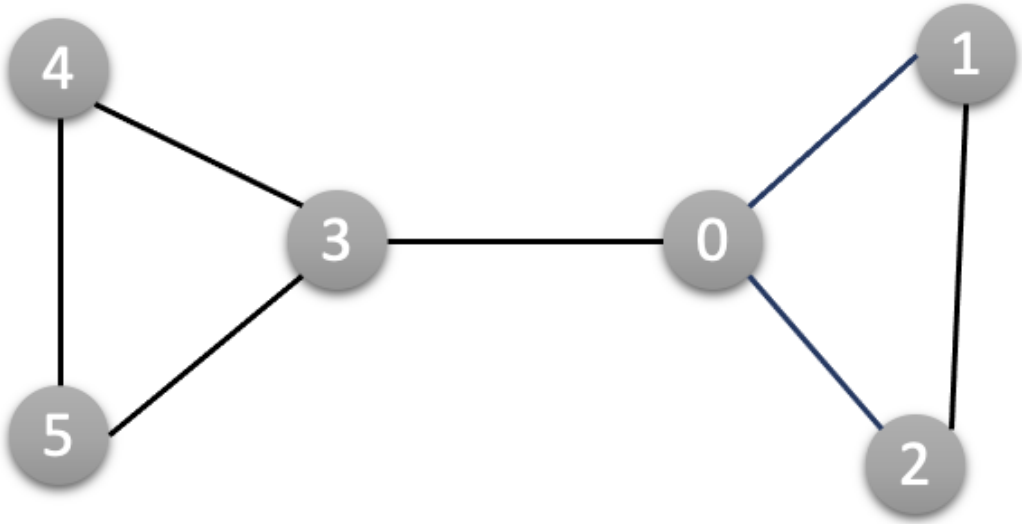
- Given a graph with the following adjacency matrix and node features
- $A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, X = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$

Example



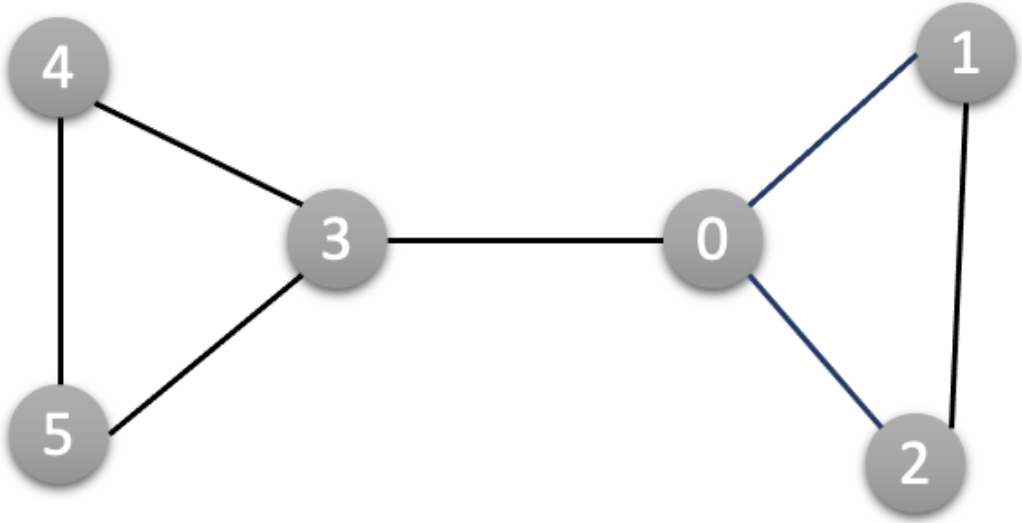
- What does AX do?
- $AX = \begin{pmatrix} 6 \\ 2 \\ 1 \\ 9 \\ 8 \\ 7 \end{pmatrix}$, problem?

Example: with self-loop



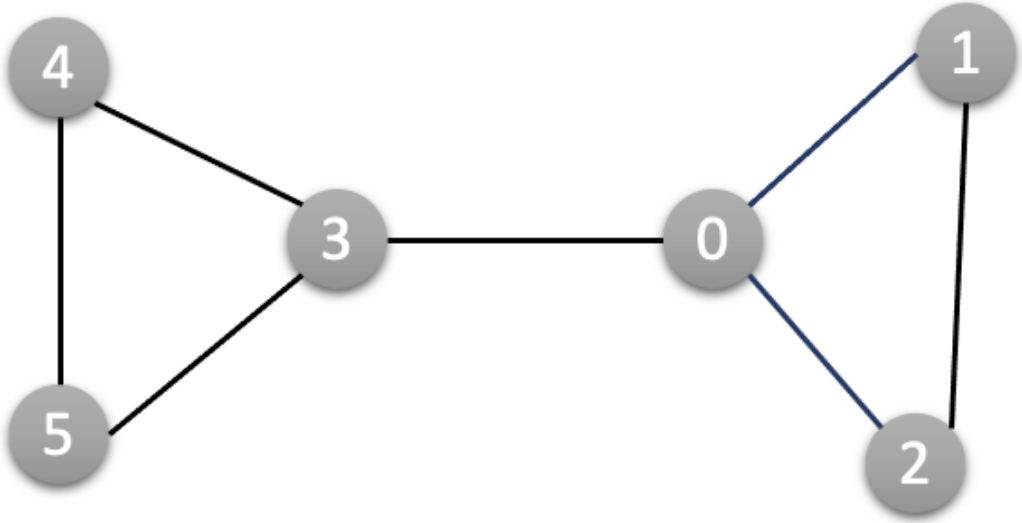
- How about $\tilde{A}X = (A + I)X$?
- $\tilde{A}X = \begin{pmatrix} 6 \\ 3 \\ 3 \\ 12 \\ 12 \\ 12 \end{pmatrix}$, problem?

Example: degree matrix



- Degree matrix: $\tilde{D}$
- Self-loop is counted as 2.
- $\tilde{D} = \text{diag}(5, 4, 4, 5, 4, 4)$
  $\tilde{D}^{-1} = \text{diag}(0.2, 0.25, 0.25, 0.2, 0.25, 0.25)$

Example: degree matrix



- Degree matrix: $\tilde{D}$
- Self-loop is counted as 2.
- $\tilde{D} = \text{diag}(5, 4, 4, 5, 4, 4)$
  $\tilde{D}^{-1} = \text{diag}(0.2, 0.25, 0.25, 0.2, 0.25, 0.25)$
- How about $\tilde{D}^{-1}\tilde{A}X$?
- $\tilde{D}^{-1}\tilde{A}X = (1.2, 0.75, 0.75, 2.4, 3, 3)^T$
- $P = \tilde{D}^{-1}\tilde{A}$

Graph Convolution Layer

- GCN: multiple graph convolution layers
- P: normalized version of A:

$$\tilde{A} = A + I, \quad P = \tilde{D}^{-1}\tilde{A}$$

- Graph convolution:
  - Input: features for each node $H^{(l)} \in \mathbb{R}^{N \times D}$
  - Output: features for each node $H^{(l+1)}$ after gathering neighborhood information
  - Convolution: $PH^{(l)}$: Aggregate features from neighbors

$$H^{(l+1)} = \sigma(PH^{(l)}W^{(l)}),$$

$W^{(l)}$ is the weights for the linear layer
$\sigma(\cdot)$: usually ReLU function

Graph convolutional network

- Initial features $H^{(0)} := X$
- For layer $l = 0, \ldots, L$

$$Z^{(l+1)} = PH^{(l)}W^{(l)}, \quad H^{(l+1)} = \sigma(Z^{(l+1)}),$$

- Use final layer feature $H^{(L)} \in \mathbb{R}^{N \times K}$ for classification:

$$\text{Loss} = \frac{1}{|S|} \sum_{s \in S} \text{loss}(y_s, H_s^{(L)})$$

  - Each row of $Z_s^{(L)}$ corresponds to the output score for each label.
  - Cross-entropy loss for classification.

Graph convolutional network

- Model parameters: $W^{(1)}, \cdots, W^{(L)}$
- Can be used to
  - Predict unlabeled nodes in the training set
  - Predict labels for a new graph
- Also, features extracted by GCN $H^{(L)}$ is usually very useful for other tasks

GCN training

- Full Gradient descent in the original paper (Kipf & Welling, 2017):
  - Need many iterations (epochs)
  - Large memory requirement for storing all the intermediate embeddings
- GraphSAGE (NeurIPS'17)
- VRGCN (ICML'18)
- Cluster-GCN (KDD'19)

Conclusions

- Graph Basics
- Graph Neural Networks

Questions?