GNN Architectures

AI602: Recent Advances in Deep Learning

Lecture 7

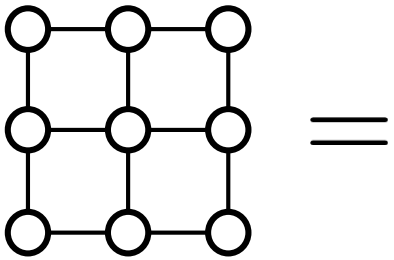
Slide made by

Insu Han and Sungsoo Ahn

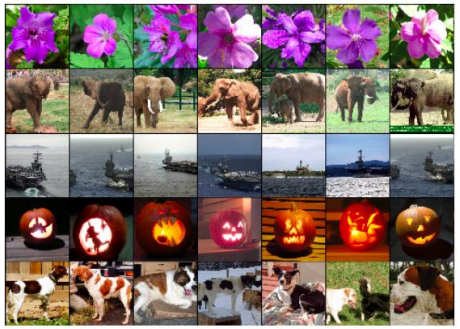
KAIST EE

Motivation

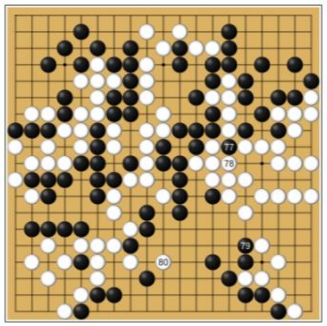
- Successful Deep Neural Networks (DNNs) have mainly focused on grid or sequence data, e.g., CNN and RNN.



Grid 2D

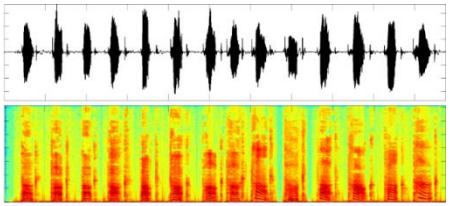


Image

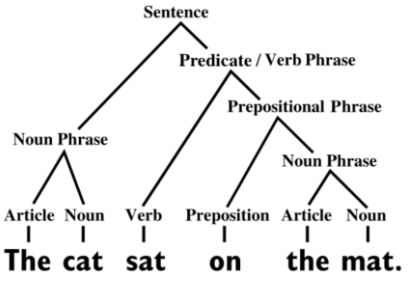


Game

Grid 1D (Sequence)



Speech



Language

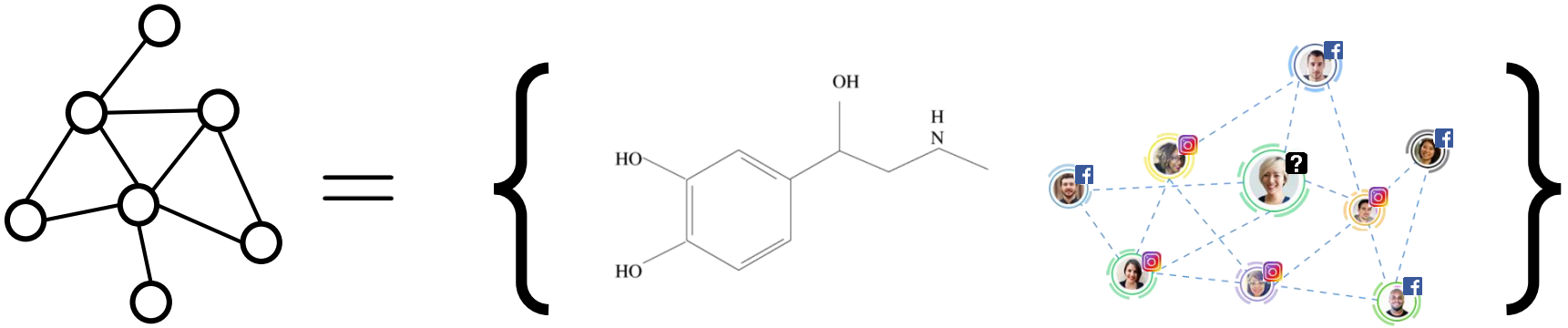
*reference: http://cs231n.stanford.edu/2017/

Motivation

- Successful Deep Neural Networks (DNNs) have mainly focused on grid or sequence data, e.g., CNN and RNN.

- Many practical data are often not on the grid (i.e., non-Euclidean space)
  - E.g., distance between two nodes are not defined

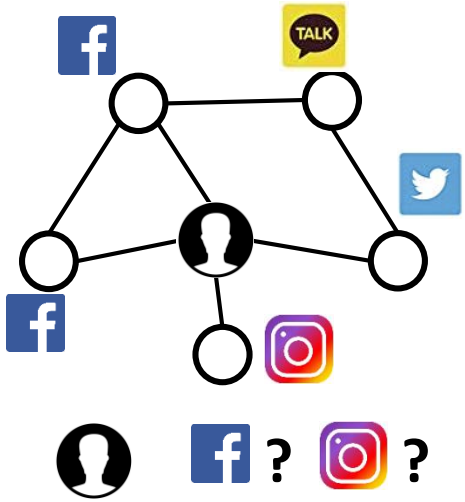
- These data (e.g., molecule and social networks) can be represented by graphs



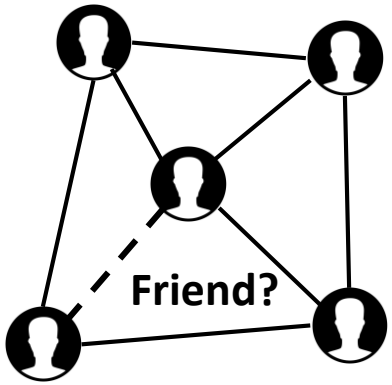
Graph *Molecule* *Social Network*

Motivation

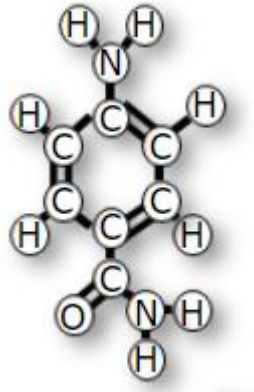
- Practical problems of graph data
  - Community detection → Node classification
  - Friend recommendation → Link prediction
  - Toxic detection → Graph classification



Node classification



Link prediction



Toxic ?

Graph classification

- How can solve graph problems using DNNs?

Next, Graph Neural Networks

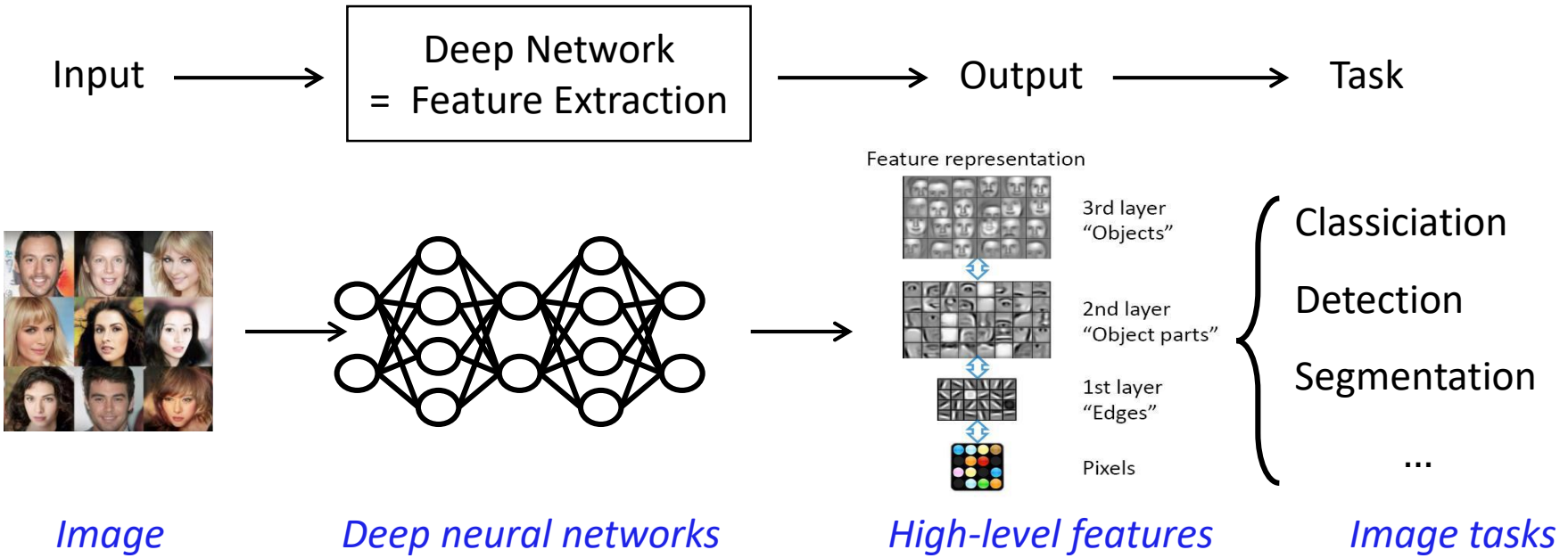
*source: https://slideplayer.com/slide/8271410/

Table of Contents

Table of Contents

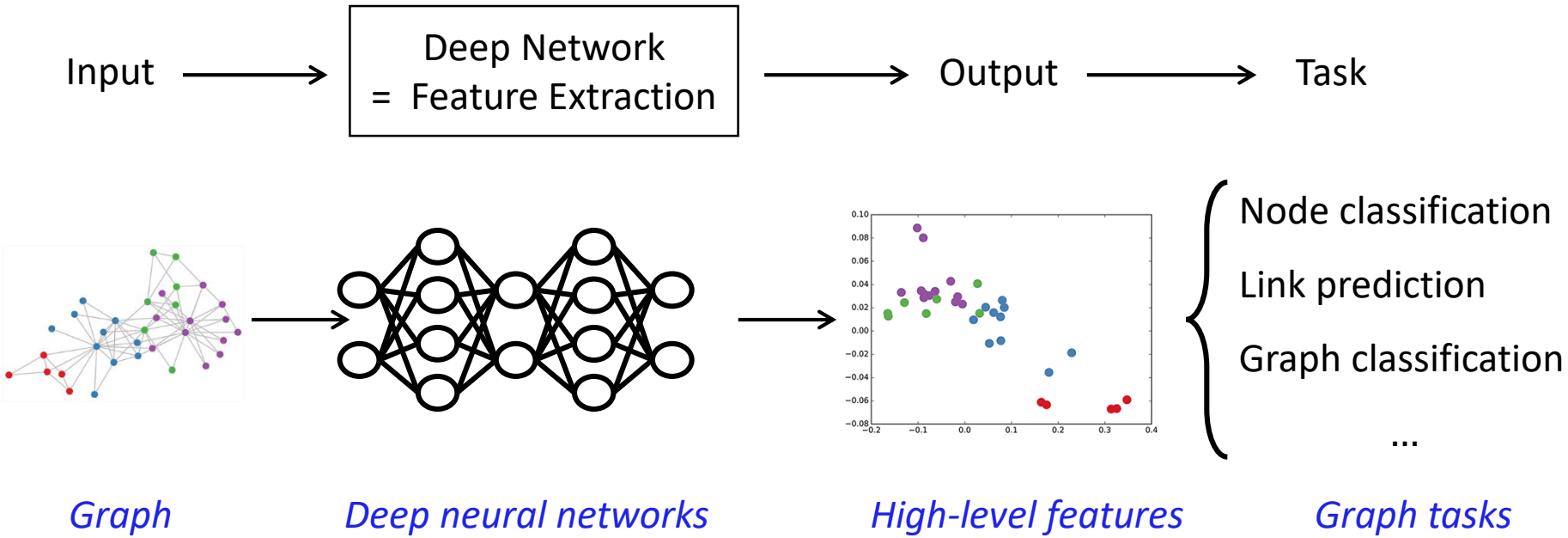
Recap: Deep Learning = Feature Learning

- Deep learning extracts features from the input data
  - For **image** data, deep networks finds high-level features that used for various image tasks, e.g., classification, object detection and segmentation.



Recap: Deep Learning = Feature Learning

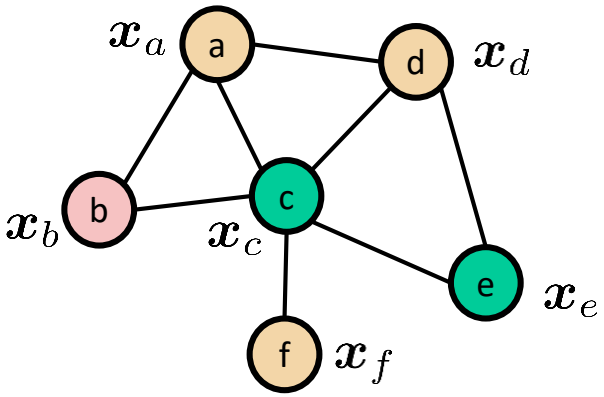
- Deep learning extracts features from the input data
  - For **image** data, deep networks finds high-level features that used for various image tasks, e.g., classification, object detection and segmentation.
  - For **graph** data, we want to find good features for graph tasks, e.g., node classification, link prediction and graph classification.



- How can DNNs extract features on graph dataset?

Graph Neural Networks (GNNs): Basics

- Consider (undirected) graph $G := (V, E)$ given by adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$
- Each node feature is given by $X \in \mathbb{R}^{|V| \times d}$
  - For example, node feature can include the number of neighbors of each node



$$V = \{a, b, c, d, e, f\}$$

$$E = \{\{a, b\}, \ldots, \{c, f\}\}$$

1 if node a connects to node b, 0 otherwise

$$X = \begin{matrix} x_a \\ \vdots \\ \\ \\ \\ x_f \end{matrix} \begin{pmatrix} 3 & 0.4 \\ 2 & 0.8 \\ 5 & 0.2 \\ 3 & 2.1 \\ 2 & 0.9 \\ 1 & 1.7 \end{pmatrix}$$

Degree

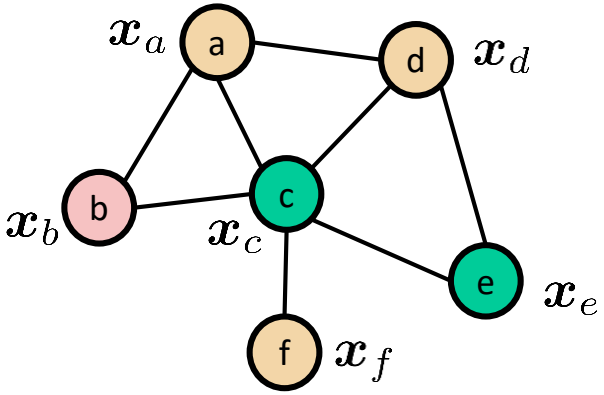
$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Adjacency matrix

Graph Neural Networks (GNNs): Basics

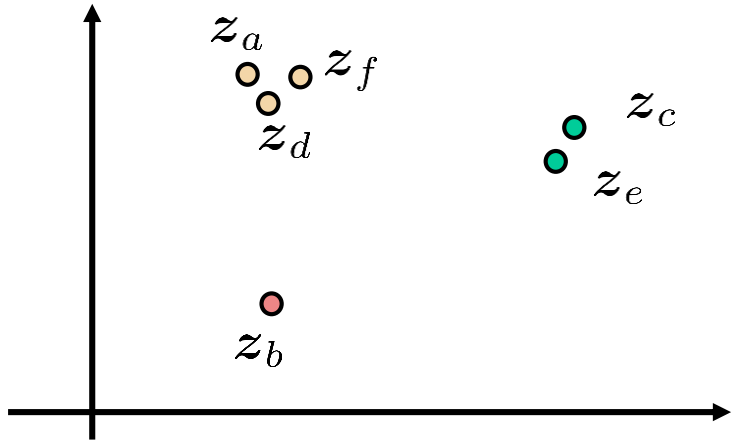
- Consider (undirected) graph $G := (V, E)$ given by adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$
- Each node feature is given by $X \in \mathbb{R}^{|V| \times d}$
  - For example, node feature can include the number of neighbors of each node
- **Graph Neural Networks (GNNs)** can extract graph node features

A, X → Graph Neural Networks → $Z \in \mathbb{R}^{|V| \times d'}$



$$V = \{a, b, c, d, e, f\}$$

$$E = \{\{a, b\}, \ldots, \{c, f\}\}$$

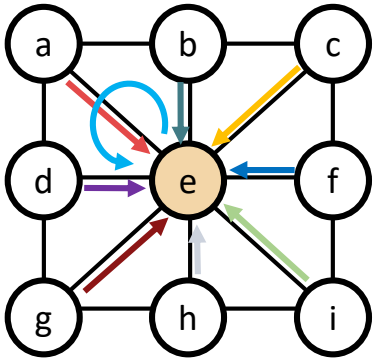


A good embedding for node classification

Next, Convolution on Graph

Graph Neural Networks (GNNs): Basics

- Consider **convolution** on image data
  - Weighted sum of features of neighbor nodes
  - The feature is multiplied with individual parameter then aggregated to next layer
  - The parameters are shared for all pixels



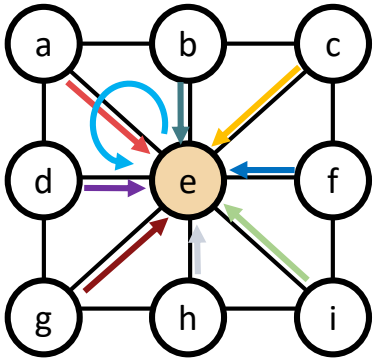
non-linear function

$$\boldsymbol{x}_e^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_a^{(\ell)}\boldsymbol{x}_a^{(\ell)} + \cdots + \boldsymbol{\theta}_i^{(\ell)}\boldsymbol{x}_i^{(\ell)}\right)$$

individual weighted sum

Graph Neural Networks (GNNs): Basics

- Consider **convolution** on image data
  - Weighted sum of features of neighbor nodes
  - The feature is multiplied with individual parameter then aggregated to next layer
  - The parameters are shared for all pixels

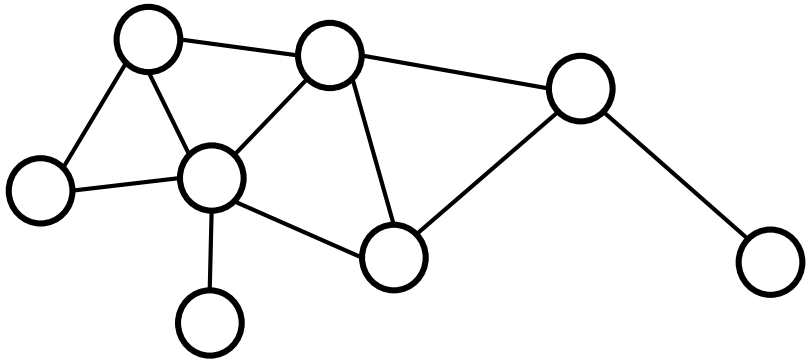


non-linear function

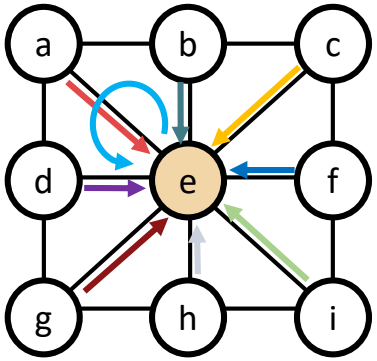
$$\boldsymbol{x}_e^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_a^{(\ell)}\boldsymbol{x}_a^{(\ell)} + \cdots + \boldsymbol{\theta}_i^{(\ell)}\boldsymbol{x}_i^{(\ell)}\right)$$

individual weighted sum

- In graph, each node has different number of neighbor nodes
- No fixed order of neighbor nodes



- Consider **convolution** on image data
  - Weighted sum of features of neighbor nodes
  - The feature is multiplied with individual parameter then aggregated to next layer
  - The parameters are shared for all pixels

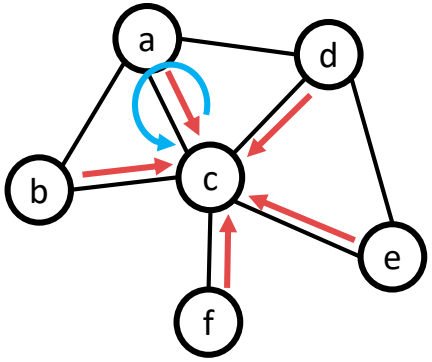


non-linear function

$$\boldsymbol{x}_e^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_a^{(\ell)}\boldsymbol{x}_a^{(\ell)} + \cdots + \boldsymbol{\theta}_i^{(\ell)}\boldsymbol{x}_i^{(\ell)}\right)$$

individual weighted sum

- **Basic approach in GNNs** – share the neighbor weights



weight for self node

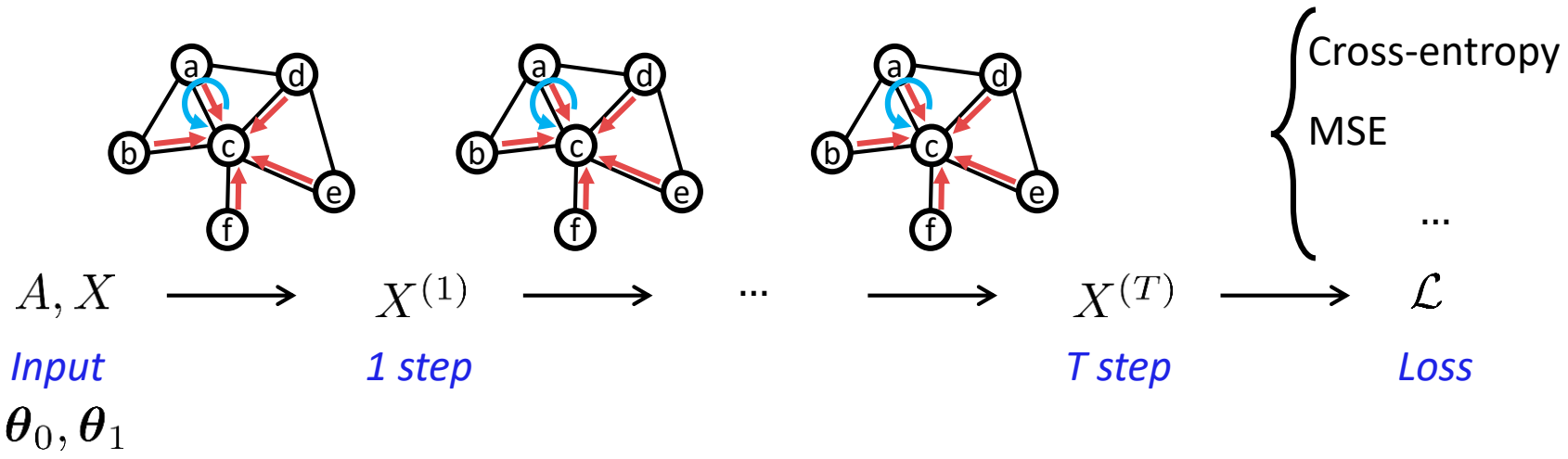
$$\boldsymbol{x}_c^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_0^{(\ell)}\boldsymbol{x}_c^{(\ell)} + \boldsymbol{\theta}_1^{(\ell)}\frac{1}{|\mathcal{N}_c|}\sum_{j\in\mathcal{N}_c}\boldsymbol{x}_j^{(\ell)}\right)$$

weight for neighbors

e.g., non-linear function,
long short-term memory (LSTM)

- [Scarselli et al., 2008] propose a GNN with a single layer (similar to RNN)
- Node embedding (feed-forward)
  - Node features is computed iteratively until converge (or maximum T steps)

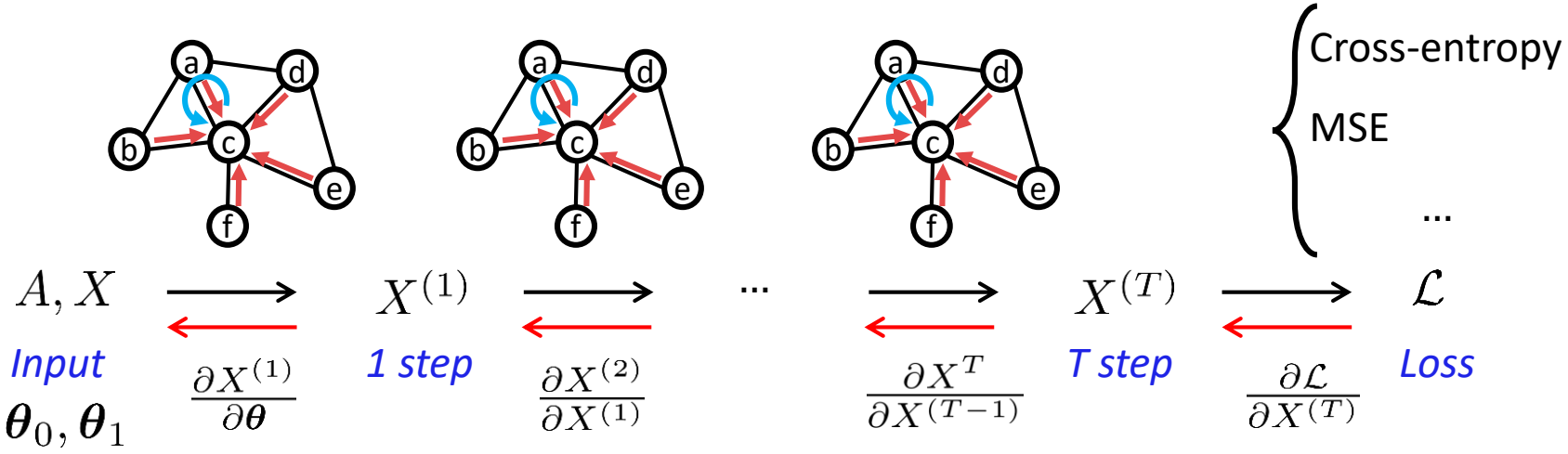
$$\boldsymbol{x}_c^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_0 \boldsymbol{x}_c^{(\ell)} + \boldsymbol{\theta}_1 \frac{1}{|\mathcal{N}_c|} \sum_{j \in \mathcal{N}_c} \boldsymbol{x}_j^{(\ell)}\right)$$



- [Scarselli et al., 2008] propose a GNN with a single layer (similar to RNN)
- Node embedding (feed-forward)
  - Node features is computed iteratively until converge (or maximum T steps)

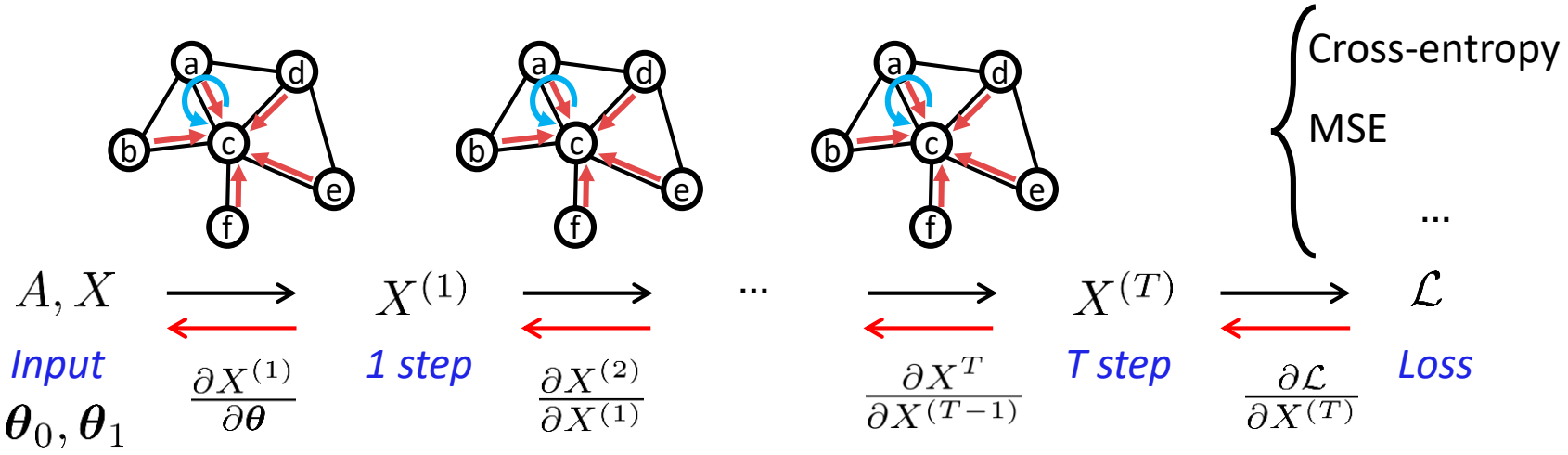
$$\boldsymbol{x}_c^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}_0 \boldsymbol{x}_c^{(\ell)} + \boldsymbol{\theta}_1 \frac{1}{|\mathcal{N}_c|} \sum_{j \in \mathcal{N}_c} \boldsymbol{x}_j^{(\ell)}\right)$$

- Parameter learning (back-propagation)
  - Gradients are also back-propagated iteratively until converge (or maximum T steps)



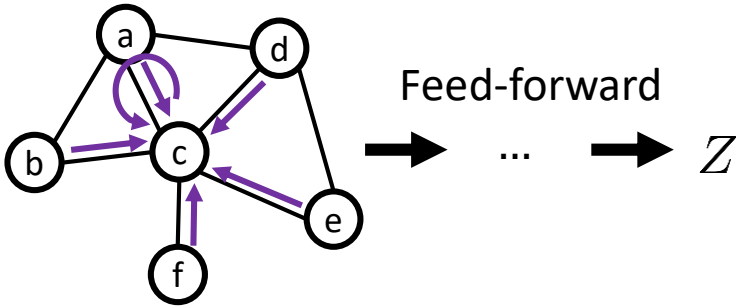
Graph Neural Networks (GNNs): Basics

- [Scarselli et al., 2008] propose a GNN with a single layer (similar to RNN)
- Forward/backward are computed iteratively until converge
- (-) In general, the states (node features) may not converge
- (-) The iteration makes computationally expensive, not scalable
- How can make GNNs computationally efficient?



Next, Graph Convolutional Networks

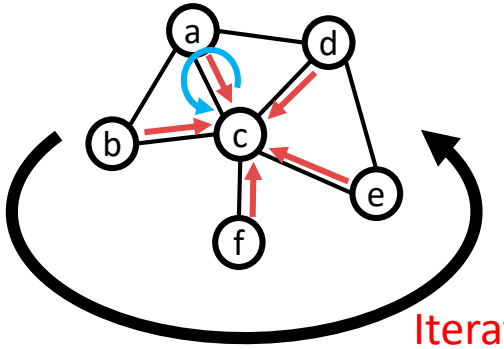
- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
  - It performs fixed number of feed-forwards (e.g., 2-layer convolution)
  - Parameters for self and neighbor are tied (shared) as a single parameter
  - Normalized the features with the products of degree,



$$x_c^{(\ell+1)} = \sigma\left(\theta^{(\ell)} \sum_{j \in \mathcal{N}_c \cup \{c\}} \frac{x_j^{(\ell)}}{\sqrt{|\mathcal{N}_c||\mathcal{N}_j|}}\right)$$

shared weight

Recall Graph Neural Networks is of RNN type



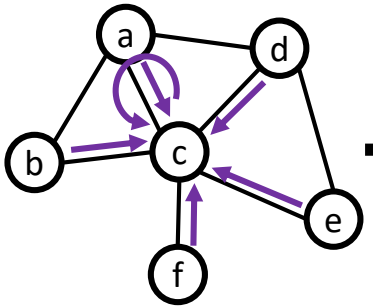
Iteratively updates

weight for self node

$$x_c^{(\ell+1)} = \sigma\left(\theta_0^{(\ell)} x_c^{(\ell)} + \theta_1^{(\ell)} \frac{1}{|\mathcal{N}_c|} \sum_{j \in \mathcal{N}_c} x_j^{(\ell)}\right)$$

weight for neighbors

- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
  - It performs fixed number of feed-forwards (e.g., 2-layer convolution)
  - Parameters for self and neighbor are tied (shared) as a single parameter
  - Normalized the features with the products of degree,



Feed-forward → ... → Z

$$x_c^{(\ell+1)} = \sigma\left(\theta^{(\ell)} \sum_{j \in \mathcal{N}_c \cup \{c\}} \frac{x_j^{(\ell)}}{\sqrt{|\mathcal{N}_c||\mathcal{N}_j|}}\right)$$

shared weight

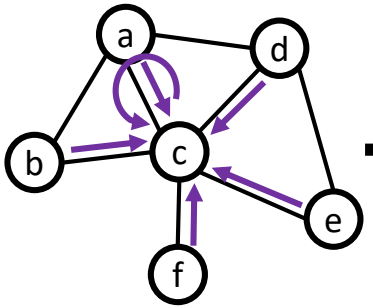
  - (+) Number of parameters is reduced with more parameter sharing
  - (+) No need to recursive operations (easy to optimize)
  - (+) Easy to implement using matrix multiplication, requiring $\mathcal{O}(|E|)$ operations

diagonal matrix

$$X^{(\ell+1)} = \sigma\left(D^{-1/2}(A+I)D^{-1/2}X^{(\ell)}\Theta^{(\ell)}\right)$$

sparse matrix, $\mathcal{O}(|E|+|V|)$ non-zero entries

- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
  - It performs fixed number of feed-forwards (e.g., 2-layer convolution)
  - Parameters for self and neighbor are tied (shared) as a single parameter
  - Normalized the features with the products of degree,



Feed-forward → ... → Z

$$x_c^{(\ell+1)} = \sigma\left(\theta^{(\ell)} \sum_{j \in \mathcal{N}_c \cup \{c\}} \frac{x_j^{(\ell)}}{\sqrt{|\mathcal{N}_c||\mathcal{N}_j|}}\right)$$

shared weight

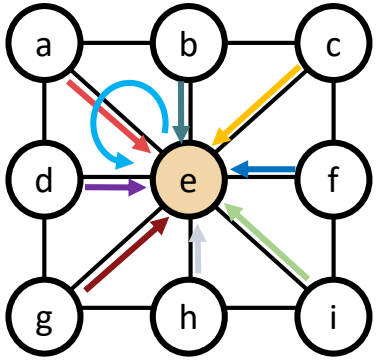
- (+) Number of parameters is reduced with more parameter sharing
- (+) No need to recursive operations (easy to optimize)
- (+) Easy to implement using matrix multiplication, requiring $\mathcal{O}(|E|)$ operations
- (+) **Residual connections** can improve the performance

$$X^{(\ell+1)} = \sigma\left(D^{-1/2}(A+I)D^{-1/2}X^{(\ell)}\Theta^{(\ell)}\right) + X^{(\ell)}$$

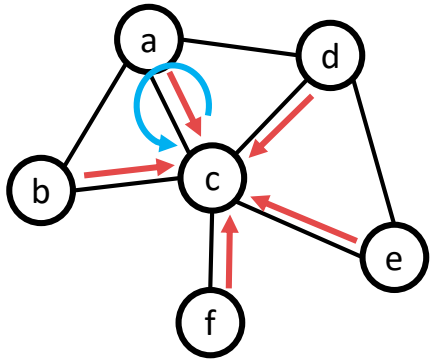
residual connection

Graph Convolutional Networks (GCNs): Basics

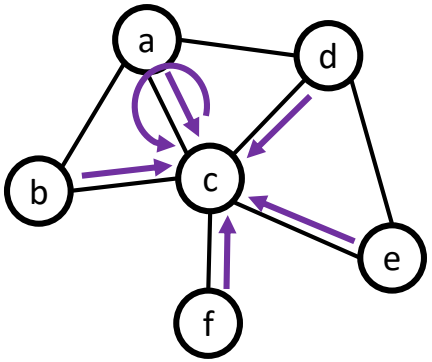
- Overview of CNNs, GNNs and GCNs
  - CNNs have individual parameters for all neighbors and self nodes
  - GNNs have 2 type of parameters; only parameters for neighbor nodes are shared
  - GCNs have a single parameter shared for both self and neighbor nodes



CNNs



GNNs



GCNs

Graph Convolutional Networks (GCNs): Basics

- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
- Goal – classify the paper category (e.g., stat.ML, cs.LG, and etc) for citation networks (graph node classification)
- Model – 2 layer GCN

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

$$Z = \text{softmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}X\Theta^{(0)}\right)\Theta^{(1)}\right)$$

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Test accuracy (training time) for graph node classification

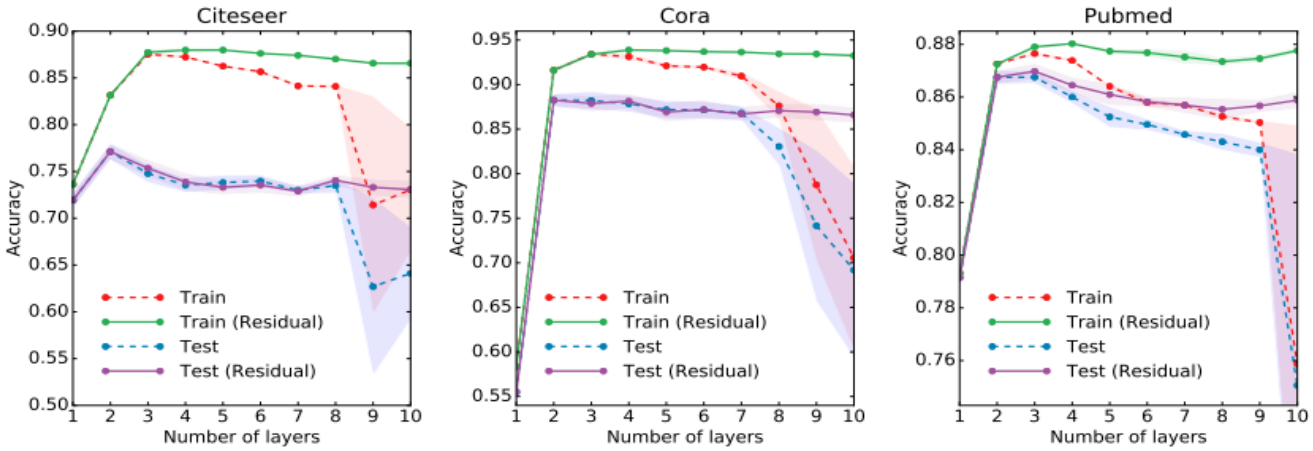
Graph Convolutional Networks (GCNs): Basics

- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
- Goal – classify the paper category (e.g., stat.ML, cs.LG, and etc) for citation networks (graph node classification)
- Model – 2 layer GCN

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

$$Z = \text{softmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}X\Theta^{(0)}\right)\Theta^{(1)}\right)$$

- Why 2 layer is enough?
  - Most of graph dataset can be covered with 4 hop-neighbors
  - Large number of layers can cause over-smoothing on the node features



Graph Convolutional Networks (GCNs): Basics

- **Graph Convolutional Networks** (GCNs) [Kipf et al., 2016a]
  - (+) Number of parameters is reduced with more parameter sharing
  - (+) No need to recursive operations (easy to optimize)
  - (+) Easy to implement using matrix multiplication, requiring $\mathcal{O}(|E|)$ operations
  - (+) Empirically works well even for a small number of layers

- Some limitations on GCNs
  - (-) Can be overfit easily
  - (-) Still expensive for large-scale graphs
  - (-) Cannot embedding edge features

- How can overcome these drawbacks?

Next, GCNs with Neighbor Sampling

Table of Contents

Sampling Neighbors for Regularization

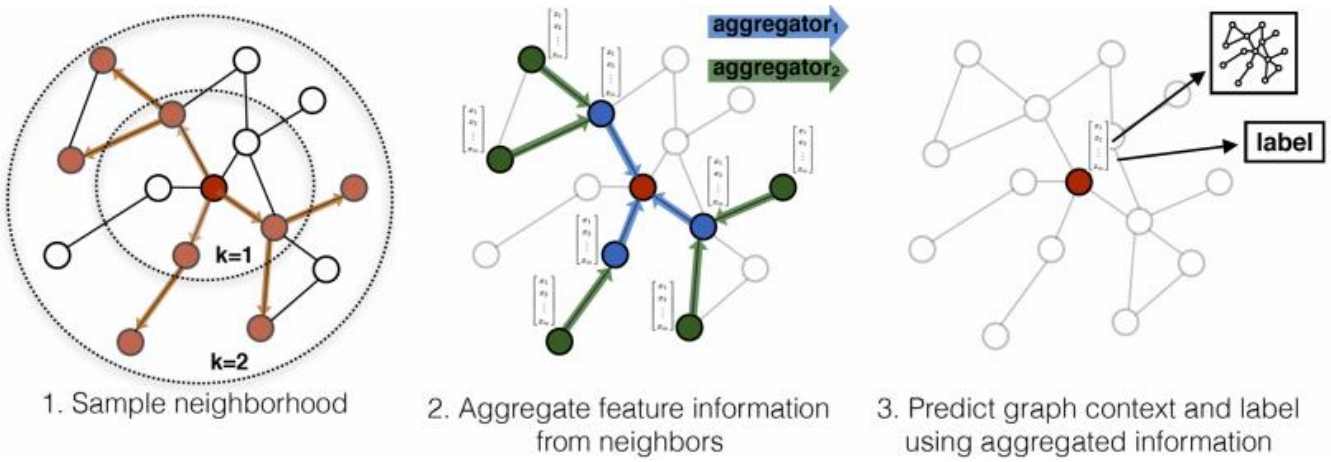
- **Problem** – GCNs can be overfit easily
- **Solution** – Train the model stochastically based on sampling neighbors

Sampling Neighbors for Regularization

- **Problem** – GCNs can be overfit easily
- **Solution** – Train the model stochastically based on sampling neighbors
- **Graph Sampling and Aggregation** (GraphSAGE) [Hamilton et al., 2017]
- Sample neighbor nodes randomly
  - (+) Avoid overfitting
  - (+) Training time can be reduced

$$\text{Aggregate}\left(\left\{\boldsymbol{x}_j^{(\ell)}; j \in S_c, S_c \subseteq \mathcal{N}_c\right\}\right)$$

neighbor sampling



1. Sample neighborhood

2. Aggregate feature information from neighbors

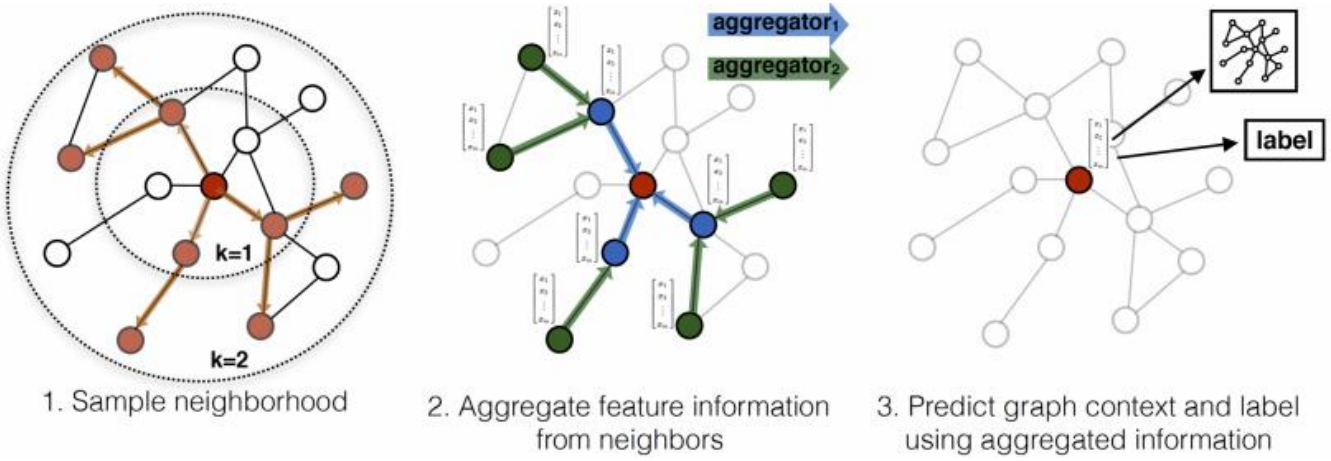
3. Predict graph context and label using aggregated information

Sampling Neighbors for Regularization

- **Problem** – GCNs can be overfit easily
- **Solution** – Train the model stochastically based on sampling neighbors
- **Graph Sampling and Aggregation** (GraphSAGE) [Hamilton et al., 2017]
- Sample neighbor nodes randomly
  - (+) Avoid overfitting
  - (+) Training time can be reduced
- Instead of summing the self and neighbor features, concatenate them

$$\boldsymbol{x}_c^{(\ell+1)} = \sigma\left(\left[\boldsymbol{\theta}_0^{(\ell)}\boldsymbol{x}_c^{(\ell)}\ ,\ \text{Aggregate}\left(\left\{\boldsymbol{x}_j^{(\ell)}; j \in S_c, S_c \subseteq \mathcal{N}_c\right\}\right)\right]\right)$$

neighbor sampling



1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Sampling Neighbors for Regularization

- **Problem** – GCNs can be overfit easily
- **Solution** – Train the model stochastically based on sampling neighbors
- **Graph Sampling and Aggregation** (GraphSAGE) [Hamilton et al., 2017]
- Sample neighbor nodes randomly
  - (+) Avoid overfitting
  - (+) Training time can be reduced
- Instead of summing the self and neighbor features, concatenate them

$$\boldsymbol{x}_c^{(\ell+1)} = \sigma\left(\left[\boldsymbol{\theta}_0^{(\ell)}\boldsymbol{x}_c^{(\ell)}\ ,\ \text{Aggregate}\left(\left\{\boldsymbol{x}_j^{(\ell)}; j \in S_c, S_c \subseteq \mathcal{N}_c\right\}\right)\right]\right)$$

neighbor sampling

- Aggregation function can be generalized including

Mean	$\frac{1}{\lvert\mathcal{N}_c\rvert}\sum_{j\in\mathcal{N}_c}\boldsymbol{x}_j^{(\ell)}$	
Pool	$\max\left(\left\{W^{(\ell)}\boldsymbol{x}_j^{(\ell)}; j \in \mathcal{N}_c\right\}\right)$	$W^{(\ell)}$: learnable parameter
LSTM	$\text{LSTM}\left(\left[\boldsymbol{x}_j^{(\ell)}; j \in \pi(\mathcal{N}_c)\right]\right)$	π : random permutation

Sampling Neighbors for Regularization

- **Problem** – GCNs can be overfit easily
- **Solution** – Train the model stochastically based on sampling neighbors
- **Graph Sampling and Aggregation** (GraphSAGE) [Hamilton et al., 2017]
- Aggregation function can be {mean, pool, LSTM}
- Experiments for node classification
  - Performs better than GCNs

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Next, GCNs with Edge Attention

Learning Attentions of Neighbors

- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights

$$\alpha_{ij} = \mathbf{f}\left(\boldsymbol{x}_i, \{\boldsymbol{x}_j : j \in \mathcal{N}_i\} \right)$$

$$\boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}^{(\ell)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{x}_j^{(\ell)} \right)$$

edge weight

Learning Attentions of Neighbors

- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights

$$\alpha_{ij} = \mathbf{f}\left(\boldsymbol{x}_i, \{\boldsymbol{x}_j : j \in \mathcal{N}_i\} \right) \qquad \boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}^{(\ell)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{x}_j^{(\ell)} \right)$$

edge weight

  - GCNs have constant edge weights, i.e.,

$$\alpha_{ij}^{(\mathrm{GCN})} = \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}$$

  - GraphSAGE has binary random edge weights, i.e.,

$$\alpha_{ij}^{(\mathrm{SAGE})} = \begin{cases} 1 & \text{w.p.} \quad \frac{1}{|\mathcal{N}_i|} \\ 0 & \text{otherwise} \end{cases}$$

- Edge weights can be a trainable neural network

Learning Attentions of Neighbors

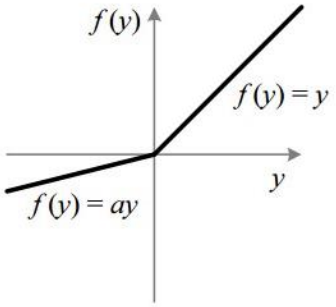
- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights
- **Graph Attention Networks** (GAT) [Veličković et al., 2017]

$$\boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}^{(\ell)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{x}_j^{(\ell)}\right)$$

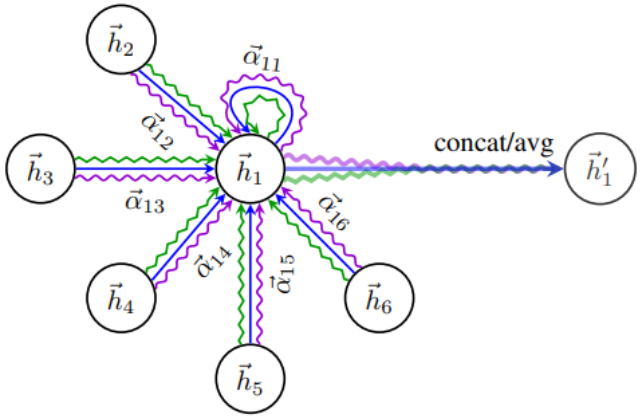
edge weight

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_j])\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_k])\right)\right)}$$

single layer neural network



LeakyReLU activation



Learning Attentions of Neighbors

- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights
- **Graph Attention Networks** (GAT) [Veličković et al., 2017]

$$\boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}^{(\ell)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{x}_j^{(\ell)}\right)$$

edge weight

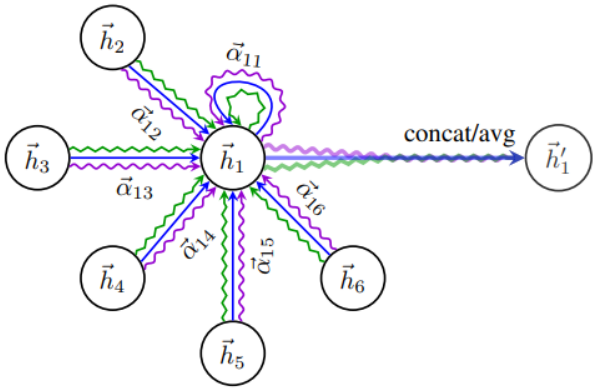
$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_j])\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_k])\right)\right)}$$

single layer neural network

- Multiple attentions can be used with averaging or concatenating

$$\boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\left(\boldsymbol{\theta}^{(\ell,K)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^{(k)} \boldsymbol{x}_j^{(\ell)}\right)\right)$$

concatenate/average of K independent attentions



Learning Attentions of Neighbors

- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights
- **Graph Attention Networks** (GAT) [Veličković et al., 2017]

$$\boldsymbol{x}_i^{(\ell+1)} = \sigma\left(\boldsymbol{\theta}^{(\ell)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{x}_j^{(\ell)}\right)$$

edge weight

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_j])\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\text{NN}([\boldsymbol{x}_i, \boldsymbol{x}_k])\right)\right)}$$

Single layer neural network

- Multiple attentions can be used with averaging or concatenating
- (+) Edge features can be learned
- (+) Generalize the neighbor sampling (soft version)
- (-) Traning is slower than GCN and difficult to optimize

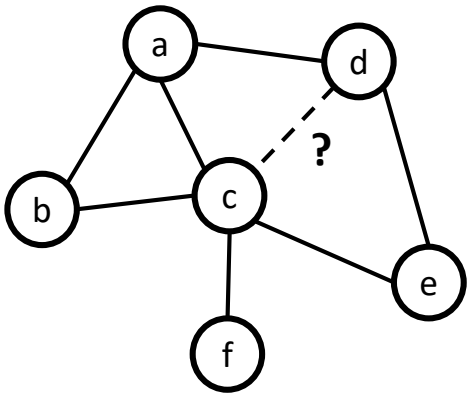
Learning Attentions of Neighbors

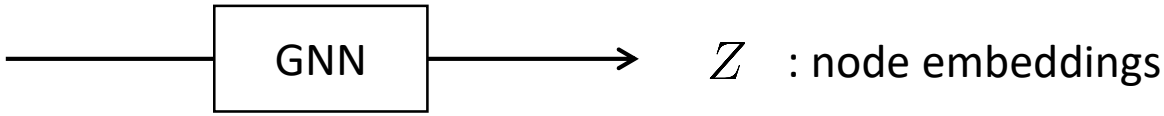
- **Problem** – GCNs use the fixed weight (or importance) of the edges
- **Solution** – Use the learnable edge attention weights
- **Graph Attention Networks** (GAT) [Veličković et al., 2017]
- Experiments for node classification
  - Performs better than GCNs

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

Learning Edge Features

- GNNs basically learns a good node embeddings
- How to use the node embeddings of GNNs for other tasks?
- **Link (or edge) prediction**
  - Edge feature can be the dot-product of adjacent node features [Kipf et al., 2016b]
  - Loss function can be the cross-entropy for binary classification





Z : node embeddings

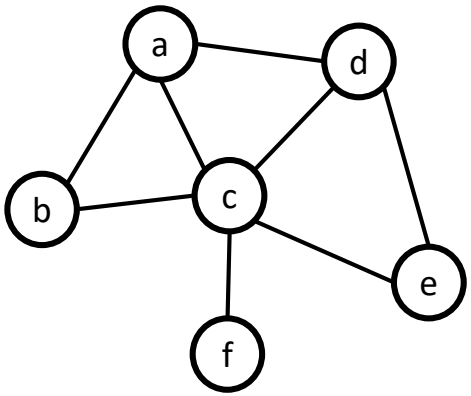
$G = (V, E), \quad X \in \mathbb{R}^{|V| \times d}$

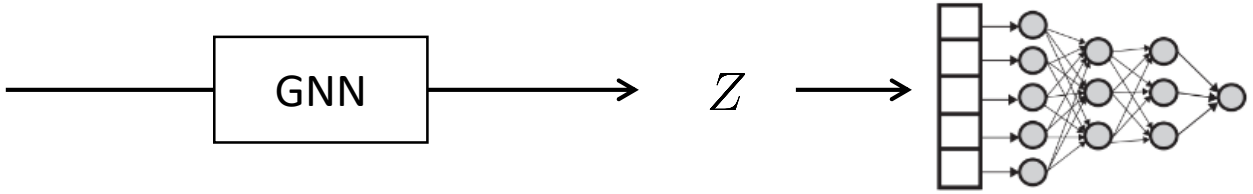
Edge score : $\hat{y}_{(i,j)} := \text{sigmoid}(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$

Loss function: $\mathcal{L} := \sum_{(i,j) \in E} -\log\left(\widehat{y}_{(i,j)}\right)$

Learning Graph Features

- GNNs basically learns a good node embeddings
- How to use the node embeddings of GNNs for other tasks?
- **Graph-level classification**
  - Graph features can be the summation or max pooling of all node features [Duvenaud et al., 2015]
  - A feed forward network can be used as a classifier
  - Graph pooling can be important to compress the graph feature





$G = (V, E), \quad X \in \mathbb{R}^{|V| \times d}$

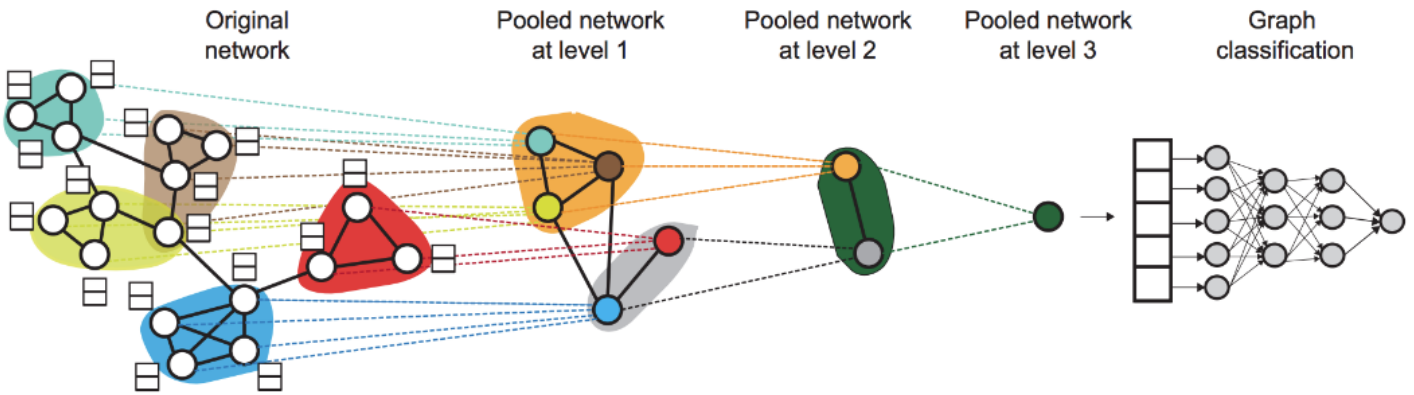
Graph feature : $\boldsymbol{z}_G := \sum_{i \in V} \boldsymbol{z}_i$ or $\max\left(\{\boldsymbol{z}_i : i \in V\}\right)$

Loss function: $\mathcal{L} := \sum_{G} \text{cross-entropy}\left(\text{NN}(\boldsymbol{z}_G)\right)$

Feed-forward neural network

Learning Graph Features

- GNNs basically learns a good node embeddings
- How to use the node embeddings of GNNs for other tasks?
- **Graph-level classification**
  - Graph features can be the summation or max pooling of all node features [Duvenaud et al., 2015]
  - A feed forward network can be used as a classifier
  - Graph pooling can be important to compress the graph feature
- How to graph pooling ?
  - Pooling methods can be learned with parameters [Ying et al., 2018]

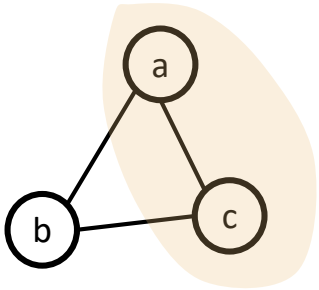


Next, Differentiable Pooling

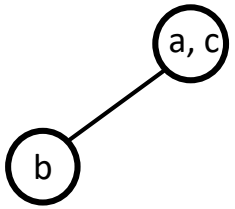
*source: https://arxiv.org/pdf/1806.08804.pdf

Learning Graph Features: Differentiable pooling

- Differentiable pooling [Ying et al., 2018] – graph pooling can be represented using matrix multiplication on adjacency matrix and node feature matrix



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



$$\hat{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\hat{A} = S^\top A S$$

Assignment matrix

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

- Assignment matrix has the same dimension of node features
- **Idea** – learning the assignmet matrix using GNNs

$$S = \mathrm{softmax}\left(\mathrm{GNN}(A, X)\right)$$

- Soft pooling operation can be differentiable and shows improvement for graph classification

Table of Contents

Boolean Satisfiability Problem

- Boolean satisfiability (SAT) problem determines the existence of interpretation satisfying the Boolean formula:

Literal

Clause

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$

Example of boolean formula

  - The example formula is satisfiable by following interpretation:

$$x_1 = \text{True}, \quad x_2 = \text{False}, \quad x_3 = \text{True}$$

Example of satisfying interpretation

  - It is very hard to solve in general, i.e., NP-hard
  - Various applications, e.g., circuit design and automatic theorem proving

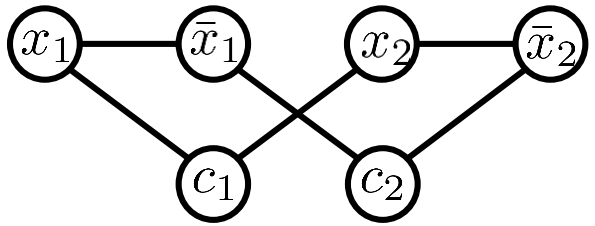
- A GNN can be trained to determine the existence [Selsam et al., 2018]

Boolean Satisfiability Problem

- **Step 1**: Represent the Boolean formula as a graph
  - Nodes represent literals and clauses in the formula
  - Edges are added for each literal-clause pair
  - Edges are added for literals with opposite signs

$$\overbrace{(x_1 \vee x_2)}^{c_1} \wedge \overbrace{(\bar{x}_1 \vee \bar{x}_2)}^{c_2}$$

4 literals and 2 clauses

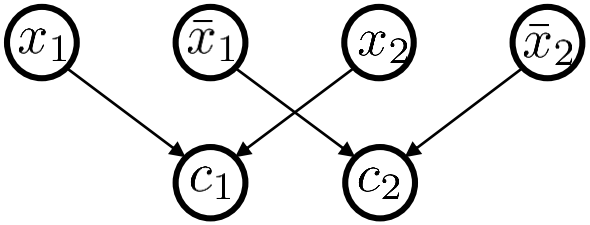


Graph representation of formula

- **Step 2**: Train the GNN to predict satisfiability of formula (True / False)
  - This becomes a binary graph classification problem
  - Labels of graphs, i.e., true satisfiability, are obtained by existing solvers

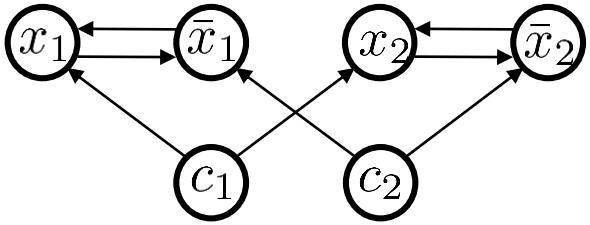
Boolean Satisfiability Problem

- **More details:** the GNN consists of two phases



$$c_1 \leftarrow \mathrm{LSTM}(x_1, x_2)$$

"clause-update" phase



$$x_1, x_2 \leftarrow \mathrm{LSTM}^{(1)}(c_1), \mathrm{LSTM}^{(2)}(c_1)$$

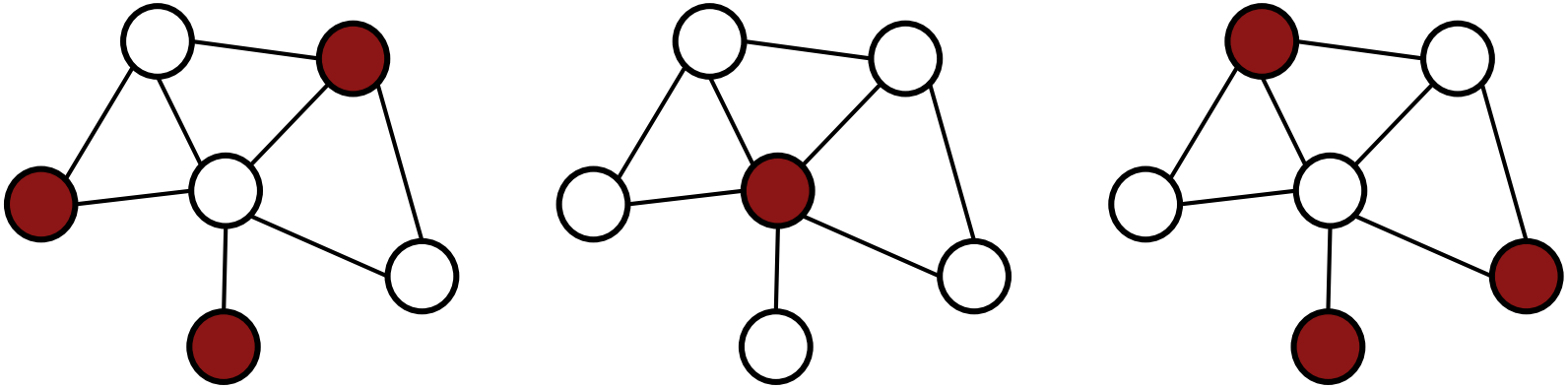
"literal-update" phase

  - Long short-term memory (LSTM) can be used for embedding features
  - Two phases (or layers) are repeated multiple times

- **Result:** the trained GNN is better than random guessing
  - Trained on SAT problems with up to 40 variables
  - Achieves 85% test accuracy (existing solvers get 100% correct)

Maximum Independent Set Problem

- Independent set is a set of nodes, where no two nodes are adjacent



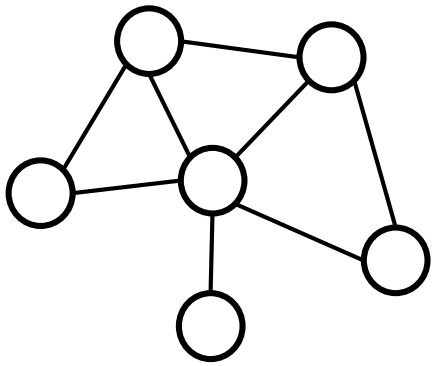
Examples of independent set (with size 3, 1, 3)

- Maximum independent set (MIS) problem search for the independent set with maximum cardinality
  - Again, hard to solve, i.e., NP-hard
  - Applications to computer vision, communication, physics, ...

- A GCN can be trained to find the MIS [Li et al., 2018]

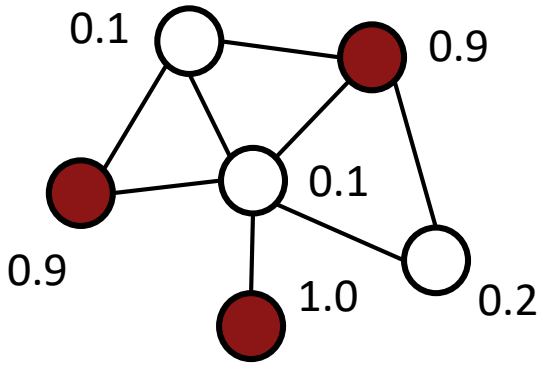
Maximum Independent Set Problem

- The naïve approach: train a GCN to predict the MIS
  - Becomes a node classification problem



What is prob. of node being in MIS?

GCN



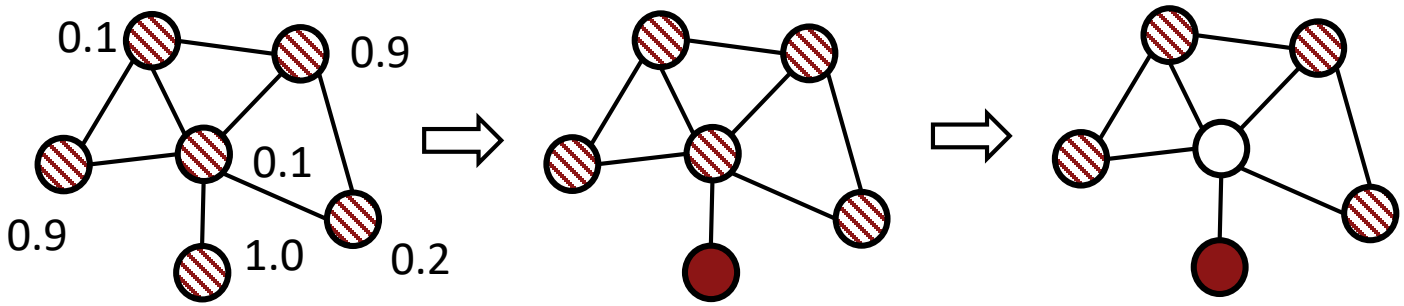
Label "1" means "included in the independent set"

$$\mathcal{L} := \sum_{G} \text{cross-entropy}\left(\text{GCN}(X_G)\right)$$ ← *Labels are given by existing solvers*

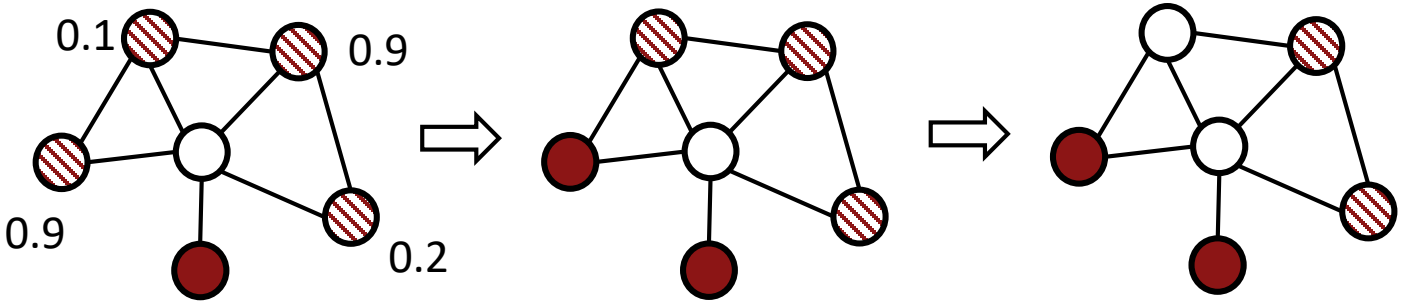
- **Problem:** at generation, the output sometimes have a pair of adjacent nodes
  - This violates the assumption of independent set

Maximum Independent Set Problem

- **Problem:** the output sometimes can have a pair of adjacent nodes
- **Solution**: label them one-by-one
- Repeatedly update the labels as follows:
  1. Assign **1** to the node with the highest probability of being chosen
  2. Assign **0** to the adjacent nodes



1st step of labeling



2nd step of labeling

Maximum Independent Set Problem

- Result: the trained GNN successfully solves the MIS problems

Name	MIS				MVC			
	Classic	S2V-DQN	ReduMIS	Ours	Classic	S2V-DQN	ReduMIS	Ours
ego-Facebook	993	1,020	**1,046**	**1,046**	3,046	3,019	**2,993**	**2,993**
ego-Gplus	56,866	56,603	**57,394**	**57,394**	50,748	51,011	**50,220**	**50,220**
ego-Twitter	36,235	36,275	**36,843**	**36,843**	45,071	45,031	**44,463**	**44,463**
soc-Epinions1	53,457	53,089	**53,599**	**53,599**	22,422	22,790	**22,280**	**22,280**
soc-Slashdot0811	53,009	52,719	**53,314**	**53,314**	24,351	24,641	**24,046**	**24,046**
soc-Slashdot0922	56,087	55,506	**56,398**	**56,398**	26,081	26,662	**25,770**	**25,770**
wiki-Vote	4,730	4,779	**4,866**	**4,866**	2,385	2,336	**2,249**	**2,249**
wiki-RfA	8,019	7,956	**8,131**	**8,131**	2,816	2,879	**2,704**	**2,704**
bitcoin-otc	4,330	4,334	**4,346**	**4,346**	1,551	1,547	**1,535**	**1,535**
bitcoin-alpha	2,703	2,705	**2,718**	**2,718**	1,080	1,078	**1,065**	**1,065**

Objective of MIS problems in large graphs (bold is best)

- The GNN solver performs well even when training and test graphs are different

Summary

- GNNs have similar operations with CNNs, but specialized for graph-structured dataset, i.e., neighbor aggregation
- GNNs basically extracts node features that suitable for graph tasks, e.g., node classification; From these node features, it can be used for link prediction and graph-level classicification
- GNNs are first proposed using iterative feed-forwards, after that GCNs are developed which are more efficient and perform better
- Several variants of GCNs, e.g., GraphSAGE, GAT, have been proposed to overcome drawbacks of GCNs
- GCNs practically are used for many graph problems including node classification, link prediction, graph classification and combinatorial optimization

References

[Scarselli et al., 2008] Scarselli, Franco, et al. "The graph neural network model." IEEE Transactions on Neural Networks. 2008.
link: https://ieeexplore.ieee.org/abstract/document/4700287

[Duvenaud et al., 2015] Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." Advances in neural information processing systems. 2015.
link: http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf

[Kipf et al., 2016a] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907. 2016.
link: https://openreview.net/pdf?id=SJU4ayYgl

[Kipf et al., 2016b] Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." arXiv preprint arXiv:1611.07308. 2016.
link: https://arxiv.org/pdf/1611.07308.pdf

[Hamilton et al., 2017] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in Neural Information Processing Systems. 2017.
link: https://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[Veličković et alk ., 2017] Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903. 2017.
link: https://openreview.net/forum?id=rJXMpikCZ

[Ying et al., 2018] Ying, Zhitao, et al. "Hierarchical graph representation learning with differentiable pooling." Advances in Neural Information Processing Systems. 2018.
link: http://papers.nips.cc/paper/7729-hierarchical-graph-representation-learning-with-differentiable-pooling.pdf

[Selsam et al., 2018] Selsam et al. “Learning a SAT solver from single-bit super-vision" International Conference of Learning Representations (2018).
link: https://openreview.net/pdf?id=HJMC_iA5tm

References

[Li et al., 2018] Li et al. "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search.“ Advances in Neural Information Porcessing Systems. 2018.
link: https://arxiv.org/pdf/1810.10659.pdf