Semi-Supervised Classification with Graph Convolutional Networks

2020.02.27.

Seohui Bae

shbae73@kaist.ac.kr

KAIST

MLILAB Machine Learning & Intelligence

Table of Contents

Introduction: Data

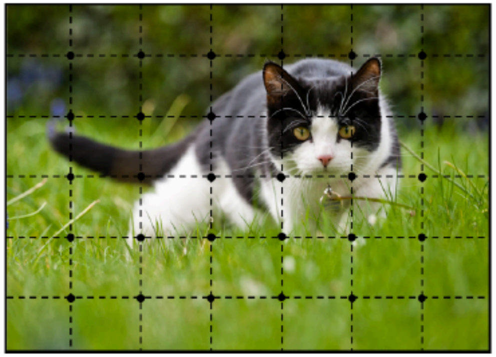
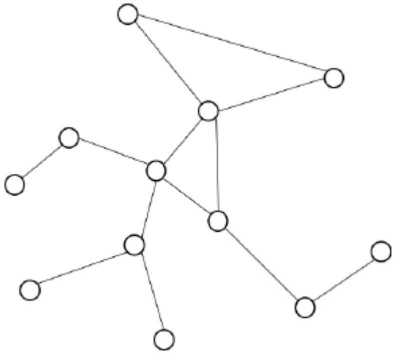


Image (Euclidean)

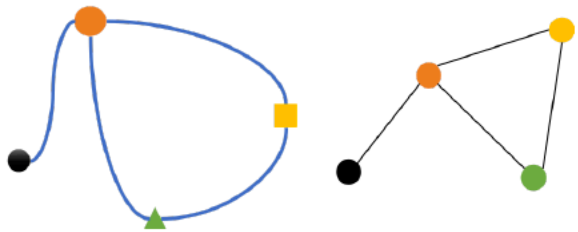


Graph (Non-Euclidean)

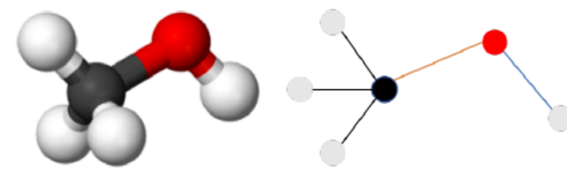
Image: Data on Grid

Graph: Data, Relation

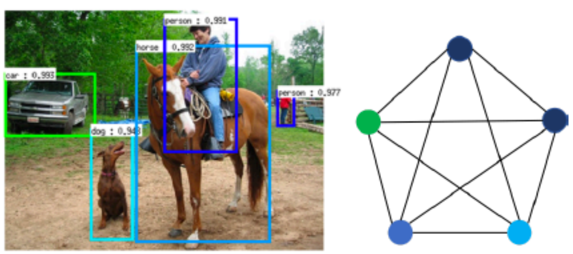
Introduction: Problems defined on Graph



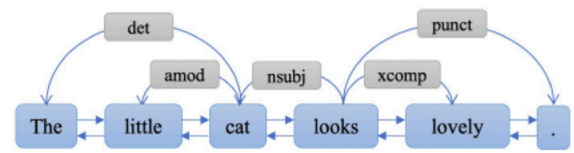
(a) physics



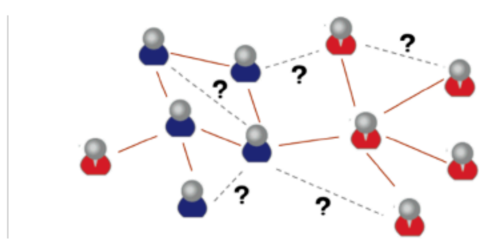
(b) molecule



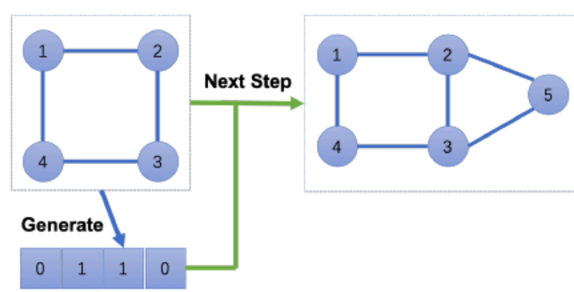
(c) image



(d) text



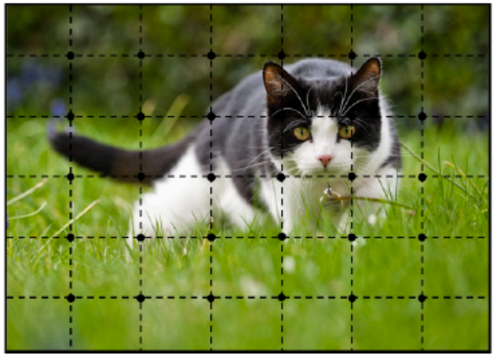
(e) social network



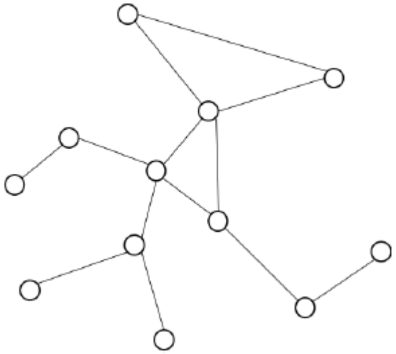
(f) generation

[1] Zhou et al., ArXiv 2018

Introduction: Deep Neural Network on Graph



Euclidean



Non-Euclidean

GNN are recurrent networks with vector valued nodes h_i whose states are iteratively updated by trainable nonlinear functions that depend on the states of neighbor nodes $h_j : j \in N_i$ on a specified graph

Introduction: Graph Convolutional Network

- GCN is one of GNNs!
- Convolution on Graph ?

Spectral Networks and Deep Locally Connected Networks on Graphs

Joan Bruna
New York University
bruna@cims.nyu.edu

Arthur Szlam
The City College of New
aszlam@ccny.cuny.

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Michaël Defferrard
E
{michael.defferrard,xa

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

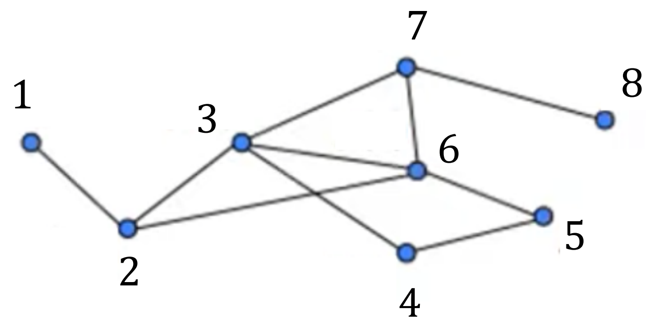
Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

Graph

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

Vertex V



Edge E

Data Feature
: Scalar or *C-dimension vector*

Scalar
(binary or $\mathbb{R}$*)*

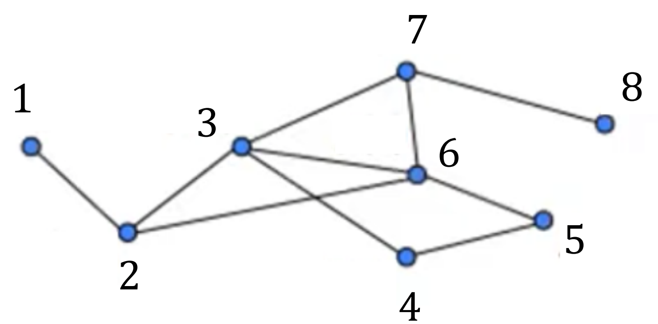
e.g. adjacency, degree matrix

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

$$D = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

Graph Laplacian

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



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

A

$$\begin{matrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{matrix}$$

L

A : Adjacency matrix

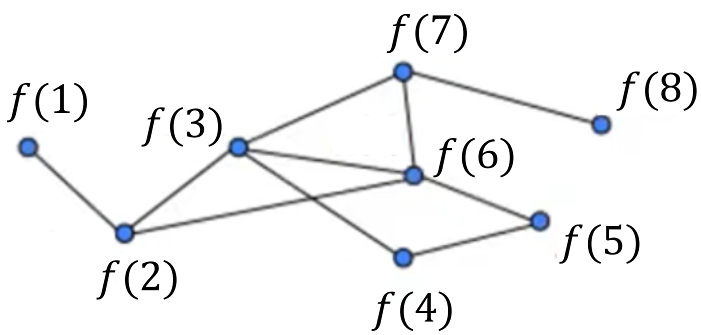
$D = diag(degree(v_1)\,..\,degree(v_n))$

: Degree matrix

$\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{A}$: *Laplacian matrix*

Signal on Graph

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

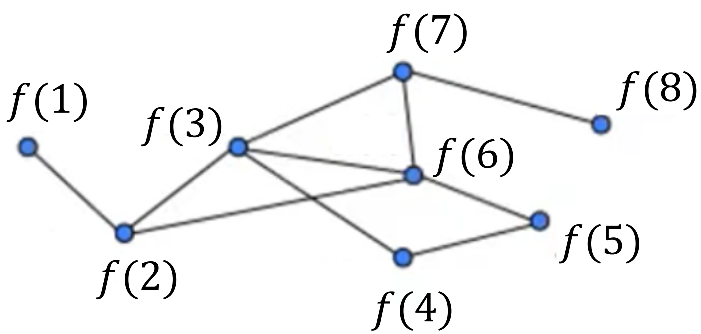


Graph signal $f: V \rightarrow \mathbb{R}^N$

: a function that assigns real values to each vertex of graph

Properties of Graph Laplacian

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



- Laplacian is a difference operator

$$(Lf)(i) = \sum_{j=N_i}^{N} A_{ij}(f(i) - f(j))$$

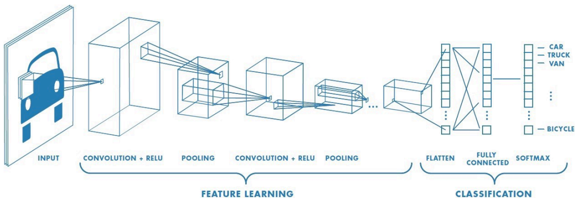
  - A real, symmetric matrix
  - Off-diagonal entries nonpostiive
  - Rows sum up to zero

- $L = U \Lambda U^T$
  - Has a complete set of orthonormal eigenvectors $\{ u_l \}_{l=0,1,\ldots,N-1}$ and associated real, nonnegative eigenvalues $\{ \lambda_l \}_{l=0,1,\ldots,N-1}$

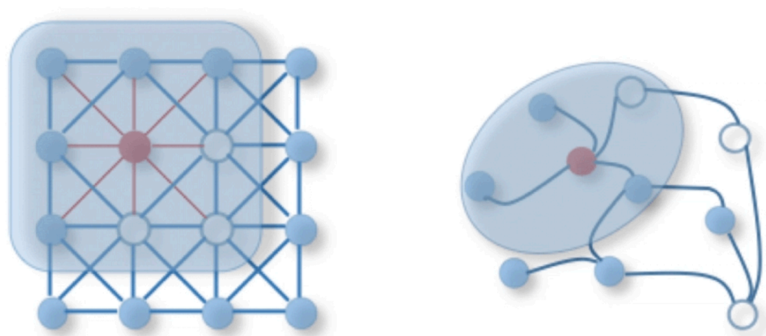
- Quadratic form on Laplacian L : Graph Signal Smoothness → notion of frequency

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{N} A_{ij}(f(i) - f(j))^2$$

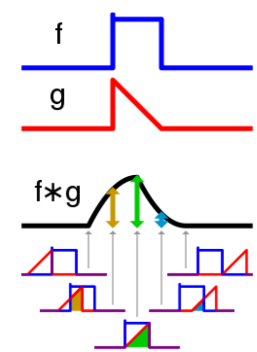
Convolution on Grid(Euclidean) vs Graph(non-Euclidean)



?



$$f * g\,(x) = \sum_{y} f(y)g(x-y)$$



Graph has no spatial axis!

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution on Grid(Euclidean) vs Graph(non-Euclidean)

Convolution theorem

$$f * g\,(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

thus,

$$f \xrightarrow{\text{FT}} \mathcal{F}(f) \xrightarrow{\mathcal{F}(g)} \mathcal{F}(f)\mathcal{F}(g) \xrightarrow{i\text{FT}} \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

$\mathbb{R}^{\boldsymbol{d}}$

$$f \xrightarrow{\text{FT}} \int_{\mathbb{R}} f\, e^{-i\omega t} dt \xrightarrow{\int_{\mathbb{R}} g\, e^{-i\omega t} dt} \mathcal{F}(f)\mathcal{F}(g) \xrightarrow{i\text{FT}} \int_{\mathbb{R}} \mathcal{F}(f)\mathcal{F}(g)\, e^{i\omega t} dt$$

graph

$$f \xrightarrow{\text{GFT}} u^T f \xrightarrow{\hat{g}(\Lambda)} \hat{g}(\Lambda)\, u^T f \xrightarrow{i\text{GFT}} u\, \hat{g}(\Lambda)\, u^T f$$

(Spectral graph processing)

what is $u\,,\,\Lambda$?

Graph Fourier Transform

- **Graph Laplacian is analogous to Fourier Transform!**
- $L = U \Lambda U^T$

$$L = \underbrace{U}_{\text{eigenvector}} \underbrace{\Lambda}_{\text{eigenvalue}} U^T$$

- <u>*The eigenvectors of the graph Laplacian are used for defining the Graph Fourier Transform*</u>

GFT: $\hat{f}(\lambda_l) \coloneqq < \mathbf{f}, \mathbf{u}_l > = \sum_{i=1}^{N} f(i) u_l^T(i)$

*i*GFT: $f(i) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\, u_l(i)$

- ,which is analogous to the classical Fourier Transform built on eigenfunctions on the 1-D Laplace operator

FT: $\hat{f}(\omega) \coloneqq < \mathbf{f}, e^{-i\omega t} > = \int_{\mathbb{R}} f(t) e^{-i\omega t} dt$

*i*FT: $f(t) = \int_{\mathbb{R}} \hat{f}(\omega) e^{i\omega t} dt$

<u>*Eigenvectors of Laplacian has FT like characteristics!*</u>

$$\mathcal{F}(\mathbb{x}) = u^T \mathbb{x}$$
$$\mathcal{F}^{-1}(\mathbb{x}) = u \mathbb{x}$$

one layer

$$\mathbb{x}$$

$$\downarrow$$

$$g_\theta \star \mathbb{x} = U g_\theta(\Lambda) U^T \, \mathbb{x}$$

$$\downarrow$$

$$\mathbb{x}' = \sigma(g_\theta \star \mathbb{x})$$

Spectral Graph Convolution (ICLR 2014, NIPS 2016)

- Spectral CNN (ICLR 2014)

$$\mathbb{x}^{k+1} = \sigma(\sum_i U\theta_i^k U^T \, \mathbb{x}_i^k) \qquad \theta_i^k = g_{\theta'}(\Lambda)$$

- Chebyshev Spectral CNN (NIPS 2016)

Chebyshev polynomials $T_k(x)$ up to K-th order

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

$$\mathbb{x}^{k+1} = \sigma(\sum_i U\theta_i^k U^T \, \mathbb{x}_i^k) \qquad \theta_i^k = g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta_k' T_k(\tilde{\Lambda})$$

thus, convolution of signal $\mathbb{x}^k$ with filter g_θ

$$U\Lambda^k U^T = (U\Lambda U^T)^k$$

$$\mathbb{x}^{k+1} = \sigma(g_\theta \star \mathbb{x}^k) = \sigma\left(\sum_{k=0}^{K} \theta_k' T_k(\tilde{L}) \, \mathbb{x}_i^k\right)$$

" K-localized "

$$\tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N \qquad \tilde{L} = \frac{2}{\lambda_{max}}L - I_N$$

[1] Bruna et al., ICLR 2014 , [2] Defferrad et al., NIPS 2016

GCN (ICLR 2017) *fast apprx.*

$$g_\theta \star \mathbb{x} \quad = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) \mathbb{x}_i^k)$$

(Chebyshev Spectral CNN)

$K = 1 \qquad \lambda_{max} \approx 2$ 1st-order only

$$\approx \theta'_0 x + \theta'_1 (L - I_N)$$

(L : normalized by D, $L = I_N - D^{-\frac{1}{2}} A\, D^{-\frac{1}{2}}$)

$^1/_{\sqrt{d_i d_j}}$

$$= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A\, D^{-\frac{1}{2}}\, x$$

$\theta = \theta'_0 = -\theta'_1$ single parameter

$$\approx \theta \left(I_N + D^{-\frac{1}{2}} A\, D^{-\frac{1}{2}} \right) x$$

$(\tilde{A} = A + I_N,\ \widetilde{D}_{ii} = \sum_j \tilde{A}_{ij})$

renormalization trick

$$\rightarrow \theta \left(\widetilde{D}^{-\frac{1}{2}} \tilde{A}\, \widetilde{D}^{-\frac{1}{2}} \right) x$$

GCN (ICLR 2017)

Given signal $X \in \mathbb{R}^{N \times C}$ (N nodes each with C – dimensional feature vector)

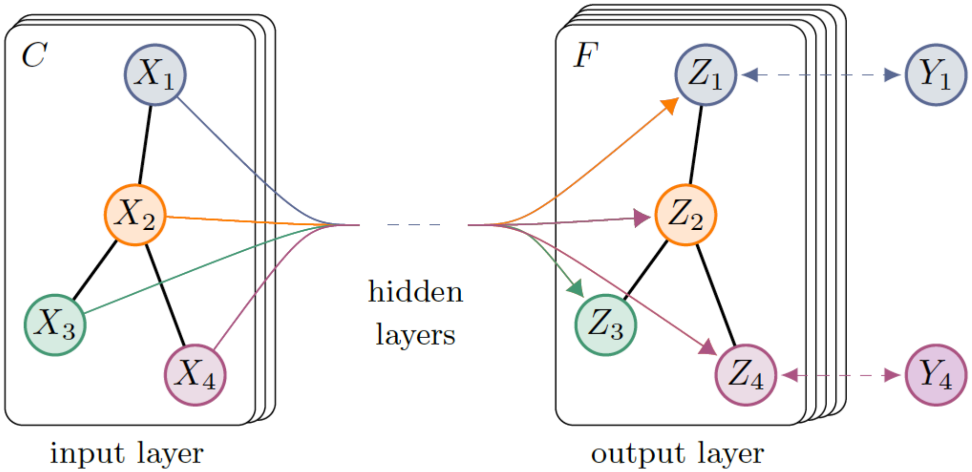
a convolved signal matrix $Z \in \mathbb{R}^{N \times F}$ would be,

$$Z = f(X, A) = \hat{A}\, X\, W \qquad where\ \hat{A} = \widetilde{D}^{-\frac{1}{2}} \tilde{A}\, \widetilde{D}^{-\frac{1}{2}}$$

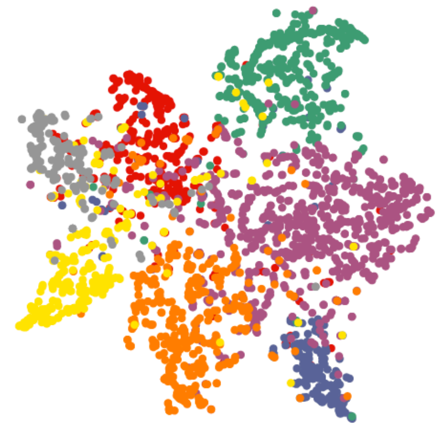
W : filter params matrix $W \in \mathbb{R}^{C \times F}$

F : number of filters / feature maps
C : number of input channels
N : number of nodes

GCN layer for semi-supervised node classification (ICLR 2017)



(a) Graph Convolutional Network



(b) Hidden layer activations

$$Z = f(X, A) = softmax(\hat{A}\ ReLU\left(\hat{A}XW^{(0)}\right) W^{(1)}$$

$X \in \mathbb{R}^{N \times C}$ $\quad W^{(0)} \in \mathbb{R}^{C \times H}$

$Z \in \mathbb{R}^{C \times F}$ $\quad W^{(1)} \in \mathbb{R}^{H \times F}$

A : symmetric adjacency matrix

$\hat{A}$: normalized A

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

(cross-entropy)

Result

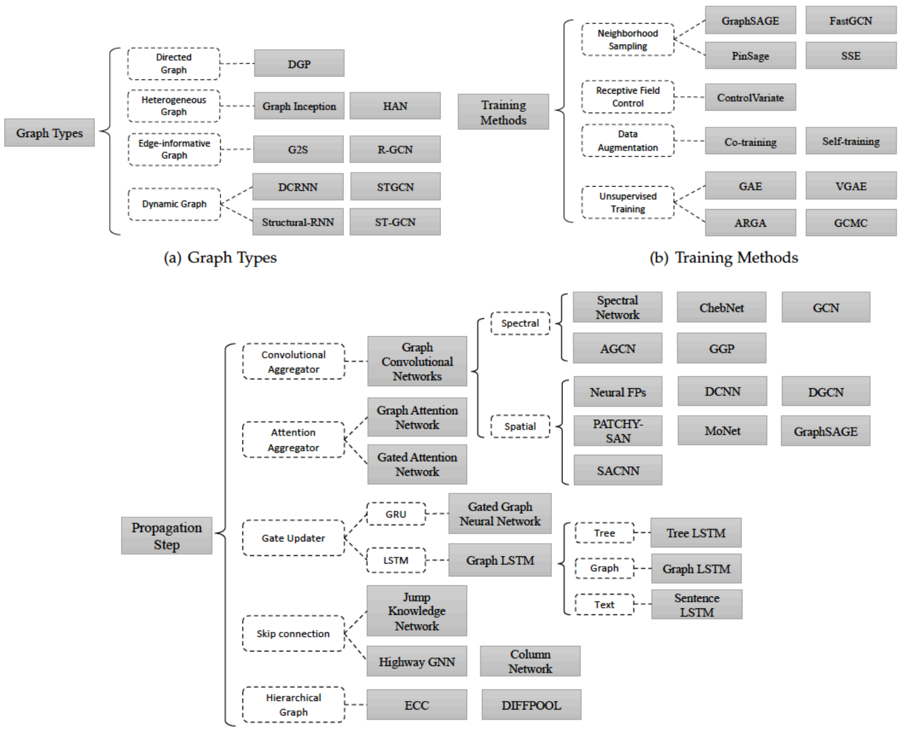
Table 2: Summary of results in terms of classification accuracy (in percent).

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)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Table 3: Comparison of propagation models.

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K = 3$	$\sum_{k=0}^{K} T_k(\tilde{L}) X \Theta_k$	69.8	79.5	74.4
	$K = 2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X \Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Related Works



(a) Graph Types

(b) Training Methods

(c) Propagation Steps

TABLE 2
Different variants of graph neural networks.

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^{K} \mathbf{N}_k \mathbf{\Theta}_k$
	1^{st}-order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}_0\mathbf{\Theta}_0 + \mathbf{N}_1\mathbf{\Theta}_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
	GCN	$\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
Non-spectral Methods	Neural FPs	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t \mathbf{W}_L^{\mathcal{N}_v})$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^*\mathbf{X}$ Graph classification: $\mathbf{N} = 1_N^T\mathbf{P}^*\mathbf{X}/N$	$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{N})$
	GraphSAGE	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t\left(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}\right)$	$\mathbf{h}_v^t = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \Vert \mathbf{h}_{\mathcal{N}_v}^t]\right)$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \Vert \mathbf{W}\mathbf{h}_k]\right)\right)}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \Vert \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \Vert_{m=1}^{M} \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$ Multi-head average: $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k\in\mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$	$\mathbf{h}_v^t = \mathbf{h}_{\mathcal{N}_v}^t$
Gated Graph Neural Networks	GGNN	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z\mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r\mathbf{h}_v^{t-1})$ $\widetilde{\mathbf{h}_v^t} = \tanh(\mathbf{W}\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^t}$
Graph LSTM	Tree LSTM (Child sum)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{U}^i\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma\left(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}^f\mathbf{h}_k^{t-1} + \mathbf{b}^f\right)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{U}^o\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{U}^u\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Tree LSTM (N-ary)	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{l=1}^{K} \mathbf{U}_l^i \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v k}^{tf} = \sum_{l=1}^{K} \mathbf{U}_{kl}^f \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{l=1}^{K} \mathbf{U}_l^o \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{l=1}^{K} \mathbf{U}_l^u \mathbf{h}_{vl}^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v k}^{tf} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^{K} \mathbf{f}_{vl}^t \odot \mathbf{c}_{vl}^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Graph LSTM in [44]	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

Zhou et al., ArXiv 2018

Reference

1) Kipf, Welling et al., SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONA L NETWORKS, ICLR 2017
2) Bruna et al., Spectral Networks and Deep Locally Connected Networks on Graphs, ICL R 2014
3) Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spect ral Filtering, NIPS 2016
4) Zhou et al., Graph Neural Networks: A Review of Methods and Applications, ArXiv 20 18
5) Wu et al. A Comprehensive Survey on Graph Neural Networks. ArXiv. 2018
6) How powerful are Graph Neural Network, ICLR 2019

TODO

Dataset: CORA

Cora.contents	index	(2708,)	[23452 1345 14563 ...]
	features	(2708, 1433)	[[0 1 0 0 0 ...] [1 0 0 1 1 ...] ...]
	label	(2708,)	['Neural_Networks' 'Rule_Learning' ...]
Cora.cites	edges	(5429, 2)	[[35 20304] [2456 12333] ...]

Data preprocessing

$$Z = softmax(\hat{A}\ ReLU(\hat{A}XW^{(0)})\ W^{(1)}$$

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf}\ ln\ Z_{lf}$$

index $\mathbb{R}^N$

edges → adjacency matrix $\mathbb{R}^{N \times N}$ → A (symmetric adj)

→ $\tilde{A} = A + I_N$ → $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\,\tilde{D}^{-\frac{1}{2}}$ (normalized $\tilde{A}$)

features $\mathbb{R}^{N \times C}$ → X (normalized features)

labels $\mathbb{R}^N$ → Y (one-hot encoded label) $\mathbb{R}^{N \times 7}$

Model/ Training

$$Z = f(X, A) = softmax(\hat{A}\ ReLU\left(\hat{A}XW^{(0)}\right) W^{(1)}$$

$$X \in \mathbb{R}^{N \times C} \qquad W^{(0)} \in \mathbb{R}^{C \times H}$$

$$Z \in \mathbb{R}^{C \times F} \qquad W^{(1)} \in \mathbb{R}^{H \times F}$$

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf}\ ln\ Z_{lf}$$

MLILAB Machine Learning & Intelligence

TODO

utils.py

```
def load_data(path, dataset):
    # TODO build adjacency matrix using edge_unordere

    # TODO build symmetric adjacency matrix

    # TODO normalize features, adj
```

```
def normalize(mx):
    # TODO normalization of given matrix mx
    return mx
```

```
def accuracy(output, labels):
    # TODO preds
```

main.py

```
# TODO
output = model(features, adj)
loss_train = None
acc_train = None
loss_train.backward()
optimizer.step()

loss_val = None
acc_val = None
```

```
# TODO
output = model(features, adj)
loss_test = None
acc_test = None
```

model.py

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()
        # TODO
        pass

    def forward(self, x, adj):
        # TODO
        pass
```

layer.py

```
class GraphConvolution(Module):

    def forward(self, input, adj):
        # TODO one layer of GCN
        output = None
```

https://github.com/seohuibae/GCN-week08

MLILAB Machine Learning & Intelligence

Thank you !

Any Questions ?

2020.02.27.

Seohui Bae

shbae73@kaist.ac.kr