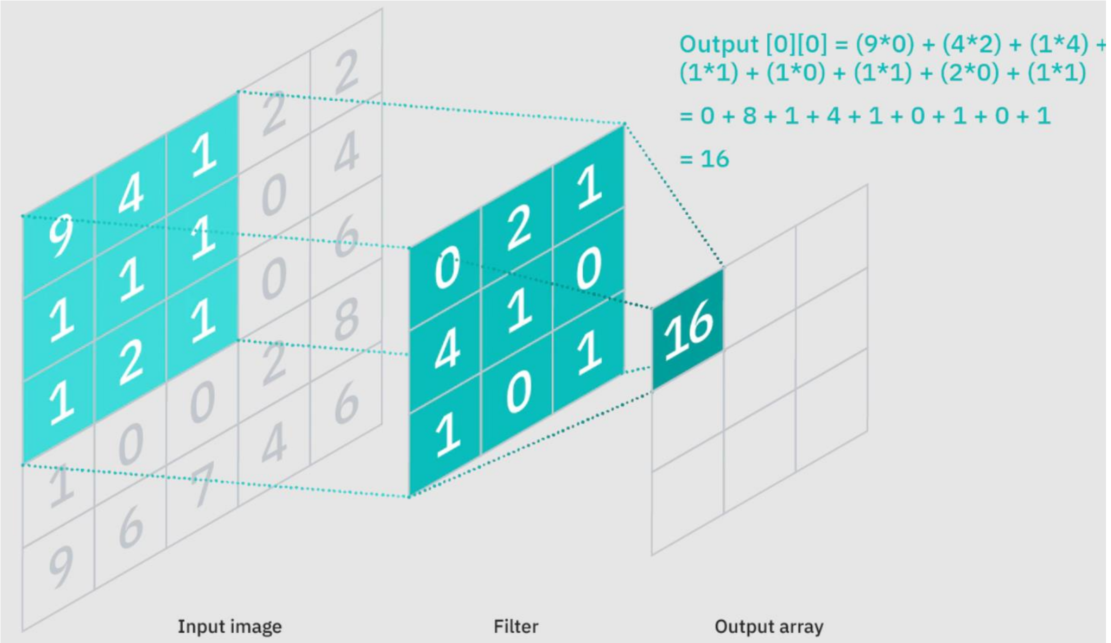
SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

By Thomas N. Kipf, Max Welling

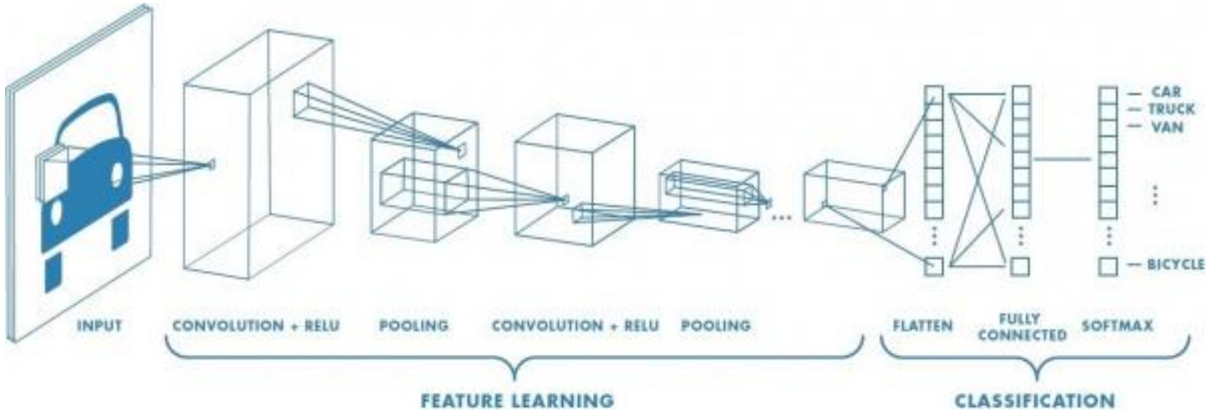
김이삭

Contents

Background
CNN



convolution



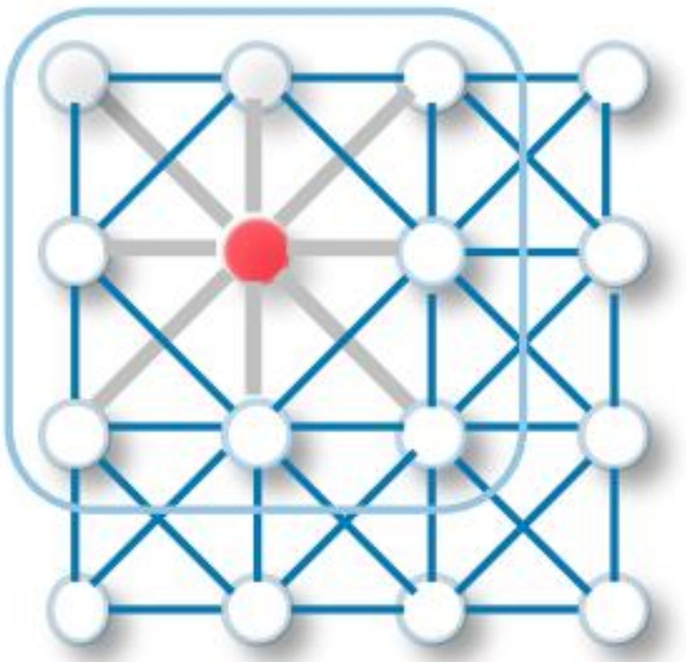
Weight Sharing

-less parameter

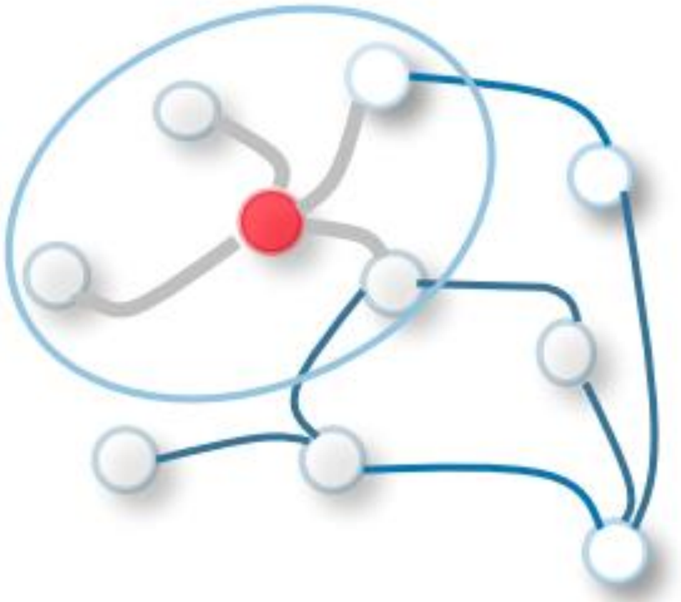
Local feature

Background
Image vs Graph

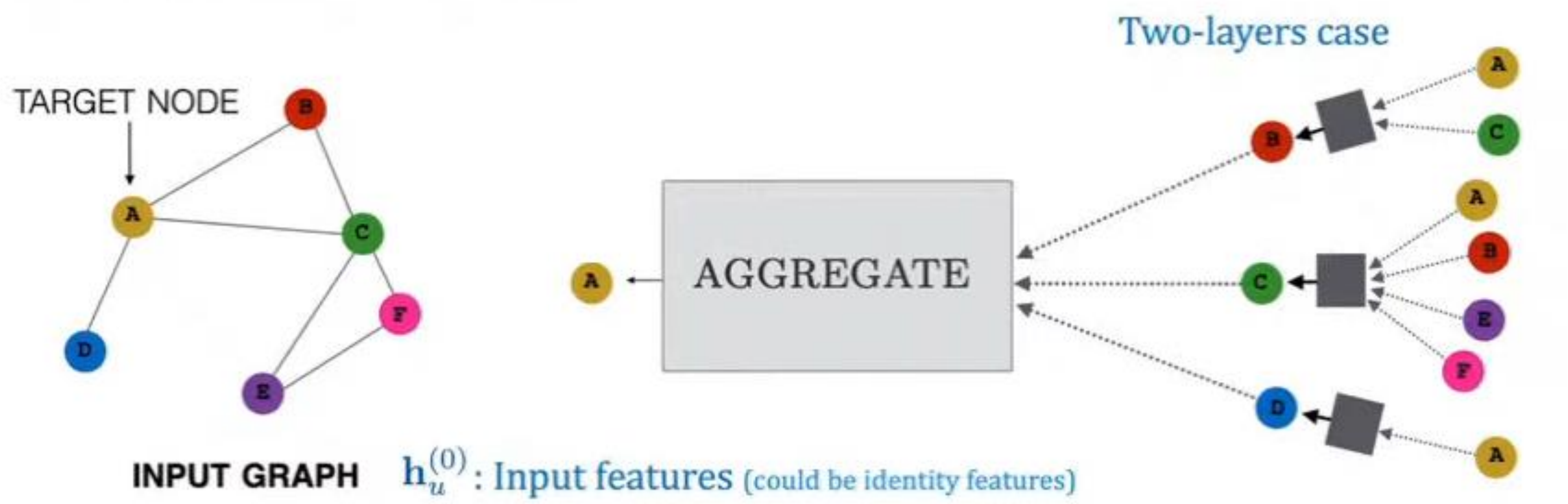
이웃한 노드와의 관계를 통해 계산할 수 없을까



Convolution filter 의 크기가 고정되어 있지 않음
순서가 없음



Background
GNN



$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

Permutation invariance

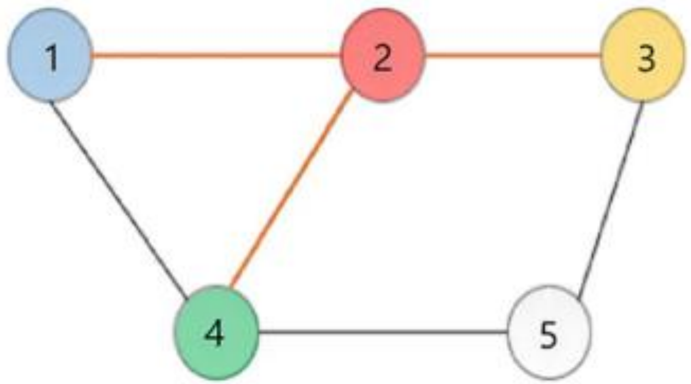
Background

Adjacency Matrix and Convolution on Graph

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

Adjacency matrix

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v\in\mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

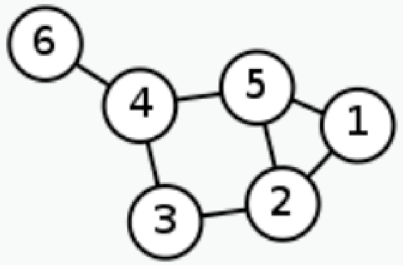


$$H_2^{(l+1)} = \sigma\left(H_1^{(l)}W^{(l)} + H_2^{(l)}W^{(l)} + H_3^{(l)}W^{(l)} + H_4^{(l)}W^{(l)} + b^{(l)}\right)$$

$$\Rightarrow \quad H_i^{(l+1)} = \sigma\left(\sum_{j\in N(i)} H_j^{(l)}W^{(l)} + b^{(l)}\right)$$

Background
Laplacian Matrix

Laplacian Matrix = Degree matrix – Adjacency matrix

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&3&0&0\\ 0&0&0&0&3&0\\ 0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\ 1&0&1&0&1&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&1\\ 1&1&0&1&0&0\\ 0&0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 2&-1&0&0&-1&0\\ -1&3&-1&0&-1&0\\ 0&-1&2&-1&0&0\\ 0&0&-1&3&-1&-1\\ -1&-1&0&-1&3&0\\ 0&0&0&-1&0&1 \end{pmatrix}$

Undirected graph

Positive Semi-Definite matrix
→non-negative eigen value

$$\mathbf{x}^T A\mathbf{x} \geq 0, \quad \forall \mathbf{x} \quad \leftrightarrow \quad A \geq 0$$

$$x^T L x = x^T D x - x^T W x = \sum_i D_{ii} x_i^2 - \sum_{i,j} x_i W_{ij} x_j$$
$$= \frac{1}{2}\left(2\sum_i D_{ii}x_i^2 - 2\sum_{i,j} W_{ij}x_i x_j\right)$$
$$\overset{(a)}{=} \frac{1}{2}\left(2\sum_i\left\{\sum_j W_{ij}\right\}x_i^2 - 2\sum_{i,j}W_{ij}x_ix_j\right)$$
$$\overset{(b)}{=} \frac{1}{2}\left(\sum_{i,j}W_{ij}x_i^2+\sum_{i,j}W_{ij}x_j^2-2\sum_{i,j}W_{ij}x_ix_j\right)$$
$$=\frac{1}{2}\sum_{i,j}W_{ij}(x_i-x_j)^2\geq 0$$

proof

Background

normalized Laplacian Matrix

Laplacian Matrix의 문제점
– sensitive to degree

Laplacian matrix

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

$$\begin{pmatrix} 1 & \frac{1}{\sqrt{6}} & 0 & 0 & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{6}} & 1 & \frac{1}{\sqrt{6}} & 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{\sqrt{6}} & 1 & \frac{1}{\sqrt{6}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{6}} & 1 & \frac{1}{3} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{3} & 0 & \frac{-1}{3} & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & 0 & 1 \end{pmatrix}$$

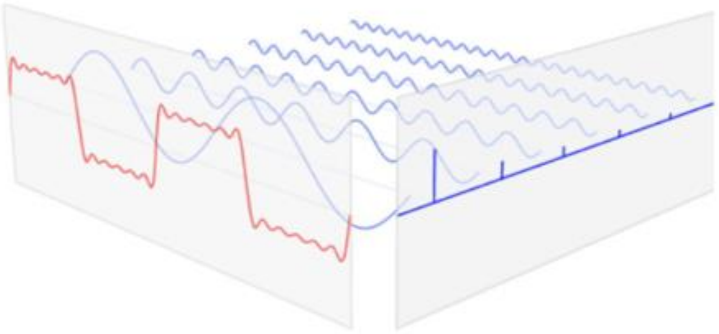
정규화식 $L = I - D^{-1/2}AD^{-1/2}$

$$L_{ii} = 1, L_{ij(i \neq j)} = -A_{ij}/\sqrt{degree(i)degree(j)}$$

Background

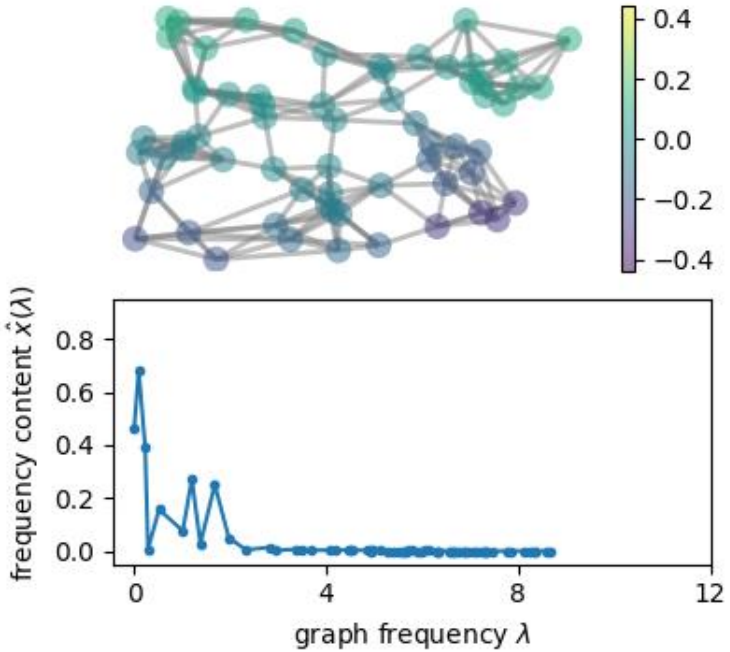
Graph Fourier Transform

푸리에 변환



시간영역에서 주파수 영역으로 분해, 변환

그래프 푸리에 변환



고유값 분해를 통해 frequency domain으로 변환
eigen value -> frequency
eigen vector -> Fourier basis

$$\mathscr{F}(\mathrm{x}) = U^T \mathrm{x}$$

$$\mathscr{F}^{-1}(\hat{\mathrm{x}}) = U\hat{\mathrm{x}}$$

$$L = U \Lambda U^T, U = [u_0, u_1 .. u_{n-1}] \in R^{n \times n}$$

Background

- Adjacency Matrix를 활용해 Convolution 기법을 그래프에도 적용 가능
- Laplacian Matrix를 eigen decomposition 하여 그래프를 frequency domain으로 변환 가능

Introduction

- Classifying nodes (small subset of nodes are labeled)
- 레이블 정보들을 전체 그래프에 smooth시키기

Loss function: graph Laplacian regularization term

$$\mathcal{L}_{reg} = \sum_{i}, j A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^T \Delta f(X), \ \mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}$$

Δ:D-A=Laplacian Matrix
L0: label된 데이터의 loss

Fast Approximate Convolutions

Propagation rule

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

$$\Downarrow$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

$\tilde{A} = A + I_N$

$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

SPECTRAL GRAPH CONVOLUTIONS

$$\text{2D Convolution} : f * g(x) = \sum_{y} f(y)g(x-y)$$

$$\text{Graph Convolution} : f * g(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

Spectral convolution 식

$$g_\theta * x = U g_\theta U^T x$$

U: L의 고유벡터

그래프 푸리에 변환 -> frequency영역에서 주파수 처리($g\theta$) ->역 푸리에 변환

$g\theta$ 도 학습 파라미터로 사용

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$$

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x),\ T_0(x) = 1, T_1(x) = x$$

$$\tilde{L} = 2L/\lambda_{max} - I_N$$

고유값 분해의 연산 비용이 크기 때문에 Chebyshev 다항식으로 근사

Layer-wise Linear Model

K=1, λmax =2로 근사

$$g_{\theta'} * x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x + \theta'_1 D^{-1/2} A D^{-1/2} x$$

파라미터를 줄이기 위해 근사

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

$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$

일반화

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

convolution filter Θ를 학습

Semi-supervised Node Classification

filtering

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$$

예시

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

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$$

Loss function: label된 노드에 대해 cross entropy 계산

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

Experiment

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

Experiment

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
1^{st}-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	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**
1^{st}-order term only		$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Implementation

```python
class GCN(nn.Module):

    def __init__(self, feature_num, node_representation_dim, nclass, droupout=0.5
        super().__init__()

        self.gconv1 = GraphConvolution(feature_num, node_representation_dim)
        self.gconv2 = GraphConvolution(node_representation_dim, nclass)
        self.dropout = droupout

    def forward(self, x, adj):
        x = F.relu(self.gconv1(x, adj))
        x = F.dropout(x, self.dropout, self.training)
        x = self.gconv2(x, adj)
        return F.log_softmax(x, dim=1)
```

```
Training epoch 0 ; accuracy: 0.19285714285714287; loss: 1.9439032077789307
Validation epoch 0 ; accuracy: 0.47333333333333333; loss: 1.8687959909439087
Training epoch 1 ; accuracy: 0.5785714285714286; loss: 1.848462343215942 4
Validation epoch 1 ; accuracy: 0.5066666666666667; loss: 1.780288934707641 6
Training epoch 2 ; accuracy: 0.6071428571428571; loss: 1.736638188362121 6
Validation epoch 2 ; accuracy: 0.54; loss: 1.679686427116394
Training epoch 3 ; accuracy: 0.6571428571428571; loss: 1.611626148223877
Validation epoch 3 ; accuracy: 0.5666666666666667; loss: 1.576048135757446 3
Training epoch 4 ; accuracy: 0.6928571428571428; loss: 1.477960228919983
Validation epoch 4 ; accuracy: 0.59; loss: 1.474118709564209
Training epoch 5 ; accuracy: 0.7357142857142858; loss: 1.344674706459045 4
Validation epoch 5 ; accuracy: 0.62; loss: 1.376136779785156 2
Training epoch 6 ; accuracy: 0.7357142857142858; loss: 1.215998411178588 9
Validation epoch 6 ; accuracy: 0.6466666666666666; loss: 1.282765984535217 3
Training epoch 7 ; accuracy: 0.7857142857142857; loss: 1.087708830833435
Validation epoch 7 ; accuracy: 0.6766666666666666; loss: 1.194754481315612 8
Training epoch 8 ; accuracy: 0.8214285714285714; loss: 0.971481204032898
Validation epoch 8 ; accuracy: 0.7; loss: 1.112548589706421
Training epoch 9 ; accuracy: 0.8357142857142857; loss: 0.864678680896759
Validation epoch 9 ; accuracy: 0.72; loss: 1.036184430122375 5
Training epoch 10 ; accuracy: 0.8642857142857143; loss: 0.760062217712402 3
Validation epoch 10 ; accuracy: 0.7666666666666667; loss: 0.965624988079071
```

```
Training epoch 90 ; accuracy: 1.0; loss: 0.0035056455526500 94
Validation epoch 90 ; accuracy: 0.8233333333333334; loss: 0.7939309477806091
Training epoch 91 ; accuracy: 1.0; loss: 0.003209288930520415 3
Validation epoch 91 ; accuracy: 0.8233333333333334; loss: 0.7951657772064209
Training epoch 92 ; accuracy: 1.0; loss: 0.00313510000705719
Validation epoch 92 ; accuracy: 0.8233333333333334; loss: 0.7962270379066467
Training epoch 93 ; accuracy: 1.0; loss: 0.0029499891679733 99
Validation epoch 93 ; accuracy: 0.8233333333333334; loss: 0.7972925305366516
Training epoch 94 ; accuracy: 1.0; loss: 0.0030352470930665 73
Validation epoch 94 ; accuracy: 0.8233333333333334; loss: 0.7983310222625732
Training epoch 95 ; accuracy: 1.0; loss: 0.002908715512603521 3
Validation epoch 95 ; accuracy: 0.8233333333333334; loss: 0.7994802594184875
Training epoch 96 ; accuracy: 1.0; loss: 0.0029072735924273 73
Validation epoch 96 ; accuracy: 0.8233333333333334; loss: 0.8006298542022705
Training epoch 97 ; accuracy: 1.0; loss: 0.002795508829876780 5
Validation epoch 97 ; accuracy: 0.8233333333333334; loss: 0.8018730282783508
Training epoch 98 ; accuracy: 1.0; loss: 0.0030458415858447 55
Validation epoch 98 ; accuracy: 0.8233333333333334; loss: 0.803171694278717
Training epoch 99 ; accuracy: 1.0; loss: 0.002772687235847115 5
Validation epoch 99 ; accuracy: 0.8233333333333334; loss: 0.8045842051506042
```

Cora data에 GCN 기법을 적용한 결과
82%정도의 높은 정확도를 얻을 수 있었다.

Conclusion

- Using normalized $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, efficiently apply convolution network to graph data
- Using Chebynet approximation, reduce the computing cost and utilize local features

Limitation

- Memory problem – use mini-batch instead of full-batch
- limited to undirected graphs
- Self-connection problem – Give weight to self features

$$\tilde{A} = A + I_N \Rightarrow \tilde{A} = A + \lambda I_N$$

감사합니다