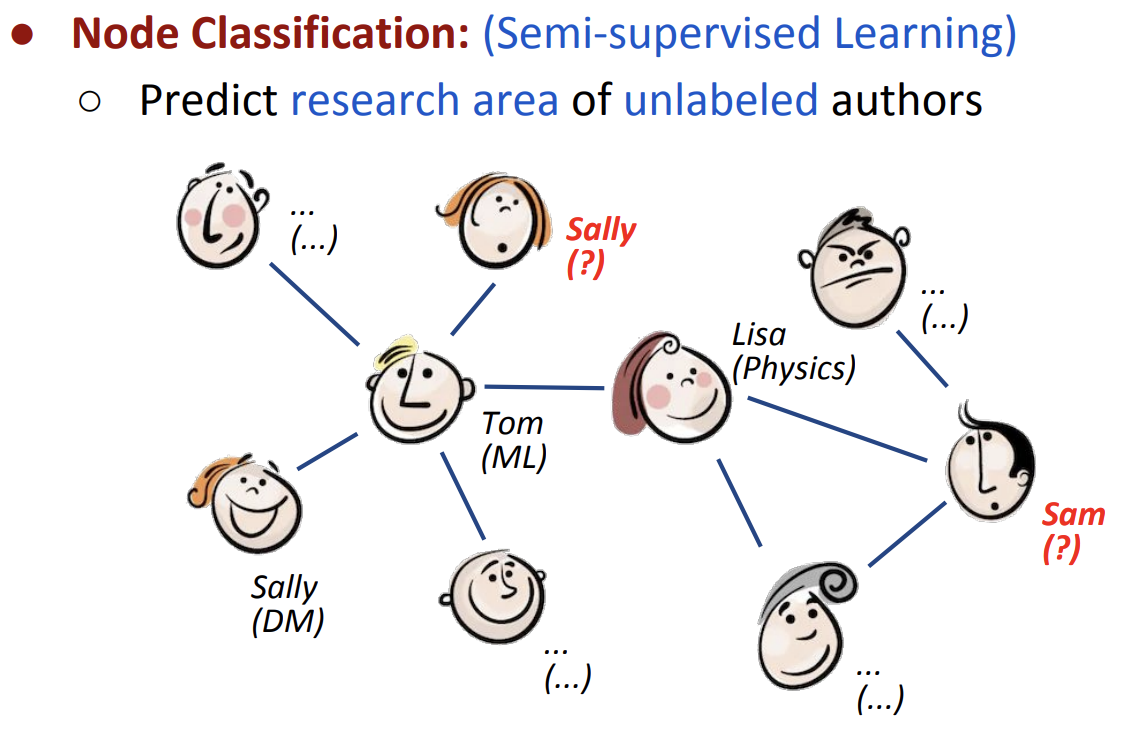
<https://baekyeongmin.github.io/paper-review/gcn-review/>

**Semi-Supervised Classification with Graph Convolutional Network(ICLR 2017)**



위 그림처럼 그래프의 노드 & 주어진 레이블을 이용해 다른 노드들의 레이블을 예측하는 semi-supervised classification 문제를 푸는 것

**Main Idea**

1. 그래프 각 노드의 Feature X & 인접 행렬 A가 주어졌을 때 Multi-layer Graph Convolutional Network(GCN), f(X,A)를 제시  
   이 방법이 spectral graph convolution(??)의 빠르고 효율적인 1차 근사임을 증명
2. 이전의 semi-supervised 조건의 연구들에서는 와 같은 loss 형태로 학습  
   여기서 L0는 레이블이 있는 노드에 대한 classification loss  
   L-reg는 Graph Laplacian regularization term = 연결된 노드가 비슷한 representation을 갖도록 하는 loss  
   하지만 이는 연결된 노드는 유사하도록 가정 = 동일한 레이블을 갖도록 유도 => 유사도 이외의 추가적인 정보 담기 힘들게 된다 => 모델 능력 제한  
   GCN에서는 인접행렬을 입력으로 이용해 어떤 노드들이 연결돼 있는지에 대한 정보를 직접 이용해 제한 해결

**GCN**

Graph Convolution 연산을 여러 번(multi-layer) 진행한다.



H-(l): l번째 layer의 hidden state, H-(0) = X(그래프 노드의 초기 feature)

A~: A+I-N, 인접행렬에 자기 자신으로의 연결(항등행렬)을 추가함.

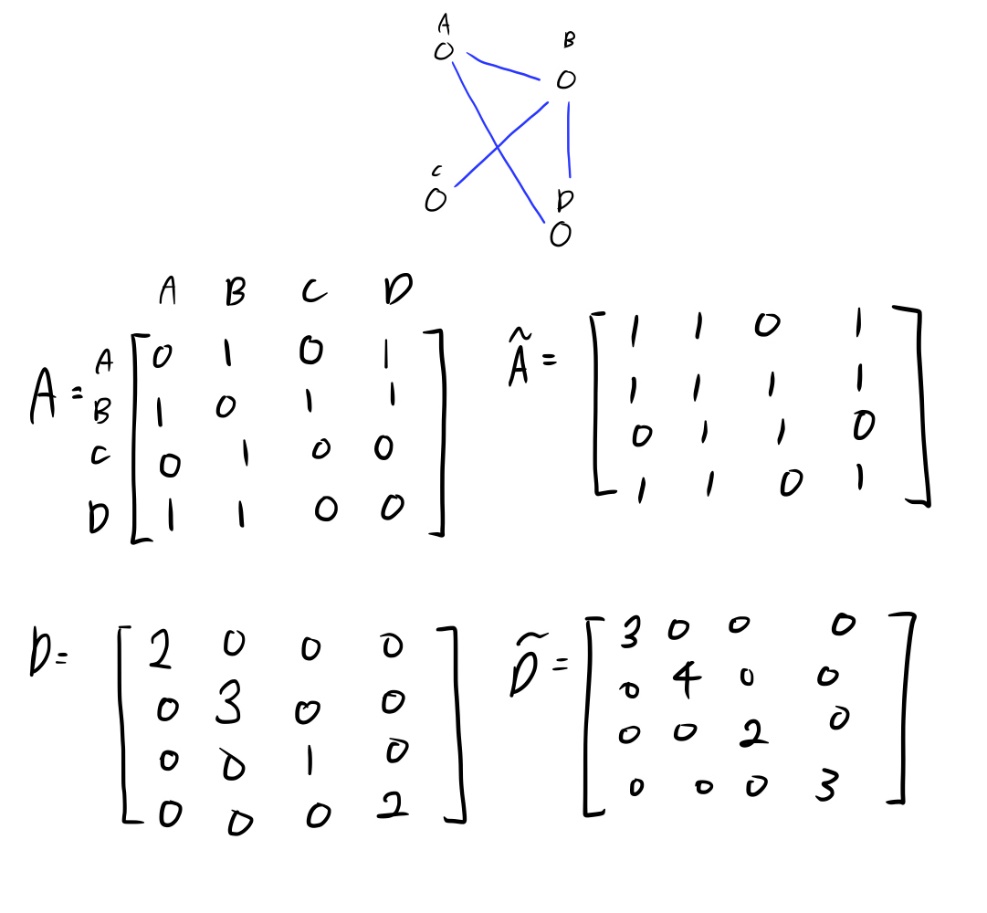
D~: 각 노드의 degree(차수)를 나타내는 대각 행렬 

W-(l): l번째 layer의 학습 parameter

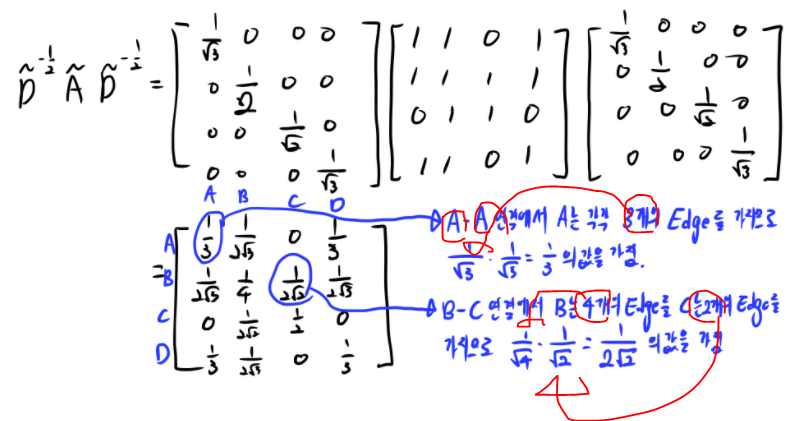
Activation은 ReLU를 사용

예시로 flow를 살펴보자

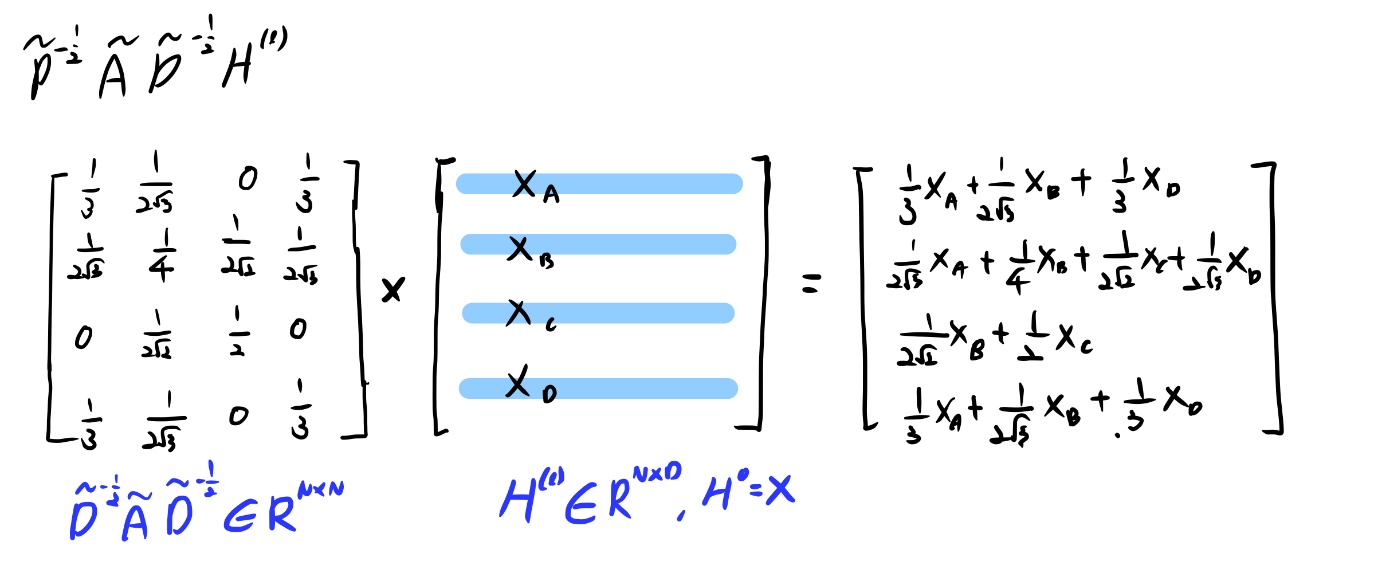
1. 노드 4개로 구성된 무방향, 무가중치 그래프일 때, 아래 그림과 같다.



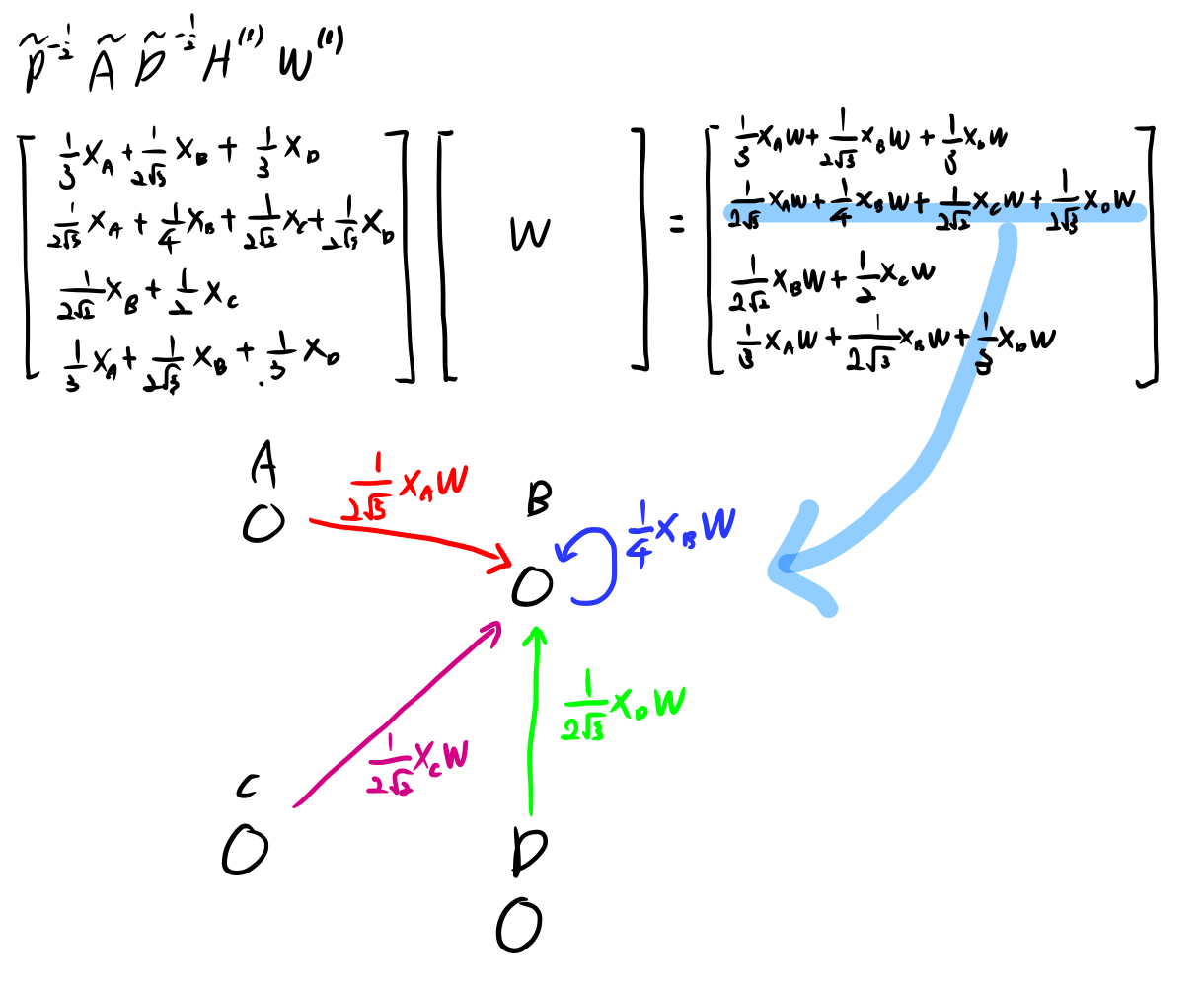
2. 를 계산 이때 A~에 따라 정규화한 값이 적용된다. => 각 노드별 edge의 갯수에 상관없이 모든 노드를 잘 학습하기 위한 장치



3. 각 노드의 피처 벡터와 곱하면 되는데   
각 노드의 피처가 특정 단어인 경우 word embedding을 곱하면 되며, 문서인 경우 BoW 벡터를, 아니면 one-hot 벡터를 피처로 이용해도 된다.



4. 그리고 학습 파라미터를 곱하게 된다. 결과적으로 GCN의 식은 특정 노드의 representation(각 행이 한 노드)으로 해당 노드에 연결되어 있는 노드들을 가중합하는 방법이다.



5. 마지막은 비선형함수를 거쳐 비선형성을 배우게 돈다.

Layer가 하나일 때는 각 노드들의 인접한 노드들만 탐색하여 그 노드를 표현하게(hidden state로) 된다.

하지만 이 layer를 K개 쌓으면 해당 노드에서 K개 떨어져 있는 노드까지 이용하여 그 노드를 표현하게 된다.

따라서 그 노드가 만약 레이블이 없다 하더라도 K 다리 건너있는 노드까지 고려하여 그 노드의 레이블을 예측할 수 있으며, 서론에서 언급한 인접 노드의 레이블과 유사하도록 하는 제한을 해결할 수 있다.(여러 정보를 파악하니까)

**Experiments**

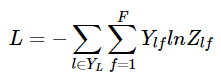
**Model**

본 논문에서는 2-layer GCN으로 진행





이때 D2는 class의 개수가 된다. ReLU -> Softmax(classification을 위해) 을 이용

그리고 레이블이 있는 데이터(노드)는 다음과 같이 loss 계산

**Dataset**

Citation Network: 각 노드가 문서이고 edge는 citation link  
label rate에 따라 label이 있는 노드가 있다.  
각 노드의 초기 피처는 BoW vectors  
학습 시 각 class에 대해 20개의 label data를 이용하여 학습 진행

NELL: NELL은 KG에서 추출된 biparitite graph(이분 그래프)  
(e1, r, e2)의 형태 관계, 이를 표현하기 위해 relation 노드를 따로 만들어 (e1,r1), (e2,r2) 같이 변형, 노드는 entity와 relation이라고 볼 수 있음  
초기 피처로 entity의 경우 sparse feature vector, relation의 경우 one-hot vector

**Setup**

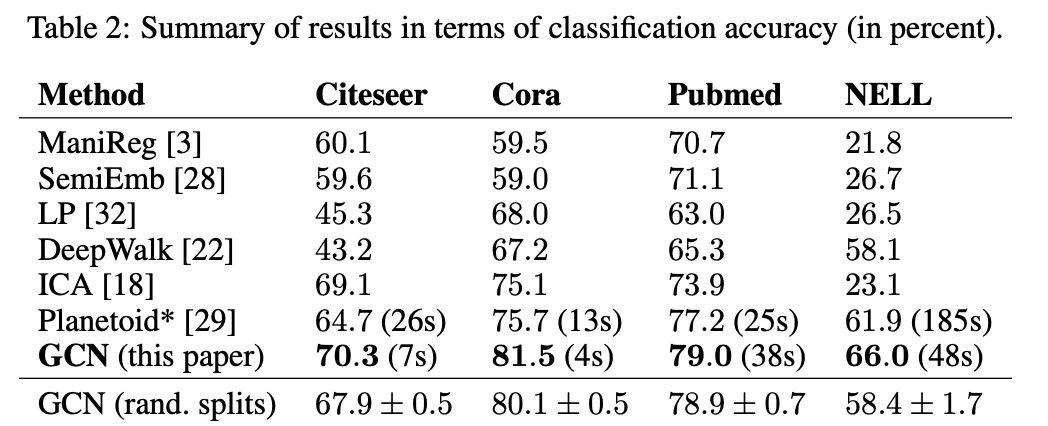
1000개 labeled data => testset, 500개 => validation set(하이퍼 파라미터 조절)

200 epoch, window size = 10, validation loss로 early stop

Adam optimizer, lr=0.01, 미니배치말고 풀배치

히든사이즈 D1 => citation net, nell 각각 16, 64

**Results**



이전 SOTA 모델과 동일한 데이터 split 사용, initialize를 다르게 한 100개의 모델 결과를 평균(앙상블 느낌인가)

Rand.splits은 10개의 랜덤으로 뽑힌 같은 크기의 학습 데이터셋(데이터셋을 랜덤) 했을 때 평균과 표준편차

**Limitation and Future Work**

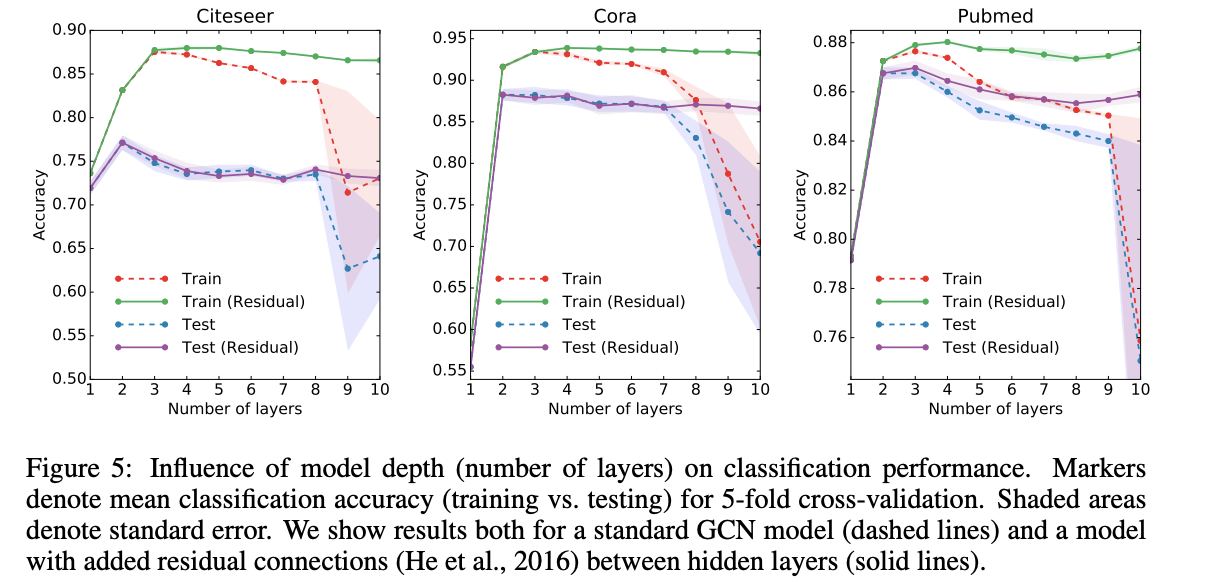
풀배치라 메모리 요구량 는다. 미니배치 사용 시 K개의 레이어의 경우 미니배치에 포함된 노드들의 K번째 이웃 노드들 또한 메모리에 연쇄적으로 가지고 있어야함

무방향, 무가중치 그래프에서만 적용, NELL 같은 유방향, 유가중치의 경우 추가적인 노드를 이용하여 설정을 지정한 다음에 undirected bipartite graph 형태로 바뀐 뒤 작동

**Appendix**

레이어 늘려가며 + 레지듀얼 이용하면서 진행

2~3개 사용했을 때 가장 좋음, 7개 넘어가면서 레지듀얼 없으면 optimize 안되고 성능도 떨어짐, 레이어 수 많아지면 파라미터 많아지고 오버핏 가능성



**TransGCN에서 따온 여러 그래프 임베딩의 차이점**

KG를 저차원으로 임베딩하여야 잘 사용할 수 있다. 그래야 link prediction과 같은 downstream task에서 잘 작동.

1. Head와 Tail의 거리를 측정해 triple의 존재를 예측 = translation-based methods  
   transe,transd,transr
2. Semantic Matching Energy based methods: triples의 존재를 예측하는 것은 같다, 두 엔티티와 릴레이션의 상호성을 이용하여  
   rescal, distmult, complex
3. Rotate등도 있다.

이 방법들은 triples 단위로 학습(triple 안의 head relation tail 이 중요) => 이웃과의 관계는 따지지 않으며

즉, 각 triples를 이걸로 임베딩하여 표현한 뒤 전체 그래프 임베딩하면 좋을 듯