그래프를 이용한 기계 학습

#10 그래프 신경망이란 무엇일까? (심화)

신기정

(KAIST AI대학원)

boostcamp AI Tech

1. 그래프 신경망 복습

2. 그래프 신경망에서의 어텐션

3. 그래프 표현 학습과 그래프 풀링

4. 지나친 획일화 문제

5. 그래프 데이터 증강

6. 실습: GraphSAGE의 집계 함수 구현

7. “그래프를 위한 기계 학습” 수업 복습

1. 그래프 신경망 복습

1.1 귀납식 정점 표현 학습
1.2 그래프 신경망의 구조
1.3 그래프 신경망의 학습
1.4 그래프 신경망의 활용

1.1 귀납식 정점 표현 학습

정점을 임베딩하는 함수, 즉 인코더를 학습하는 귀납식 정점 표현 학습은 여러 장점을 갖습니다

1) 학습이 진행된 이후에 추가된 정점에 대해서도 임베딩을 얻을 수 있습니다

2) 모든 정점에 대한 임베딩을 미리 계산하여 저장해둘 필요가 없습니다

3) 정점이 속성(Attribute) 정보를 가진 경우에 이를 활용할 수 있습니다

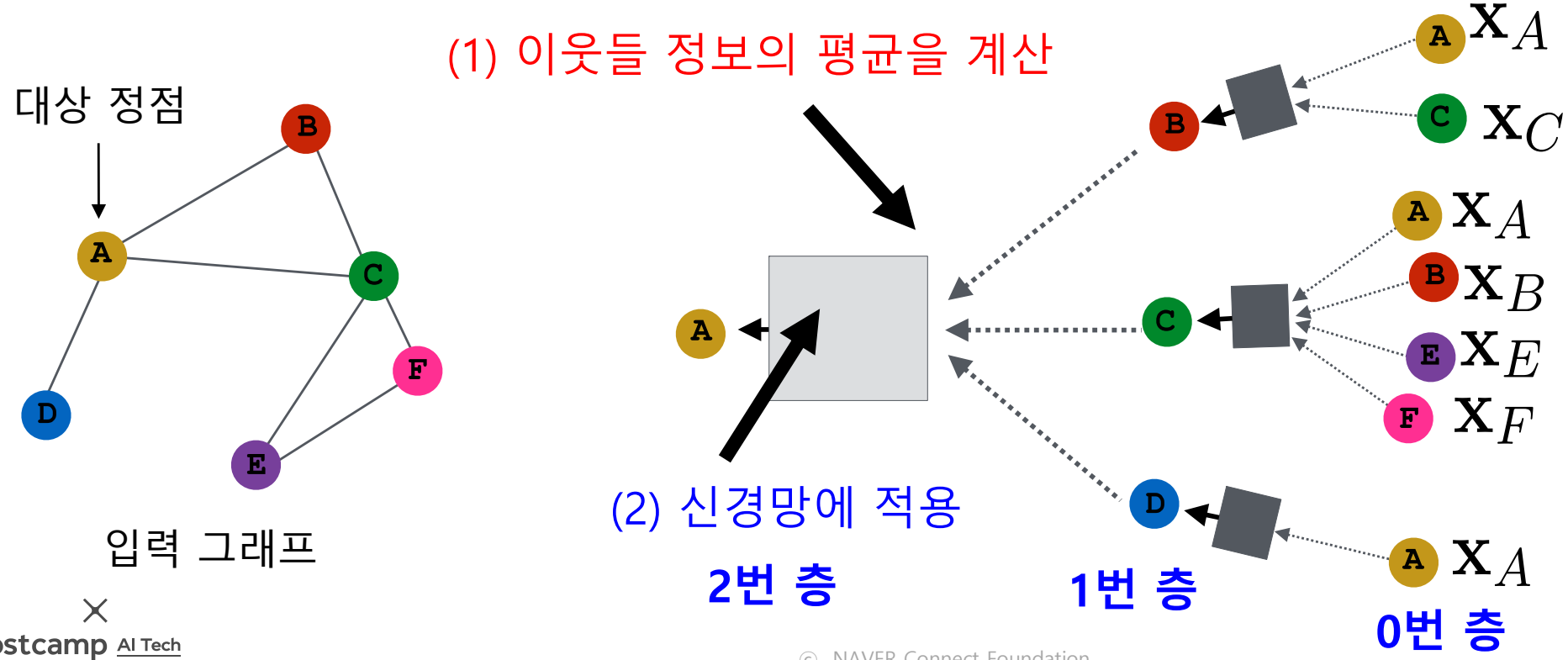
$$\text{ENC}(v) = \text{그래프 구조와 정점의 부가 정보를 활용하는 복잡한 함수}$$

그래프 신경망(Graph Neural Network)은 대표적인 귀납식 임베딩 방법입니다

1.2 그래프 신경망의 구조

그래프 신경망은 이웃 정점들의 정보를 집계하는 과정을 반복하여 임베딩을 얻습니다

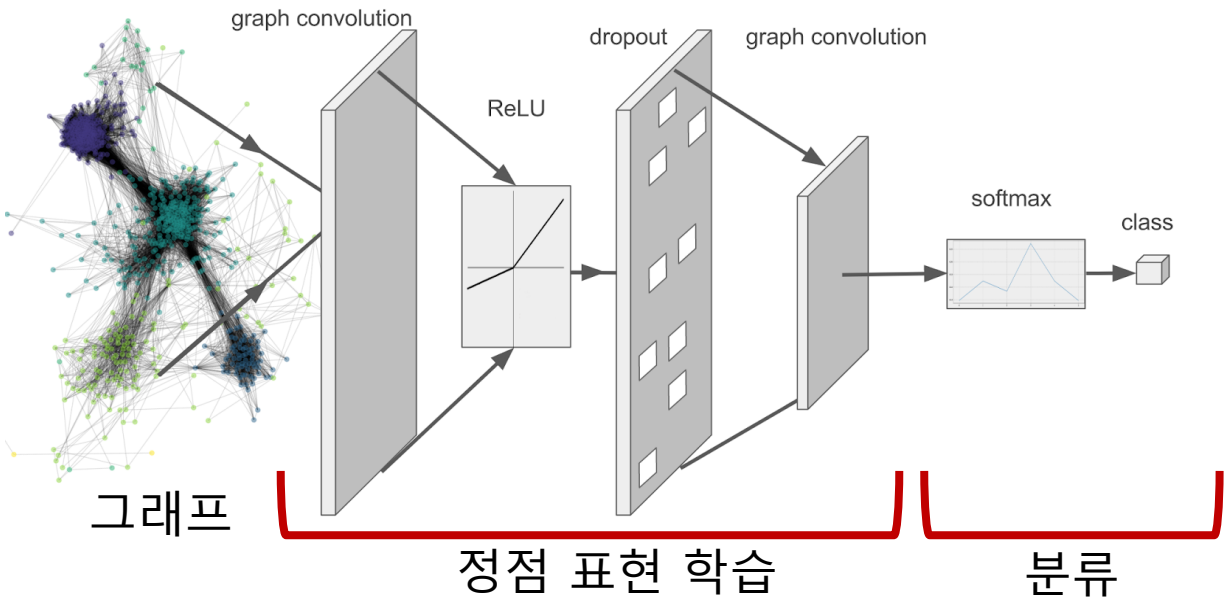
집계 함수의 형태에 따라, 그래프 신경망, 그래프 합성곱 신경망, GraphSAGE 등이 구분됩니다



1.2 그래프 신경망의 학습

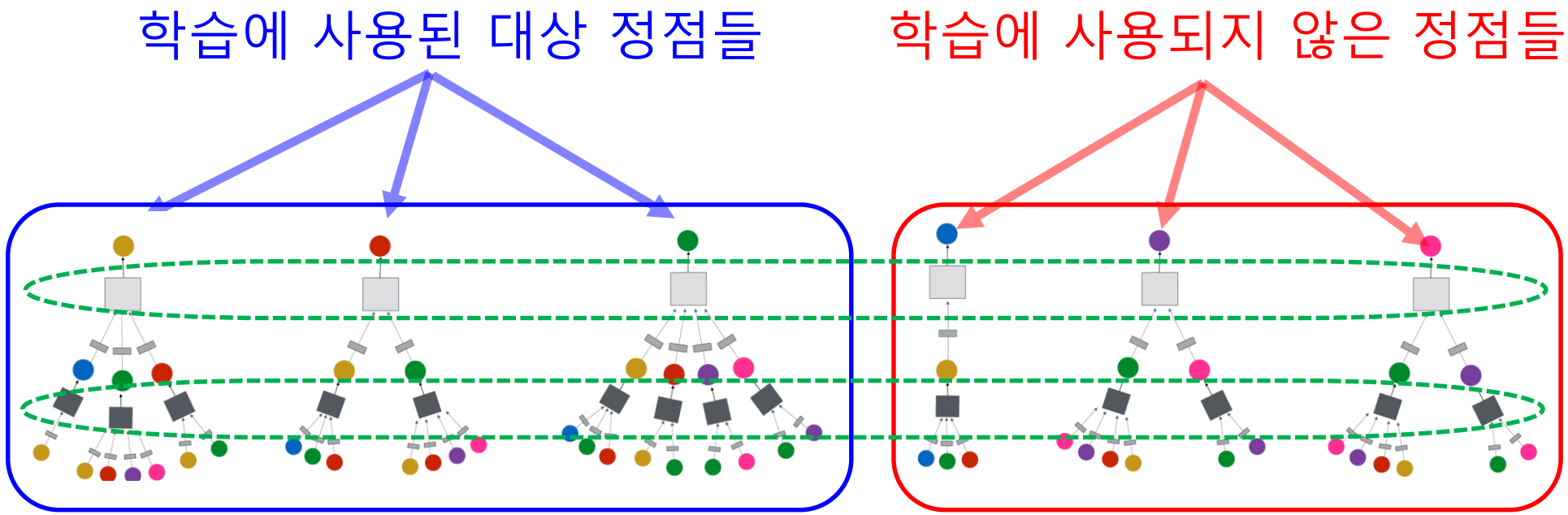
그래프 신경망은 비지도 학습, 지도 학습이 모두 가능합니다

비지도 학습에서는 정점간 거리를 "보존"하는 것을 목표로 합니다
지도 학습에서는 후속 과제의 손실함수를 이용해 종단종 학습을 합니다



1.3 그래프 신경망의 활용

학습된 신경망을 적용하여, 학습에 사용되지 않은 정점, 학습 이후에 추가된 정점,
심지어 새로운 그래프의 정점의 임베딩을 얻을 수 있습니다



2. 그래프 신경망에서의 어텐션

2.1 기본 그래프 신경망의 한계

2.2 그래프 어텐션 신경망

2.1 기본 그래프 신경망의 한계

기본 그래프 신경망에서는 이웃들의 정보를 동일한 가중치로 평균을 냅니다
그래프 합성곱 신경망에서 역시 단순히 연결성을 고려한 가중치로 평균을 냅니다

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, ..., K\}$$

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

2.2 그래프 어텐션 신경망

그래프 어텐션 신경망(Graph Attention Network, GAT)에서는 가중치 자체도 학습합니다

실제 그래프에서는 이웃 별로 미치는 영향이 다를 수 있기 때문입니다
가중치를 학습하기 위해서 셀프-어텐션(Self-Attention)이 사용됩니다

2.2 그래프 어텐션 신경망

각 층에서 정점 i로부터 이웃 j로의 가중치 α_{ij}는 세 단계를 통해 계산합니다

1) 해당 층의 정점 i의 임베딩 $\mathbf{h}_i$에 신경망 $\boldsymbol{W}$를 곱해 새로운 임베딩을 얻습니다

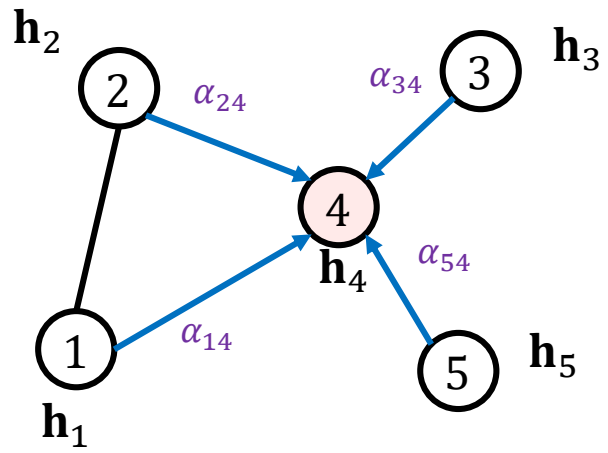
$$\tilde{\mathbf{h}}_i = \mathbf{h}_i \mathbf{W}$$

2) 정점 i와 정점 j의 새로운 임베딩을 연결한 후, 어텐션 계수 $\boldsymbol{a}$를 내적합니다
어텐션 계수 $\boldsymbol{a}$는 모든 정점이 공유하는 학습 변수입니다

$$e_{ij} = \boldsymbol{a}^{\top}\left[\text{CONCAT}(\tilde{\mathbf{h}}_i, \tilde{\mathbf{h}}_j)\right]$$

3) 2)의 결과에 소프트맥스(Softmax)를 적용합니다

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

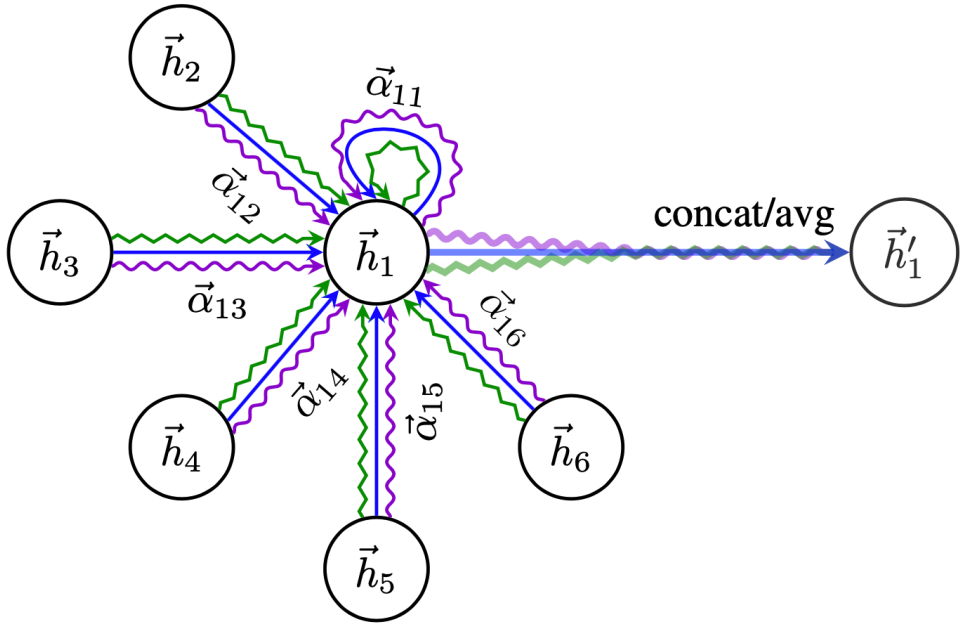


2.2 그래프 어텐션 신경망

여러 개의 어텐션을 동시에 학습한 뒤, 결과를 연결하여 사용합니다

멀티헤드 어텐션(Multi-head Attention)이라고 부릅니다

$$\mathbf{h}_i' = \underset{1\le k\le K}{\text{CONCAT}}\,\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{h}_j\mathbf{W}_k\right)$$



세 개의 어텐션을 사용한 GAT

2.2 그래프 어텐션 신경망

어텐션의 결과 정점 분류의 정확도(Accuracy)가 향상되는 것을 확인할 수 있었습니다

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

3. 그래프 표현 학습과 그래프 풀링

2.1 그래프 표현 학습

2.2 그래프 풀링

3.1 그래프 표현 학습

그래프 표현 학습, 혹은 그래프 임베딩이란 그래프 전체를 벡터의 형태로 표현하는 것입니다

개별 정점을 벡터의 형태로 표현하는 정점 표현 학습과 구분됩니다
그래프 임베딩은 벡터의 형태로 표현된 그래프 자체를 의미하기도 합니다

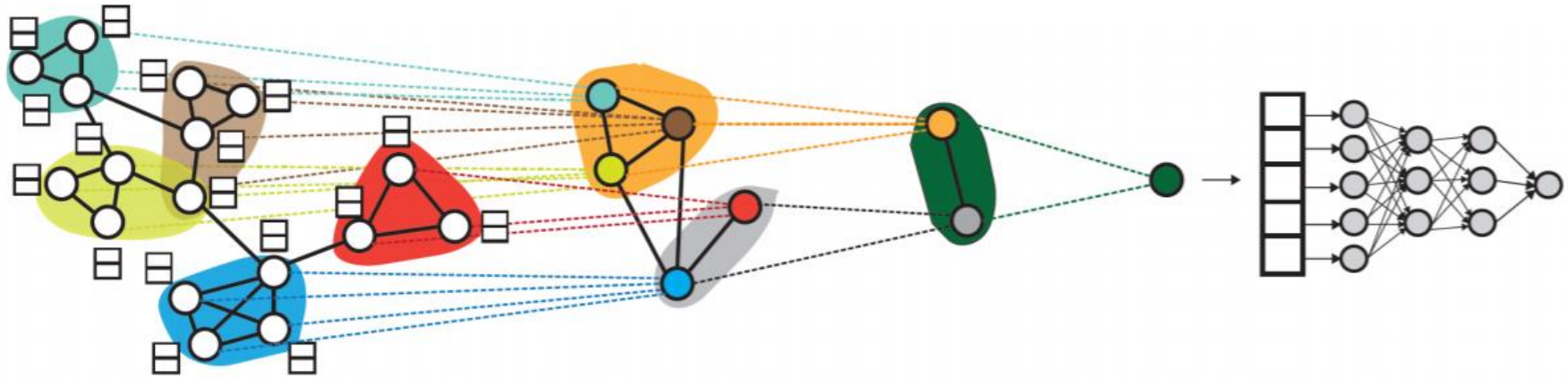
그래프 임베딩은 그래프 분류 등에 활용됩니다
그래프 형태로 표현된 화합물의 분자 구조로부터 특성을 예측하는 것이 한가지 예시입니다

3.2 그래프 풀링

그래프 풀링(Graph Pooling)이란 정점 임베딩들로부터 그래프 임베딩을 얻는 과정입니다

평균 등 단순한 방법보다 그래프의 구조를 고려한 방법을 사용할 경우
그래프 분류 등의 후속 과제에서 더 높은 성능을 얻는 것으로 알려져 있습니다

아래 그림의 미분가능한 풀링(Differentiable Pooling, DiffPool)은
군집 구조를 활용 임베딩을 계층적으로 집계합니다



이미지 출처: https://arxiv.org/pdf/1806.08804.pdf

4. 지나친 획일화 문제

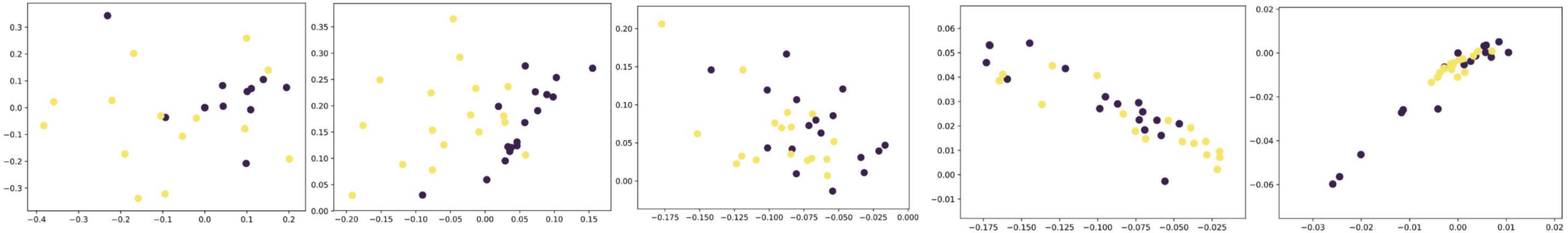
4.1 지나친 획일화 문제

4.2 지나친 획일화 문제에 대한 대응

4.1 지나친 획일화 문제

지나친 획일화(Over-smoothing) 문제란 그래프 신경망의 층의 수가 증가하면서 정점의 임베딩이 서로 유사해지는 현상을 의미합니다

지나친 획일화 문제는 작은 세상 효과와 관련이 있습니다
적은 수의 층으로도 다수의 정점에 의해 영향을 받게 됩니다



(a) 1-layer (b) 2-layer (c) 3-layer (d) 4-layer (e) 5-layer

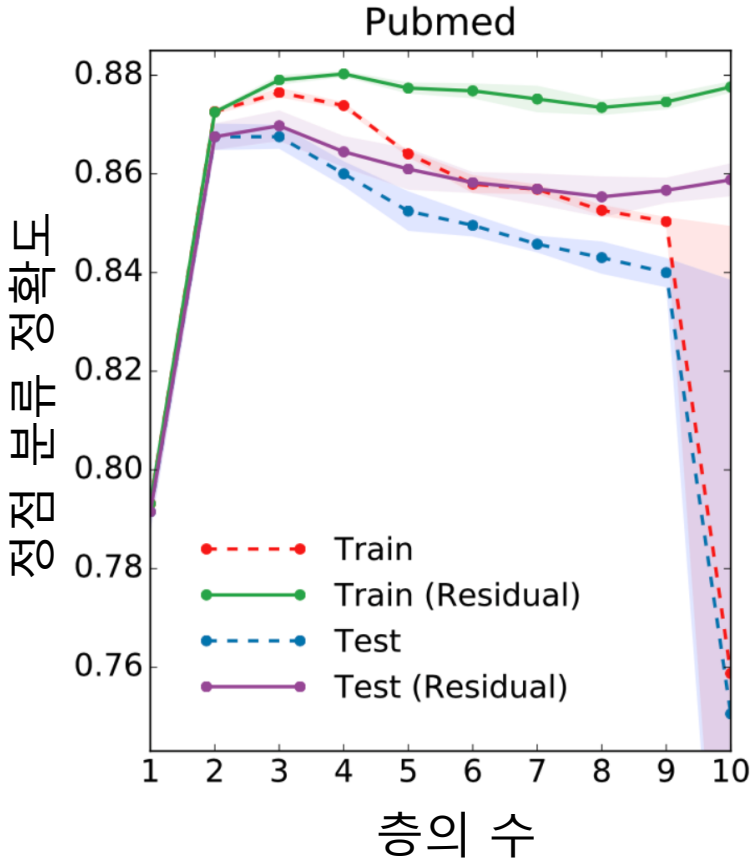
4.1 지나친 획일화 문제

지나친 획일화의 결과로 그래프 신경망의 층의 수를 늘렸을 때, 후속 과제에서의 정확도가 감소하는 현상이 발견되었습니다

오른쪽 그림에서 보듯이 그래프 신경망의 층이 2개 혹은 3개 일 때 정확도가 가장 높습니다

잔차항(Residual)을 넣는 것,
즉 이전 층의 임베딩을 한 번 더 더해주는 것 만으로는 효과가 제한적입니다

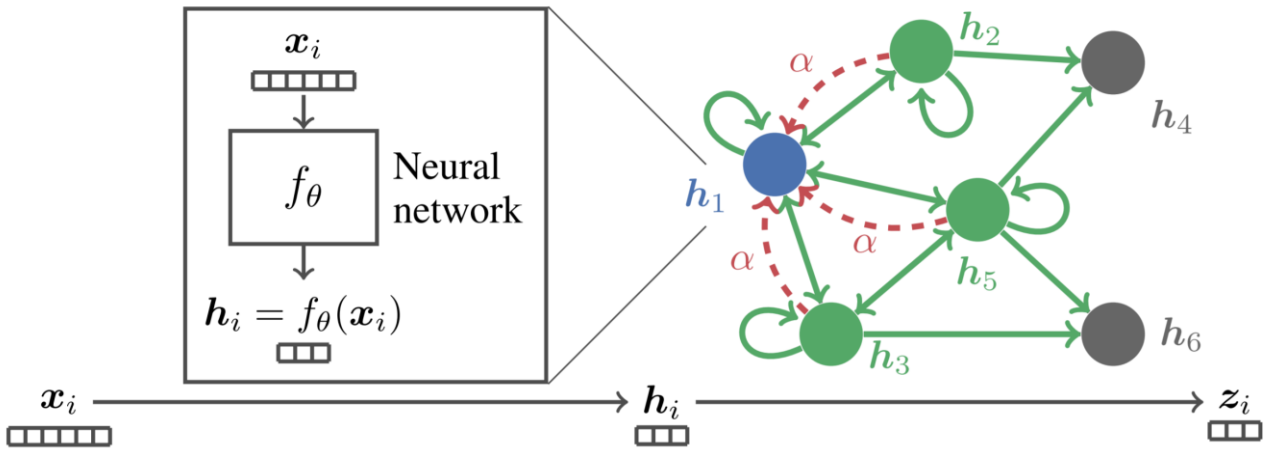
$$\mathbf{h}_u^{(l+1)} = \mathbf{h}_u^{(l+1)} + \mathbf{h}_u^{(l)}$$



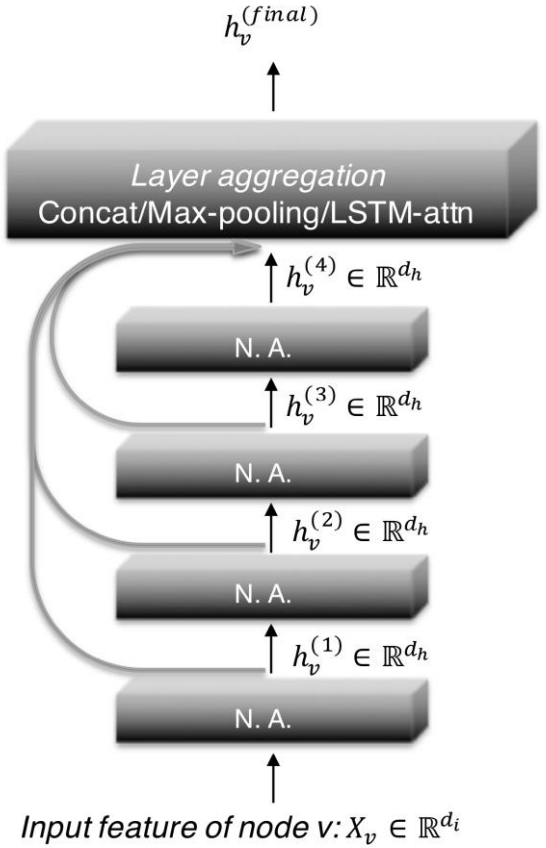
4.2 지나친 획일화 문제에 대한 대응

획일화 문제에 대한 대응으로 JK 네트워크(Jumping Knowledge Network)는 마지막 층의 임베딩 뿐 아니라, 모든 층의 임베딩을 함께 사용합니다

APPNP는 0번째 층을 제외하고는 신경망 없이 집계 함수를 단순화하였습니다



APPNP



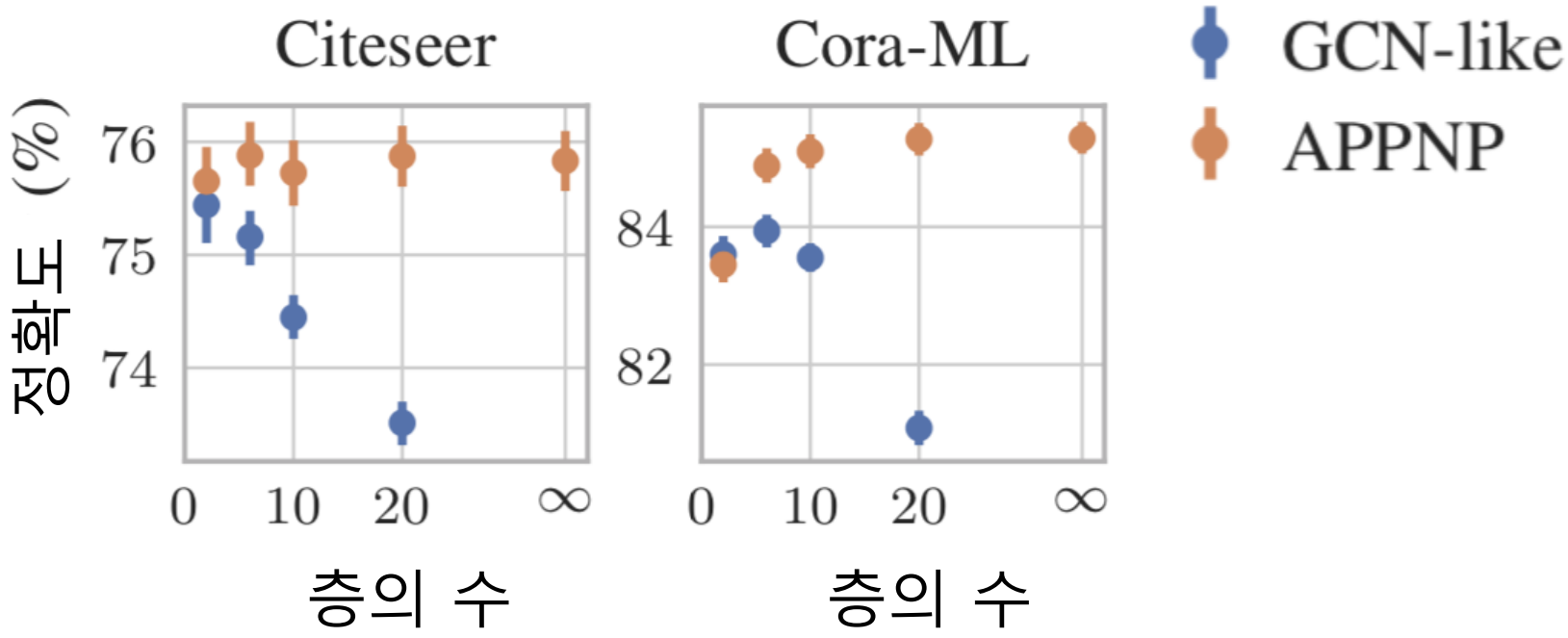
JK 네트워크

이미지 출처: https://arxiv.org/pdf/1806.03536.pdf, https://arxiv.org/pdf/1810.05997.pdf

4.2 지나친 획일화 문제에 대한 대응

APPNP의 경우, 층의 수 증가에 따른 정확도 감소 효과가 없는 것을 확인했습니다

후속 과제로는 정점 분류가 사용되었습니다



5. 그래프 데이터의 증강

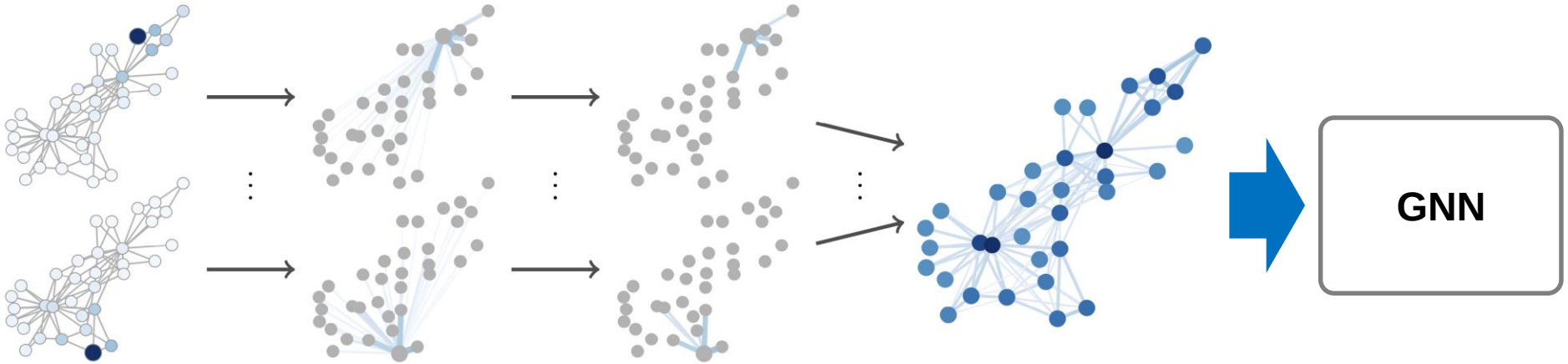
5.1 그래프 데이터 증강

5.2 그래프 데이터 증강에 따른 효과

5.1 그래프 데이터 증강

데이터 증강(Data Augmentation)은 다양한 기계학습 문제에서 효과적입니다

그래프에도 누락되거나 부정확한 간선이 있을 수 있고, 데이터 증강을 통해 보완할 수 있습니다
임의 보행을 통해 정점간 유사도를 계산하고,
유사도가 높은 정점 간의 간선을 추가하는 방법이 제안되었습니다



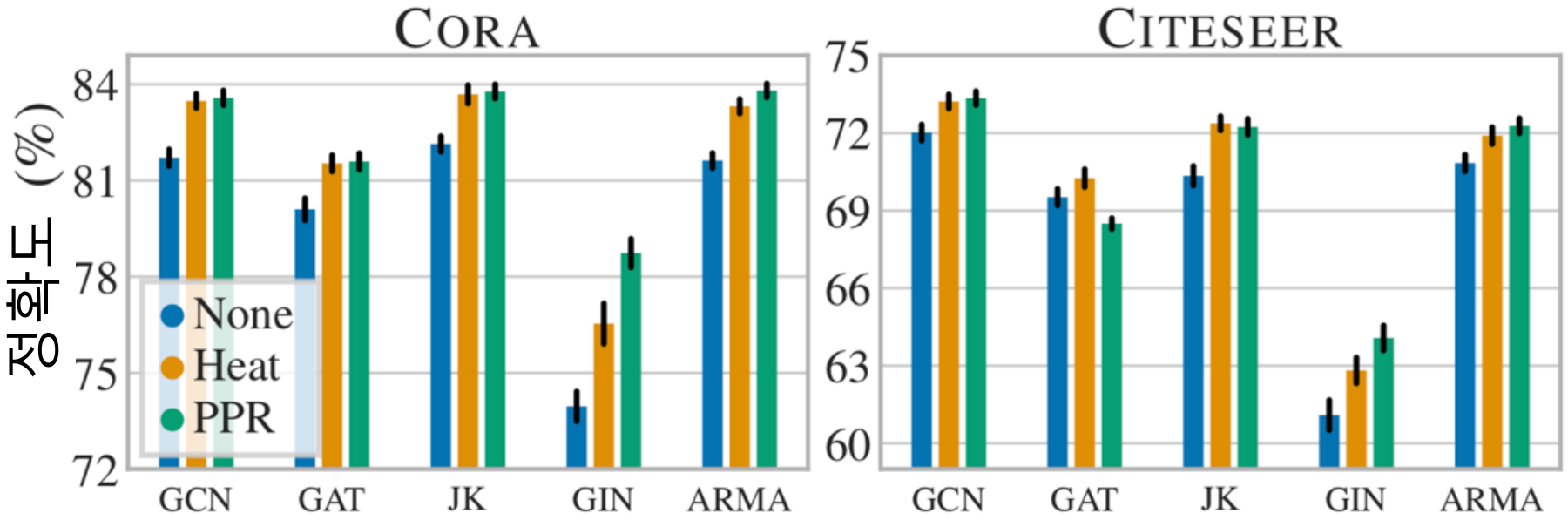
정점간 유사도 계산 × **유사도를 사용하여 필터링** **간선이 추가된 그래프**

이미지 출처: https://arxiv.org/abs/1911.05485

5.2 그래프 데이터 증강의 효과

그래프 데이터 증강의 결과 정점 분류의 정확도가 개선되는 것을 확인했습니다

아래 그림의 HEAT과 PPR은 제안된 그래프 데이터 증강 기법을 의미합니다



이미지 출처: https://arxiv.org/abs/1911.05485

6. 실습: GraphSAGE의 집계 함수 구현

5.1 데이터 불러오기
5.2 GraphSAGE 구현
5.3 GraphSAGE 학습
5.4 GraphSAGE 평가

6.1 데이터 불러오기

필요한 라이브러리를 불러옵니다

```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import dgl
from dgl.data import CoraGraphDataset
from sklearn.metrics import f1_score
import dgl.function as fn
```

6.1 데이터 불러오기

실습에 사용할 Cora 인용 그래프를 불러옵니다

Cora 데이터셋은 2,708개의 정점(논문)과, 10,556개의 간선(인용 관계)로 구성됩니다
각 정점은 1,433개의 속성을 가지며, 이는 1433개의 단어의 등장 여부를 의미합니다
각 정점은 7개의 유형 중 하나를 가지며, 대응되는 논문의 주제를 의미합니다
학습은 140개의 정점을 사용해 이루어집니다

```
G = CoraGraphDataset()
numClasses = G.num_classes
G = G[0]
features = G.ndata['feat']
inputFeatureDim = features.shape[1]
labels = G.ndata['label']
trainMask = G.ndata['train_mask']
testMask = G.ndata['test_mask']
```

```
Finished data loading and preprocessing.
  NumNodes: 2708
  NumEdges: 10556
  NumFeats: 1433
  NumClasses: 7
  NumTrainingSamples: 140
  NumValidationSamples: 500
  NumTestSamples: 1000
```

×

6.2 GraphSAGE 구현

GraphSAGE 구현의 첫 단계로 GraphSAGE의 집계 함수를 직접 구현합니다

AGG 함수로는 평균을 사용합니다

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \cdot \left[\mathrm{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right), \mathbf{h}_v^{k-1}\right]\right)$$

자신의 임베딩과 이웃의 임베딩을 연결

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

6.2 GraphSAGE 구현

GraphSAGE 구현의 첫 단계로 GraphSAGE의 집계 함수를 직접 구현합니다

```
class SAGEConv(nn.Module):

    def __init__(self, in_feats, out_feats, activation):
        super(SAGEConv, self).__init__()
        self._in_feats = in_feats
        self._out_feats = out_feats
        self.activation = activation
        self.W = nn.Linear(in_feats+in_feats, out_feats, bias=True)
```

6.2 GraphSAGE 구현

GraphSAGE 구현의 첫 단계로 GraphSAGE의 집계 함수를 직접 구현합니다

```
class SAGEConv(nn.Module):
    ...
    def forward(self, graph, feature):
        graph.ndata['h'] = feature
        graph.update_all(fn.copy_src('h', 'm'), fn.sum('m', 'neigh'))
        degs = graph.in_degrees().to(feature)
        hkNeigh =  graph.ndata['neigh'] /  degs.unsqueeze(-1)
        hk = self.W(torch.cat((graph.ndata['h'], hkNeigh), dim=-1))
        if self.activation != None:
            hk = self.activation(hk)
        return hk
```

6.2 GraphSAGE 구현

위에서 구현한 집계 함수를 이용하여 각 층을 정의합니다

```
class GraphSAGE(nn.Module):

    def __init__(self, graph, inFeatDim, numHiddenDim, numClasses, numLayers, activationFunction):
        super(GraphSAGE, self).__init__()
        self.layers = nn.ModuleList()
        self.graph = graph
        self.layers.append(SAGEConv(inFeatDim, numHiddenDim, activationFunction))
        for i in range(numLayers):
            self.layers.append(SAGEConv(numHiddenDim, numHiddenDim, activationFunction))
        self.layers.append(SAGEConv(numHiddenDim, numClasses, activation=None))
```

6.2 GraphSAGE 구현

GraphSAGE 구현의 다음 단계로 순전파(Forward Propagation)을 정의합니다

```
class GraphSAGE(nn.Module):
    def __init__(self,graph, inFeatDim, numHiddenDim, numClasses, numLayers,
        ...

    def forward(self, features):
        x = features
        for layer in self.layers:
            x = layer(self.graph, x)
        return x
```

```
model = GraphSAGE(G, inputFeatureDim, numHiddenDim, numClasses, numLayers, F.relu)
```

6.3 GraphSAGE 학습

역전파(Backpropagation)을 통해 GraphSAGE를 학습합니다

```
def train(model, lossFunction, features, labels, trainMask, optimizer, numEpochs):
    for epoch in range(numEpochs):
        model.train()
        logits = model(features)
        loss = lossFunction(logits[trainMask], labels[trainMask])
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
lossFunction = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learningRate, weight_decay=weightDecay)
train(model, lossFunction, features, labels, trainMask, optimizer, numEpochs)
```

6.4 GraphSAGE 평가

정점 분류의 성능을 평가합니다

```
def evaluateTest(model, features, labels, mask):
    model.eval()
    with torch.no_grad():
        logits = model(features)
        logits = logits[mask]
        labels = labels[mask]
        _, indices = torch.max(logits, dim=1)
        macro_f1 = f1_score(labels, indices, average = 'macro')
        correct = torch.sum(indices == labels)
        return correct.item() * 1.0 / len(labels), macro_f1
```

6.4 GraphSAGE 평가

정점 분류의 성능을 평가합니다

```
def test(model, feautures, labels, testMask):
    acc, macro_f1 = evaluateTest(model, features, labels, testMask)

test(model, features, labels, testMask)
```

10강 정리

1. 그래프 신경망 복습

2. 그래프 신경망에서의 어텐션

- 그래프 어텐션 신경망은 이웃 정점들의 임베딩을 평균내는 과정에서의 가중치도 함께 학습함

3. 그래프 표현 학습과 그래프 풀링

- 정점 임베딩으로부터 그래프 풀링을 통해 전체 그래프 임베딩, 즉 전체 그래프의 벡터 표현을 얻음

4. 지나친 획일화 문제

- 그래프 신경망의 층 수를 증가시킬 때, 정점 임베딩이 서로 유사해지고, 후속 과제의 정확도가 떨어지는 현상

5. 그래프 데이터 증강

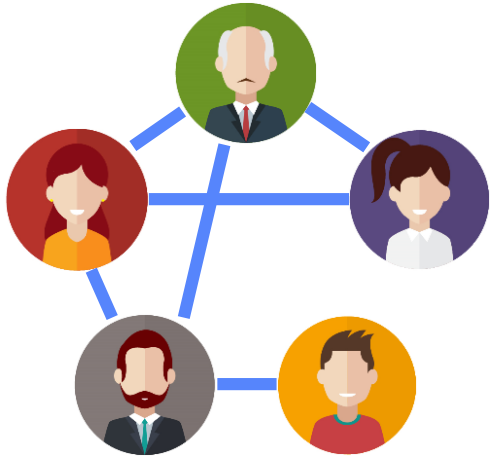
- 그래프에 간선을 추가한 뒤 그래프 신경망을 학습시키는 방법으로 후속 과제의 정확도가 향상 시킴

6. 실습: GraphSAGE의 집계 함수 구현

“그래프를 위한 기계 학습” 복습

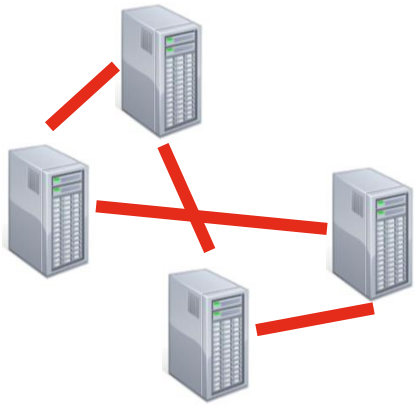
1강. 그래프란 무엇이고 왜 중요할까?

그래프는 복잡계를 효과적으로 표현하고 분석하기 위한 언어임을 배웠습니다

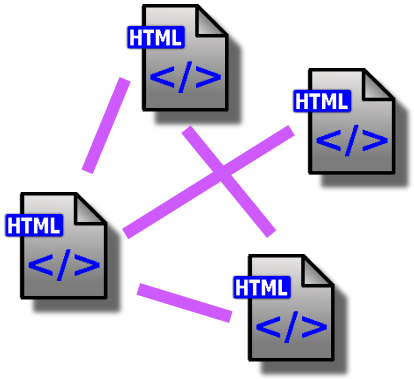


소셜 네트워크

전자상거래 구매 내역



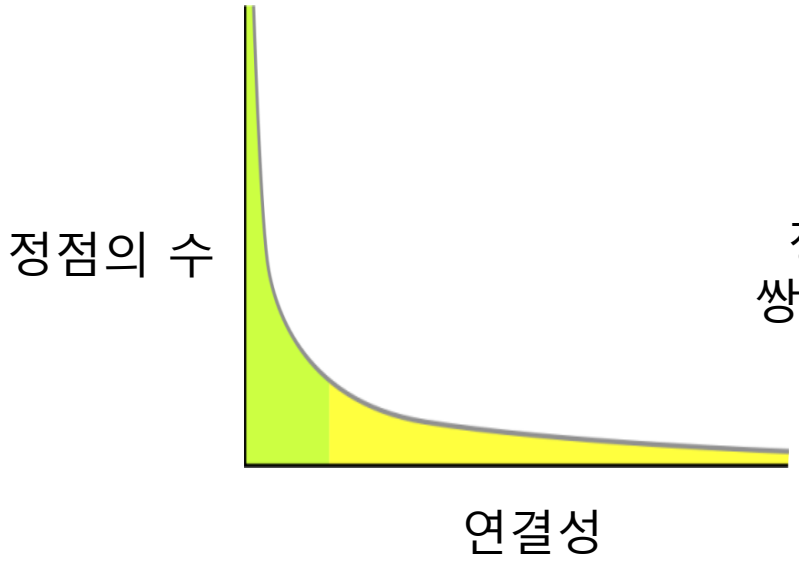
인터넷

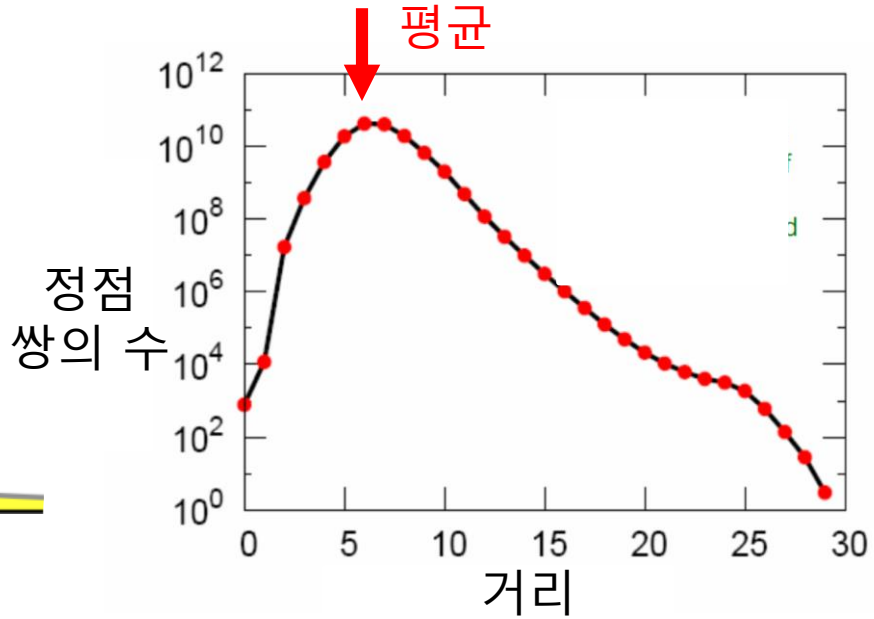


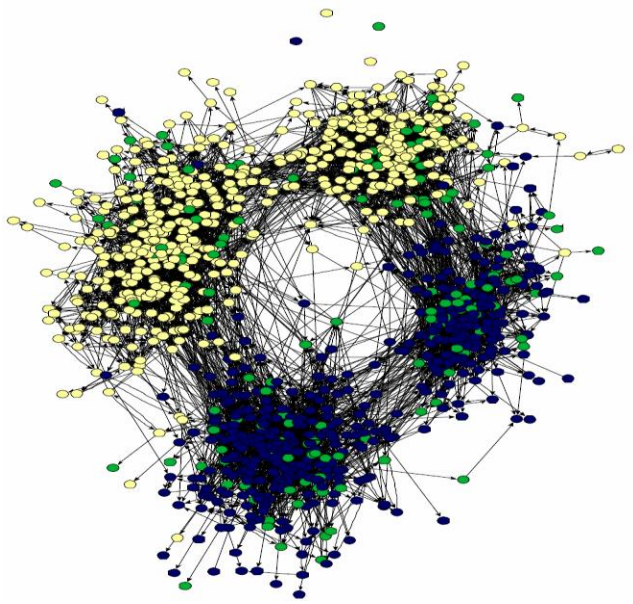
웹 그래프

2강. 실제 그래프는 어떻게 생겼을까?

랜덤 그래프와 구분되는 실제 그래프의 구조적 특성에 대해서 배웠습니다



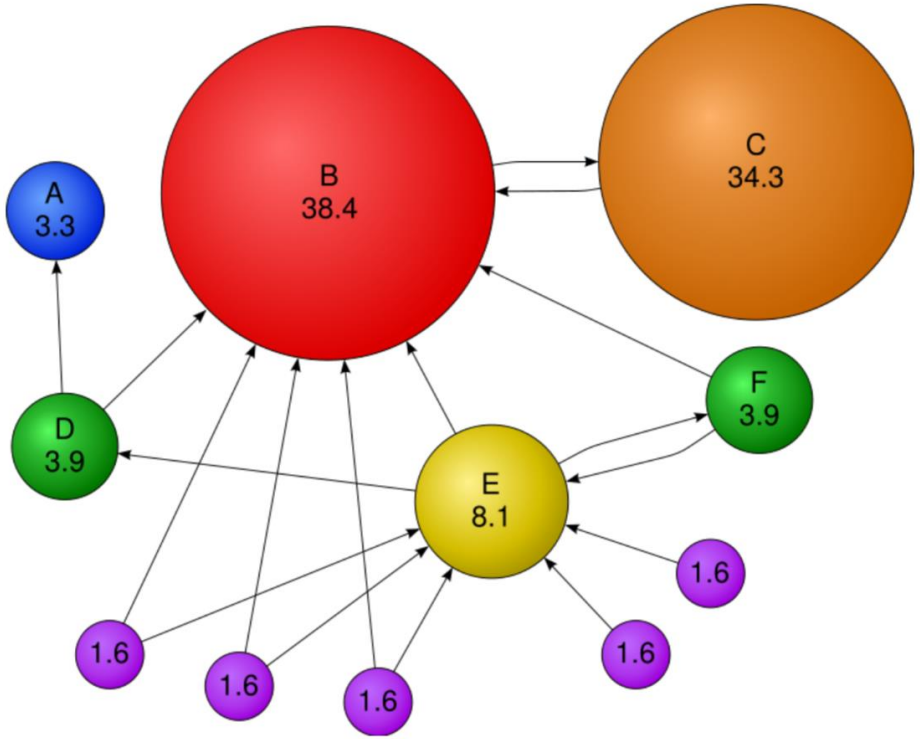




이미지 출처: https://web.stanford.edu/class/cs224w/slides/02-gnp-smallworld.pdf

3강. 검색 엔진에서는 그래프를 어떻게 활용할까?

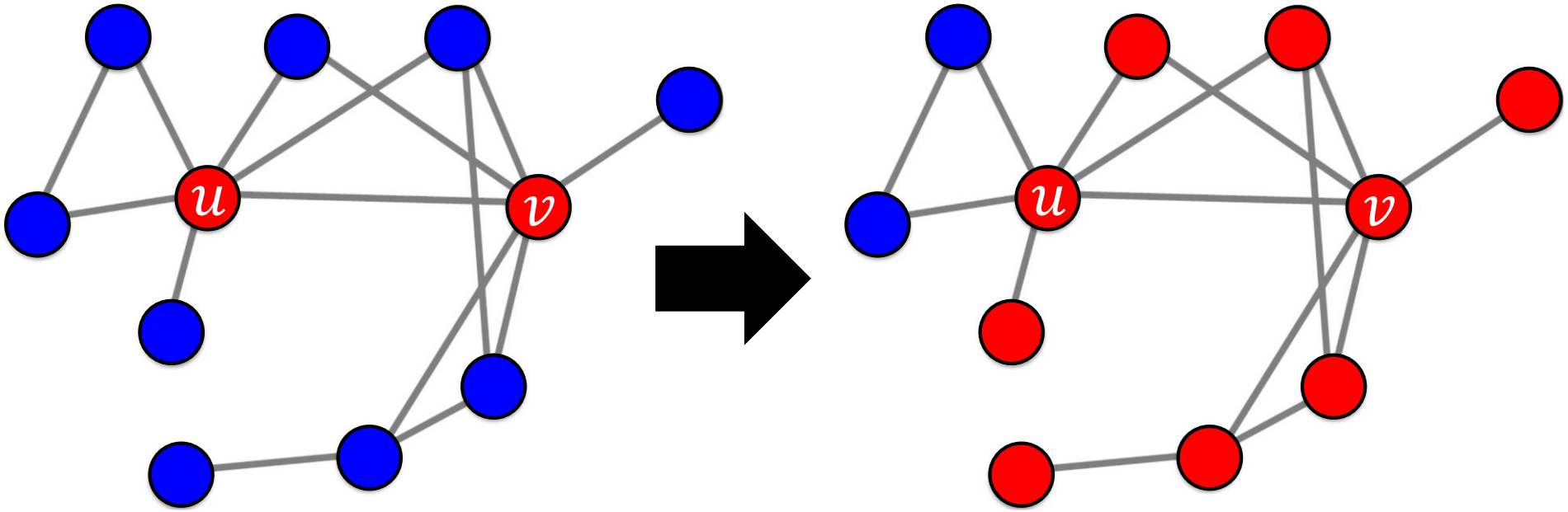
페이지랭크 알고리즘을 통해 웹페이지의 관련성과 신뢰도를 평가하는 방법을 배웠습니다



이미지 출처: http://infolab.stanford.edu/~ullman/mmds/ch5.pdf

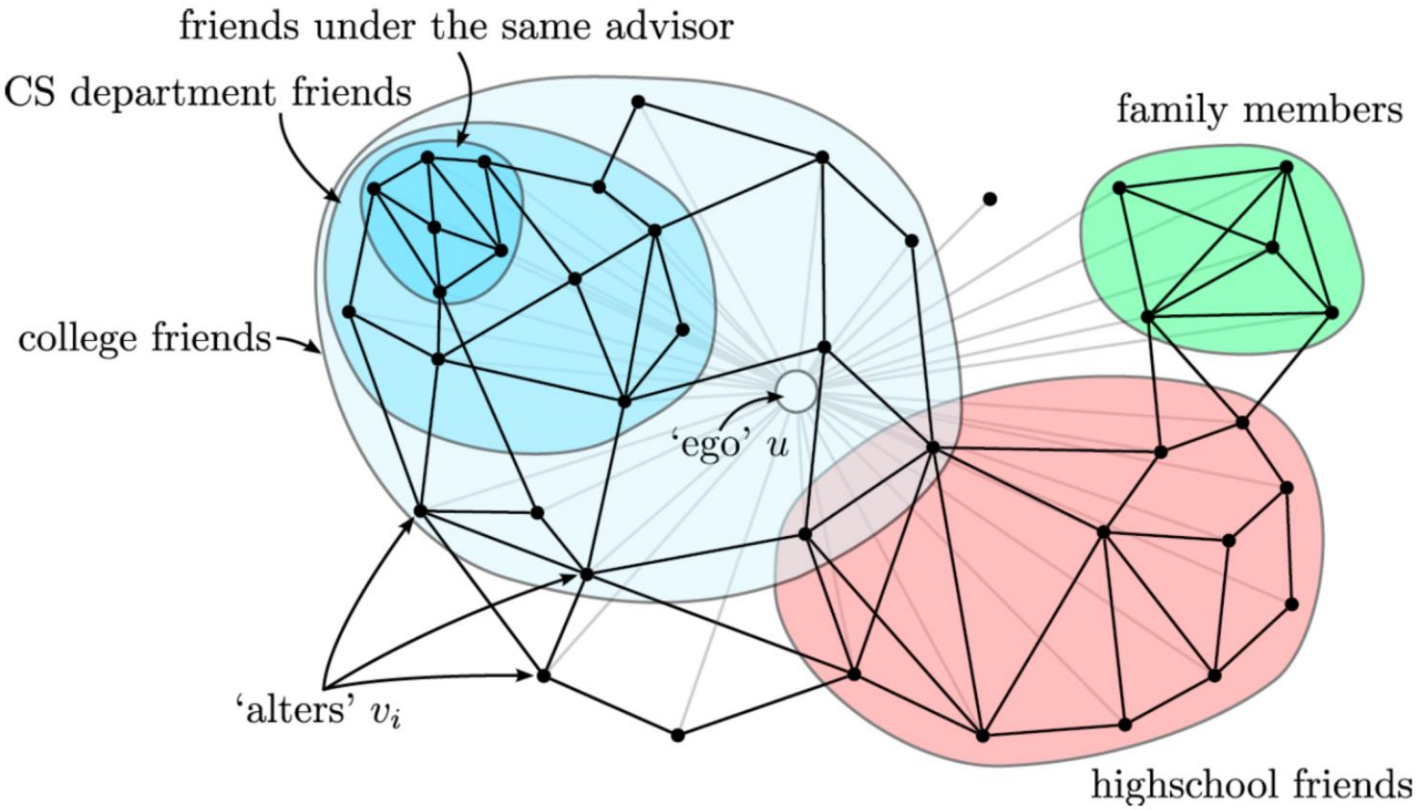
4강. 그래프를 바이럴 마케팅에 어떻게 활용할까?

그래프에서의 전파 과정을 모형화하고, 전파를 최대화하기 위한 방법을 배웠습니다



5강. 그래프의 구조를 어떻게 분석할까?

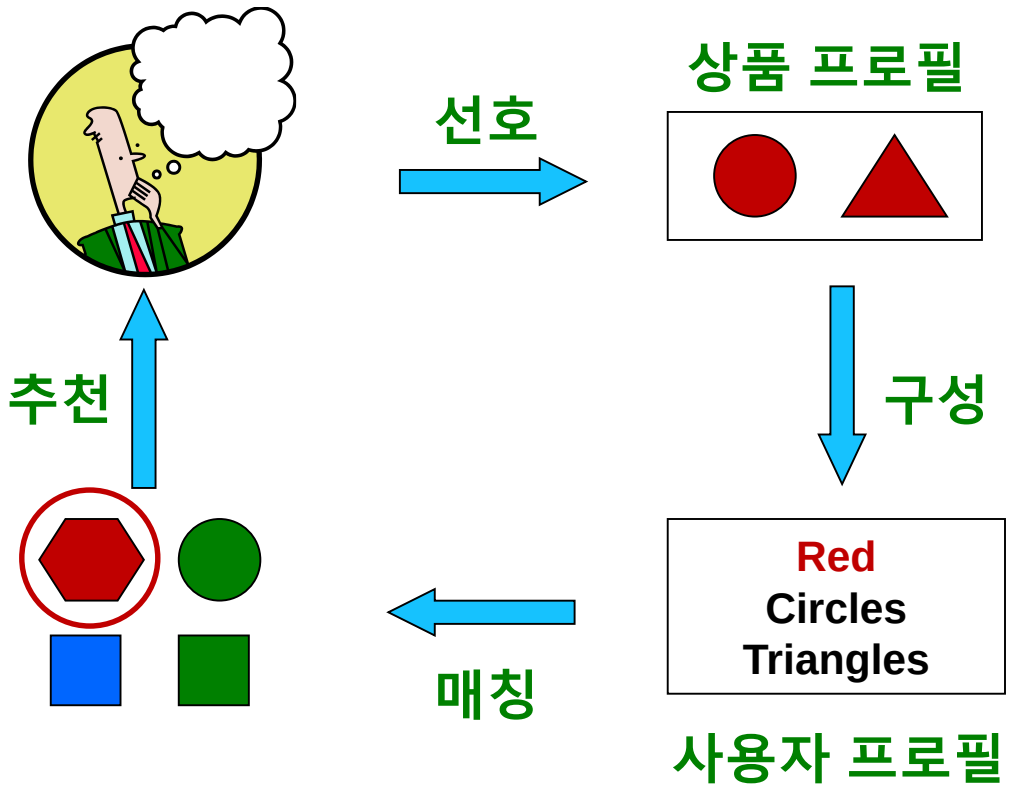
밀접하게 연결된 정점의 집합, 즉 군집을 찾는 방법에 대해서 공부했습니다

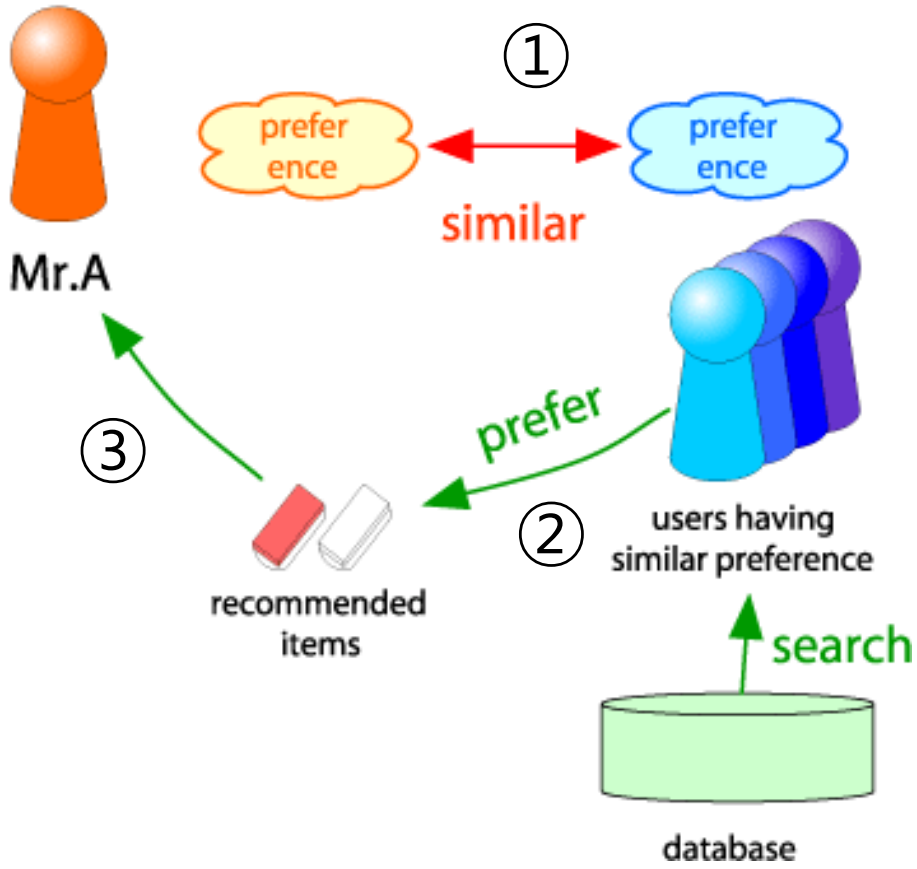


이미지 출처: https://arxiv.org/pdf/1210.8182.pdf

6강. 그래프를 추천시스템에 어떻게 활용할까? (기본)

내용 기반 추천시스템과 협업 필터링에 대해 배웠습니다

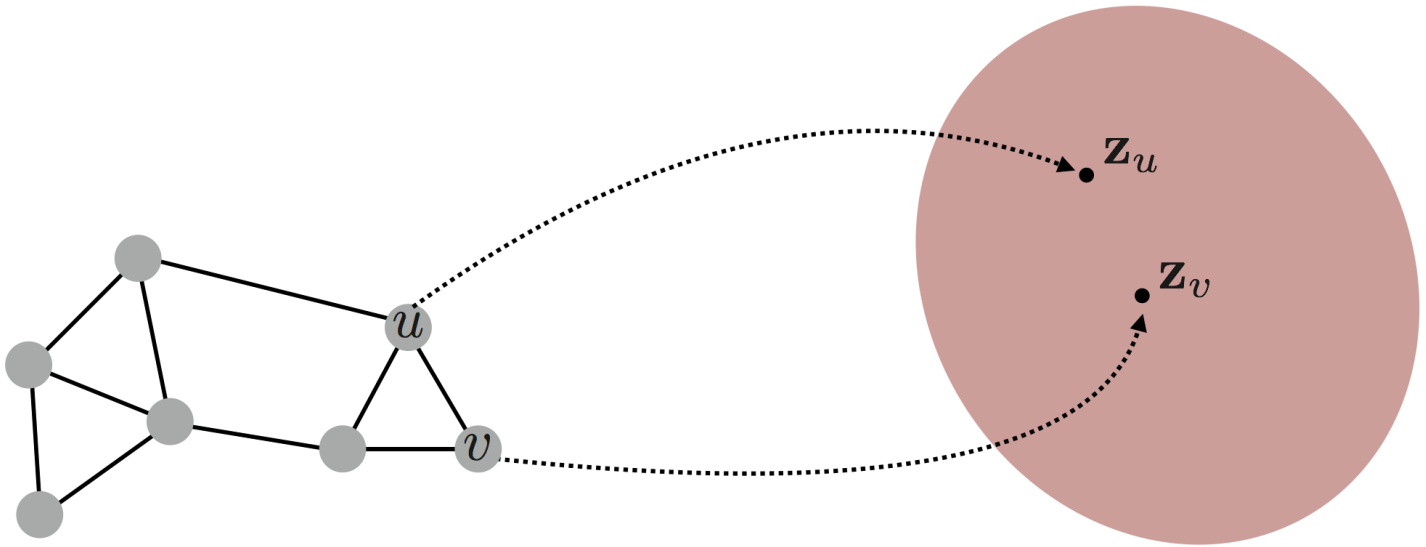




이미지 출처: http://www.mmds.org/mmds/v2.1/ch09-recsys1.pdf

7강. 그래프의 정점을 어떻게 벡터로 표현할까?

DeepWalk, Node2Vec 등 정점 표현 학습에 대해서 배웠습니다

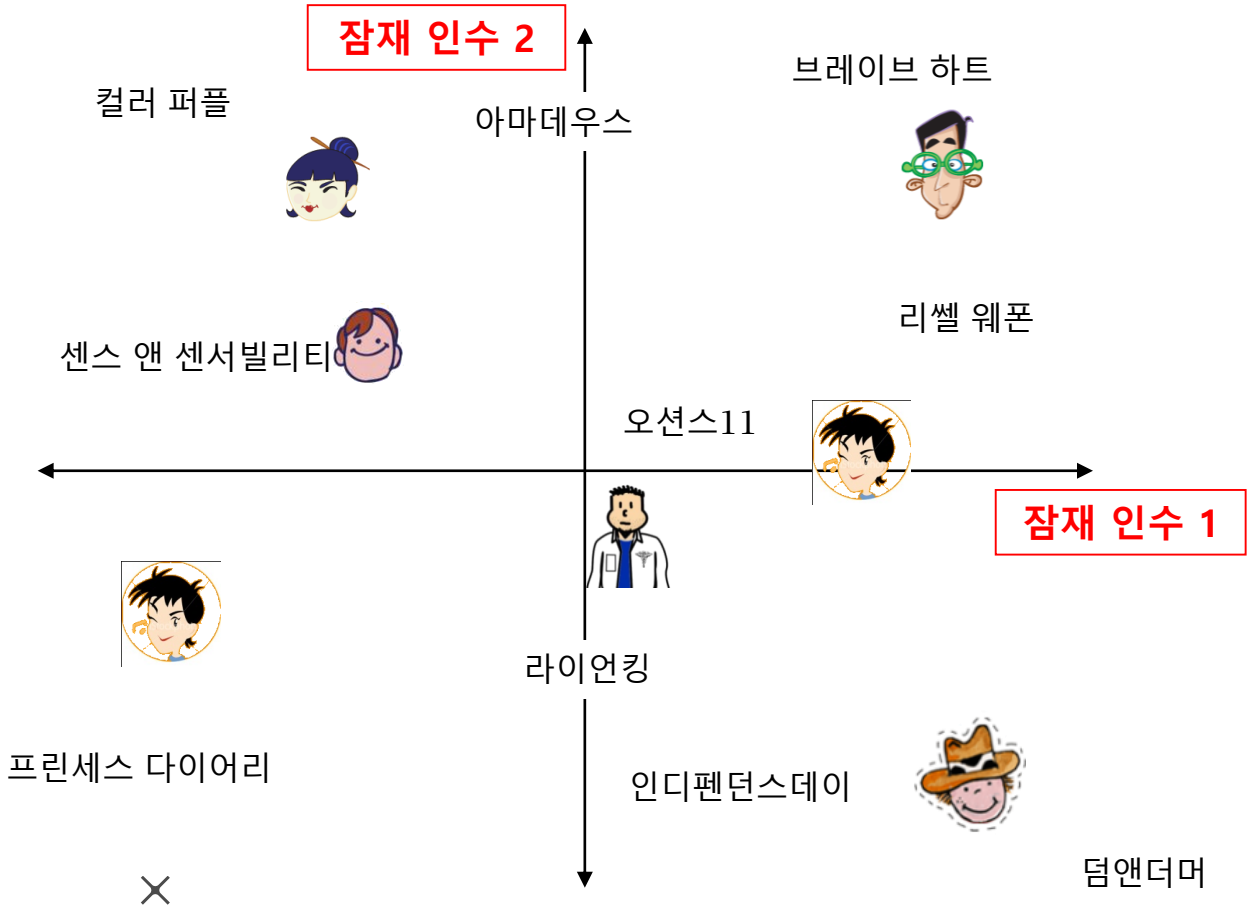


입력 그래프 임베딩 공간

이미지 출처: http://snap.stanford.edu/proj/embeddings-www/

8강. 그래프를 추천시스템에 어떻게 활용할까? (심화)

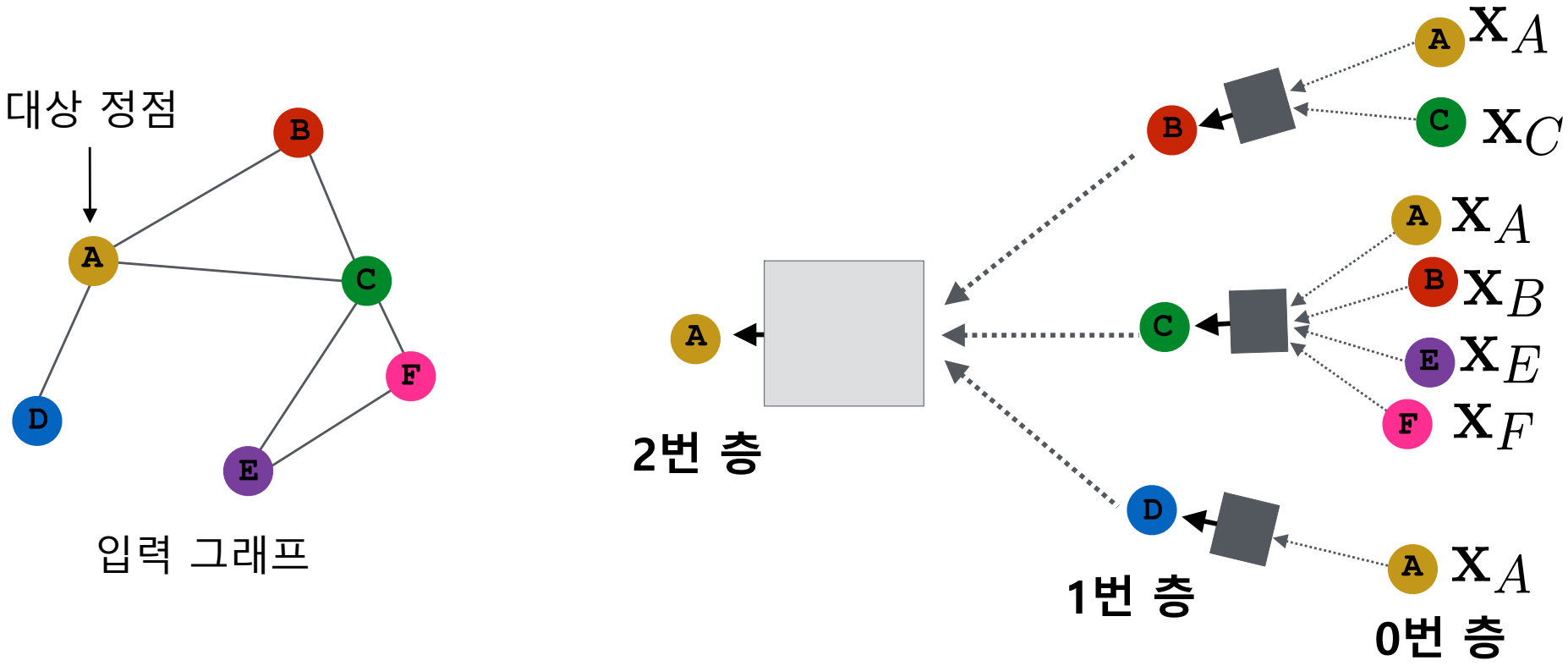
잠재 인수 모형과 넷플릭스 챌린지에 대해서 배웠습니다



이미지 출처: http://www.mmds.org/mmds/v2.1/ch09-recsys2.pdf

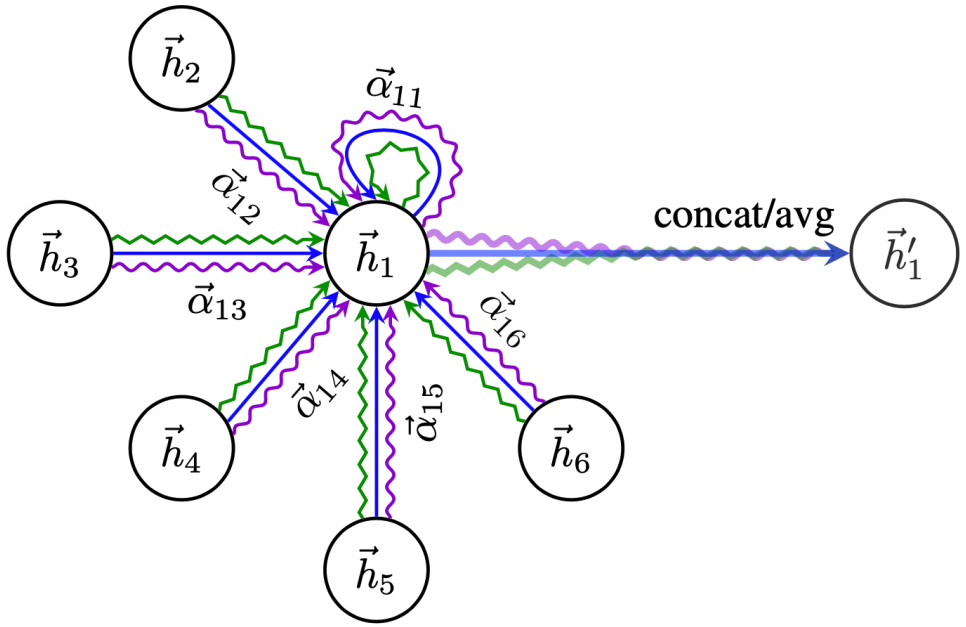
9강. 그래프 신경망이란 무엇일까? (기본)

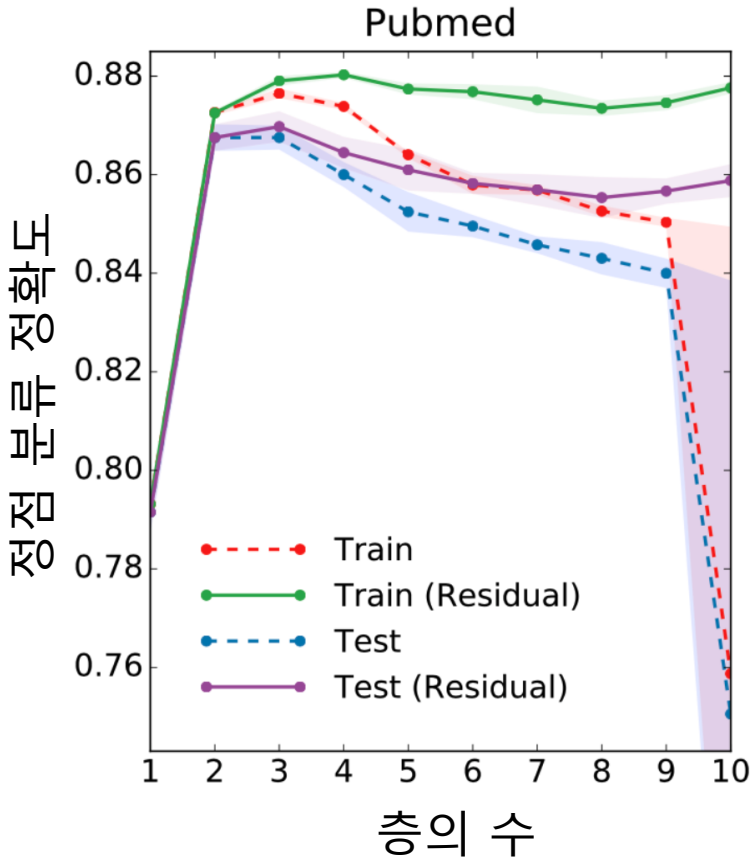
그래프 신경망의 구조, 학습, 활용에 대해서 배웠습니다

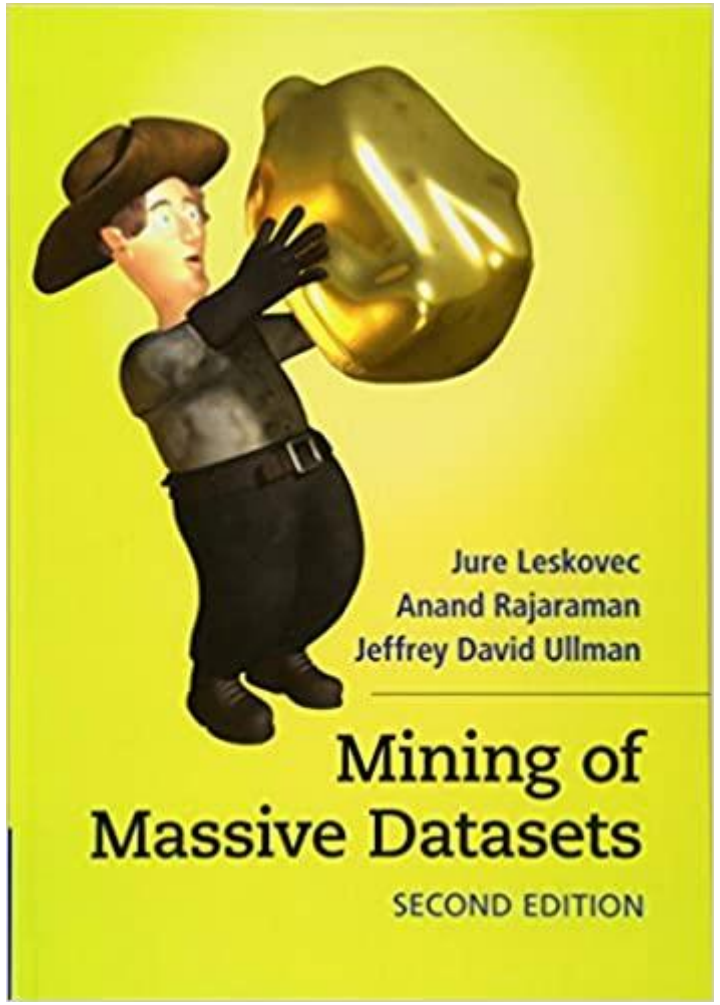


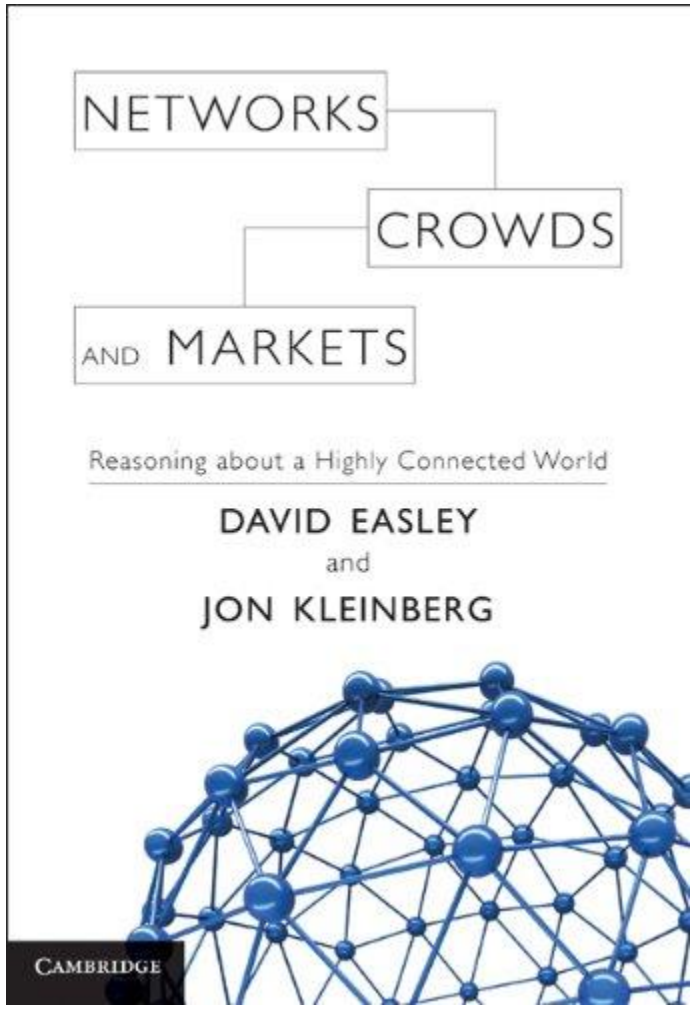
10강. 그래프 신경망이란 무엇일까? (심화)

그래프 신경망에서의 어텐션, 데이터 증강, 지나친 획일화 문제 등에 대해서 배웠습니다









CS224W | Content | Schedule | Course Info | Projects | Office Hours | FAQ | Gradescope | Piazza

CS224W: Machine Learning with Graphs

Stanford / Winter 2021

CS246 | Content | Schedule | Course Info | Office Hours | FAQ | Gradescope | Campuswire

CS246: Mining Massive Data Sets

Stanford / Winter 2020

수고하셨습니다!