Lab Study Report

배홍섭

Index

5. The Graph Neural Network Model

실제 실행 예제

이 장에서는 보다 복잡한 인코더 모델에 중점을 둡니다. 그래프 데이터에 대한 심층 신경망을 정의하기 위한 일반적인 프레임워크인 그래프 신경망(GNN) 형식주의를 소개합니다.

동기와 상관없이 GNN의 정의적 특징은 벡 터 메시지가 노드 간에 교환되고 신경망을 사용하여 업데이트되는 신경 메시지 전달 형식을 사용한다는 것입니다

이 장의 나머지 부분에서는 이 신경 메시지 전달 프레임워크의 기초를 자세히 설명합니다. 메시지 전달 프레 임워크 자체에 초점을 맞추고 GNN 모델 교육 및 최적화에 대한 논의는 6장으로 미루겠습니다.

실제 실행 예제

여기서 UPDATE 및 AGGREGATE 는 임의의 미분 가능한 함수(즉, 신경망)이고 mN(u) 는 u의 그래프 이웃 N(u)에서 집계된 "메시지"입니다

지금까지 UPDATE 및 AGGREGATE 함수 를 사용하여 일련의 메시지 전달 반복으로서 GNN 프레임워크를 비교적 추상적인 방식으로 논의했습니다.

이제는 조금 더 구체적으로 알아보자

실제 실행 예제

The Basic GNN
- 1. aggregation 더 자세하게 살펴보자

5.2.1 이웃 정규화 가장 기본적인 이웃 집계 연산(방정식 5.8)은 단순히 이웃 임베딩의 합을 취합니다. 이 접근 방식의 한 가지 문제는 불안정하고 노 드 수준에 매우 민감할 수 있다는 것입니다.
-> 이 문제에 대한 한 가지 해결책은 관련된 노드의 정도에 따라 집계 작업을 단순히 정규화 하는 것입니다. 가장 간단한 방법은 합계가 아닌 평균을 취하는 것입니다.
-> GCN

다른방법 : 이웃 집계 연산(단순 합 x) = pooling
Janossy pooling

실제 실행 예제

5.2.3 Neighborhood Attention : 이웃 중요도 반영

실제 실행 예제

5.3 over-smooth 과도한 평활화 (정규화 너무 많이함)

- 따라서 RNN gate에서 영감을 받아 정보 손실없이 보존하기 시작함.
- RNN gate란 거의 건너뛰기 & 연결

5.4. Edge Features and Multi-relational GNNs

05| Edge Features and Multi-relational GNNs

Graph data가 Multi-relational 이거나 edge(link) feature 데이터일 경우 인기있는 approach를 알아보자.

5.4.1. Relational Graph Neural Networks

05| Relational Graph Neural Networks

- RGCN (Relational Graph Convolution Network)
  - relation type(τ) 별로 transformation matrix 지정
  - aggregation function이 여러 유형의 relation을 수용할 수 있음.

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \frac{W_\tau h_v}{f_n(N(u), N(v))}$$

  - 이때 f_n 은 정규화 함수(normalization function)
  - W_τ : $weight$
  - h_v : $node\ v(node\ u's\ neighborhood)$의 $hidden\ embedding$
  - $N(u)$: u의 $graph\ neighborhood$

  - 단점 :
    relation type 별로 transformation matrix 존재(=학습 될 행렬, weight)
    따라서 weight수가 매우 많음(= overfitting & time consuming)

따라서 Schlichtkrull et al. [2017] 가 제안한 parameter sharing 사용.

05| Relational Graph Neural Networks

- Parameter sharing

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \frac{W_\tau h_v}{f_n(N(u), N(v))}$$

$$W_\tau = \sum_{i=1}^{b} \alpha_{i,\tau} B_i$$

- 각 $relation(\tau)$ 의 특정 $parameters$는 b $combination$ $weights(\alpha_{1,\tau}, \alpha_{2,\tau}, \dots, \alpha_{b,\tau})$이다.
- Parameter sharing에 쓰이는 weight(W_τ)는 b개의 relation의 각각 weight * relation matrix의 합이다.

- Rewrite the full aggregation function

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \frac{\alpha_\tau \times_1 \mathcal{B} \times_2 h_v}{f_n(N(u), N(v))}$$

- W_τ 에 위의 수식 대입.
- $\alpha_\tau = (\alpha_{1,\tau}, \alpha_{2,\tau}, \dots, \alpha_{b,\tau})$는 $relation\ \tau$ 에 대한 $combination(B_i)$ $weights(\alpha_{i,\tau})$를 포함한 $vector$.
  즉 $\alpha_{1,\tau}$는 스칼라 값.
- $\mathcal{B} = (B_1, B_2, \dots, B_b)$ 를 $stacking$ 하여 만든 $tensor$
- $\times_i$ 는 $mode\ i$ 일때 $tensor$ 곱

결론 : parameter sharing은 각 relation에 대해 embedding을 학습할 뿐만 아니라 모든 relations에서 공유되는 tensor 또한 학습한다.

05| Extensions and variations

- Relational Graph Neural Networks

  - relation 별 aggregation matrix를 따로 정의하는 접근 방식.
  - parameter sharing이 없는 접근방식을 의미.

- RGCN에 Attention개념을 도입한 방법론에 대해 알아보자.
  - [Teru et al., 2020].

5.4.2. Attention and Feature Concatenation

05| Attention and Feature Concatenation

- Relational GNN(no parameter sharing)
  - multi-relational graph나 edge features에 적용가능.

- Multi-relational graph / edge features information을 message passing과정에서 neighbor embeddings와 연결하는 새 Aggregation function을 정의.

$$m_{N(u)} = AGGREGATE(\{h_v \oplus e_{(u,\tau,v)}, \forall v \in N(u)\})$$

  - $e_{(u,\tau,v)}$ 는 $edge(u,\tau,v)$의 임의의 $vector\ value$를 가진 $feature$
  - $node$: u , v
  - $\mathrm{relation}$: τ

결론 : 이 방법은 매우 간단하고, 최근 attention-based 접근 방식으로 크게 성공함
base aggregation function [Sinha et al., 2019].

해당 파트에서는 간단하게 Attention과 결합한 GNN 방식도 있음을 간략하게 언급만 하고 넘어감.

5.5. Graph Pooling

05| Graph Pooling

- $Graph\ pooling$
  - 목표 : 그래프 전체 수준 embedding을 학습
  - 따라서 node embedding을 같이 pool하는 작업을 graph pooling이라고 한다.

- Pooling 접근방식

$$Z_G = \frac{\sum_{v \in V} Z_u}{f_n(|V|)}$$

  - 가장 쉬운 방식은 단순 합 또는 평균을 구하는 방법임.
  - Z_G : Graph embedding
  - Z_u : node u embedding

- LSTM & attention (다른 pooling 접근 방식)

$$q_t = LSTM(o_{t-1}, q_{t-1})$$
$$e_{v,t} = f_a(z_v, q_t), \forall v \in V$$
$$a_{v,t} = \frac{\exp(e_{v,i})}{\sum_{u \in V} \exp(e_{u,t})}, \forall v \in V$$
$$o_t = \sum_{v \in V} a_{v,t}, \quad {}_t z_v$$

05| Graph Pooling

- LSTM & attention (다른 pooling 접근 방식)

$$q_t = LSTM(o_{t-1}, q_{t-1})$$

- q_t는 각 반복 t에서 attention $query\ vector$.

$$e_{v,t} = f_a(z_v, q_t), \forall v \in V$$

- f_a 은 attention function
- $e_{v,t}$ 은 attention score

$$a_{v,t} = \frac{\exp(e_{v,i})}{\sum_{u \in V} \exp(e_{u,t})}, \forall v \in V$$

- $e_{v,t}$ $normalizeation$과정, exp: 지수함수

$$o_t = \sum_{v \in V} a_{v,t}\, z_v$$

- $weight\ sum(o_t)$을 구하고 이는 다시 $query\ vector\ update$에 사용된다. ($LSTM$구조)

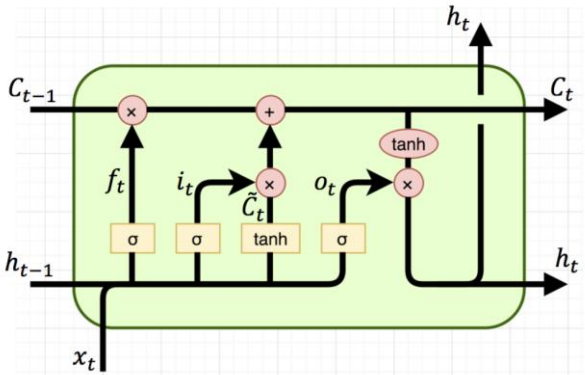
최종 그래프 $embedding$

$$Z_G = o_1 \oplus o_2 \oplus \cdots \oplus o_t$$

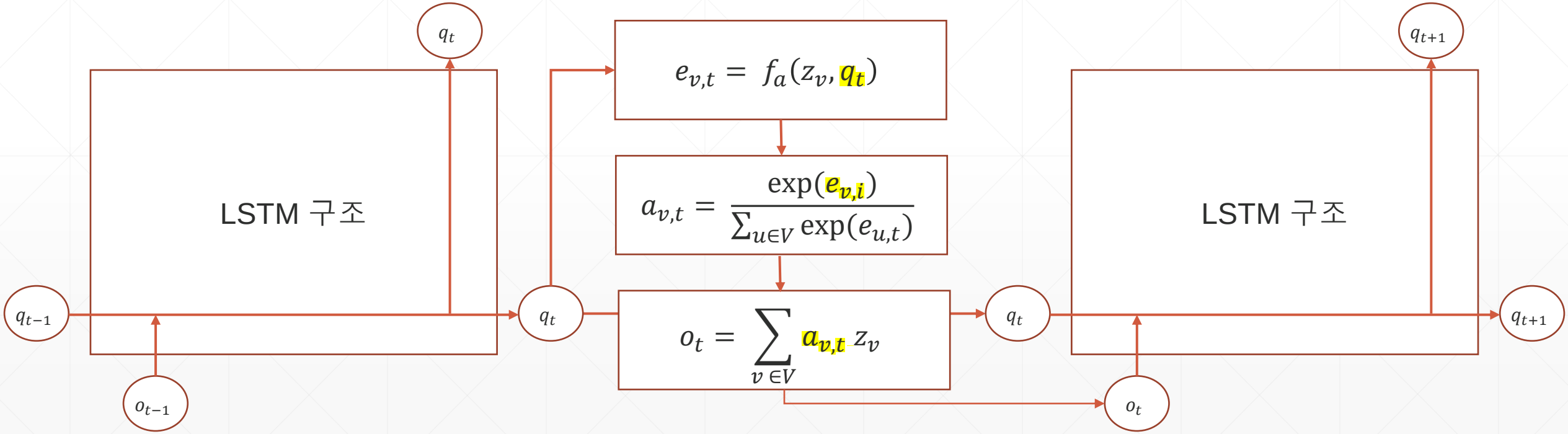
LSTM pooling approach :

$$q_t = LSTM(o_{t-1}, q_{t-1})$$
$$e_{v,t} = f_a(z_v, q_t), \forall v \in V$$
$$a_{v,t} = \frac{\exp(e_{v,i})}{\sum_{u \in V} \exp(e_{u,t})}, \forall v \in V$$
$$o_t = \sum_{v \in V} a_{v,t}, \quad {}_t z_v$$

05 Graph coarsening approaches



(a) Long Short-Term Memory



05| Graph coarsening approaches

- Pooling은 그래프 구조를 이용하지 않는 단점 존재(node embedding만 사용)
  - $node$ 표현 수단으로 $clustering\ or\ coarsening$ 사용

$$f_c \rightarrow G \times \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^{+|V| \times c}$$

  - f_c은 $clustering\ function$: 모든 $node$를 $c\ cluster$에 $mapping$
  - 출력값 : $assignment\ matrix\ S$
  - $S[u, c]$ 는 $node\ u$ 와 $cluster\ c$ 의 연관정도를 나타냄.

$$A^{new} = S^T A S$$

  - S를 이용하여 A(인접행렬)을 재정의
  - 따라서 A^{new}은 $cluster$ 간의 연관성$(= edge)$을 나타낸다.

$$X^{new} = S^T X$$

  - X는 $node\ feature\ matrix$
  - X^{new} 각 $cluster$에 할당된 모든 $node$들의 $aggregate$된 $embedding$을 나타낸다.

05| Graph coarsening approaches

- 그래프 전체 수준을 embedding 하기 위해 pooling 사용.
  - 그러나 그래프 구조는 이용을 하지 않는 단점 존재

- 해결책으로 clustering or coarsen 방법 사용
  - node와 cluster, cluster와 cluster간의 연관성(=그래프 구조)를 나타냄.
  - 하지만 학습 프로세스 종단 간 미분 가능하려면

f_c 가 미분가능해야하는데 이는 기존 클러스터링 알고리즘을 배제시키는 단점이 존재함.

5.6. Generalized Message Passing

05| Generalized Message Passing

- Message passing에서 node level 정보뿐만 아니라 edge, graph level 정보도 사용하도록 일반화할 수 있음.

$$h_{(u,v)}^{(k)} = UPDATE_{edge}(h_{(u,v)}^{(k-1)}, h_u^{(k-1)}, h_v^{(k-1)}, h_G^{(k-1)})$$
$$m_{N(u)} = AGGREGATE_{node}(\{h_{(u,v)}^{(k)} \forall v \in N(u)\})$$
$$h_u^{(k)} = UPDATE_{node}(h_u^{(k-1)}, m_{N(u)}, h_G^{(k-1)})$$
$$h_G^{(k)} = UPDATE_{graph}\left(h_G^{(k-1)}, \{h_u^{(k)}, \forall u \in V\}, \{h_{(u,v)}^{(k)} \forall (u,v) \in \varepsilon\}\right).$$

- edge, node, graph에 대해 embedding $h_{(u,v)}^{(k)}$ 생성.
- 따라서 edge,graph level features의 정보를 쉽게 사용할 수 있음.
- 다음과 같은 순서에 따라 update 진행.
- Message Passing은 앞에서 논의한 다양한 approach로 구현가능.
  예를 들면 graph-level update 시 pooling 사용

6. Graph Neural Networks in Practice

06| Graph Neural Networks in Practice

- 논의된 GNN 아키텍처가 어떻게 최적화 되는지
- 어떤 loss function을 사용하는지
- 어떤 정규화 기법이 사용되는지.

6.1. Applications and Loss Functions

06| Applications and Loss Functions

- GNN은 node classification, graph classification, edge prediction에 사용된다.

  Node classification : 소셜 네트워크에서 봇인지 여부 예측
  Graph classification : 분자 그래프에서 속성 예측(무슨 냄새?)
  Edge prediction : 온라인 플랫폼 콘텐츠 추천

- z_u : $node\ embedding. by\ final\ layer\ of\ aGNN$

- 구체적인 loss function
- 어떻게 비지도학습(unsupervised) 방식으로 pre-trained 될 수 있는지

6.1.1. Loss function for each level

06| Loss function for each level

- Node Classification Loss function :

$$L = \sum_{u \in V_{train}} -\log\left(softmax(z_u, y_u)\right).$$

- y_u 는 $node\ u$의 $class$를 나타내는 $one-hot\ vector$
  (예시. y는 feature 중 논문을 뜻하고, 그 중 주제 u)
- $softmax$를 사용하여 $node\ u$가 $class\ y_u$에 속할 예측 확률을 계산함.

- Graph Classification Loss function :

$$L = \sum_{G_i \in T} ||MLP(Z_{G_i}) - y_{G_i}||.$$

- MLP(Multi-Layer Perceptron) : output이 하나인(univariate) densely connected NN
- y_{G_i} : $training\ graph\ G_i$의 $target\ value$ (=정답)

- Edge prediction Loss function :

- 3,4 장에서 설명된 pairwise node embedding loss function 사용(Encoder-Decoder model)

6.1.4. Pre-training GNNs

06| Pre-training GNNs

- $Veli\check{c}kovi\acute{c}\ et\ al.$ [2019] 에 따르면 $random\ initialized$ GNN이 $neighborhood\ reconstruction\ loss$를 사용한 $pre-train$과 비교했을 때, 동등하거나 더 뛰어남을 입증함.

- $DGI(Deep\ Graph\ Informax)$
  - node embedding과 graph embedding의 상호 정보를 maximaize 하는 pre-train strategies.
  - neighborhood reconstruction loss에 비해 성능 개선 효과가 있는 strategies.

$$L = -\sum_{u \in V_{train}} \mathbb{E}_G \log\big(D(z_u, z_G)\big) + \gamma \mathbb{E}_{\tilde{G}} \log\big(1 - D(\tilde{z}_u, z_G)\big).$$

- z_u 는 $node\ u\ embedding(graph\ G$에 의해 생성된$)$
- $\tilde{z}_u$ 는 from $\tilde{G}(corrupted\ version\ of\ graph)$: 일부러 오차값(corrupted) 생성
- $D(Discriminator\ function)$: $node\ embedding$이 G에 의한 것인지 $\tilde{G}$에 의한 것인지 예측 & 학습
- 이러한 방식은 $node\ embedding$ 이 $real\ graph(G)$에 의한 것인지, $corrupted(\tilde{G})$ 에 의한 것인지 구분할 수 있게 해준다.

  즉 $node\ embedding$과 $graph-level\ embedding$이 밀접한 관계에 있음을 보여준다.

- 이러한 pre-train approach는 supervised training에서 보조 loss값으로 사용되기도 한다.
- 또한 pre-train approach 개발은 아직 활발한 연구 영역이다.

6.2. Efficiency Concerns and Node Sampling

06| Efficiency Concerns and Node Sampling

- 여러 노드가 이웃을 공유할 경우, 그래프의 모든 노드에 대해 독립적으로 message passing을 구현하면 중복 계산될 수 있음.

- 따라서 희소 행렬 곱셈을 기반으로 message passing 을 구현(Graph-level)

$$H^{(k)} = \sigma\left(AH^{(k-1)}W_{neigh}^{(k)} + H^{(k-1)}W_{self}^{(k)}\right),$$

  - $H^{(k)}$ 는 그래프 내의 모든 node의 layer(k번째) embedding을 포함하는 행렬
  - $H^{(k)}$ 는 각 $node$에 대한 $embedding$ H를 정확히 한번만 계산되면 되기 때문에 중복 계산 문제점을 피할 수 있음.
  - BUT, 전체 그래프와 node feature를 동시에 작업해야 하므로 메모리 문제 발생.

- Subsampling and Mini-Batching 사용(메모리 문제 해결을 위한 방법)
  - BUT, 임의의 node subsets으로 나누면 이 subsets의 사이의 edge가 사라짐은 피할 수 없고, 따라서 연결된 전체 그래프가 생성된다는 보장이 없으므로 모델 성능에 악영향 미칠 수 있음
  - Solution : 첫번째 node subsets을 선택한 후, 이 노드들의 neighborhood을 재귀적으로 샘플링
    그래프의 연결성(connectivity) 유지 가능 (참고 : $Hamilton\ et\ al.$ [2017b])

6.3. Parameter Sharing and Regularization

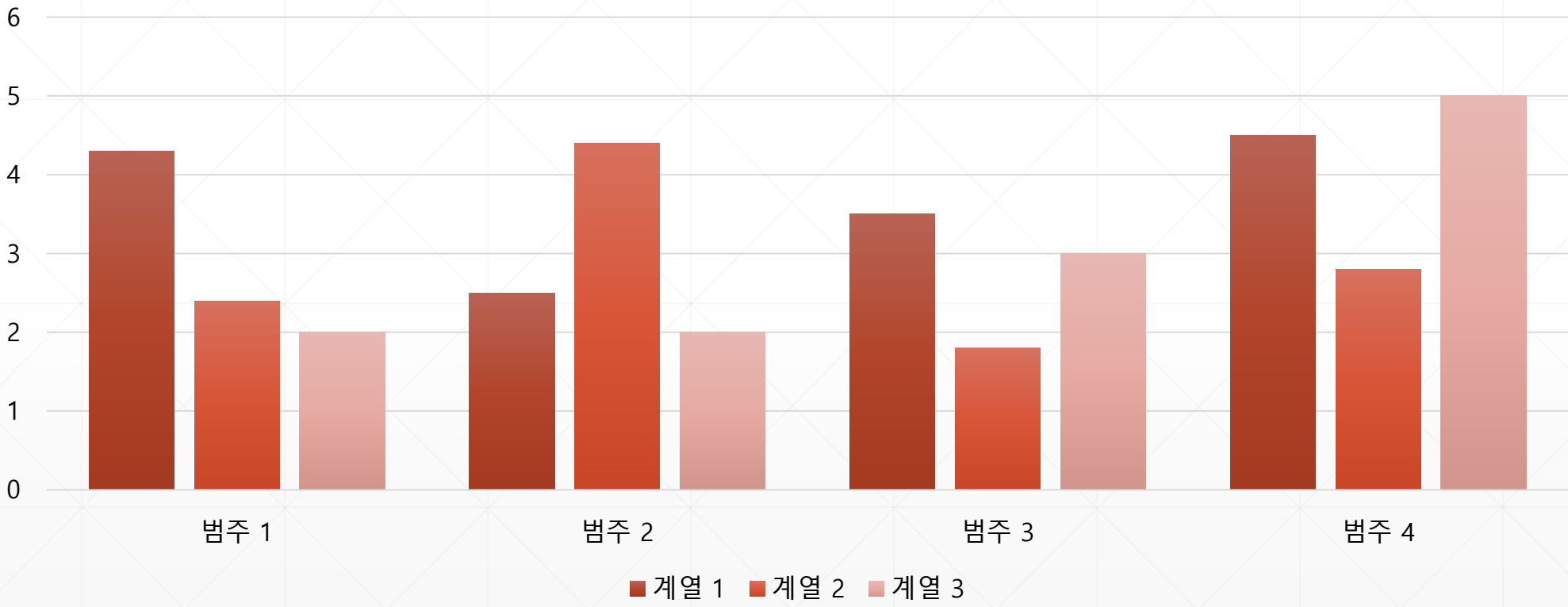
06| Parameter Sharing and Regularization

- GNN에 특화된 regularization strategies

1. Parameter Sharing Across Layers
   - GNN의 모든 Aggregate 및 update 시 동일한 매개변수 사용(parameter sharing)
   - 일반적으로 layer 6개이상 구조에서 효과적.
   - weight 수 줄일 수 있음(=avoid overfitting, time consuming problem)
   - 참고 : (5장 참 조)[$Li\ et\ al., 2015, Selsam\ et\ al., 2019$].

2. Edge Dropout
   - 인접행렬에서 edge를 무작위로 제거하는 regularization 전략.
   - 과적합을 피하고, noise에 강건해진다.
   - GAT에 사용된 필수 방법
   - 하지만 neighborhood 중심 subsampling 시 edge에 대한 정보가 사라짐을 주의했던 것처럼, 이 또한 모델 성능에 악영향을 끼칠 수 있으므로 부작용 조심.
   - large-scale GNN에서 매우 흔한 전략.

차트를 사용한 제목 및 내용 레이아웃



표를 사용한 두 개의 내용 레이아웃

- 여기에 첫 번째 글머리 기호
- 여기에 두 번째 글머리 기호
- 여기에 세 번째 글머리 기호

클래스	그룹 1	그룹 2
클래스 1	82	95
클래스 2	76	88
클래스 3	84	90

SmartArt가 있는 제목 및 내용 레이아웃

슬라이드 제목 추가 - 3

슬라이드 제목 추가 - 4

슬라이드 제목 추가 - 5