✅

GCN

Domain	Graph
tag	Graph learning Spatial method
Conference / Journal	ICLR
Publish year	2017
정리 날짜	@2024년 1월 5일
AI summary	ICLR 2017에서 발표된 GCN 논문은 그래프 신경망을 활용한 semi-supervised learning에 대한 연구입니다. GCN은 그래프 구조를 이용하여 노드의 feature를 전파시키는 방식으로 학습을 진행하며, Spectral Graph Convolution과 Layer-wise Linear model을 통해 그래프 신호를 처리합니다. 논문은 GCN의 의의와 한계에 대해 논의하고 있습니다.
AI key info	Semi-supervised learning, Graph Laplacian regularization, GCN, Spectral Graph Convolution, Layer-wise Linear model, Renormalization trick, Undirected graph

Summary

Background

Semi-supervised learning

- Supervised learning의 한계
  - label data로만 학습하기 때문에, 데이터 확보의 어려움 등
- 적은 label data와 많은 unlabeled data를 같이 사용해서 학습해보자

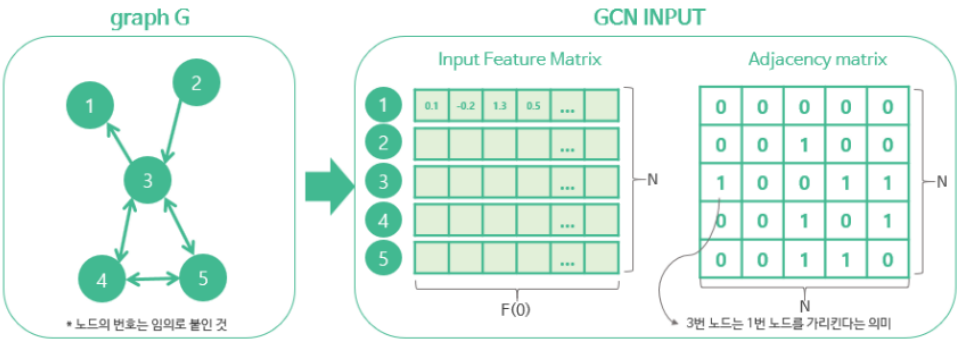
    - supervised loss와 unsupervised loss를 1-stage로 동시에 학습

$$Loss = L_s + L_u$$

- label가진 node가 적은 graph에서 label의 정보를 전체 graph로 smoothing 시키자
    - Graph Laplacian regularization

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

        - L_0: label node에 대한 supervised loss
        - f: neural network등의 함수
        - $\Delta = D - A$, normalize되지 않은 graph Laplacian matrix
        - ▼ A: adjacent matrix, 이웃들 정보 저장



        - D: graph의 dimension을 $D_{ii} = \Sigma_j A_{ij}$처럼 main diagnonal위의 원소로 표현
    - 그래프 구조를 $f(X, A)$로 encoding하여 L_0에 대해 supervised learning
        - 이 모델 $f(X, A)$를 어떻게 만들까가 이 논문 핵심

Methodology

GCN

- GCN의 propagation rule

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$$

- $\tilde{A} = A + I_N$: Adjacent matirx에 Identity matrix 더한 행렬
- $\tilde{D}_{ii} = \Sigma_j \tilde{A}_{ij}$: $\tilde{A}$에 대한 dimension
- $W^{(l)}$: l번째 layer에 대한 학습 weight
- σ: activation function
- $H^{(l)}$: l번째 layer의 output, H^0 = 입력 X

- Spectral Graph Convolution
  - Spectral convolution : 각 신호 x에 대한 필터 g_θ의 곱

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

    - U: normalized Laplacian 행렬 L의 eigenvector 행렬

$$L = I_N - D^{-1/2}AD^{-1/2} = U\Lambda U^T$$

    - Λ: eigenvalue diagonal matrix
    - $U^T x$: graph Fourier transform on x
  - 행렬이 커질수록 연산 비용이 너무 큼
    - → 필터를 근사 using Chebyshev 다항식

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

      - $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_N$: Λ의 크기를 재조정한 행렬
      - λ_{max}: L에서 가장 큰 eigenvalue
      - θ': Chebyshev 계수 나타내는 벡터

        - Chebyshev 다항식의 재귀적 정의

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

    - ∴ 원래 식을 근사

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

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

- Layer-wise Linear model
    - 위 spectral graph convolution을 딥러닝에 접목
        - 다항식을 선형의 convolution layer로 변형하기 위해 K=1 적용
            - convolution filter를 여러 겹 쌓기 때문에 다항식처럼 많은 클래스 유지
            - Chebyshev처럼 다항식 구조에 제한되지 X
            - 큰 그래프에서 local neighborhood 구조에 overfitting 방지
        - $\lambda_{max} \sim 2$로 근사

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

        - $\theta = \theta'_0 = -\theta'_1$로 통일하여 필터 근사

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

        - DNN에서 레이어 반복 적용시 exploding/vanishing gradient 문제

1) Vanishing gradient의 원인

딥러닝은 Back propagation 연산 시, Activation function의 편미분 값과 모델의 가중치 값들을 이용하게 된다.
Activation function 글에서 살펴봤듯이, sigmoid나 tanh를 사용하게 되면 편미분이 최대 0.25, 1의 값을 갖는다. 레이어가 깊어질 수록 편미분 값들이 반복적으로 곱해지면서 1보다 작아지게 되고, 결국 기울기가 소실되는 Vanishing gradient를 야기한다.

1) Exploding gradient의 원인

Back propagation 연산에는 Activation function의 편미분 값뿐만 아니라 가중치 값들도 관여하게 된다.
만약, 모델의 가중치들이 충분히 큰 값이라고 가정을 하면, 레이어가 깊어질수록 충분히 큰 가중치들이 반복적으로 곱해지면서 backward value가 폭발적으로 증가하게 된다. 이를 Exploding gradient라 한다.

- Renormalization trick
  - $I_N + D^{-1/2}AD^{-1/2} \in [0, 2]$를 변형

$$I_N + D^{-1/2}AD^{-1/2} \rightarrow \hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$$
$$(\tilde{A} = A + I_N, \tilde{D}_{ii} = \sum_j \tilde{A}_{ij})$$

- 최종 일반화 정의

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

  - Graph의 신호 $X \in R^{N \times C}$가 C개의 input과 F개의 필터 가질 때
  - $\Theta \in R^{C \times F}$의 각 필터에 대한 convolution layer의 출력 = $Z \in R^{N \times F}$
- 그래프가 주어지면 $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$를 전처리에서 미리 수행

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

Conclusion

의의

이 모델은 우리가 알고 있는 일반적인 딥러닝의 레이어와 가장 다른 점이 하나 있다. 바로 각 레이어의 입력마다 $\hat{A}$를 곱해준다는 점이다. 기존의 레이어는 각 레이어의 출력이 그냥 다음 레이어의 입력 feature가 되지만, $\hat{A}$는 일종의 normalize 처리가 된 인접행렬라고 볼 수 있으므로, 결국 해당 노드와 인접한 노드의 representation만 필터링하여 다음 레이어를 계산한다고 볼 수 있다. 따라서 이를 통해 hidden layer의 출력이 계속 각 노드에 대해서 이웃 노드의 정보가 더해진 representation의 역할을 수행하게 만드는 것이다.

이렇게 입력에 인접행렬을 곱해서 이웃개념을 적용하는 방식을 생각해보기는 쉽다. 하지만, 결국 $\hat{A}$가 제대로 normalize 되지 않으면 좋은 결과를 얻기 어렵다. 이 논문은 결국 spectral graph convolution의 라플라시안 행렬의 고유값 분해로부터 여러가지 처리를 거쳐서, 최종적으로 좋은 $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$를 얻을 수 있던 것이라고 생각한다. 논문의 실험 결과를 살펴보면, 선형 모델에서 이 $\hat{A}$를 구성하는 방법에 대한 여러가지 방법들을 실험했지만, 결국 renormalization 트릭을 적용하고 나서야 기존의 Chebyshev 다항식의 모델보다 좋은 결과를 얻게 되었다.

결국 이 논문은 spectral-based의 그래프 이론으로부터 spatial-based의 그래프 이론을 도출했다고 할 수 있다. 최초의 spatial-based 이론인 NN4G가 있지만, 이 논문은 normalize 없는 인접행렬 A를 활용했었다. 결국 다른 그래프 이론들보다 좋은 결과를 내지 못했기 때문에 다른 방향의 연구가 많이 이루어졌다. 결국 GCN이 좋은 결과를 얻게 되면서, 이를 기반으로 spatial-based 그래프 연구가 활발히 이루어지게 된 것으로 보인다.

한계

1. 메모리 문제 : 논문에서는 mini-batch 가 아닌, full-batch 의 gradient descent 를 적용하고 있어서 그래프가 커질수록 상당한 메모리가 필요하게 된다. 만약 mini-batch 를 활용한다면, K개의 레이어들이 메모리에 따로 저장되어야 한다.
2. 방향성 그래프 : 기본적으로 Laplacian 행렬 자체가 무방향 그래프를 전제하고 있기 때문에, 논문의 방식은 무방향 그래프에 대해서 제한된다.
3. self-connection의 적용 : 논문에서는 self-connection을 $\tilde{A} = A + I_N$을 통해서 적용했는데, 이는 인접행렬과 self-connection을 동등하게 적용하는 것을 의미한다. 하지만 이 비율에 대한 근거가 없기 때문에, 파라매터 λ를 적용해서 $\tilde{A} = A + \lambda I_N$을 사용해서 λ에 대해 학습시키는 것이 맞다. 하지만 여기에 파라매터가 추가될 경우, $\hat{A}$를 미리 구해놓은 후 학습을 진행하는 것이 아니라 매번 새로 계산해야 하기 때문에 논문에서는 이러한 방법을 사용하지 않은 것으로 보인다.

Questions

1. CNN이 이미지의 전체적인 구조를 파악하는 데에 사용되는 것처럼, 그래프의 전체적인 구조를 파악하기 위해서라 하면 노드의 feature보다 edge의 feature를 사용하는 게 더 유리할 것이라 생각했음. 논문에서 edge가 아니라 이웃 노드의 feature만 반영하는 식으로 만든 것으로 이해했는데 이게 맞는지
    a. 그래프 laplacian matrix를 만들 때 이웃 정보를 반영함
    b. 마냥 반영 안하는게 아닌듯
    c. 전반적인 이해가 먼저 필요할듯
2. Undirected graph에 적용이 안되는 이유: node feature로 라플라시안 matrix를 만들 때 assymmetric하게 만들면 되지 않나

a. 논문에서도 가능성 얘기함