Semi-Supervised Classification With Graph Convolutional Networks

2021.05.22

논문 Graph

- 논문 : arXiv

이 논문에서 GCN이 탄생하면서 그래프에 대한 convolution 방법이 좋은 결과를 얻기 시작하고, 많은 연구가 되었다. 이 논문은 spectral graph convolution 분야로부터 진행되어서 상당히 많은 수식들로부터 유도되고 있다. 그래프를 먼저 라플라시안 행렬로 나타내고, 이에 대한 고유값 분해를 하면서 계산을 진행하기 때문이다.

하지만 결국 이 고유값을 계산하는 것은 그래프가 커질수록 상당한 비용이 된다. 물론 이 논문에서는 선형 근사를 통해서 기존보다 계산 비용을 많이 줄였지만, 여전히 많은 계산 비용을 발생시키고, 그래프가 변화할 때마다 고유벡터를 다시 계산해줘야 한다. 그래서 이 논문 이후에는 수식들이 더욱 간단해지고, 계산 비용을 더욱 적게하면서 좋은 결과를 내는 방법들이 많이 탄생하게 되었다. 따라서 이 수식들에 대해 굳이 자세히 파고들 필요는 없기 때문에, 그리고 GNN 개념정리과 내용이 많이 중복되기 때문에 여기서는 이 수식들에 대해 자세한 설명을 붙이지 않고 최대한 간략하게 설명하고자 한다.

1. 논문의 목표

이 논문에서는 그래프에서 노드를 분류하는 문제에 대해서 해결하고자 하고 있는데, 이 때 각 label들은 전체 노드에서 아주 작은 부분에만 정의되어있다고 가정한다. 이 문제는 각 레이블 정보들을 전체 그래프로 smooth시키는 일종의 그래프 기반의 semi-supervised learning으로 볼 수 있다. 이 smoothing에 다음가 같이 graph Laplacian regularization 식을 사용할 수 있다.

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

이 식에 사용된 변수들은 다음과 같다.

- $\mathcal{L}_0$은 그래프에서 label되어있는 부분에 대한 supervised loss를 나타낸다.
- f는 neural network 등의 파라매터를 가진 임의의 함수를 말한다.
- $\Delta = D - A$로, normalize 되지 않은 graph Laplacian 행렬을 나타낸다.
- A는 해당 그래프의 인접행렬, D는 해당 그래프의 차수를$D_{ii} = \sum_j A_{ij}$와 같이 2차원 대각원소로 표현하고, 나머지는 값이 0인 행렬이다.

여기에서는 그래프 구조를 직접 모델 $f(X, A)$를 이용하여 encoding을 하고, 이에 대해서 타겟 $\mathcal{L}_0$에 대해 supervised learning 으로 모든 노드에 대한 레이블을 학습한다. 따라서 이 모델 $f(X, A)$를 어떻게 만드냐가 이 논문의 주요 내용이라고 할 수 있다.

2. Fast Approximate Convolutions 모델

그래프 기반의 NN 모델 $f(X, A)$에 대해서 이 논문에서는 여러 레이어에 걸친 Graph Convolutional Network를 구상하였다. 이 GCN의 전파 규칙은 다음과 같다.

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

- $\tilde{A} = A + I_N$이다. 즉, 인접 행렬 A와 자기자신에 대한 연결 I_N을 더한 행렬이다.
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$이다. 즉 $\tilde{A}$에 대한 차수가 된다.
- $W^{(l)}$은 l번째 레이어에 대한 학습 weight가 된다.
- σ는 ReLU등과 같은 activation function이다.
- $H^{(l)}$은 l번째 레이어의 output이고, H^0는 입력 X와 같다.

이 GCN이 탄생하게 된 과정은 다음과 같다.

2.1. Spectral Graph Convolutions

그래프에서 spectral convolutions를 다음과 같이 각 신호 x에 대한 필터 g_θ에 대한 곱으로 정의할 수 있다.

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

여기서 U는 그래프의 normalized Laplacian 행렬 $L = I_N - D^{-1/2}AD^{-1/2} = U\Lambda U^T$의 고유벡터 행렬을 나타낸다. Λ는 고유값을 나타내는 대각행렬이고, $U^T x$는 x에 대한 그래프 푸리에 변환을 나타낸다.

위 식을 계산하려면, 그리고 L에 대한 고유값 분해를 수행하려면 행렬이 커질수록 연산 비용이 상당히 많이 들게 된다. 이를 해결하는 방법에는 다음과 같이 Chebyshev 다항식으로 필터를 근사하는 방법이 있다.

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

여기서 $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_N$은 Λ의 크기를 재조정한 행렬을 나타내며, 이 때 λ_{max}는 L에서 가장 큰 고유값을 의미한다. θ'는 Chebyshev 계수를 나타내는 벡터가 된다. 이 Chebyshev 다항식은 다음과 같이 재귀적으로 정의할 수 있다.

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

따라서 이를 이용해서 원래 식을 다음과 같이 근사시킬 수 있다.

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

여기서 $\tilde{L} = 2L/\lambda_{max} - I_N$을 나타낸다.

2.2. Layer-wise Linear Model

이 논문의 핵심 아이디어는 위 spectral graph convolution을 딥러닝과 접목시키는 것이다. 딥 러닝의 뉴럴 네트워크 모델은 대부분 여러개의 선형의 convolution 레이어를 쌓아서 만들지만 위 식은 다항식이기 이러한 구조에 적합하지 않다. 따라서 이를 선형모델로 만들기 위해서 논문에서는 위 식에서 $K = 1$을 적용하여 선형 convolution 필터가 되고, 이 레이어를 여러번 쌓는 구조를 생각했다.

이 방법은 여러개의 선형 convolution 필터를 쌓기 때문에 기존 다항식만큼의 많은 클래스를 유지하면서도, Chebyshev 와 같이 한정된 다항식 구조에 제한되지 않을 수 있다. 또한 직관적으로 매우 큰 그래프에서의 지역적인 neighborhood 구조에 대한 overfitting을 방지할 것으로 보인다.

여기서는 $\lambda_{max} \approx 2$로 근사하여 계산을 했는데, 어차피 뉴럴 네트워크의 파라매터들이 이 λ_{max}의 역할을 대체할 수 있기 때문이다. 따라서 이를 적용하면 다음과 같은 선형 convolution 필터가 된다.

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

여기서의 두 파라매터 θ'_0, θ'_1이 전체 그래프에 걸쳐서 공유하는 파라매터가 된다. 논문에서는 파라매터의 수를 줄이기 위해 다음과 같이 $\theta = \theta'_0 = -\theta'_1$로 통일하여 필터를 근사시켰다.

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

이 경우에, $I_N + D^{-1/2}AD^{-1/2}$이 $[0, 2]$의 범위를 갖게 되는데 딥 뉴럴 네트워크에서 레이어를 반복해서 적용하면 exploding/vanishing gradient 문제가 발생했다고 한다. 따라서 논문에서는 이 식에 대해서 다음과 같이 renormalization 트릭을 사용했다.

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

이 식을 통해서 최종적으로 다음과 같이 일반화하여 정의할 수 있다.

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

이는 그래프 신호 $X \in R^{N\times C}$가 C개의 input 채널과 F개의 필터를 가질 때, $\Theta \in R^{C\times F}$의 각 필터에 대한 파라매터를 갖는 convolution 레이어의 출력이 바로 $Z \in R^{N\times F}$가 되는 것이다.

3. Semi-supervised Node Classification

위의 모델 $f(X, A)$가 바로 논문의 핵심이고, 나머지 내용은 이를 통해서 graph convolution을 구하고, 이를 통해 semi-supervised node classification 문제를 풀어내는 과정을 소개하고 있다. 이는 일반적인 딥러닝 모델에서의 classification과 같기 때문에 더 이상 자세한 설명을 할 필요가 없다. 따라서 논문에서 나온 두개의 레이어로 된 GCN으로 semi-supervised node classification 문제를 풀어보는 예제만 소개하도록 한다.

먼저 그래프가 주어지면, $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$의 값을 전처리과정에서 미리 계산을 해둔다. 그리고 나서 다음과 같은 모델을 구현한다.

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

여기서 W는 각 레이어의 weight가 되고, 마지막에 softmax 함수를 통해서 classification 결과가 나오게 된다. 논문에서는 bias를 활용하지는 않았다. 이 출력 Z에 대해서 cross entropy loss를 통해서 흔히 사용되는 classification 작업을 수행하면 된다.

4. GCN 모델의 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 이

루어졌다. 결국 GCN이 좋은 결과를 얻게 되면서, 이를 기반으로 spatial-based 그래프 연구가 활발히 이루어지게 된 것으로 보인다.

5. 한계점

논문에서는 다음과 같은 한계점에 대해서 이야기하고 있다.

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.