EasyChair Preprint

№ 11131

A Fast Method for Updating Node Representations Learned by Graph Convolutional Networks When Node Features Change

Yeonju Song and Ki Yong Lee

October 23, 2023

그래프 합성곱 신경망이 학습한 노드 표현을 노드 특징이 변화했을 때 빠르게 갱신하는 방법

송연주○, 이기용
숙명여자대학교 컴퓨터과학과
{eiwisi99, kiyonglee}@sookmyung.ac.kr

A Fast Method for Updating Node Representations Learned by Graph Convolutional Networks When Node Features Change

Yeonju Song○, Ki Yong Lee
Department of Computer Science, Sookmyung Women's University

요 약

최근 여러 분야의 그래프 데이터 분석에 그래프 합성곱 신경망(graph convolutional network, GCN)이 널리 사용되고 있다. GCN은 주어진 그래프에 포함된 각 노드의 특징과 노드 간의 연결 상태를 입력으로 받아 각 노드에 대한 표현(representation)을 학습한다. 하지만 학습 후 시간이 지나 일부 노드만 특징이 변경되더라도 GCN은 전체 노드의 표현을 모두 다시 학습해야 한다는 문제가 있다. 따라서 본 논문에서는 그래프에 포함된 노드 중 일부만 특징이 변경되었을 때, 이전 학습 결과를 활용하여 전체 노드의 표현을 빠르게 갱신하는 방법을 제안한다. 제안 방법은 이전에 학습된 표현에서 특징이 변경된 노드들에 의해 변경된 부분만을 빠르게 계산한다. 가상 데이터를 사용한 실험 결과 제안 방법은 특징이 변경된 노드 수가 적을수록 전체 노드의 표현을 모두 다시 학습하는 기존 방법에 비해 수행 시간을 크게 줄임을 확인하였다.

1. 서 론

그래프(graph)는 노드(node)들과 이들의 관계를 나타내는 간선(edge)들로 이루어진 데이터 구조이다. 그래프는 소셜 네트워크, 화합물, 교통망 등 실생활의 다양한 데이터를 표현하는 데 사용된다.

최근 들어 그래프 데이터 분석에 그래프 신경망(graph neural network, GNN)이 활발히 사용되고 있다 [1][2]. 그 중 대표적인 그래프 신경망인 그래프 합성곱 신경망(graph convolutional network, GCN)은 그래프 데이터에 합성곱 연산을 적용한 것으로써, 많은 응용 분야에서 좋은 성능을 보이고 있다 [3]. GCN은 주어진 그래프에 포함된 각 노드의 특징과 노드 간의 연결 상태를 입력으로 받아 각 노드에 대한 표현(representation)을 학습한다. GCN이 학습한 각 노드의 표현은 노드 분류, 노드 클러스터링, 그래프 분류 등과 같은 최종 태스크에 활용된다.

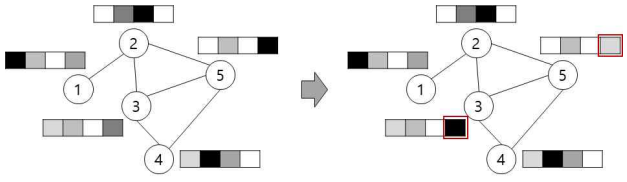

그림 1 그래프에 포함된 노드의 특징이 변화하는 예

하지만 그래프는 시간이 지나면서 변화될 수 있다. 특히 소셜 네트워크와 같이 노드 수가 많은 대규모 그래프에서는 시간의 흐름에 따라 일부 노드만 특징이 변화되는 경우가 있다. 그림 1은 그래프에 포함된 5개의 노드 중 3번과 5번 노드만 특징이 변화된 그래프의 예를 나타낸다. 그러나 이렇게 일부 노드의 특징만 변화된 경우에도 GCN은 그래프에 포함된 전체 노드의 표현을 모두 다시 학습해야 한다는 문제가 있다.

따라서 본 논문에서는 그래프에 포함된 노드 중 일부만 특징이 변경되었을 때, 이전의 학습 결과를 활용하여 전체 노드의 표현을 빠르게 갱신하는 방법을 제안한다. 제안 방법은 변경된 특징에 의해 각 노드의 표현이 변경된 부분만을 빠르게 계산한다. 그래프에 포함된 노드가 n개이고 각 노드의 특징이 d개이며, 이 중 특징이 변경된 노드가 c개일 때 ($c \ll n$), 전체 노드의 표현을 모두 다시 학습하는 방법의 시간 복잡도는 $O(n^2d)$이지만 제안 방법의 시간 복잡도는 $O(ncd)$에 불과하다. 가상 데이터를 사용한 실험 결과 제안 방법은 특징이 변경된 노드 수가 적을수록 전체 노드의 표현을 모두 다시 학습하는 기존 방법에 비해 수행 시간을 최대 50%까지 감소시킴을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 간략히 설명하고, 3장에서는 제안 방법을 상세히 설명한다. 4장에서는 가상 데이터를 사용한 실험 결과를 제시하고, 마지막 5장에서는 결론을 맺는다.

2. 관련 연구

1장에서 언급한 바와 같이 지금까지의 GNN 연구들은 대부분 고정된 그래프에 대해 각 노드의 표현을 학습하는 방법에 집중되었다. 하지만 그래프의 변화를 고려하

는 연구도 이미 일부 진행된 바가 있으며, 본 절에서는 이들에 대해 간략히 살펴본다.

[4]은 사용자와 아이템 간의 구매 관계 등을 나타내는 사용자-아이템 상호작용 그래프가 변화했을 때, 이를 반영하여 추천에 사용되는 GNN 모델을 갱신하는 방법을 제안하였다. 하지만 [4]은 그래프에 새로운 노드 또는 간선이 추가되었을 때 GNN 모델을 갱신하는 방법에 대한 것이며, 노드들의 특징이 변화했을 때 각 노드의 표현을 갱신하는 방법에 관한 연구는 아니다.

[5]는 시간의 흐름에 따라 동적으로 변하는 그래프를 분석하기 위한 시간 그래프 네트워크(temporal graph network, TGN)를 제안하였다. TGN은 시간에 따라 변하는 그래프의 구조를 이벤트 시퀀스로 표현하고 이 시퀀스의 특징을 학습한다. 하지만 [5]는 그래프의 구조가 동적으로 변할 때 이 변화의 특징 또는 패턴을 학습하는 방법에 관한 연구이며, 노드들의 특징이 변화했을 때 각 노드의 표현을 효율적으로 갱신하는 방법에 관한 연구는 아니다.

[6]은 소셜 네트워크를 표현하는 그래프가 동적으로 변화한다고 가정하고, 주어진 미래의 특정 기간 동안 그래프에 새로 추가될 간선을 예측하는 기법을 제안하였다. 이를 위해 [6]은 그래프의 변화율과 노드 간의 친밀도 등을 계산하고 이를 통해 미래에 새로 추가될 간선을 예측한다. 하지만 [6]은 그래프의 구조가 미래에 어떻게 바뀔지 예측하는 연구이며, 노드들의 특징이 변화했을 때 각 노드의 표현을 효율적으로 갱신하는 방법에 관한 연구는 아니다.

3. 제안 방법

본 장에서는 그래프에 포함된 노드들의 표현을 GCN으로 학습한 후, 일부 노드들의 특징이 변화했을 때 GCN으로 학습한 노드들의 표현을 빠르게 갱신하는 방법을 설명한다. 우선 3.1절에서는 GCN이 노드들의 표현을 학습하는 방법을 설명하고, 3.2절에서는 GCN이 학습한 노드들의 표현을 빠르게 갱신하는 방법을 설명한다.

3.1 GCN의 노드 표현 학습 방법

n개의 노드를 가진 그래프 G가 주어졌다고 하자. GCN은 G에 포함된 각 노드의 특징을 나타내는 노드 특징행렬 $X \in R^{n \times d}$과 노드 간의 연결 상태를 나타내는 인접 행렬 $A \in R^{n \times n}$를 입력으로 받는다. X의 i번째 행은 i번째 노드에 대한 d차원의 특징 벡터를 나타낸다 $(i = 1,...,n)$. GCN의 l번째 층에서 얻어진 노드 특징행렬을 $H^{(l)} \in R^{n \times d}$이라 했을 때, (식 1)은 GCN의 $(l+1)$번째 층에서 수행되는 연산을 나타낸다.

$$Z^{(l)} = \hat{A} H^{(l)} W^{(l)} \qquad \text{(식 1)}$$
$$H^{(l+1)} = \sigma(Z^{(l)})$$

여기서 $W^{(l)}$는 학습 가능한 매개변수 행렬, $\sigma()$는 활성화 함수를 나타내며, $H^{(0)} = X$로 정의된다. 한편 크기가 $n \times n$인 단위행렬 I_n에 대해 $\tilde{A} = A + I_n$이라 하고, 행렬 $\tilde{D}$를 $\tilde{D}_{ii} = \sum_j \tilde{A}_{jj}$인 대각행렬이라 했을 때, $\hat{A} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$로 정의된다. GCN은 (식 1)을 사용하여 각 층에서 각 노드에 대한 표현을 학습하며, $H^{(l)}$의 i번째 행은 GCN이 l번째 층에서 i번째 노드에 대해 학습한 d차원의 표현을 나타낸다.

3.2 GCN의 노드 표현 갱신 방법

시간이 지나면서 n개 노드 중 c개 노드의 특징이 변경되었고 $(c \ll n)$, 그에 따라 X가 X'로 변경되었다고 하자. X가 X'로 변경됨에 따라 $H^{(1)}$이 $H'^{(1)}$로 변경되었다고 하면 $H'^{(1)}$은 (식 2)로 나타낼 수 있다.

$$Z'^{(0)} = \hat{A} X' W^{(0)} \qquad \text{(식 2)}$$
$$H'^{(1)} = \sigma(Z'^{(0)})$$

X'와 X의 차이를 ΔX라고 하고 $X' = X + \Delta X$라 하면, (식 2)의 $H'^{(1)}$은 (식 3)과 같이 쓸 수 있다.

$$\begin{aligned} H'^{(1)} &= \sigma(Z'^{(0)}) \qquad \text{(식 3)} \\ &= \sigma(\hat{A}(X + \Delta X) W^{(0)}) \\ &= \sigma(\hat{A} X W^{(0)} + \hat{A} \Delta X W^{(0)}) \\ &= \sigma(Z^{(0)} + \hat{A} \Delta X W^{(0)}) \end{aligned}$$

여기서 $Z^{(0)}$는 이미 $H^{(1)}$를 구할 때 계산된 값이므로 $H'^{(1)}$를 구하기 위해서는 $\hat{A} \Delta X W^{(0)}$만 새로 계산하면 된다. 이때 특징이 변경된 c개 노드를 $i_1, i_2, ..., i_c$번째 노드라고 하자. 그러면 $\hat{A} \Delta X W^{(0)}$는 다음과 같이 쓸 수 있다.

$$\begin{aligned} \hat{A} \Delta X W^{(0)} &= [\hat{A}_1 \ \cdots \ \hat{A}_n] \begin{bmatrix} \Delta X_1 \\ \vdots \\ \Delta X_n \end{bmatrix} W^{(0)} \qquad \text{(식 4)} \\ &= [\hat{A}_1 \ \cdots \ \hat{A}_n] \begin{bmatrix} 0 \\ \vdots \\ \Delta X_{i_1} \\ \vdots \\ 0 \\ \vdots \\ \Delta X_{i_c} \\ \vdots \\ 0 \end{bmatrix} W^{(0)} \\ &= (\hat{A}_{i_1} \Delta X_{i_1} + \ \cdots \ + \hat{A}_{i_c} \Delta X_{i_c}) W^{(0)} \\ &= [\hat{A}_{i_1} \ \cdots \ \hat{A}_{i_c}] \begin{bmatrix} \Delta X_{i_1} \\ \vdots \\ \Delta X_{i_c} \end{bmatrix} W^{(0)} \\ &= \hat{A}_{[i_1, ..., i_c]} \Delta X_{[i_1, ..., i_c]} \end{aligned}$$

여기서 $\hat{A}_i$은 $\hat{A}$의 i번째 열을, ΔX_i은 ΔX의 i번째 행을 나타낸다. 또한 $\hat{A}_{[i_1, i_2, ..., i_c]} \in R^{n \times c}$는 $\hat{A}$의 $i_1, i_2, ..., i_c$번째 열들로만 이루어진 행렬을 나타내며,

$\Delta X_{[i_1, i_2, \ldots, i_c]} \in R^{c \times d}$는 ΔX의 $i_1, i_2, \ldots, i_c$번째 행들로만 이루어진 행렬을 나타낸다. (식 3)과 (식 4)로부터 최종적으로 제안 방법은 (식 5)를 사용하여 $H'^{(1)}$을 계산한다.

$$Z'^{(0)} = Z^{(0)} + \hat{A}_{[i_1,\ldots,i_c]} \Delta X_{[i_1,\ldots,i_c]} W^{(0)} \qquad \text{(식 5)}$$
$$H'^{(1)} = \sigma(Z'^{(0)})$$

(식 5)를 사용하여 $H'^{(1)}$를 구하고 나면 $l \geq 1$일 때 제안 방법은 (식 6)을 사용하여 $H'^{(l+1)}$를 계산한다.

$$\Delta H^{(l)} = H'^{(l)} - H^{(l)} \qquad \text{(식 6)}$$
$$Z'^{(l)} = Z^{(l)} + \hat{A}_{[i_1,\ldots,i_c]} \Delta H^{(l)}_{[i_1,\ldots,i_c]} W^{(l)}$$
$$H'^{(l+1)} = \sigma(Z'^{(l)})$$

단, (식 6)에서 $i_1, i_2, \ldots, i_c$는 $H^{(l)}$에서 변경이 발생한 노드들의 번호를 의미한다. (식 6)을 사용하여 제안 방법은 GCN의 $(l+1)$번째 층에서 $\hat{A}_{[i_1,\ldots,i_c]} \Delta H^{(l)}_{[i_1,\ldots,i_c]} W^{(l)}$만 새로 계산함으로써 $H'^{(l+1)}$를 빠르게 얻을 수 있다.

3.3 시간 복잡도 분석

본 절에서는 노드 특징행렬 X이 X'로 갱신되었을 때, GCN의 $(l+1)$번째 층에서 기존 방법과 제안 방법의 시간 복잡도를 비교한다. 전체 노드의 표현을 모두 다시 학습하는 기존 방법은 GCN의 $(l+1)$번째 층에서 (식 7)을 사용하여 $H'^{(l+1)}$를 계산한다.

$$Z'^{(l)} = \hat{A} H'^{(l)} W^{(l)} \qquad \text{(식 7)}$$
$$H'^{(l+1)} = \sigma(Z'^{(l)})$$

(식 7)에서 $\hat{A} \in R^{n \times n}$, $H'^{(l)} \in R^{n \times d}$, $W^{(l)} \in R^{d \times d}$이므로 $Z'^{(l)}$을 계산하는 시간 복잡도는 행렬 곱하기의 시간 복잡도에 의해 $O(n^2 d + nd^2)$이 된다. 한편 $H'^{(l+1)} = \sigma(Z'^{(l)})$을 계산하는 시간 복잡도는 $Z'^{(l)} \in R^{n \times d}$이므로 $O(nd)$이다. 따라서 (식 7)의 시간 복잡도는 $O(n^2 d + nd^2)$이 된다.

이에 비해 제안 방법은 GCN의 $(l+1)$번째 층에서 (식 6)을 사용하여 $H'^{(l+1)}$를 계산한다. 우선 $\Delta H^{(l)} = H'^{(l)} - H^{(l)}$를 계산하는 시간 복잡도는 $O(nd)$이다. 또한 $\hat{A}_{[i_1,\ldots,i_c]} \in R^{n \times c}$, $\Delta H^{(l)}_{[i_1,\ldots,i_c]} \in R^{c \times d}$, $W^{(l)} \in R^{d \times d}$이므로 $Z'^{(l)}$을 계산하는 시간 복잡도는 $O(ncd + nd^2)$가 되며, $H'^{(l+1)} = \sigma(Z'^{(l)})$을 계산하는 시간 복잡도는 $O(nd)$이므로 (식 6)의 시간 복잡도는 $O(ncd + nd^2)$가 된다. 따라서 $c \ll n$일수록 (식 6)의 시간 복잡도인 $O(ncd + nd^2)$는 (식 7)의 시간 복잡도인 $O(n^2 d + nd^2)$에 비해 작아지는 것을 알 수 있다.

4. 실험 결과

본 장에서는 일부 노드의 특징만 변화된 경우에도 그래프에 포함된 전체 노드의 표현을 모두 다시 학습하는 기존 방법의 수행 시간과 제안 방법의 수행 시간을 비교하였다.

- **실험 데이터**: 본 실험에서는 가상 데이터를 사용하여 기존 방법과 제안 방법의 수행 시간을 비교하였다. 가상 데이터의 구체적인 생성 방법은 다음과 같다. 그래프에 포함된 노드의 개수는 10,000개부터 60,000개까지 늘려가며 생성하였다. 그래프에 포함된 간선은 그래프에 포함된 노드의 개수가 n개일 때, 각 노드의 평균 연결 수가 $n/1000$이 되도록 임의로 생성하였다. 또한 각 노드의 특징은 10개에서 50개까지 늘려가며 생성하였으며, 각 특징의 값은 임의의 숫자로 생성하였다.
- **실험 환경**: 실험에 사용된 모델들은 모두 PyTorch로 구현하였으며, 기존 방법과 제안 방법의 수행 시간은 Titan RTX GPU, Intel i9-9900K 3.6GHz CPU, 64GB RAM, 2TB HDD가 탑재된 PC에서 측정하였다.
- **모델 훈련**: GCN을 훈련할 때 최적화 알고리즘은 Adam을 사용하였으며, 학습률은 0.001로 설정하였다. 과적합을 방지하기 위해 드롭아웃은 0.5로 설정하였으며 L2 규제를 적용하였다. 훈련 반복 횟수(epoch)는 최대 100회로 설정하였다.

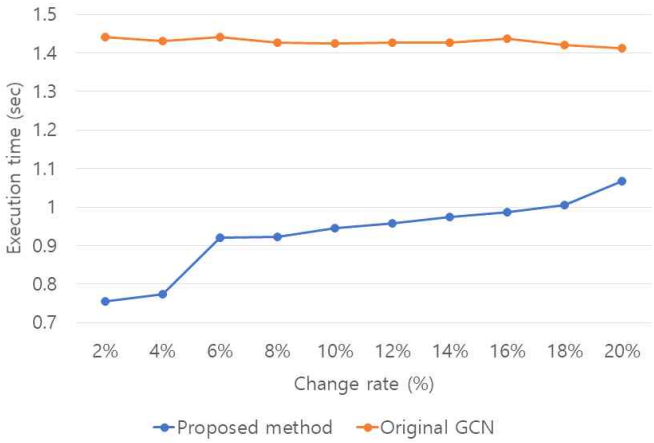

그림 2 특징이 변경된 노드 비율에 따른 성능 비교

그림 2는 그래프에 포함된 노드 중 특징이 변경된 노드의 비율을 2%에서 20%까지 증가시키며 기존 방법과 제안 방법의 수행 시간을 비교한 결과이다. 그래프에 포함된 노드의 개수는 20,000개로 고정하고, 각 노드의 특징 개수는 30개로 하였다. 예상한 바와 같이 제안 방법은 특징이 변경된 노드의 비율이 낮을수록 기존 방법에 비해 좋은 성능을 보인다. 이것은 3.3절에서 (식 6)과 (식 7)의 시간 복잡도를 비교한 것처럼 특징이 변경된 노드의 개수가 줄어들수록 제안 방법의 시간 복잡도가 줄어들기 때문이다. 또한 제안 방법은 특징이 변경된 노드의 비율이 20%에 이르러도 기존 방법에 비해 수행 시간을 25% 이상 단축시킴을 관찰할 수 있다.

그림 3은 그래프에 포함된 노드의 개수를 10,000개에서 60,000개로 증가시키며 기존 방법과 제안 방법의 확장성을 비교한 결과이다. 각 노드의 특징 개수는 30개로

하였으며, 특징이 변경된 노드의 비율은 10%로 고정하였다. 그림 3을 통해 제안 방법은 노드의 개수가 증가하더라도 항상 기존 방법에 비해 좋은 성능을 보임을 확인할 수 있다.

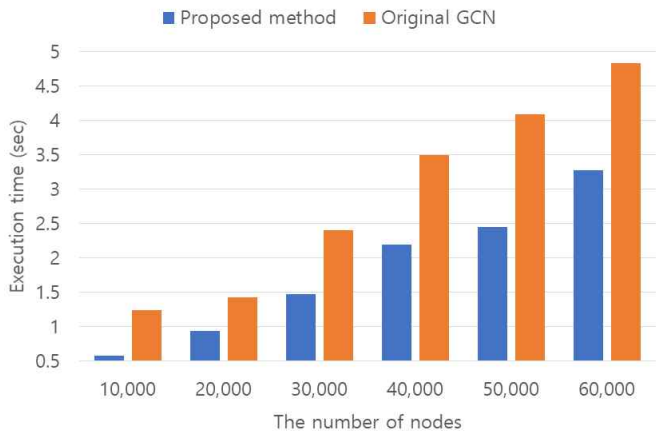

그림 3 노드 개수 증가에 따른 성능 비교

그림 4는 노드 특징 개수를 10개에서 50개로 증가시키며 기존 방법과 제안 방법의 성능 변화를 관찰한 결과이다. 노드의 개수는 20,000개로, 특징이 변경된 노드의 비율은 10%로 고정하였다. 두 방법 모두 노드 특징 개수가 증가할수록 수행 시간이 증가한다. 이것은 3.3절에서 분석한 것처럼 두 방법 모두 특징 개수가 증가하면 시간 복잡도가 증가하기 때문이다. 하지만 이 경우에도 제안 방법은 노드 특징 개수에 상관없이 항상 기존 방법에 비해 좋은 성능을 보임을 알 수 있다.

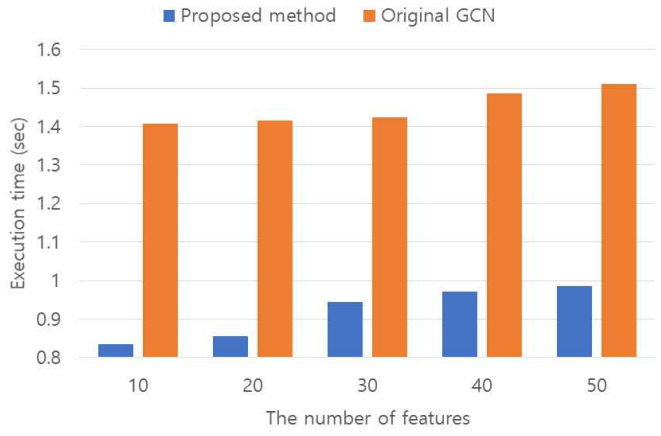

그림 4 노드 특징 개수 증가에 따른 성능 비교

마지막으로 그림 5는 그래프에 포함된 노드 중 특징이 변경된 노드의 비율을 100%까지 증가시키며 기존 방법과 제안 방법의 수행 시간을 비교한 결과이다. 그래프에 포함된 노드의 개수는 20,000개로 고정하고, 각 노드의 특징 개수는 30개로 하였다. 그림 5에서 볼 수 있듯이 제안 방법은 특징이 변경된 노드의 비율이 증가할수록 수행 속도가 증가하는 한편, 기존 방법은 항상 유사한 성능을 보인다. 이것은 제안 방법의 시간 복잡도는 특징이 변경된 노드 수에 따라 증가하지만 기존 방법은 전체 노드의 표현을 항상 모두 다시 계산하기 때문이다. 그에 따라 특징이 변경된 노드의 비율이 전체 노드의 60%를 넘어가면 제안 방법의 성능이 오히려 기존 방법보다 나빠짐을 알 수 있다. 따라서 제안 방법은 특징이 변경된 노드의 비율이 그리 높지 않은 환경에서 사용되는 것이 바람직하며, 특징이 변경된 노드의 비율이 매우 높을 때는 기존 방법을 사용하는 것이 바람직하다.

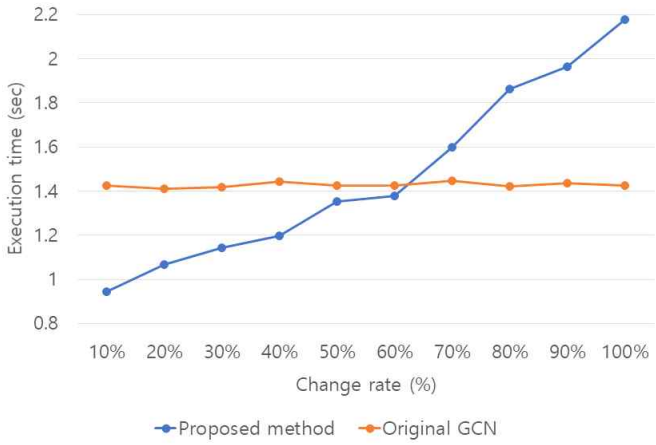

그림 5 특징이 변경된 노드 비율이 매우 높은 경우

5. 결론

본 논문은 GCN으로 학습된 각 노드의 표현을 이후 일부 노드의 특징이 변경되었을 때 빠르게 갱신하는 방법을 제안하였다. 제안 방법은 모든 노드의 표현을 재계산하는 기존 방법과 달리 변경되는 부분만을 계산함으로써 노드의 표현을 더욱 빠르게 갱신할 수 있다. 또한 본 논문에서는 제안 방법의 성능을 이론적으로도 분석하였다. 실험 결과 제안 방법은 특징이 변경된 노드가 전체 노드의 60%를 넘지 않을 때 기존 방법에 비해 항상 좋은 성능을 보임을 확인하였다.

참 고 문 헌

[1] Z. Wu, et al., “A comprehensive survey on graph neural networks,” IEEE Transactions on Neural Networks and Learning Systems, 32(1), pp. 4-24, 2020.

[2] L. Wu, et al., “Graph neurnal networks: foundation, frontiers and applications,” ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022.

[3] T. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” arXiv:1609.02907, 2016.

[4] J. Xia et al, “Incremental GCN for collaborative filtering,” ACM International Conference on Information and Knowledge Management, pp. 2170-2179, 20121.

[5] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, “Temporal graph networks for deep learning on dynamic graphs,” arXiv:2006.10637, 2020.

[6] L. Yao, L. Wang, L. Pan, and K. Yao, “Link prediction based on common-neighbors for dynamic social network,” Procedia Comput. Sci., vol. 83, pp. 82-89, May 2016.