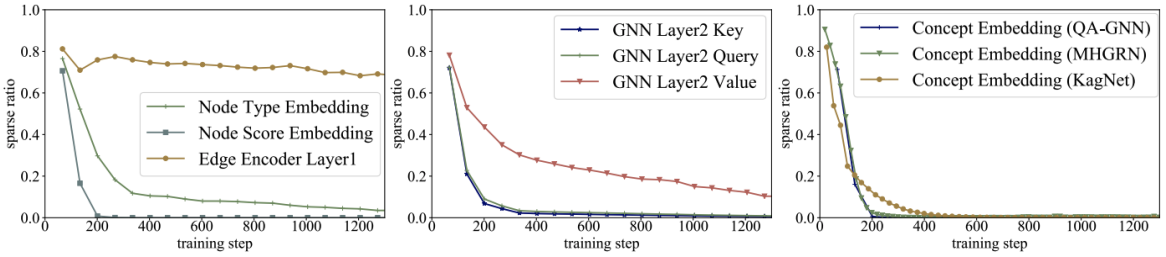
10월 13,14,15,16,17,18,19일

1. 기존에 진행했던 Cycle Encoder는 사이클 개수를 노드 임베딩 값에 주입했다면 이번에는 random number, 각 노드의 degree를 주입하는 방식으로 실험을 해보고자 한다.
2. New context score를 사용할 때 기존에는 Cycle Passage(세개의 트리플)를 사용했지만 이번에는 Triple을 대입해보고자 한다.

여기서 오늘은 1번 내용을 진행해보고자 한다.

GSC 내용



According to thees plots, we summarize our key observations as follows: 1) the left plot shows that the edge encoder that encodes edge triplets (edge/node type information) preserves a relatively higher sparse ratio, while the node score embedding layer can be fully pruned; 2) the middle plot shows the layers inside GNN, and all the sparse ratios are low while the value layer has a relatively higher sparse ratio than key/query layers; 3) the right plot shows the concept embedding layers of three representative GNN methods, which process the initial node embeddings, can be completely discarded. Inspired by these findings, we come up with the following design guidelines for a much simpler yet effective graph neural module for knowledge-aware reasoning:

- Node embeddings : The initial node embedding and the extra score embedding are shown to be **dispensable** in these scenarios, thus we can directly remove the embedding layers.
- Edge embeddings : The edge encoder layers are hard to prune which indicates that edge/node type information are essential to reasoning, so we further leverage it to get a better representation.
- Message passing layers : The linear layers inside GNN layers (query, key, value, etc) can be pruned to a very low sparse ratio, suggesting that GNN may be over parameterized in these systems, and we can use fewer parameters for these layers.
- Graph pooler : The final attention-based graph pooler aggregates the node representation over the graph to get a graph representation. We observe that the key/query layers inside the pooler can be pruned out; as a result, the graph pooler can be reduced to a linear transformation.

이래서 GSC에서는 노드 임베딩을 0으로 주었다.(1차원)

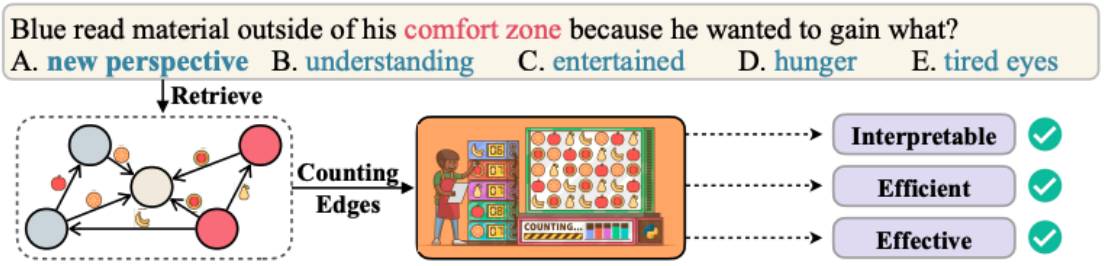


Figure 1: We analyze state-of-the-art GNN modules for the task of KG-powered question answering, and find that the counting of edges in the graph plays an essential role in knowledge-aware reasoning. Accordingly, we design an efficient, effective and interpretable graph neural counter module for knowledge-aware QA reasoning.

노드의 임베딩을 0을 주고 에지의 차원도 1이므로 에지를 카운팅하는것만으로도 성능을 뽑을 수 있다고 주장하는 논문이다.

그만큼 에지가 중요하다는 것인데, 나는 여기서 노드의 임베딩과 에지의 임베딩도 1차원인데 굳이 노드 임베딩값을 0을 주는것보단 그래프의 정보를 담을 수 있는 정보면 괜찮지 않을까 생각한 것이고 그래서 사이클 개수를 추가했던 것이다.

여기서 사이클의 영향을 파악하기 위해선 다른 숫자도 넣어봐서 실험을 진행하기로 했다.

1. 1~10의 random number
2. node의 degree

아 그리고 그대로 넣는것이 아니라. 9차원으로 만들고 concat을 하고 2-layer MLP를 거쳐서 1차원으로 변경

```
aggr_out = torch.zeros(n_node_total, 1).to(node_type_ids.device)
upgrade_cycle_count_features = upgrade_cycle_count_features.expand(-1,9).to(node_type_ids.device)
aggr_out = self.regulator1(torch.cat([aggr_out,upgrade_cycle_count_features],dim=-1)).to(node_type_ids.device)
```

이렇게 진행했었다.

그런데 가만 생각해보니 aggr_out은 어차피 0차원인데…? concat을 하지 말고

그냥 upgrade_cycle_count_features를 그대로 넣어보는건 어떨까?

방법은 다음과 같이 정할 수 있다.

1. aggr_out(1차원)대신 cycle count를 그대로 넣기
2. aggr_out(1차원)대신 cycle count를 넣고 차원 확장 후 MLP를 돌려서 넣기
3. [aggr_out; 차원 확장한 cycle count] 를 MLP에 돌려서 넣기

위 예시는 cycle count를 했지만

orig cycle count, upgrade cycle count, random number, node degree로 실험을 진행해보자

Original Cycle count

1. aggr_out(1차원)대신 cycle count를 그대로 넣기

- seed0(20231013_070657)
  - dev : 75.92, test : 71.31
- seed1(20231013_070824)
  - dev : 76.90, test : 71.96
- seed2(20231013_070936)
  - dev : 76.25, test : 71.96

dev : 76.36(**±0.41), test : 71.74(±0.31)**

2. aggr_out(1차원)대신 cycle count를 넣고 차원 확장 후 MLP를 돌려서 넣기(9차원)

- seed0(20231013_105941)
  - dev : 78.79, test : 74.30
- seed1(20231013_110032)
  - dev : 79.03, test : 74.21
- seed2(20231013_110039)
  - dev : 79.69, test : 74.94

dev : 79.17(**±0.38), test : 74.48(±0.33)**

3. aggr_out(1차원)대신 cycle count를 넣고 차원 확장 후 MLP를 돌려서 넣기(5차원)

- seed0(20231013_132522)
  - dev : 78.21, test : 74.62
- seed1(20231013_141416)
  - dev : 79.12, test : 75.34
- seed2(20231013_141443)
  - dev : 79.69, test : 74.94

dev : 79.04(**±0.62), test : 74.97(±0.30)**

4. [aggr_out; 차원 확장한 cycle count] 를 MLP에 돌려서 넣기(9+1 = 10차원)

- seed0(20231014_123412)
  - dev : 78.21, test : 74.62
- seed1(20231014_144913)

    - dev : 77.72, test : 72.20
- seed2(20231014_123536)
    - dev : 78.71, test : 75.02

dev : 78.21(**±0.40), test : 73.95(±1.25)**

New Cycle count

1. aggr_out(1차원)대신 cycle count를 그대로 넣기

- seed0(20231014_123743)
    - dev : 77.72, tet : 72.93
- seed1(20231014_123908)
    - dev : 77.15, test : 72.04
- seed2(20231014_124000)
    - dev : 77.81,test : 71.80

dev : 77.56(**±0.30), test : 72.26(±0.49)**

2. aggr_out(1차원)대신 cycle count를 넣고 차원 확장 후 MLP를 돌려서 넣기(9차원)

- seed0(20231015_065844)
    - dev : 78.05, test : 73.81
- seed1(20231015_065927)
    - dev : 78.54, test : 75.02
- seed2(20231015_065951)
    - dev : 78.38, test : 73.73

dev : 78.23(**±0.21), test : 74.19(±0.59)**

3. aggr_out(1차원)대신 cycle count를 넣고 차원 확장 후 MLP를 돌려서 넣기(5차원)

- seed0(20231015_070134)
    - dev : 79.03, test : 74.13
- seed1(20231015_070219)
    - dev : 79.12, test : 75.58
- seed2(20231015_070301)
    - dev : 79.36, test : 74.94

dev : 79.17(**±0.14), test : 74.88(±0.59)**

4. [aggr_out; 차원 확장한 cycle count] 를 MLP에 돌려서 넣기(9+1 = 10차원)

- seed0(20231015_111328)

    - dev : 78.79, test : 75.10
- seed1(20231015_111355)
    - dev : 79.44, test : 74.86
- seed2(20231015_111414)
    - dev : 79.20, test : 74.46

dev : 79.14(**±0.27), test : 74.81(±0.26)**

Random Number

1. aggr_out(1차원)대신 random count를 그대로 넣기

- seed0(20231015_124002)
    - dev : 77.07, test : 72.60
- seed1(20231015_124111)
    - dev : 77.31, test : 71.96
- seed2(20231015_124131)
    - dev : 76.58, test : 72.20

dev : 76.99(**±0.30), test : 72.25(±0.26)**

2. aggr_out(1차원)대신 random count를 넣고 차원 확장 후 MLP를 돌려서 넣기(9차원)

- seed0(20231015_135151)
    - dev : 79.52, test : 74.70
- seed1(20231015_135238)
    - dev : 79.20, test : 75.18
- seed2(20231015_135303)
    - dev : 78.79, test : 73.49

dev : 79.17,(**±0.30), test : 74.46(±0.71)**

3. aggr_out(1차원)대신 random count를 넣고 차원 확장 후 MLP를 돌려서 넣기(5차원)

- seed0(20231016_015536) → 20231016_113030
    - dev : 78.54, test : 74.54
- seed1(20231016_015616)
    - dev : 79.20,test : 75.18
- seed2(20231016_015630)
    - dev : 79.69, test : 75.26

dev : 79.14(**±0.47), test : 74.99(±0.32)**

4. [aggr_out; 차원 확장한 random count] 를 MLP에 돌려서 넣기(9+1 = 10차원)

- seed0(20231016_172706)
  - dev : 78.62, test : 74.05
- seed1(20231016_172810)
  - dev : 79.03, test : 74.78
- seed2(20231016_172825)
  - dev : 79.12, test : 75.50

dev : 78.92(**±0.22), test : 74.78(±0.59)**

Degree count

1. aggr_out(1차원)대신 degree count를 그대로 넣기

- seed0(20231017_041846)
  - dev : 77.00, test : 71.72
- seed1(20231017_042008)
  - dev : 77.81, test : 72.36
- seed2(20231017_042039)
  - dev : 78.54, test : 71.88

dev : 77.78(**±0.63), test : 71.99(±0.27)**

2. aggr_out(1차원)대신 degree count를 넣고 차원 확장 후 MLP를 돌려서 넣기(9차원)

- seed0(20231017_083748)
  - dev : 78.95, test : 74.29
- seed1(20231017_115119)
  - dev : 79.20, test : 74.62
- seed2(20231017_115140)
  - dev : 77.72, test : 72.68

dev : 78.62(**±0.65), test : 73.86(±0.85)**

3. aggr_out(1차원)대신 degree count를 넣고 차원 확장 후 MLP를 돌려서 넣기(5차원)

- seed0(20231017_154829) →20231018_042217
  - dev : 79.20, test : 74.54
- seed1(20231017_154908)
  - dev : 79.28, test : 75.10

- seed2(20231017_154914)
  - dev : 79.52, test : 74.78

dev : 79.33(**±0.14), test : 74.81(±0.23)**

4. [aggr_out; 차원 확장한 degree count] 를 MLP에 돌려서 넣기(9+1 = 10차원)

- seed0(20231018_042403)
  - dev : 79.69, test : 75.42
- seed1(20231018_042505) → 20231018_073332
  - dev : 78.05, test : 72.00
- seed2(20231018_042546)
  - dev : 80.10, test : 75.83

dev : 79.28(**±0.89), test : 74.42(±1.72)**

계속해서 실험만 했네?

이것만 하면

1. 기존에 진행했던 Cycle Encoder는 사이클 개수를 노드 임베딩 값에 주입했다면 이번에는 random number, 각 노드의 degree를 주입하는 방식으로 실험을 해보고자 한다.
2. New context score를 사용할 때 기존에는 Cycle Passage(세개의 트리플)를 사용했지만 이번에는 Triple을 대입해보고자 한다.

1번은 완료

2번을 진행해야 한다.