9월 14일

오늘 새벽에 위 1번까지 진행 해버림

그리고 upgrade cycle count와 orig cycle count를 제작함

Train 데이터의 Upgrade cycle count 분포

- max 값 : 40 → one-hot : 41
- 전처리해서 저장한 size : (4875,32)

DEV 데이터의 Upgrade cycle count 분포

- max 값 : 31 → one-hot : 32
- 전처리해서 저장한 size : (6105,32)

Train 데이터의 orig cycle count 분포

- max 값 : 492 → one-hot : 493
- 전처리해서 저장한 size : (4875,32)

DEV 데이터의 orig cycle count 분포

- max 값 : 268 → → one-hot : 269
- 전처리해서 저장한 size : (6105,32)Rㅊ

이제는 upgrade cycle count와 orig cycle count를 기반으로 실험을 진행해보자

1. 일단은 count 값을 그대로 사용해보기(concat이 아닌 진짜 더하기, 노드 임베딩 값이 0이므로)
   a. 이건 실패
   b. concat해서 해보기 → 2차원 concat을 진행하니 역시 실패
2. one-hot vector를 사용하기
   a. upgrade 만 먼저 해보기
      - cycle이 없는 노드의 cycle count를 -1로 지정하고 one-hot vector로 변경
        - 이렇게 되면 최솟값 -1, 최댓값 :40(train), 31(dev), edge_embedding은 추가 ㄴㄴ
   b. edge embedding에도 추가하기
   c. orig, upgrade, 이 두개를 구분해주는 flag까지 사용하기

For converting a one-hot encoded vector into a dense embedding vector, using `nn.Embedding` is a more appropriate choice. Here's why:

1. **Specifically Designed for Categorical Data**: `nn.Embedding` is designed to handle categorical or discrete data, which includes one-hot encoded vectors. It efficiently maps discrete indices (corresponding to categories) to dense continuous vectors (embeddings).
2. **Memory Efficiency**: One-hot encoded vectors are sparse by nature, with most values being zero. `nn.Embedding` takes advantage of this sparsity by only storing and processing the non-zero values. This results in significant memory savings compared to using a dense layer like the `MLP`, which would treat all input features as non-zero.

3. **Dimension Reduction**: When you convert a one-hot encoded vector using `nn.Embedding`, you effectively reduce the dimensionality from the number of categories to the size of the embedding vector. This can be especially important when dealing with high-dimensional categorical features.
4. **Learnable Embeddings**: The embedding vectors produced by `nn.Embedding` are learnable, meaning the model can update them during training to better represent the underlying data patterns. This is beneficial when you want your model to capture meaningful relationships between categories.

그래서 context score 를 만들기 위한 코드를 만들다가 뭔가 cycle_passage가 이상해서 확인해보았다.

보니까 context노드들이 제대로 text화가 되지 않았다…

{"answerKey": "A", "id": "075e483d21c29a511267ef62bedc0461", "question": {"question_concept": "punishing", "choices": [{"label": "A", "text": "ignore"}, {"label": "B", "text": "enforce"}, {"label": "C", "text": "authoritarian"}, {"label": "D", "text": "yell at"}, {"label": "E", "text": "avoid"}],

"stem": "The sanctions against the school were a punishing blow, and they seemed to what the efforts the school had made to change?"},

"statements":

[{"label": true, "statement": "The sanctions against the school were a punishing blow, and they seemed to ignore the efforts the school had made to change."},

{"label": false, "statement": "The sanctions against the school were a punishing blow, and they seemed to enforce the efforts the school had made to change."},

{"label": false, "statement": "The sanctions against the school were a punishing blow, and they seemed to authoritarian the efforts the school had made to change."},

{"label": false, "statement": "The sanctions against the school were a punishing blow, and they seemed to yell at the efforts the school had made to change."},

{"label": false, "statement": "The sanctions against the school were a punishing blow, and they seemed to avoid the efforts the school had made to change."}]}

{"answerKey": "A", "id": "4c1cb0e95b99f72d55c068ba0255c54d", "question": {"question_concept": "choker", "choices": [{"label": "A", "text": "jewelry store"}, {"label": "B", "text": "neck"}, {"label": "C", "text": "jewlery box"}, {"label": "D", "text": "jewelry box"}, {"label": "E", "text": "boutique"}], "stem": "To locate a choker not located in a jewelry box or boutique where would you go?"},

"statements": [{"label": true, "statement": "choker is a jewelry and jewelry is related to boutique and boutique is considered as the location of choker<SEP>choker is at location of boutique and boutique is related to jewelry and jewelry is a choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is a choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is related to choker<SEP>choker is a jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker<SEP>choker is related to jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker"},

{"label": false, "statement": "choker is a jewelry and jewelry is related to boutique and boutique is considered as the location of choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is related to choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is a choker<SEP>choker is at location of boutique and boutique is related to jewelry and jewelry is a choker<SEP>choker is a jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker"},

{"label": false, "statement": "choker is a jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is a choker<SEP>choker is at location of boutique and boutique is related to jewelry and jewelry is a choker<SEP>choker is at location of boutique and boutique is related to jewelry and jewelry is related to choker<SEP>jewelry_box is considered as the location of jewelry and jewelry is at location of box and box is related to jewelry_box"}, {"label": false, "statement": "choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is related to choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is a choker<SEP>choker is a jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker<SEP>choker is related to jewelry and jewelry is at location of jewelry_box and jewelry_box is considered as the location of choker<SEP>box is considered as the location of jewelry and jewelry is at location of jewelry_box and jewelry_box is related to box"},

{"label": false, "statement": "choker is at location of boutique and boutique is related to jewelry and jewelry is a choker<SEP>choker is related to jewelry and jewelry is related to boutique and boutique is considered as the location of choker<SEP>choker is at location of boutique and boutique is related to jewelry and jewelry is related to choker<SEP>choker is a jewelry and jewelry is related to boutique and boutique is considered as the location of choker<SEP>choker is at location of jewelry_box and jewelry_box is considered as the location of jewelry and jewelry is a choker"}]}

{"answerKey": "C", "id": "23505889b94e880c3e89cff4ba119860", "question": {"question_concept": "fox", "choices": [{"label": "A", "text": "pretty flowers."}, {"label": "B", "text": "hen house"}, {"label": "C", "text": "natural habitat"}, {"label": "D", "text": "storybook"}, {"label": "E", "text": "dense forest"}], "stem": "The fox walked from the city into the forest, what was it looking for?"}, "statements": [{"label": false, "statement": ""}, {"label": false, "statement": "varying_camera_focal_point is context and that is answer hen_house and hen_house is considered as the location of fox and fox is question and that is context varying_camera_focal_point<SEP>varying_camera_focal_point is context and that is question fox and fox is at location of hen_house and hen_house is answer and that is context varying_camera_focal_point<SEP>varying_camera_focal_point is context and that is question fox and fox is related to hen_house and hen_house is answer and that is context varying_camera_focal_point<SEP>varying_camera_focal_point is context and that is answer hen_house and hen_house is related to fox and fox is question and that is context varying_camera_focal_point"}, {"label": true, "statement": ""}, {"label": false, "statement": ""}, {"label": false, "statement": ""}]}

success_rerank.py

```
import torch
import numpy as np
import gzip
import pickle
import time
import statistics




from sentence_transformers import CrossEncoder
import random
# reranking
def rank_documents_for_index_high(qa_context,BM25,DPR): # qa_context : 48705(6105), bm25 : 48705(6105)
    top_results = []  # List to store the top results
    model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-12-v2', max_length=256)

    for i, query in enumerate(qa_context):
        combined_result = list(set([(idx, paragraph) for idx, paragraph in BM25[i]+DPR[i]]))
        random.shuffle(combined_result)

        # Calculate similarity scores using the model
        scores = []
        for idx, doc_text in combined_result:
            if not doc_text:
                # Skip calculation for empty documents
                continue
            score = model.predict([(query, doc_text)])[0]
            scores.append(score)

        # Combine the scores with the corresponding paragraphs
        combined_scores = list(zip(combined_result, scores))

        # Sort the combined_scores by score in descending order
        combined_scores.sort(key=lambda x: x[1], reverse=True)
        if len(combined_scores) < 5:
            top_n = len(combined_scores)
        elif 5 <= len(combined_scores) < 10:
            top_n = 5
        elif 10 <= len(combined_scores) < 20:
            top_n = int(0.5 * len(combined_scores))
        elif 20 <= len(combined_scores) < 40:
            top_n = 10
        elif 40 <= len(combined_scores) < 60:
            top_n = 20
        elif 60 <= len(combined_scores) < 100:
            top_n = 30
        else:
            top_n = 40
        # Append the indexes from combined_result to top_results
        top_indexes = [item[0][0] for item in combined_scores[:top_n]]
        top_results.append(top_indexes)
        print('i처리',i)
    return top_results


if __name__ =='__main__':


    # with open('./data/train_qa_context.pk','rb') as f:
    #     train_qa_context = pickle.load(f)
    # with open('./data/train_DPR_result.pkl','rb') as f:
    #     train_DPR_result = pickle.load(f)
    # with open('./data/train_BM25_result.pkl','rb') as f:
    #     train_BM25_result = pickle.load(f)

    # flattened_trian_qa_context = [query for problem in train_qa_context for query in problem]


    # train_rerank_index=rank_documents_for_index_high(flattened_trian_qa_context,train_BM25_result,train_DPR_result)

    # print(len(train_rerank_index))
    # with open('./data/train_rerank_index.pkl','wb') as f:
    #     pickle.dump(train_rerank_index,f)

    with open('./data/preprocess/CSQA/dev/dev_qa_context.pk','rb') as f:
        dev_qa_context = pickle.load(f)
    with open('./data/preprocess/CSQA/dev/dev_DRP_result1.pkl','rb') as f:
        dev_DPR_result = pickle.load(f)
    with open('./data/preprocess/CSQA/dev/dev_BM25_result1.pkl','rb') as f:
        dev_BM25_result = pickle.load(f)
    with open('./data/preprocess/CSQA/dev/dev_cycle_passage.pkl','rb') as f:
        dev_cycle_passage = pickle.load(f)
    flattened_dev_qa_context = [query for problem in dev_qa_context for query in problem]
    flattened_dev_cycle_passage = [doc_list if doc_list else [] for problem in dev_cycle_passage for doc_list in problem]

    dev_rerank_index=rank_documents_for_index_high(flattened_dev_qa_context,dev_BM25_result,dev_DPR_result)

    print(len(dev_rerank_index))
    with open('./data/preprocess/CSQA/dev/dev_rerank_index1.pkl','wb') as f:
        pickle.dump(dev_rerank_index,f)

    dev_rerank_text = []
    dev_rerank_only_text = []
    for i in range(len(dev_rerank_index)):  # Iterate over train_dpr_result
        if not flattened_dev_cycle_passage[i]:  # Handle cases where flattened_train_cycle_passage is empty
            dev_rerank_text.append([])
            continue

        pairs = [(j, flattened_dev_cycle_passage[i][j][0]) for j in dev_rerank_index[i]]
        pairs1 =[flattened_dev_cycle_passage[i][j][0] for j in dev_rerank_index[i]]
        dev_rerank_text.append(pairs)  # Extend the train_DPR_result list with pairs
        dev_rerank_only_text.append(pairs1)
    print(len(dev_rerank_text))

    with open('./data/preprocess/CSQA/dev/dev_rerank_text1.pkl','wb') as f:
        pickle.dump(dev_rerank_text,f)
    with open('./data/preprocess/CSQA/dev/dev_rerank_only_text1.pkl','wb') as f:
        pickle.dump(dev_rerank_only_text,f)
```

success_DPR.py

```
import faiss
import numpy as np
from transformers import DPRContextEncoder, DPRContextEncoderTokenizer, DPRQuestionEncoder, DPRQuestionEncoderTokenizer
import torch
import numpy as np
import gzip
import pickle
import time
import statistics
import sys
def calculate_dpr_similarity_with_faiss_gpu(model_name_or_path, qa_context, cycle_passage):
    # Load the DPR models and tokenizers
    context_model = DPRContextEncoder.from_pretrained(model_name_or_path)
    context_tokenizer = DPRContextEncoderTokenizer.from_pretrained(model_name_or_path)
    question_model = DPRQuestionEncoder.from_pretrained(model_name_or_path)
    question_tokenizer = DPRQuestionEncoderTokenizer.from_pretrained(model_name_or_path)

    top_documents = []

    for query_list, doc_list_for_query in zip(qa_context, cycle_passage):
        if not doc_list_for_query:
            top_documents.append([])
            continue

        query = query_list if query_list else ""
        doc_texts = [doc[0] for doc in doc_list_for_query if doc]

        # Tokenize and encode the query and documents using the DPR models
        query_inputs = context_tokenizer(query, return_tensors="pt", padding=True, truncation=True)
        doc_inputs = context_tokenizer(doc_texts, return_tensors="pt", padding=True, truncation=True)

        with torch.no_grad():
            query_embeddings = context_model(**query_inputs).pooler_output
            passage_embeddings = context_model(**doc_inputs).pooler_output


        # Initialize a GPU device
        res = faiss.StandardGpuResources()

        # Initialize your FAISS index on the GPU
        dimension = 768  # The dimension of DPR embeddings
        index = faiss.index_factory(dimension, "Flat", faiss.METRIC_INNER_PRODUCT)
        index = faiss.index_cpu_to_all_gpus(index)

        # Add data to the GPU-based index
        index.add(passage_embeddings)

        print(index.ntotal)
        # Perform similarity search on the GPU
        # Calculate top_n based on the length of doc_texts
        print(passage_embeddings.shape[0]==len(doc_texts))


        if len(doc_texts) < 5:
            top_n = len(doc_texts)
        else:
            top_n = int(0.2 * len(doc_texts))

        distances, top_doc_indices = index.search(query_embeddings, top_n)


        top_docs = top_doc_indices[0].tolist()  # Convert to a list of top document indices

        top_documents.append(top_docs)
        print('hi')

    return top_documents



if __name__ =='__main__':


    with open('./data/preprocess/CSQA/train/train_cycle_passage.pkl','rb') as f:
        train_cycle_passage = pickle.load(f)
    with open('./data/preprocess/CSQA/train/train_qa_context.pk','rb') as f:
        train_qa_context = pickle.load(f)
    model_name_or_path = "facebook/dpr-ctx_encoder-multiset-base"
    flattened_train_cycle_passage = [doc_list if doc_list else [] for problem in train_cycle_passage for doc_list in problem]
    flattened_train_qa_context = [query for problem in train_qa_context for query in problem]


    print('Start')
    a = time.time()

    top_doc = calculate_dpr_similarity_with_faiss_gpu(model_name_or_path, qa_context=flattened_train_qa_context, cycle_passage=flatten
    print('길잉',len(top_doc))
    train_DPR_result = []

    for i in range(len(top_doc)):  # Iterate over train_dpr_result
        if not flattened_train_cycle_passage[i]:  # Handle cases where flattened_train_cycle_passage is empty
            train_DPR_result.append([])
            continue

        pairs = [(j, flattened_train_cycle_passage[i][j][0]) for j in top_doc[i]]
    dev_train_result.append(pairs)  # Extend the train_DPR_result list with pairs

print(len(dev_DPR_result))




with open('./data/preprocess/CSQA/train/train_DRP_result1.pkl','wb') as f:
    pickle.dump(train_DPR_result,f)


print(time.time()-a) #

# with open('./data/preprocess/CSQA/dev/dev_cycle_passage.pkl','rb') as f:
#     dev_cycle_passage = pickle.load(f)
# with open('./data/preprocess/CSQA/dev/dev_qa_context.pk','rb') as f:
#     dev_qa_context = pickle.load(f)
# model_name_or_path = "facebook/dpr-ctx_encoder-multiset-base"
# flattened_dev_cycle_passage = [doc_list if doc_list else [] for problem in dev_cycle_passage for doc_list in problem]
# flattened_dev_qa_context = [query for problem in dev_qa_context for query in problem]


# print('Start')
# a = time.time()

# top_doc = calculate_dpr_similarity_with_faiss_gpu(model_name_or_path, qa_context=flattened_dev_qa_context, cycle_passage=flatten
# print('길잉',len(top_doc))
# dev_DPR_result = []

# for i in range(len(top_doc)):  # Iterate over train_dpr_result
#     if not flattened_dev_cycle_passage[i]:  # Handle cases where flattened_train_cycle_passage is empty
#         dev_DPR_result.append([])
#         continue

#     pairs = [(j, flattened_dev_cycle_passage[i][j][0]) for j in top_doc[i]]

#     dev_DPR_result.append(pairs)  # Extend the train_DPR_result list with pairs

# print(len(dev_DPR_result))




# with open('./data/preprocess/CSQA/dev/dev_DRP_result1.pkl','wb') as f:
#     pickle.dump(dev_DPR_result,f)


# print(time.time()-a) # 3175.5436959266663
```