9월 19일

```
document_cnt = []
for i in range(len(flattened_train_cycle_passage)):
    document_cnt.append(len(flattened_train_cycle_passage[i]))


print('cycle_passage가 제일 많은 개수',max(document_cnt))
print('cycle_passage가 제일 적은 개수',min(document_cnt))
print('total cycle_passage 개수',sum(document_cnt))
print('평균 cycle_passage 개수',sum(document_cnt)/len(document_cnt))

median_value = statistics.median(document_cnt)
print('cycle_passage 개수 중간 값',median_value)
```

1. 노드 64개로 만드는 과정 DPR 부터 진행하면 됨
2. 노드 32개는 오늘 밤에 colab에 돌리기

그러면 인제 진행해야 할 것은 세이프디…

한 번 정리해보자면 다음과 같다.

1. object detection을 통해 건설현장에서 위험요인의 위치 인식
2.
3. 인식된 이미지(bounding box)와 위험요인(text)를 simVLM모델에 넣는다.
4. 넣어서 나오는 output은 재해예방대책
5. classification 부분에서 재해예방대책과

```
import faiss
import torch
from transformers import DPRContextEncoder, DPRContextEncoderTokenizer, DPRQuestionEncoder, DPRQuestionEncoderTokenizer
# Initialize a GPU device
# res = faiss.StandardGpuResources()
ngpus = faiss.get_num_gpus()

print("number of GPUs:", ngpus)

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 = question_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
        cpu_index = faiss.index_factory(dimension, "Flat", faiss.METRIC_INNER_PRODUCT)
        # index = faiss.IndexFlatL2(dimension)
        gpu_index = faiss.index_cpu_to_all_gpus(  # build the index
            cpu_index
        )

        # Convert PyTorch tensors to NumPy arrays
        query_embeddings_np = query_embeddings.cpu().contiguous().numpy()
        passage_embeddings_np = passage_embeddings.cpu().contiguous().numpy()

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

        # Calculate top_n based on the length of doc_texts
        if len(doc_texts) < 50:
            top_n = len(doc_texts)
        else:
            top_n = 50

        distances, top_doc_indices = gpu_index.search(query_embeddings_np, top_n)

        top_docs = top_doc_indices[0].tolist()  # Convert to a list of top document indices
        # print(top_docs)
        # assert -1 == 0
        top_documents.append(top_docs)
        print('hi')

    return top_documents
```

노드 64개를 대상으로하는 DPR은 너무 오래걸리는 상황

→ 일단 이것은 BM25 + reranking만 진행하자

노드 32개로 하는것은 일단 새로 진행해야하므로 DPR 진행 중

중간에 바람쐬러 나가서 생각한것 정리

- [x] ~~GSC에서 노드 개수 32개가 only qanode라고 적혀있지만 only qanode가 아님~~
- [x] ~~DPR algorithm을 노드 개수 64개를 대상으로 진행하고 있는거 중단하기~~
- [ ] original gsc 돌리기
- [ ] edge masking 생각하기
    - 기존에는 graph-pruning을 생각했었음. → 이것은 cycle count 실험을 완료 후 진행
- [x] ~~노드개수 64개에 대해서 BM25 적용하기~~
    - [x] ~~reranking과정도 진행하기에 앞서 비율 상태 확인 + dev 데이터도 진행하기~~
- [ ] reranking 진행하기 → 진행중
- [ ] 32개에 대해서 DPR 알고리즘은 colab에서 진행 중, 진행 완료하고 전처리 끝까지(statement.jsonl) 진행하기

```
import json

# Load the original data from the existing input_file
original_data = []
with open('./data/csqa/statement/dev.statement.jsonl', 'r', encoding='utf-8') as f:
    for line in f.readlines():
        data = json.loads(line)
        original_data.append(data)
```

```
# Create a new list to hold the modified data
new_data = []

# Now, for each example, modify the 'statements' to match 'rerank_text'
# rerank_only_text : 1221
for data, rerank_labels in zip(original_data, dev_rerank_only_text):  # rerank_labels = 1번문제
    for statement, rerank_text_list in zip(data["statements"], rerank_labels): # rerank_text_list = 1번 문제 1번 정답
        if not rerank_text_list:
            statement["statement"] = ""
            continue


        statement["statement"] = "<SEP>".join(rerank_text_list)
        # print(statement['statement'])


    # Append the modified data to the new list
    new_data.append(data)

# Write the modified data to a new JSONL file
output_file_path = './data/csqa/statement/dev_modified.statement.jsonl'
with open(output_file_path, 'w', encoding='utf-8') as output_file:
    for data in new_data:
        json.dump(data, output_file, ensure_ascii=False)
        output_file.write('\n')

print(f"Modified data saved to {output_file_path}")
```

1. 목요일까지 cycle count에 대해서 실험을 진행한다. 실험은 0,1,2, 3개의 ssed에 대해서 실험을 진행한다.
2. 실험 순서는 다음과 같다. 각 방법은 노드 개수 32, 64개를 대상으로 진행한다.
    a. original gsc
    b. original gsc + original cycle count
    c. original gsc + new cycle count(not one-hot vector)
    d. original gsc + new cycle count(one-hot vector)
    e. new context score

New context score 제작 코드

```
def load_bert_xlnet_roberta_input_tensors(statement_jsonl_path, model_type, model_name, max_seq_length, pretrained=None):
    cache_path = statement_jsonl_path + '.' + model_name +'.sentvecs.loaded_cache'
    import json

    class InputExample(object):
        def __init__(self, example_id, question, contexts, endings, label=None, statements=None):
            self.example_id = example_id
            self.question = question
            self.contexts = contexts
            self.endings = endings

            self.label = label
            self.statements = statements  # Add statements attribute
    class InputFeatures(object):

        def __init__(self, example_id, choices_features, label):
            self.example_id = example_id
            self.choices_features = [
                {
                    'input_ids': input_ids,
                    'input_mask': input_mask,
                    'segment_ids': segment_ids,
                    'output_mask': output_mask,
                }
                for _, input_ids, input_mask, segment_ids, output_mask in choices_features
            ]
            self.label = label

    def read_examples(input_file,input_file1='./data/csqa/statement/train_modified.statement.jsonl'):
        with open(input_file, "r", encoding="utf-8") as f:
            examples = []
            for line in f.readlines():
                json_dic = json.loads(line)
                label = ord(json_dic["answerKey"]) - ord("A") if 'answerKey' in json_dic else 0
```

```
                contexts = json_dic["question"]["stem"]
                if "para" in json_dic:
                    contexts = json_dic["para"] + " " + contexts
                if "fact1" in json_dic:
                    contexts = json_dic["fact1"] + " " + contexts
                examples.append(
                    InputExample(
                        example_id=json_dic["id"],
                        contexts=[contexts] * len(json_dic["question"]["choices"]), # 이건 말그대로 문제(question)
                        # contexts=[statement["statement"] for statement in json_dic["statements"]],
                        # contexts=[ending["text"] + "." for ending in json_dic["question"]["choices"]],
                        question="",
                        # question=json_dic["question"]["stem"],
                        endings=[ending["text"] for ending in json_dic["question"]["choices"]],
                        # endings=["" for ending in json_dic["question"]["choices"]],
                        label=label
                    ))
        # print('input_file',examples[1].contexts)
        # print('################################')
        # print('input_file',examples[1].endings)

        # Now, add examples from input_file1
        with open('./data/csqa/statement/train_modified.statement.jsonl', "r", encoding="utf-8") as f:
            examples1 = []
            for line in f.readlines():
                json_dic = json.loads(line)
                label = ord(json_dic["answerKey"]) - ord("A") if 'answerKey' in json_dic else 0
                contexts=[statement["statement"] for statement in json_dic['statements']]

                # print('contexts',contexts)
                # assert -1 == 0
                # if "para" in json_dic:
                #     contexts = json_dic["para"] + " " + contexts
                # if "fact1" in json_dic:
                #     contexts = json_dic["fact1"] + " " + contexts


                examples1.append(
                    InputExample(
                        example_id=json_dic["id"],
                        contexts=contexts, # 이건 말그대로 문제(question)
                        question="",
                        # endings=[ending["text"] for ending in json_dic["question"]["choices"]],
                        endings="",
                        # endings=["" for ending in json_dic["question"]["choices"]],
                        label=label
                    ))

        # print('input_file1',examples1[1].contexts)
        # print('################################')
        # print(examples1[1].endings)
        # assert -1 == 0
        return examples,examples1

    # class InputExample(object):

    #     def __init__(self, example_id, question, contexts, endings, label=None):
    #         self.example_id = example_id
    #         self.question = question
    #         self.contexts = contexts
    #         self.endings = endings
    #         self.label = label

    # class InputFeatures(object):

    #     def __init__(self, example_id, choices_features, label):
    #         self.example_id = example_id
    #         self.choices_features = [
    #             {
    #                 'input_ids': input_ids,
    #                 'input_mask': input_mask,
    #                 'segment_ids': segment_ids,
    #                 'output_mask': output_mask,
    #             }
    #             for _, input_ids, input_mask, segment_ids, output_mask in choices_features
    #         ]
    #         self.label = label

    # def read_examples(input_file):
    #     with open(input_file, "r", encoding="utf-8") as f:
    #         examples = []
    #         for line in f.readlines():
    #             json_dic = json.loads(line)
    #             label = ord(json_dic["answerKey"]) - ord("A") if 'answerKey' in json_dic else 0
```

```
    #             contexts = json_dic["question"]["stem"]
    #             if "para" in json_dic:
    #                 contexts = json_dic["para"] + " " + contexts
    #             if "fact1" in json_dic:
    #                 contexts = json_dic["fact1"] + " " + contexts
    #             examples.append(
    #                 InputExample(
    #                     example_id=json_dic["id"],
    #                     contexts=[contexts] * len(json_dic["question"]["choices"]), # 이건 말그대로 문제(question)
    #                     # contexts=[statement["statement"] for statement in json_dic["statements"]],
    #                     # contexts=[ending["text"] + "." for ending in json_dic["question"]["choices"]],
    #                     question="",
    #                     # question=json_dic["question"]["stem"],
    #                     endings=[ending["text"] for ending in json_dic["question"]["choices"]],
    #                     # endings=["" for ending in json_dic["question"]["choices"]],
    #                     label=label
    #                 ))


    #             # print('examples',examples[0].endings)
    #             '''examples ['The sanctions against the school were a punishing blow, and they seemed to what the efforts the school
    #             'The sanctions against the school were a punishing blow, and they seemed to what the efforts the school had made to c
    #             'The sanctions against the school were a punishing blow, and they seemed to what the efforts the school had made to c
    #             'The sanctions against the school were a punishing blow, and they seemed to what the efforts the school had made to c
    #             'The sanctions against the school were a punishing blow, and they seemed to what the efforts the school had made to c
    #             '''
    #             # assert -1 == 0
    #     return examples

    def convert_examples_to_features(examples,examples1, label_list, max_seq_length,
                                     tokenizer,
                                     cls_token_at_end=False,
                                     cls_token='[CLS]',
                                     cls_token_segment_id=1,
                                     sep_token='[SEP]',
                                     sequence_a_segment_id=0,
                                     sequence_b_segment_id=1,
                                     sequence_c_segment_id=2,
                                     sep_token_extra=False,
                                     pad_token_segment_id=0,
                                     pad_on_left=False,
                                     pad_token=0,
                                     mask_padding_with_zero=True):
        """ Loads a data file into a list of `InputBatch`s
            `cls_token_at_end` define the location of the CLS token:
                - False (Default, BERT/XLM pattern): [CLS] + A + [SEP] + B + [SEP]
                - True (XLNet/GPT pattern): A + [SEP] + B + [SEP] + [CLS]
            `cls_token_segment_id` define the segment id associated to the CLS token (0 for BERT, 2 for XLNet)
        """
        label_map = {label: i for i, label in enumerate(label_list)}
        tokenc_len = []
        features = []
        for ex_index, (example,example1) in enumerate(zip(examples,examples1)):
            # print('ex_index',ex_index)
            # print(example)
            # print('example',example.contexts)
            # print('examples1',example1.contexts)
            # print('example1',example.endings)
            # print('example.question',example.question)
            # assert -1 == 0
            choices_features = []
            for ending_idx, (context, ending,cycle_passage) in enumerate(zip(example.contexts, example.endings,example1.contexts)):
                tokens_a = tokenizer.tokenize(context)
                tokens_b = tokenizer.tokenize(example.question + " " + ending)
                # print('tokens_a',tokens_a)
                # print('tokens_b',tokens_b)
                # assert -1 == 0
                special_tokens_count = 4 if sep_token_extra else 3
                if not cycle_passage:

                    continue
                else:

                    special_tokens_count += cycle_passage.count('<SEP>')
                    # cycle_passage = cycle_passage.replace()
                    cycle_passage = cycle_passage.replace('<SEP>',' ')

                    tokens_c = tokenizer.tokenize(example1.question + " " + cycle_passage)
                    # print('tokens_c',tokens_c)
                    # print(len(tokens_c)) # 412
                tokenc_len.append(len(tokens_c))

                # assert -1 == 0
                # special_tokens_count = 4 if sep_token_extra else 3
```

```
                # special_tokens_count = 5 if sep_token_extra else 3
                # print('special_tokens_count',special_tokens_count)  # 4
                # assert -1 == 0
                _truncate_seq_pair(tokens_a, tokens_b,tokens_c, max_seq_length - special_tokens_count)

                # print('after_tokens_a',tokens_a)
                # print('after_tokens_b',tokens_b)
                # print('after_tokens_c',tokens_c)
                # print(len(tokens_c))
                # print('special_token_extra',sep_token_extra) # True
                # assert -1 == 0

                    # The convention in BERT is:
                    # (a) For sequence pairs:
                    #  tokens:   [CLS] is this jack ##son ##ville ? [SEP] no it is not . [SEP]
                    #  type_ids:   0   0  0    0    0     0       0   0   1  1  1  1   1  1
                    # (b) For single sequences:
                    #  tokens:   [CLS] the dog is hairy . [SEP]
                    #  type_ids:   0   0   0   0  0     0   0
                    #
                    # Where "type_ids" are used to indicate whether this is the first
                    # sequence or the second sequence. The embedding vectors for `type=0` and
                    # `type=1` were learned during pre-training and are added to the wordpiece
                    # embedding vector (and position vector). This is not *strictly* necessary
                    # since the [SEP] token unambiguously separates the sequences, but it makes
                    # it easier for the model to learn the concept of sequences.
                    #
                    # For classification tasks, the first vector (corresponding to [CLS]) is
                    # used as as the "sentence vector". Note that this only makes sense because
                    # the entire model is fine-tuned.
                tokens = tokens_a + [sep_token]
                if sep_token_extra:
                    # roberta uses an extra separator b/w pairs of sentences
                    tokens += [sep_token]
                    # print('tokens',tokens)
                    # assert -1 == 0

                segment_ids = [sequence_a_segment_id] * len(tokens)
                # print('segment_ids',segment_ids)
                # print(len(segment_ids))
                # assert -1 == 0

                if tokens_b and tokens_c: # roberta 경우에는 tokens_b까지 type_ids를 0으로봄 그래야 qa_context가 되기 때문임
                    tokens += tokens_b + [sep_token]
                    segment_ids += [sequence_b_segment_id] * (len(tokens_b) + 1)

                    # print('segment_ids',segment_ids)
                    # print(len(segment_ids))
                    # print('tokensb',tokens)
                    # print('cls_token_at_end',cls_token_at_end) # false

                    tokens += tokens_c + [sep_token]
                    segment_ids += [sequence_c_segment_id] * (len(tokens_c)+1)
                    # print('3segment_ids',segment_ids)
                    # print(len(segment_ids))
                    # print('3tokensb',tokens)
                    # print('3cls_token_at_end',cls_token_at_end) # false
                    # assert -1 == 0




                # Roberta 인ㄱ 경우 cls_token_segment_id = 0

                if cls_token_at_end:
                    tokens = tokens + [cls_token]
                    segment_ids = segment_ids + [cls_token_segment_id]
                else:
                    tokens = [cls_token] + tokens
                    # print('##############################')
                    # print('tokens',tokens)
                    # assert -1 == 0
                    segment_ids = [cls_token_segment_id] + segment_ids
                    # print('3segment_ids',segment_ids)
                    # print(len(segment_ids))
                    # assert -1 == 0
                input_ids = tokenizer.convert_tokens_to_ids(tokens)
                # print('input_ids',input_ids) # 길이 : 29
                # print(len(input_ids))
                # assert -1 == 0
                # print('mask_padding_with_zero',mask_padding_with_zero) # True
                # print('Cls_token',cls_token)
                # print('sep_token',sep_token)
                # print('max_seq_length',max_seq_length) # 88
```

```
                # assert -1 == 0
                # The mask has 1 for real tokens and 0 for padding tokens. Only real
                # tokens are attended to.

                input_mask = [1 if mask_padding_with_zero else 0] * len(input_ids)
                # print('input_mask',input_mask)
                # print(len(input_mask))
                # assert -1 == 0
                special_token_id = tokenizer.convert_tokens_to_ids([cls_token, sep_token])
                # print('special_token_id',special_token_id)
                # print(len(special_token_id))

                output_mask = [1 if id in special_token_id else 0 for id in input_ids]  # 1 for mask
                # print('output_mask',output_mask)
                # assert -1 == 0
                # Zero-pad up to the sequence length.

                padding_length = max_seq_length - len(input_ids)
                # print(max_seq_length) # 88
                # print('padding_length',padding_length)
                # print('pad_on_left',pad_on_left) # false
                # print('pad_token',pad_token)
                # assert -1 == 0
                if pad_on_left:
                    input_ids = ([pad_token] * padding_length) + input_ids
                    input_mask = ([0 if mask_padding_with_zero else 1] * padding_length) + input_mask
                    output_mask = ([1] * padding_length) + output_mask

                    segment_ids = ([pad_token_segment_id] * padding_length) + segment_ids
                else:
                    input_ids = input_ids + ([pad_token] * padding_length)
                    # print('input_ids',input_ids)
                    # print(len(input_ids))
                    # assert -1 == 0
                    input_mask = input_mask + ([0 if mask_padding_with_zero else 1] * padding_length)
                    output_mask = output_mask + ([1] * padding_length)
                    segment_ids = segment_ids + ([pad_token_segment_id] * padding_length)
                    # print('input_mask',input_mask)
                    # print(len(input_mask))
                    # print('output_mask',output_mask)
                    # print(len(output_mask))
                    # print('segment_ids',segment_ids)
                    # print(len(segment_ids))
                    # assert -1 == 0

                assert len(input_ids) == max_seq_length
                assert len(output_mask) == max_seq_length
                assert len(input_mask) == max_seq_length
                assert len(segment_ids) == max_seq_length
                choices_features.append((tokens, input_ids, input_mask, segment_ids, output_mask))
            label = label_map[example.label]
            features.append(InputFeatures(example_id=example.example_id, choices_features=choices_features, label=label))
        print(max(tokenc_len))
        return features

    # def _truncate_seq_pair(tokens_a, tokens_b,tokens_c, max_length):
    #     """Truncates a sequence pair in place to the maximum length."""

    #     # This is a simple heuristic which will always truncate the longer sequence
    #     # one token at a time. This makes more sense than truncating an equal percent
    #     # of tokens from each, since if one sequence is very short then each token
    #     # that's truncated likely contains more information than a longer sequence.
    #     # print('max_length',max_length) # 84
    #     # assert -1 == 0
    #     while True:
    #         total_length = len(tokens_a) + len(tokens_b) + len(tokens_c)
    #         if total_length <= max_length:
    #             break
    #         if len(tokens_a) > len(tokens_b):
    #             tokens_a.pop()
    #         else:
    #             tokens_b.pop()
    def _truncate_seq_pair(tokens_a, tokens_b, tokens_c, max_length):
        """Truncates a sequence pair in place to the maximum length."""

        # Calculate the total length of the three sequences
        total_length = len(tokens_a) + len(tokens_b) + len(tokens_c)

        # Check if truncation is necessary
        if total_length <= max_length:
            return  # No truncation needed

        # Calculate how many tokens to remove from each sequence
        tokens_to_remove = total_length - max_length
```

```
    # Remove tokens from the sequences
    while tokens_to_remove > 0:
        # Remove tokens from tokens_c if there are any left to remove
        if len(tokens_c) > 0:
            tokens_c.pop()
        # Remove tokens from tokens_b if there are any left to remove
        elif len(tokens_b) > 0:
            tokens_b.pop()
        # Remove tokens from tokens_a if there are any left to remove
        elif len(tokens_a) > 0:
            tokens_a.pop()
        tokens_to_remove -= 1


def select_field(features, field):
    return [[choice[field] for choice in feature.choices_features] for feature in features]

def convert_features_to_tensors(features):
    all_input_ids = torch.tensor(select_field(features, 'input_ids'), dtype=torch.long)
    all_input_mask = torch.tensor(select_field(features, 'input_mask'), dtype=torch.long)
    all_segment_ids = torch.tensor(select_field(features, 'segment_ids'), dtype=torch.long)
    all_output_mask = torch.tensor(select_field(features, 'output_mask'), dtype=torch.bool)
    all_label = torch.tensor([f.label for f in features], dtype=torch.long)
    # print('pretrained',pretrained) # None
    # assert -1==0
    if pretrained:

        if os.path.exists(cache_path):
            with open(cache_path, 'rb') as f:
                all_sent_vecs, all_logits = pickle.load(f)
        else:
            model, old_args = torch.load(pretrained)
            _all_input_ids = all_input_ids.view(-1, all_input_ids.size(-1)).cuda()
            _all_input_mask = all_input_mask.view(-1, all_input_mask.size(-1)).cuda()
            _all_segment_ids = all_segment_ids.view(-1, all_segment_ids.size(-1)).cuda()
            _all_output_mask =  all_output_mask.view(-1, all_output_mask.size(-1)).cuda()
            model.cuda()
            model.eval()
            bs = 256
            n = _all_input_ids.size(0)
            all_sent_vecs = []
            all_logits = []
            for a in tqdm(range(0, n, bs)):
                b = min(n, a + bs)
                lm_inputs = _all_input_ids[a:b], _all_input_mask[a:b], _all_segment_ids[a:b], _all_output_mask[a:b]
                with torch.no_grad():
                    sent_vecs, all_hidden_states = model.encoder(*lm_inputs)
                    logits = model.decoder(sent_vecs)

                all_sent_vecs.append(sent_vecs.to(all_input_ids.device))
                all_logits.append(logits.to(all_input_ids.device))
            all_sent_vecs = torch.cat(all_sent_vecs, dim=0).view(all_input_ids.size(0), all_input_ids.size(1), -1)
            all_logits = torch.cat(all_logits, dim=0).view(all_input_ids.size(0), all_input_ids.size(1), -1)
            with open(cache_path, 'wb') as f:
                pickle.dump([all_sent_vecs, all_logits], f)
        return all_input_ids, all_input_mask, all_segment_ids, all_output_mask, all_sent_vecs, all_logits, all_label
    return all_input_ids, all_input_mask, all_segment_ids, all_output_mask, all_label

try:
    tokenizer_class = {'bert': BertTokenizer, 'xlnet': XLNetTokenizer, 'roberta': RobertaTokenizer, 'albert': AlbertTokenizer}.get(
except:
    tokenizer_class = {'bert': BertTokenizer, 'xlnet': XLNetTokenizer, 'roberta': RobertaTokenizer}.get(model_type)
tokenizer = tokenizer_class.from_pretrained(model_name)
examples,examples1 = read_examples(statement_jsonl_path,'./data/csqa/statement/train_modified.statement.jsonl')
# format of 'add_qa_prefix' or 'fairseq'
# for example in examples:
    # example.contexts = ['Q: ' + c for c in example.contexts]
    # example.endings = ['A: ' + e for e in example.endings]
    # # example.contexts = ['question is ' + c for c in example.contexts]
    # example.endings = ['answer is ' + e + '.' for e in example.endings]
features = convert_examples_to_features(examples,examples1, list(range(len(examples[0].endings))), max_seq_length, tokenizer,
                                        cls_token_at_end=bool(model_type in ['xlnet']),  # xlnet has a cls token at the end
                                        cls_token=tokenizer.cls_token,
                                        sep_token=tokenizer.sep_token,
                                        sep_token_extra=bool(model_type in ['roberta', 'albert']),
                                        cls_token_segment_id=2 if model_type in ['xlnet'] else 0,
                                        pad_on_left=bool(model_type in ['xlnet']),  # pad on the left for xlnet
                                        pad_token_segment_id=4 if model_type in ['xlnet'] else 0,
                                        sequence_b_segment_id=0 if model_type in ['roberta', 'albert'] else 1,sequence_c_segment_id
example_ids = [f.example_id for f in features]
*data_tensors, all_label = convert_features_to_tensors(features)
return (example_ids, all_label, *data_tensors)
```

```
def load_input_tensors(input_jsonl_path, model_type, model_name, max_seq_length, pretrained):
    if model_type in ('lstm',):
        raise NotImplementedError
    elif model_type in ('gpt',):
        return load_gpt_input_tensors(input_jsonl_path, max_seq_length)
    elif model_type in ('bert', 'xlnet', 'roberta', 'albert'):
        return load_bert_xlnet_roberta_input_tensors(input_jsonl_path, model_type, model_name, max_seq_length, pretrained)


def load_info(statement_path: str):
    n = sum(1 for _ in open(statement_path, "r"))
    num_choice = None
    with open(statement_path, "r", encoding="utf-8") as fin:
        ids = []
        labels = []
        for line in fin:
            input_json = json.loads(line)
            labels.append(ord(input_json.get("answerKey", "A")) - ord("A"))
            ids.append(input_json['id'])
            if num_choice is None:
                num_choice = len(input_json["question"]["choices"])
        labels = torch.tensor(labels, dtype=torch.long)

    return ids, labels, num_choice


def load_statement_dict(statement_path):
    all_dict = {}
    with open(statement_path, 'r', encoding='utf-8') as fin:
        for line in fin:
            instance_dict = json.loads(line)
            qid = instance_dict['id']
            all_dict[qid] = {
                'question': instance_dict['question']['stem'],
                'answers': [dic['text'] for dic in instance_dict['question']['choices']]
            }
    return all_dict
```

gsc.py에서 dataset 호출하는 부분이다.

```
dataset = LM_QAGSC_DataLoader(args, args.train_statements, args.train_adj,
                                            args.dev_statements, args.dev_adj,
                                            args.test_statements, args.test_adj,
                                            batch_size=args.batch_size, eval_batch_size=args.eval_batch_size,
                                            device=(device0, device1),
                                            model_name=args.encoder,
                                            max_node_num=args.max_node_num, max_seq_length=args.max_seq_len,
                                            is_inhouse=args.inhouse, inhouse_train_qids_path=args.inhouse_train_qids,
                                            subsample=args.subsample, use_cache=args.use_cache)
```

여기서 args.train_cycle_statements, args.dev_cycle_statements를 주어야함

modeling.py에서

LM_QAGSC_DataLoader 부분을보면

```
self.train_qids, self.train_labels, *self.train_encoder_data = load_input_tensors(train_statement_path, model_type, model_name, max_se
        self.dev_qids, self.dev_labels, *self.dev_encoder_data = load_input_tensors(dev_statement_path, model_type, model_name, max_se
```

load_input_tensors의 첫번째 인자가 train_statement_path, dev_statement_path

추가로 train_cycle_statements, dev_cycle_statements 넣기

그러면 sh파일에서도 수정해야함

```
export CUDA_VISIBLE_DEVICES=1,2
dt=`date '+%Y%m%d_%H%M%S'`

dataset="csqa"
```

```
model='roberta-large'
shift
shift
args=$@

elr="1e-5"
dlr="1e-2"
weight_decay="1e-3"

n_epochs=30
bs=128
# mbs=16
# ebs=32
mbs=4
ebs=8


k=2 #num of gnn layers
enc_dim=32

echo "***** hyperparameters *****"
echo "dataset: $dataset"
echo "enc_name: $model"
echo "batch_size: $bs"
echo "learning_rate: elr $elr dlr $dlr"
echo "edge_encoder_dim $enc_dim gsc_layer $k"
echo "******************************"

save_dir_pref='saved_models'
logs_dir_pref='logs/gsc'
mkdir -p $save_dir_pref
mkdir -p $logs_dir_pref

###### Training ######
for seed in 0; do
  echo "Training with seed: $seed"
  python3 -u gsc1.py --dataset $dataset \
      --encoder $model -k $k --enc_dim $enc_dim  \
      -elr $elr -dlr $dlr -bs $bs -mbs ${mbs} -ebs ${ebs} --weight_decay ${weight_decay} --seed $seed \
      --n_epochs $n_epochs --max_epochs_before_stop 10  \
      --train_adj data/${dataset}/graph/train.graph.adj.pk \
      --dev_adj   data/${dataset}/graph/dev.graph.adj.pk \
      --test_adj  data/${dataset}/graph/test.graph.adj.pk \
      --train_statements  data/${dataset}/statement/train.statement.jsonl \
      --dev_statements  data/${dataset}/statement/dev.statement.jsonl \
      --test_statements  data/${dataset}/statement/test.statement.jsonl \
      --max_seq_len 88     \
      --num_relation 38    \
      --inhouse True \
      --unfreeze_epoch 4 \
      --log_interval 20 \
      --save_model \
      --save_dir ${save_dir_pref}/${dataset}/enc-${model}__k${k}_encdim${enc_dim}_bs${bs}__seed${seed}_${dt} $args \
  | tee -a $logs_dir_pref/train_dev_${dataset}_enc-${model}_k${k}_encdim${enc_dim}_bs${bs}_seed${seed}_${dt}.log.txt
done
```

sh파일에서 받는 인자가 gsc.py의 dataset에 들어가고, dataset에 맞는 클래스(LM_QAGSC_dataloader)은 modeling.py에 있고 이 값은 load_input_tensor 함수에 들어감

이것은 Load_input_tensor

```
def load_input_tensors(input_jsonl_path, model_type, model_name, max_seq_length, pretrained):
    if model_type in ('lstm',):
        raise NotImplementedError
    elif model_type in ('gpt',):
        return load_gpt_input_tensors(input_jsonl_path, max_seq_length)
    elif model_type in ('bert', 'xlnet', 'roberta', 'albert'):
        return load_bert_xlnet_roberta_input_tensors(input_jsonl_path, model_type, model_name, max_seq_length, pretrained)
```

우리는 load_bert_xlnet_roberta_input_tensors에 input_cycle_jsonl_path를 넣는다.

아예 context score에 추가하는 cycle passage를 위해

run_idgsc__csqa_cycle.sh, gsc1_cycle.py, modeling_model_cycle.py, data_utils_cycle.py 에서 진행하자.

테스트 데이터에 대해서 context score에 추가할 cycle_passage를 만들경우는 in-house가 아닌 경우에 필요함

노드 개수가 늘어날수록 DPR이 너무 오래걸려 64개부터 진행 ㄴㄴ. 그 이유는 이렇게 둘러대면 되지 않을까?

1. 노드 개수가 늘어날수록 Other node가 증가한다. 증가하면 물론 보조적인 도움을 줘서 영향을 줄 수도 있지만 그렇지 않을수도 있다. 그래서 BM25만으로 먼저 많이 중복되는 단어를 거르는것만으로도 효과가 있을것이다. 그 후에 reranking을 진행하는게 time-saving이다.