RAG

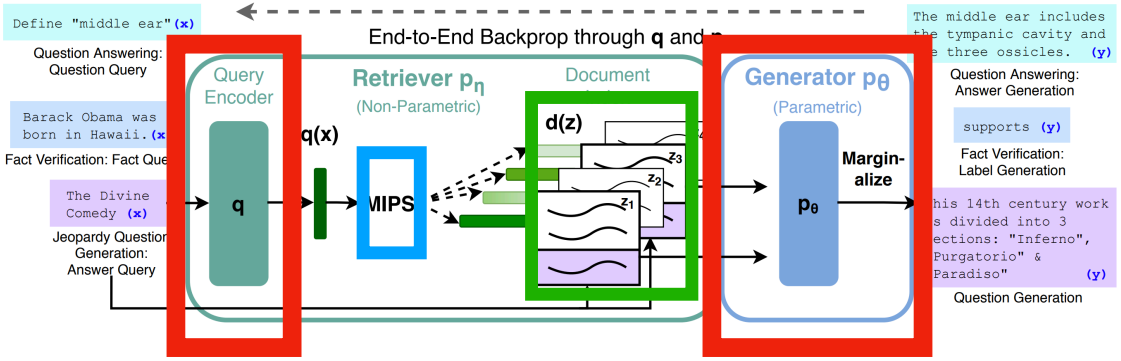
Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

†Facebook AI Research; ‡University College London; *New York University;

자여너학땅 BaekTree

핵심포인트

* 아주 간략히 말하는 중...



- Generator으로 주어진 query에 대해 prediction 수행
  - 이거 돼요? generation 된 sentence와 ground truth answer의 글자 단위로 loss 구성
- 학습할 때, question encoder와 seq2seq 모델을 동시에 학습(**그림의 빨간 네모만 보세요**)
- FIASS 필요(**파랑 네모**)
  - DPR의 dense embedding과 query의 embedding의 유사도
  - FIASS의 MIPS(Maximum Inner Product Sum)을 사용
- Passage endoer은 별개로 학습해요
  - DPR 논문의 결과로 나오는 dense vector index가 필요해요.(**초록 네모**)
- seq2seq으로 SKT에서 만든 KoBART을 써보면 어떨까요?
  - 메모리가 V100 1개로 버틸 수 있는지는 모르겠어요.
- Inference할 때, 가장 높은 확률이 나온 sentence을 선택!

무서운 부분

- 우리의 retrieval 데이터
  - Retrieval 과정에서 사용하는 문서 집합(corpus)은 `./data/wikipedia_documents.json` 으로 저장되어있습니다. 약 5만 7천개의 unique 한 문서로 이루어져 있습니다.
- RAG 성능

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

	Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- /50.1	37.4	-
	T5-11B+SSM[52]	36.6	- /60.5	44.7	-
Open Book	REALM [20]	40.4	- / -	40.7	46.8
	DPR [26]	41.5	**57.9**/ -	41.1	50.6
	RAG-Token	44.1	55.2/66.1	**45.5**	50.0
	RAG-Seq.	**44.5**	56.8/**68.0**	45.2	**52.2**

- 각 데이터 셋 설명

Natural Questions (NQ) (Kwiatkowski et al., 2019) was designed for end-to-end question answering. The questions were mined from real Google search queries and the answers were spans in Wikipedia articles identified by annotators.
TriviaQA (Joshi et al., 2017) contains a set of trivia questions with answers that were originally scraped from the Web.
WebQuestions (WQ) (Berant et al., 2013) consists of questions selected using Google Suggest API, where the answers are entities in Freebase.
CuratedTREC (TREC) (Baudiš and Šedivỳ, 2015) sources questions from TREC QA tracks

- 결론: TQA의 wiki 비중이 얼마나 되는지…

모델 불러오기

- Model = RagTokenForGeneration.from_pretrained(…)
- pretrained의 argument으로 DPR retrieval, seq2seq model, question_encoder 넣을 수 있음.
- DPR을 직접 만들고!
  - 인코더로 klue/roberta 쓰고!
- seq2seq은 KoBART 넣고!
- question_encoder은 klue/roberta 넣고!
- ㄱㄱㄱ

그래서 우리는?

- DPR
  - 구현: 3가지 방법
  - BM25 구현
  - 각종 하이퍼 파라미터
  - 리트리버 validation

Formally speaking, a retriever $R : (q, \mathcal{C}) \rightarrow \mathcal{C}_{\mathcal{F}}$ is a function that takes as input a question q and a corpus $\mathcal{C}$ and returns a much smaller *filter set* of texts $\mathcal{C}_{\mathcal{F}} \subset \mathcal{C}$, where $|\mathcal{C}_{\mathcal{F}}| = k \ll |\mathcal{C}|$. For a fixed k, a *retriever* can be evaluated in isolation on *top-k retrieval accuracy*, which is the fraction of questions for which $\mathcal{C}_{\mathcal{F}}$ contains a span that answers the question.

- RAG
  - DPR 그대로 사용
  - Token 방식, sentence 방식
  - 하이퍼 파라미터

여기서부터 논문 설명이에요...

사용하는 Pretrained model

- Passage Encoder
  - DPR
    - In batch gold passage + one BM25 negative
    - Optimize metric loss: cosine similarity
- Query Encoder(gonna fine tune)
  - BERT
- Generator(gonna fine tune)
  - BART(Or any seq2seq model = encoder-decoder model)

학습 과정

*** 또 아주 간략히 말하는 중...**

- query을 BERT에 넣고 encoding
- query_enc와 DPR의 passage_enc의 유사도로 top k을 구한다
- Top k에서 나온 passage들로 generation 수행 -> seq2seq 모델에 넣는다
- Input: question + SEP(아마도?) + passage 1개가 하나의 input
  - k개의 input이 들어가고, (beam search의 결과로) k개의 output(토큰들)이 나와야 해요.
- output이 나오면 ground truth와 글자 글자를 비교해서 loss을 만들어요.
- 그리고 backward해서 query encoder와 seq2seq의 가중치를 fine tuning해요.

제대로 말하기1

동일한 결과에 대한 marginalize 해야 해요.

- query 1개 들어감 -> k개의 retrieval 나옴 -> 각 k개와 query 넣어서 k개의 generation 만듦
- 학습이 좀 되면서 k개의 output 중에서 동일한 prediction들이 나올 수 있음.
  - 그러면 passage들에 대해서 marginalize 해야 함.
  - 답만 맞으면 되지, 다른 passage 라는게 뭔상관?

제대로 말하기2

사실 모델 종류가 두개ㅋㅋ

- RAG-Sequence Model

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x,z,y_{1:i-1})$$

  - 1개 passage -> 1개 output. 동일한 answer에 대해서 passage에 대해 marginalize

- RAG-Token Model

$$p_{\text{RAG-Token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x,z,y_{1:i-1})$$

  - k개 passage에서 토큰 하나씩. 동일한 token이면 marginalize. N개의 토큰 생성

2개 모델의 성능이 다름

- 데이터 셋에 따라 성능이 달라서 두개 다 해봐야 한다.
- 다행히 huggingface에 다 있음 ^^

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

	Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- /50.1	37.4	-
	T5-11B+SSM[52]	36.6	- /60.5	44.7	-
Open Book	REALM [20]	40.4	- / -	40.7	46.8
	DPR [26]	41.5	**57.9**/ -	41.1	50.6
	RAG-Token	44.1	55.2/66.1	**45.5**	50.0
	RAG-Seq.	**44.5**	56.8/**68.0**	45.2	**52.2**

코-오-드-으

- RAG_model = RagTokenForGeneration.from_pretrained(…)
- Query = tokenizer(data) # data has question and answer
- Output = RAG_model(query[input_ids],query[label])
- output.loss.backward()
- 끗.

```
def forward(
    self,
    input_ids=None,
    attention_mask=None,
    encoder_outputs=None,
    decoder_input_ids=None,
    decoder_attention_mask=None,
    past_key_values=None,
    context_input_ids=None,
    context_attention_mask=None,
    doc_scores=None,
    use_cache=None,
    output_attentions=None,
    output_hidden_states=None,
    output_retrieved=None,
    exclude_bos_score=None,
    reduce_loss=None,
    labels=None,
    n_docs=None,
    **kwargs  # needs kwargs for generati
):
```

코-오-드-으 내부 구조

- RagSequenceForGeneration(query, answer) -> RagModel(query, label)을 호출.
- RagModel 내부에서 query을 인코더에 넣어서 임베딩하고, 그 값을 리트리버에 넣는다. 그러면 유사도 높은 passage들 반환.

```
question_enc_outputs = self.question_encoder(
    input_ids, attention_mask=attention_mask, return_dict=True
)
question_encoder_last_hidden_state = question_enc_outputs[0]  # hidden states of question encoder

retriever_outputs = self.retriever(
    input_ids,
    question_encoder_last_hidden_state.cpu().detach().to(torch.float32).numpy(),
    prefix=self.generator.config.prefix,
    n_docs=n_docs,
    return_tensors="pt",
)
```

- context_input_ids: top k passages에서 query와 pasage을 cancat된 결과.

```
gen_outputs = self.generator(
    input_ids=context_input_ids,
    attention_mask=context_attention_mask,
    encoder_outputs=encoder_outputs,
    decoder_input_ids=decoder_input_ids,
    decoder_attention_mask=decoder_attention_mask,
    past_key_values=past_key_values,
    use_cache=use_cache,
    output_attentions=output_attentions,
    return_dict=True,
)
```

- decoder_input_ids: ground truth answer가 들어가서 teacher forcing

- 질문: 백재형은 몇살인가? Answer:19살
- Top k -> generator
  - 1: 백재형은 ~~19살이다 -> 답변
  - 2. 누구의 친구 백재형은 ~~~19살이다 -> 답변
  - 3. 송민재는 19살이다. -> 답변

  - 몇살인가?+passage -> 19살.

  - 3개 중에 가장 확률이 높은 답은? 1번이다!