In-Context Retrieval-Augmented Language Models (프로젝트 논문 후보 1)

[논문 링크]

https://arxiv.org/pdf/2302.00083.pdf

💡 **Model의 추가 training없이 retriever로 검색한 document를 LM에 바로 적용할 수 있다는 점**

- ground : AI가 답변을 더 잘할 수 있도록 정보를 제공하다.

Abstract

Retrieval-Augmneted Language Modeling (RALM)은 generation동안 grounding corpus로부터 관련 문서를 conditioning하는 방법이다.

이는 performance를 큰 폭으로 향상 시켰으며, 부정확한 text generation을 감소 시키고 natural source attribution mechanism을 제공한다.

기존 RALM 방법들은 LM의 architecture르르 수정하고자 했으므로 매우 복잡했지만 이 논문에서 제시한 방법은 **In-context RALM으로 LM architecture는 그대로 두고 grounding documents를 input으로 함께 제공하는 방식**이다.

즉, 어떠한 추가적인 training도 필요가 없다.

특별한 훈련 없이 적용할 수 있는 general purpose retrievers을 기반으로 만든 In-context RALM이 model size와 diversity 측면에서 큰 이득을 얻을 수 있었다.

또한 document retrieval과 ranking mechanism이 RALM setting에 특화 되도록 만들어서 pre-trained LM, 심지어 API에도 사용할 수 있을 정도로 prevalence를 증가 시킬 수 있었음을 확인했다.

---

1. Introduction

LM이 많은 발전을 거듭하고 있지만, 외부 정보에 대한 access에 한계가 있다는 inherent limitation은 아직 해결되지 않았다.

1. LM은 어떤 source attribution도 사용할 수 없으며, training 단계에서 보지 못한 정보에 대해서는 incorporate 해야 한다.
    a. 즉, 추가적인 training이 필요하다는 것
2. 더 중요한 것은, LM은 factual inaccuracies와 errors를 생산하는 경향이 있다.

이러한 문제들은 어느 LM에서나 존재하며, 특히 uncommon domain이나 private data에 대해서는 문제가 더 악화된다.

→ 이를 해결하기 위한 approach로 등장한 것이 **Retrieval-Augmented Language Modeling**이다.

- Generation 동안, external knowledge source로부터 관련 있는 문서들을 추출하여 LM에 정보를 제공하는 방식

RALM systems은 2가지 high-level components를 포함하고 있다.

- document selection
    - condition으로 넣을 set of documents를 선정
- document reading
    - 선정된 documents를 LM generation process에 어떻게 통합 시킬 것인가.

최근에는 LM의 architecture를 수정하고자 하는 시도들이 많았다.

- frozen BERT retriever로 document를 selection하면서 LM architecture에 추가적인 training을 진행하는 featuring document reading

- performance가 좋아지긴 했으나 이는 모델이 넓게 적용할 수 없다는 단점과 architecture를 수정해야 한다는 단점이 존재

따라서, 매우 단순한 document reading mechanism을 여기서 제시 → In-Context RALM

Figure 2: An example of *In-Context RALM*: we simply prepend the retrieved document before the input prefix.

- zero-effort document reading mechanism
  - 단순히 LM의 input text에 selected documents를 덧붙이는 것.
- off-the-shelf LM뿐만 아니라 API access로도 충분히 성능 향상이 가능

---

2. Related Work

RALM approach는 2개의 모델로 rough하게 나눌 수 있다.

본 연구에서는 2번째 모델들을 사용하지만, 추가적인 training이 필요 없다는 점에서는 차이점을 보인다.

- Nearest Neighbor Language Models
- Retrieve and Read Models

Nearest Neighbor Language Models

KNN-LM approach → simple inference-time model

- 2개의 next-token distribution 사이를 interpolate
  - 1개는 LM 자체에서 유도된 것
  - 1개는 retrieval corpus로부터 k개의 neighbors로부터 유도된 것

하지만 corpus 안에 있는 모든 token들을 저장해야 해서, small corpus에도 expensive requirement.

Retrieve and Read Models

*document selection*과 *document reading*이 명백하게 구분되어 있는 모델들을 의미.

이전 모든 연구들은 LM을 training하는 것까지 포함하고 있었다.

- 이전 연구들 중에서, RETRO가 이 paper와 가장 관련 있다.
  - auto-regressive LM을 chuncked cross-attention을 통해 관련 있는 document끼리 attention 수행 → parameter update

본 논문에서 제시한 In-Context RALM은 다음과 같은 2가지 key aspect가 있다.

- o**ff-the-shelf LM을 추가 training 없이 그대로 사용**
- LM performance 향상을 위해 **어떻게 document choose를 수행** 했는지에 초점을 맞춤.

---

3. Our Framework

3.1 In-Context RALM

LM은 token sequence의 probability density를 정의한다.

x_1,...,x_n이라는 sequence가 주어지면 일반적으로는 다음과 같이 probability를 모델링 한다.

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i | x_{<i}) \tag{1}$$

- (1)의 식은 auto-regressive model이며 causal self-attention mask (이 후에 있는 token들은 고려하지 않는 것)를 사용한 conditional probability로 나타난다.
- GPT-2, GPT-3 등과 같은 강력한 모델들은 위와 같은 방법을 사용한다.

Retrieval Augmented Language Models (RALMs)은 여기서 외부 corpus로부터 여러 개의 document를 retrieve하는 operation을 추가한다.

그리고 위에서 정의한 (1)의 식에 retrieve한 document를 함께 conditioning한다.

이 때, 기존의 방법들은 retrieval document를 처리하는 특별한 architecture를 사용해야 했으나, GPT-3에서 언급한 in-context learning의 성공으로 인해 **In-Context RALM은 단순히 transformer의 input에 retrieved documents를 concat하는 방식으로 사용**한다.

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i | \mathcal{R}_\mathcal{C}(x_{<i}); x_{<i}) \tag{2}$$

- $\mathcal{R}_\mathcal{C}(x_{<i})$: prefix에 따라 달라지는 external corpus $\mathcal{C}$로부터 retrieval operation
- transformer-based LM은 일반적으로 제한된 길이의 input sequence를 받는다.
  - 따라서, 만약에 document와 input sequence를 concat했을 때, 길이가 초과한다면 ?
    - **model이 받을 수 있는 길이가 될 때까지 x의 시작으로부터 token들을 제거하는 방식을 사용한다.**
- 우리가 retrieve한 document들은 제한된 길이의 passage이기 때문에 x로부터 충분한 context를 받는다. → $4.3 참고
  - dense retrieval code on the DPR repository를 기반으로 만듬
  - retrieval corpora는 중복되지 않는 100-150개의 단어의 passages들로 구성되어 있으며, 입력될 때는 256개의 tokens으로 자르는 과정을 거친다.

3.2 RALM Design Choices

2가지 practical design choice에 대한 설명

Retrieval Stride

retrieval stride = s

원래는 retrieval operation이 generation을 할 때마다 실행되지만, 이는 retriever를 calling하는 데에 cost가 발생하고 generation동안 LM prefix에서 documents를 대체해야 하는 과정이 필요하다.

따라서, 여기서 제안한 In-Context RALM에서는 모든 s(1보다 큰 양수)개의 tokens에서 단 한번의 retrieval만 수행하도록 만든다.

$$p(x_1, \ldots, x_n) = \prod_{j=0}^{n_s-1} \prod_{i=1}^{s} p_\theta(x_{s \cdot j+i} | [\mathcal{R}_\mathcal{C}(x_{\leq s \cdot j}); x_{\leq (s \cdot j+i)}]) \tag{3}$$

- 즉, stride를 정해서 여러 개의 tokens들을 바탕으로 retrieval을 진행한다는 뜻.

stride를 작게 설정해서 더 자주 retrieval을 진행할수록 성능은 더 좋아지지만, runtime이 오래 걸린다.

따라서, runtime과 performance의 trade-off를 적절하게 조절할 수 있는 retrieval stride를 설정해야 한다.

Retrieval Query Length

retrieval query는 보통 모든 prefix token $x_{\leq s \cdot j}$이다.

일반적으로 생각했을 때, 우리가 생성하고자 하는 token과 가장 관련 있는 token은 아마 retrieval query (prefix tokens)의 가장 마지막에 있는 token일 것이다.

만약에 retrieval query가 너무 길어지면, 마지막에 있는 token이 가장 중요함에도 이 영향이 제대로 전달되지 않을 수 있으므로, 이를 제한하기 위해 retrieval query length인 ℓ을 설정한다.

마지막 token을 기준으로 ℓ만큼의 token만 retrieval하는 데에 참고한다는 뜻이다.

$$p(x_1, \ldots, x_n) = \prod_{j=0}^{n_s-1} \prod_{i=1}^{s} p_\theta(x_{s \cdot j+i} | [\mathcal{R}_\mathcal{C}(q_j^{s \cdot \ell}); x_{\leq (s \cdot j+i)}]) \tag{4}$$

---

4. Experimental Details

4.1 Datasets

Language modeling

- WikiText-103

    - RALM을 평가하는 데에 가장 많이 사용되는 dataset
- The Pile
    - Arxiv
    - Stack Exchange
    - FreeLaw
- Real-News
    - The Pile이 news에 대한 corpus가 부족하기 때문에.

Open-Domain Question Answering

- Natural Questions (NQ)
- TriviaQA

4.2 Models

Language Models

- GPT-2의 4개 모델
- GPT-Neo와 GPT-J의 3개 모델
- OPT의 8개 모델
- LLaMa의 3개 모델

→ 모두 open source, available

Retrievers

- sparse (word-based) retrievers
    - BM25
- dense (neural) retrievers
    - frozen BERT-base
    - Contriever and Spider

Reranking

- RoBERTa-base로 시작하여 reranker를 학습

4.3 Implementation Details

transformers library를 사용해서 code base를 implement

DPR repository에서 dense retrieval code를 사용

Retrieval Corpora

- WikiText-103을 사용
  - Contamination을 피하기 위해서, WikiText-103 데이터 셋에 대한 development, test set의 내용들을 모두 삭제한 후, 나머지 데이터로 학습
- retrieval corpora는 중복되지 않은 100개의 단어로 된 passages들로 구성되어 있음.
  - 150개 tokens 이하로 translate 됨.
- **따라서, 256 tokens에서 retrieved passages를 잘라서 input으로 들어감**

Retrieval

- Sparse retrieval을 위해 Pyserini library를 사용
- Dense retrieval을 위해 FAISS를 사용한 exact search를 적용

---

5. The Effectiveness of In-Context RALM with Off-the-Shelf Retrievers

간단한 document reading mechanism에도 불구하고 In-Context RALM이 다양한 evaluation에 대해서 얼마나 많은 이득을 얻는 지를 확인할 수 있다.

이 section에서는 In-Context RALM을 위해 off-the-shelf retrievers의 효과를 살펴본다.

여기서 진행한 experiments는 In-Context RALM을 적용할 때, **recommended configuration**에 대한 결과를 제공해준다.

- Sparse BM25 retriever
  - $\ell = 32$ query token을 받는 retriever이며, 가능한 자주 사용된다.
- 매번 s=4 tokens을 retrieve

Model	Retrieval	Reranking	WikiText-103	RealNews	ArXiv	Stack Exch.	FreeLaw
			word ppl	token ppl	token ppl	token ppl	token ppl
GPT-2 S	–	–	37.5	21.3	12.0	12.8	13.0
	BM25 §5	–	29.6	16.1	10.9	11.3	9.6
	BM25	Zero-shot §6.1	28.6	15.5	10.1	10.6	8.8
	BM25	Predictive §6.2	26.8	–	–	–	–
GPT-2 M	–	–	26.3	15.7	9.3	8.8	9.6
	BM25 §5	–	21.5	12.4	8.6	8.1	7.4
	BM25	Zero-shot §6.1	20.8	12.0	8.0	7.7	6.9
	BM25	Predictive §6.2	19.7	–	–	–	–
GPT-2 L	–	–	22.0	13.6	8.4	8.5	8.7
	BM25 §5	–	18.1	10.9	7.8	7.8	6.8
	BM25	Zero-shot §6.1	17.6	10.6	7.3	7.4	6.4
	BM25	Predictive §6.2	16.6	–	–	–	–
GPT-2 XL	–	–	20.0	12.4	7.8	8.0	8.0
	BM25 §5	–	16.6	10.1	7.2	7.4	6.4
	BM25	Zero-shot §6.1	16.1	9.8	6.8	7.1	6.0
	BM25	Predictive §6.2	15.4	–	–	–	–

Table 1: Perplexity on the test set of WikiText-103, RealNews and three datasets from the Pile. For each LM, we report: (a) its performance without retrieval, (b) its performance when fed the top-scored passage by BM25 (§5), and (c) its performance when applied on the top-scored passage of each of our two suggested rerankers (§6). All models share the same vocabulary, thus token-level perplexity (*token ppl*) numbers are comparable. For WikiText we follow prior work and report word-level perplexity (*word ppl*).

Perplexity : '헷갈리는 정도'를 의미함. 낮을 수록 더 좋다.

- BM25 retriever를 사용했을 때, 2-3배 더 큰 GPT-2 모델을 그냥 사용하는 수준의 perplexity와 일치하는 모습을 보이면서 상당한 수준의 improvement를 얻을 수 있음을 확인할 수 있다.

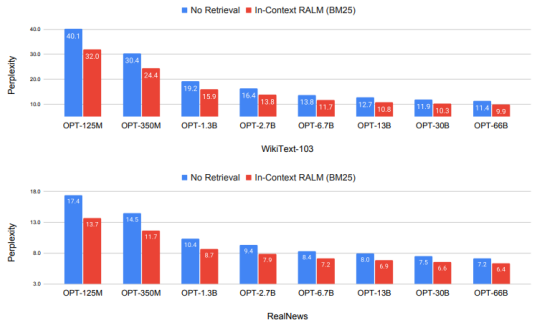

Figure 4: Results of OPT models (Zhang et al., 2022) on the test set of WikiText-103 (word-level perplexity) and the development set of RealNews (token-level perplexity). In-Context RALM models use a BM25 retriever with $s = 4$ (*i.e.*, the retriever is called every four tokens) and $\ell = 32$ (*i.e.*, the retriever query is comprised of the last 32 tokens of the prefix). *In-Context RALM with an off-the-shelf retriever improved the performance of a 6.7B parameter OPT model to match that of a 66B parameter OPT model.*

Model	Retrieval	WikiText-103 word ppl
LLaMA-7B	-	9.9
	BM25, §5	8.8
LLaMA-13B	-	8.5
	BM25, §5	7.6
LLaMA-33B	-	6.3
	BM25, §5	6.1

Table 2: The performance of models from the LLaMA family, measured by word-level perplexity on the test set of WikiText-103.

- LLaMA에서도 BM25를 사용할 때마다 ppl이 낮아지는 모습을 확인할 수 있고, figure 4를 통해 모델의 parameter가 66B이나 증가 될 때까지 retriever를 적용할 때마다 ppl이 낮아지는 흐름을 확인할 수 있다.

5.1 BM25 Outperforms Off-the-Shelf Neural Retrievers in Language Modeling

여러 개의 off-the-shelf general purpose retrievers를 사용해서 실험을 해보았으나 **sparse BM25 retriever가 유명한 3개의 dense retriever보다 성능이 좋았음을 확인**했다.

- Contriever
- Spider
- RETRO system에 사용되는 BERT embedding의 average pooling을 기반으로 하는 retriever

또한, $\ell = 32$일 때, BM25에서 가장 optimal하다는 것을 확인했으며 dense retriever의 경우 $\ell = 64$ 일 때 가장 좋은 결과를 얻을 수 있었다.

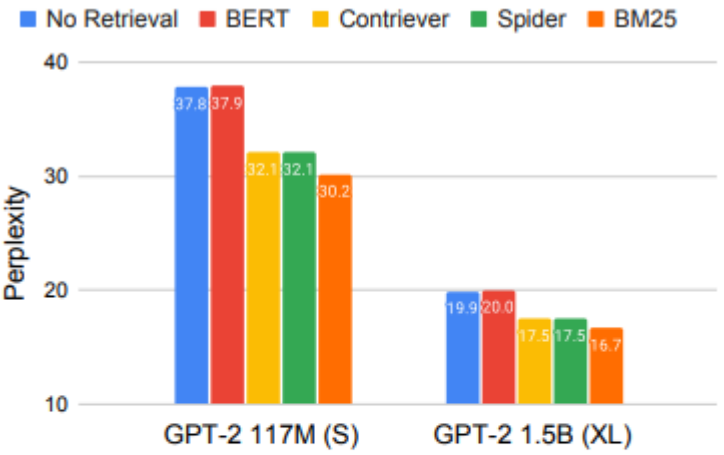

Figure 3: The performance of four off-the-shelf retrievers used for In-Context RALM on the development set of WikiText-103. All RALMs are run with $s = 4$ (*i.e.*, retrieval is applied every four tokens). For each RALM, we report the result of the best query length ℓ (see Figures 6, 9, 10).

- 3개의 dense retriever와 sparse retriever인 BM25의 성능을 비교해본 figure
- BM25를 사용했을 때, 가장 ppl이 낮은 것을 확인할 수 있다.
  - 이는 이전 연구에서 BM25가 많은 task에서 사용되는 거의 모든 neural retrievers보다 훨씬 성능이 좋았다는 결과에 대한 consistency에 대한 증거
- 이 결과는 In-Context RALM이 BM25를 사용하는 것이 neural retriever를 사용하는 것보다 훨씬 cheaper하고 성능이 좋기 때문에 훨씬 유용하다는 것을 암시.

5.2 Frequent Retrieval Improves Language Modeling

앞서 말한 retrieval stride s를 다르게 했을 때의 effect를 탐구해보았다.

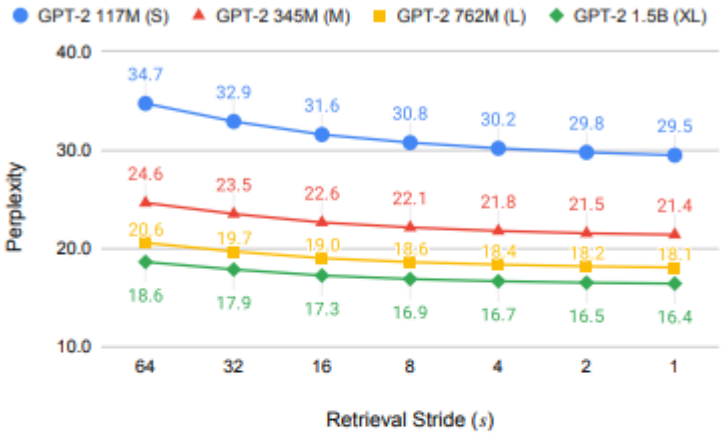

Figure 5: An analysis of perplexity as a function of s, the *retrieval stride*, *i.e.*, the number of tokens between consecutive retrieval operations, on the development set of WikiText-103. Throughout the paper, we use $s = 4$ to balance perplexity and runtime.

- s를 작게 한다 → retrieval을 더 자주 한다.
- s가 작을수록 성능이 더 향상되는 것을 확인할 수 있지만, 이에 따라 runtime cost는 증가하기 때문에 ppl과 runtime의 trade-off를 적절하게 조절 → s=4로 설정.
- 더 작은 모델일수록 s에 따른 성능 향상의 폭이 커진다.

5.3 A Contextualization vs. Recency Tradeoff in Query Length

ℓ의 값을 변화하면서 effect를 확인.

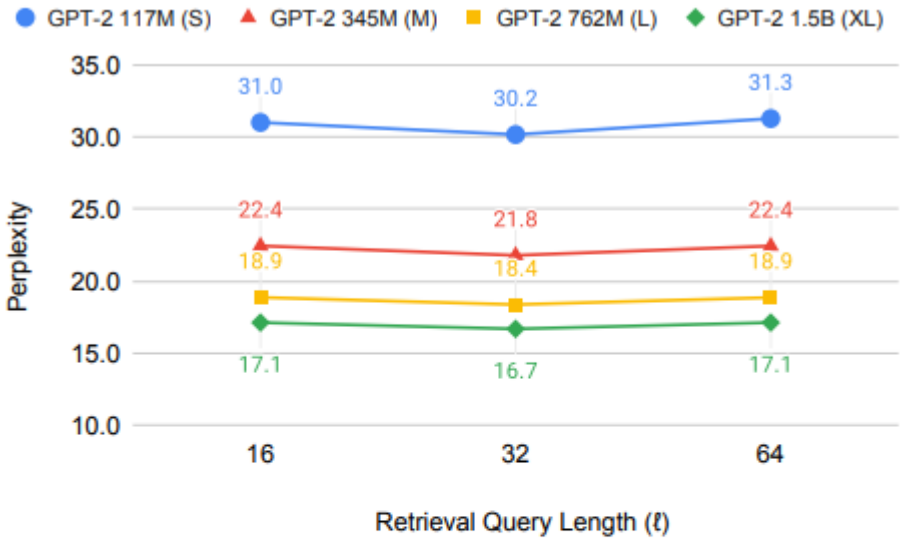

Figure 6: An analysis of perplexity as a function of *the number of tokens in the query* ℓ for BM25 on the development set of WikiText-103. In the appendix, we show similar trade-offs for dense retrievers within WikiText-103. Throughout the paper, we use a query length of $\ell = 32$ tokens.

- 32 token일 때, 가장 괜찮은 성능을 보인다.
  - 16 token이면, 너무 짧은 context만 고려해서 덜 정확한 것 같고, 64 token이면 너무 많은 context를 고려해서 쓸모 없는 정보까지 추가되어 그런 것 같다.

---

6. Improving In-Context RALM with LM-Oriented Reranking

'In-Context RALM이 고정된 document reading component를 사용하기 때문에, LM task에 맞게 document retrieval mechanism을 조금 더 특화하면 성능이 나아지지 않을까?' 라는 질문은 당연한 것.

→ Yes, improvement에 대한 considerable scope가 존재.

- 이전 section에서는 '가장 먼저 retrieval된 document에 대해서 conditioning을 어떻게 할 것인가?' 에 대한 내용을 다뤘는데 이는 **query에 대해서 한정된 semantic**

understanding을 일으킬 수도 있다.

- BM25는 오직 bag of words signal만을 기반으로 하기 때문.
- 또한, 나중에 오는 token일수록 generated token과 더 관계가 있다는 중요성을 반영할 방법이 없다.

따라서, 이 section에서 어떤 document를 선택하여 model에 제공할 것인가에 대해 초점을 맞춘다.

How? ⇒ BM25 retriever로부터 top-*k* documents를 re-ranking

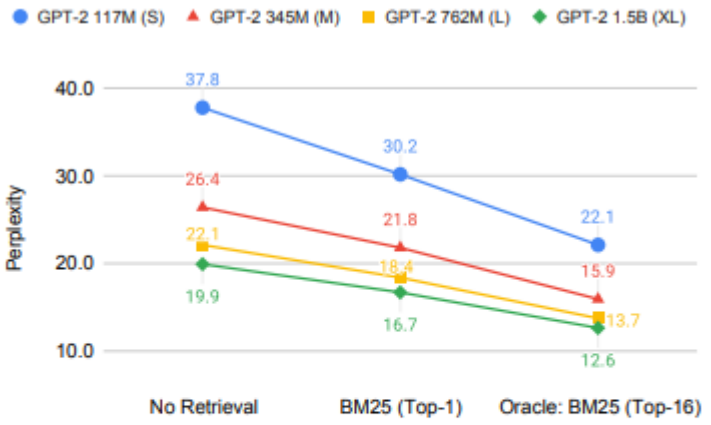

Figure 7: *Potential for gains* from reranking: perplexity improvement (on the development set of WikiText-103) from an oracle that takes the best of the top-16 documents retrieved by BM25 rather than the first.

- top-1개만을 retrieval할 때보다, top-16개를 retrieval할 때 더 성능이 증가하는 것을 확인
- Figure 7을 motivation으로 사용

결론 : 2개의 reranker를 사용 → zero-shot rerankers / predictive rerankers

6.1 LMs as Zero-Shot Rerankers

off-the-shelf language model을 document reranker로 사용했다.

마지막 ℓ개의 token으로 이루어진 Query q에 대해 BM25가 retrieval한 top-k documents를 $\{d_1, \ldots, d_k\}$이라고 하자.

j번의 retrieval iteration동안, generation을 위한 text를 다음과 같이 정의한다.

여기서 y는 생성을 위한 텍스트, 즉 우리가 생성을 해야 하는 target text가 된다.

$$y := x_{s \cdot j+1}, \ldots, x_{s \cdot j+s}$$

여기서 우리는 generation을 위한 text의 probability를 최대로 만들어주는 ideal한 document, d_{i*}를 찾는 것이 목적이며 이는 (5)의 식으로 구할 수 있다.

$$i* = arg\ max_{i \in [k]}\ p_\theta(y | [d_i; x_{\leq s \cdot j}]) \tag{5}$$

하지만 test time에서 우리는 y에 접근할 수가 없다. 대신 우리는 test time에서 마지막 prefix tokens들을 사용할 수 있는데, 이를 y′ 이라고 한다. → y′을 reranking에 사용.

즉, y′는 현재까지 주어진 입력, 혹은 현재까지 생성된 text를 의미한다.

$$y\prime := x_{s \cdot j - s\prime+1}, \ldots, x_{s \cdot j}$$

- 여기서 s′는 y 앞에 위치하는 stride를 의미한다.

이 y′을 바탕으로, document인 d_i를 선택한다.

$$\hat{i} = arg\ max_{i \in [k]}\ p_\phi(y\prime | [d_i; x_{s \cdot j - s\prime}]) \tag{6}$$

BM25가 lexical retriever (어휘 기반 검색) 이기 때문에, 우리가 LM에 의해 유도된 semantic signal을 포함하기 원한다는 것이 주된 motivation이다.

또한, 이러한 reranking은 open-domain에서의 QA를 위한 reranking framework와 conceptual similarity를 공유하는데 이는 y′이 ‘question’으로 간주될 수 있다는 것을 의미한다.

우리가 제시한 Zero-shot reranking은 generation하는 LM과 reranking하는 LM이 반드시 같을 필요가 없다는 것을 암시하고 있다. → (5)와 (6)의 parameter가 다름.

이 말은, reranking을 위해서는 더 작은 LM을 사용할 수도 있다는 것이고, 이는 2가지의 중요한 이유를 포함한다.

- k개의 documents를 reranking하려면 k번의 forward pass가 필요하다.
- 실제 LM의 log probability를 사용할 수 없는 상황에서도 사용이 가능하다.

- 예를 들어, 오로지 API를 통해서만 LM에 접근할 수 있을 때.

Result

- s′ = 16일 때, 그리고 top-16 documents를 retrieval했을 때 가장 성능이 좋았다.

Model	Reranking Model	WikiText-103	RealNews
		word ppl	token ppl
GPT-2 345M (M)	GPT-2 110M (S)	20.8	12.1
	GPT-2 345M (M)	20.8	12.0
GPT-2 762M (L)	GPT-2 110M (S)	17.7	10.7
	GPT-2 762M (L)	17.6	10.6
GPT-2 1.5B (XL)	GPT-2 110M (S)	16.2	9.9
	GPT-2 1.5B (XL)	16.1	9.8

Table 3: Perplexity for zero-shot reranking (§6.1) where the reranking models is smaller than the LM, or the LM itself. Reranking is performed on the top 16 documents retrieved by BM25. Using a GPT-2 110M (S) instead of a larger language model as a reranker leads to only a minor degradation.

- Reranking을 위해 small model인 GPT-2 100M을 사용해도 ppl의 차이가 크게 없기 때문에, small model을 사용해도 괜찮다.

6.2 Training LM-dedicated Rerankers

다음으로, 우리가 직접 BM25에서 추출한 top-k documents 중에 하나를 고를 수 있는 reranker를학습 시키는 것이다.

이 방법을 ***predictive reranking***이라고 부르는데, 이는 reranker가 어떤 document가 다음 text를 '예측'하는 데에 있어서 더 도움이 될 지를 학습하기 때문이다.

Reranker를 classifier로 학습

- prefix $x_{\leq s \cdot j}$와 document d_i를 입력으로 받고 scalar $f(x_{\leq s \cdot j}, d_i)$를 생성
- $f(x_{\leq s \cdot j}, d_i)$는 d_i가 $x_{\leq s \cdot j}$의 continuation에 얼마나 관련성이 있는 지를 수치화 → 일종의 score라고 보면 될 것 같다.

그리고 relevance scores를 normalize

$$p_{\text{rank}}(d_i|x_{\leq s \cdot j}) = \frac{\exp(f(x_{\leq s \cdot j}, d_i))}{\sum_{i'=1}^{k} \exp(f(x_{\leq s \cdot j}, d_{i'}))}, \quad (7)$$

그리고 (8)의 식으로 document, d_i를 선택한다.

$$\hat{i} = arg\ max_{i \in [k]}\ p_{rank}(d_i | x_{\leq s \cdot j}) \tag{8}$$

Collecting Training Examples + Training

predictive reranker를 훈련하기 위해 다음과 같은 training example을 모아야 한다.

1. $x_{\leq s \cdot j}$를 훈련 데이터로부터 우리가 샘플링한 prefix라고 하자. $y := x_{s \cdot j+1}, \ldots, x_{s \cdot j+s}$는 다음 stride에 올 generation text라고 하자.
2. BM25를 $x_{\leq s \cdot j}$로부터 query $q_j^{s,\ell}$를 얻고 k개의 document를 얻는다.
3. 각 document d_i에 대해서 LM을 이용하여 $p_\theta(y|[d_i; x_{\leq s \cdot j}])$을 계산한다.

$$-log \sum_{i=1}^{k} p_{rank}(d_i | x_{\leq s \cdot j})\ \cdot\ p_\theta(y|[d_i; x_{\leq s \cdot j}]) \tag{9}$$

4. (9)의 식을 가지고 전체 loss function을 정의하여 training한다.

이 때, training은 RoBERTa-base로 fine-tuning하는 방식으로 진행한다.

Result

이와 같은 방법에 대한 performance는 table 1을 참고하면 되며, domain-specific data에서 reranker를 사용할 때는 이렇게 훈련 시킨 reranker가 더 효과적인 것을 확인했다.

---

7. In-Context RALM for Open-Domain Question Answering

지금까지는 framework에 대한 evaluation을 language modeling benchmark로 했으나, additional scenario 혹은 특별한 downstream task에 대해 framework의 effectiveness를 측정하기 위해 open-domain QA를 진행한다.

이 실험들은 control 가능한 환경에서 LM이 추가적인 training 없이 retrieved documents를 활용 가능한지 확인할 수 있다.

LLaMA를 통해 In-Context RALM을 추가 / 추가하지 않은 상태로 ODQA를 진행

- literature을 open-book / closed book 형태로 각각 사용

  ▼ prompt setting (Appendix)

Closed-Book Setting For the closed-book setting, we adopt the prompt of Touvron et al. (2023):

```
Answer these questions:
Q: Who got the first nobel
prize in physics?
A:
```

Open-Book Setting For the open-book setting, we extend the above prompt as follows:

```
Nobel Prize

A group including 42
Swedish writers, artists,
and literary critics
protested against this
decision, having expected
Leo Tolstoy to be awarded.
Some, including Burton
Feldman, have criticised
this prize because they...

Nobel Prize in Physiology
or Medicine

In the last half century
there has been an
increasing tendency
for scientists to work
as teams, resulting in
controversial exclusions.
Alfred Nobel was born
on 21 October 1833 in
Stockholm, Sweden, into
a family of engineers...

Based on these texts,
answer these questions:
Q: Who got the first nobel
prize in physics?
A:
```

DPR을 retriever로 사용했다.

Varying the Number of Documents

model에 몇 개의 document를 보여주느냐에 따른 effectiveness를 측정하기 위해, NQ와 TriviaQA의 set에서 minimal analysis를 수행

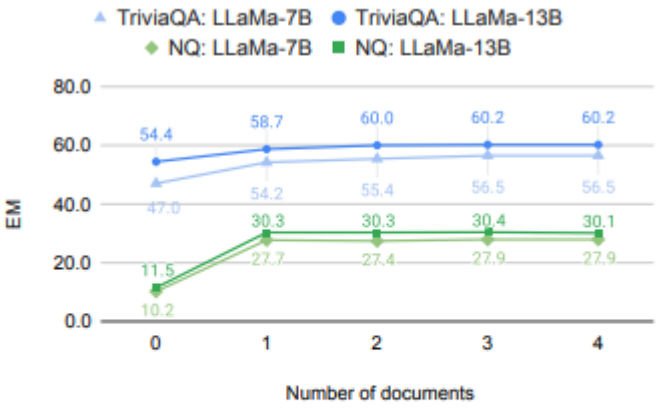

Figure 8: Zero-shot performance of In-Context RALM on the development set of Natural Questions and TriviaQA, when varying the number of documents (retrieved by DPR) shown in-context.

- 1개를 사용하면 확실하게 성능이 올라감
- 1개 이상을 사용하는 것에 대해서는 performance에 대해 큰 차이가 없음

Results

Model	Retrieval	NQ	TriviaQA
LLaMA-7B	-	10.3	47.5
	DPR	28.0	56.0
LLaMA-13B	-	12.0	54.8
	DPR	31.0	60.1
LLaMA-33B	-	13.7	58.3
	DPR	32.3	62.7

Table 4: Zero-shot results of In-Context RALM on the test set of Natural Questions and TriviaQA measured by exact match. In the open-book setting, we include the top two documents returned by DPR.

accuracy

- 2개의 retrieved documents를 사용
- 성능이 확실하게 향상된 모습을 확인할 수 있음.

---

8. Discussion

- external source로부터 retrieval하는 것은 knowledge-intensive task에 일반적인 관행이 되었다.
    - 동시에, LM generation capabilities에 대한 최근 비약적인 발전은 유용한 긴 text를 생성하는 것이다.
    - 하지만 facutal inaccuracy는 여전히 문제가 되며, 생성된 text에 직접적인 출처가 없다는 점은 신뢰도를 떨어뜨린다.
- 따라서 RALM의 접근 방식이 동기 부여되었고, 해당 연구 분야가 필요하다는 이유이다.
- 하지만, 널리 배포되기 힘들었는데 이는 LLM이 API를 통해서만 접근 가능한 경우들이 많기 때문이다.
- 따라서 API를 통해서만 접근 가능한 LLM에도 적용할 수 있는 In-Context RALM을 제시한다.

Limitations?

- In-Context에 하나의 문서만 추가하는 경우를 고려 → 더 많은 document를 추가할수록 더 큰 이득을 가져올 수 있다.
    - 이건 근데 top-k documents와 상응하는 거 아닌가? 그건 그냥 후보군만 여러 개 만든건가..?
- 일정한 간격 (s)마다 문서를 retrieval → retrieval이 필요할 때만, 혹은 예측할 때만, 혹은 더 sparse하게 검색 ⇒ 시간과 cost를 줄일 수 있음.