[논문 리뷰] REALM: Retrieval-Augmented Language Model Pre-Training

이번 게시물에서는 최초로 retrieval와 language model을 같이 pre-training을 진행한 REALM을 제안한 논문인 REALM: Retrieval-Augmented Language Model Pre-Training에 대해 다뤄보겠다.

원문 링크는 아래와 같다.

REALM: Retrieval-Augmented Language Model Pre-Training

Introduction

BERT, RoBERTa, T5와 같은 Language model들은 학습 과정에서 parameter 안에 지식과 정보를 저장한다. 이때, 명확히 어느 부분에 어떠한 정보를 저장하는지는 알 수 없다. 또한, 이러한 model들은 결국 크기가 정해져 있기 마련인데, 이렇게 한정된 크기의 model에 세상의 모든 정보를 담기는 힘들다는 문제점이 있다.

저자들은 이러한 문제를 해결하기 위해, task 수행을 위한 knowledge를 제공하는 knowledge retrieval를 language model과 결합시키고, 두 요소를 한 번에 pre-training 시키는 새로운 model인 Retrieval-Augmented Language Model(REALM)을 소개한다.

기존 Language model들에서는, task를 수행할 때 knowledge가 필요한 경우, model의 parameter안에 저장된 knowledge를 활용하여 task를 수행하였다. 그러나, REALM은 retriever를 사용하여 large corpus로부터 knowledge를 가져오고, 이를 이용하여 task를 수행한다는 차이점이 있다.

이 과정을 그림과 함께 살펴보자

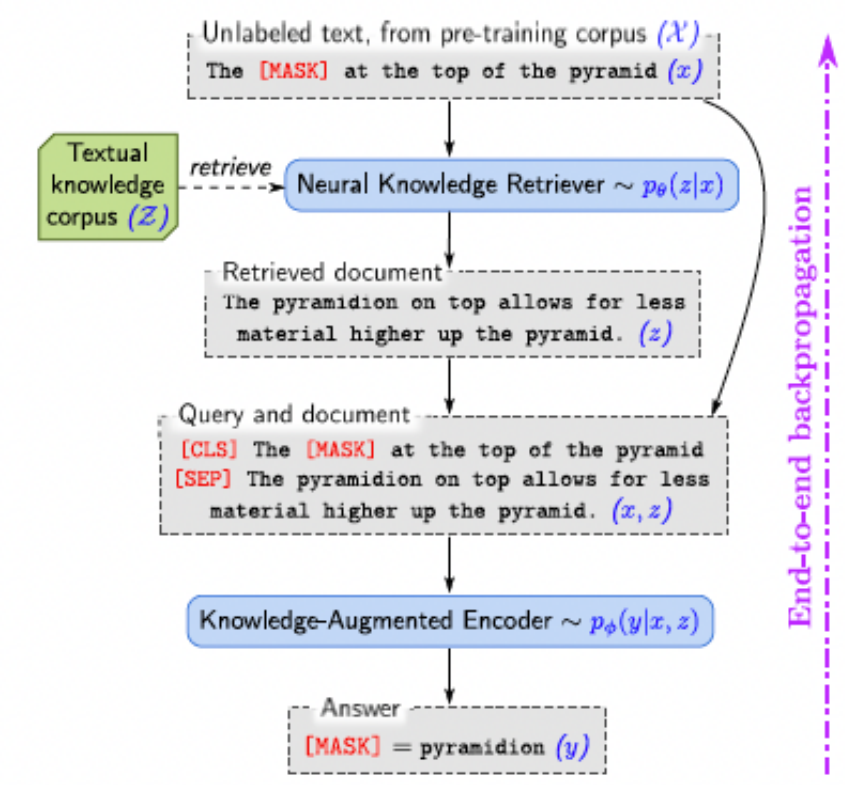

Figure 1. REALM augments language model pre-training with a **neural knowledge retriever** that retrieves knowledge from a **textual knowledge corpus**, $\mathcal{Z}$ (e.g., all of Wikipedia). Signal from the language modeling objective backpropagates all the way through the retriever, which must consider millions of documents in $\mathcal{Z}$—a significant computational challenge that we address.

해당 그림은 REALM의 pre-training과정을 담고 있다. Pre-training task는 MLM(Masked-Language Model)이다.

즉, [MASK] token으로 치환된 원래의 token을 맞춰야 하는 task이다.

이러한 task를 해결하기 위해, REALM은 model 외부의 knowledge corpus를 활용하게 되고, retriever가 이러한 knowledge corpus에서 정보를 가져오는 역할을 수행하는 것이다.

사진을 보면, Neural Knowledge Retriever는 input x를 받았을 때, knowledge corpus의 구성 요소인 document z에 대한 확률을 산출하는 것을 확인할 수 있다. 이는 아래에서 더 자세하게 다루겠지만, 간단히 말하면 input, 즉 task가 주어졌을 때 이에 적합한 document를 가져오는 것이다.

이후, 이렇게 산출된 retrieved document와 기존 input을 concat하여 하나의 sequence를 형성하고, 이를 Knowledge-Augmented Encoder에 입력하여 최종적으로 우리가 원하는 output, [MASK] token으로 치환된 원래의 token을 산출하게 된다.

이러한 과정을 통해, retriever를 model이 MLM task을 수행하는 성능에 따라 학습시키게 된다.

이를 조금 더 자세히 설명해보기 위해, 동일한 특정 task를 수행함에 있어 retriever가 document A를 가져왔을 때와 document B를 가져왔을 때가 있으며, document A를 가져왔을 때 조금 더 성능이 좋았다고 가정해 보자.

우리가 language model을 학습시킬 때, 당연히 task 수행 능력(성능)을 최대화하는 목적 함수를 구성할 것이고, 이 방향으로 학습을 하게 된다. 이때, A를 가져왔을 때 model이 성능이 더 좋았으니, A를 가져온 경우에는 당연히 retriever에게 reward를 주는 방향으로 학습할 것이고, B를 가져왔을 때는 penalty를 주는 방향으로 학습을 할 것이다.

이 과정을 계속 반복하는 것이 pre-training이며, 이러한 pre-training을 진행하면서 retriever는 주어진 task에 대해 model이 최고의 성능을 낼 수 있도록 input에 알맞은 document를 가져오게끔 학습되는 것이 REALM의 핵심 아이디어이다.

정리하자면, knowledge를 담고 있는 knowledge corpus 중에서 task에 알맞은 document를 골라서 가져오고, 이를 활용하여 풍부해진 진 knowledge와 함께 성능 향상을 꾀하자는 것이다.

그런데, 여러 document들 중에서 task에 알맞은 document라는 것을 어떻게 알 수 있을까? 저자들은 MIPS(Maximum Inner Product Search) 알고리즘을 이용한다. 간단하게 말하자면, document z와 input x의 embedding vector 간의 inner product(내적) 값을 계산하여, 이를 이용한다는 것이다. 이에 관해서도 아래에서 보다 자세히 다루도록 하겠다.

Approach

지금부터, REALM의 작동 원리 및 process에 대해 자세히 다뤄보도록 하겠다.

REALM's generative process

Pre-training과 Fine-tuning에서, REALM은 궁극적으로 $p(y|x)$, 즉 input이 주어질 때 y값의 분포를 학습하는 것을 목표로 한다.

특정 sequence가 input으로 주어졌을 때, model의 vocab에 대한 분포를 학습한다는 것이다.

("The [MASK] at the top of the pyramid"라는 input이 주어진다면, y는 [MASK]에 들어갈 model의 vocab에 포함된 token들의 분포일 것이다. 현실 세계에서는 "pyramidion"이라는 token이 들어갈 확률이 다른 token들에 비해 높을 것이며, 실제로 사람은 무의식적으로 이러한 과정을 거쳐 자연스럽게 [MASK] 자리에 "pyramidion"이 들어간다고 인지하게 된다.

Deep learning model은 궁극적으로 이러한 현실 세계의 분포를 최대한 잘 모사하도록 학습하는 것이고, 학습이 잘 된 model은 현실 세계의 token 확률 분포와 마찬가지로 [MASK] 자리에 "pyramidion"이 들어간다고 추론할 것이다. 따라서, 현실 세계에서 사용되는 language 및 knowledge로 구성된 large corpus를 통해 최대한 현실 세계의 확률 분포를 학습한다는 의미인 것이다)

Pre-training의 경우, 앞서 잠깐 이야기한 것처럼 MLM task를 수행한다. Fine-tuing시에는 Open-QA task를 수행하게 된다.

REALM은 $p(y|x)$을 구하는 과정을 retrieve, predict 두 단계로 나누어 수행한다.

먼저 input x가 주어지면, knowledge corpus $\mathcal{Z}$로부터 document z를 가져온다 (retrieve)

이는 model로 하여금 $p(z|x)$ 분포 (input x가 주어졌을 때, 각 document z가 출현할 확률 분포)를 산출하게끔 한다.

이후, retrieve 된 document z와 본래의 original input x를 입력하여, 최종 output y를 산출하게 된다. 아까 document z와 마찬가지로, model로 하여금 $p(y|z,x)$ 분포 (input x와 document z가 주어졌을 때, 각 output y가 출현할 확률 분포)를 산출하게 한다.

이후, 원래 목표로 했던 $p(y|x)$의 분포를 구하기 위해, z에 대해 marginalize 하여 $p(y|x)$를 산출하게 된다

이에 대한 수식은 아래와 같다.

$$p(y|x) = \sum_{z \in \mathcal{Z}} p(y|z,x)p(z|x)$$

knowledge corpus $\mathcal{Z}$는 여러 document z로 이루어져 있기에, 결국 $p(z|x)$ 분포는 input x가 주어졌을 때 각 document z가 출현할 확률의 모임이다. 또한, $p(y|z,x)$의 경우에도 input x와 document z가 주어질 때 각 token y가 출현할 확률이다. 우리가 model에 input으로 x를 넣어서 output y가 나오는 것은 결국 특정 document z_1가 선택된 사건과 output y가 선택될 사건이 동시에 발생하는 것이다 (곱사건).

따라서 두 사건의 확률인 $p(z_1|x)$과 $p(y|z_1,x)$을 곱해준다. 이 부분이 수식의 $p(y|z,x)p(z|x)$이다.

이 과정을 거치면 $p(y, z_1|x)$, 즉 x가 주어질 때 z_1과 y가 동시에 출현하는 확률을 구할 수 있다.

그런데, 우리는 결과적으로 $p(y|x)$를 구하고 싶다. 이를 위해, 모든 document $z_1, z_2 \dots z_n$ 에 대해 구한 $p(y, z_n|x)$값을 다 더해준다. 이 과정을 marginalize라 하며, 수식의 $\sum_{z \in \mathcal{Z}}$ 부분이다.

marginalize에 대한 부연 설명을 하자면, 동전 A, B가 있으며, 동전 A를 먼저 던진 이후 동전 B를 던지며 B의 앞뒷면 결과는 A의 앞뒷면 결과에 영향을 받는다고 가정해 보자. (두 동전은 독립적이지 않음)

동전 A는 앞면이 나올 확률이 1/4, 뒷면이 나올 확률이 3/4이며, 동전 B는 A의 결과에 따라 다음과 같은 확률을 가진다고 가정해보자.

동전 B - 앞면	동전 B - 앞면	동전 B - 뒷면	동전 B - 뒷면
동전 A - 앞면	동전 A - 뒷면	동전 A - 앞면	동전 A - 뒷면
1/12	3/12	2/12	6/12

이러한 가정 속에서, 우리는 오로지 동전 B의 결과에만 관심이 있다고 해보자 (마치 위에서 y에만 관심이 있듯이)

그래서 동전 B의 앞면 뒷면 결과에 대한 확률을 추정하고 싶은데, 이를 어떻게 구할 수 있을까?

이는 동전 B가 앞면이 나왔을 때의 확률과, 뒷면이 나왔을때의 확률을 각각 더해주면 구할 수 있다.

그러면 동전 B가 앞면이 나올 확률은 1/12 + 3/12 = 1/3, 뒷면이 나올 확률은 2/12 + 6/12 = 2/3이라는 것을 추정할 수 있다.

이와 같은 원리로 output y에 대한 확률만을 구하기 위해 각 document z에 대해 구해진 곱사건의 확률을 모두 더하는 것이 위 수식에서의 marginalize이다

정리하자면, REALM은 retrieve, predict 두 단계를 거치면서 $p(y|z,x)$와 $p(z|x)$ 를 구하게 되고, 이를 이용하여 $p(y|x)$를 산출하게 된다.

Model architecture

REALM의 구성 요소 중에서, 핵심 요소는 $p(z|x)$를 구하는 neural knowledge retriever와 $p(y|z,x)$를 구하는 knowledge-augmented encoder이다.

먼저, knowledge retriever부터 살펴보겠다. knowledge retrievers는 x가 주어졌을 때, document z에 대한 확률인 $p(z|x)$를 modeling 하며, 이는 아래와 같이 modeling 된다

$$p(z|x) = \frac{\exp f(x,z)}{\sum_{z'} \exp f(x,z')}$$
$$f(x,z) = \mathrm{Embed}_{\mathrm{input}}(x)^\intercal \mathrm{Embed}_{\mathrm{input}}(z)$$

$\mathrm{Embed}_{\mathrm{input}}(x)$와 $\mathrm{Embed}_{\mathrm{input}}(z)$는 각각 input x와 document z를 특정 d차원으로 embedding 한 embedding vector이다. 이러한 embedding vector는 BERT를 활용하여, BERT에 해당 input x와 document z를 BERT input representation의 형태로 변형한 뒤, BERT에 input으로 넣고 [CLS] token 위치의 output을 matrix W를 통해 linear transform 한 결과이다.

아래의 그림들은 이와 같은 과정의 수식이다.

$$\mathbf{join}_{\mathrm{BERT}}(x) = \texttt{[CLS]}\, x\, \texttt{[SEP]}$$
$$\mathbf{join}_{\mathrm{BERT}}(x_1, x_2) = \texttt{[CLS]}\, x_1\, \texttt{[SEP]}\, x_2\, \texttt{[SEP]}$$

$$\mathbf{Embed}_{\mathrm{input}}(x) = \mathbf{W}_{\mathrm{input}} \mathbf{BERT}_{\mathrm{CLS}}(\mathbf{join}_{\mathrm{BERT}}(x))$$
$$\mathbf{Embed}_{\mathrm{doc}}(z) = \mathbf{W}_{\mathrm{doc}} \mathbf{BERT}_{\mathrm{CLS}}(\mathbf{join}_{\mathrm{BERT}}(z_{\mathrm{title}}, z_{\mathrm{body}}))$$

이렇게 구해진 embedding matrix들끼리의 relevance score, 즉 inner product 결괏값을 구하는 $f(x,z)$를 진행한 뒤, 모든 document z에 대해 softmax를 취해주어 $p(z|x)$를 구해주게 된다.

추가적으로, 논문에서는 여기에 사용된 projection maxtrix W들과 BERT의 parameter들에 대해, retriever parameter θ라고 명시한다.

이어서, knowledge-augmented encoder에 대해 다뤄보겠다.

knowledge-augmented encoder는 input x와 document z가 주어졌을 때 output y에 관한 확률인 $p(y|z,x)$를 modeling 한다. 즉, x와 z를 input으로 받아 y를 산출해 내는 아키텍처이다.

다만, Pre-training시와 Fine-tuning시의 작동 방식은 살짝 다르다.

아래 그림을 통해 확인해 보자

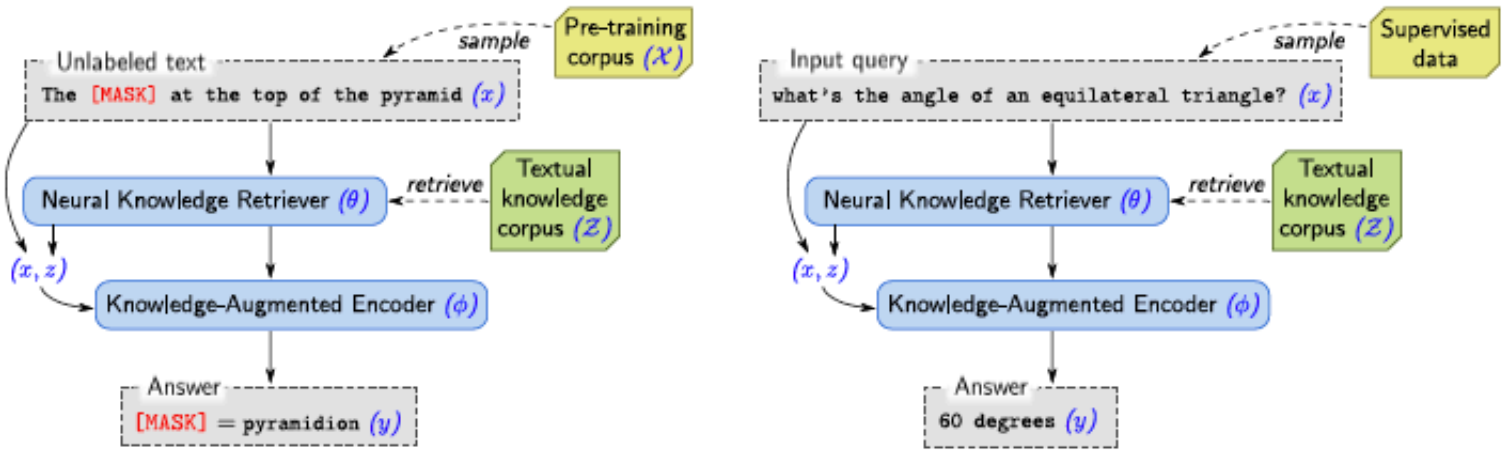

Figure 2. The overall framework of REALM. **Left:** *Unsupervised pre-training.* The knowledge retriever and knowledge-augmented encoder are jointly pre-trained on the unsupervised language modeling task. **Right:** *Supervised fine-tuning.* After the parameters of the retriever (θ) and encoder (ϕ) have been pre-trained, they are then fine-tuned on a task of primary interest, using supervised examples.

그림에서, 왼쪽이 pre-training, 오른쪽이 fine-tuning 할 때의 모습이다.

앞서 잠깐 언급했듯이, pre-training시에는 MLM task를 수행하기 때문에, [MASK] token 위치의 원래 token을 예측하게 된다. 이 과정을 나타내는 수식은 아래와 같다.

$$p(y \mid z, x) = \prod_{j=1}^{J_x} p(y_j \mid z, x)$$

$$p(y_j \mid z, x) \propto \exp\left(w_j^\top \mathbf{BERT}_{\mathtt{MASK}(j)}(\mathbf{join}_{\mathbf{BERT}}(x, z_{\mathrm{body}}))\right)$$

이 수식에서 $\mathrm{BERT}_{\mathrm{MASK}(j)}$는 [MASK] token으로 치환된 j번째 token의 위치에 해당하는 BERT output을 의미하며, J_x는 input x 안에 존재하는 [MASK] token의 전체 개수이다.

수식을 하나씩 살펴보도록 하겠다, 먼저, input x와 document의 본문 부분인 z_{body}를 BERT의 input representation 형식으로 변환한다. 즉, [CLS] x [SEP] z_{body} [SEP] 형식으로 변환된다. (이 부분이 $\mathrm{join}_{\mathrm{BERT}}$)

이후 해당 input representation을 BERT에 input으로 집어넣어, [MASK] token으로 치환된 j번째 token의 위치에 해당하는 output을 추출한다. 이 output을 vocab size로 linear transform해주는 matrix $w_j^\top$를 행렬곱을 취해주게 되면, input x와 document z가 주어졌을 때 해당 [MASK] token에 대한 확률을 구할 수 있게 된다. (여기까지가 아래 수식)

이 과정을, 하나의 input sequence x안에 존재하는 모든 [MASK] token에 대해 수행하고, 각각의 token에 대한 확률을 곱해주게 된다. (위의 수식)

Fine-tuning시에는 Open-QA task를 수행한다. 이때, input은 question이고, output y는 question에 대한 answer이다.

다만, 이때 answer y는 document z 안에 연속된 token으로 구성된 sequence로 포함되어 있다고 가정한다.

(REALM은 BERT 기반의 model이기에, generation task를 수행하지 않기 때문에 특정 corpus안에서 알맞은 부분만 추출하는 방식으로 Open-QA task를 수행하게 된다. generation task로 이러한 과정을 수행하는 model은 이후 연구에서 등장하게 된다)

해당 과정의 수식은 아래와 같다.

$$p(y \mid z, x) \propto \sum_{s \in S(z,y)} \exp\left(\mathbf{MLP}\left(\left[h_{\mathtt{START}(s)}; h_{\mathtt{END}(s)}\right]\right)\right)$$

$$h_{\mathtt{START}(s)} = \mathbf{BERT}_{\mathtt{START}(s)}(\mathbf{join}_{\mathbf{BERT}}(x, z_{\mathrm{body}})),$$
$$h_{\mathtt{END}(s)} = \mathbf{BERT}_{\mathtt{END}(s)}(\mathbf{join}_{\mathbf{BERT}}(x, z_{\mathrm{body}})),$$

이때, document z안의

추가적으로, 논문에서는 knowledge-augmented encoder의 parameter들은 ϕ 로 명시한다.

(Knowledge retriever의 parameter는 θ라고 명시했었다.)

Training

Pre-training과 fine-tuning은 공통적으로 올바른 output인 y의 log-likelihood인 $\log p(y|x)$를 maximize 하는 것을 목적으로 학습된다. 학습 과정 속에서, knowledge retriever와 knowledge augmented encoder는 diffentiable 한(훈련 가능한) neural network로 구성되었기 때문에, $\log p(y|x)$에 대한 gradient를 계산하여 update를 할 수 있다.

그런데, 이 과정에서 문제가 하나 발생한다. 위에서 언급한 것처럼, $p(y|x)$는 $\sum_{z \in \mathcal{Z}} p(y|z,x)p(z|x)$ 과정을 통해 구해진다. 수식 상 전체 document에 대한 모든 확률을 더하게 되는데, 이는 너무 많은 계산량을 요구하는 문제가 발생한다.

논문에서는 이러한 문제를 해결하기 위해, 전체 document를 대상으로 확률을 더하지 않고, top-k document에 대해서만 summation을 함으로써 전체 document에 대한 $p(y|x)$를 approximate, 즉 근사하게 된다. 저자들은 대부분의 document들이 0에 가까운 확률을 가지기에, top-k document에 대해서만 계산하여 근사하는 방식이 합리적이라고 주장한다.

이러한 top-k document만을 이용하는 방식은 계산량을 많이 줄여줄 수 있다. 그러나, 그렇다면 top-k document를 어떻게 찾을 것인가?라는 의문이 발생하기 시작한다. 만약 top-k document를 골라내는 과정이 비효율적이라면, 결국 전체적인 효율성에서 뒤처질 수 있기 때문이다.

논문에서는 이에 대해 MIPS(Maximum Inner Product Search) 알고리즘을 사용한다고 말한다.

위에서 잠깐 언급했지만, input x와 document z의 relavance score로 inner product, 내적 연산값을 사용한다.

$$p(z|x) = \frac{\exp f(x,z)}{\sum_{z'} \exp f(x,z')}$$
$$f(x,z) = \mathbf{Embed}_{\text{input}}(x)^\intercal \mathbf{Embed}_{\text{input}}(z)$$

Relavance score로 inner product를 사용하기에, MIPS 알고리즘 적용이 가능해지며 이는 top-k document를 찾을 때, document 수가 늘어날 때마다 요구되는 running time과 storage space가 sub-linearly(선형 시간 이하, eg.$O(\log n)$)한, 매우 효과적인 결과를 도출할 수 있다.

이러한 MIPS를 적용하기 위해, REALM model은 모든 document z에 대해 $\mathbf{Embed}_{\text{input}}(z)$를 미리 계산해야 한다. 그러나, 이러한 $\mathbf{Embed}_{\text{input}}(z)$는 model의 학습이 진행되면서 계속 바뀌게 된다.

앞에서 언급했던 내용을 다시 확인해 보자.

$$\mathbf{Embed}_{\text{input}}(x) = \mathbf{W}_{\text{input}}\text{BERT}_{\text{CLS}}(\text{join}_{\text{BERT}}(x))$$
$$\mathbf{Embed}_{\text{doc}}(z) = \mathbf{W}_{\text{doc}}\text{BERT}_{\text{CLS}}(\text{join}_{\text{BERT}}(z_{\text{title}}, z_{\text{body}}))$$

$\mathbf{Embed}_{\text{doc}}(z)$는 parameter θ를 가진 knowledge retriever로부터 산출된다. 그런데 학습이 진행되면서 knowledge retriever의 parameter θ와 knowledge augmented encoder의 parameter ϕ는 계속해서 update 된다.

Parameter가 update 된다는 것은 같은 input을 넣었을 때 결과도 달라진다는 의미인데, document z에 대해 $\mathbf{Embed}_{\text{doc}}(z)$를 미리 계산했을 때는 parameter θ가 update 되기 전이기 때문에, update 된 parameter로 산출된 $\mathbf{Embed}_{\text{input}}(x)$와 함께 다뤄지는 과정에서 모순이 발생하게 된다. 그렇다고 매 training step마다 전체 document에 대해 re-embedding과 re-indexing을 진행하는 것은 비효율적이다.

논문에서는 이 문제를 해결하기 위해, 몇백 training step마다 비동기적으로 document에 대해 re-embedding과 re-indexing을 진행하는 방법을 사용한다. 이 방법 또한 update가 진행되지 않는 training step들에서 살짝 모순이 발생하기는 하지만, 저자들은 실험을 통해 해당 방법론이 충분히 좋은 성능을 냈다고 말한다. (또한, top-k를 가져오는 것이기에, 발생하는 약간의 모순이 상쇄된다고도 말한다.)

이 과정을 그림으로 나타내면 아래와 같다.

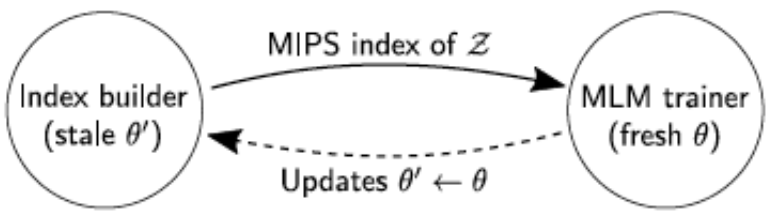

Figure 3. REALM pre-training with asynchronous MIPS refreshes.

해당 과정은 크게 두 파트로 나뉘게 된다.

- Trainer job : parameter에 대한 gradient update를 계속해서 진행 (θ)
- Index builder : document z들에 대해 embedding과 indexing을 진행 (θ')

훈련이 진행되는 과정은 아래와 같다

1. Trainer가 본인의 parameter를 index builder로 전달 (이때, 전달하는 parameter를 θ' 라고 한다)
2. Index builder는 trainer로부터 받은 parameter θ'를 가지고 document에 대해 re-embedding과 re-indexing을 진행
3. 2번에서 index builder가 re-embedding과 re-indexing을 하는 것과 독립적으로, trainer는 계속해서 학습을 진행하며 parameter update
4. Index builder가 document에 대한 작업을 모두 끝내면, trainer에게 새로운 MIPS index를 전달
5. Trainer는 새로운 MIPS index로 학습과 parameter update를 계속해서 이어가며, Index bulider에게 다시 parameter를 전달

위의 1번부터 6번까지의 과정을 반복하며 학습이 이루어진다.

이러한 asynchronous refreshing은 pre-training과 fine-tuning 모두에 적용가능하지만, 논문에서는 pre-training에만 적용하고, fine-tuning시에는 MIPS index를 pre-train 된 parameter θ로 한 번만 구축한 뒤, refresh를 하지 않았다고 한다.

간단하게 말해서, fine-tuning이 진행되는 과정 동안 $\mathbf{Embed}_{\text{doc}}$을 update 하지 않는다는 것이다.

그러나, $\text{Embed}_{\text{input}}$에 대해서는 계속 update 되기 때문에, retrieval function은 input x에 대해서는 계속해서 update가 된다고 말한다.

Injecting inductive biases into pre-training

논문에서는 REALM을 훈련시킬 때, retrieval이 보다 의미 있는 방향으로 학습되기 위해 아래와 같은 추가적인 기법을 활용했다고 한다.

- Salient span masking
- Null document
- Prohibiting trivial retrievals
- Initialization

먼저, Salient span masking부터 알아보겠다.

REALM의 pre-training은 MLM task를 수행한다. 그런데, MLM task를 수행할 때 masking이 random 한 token에 대해 적용되기 때문에, REALM의 목적이 맞지 않는 masking이 진행될 수 있다.

REALM은 knowledge가 필요한 task, 즉 knowledge-intensive task를 잘 수행하기 위해 만들어지는 model이기 때문에, 논문에서는 이러한 목적을 잘 수행하기 위해 masking도 knowledge가 필요한 token 위주로 진행하는 salient masking을 수행한다고 한다.
예를 들어, 아래와 같은 문장이 있다고 가정해 보자

> 1년 중에서 대한민국의 광복절은 8월 15일이다.

이 문장에서, knowledge가 필요한 부분은 "8월 15일", 혹은 "광복절"이다. 따라서, REALM에서의 salient masking은 "8월 15일", 혹은 "광복절"에 집중하여 masking을 한다는 것이다.

이러한 salient masking은 BERT 기반의 NER tagging model과 정규표현식을 이용하여, salient span(핵심적인 단어)를 골라내고, 해당 span에만 masking을 하게끔 구현된다.

Null document란, retriever가 가져오게 되는 top-k document에 null document를 추가하는 기법이다. 이는 masking 된 token을 predict 할 때, knowledge가 필요 없는 경우에는 이러한 null document를 retrieve 하게끔 하여 model의 retriever로 하여금 알맞은 document를 가져오게끔 하는 기법이다.

Prohibiting trivial retrievals은, pre-training corpus와 knowledge corpus가 같은 경우를 방지하는 기법이다. 만약 masking 된 input x가 document z에 있는 문장과 같은 문장이면, model은 x와 z의 관계를 학습하는 것이 아닌, 순전히 문자열이 matching 되는지 확인하는 방향으로 학습될 가능성이 있다. 따라서, 저자들은 pre-training 과정에서 이러한 경우들을 제외시켰다고 한다.

마지막으로 Initialization이다. 원활한 학습을 위해, 기존 ORQA의 parameter를 knowledge-retriever의 초깃값으로 설정하였고, knowledge-augmented encoder의 parameter의 초깃값은 uncased BERT-BASE model의 parameter로 설정하고 학습을 진행하였다고 한다.

Experiments

저자들은 지금까지 소개한 REALM을 가지고 Open-QA task에 대해 성능 확인을 해보았다.

자세한 실험 조건, 각 benchmark dataset에 대해서는 추후 소개하도록 하겠다.

Main results

Open-QA task에 대해 타 모델들과 REALM의 성능을 측정한 결과는 아래와 같다.

Table 1. Test results on Open-QA benchmarks. The number of train/test examples are shown in paretheses below each benchmark. Predictions are evaluated with exact match against any reference answer. Sparse retrieval denotes methods that use sparse features such as TF-IDF and BM25. Our model, REALM, outperforms all existing systems.

Name	Architectures	Pre-training	NQ (79k/4k)	WQ (3k/2k)	CT (1k /1k)	# params
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

REALM이 모든 task에 대해 SOTA를 달성한 것을 확인할 수 있다. 특히, T5와 비교했을 때 parameter 수에서 큰 차이를 보이는데도 불구하고 REALM의 성능이 더 우수한 것을 확인할 수 있다.