REAL: Retrieval-Augmented Language MOdel Pre-Training

Shubham Gupta

February 20, 2020

1 Introduction

- REALM is a paper mentioned in the T5 paper titled: **How Much Knowledge Can You Pack Into The Parameters of a Language Model?**
- TLDR: This paper retrieves documents that have the information present while solving Question-Answer type problems.
- Introduced a latent *knowledge retriever*, which can attend and retrieve documents over large corpus and can be trained in unsupervised manner using masked language modelling technique and backprop through retreiver which considers lots of docs.

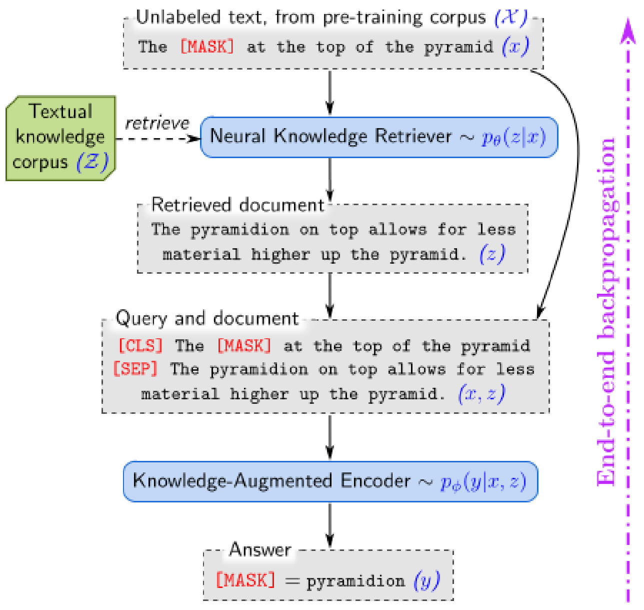
Unlabeled text, from pre-training corpus $(\mathcal{X})$
The [MASK] at the top of the pyramid (x)
Textual knowledge corpus $(\mathcal{Z})$
retrieve
Neural Knowledge Retriever $\sim p_\theta(z|x)$
Retrieved document
The pyramidion on top allows for less material higher up the pyramid. (z)
Query and document
[CLS] The [MASK] at the top of the pyramid [SEP] The pyramidion on top allows for less material higher up the pyramid. (x, z)
Knowledge-Augmented Encoder $\sim p_\phi(y|x, z)$
Answer
[MASK] = pyramidion (y)
End-to-end backpropagation

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.

Figure 1: Training process for REALM

- Key point: Train retriever using a performance-based signal from unsupervised text.
- Retrieval based LM =¿ Moar computational resources =¿ Moar money
  - Solution: Computation performed for each doc is cached and can be used again. Best doc selected using *Maximum Inner Product Search(MIPS)*. Read the paper here.
- REALM retriever can be used on downstream tasks via transfer learning.
- REALM is SOTA on NQ-Open, WQ and CuratedTrec.

2 Approach

2.1 *Retreive-then-predict generative process*

- Training: Masked-LM. Fine-tuning: Open QA task
- $p(y|x)$ decomposed into two steps:

  - Given x,retrive documents z from corpus Z. Modelled as: $p(z|x)$
  - Condition of both z and x to generate output y i.e p(y—z, x)
  - Overall likelihood y is generated by treating z as latent variable and marginalizing over all documents z

$$p(y|x) = \sum_{zZ} p(y|z,x) * p(z|x) \tag{1}$$

2.2 Architecture

- **Neural Knowledge Retriever** i.e models $p(z|x)$
- **Knowledge Augmented Encoder** i.e models p(y—z, x)

2.3 Neural Knowledge Retriever

- Dense inner product model.

$$\begin{aligned} p(z|x) &= \frac{exp(f(x,z))}{\sum_{z' exp(f(x,z')}} \\ f(x,z) &= Embed_{input}(x)^T Embed_{doc}(z) \end{aligned} \tag{2}$$

- $Embed_{input}$ and $Embed_{doc}$ are embedding functions
- f(x,z) is called **relevance score**. It is inner product of vector embeddings.
- Relevant Distribution is softmax over all relevance scores
- Embedding implement using BERT-style transformers. Join using ¡SEP¿, prefix using ¡CLS¿ and append ¡SEP¿ as the end.

$$\begin{aligned} join_{BERT}(x) &= [CLS]x[SEP] \\ join_{BERT}(x_1, x_2) &= [CLS]x_1[SEP]x_2[SEP] \end{aligned} \tag{3}$$

- Pass above into transformer, which gives over vector for each token. Perform linear projection to reduce dimensionality of vector

$$\begin{aligned} Embed_{input}(x) &= W_{input} BERT_{CLS}(join_{BERT}(x)) \\ Embed_{doc}(z) &= W_{doc} BERT_{CLS}(join_{BERT}(z_{title}, z_{body})) \end{aligned} \tag{4}$$

2.4 Knowledge-Augmented Encoder

- Given input x and relevant doc z, this defines $p(y|z,x)$
- Join x and z into single sequence and feed into transformer
- Here, training is different for pre-training vs fine-tuning
  - For pre-training, predict [MASK] token. Use same Masked LM(MLM) loss as in Transformer(Devlin)
  - For Open-QA, we need to produce string y. Assumption: y occurs as sequence of tokens in some document in the corpus. Skipping the math bit for now.

2.5 Training

- Compute gradients in θ and ϕ and optimize using SGD.
- Challenge: Computing $p(y|x)$
- Approx by summing over top k documents with highest prob under $p(z|x)$
- Question: How to find top k docs? Answer: Use MIPS
- Need to precompute $Embed_{doc}(x)$ for all docs. Problems? It changes with each step of SGD.
- *Solution*: Async refresh $Embed_{doc}$ every 500 steps
- Use MIPS to select top k docs. For these docs, recompute $p(z|x)$ using new θ.