Summary of REALM & RETRO

Mehar Deep Singh (mehars), Parth Raut (praut), Zachary Eichenberger (zeichen)

Problem and Motivation

Language model (LM) training traditionally focuses on pre-training on internet scale data and fine-tuning on a smaller dataset that is specific to a particular downstream task. Often times however, we have a corpus of documents that we want our LM to be able to reference explicitly as it generates its output. For example, consider a company having a proprietary code base that a copilot LM can use to output useful code.

Retrieval augmented generation (RAG) aims to solve this problem. It uses a corpus of selected documents and their embeddings in a vector DB. A similarity is computed over an embedding of an input prompt and each document in the DB. The documents with the highest similarity can be used to *augment* the LM and provide it extra context/evidence for next token prediction.

RAG also offers a parameter efficient method to improve LM performance. The following papers offer methods to use large datasets of documents to 1) fine-tune LMs themselves in a general fashion for retrieval & token generation, and 2) explore how increasing the scale of the DB impacts LM performance.

Related Works

Previous language models, such as BERT, T5, and GPT3 transformers learn knowledge stored implicitly in neural network weights. However, recent work has sought to integrate more explicitly information from high quality training corpuses to improve the answer quality.

Other approaches, such as Memory Networks (Weston et.a., 2014), and NeuralEditor (Guu et. al., 2018) introduce the concept of explicitly using the training corpus to generate output. The former used explicit storage of previously seen elements, searched for relevant sentences, and searched for a token among them as an answer. The latter used a similar approach for general language modeling, searching a database for a semantically similar sentence, and training a model to edit it to fit within the context, rather than generating sentences from scratch.

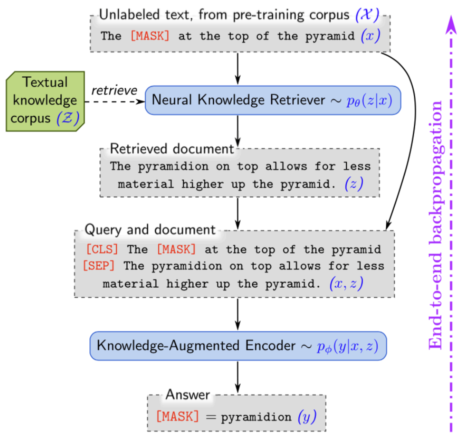
A significant amount of research in the QA domain likewise works with explicit searching to augment question answering techniques. For instance, DrQA (Chen et. al., 2017) used classical document search techniques to search for relevant passages in Wikipedia documents, appended those documents as context, and produced an output using a classical RNN model which took both the document and query as input, and produced relevant answer tokens. Meanwhile, *Key-Value Memory Networks for Directly Reading Documents* expands this to multi-hop formats, repeatedly searching for relevant documents to augment a query to perform multi-hop reasoning (Miller et.al., 2016).

Finally, *Retrieval Augmented Generation for Knowledge-Intensive NLP Tasks*, released after REALM, uses a similar approach to REALM, training a query encoder, searches for documents, and incorporates them as context with a final text-to-text generation step (Lewis et. al. 2021).

Solution Overview

REALM

The approach of REALM is to break training and finetuning into a two step process: 1) retrieving and 2) predicting. This process is shown in the diagram below:


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

This retrieve then predict approach is formalized in the equation below:

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

Where $p(z|x)$ is the probability of retrieving a document z and $p(y \mid z, x)$ is the probability of generating output y given the retrieved document z and original input x. The knowledge retriever, which models $p(z|x)$ is defined using a dense inner product model, where the relevance score between x and z (calculated as an inner product of vector embeddings) is computed for each pair before performing a softmax to find documents with the highest relevance to the input x for the retrieval. BERT-style transformers were used for the embedding functions. The knowledge-augmented encoder, which modeled $p(y \mid z, x)$, was implemented by appending x and z together before feeding this concatenated input into a transformer.

The pretraining was done in an unsupervised masked language modeling, while the finetuning was done in a supervised question-answering task (Open-QA). This process is summarized in the figure below:

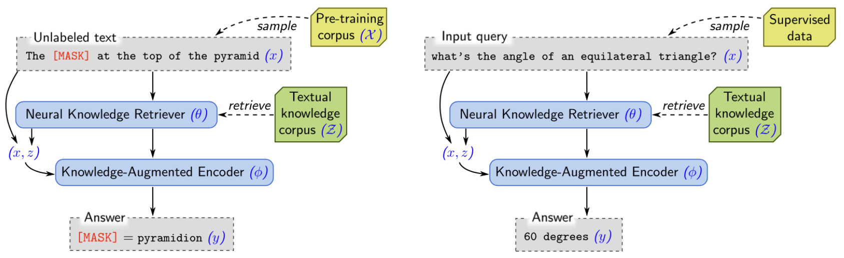

Unlabeled text
The [MASK] at the top of the pyramid (x)
sample
Pre-training corpus $(\mathcal{X})$
Neural Knowledge Retriever (θ)
retrieve
Textual knowledge corpus $(\mathcal{Z})$
(x, z)
Knowledge-Augmented Encoder (ϕ)
Answer
[MASK] = pyramidion (y)
Input query
what's the angle of an equilateral triangle? (x)
Supervised data
60 degrees (y)

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.

To ensure computational efficiency, only the top k documents under $p(z|x)$ were considered. This was accomplished through Maximum Inner Product Search (MIPS). In doing so, the embeddings for documents were precomputed and kept unmodified until they were "refreshed" by re-embedding and re-indexing the dataset after every several hundred training steps.

Many additional strategies were used to further optimize the model. Salient spans (ex. "United Kingdom") were masked together, ensuring world knowledge was required. Additionally, a null document was employed when the model should not require any world knowledge in prediction. Trivial retrievals were prohibited by removing documents that were too informative. The retriever was warmed up to ensure that retrieved documents were not random and being ignored by the downstream encoder.

The REALM approach outperformed all existing systems. Using Wikipedia and CCNews was better than just Wikipedia, and the encoder and decoder combination performed the best. The MIPS index refresh rate of 500 training steps was found to be ideal.

RETRO

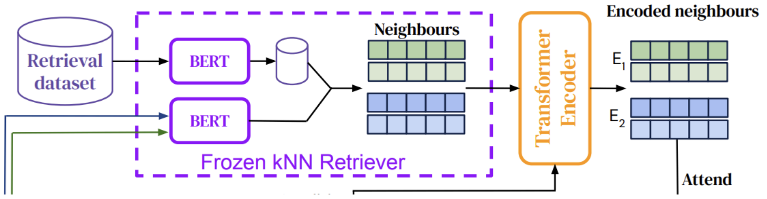
In RETRO, input tokens sequences are separated out into contiguous token chunks. For example, the input text "LLMs are a great new tool" might be separated into 2 chunks of tokens:

chunk1 = ["LLMs", "are", "a"], chunk2 = ["great", "new", "tool"]

During the retrieval step, chunks of tokens are retrieved from the DB instead of individual tokens being retrieved, which 1) reduces storage, and 2) reduces compute requirements by a "large linear factor."

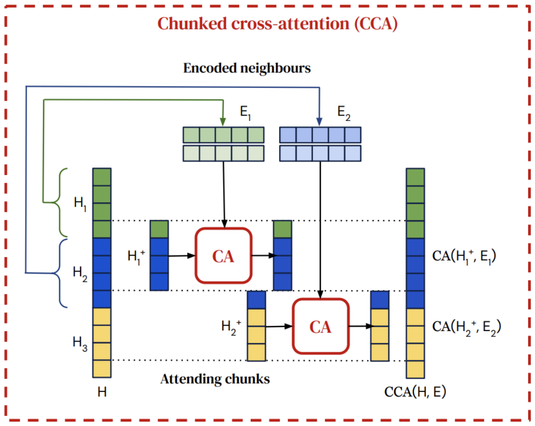
Once the DB documents are chunked, a new key-value DB is constructed by using a frozen LM (such as BERT) to encode each chunk into a vector. Using this frozen model ensures that there is no need to recompute the embeddings for each chunk during training. The K-V DB can then be represented as a set of pairs (key: embedding vector, value: chunk).

Then, giving a sequence of tokens from the training set, that sequence can similarly be segmented into contiguous chunks. Each chunk is "augmented" with their *k* nearest chunks in the DB based on the LM embedding. They use the L2 distance to determine which DB entries are similar. Instead of computing an exact *k* Nearest Neighbors, they use an approximation from the SCaNN library that can approximate *k*NN in *O(log T)*, where *T* is the number of DB chunks.



The diagram above summarizes the retrieval process. The blue and green arrow on the LHS represent chunks from the input sequence, and 2 neighbors from the DB are selected for each chunk. Then, each retrieved chunk is encoded via a bidirectional Transformer Encoder to produce retrieval encodings for each input chunk, as shown by *E1* and *E2* on the RHS.

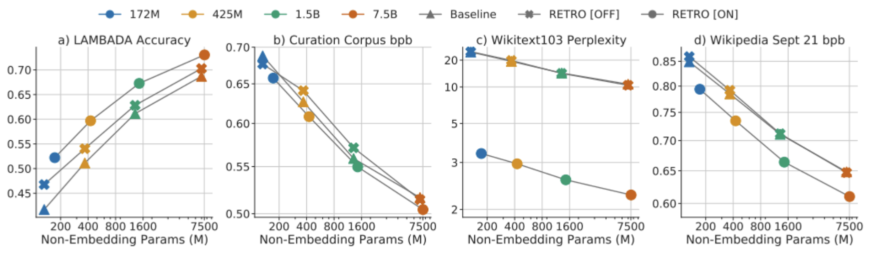
Finally, an encoder-decoder LM integrates the retrieval encodings *E1, E2, …, En* into its output next-token prediction. Specifically, the encoder-decoder LM uses a special causal cross attention called Chunked Cross Attention (CCA). The encoding for an input chunk *Hj* only attends to the retrieved encodings for previous chunks *E1, …, Ej* (since it cannot attend to future retrieved encodings, this is a causal attention mechanism). The figure below depicts the CCA graphically.



After a feed-forward head module for next-token prediction, the loss function can be written as follows:

$$L\left(X|\theta,\mathcal{D}\right) \triangleq \sum_{u=1}^{l}\sum_{i=1}^{m} \ell_\theta\left(x_{(u-1)\,m+i}|(x_j)_{j<(u-1)\,m+i},\ (\textsc{Ret}_{\mathcal{D}}(C_{u'}))_{u'<u}\right)$$

which can be simply explained as the loss of predicting the *j* th token given all previous *j-1* tokens and all previous retrieved DB chunks.



As shown above, RETRO was applied to extremely large DBs, and performance sees significant improvement the larger the DB gets (especially > 1T tokens). They use a bits per byte (bpb) metric which is similar to entropy in information theory for predicting the next token given the LLM next token distribution (lower bpb = better since the LM is confident about the next token). Additionally, RETRO out-performs baseline LMs without retrieval on many datasets, such as GitHub.

Limitations

Both RETRO and REALM require a significant offline corpus on which to perform its retrieval step, and searching of this corpus needs to be done in concert with the text generation steps required of a language model. This may pose some logistical challenges.

Additionally, REALM may have high training costs (due to additional backpropagation of the retriever) only nets in 2% gain on text in evaluation, and may have difficulty scaling, as documents are needed to be the same for evaluation and training. Finally, The paper lacks thorough testing of the MIPS index refresh rate.

Future Research Directions

REALM

1) Structured knowledge retrieval: creating a document-level embedding for vector search captures high level information about the document, but users might only be interested in a certain characteristic: relevance, truthfulness, emotion, etc.
2) Expanding retrieval to a multi-lingual and multi-modal setting
3) Using retrieval for multi-doc augmentation instead of single doc augment

RETRO

1) Evaluate the performance of RETRO on Natural Language Understanding (NLU) tasks
2) Understanding the role of test set leakage (same docs in training set and DB) and how methods like Locality Sensitive Hashing (LSH) can help remove duplicates

Summary of Class Discussion

The discussion began with a focus on the retrieval. Someone asked whether multiple documents could be retrieved, and the consensus was that this was not possible with the current design, but should not be hard to extend to implement this feature. Next, a hierarchical retriever was proposed where similar to MoE, there could be N lower level retrievers of the DB to do some type of ensemble. Another configuration was briefly discussed, where a gating network to choose the retrieval where retrievers were responsible for DB documents in particular topic areas. These were noted as possible improvements to the design.

Next, the focus shifted towards inference. The impact of REALM on the inference latency was brought up, including inference performance and the breakdown of time spent between retrieval and forward pass on the downstream encoder. The presenters hypothesized that the lookup stage would be a very small portion of the evaluation time, but might pose interesting scheduling challenges. Although database sizes for both models, but especially RETRO are large, they do not take up GPU memory, and therefore form less of a computational challenge. It was noted by the presenters that both papers lacked meaningful discussion on inference, as these were published prior to the release of ChatGPT and the democratization of LLMs.

Additionally, the frequency of MIP index updates for REALM was discussed, and an idea was proposed to dynamically adjust MIPS updates during training. For example, during initialization the updates could be more frequent to aid in the cold-start problem.

On the RETRO paper, a question was asked about whether there could be a dynamic chunking size that fits relevant tokens into the same chunk. This was seen as a good idea, but also one that could increase the time of training significantly since 1) each of the documents from the extremely large DB would have to be segmented intelligently, and 2) each input training sample would also have to be segmented intelligently.

Finally, the important question was asked: **What is grounding?** One of the presenters explained that at Microsoft, grounding provides the opportunity to base LM output on a particular DB of text *after* pretraining of the LM weights.