(Generative) Open-domain Question Answering

Tahar Chettaoui B.Sc.
ukewy@student.kit.edu

Supervisor: Fabian Retkowski M.Sc.
fabian.retkowski@kit.edu

Abstract

Question answering (QA) is concerned with building systems that automatically answer questions posed by humans in a natural language. The retriever-reader architecture is dominating, and the extractive reader is mostly chosen over the generative reader, while there are several advantages to generating answers even when it is possible to extract them. Documents with clues about the answer, but do not contain the exact answer can still contribute towards a correct answer being generated, which is not possible with standard extractive approaches. In this paper we will explore a generative approach, namely Retrieval-Augmented Generation (RAG), and show it can compete with state-of-the-art approaches.

1 Introduction

Question answering (QA) is concerned with building systems that automatically answer questions posed by humans in a natural language. Compared with a search engine, the QA system aims to present the final answer to a question directly instead of returning a list of relevant snippets or hyperlinks. Nowadays, many web search engines like Google or Bing are incorporating QA techniques into their search functionalities to respond precisely to some types of questions.

We can separate the QA systems in two categories, according to the source of information. The first category: Knowledge Base (KB)-QA fetch answers from a predefined structured database that is often manually constructed. The second category, namely Textual QA, mines answers from unstructured text documents. The second approach is more scalable since most of the unstructured text resources it exploits to obtain answers are easily accessible, such as Wikipedia, news articles and science books. Textual QA can be divided again in two categories based on the availability of the context, with the first one being Machine Reading Comprehension (MRC). MRC aims to enable machines to read and comprehend a specified context passage to answer a given question. The second approach is Open-domain QA (OpenQA) and it tries to answer a given question without any specified context. OpenQA aims to answer a question in the form of natural language based on large-scale unstructured world knowledge. So building a system that is capable of answering any input questions is the ultimate goal of QA research.

In section 2, we will go through some of the latest papers presenting solutions based on generative models. In section 3, the methodologies and theory behind each approach will be presented. And finally section 4 will present the experiments and results of the previously defined methods before concluding with an outlook in the last section.

2 Related work

In this section, we first introduce the traditional approaches of OpenQA and then review the modern solutions based on deep learning.

2.1 Traditional OpenQA

As illustrated in figure 1, traditional OpenQA systems mostly follow a pipeline consisting of three stages:

1. Question Analysis
2. Document Retrieval
3. Answer Extraction

Question Analysis have two goals. The first goal is to facilitate the retrieval of question-relevant documents, for which a query formulation module is used to generate search queries. In query formulation, linguistic techniques such as POS tagging, parsing, stemming and stop word removal are

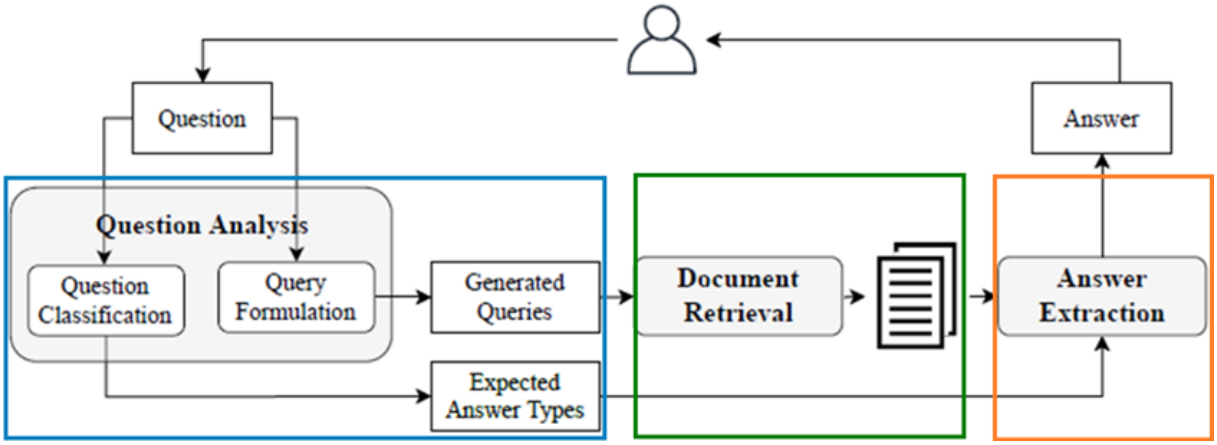

Figure 1: Traditional architecture of OpenQA systems (Zhu et al., 2021)

usually utilized to extract keywords for retrieving. However, the terms used in questions are often not the same as those appearing in the documents that contain the correct answers. This problem is called "term mismatch" and is a long-standing and critical issue in information retrieval. To address this problem, query expansion and paraphrasing techniques are often employed to produce additional search words or phrases to retrieve more relevant documents. The second goal of QA is the classification of questions. This module aims to identify the type of the given question based on a set of question types (e.g., where, when, who, what) or a taxonomy manually defined by linguistic experts. After obtaining the type of the question, expected answer types can be easily determined using rule-based mapping methods. Identifying the question type can provide constraint upon answer extraction and significantly reduce the difficulty of finding correct answers.

In the document retrieval phase, the system searches for question-relevant documents or passages with the generated search queries, usually using existing information retrieval techniques like TF-IDF and BM25, or techniques developed for web search engines. In the last stage, namely answer extraction, the final answer is extracted from related documents received from the previous stage.

2.2 Modern OpenQA

The architecture is changing with the years and nowadays, we have a Retriever-reader architecture, which is the modern architecture for OpenQA. It consists of a retriever and a reader. The retriever will get the related documents or passages that probably contain the correct answer, as well as rank them in descending order according to their relevancy. The reader have the goal of getting the final answer from the received documents. Optional modules like the document and answer post-processing could be added for better results. There are two types of Readers:

- Extractive Reader
- Generative Reader

Extractive reader is based on the assumption that the correct answer to a given question definitely exists in the context, and usually focuses on learning to predict the start and end position of an answer span from the retrieved documents. Most OpenQA systems are equipped with extractive reader. Instead of extracting answer spans, the generative reader aims to generate answers as natural as possible, usually relying on Seq2Seq models.

2.3 Language models

Models that assign probabilities to a sequence of words are called language models. These models learn linguistic knowledge, but also they tend to store relational knowledge present in the training data. Recent papers showed that they have a surprisingly strong ability to recall factual knowledge without any fine-tuning, which demonstrates their potential for the open-domain QA. The goal of language-model pre-training is to learn useful representations of language, usually from unlabeled text corpora like Wikipedia. So the resulting pre-trained model can then be further fine-tuned for a particular task, in our case OpenQA, often leading to better generalization than training from scratch.

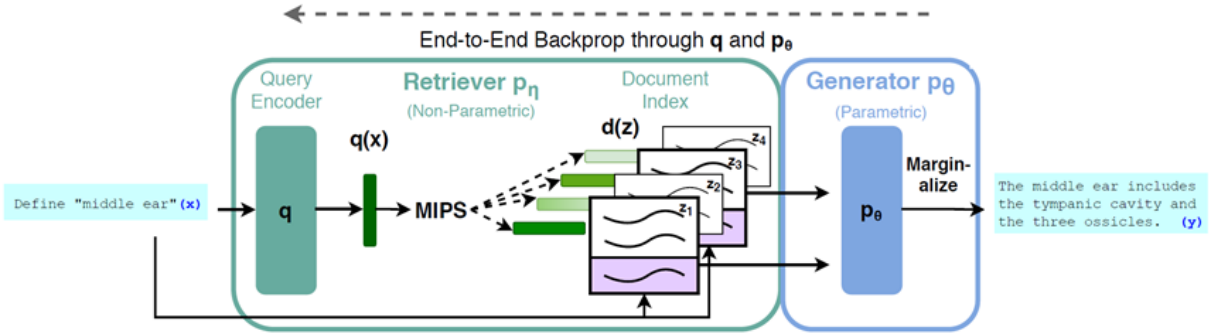

Figure 2: RAG training pipeline (Lewis et al., 2020)

Models such as BERT (Devlin et al., 2018) and T5 (Raffel et al., 2019) store a surprising amount of world knowledge, acquired from the massive text corpora they are trained on. In these language models, the learned world knowledge is stored implicitly in the parameters of the neural network. So we can't really say what knowledge is stored in the network and where. Also, the storage space is limited by the size of the network, so there will always be a trade-off between capacity, performance and speed.

3 Methodology

In this section, we first present an extractive approach REALM and then introduce RAG a generative approach, based on DPR and BART.

3.1 Retrieval-Augmented Language Model Pre-Training (REALM)

Retrieval-Augmented Language Model Pre-Training (REALM) (Guu et al., 2020) is a pre-trained masked language model. A Masked language model is trained to predict the missing tokens in an input text passage. So for a sentence from a pre-training corpus, like Wikipedia, and some words masked out, the model must predict the value of those missing tokens. In contrast to models that store knowledge in their parameters, this approach explicitly exposes the role of world knowledge by asking the model to decide what knowledge to retrieve and use during inference. REALM has a neural Retriever and a neural Reader and is able to compute the gradient w.r.t. the model parameters and back-propagate the gradient all the way throughout the network.

As shown in figure 3, REALM takes some input x and learns a distribution $p(y|x)$ over possible outputs y. REALM decomposes $p(y|x)$ into two steps: retrieve, then predict. Given an input x, we first retrieve possibly helpful documents z from a knowledge corpus Z. We model this as a sample from the distribution $p(z|x)$. Then, we condition on both the retrieved z and the original input x to generate the output y modeled as $p(y|z, x)$. The neural knowledge retriever models $p(z|x)$ And the knowledge-augmented encoder models $p(y|z, x)$. After the parameters of the retriever and encoder have been pre-trained, they are then fine-tuned on OpenQA, using supervised examples. So the model is fine-tuned with pairs of questions answers. While REALM and other approaches have shown promising results, they have only explored open-domain extracting question answering, but how about the generative approach?

3.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) architecture, illustrated in figure 2, have 2 components, a parametric and a non-parametric one. Its reader, a generator, takes the input question as well as the best K documents chosen by the retriever.

The retriever (DPR) takes the question as input and returns K indexed document. The question x is fed to the query encoder, which gives us the encoded question $q(x)$. Maximum inner product search (MIPS) finds the item with the highest inner product. It is solved using FAISS algorithm. Once the K most relevant passages obtain, they are fed to the generator along the input question x. Then the seq2seq model (BART) generate the output. To combine the input x with the retrieved content z when generating from BART, they simply concatenate them.

The retriever and generator are jointly trained. Given a fine-tuning training corpus of input/output pairs (x; y), we minimize the negative marginal

log-likelihood of each target. We only fine-tune the query encoder BERT$_q$ and the BART generator, since updating the document encoder BERT$_d$ during training is costly and not necessary.

3.2.1 The retriever - Dense Passage Retrieval (DPR)

Dense Passage Retrieval (DPR) (Karpukhin et al., 2020) paper focus on improving the retrieval component in open-domain QA. Given a collection of M text passages, the goal of DPR is to index all passages in a low-dimensional space, such that it can retrieve efficiently the top k passages relevant to the input question for the reader at run-time.

DPR, as shown in figure 4 works by using two BERT encoder models. One of those models, E_P encodes passages of text into an encoded passage vector. The other model E_Q maps a question into an encoded question vector. During training, we feed a question-context pair into the system, and the model weights will be optimized to maximize the dot product between two respective model outputs. By training the two models to output the same vector, we are training the context encoder and question encoder to output very similar vectors for related question-context pairs. The goal is to create a vector space such that relevant pairs of questions and passages will have smaller distance. The dot product value between the two model outputs $E_P(p)$ and $E_Q(q)$ measures the similarity between both vectors.

The document store will contain all passage vectors, we run every passage of text through the E_P encoder and store the output vectors in the document store. For real-time QA, we only need the E_Q encoder. When a question is asked, it will be sent to the E_Q encoder, which then outputs a vector. Then we get the K best vectors from the document store w.r.t. the question. This step might seem to need great time and computation power, but they are using FAISS, a library for similarity search and clustering of dense vectors, which can easily be applied to billions of vectors, thus retrieving fast the K most relevant documents. Finally, the E_Q vector is compared against the already indexed E_P vectors in our document store, which return the highest similarity score.

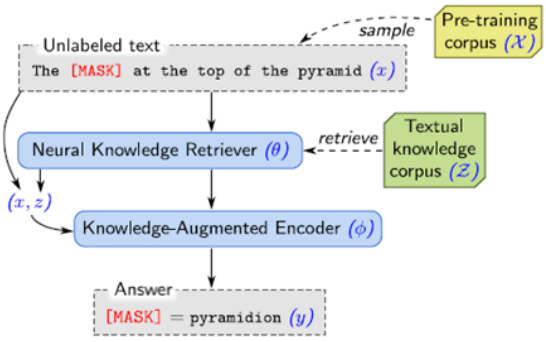

Figure 3: REALM unsupervised pre-training (Guu et al., 2020)

Let D be the training data that consists of m instances. Each instance contains one question q_i, one relevant (positive) passage p_i^+ and n irrelevant (negative) passages p_{ij}^-. Then we optimize the loss function as the negative log likelihood of the positive passage, with sim being the similarity function:

$$D = \left\{ \left\langle q_i, p_i^+, p_{i,1}^-, ., p_{i,n}^- \right\rangle \right\}_{i=1}^{m}$$

$$L(D) = -log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + \sum_{j=1}^{n} e^{sim(q_i, p_{i,j}^-)}}$$

$$sim(q, p) = E_Q(q)^T E_P(p)$$

3.2.2 The generator - Bidirectional and Auto-Regressive Transformer (BART)

The Bidirectional and Auto-Regressive Transformer (BART) (Lewis et al., 2019) is a Transformer that combines the Bidirectional Encoder (BERT) with an Autoregressive decoder (GPT) into one Seq2Seq model. It is a denoising autoencoder that maps a corrupted document to the original document. For pre-training, we optimize the negative log likelihood of the original document.

For BERT, random tokens are replaced with masks and the document is encoded bidirectionally. It learns information from both the left and right side of a token's context during the training. Missing tokens are predicted independently, so BERT cannot be used for generation.

For GPT, the task is to predict the next token. Tokens are predicted auto regressively, meaning that it uses previous information from time steps to predict the output at the current time step, making it ideal for generative tasks. BART is the combination of BERT and GPT.

4 Experiments and Results

4.1 Open-Domain QA Test Scores

For their experiments, in table **??**, they considered four popular open-domain QA datasets: Natural

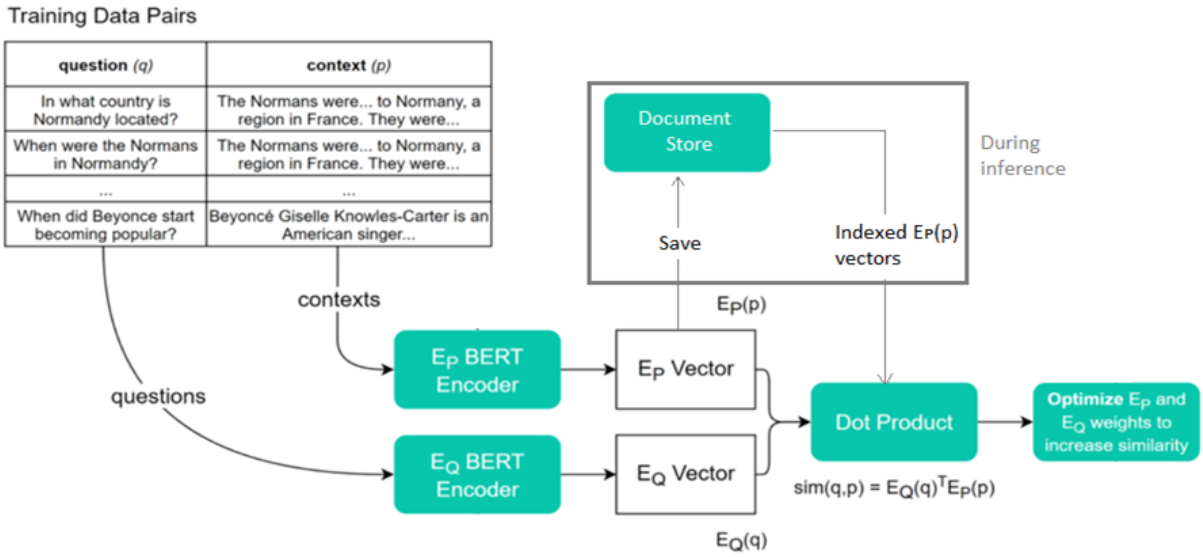

Figure 4: DPR training pipeline (Briggs, 2021)

Questions (NQ), TriviaQA (TQA), WebQuestions (WQ) and CuratedTrec (CT).

Model	NQ	TQA	WQ	CT
REALM	40.4	- / -	40.7	46.8
DPR	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**

Figure 5: Open-Domain QA Test Scores (Lewis et al., 2020)

For TQA, the left column uses the standard test set for Open-Domain QA and the right column uses the TQA-Wiki test set. On all four open-domain QA tasks, RAG sets a new state of the art and unlike REALM, RAG enjoys strong results without expensive, specialized pre-training. We can also observe that the RAG-seq approach performs slightly better, and that DPR on its own is a viable approach. RAG demonstrates that neither a re-ranker nor extractive reader is necessary for a state-of-the-art performance.

4.2 Dataset overlap

During fine-tuning, a problem with common QA datasets has been discovered. There is a significant overlap between questions in the train and test sets in several public QA datasets. Lewis, et al., in a paper published in 2020 found that 58-71 % of test-time answers are also present somewhere in the training sets. In their experiments, several models performed notably worse when duplicated or paraphrased questions were removed from the training set.

5 Summary

In this paper, we started by introducing QA and its different types. Then explained the different approaches to solve QA, starting with the traditional, then evolving towards the modern Retriever-Reader. Different papers were presented. REALM is an end-to-end approach with ab extractive reader. RAG, a generative approach, was defined along its components DPR and BART. We then concluded with results from the RAG paper and critic paper about data sets with no overlap. There are several advantages to generating answers, even when it is possible to extract them. Documents with clues about the answer, but do not contain the exact answer can still contribute towards a correct answer being generated, which is not possible with standard extractive approaches. RAG has shown that generative approaches can compete with the so far dominating extractive approaches, and could build the basis for future papers promoting generative approaches.

References

James Briggs. 2021. How to create an answer from a question with dpr.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep

bidirectional transformers for language understanding.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer.

Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. 2021. Retrieving and reading: A comprehensive survey on open-domain question answering.