Retrieval Augmented Generation: A New Paradigm for Natural Language Generation

A guide for python developers who want to learn about RAG and how to use it in their projects

Introduction

Natural language generation (NLG) is the task of producing natural language text from some input, such as a prompt, a query, a knowledge base, or a set of keywords. NLG has many applications, such as chatbots, summarization, question answering, and content creation. However, NLG also faces many challenges, such as generating fluent, coherent, diverse, and relevant texts that satisfy the user's needs and expectations.

One of the recent advances in NLG is the paradigm of retrieval augmented generation (RAG), which combines the strengths of two existing approaches: pre-trained language models (LMs) and information retrieval (IR). Pre-trained LMs, such as GPT-3, BERT, and T5, are powerful neural networks that can generate fluent and diverse texts from a given prompt, but they often lack factual accuracy and specificity. IR systems, such as Elasticsearch, Lucene, and Solr, can retrieve relevant documents or passages from a large corpus of text, but they cannot generate new texts or adapt to different contexts.

RAG bridges the gap between LMs and IR by using the input prompt to retrieve relevant documents or passages from a large text corpus, and then using them as additional context for the LM to generate the output text. This way, RAG can leverage the factual knowledge and diversity of the text corpus, while also maintaining the fluency and coherence of the LM. RAG can also dynamically adjust the retrieval and generation components based on the input and the output, allowing for more flexible and controllable NLG.

How RAG works

RAG consists of three main components: a retriever, a generator, and a fusion module. The retriever takes the input prompt and queries a text corpus to retrieve the most relevant documents or passages. The generator takes the input prompt and the retrieved documents or passages as context, and generates the output text. The fusion module combines the retrieval and generation components, and decides how to weight and select the best output text.

There are two main variants of RAG, depending on how the retrieval and generation components are integrated: RAG-token and RAG-sequence. RAG-token retrieves one document or passage for each token in the output text, and feeds it to the generator as an additional input. RAG-sequence retrieves a fixed number of documents or passages for the whole output text, and feeds them to the generator as additional inputs. Both variants use a transformer-based LM as the generator, and a dense vector-based IR system as the retriever.

RAG-token and RAG-sequence have different advantages and disadvantages. RAG-token can generate more diverse and specific texts, as it can retrieve different documents or passages for different tokens. However, RAG-token can also generate more noisy and inconsistent texts, as it can retrieve irrelevant or contradictory documents or passages. RAG-sequence can generate more coherent and consistent texts, as it can retrieve a coherent set of documents or passages for the whole output text. However, RAG-sequence can also generate more generic and vague texts, as it can retrieve less relevant or diverse documents or passages.

How to use RAG in python

One of the easiest ways to use RAG in python is to use the Hugging Face Transformers library, which provides pre-trained RAG models and easy-to-use APIs for NLG. The Transformers library supports both RAG-token and RAG-sequence variants, and allows users to customize the text corpus, the retriever, the generator, and the fusion module. The library also provides examples and tutorials for using RAG for different NLG tasks, such as question answering, summarization, and dialogue generation.

To use RAG in python, you need to install the Transformers library and its dependencies, such as PyTorch or TensorFlow, and download the pre-trained RAG models and the text corpus. You can use the following commands to install and download the required packages:

- pip install transformers
- pip install datasets
- pip install faiss-cpu
- python -c "from transformers import RagTokenizer, RagRetriever, RagTokenForGeneration; RagTokenizer.from_pretrained('facebook/rag-token-nq', use_fast=True); RagRetriever.from_pretrained('facebook/rag-token-nq', dataset='wiki_dpr', index_name='compressed'); RagTokenForGeneration.from_pretrained('facebook/rag-token-nq')"
- python -c "from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration; RagTokenizer.from_pretrained('facebook/rag-sequence-nq', use_fast=True); RagRetriever.from_pretrained('facebook/rag-

sequence-nq', dataset='wiki_dpr', index_name='exact');
RagSequenceForGeneration.from_pretrained('facebook/rag-sequence-nq')"

After installing and downloading the required packages, you can use the following code snippet to generate texts using RAG-token:

```
from transformers import RagTokenizer, RagRetriever, RagTokenForGeneration

tokenizer = RagTokenizer.from_pretrained("facebook/rag-token-nq")

retriever = RagRetriever.from_pretrained("facebook/rag-token-nq",
index_name="compressed")

model = RagTokenForGeneration.from_pretrained("facebook/rag-token-nq",
retriever=retriever)

input_prompt = "Who is the founder of Microsoft?" # You can change the input prompt
here

input_ids = tokenizer(input_prompt, return_tensors="pt").input_ids

output_ids = model.generate(input_ids)

output_text = tokenizer.batch_decode(output_ids, skip_special_tokens=True)[0]

print(output_text)
```

The output text should be something like:

```
Bill Gates is the founder of Microsoft, a multinational technology company that
develops, manufactures, licenses, supports, and sells computer software, consumer
electronics, personal computers, and related services. Gates co-founded Microsoft with
Paul Allen in 1975, and led the company as chairman and CEO until 2000, and as
chairman until 2014. He is one of the richest people in the world, and a prominent
philanthropist through his foundation, the Bill & Melinda Gates Foundation.
```

You can use the following code snippet to generate texts using RAG-sequence:

```
from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration

tokenizer = RagTokenizer.from_pretrained("facebook/rag-sequence-nq")

retriever = RagRetriever.from_pretrained("facebook/rag-sequence-nq",
index_name="exact")

model = RagSequenceForGeneration.from_pretrained("facebook/rag-sequence-nq",
retriever=retriever)

input_prompt = "Who is the founder of Microsoft?" # You can change the input prompt
here

input_ids = tokenizer(input_prompt, return_tensors="pt").input_ids
```

```
output_ids = model.generate(input_ids)

output_text = tokenizer.batch_decode(output_ids, skip_special_tokens=True)[0]

print(output_text)
```

The output text should be something like:

```
Microsoft was founded by Bill Gates and Paul Allen on April 4, 1975, to develop and sell BASIC interpreters for the Altair 8800. It rose to dominate the personal computer operating system market with MS-DOS in the mid-1980s, followed by Microsoft Windows. The company's 1986 initial public offering, and subsequent rise in its share price, created three billionaires and an estimated 12,000 millionaires among Microsoft employees. Since the 1990s, it has increasingly diversified from the operating system market and has made a number of corporate acquisitions, the largest of which was the acquisition of LinkedIn for $26.2 billion in December 2016, followed by its acquisition of Skype Technologies for $8.5 billion in May 2011.
```

Conclusion

RAG is a new paradigm for NLG that combines the strengths of pre-trained LMs and IR systems. RAG can generate fluent, coherent, diverse, and relevant texts that leverage the factual knowledge and diversity of a large text corpus. RAG can also dynamically adjust the retrieval and generation components based on the input and the output, allowing for more flexible and controllable NLG. RAG has many potential applications, such as question answering, summarization, dialogue generation, and content creation.

Python developers who want to use RAG in their projects can easily do so with the Hugging Face Transformers library, which provides pre-trained RAG models and easy-to-use APIs for NLG. The Transformers library supports both RAG-token and RAG-sequence variants, and allows users to customize the text corpus, the retriever, the generator, and the fusion module. The library also provides examples and tutorials for using RAG for different NLG tasks.

RAG is an exciting and promising direction for NLG research and development, and we hope that this article has provided a useful introduction and guide for python developers who want to learn more about it and use it in their projects.