Retriever-Generator in practice: studying some interesting models

We introduced the [Retriever-Generator Architecture (LLM_plus_Extra_Parametric.ipynb#Retriever-Generator-Architecture:-high-level-view)](LLM_plus_Extra_Parametric.ipynb#Retriever-Generator-Architecture:-high-level-view) in a previous module.

- factual knowledge about the "world"
- is obtained by an external mechanism
- rather than stored in a model's parameters at training time

The Retriever-Generator model consists of two sub-models working in concert.

We can illustrate this with a model for the Question Answering task.

A single NN (non-retrieval) model directly generates output answer $\mathbf{y}$ given question $\mathbf{x}$, computing

$$p(\mathbf{y}|\mathbf{x})$$

By contrast: the Retriever-Generator model has two separate steps, each executed by a sub-model.

The *Retriever* Neural Network

- takes the questions $\mathbf{x}$
- returns a set $\mathbf{z}$ consisting of the top K items in the Knowledge Store that are most relevant to $\mathbf{x}$

$$\mathbf{z} = \text{Retriever}(\mathbf{x})$$

The *Generator*: a Neural Network

- takes the question $\mathbf{x}$ and relevant items $\mathbf{z}$
- returns $\mathbf{y}$: the text that is the answer to questions $\mathbf{x}$

$$\mathbf{y} = \text{Generator}(\mathbf{x}, \mathbf{z})$$

The model computes

$$p(\mathbf{y}|\mathbf{x}, \text{Retriever}(\mathbf{x}))$$

the output answer conditional on the question and top facts retrieved.

In this module

- we explore each sub-model in greater detail
- explain some interesting new models in terms of the Retriever-Generator architecture.

Retriever: Retrieving World Knowledge

An issue that must be addressed by all models

- How to retrieve world knowledge relevant to a question.

Regardless of how the relevant knowledge is retrieved, the goal is to retrieve the K items most relevant to question $\mathbf{x}$.

- $\mathbf{z}$ is the subset of the knowledge store containing the K items that most closely match question $\mathbf{x}$
- called the *context* (in which the question is answered)

Notation summary

term	dimension	meaning	
$\mathbf{x}$		input sequence: question	
p		knowledge store: set of "documents" (e.g., documents, Web pages)	
		$p = { p^\ip	1 \le i \le P }$
$\mathbf{z}$	$K \times ?$	set of "items" retrieved from p (typical: top K matches)	
		$\z = { \z^\ip	1 \le i \le K }$
		$\mathbf{z} \subset p$	
$\mathbf{y}$	$N \times ?$	target sequence: answer	

A particular challenge of non-parametric World Knowledge is the **length** of documents in the Knoweldge Store

- may exceed the LLM maximum input length
- the document is only *part* of the input
    - question is also part of the input

To account for this the "items" retrieved in $\mathbf{z}$

- may be "chunks" of text (long text divided into pieces)
- "passages": short sequence of text extracted from an item in the Knowledge Store

We will use the term *item* to denote shorter blocks of text, extracted from documents in p.

Static, non-parametric knowledge

This approach uses an external (to the model) *fixed* knowledge store.

- stores documents or sentence fragments

How do we find the K items in the store most relevant to questions $\mathbf{x}$?

We will define the *Retriever* as the mechanism for finding these matches.

Sentence embeddings

A *sentence embedding* is a fixed length representation of a block of text

This is a generalization of *word embeddings*.

Given

- a sentence embedding of question $\mathbf{x}$
- a sentence embedding of each item in the knowledge store

we can use a distance metric (e.g., Dot product) of the two embeddings to find the items most similar to $\mathbf{x}$

Quick survey on constructing sentence embeddings

Assuming we have an embedding for each word in the block of text

- a simple sentence embedding can be the average (across the word embeddings)
    - weakness: does not respect ordering of words

Rather than using embeddings of words in isolation, a better approach is to use [Contextualized Word Representations (NLP_Word_Representations.ipynb#Contextualized-representations)](NLP_Word_Representations.ipynb#Contextualized-representations) of each word

- the representation of a word in *context*
    - of the preceding words
    - of the trailing words

An Encoder Transformer produces a representation of each position of the block that takes the entire block into account.

So passing a sentence of length T ($\mathbf{x}$ or an element of set $\mathbf{z}$) through an Encoder Transformer

- results in a vector of length T contextualized representations

To create a fixed length representation: we need to eliminate the T dimension

- pooling
    - average, max
- The representation of the `<START>` or `<END>` tokens that bracket the block
    - Can be a proxy for the representation of the sentence

Let E be an "Encoder" that produces a sentence embedding.

The distance metric is defined as

$$D(\mathbf{x}, \mathbf{z^{(i)}}) = E(\mathbf{x}) \cdot E(\mathbf{x^{(i)}})$$

Here is a [paper (https://arxiv.org/pdf/1908.10084.pdf)](https://arxiv.org/pdf/1908.10084.pdf) with one type of Sentence Embeddings.

Dense Passage Retrieval (DPR) (https://arxiv.org/pdf/2004.04906.pdf)

The Sentence Embedding is simple but has one potential drawback

- questions $\mathbf{x}$
- are probably different in nature (e.g., length) than the items (e.g., full documents)

So the dot product of sentence embeddings of a short question and a long document may not be ideal

- one relevant passage in a long document containing many irrelevant passages
- may result in a low dot product
- making it similar to the dot product with an irrelevant document

One possible solution is to create a model

- that extracts the relevant passage from a document of multiple passages
- output a *span*
    - start and end position (within item) of relevant passage

One could imagine creating a training set for such a model.

There is another alternative:

The DPR approach is similar to Sentence Embeddings except

- two different Encoders are used
- E_q for questions, E_p for items
- both are Neural Networks
    - e.g., fine-tuned LLM's

The resulting distance metric becomes

$$D(\mathbf{x}, \mathbf{z^{(i)}}) = E_q(\mathbf{x}) \cdot E_p(\mathbf{x^{(i)}})$$

We jointly train E_q and E_p to create embeddings with high dot products when $\mathbf{z^{(i)}}$ is relevant.

Maximizing the dot product

The Retrievers we have define thus far are essentially Multinomial Classifiers over P discrete items

- producing a vector of probabilities (e.g., softmax of the dot products of $\mathbf{x}$ and elements of p)

If $p^{(\mathbf{i})}$ is relevant to questions $\mathbf{x}$, we want

- it's probability to be high
- the probability of $p^{(i')}$ to be low, where $i' \neq i$

This is called a *Contrastive Objective*

- creating a contrast (in magnitude of probability) between matches and non-matches

A *triplet objective* can be defined

- for question $\mathbf{x}$
- matching passage $p^{(\mathbf{i})}$
- non-matching passage $p^{(i')}$

as

$$
\begin{array}{lll}
\max(0, C) & & \text{where} \\
C = & & \\
 & D\left(E_q(\mathbf{x}), E_p(p^{(\mathbf{i})})\right) & \text{Distance between } \mathbf{x} \text{ and match } p^{(\mathbf{i})} \\
 & - & \\
 & D\left(E_q(\mathbf{x}), E_p(p^{(i')})\right), & \text{Distance between } \mathbf{x} \text{ and non-match } p^{(i')}
\end{array}
$$

The triplet objective is *minimized* when

- $E_q(\mathbf{x})$ and $E_p(p^{(\mathbf{i})})$ are close
- $E_q(\mathbf{x})$ and $E_p(p^{(i')})$ are far

A training example for a NN using this objective would be

$$\langle \mathbf{x}, p^{+}, p^{-,1} \dots p^{-,n'} \rangle$$

- a questions $\mathbf{x}$
- a positive (matching) passage p^{+}
- n' negative (non-matching)passages $p^{-,1} \dots p^{-,n'}$

We convert dot products into probabilities via the softmax

$$\frac{\exp(D(E_q(\mathbf{x}), E_p(p^{+})))}{\exp(D(E_q(\mathbf{x}), E_p(p^{+}))) + \sum_{i'=1}^{n'} \exp\big(D(E_q(\mathbf{x}), E_p(p^{-,i'}))\big)}$$

and use Cross Entropy Loss.

In-batch negatives trick (https://arxiv.org/pdf/2004.04906.pdf#page=3)

The choice of negative passages $p^{-,1} \ldots p^{-,n'}$

- is arbitrary
- labor-intensive

One can get away with examples that provide **no explicit** negative passages with the following trick.

When using mini-batch gradient descent, there are B examples per mini-batch

Consider example i in the batch

- with questions $\mathbf{x}^{(\mathbf{i})}$
- Let $p^{(\mathbf{i}),+}$ denote the positive passage for $\mathbf{x}^{(\mathbf{i})}$
- Use $p^{(i'),+}$
    - the positive passage for example $i' \neq i$
    - as a negative examples for $\mathbf{x}^{(\mathbf{i})}$

Thus, we the negative passages for an example are implicitly obtained from other examples in the batch.

This can also be computationally efficient

- Can compute the dot products as one big matrix multiplication

n.b., this In-batch negative trick was also used in CLIP (CLIP.ipynb#Pseudo-code-for-Pre-Training)

Dynamic, non-parametric knowledge

This approach uses

- a constantly updated source of knowledge, e.g., the Web.
- to create the set $\mathbf{z}$ of items in the Knowledge Store most relevant to question $\mathbf{x}$

The idea is to train the Retriever NN

- to create a query to a Search Engine (e.g., Bing)
- Scroll through top results
- Visit a result
    - issue a search for a text string within the result
    - select a neighborhood of the result around the search
    - adding the neighborhood as a 'reference" (element of set $\mathbf{z}$)

We will describe this Web Retriever more in depth in the discussion of WebGPT

Generator: Generating the answers

The role of the Generator is to create the answer $\mathbf{y}$, conditioned on

- question $\mathbf{x}$
- the set of relevant passages $\mathbf{z}$

$$p(\mathbf{y}|\mathbf{x}, \text{Retriever}(\mathbf{x}))$$

We have already seen how LLM's can generate answers.

[RETRO (Deepmind) (https://download.arxiv.org/pdf/2112.04426v3.pdf)](https://download.arxiv.org/pdf/2112.04426v3.pdf)

RETRO stands for the **R**etrival **E**nhanced **TR**ansf**O**rmer

By using a non-parametric Knowledge Store,

- RETRO is able to match GPT-3 performance on some benchmarks
- using only 4% (i.e., 7.5B vs 175B) of the number of parameters.

Model summary: high-level (approximate)

Retriever

- **static** non-parametric knowledge
    - 2 trillion tokens.
- uses similarity of **sentence embeddings** of query and items in Knowledge Store to find relevant items

Generator

- Encoder/Decoder Transformer
    - Encoder
        - $\mathbf{z}$ (items returned by Retriever as relevant to $\mathbf{x}$) passed through Encoder
        - Latent states of Encoder become Keys and Values for Attention
    - Decoder
        - Attention to input (partially built $\mathbf{y}$)
        - Cross Attention (Decoder-Encoder Attention)
            - to items returned by Retriever

Mode of operation of RETRO vs standard Transformer

In a non-retrieval Transformer (parametric knowledge) with a Language Modeling objective:

- usually a [Decoder style Transformer (https://arxiv.org/pdf/1801.10198.pdf#page=4)](https://arxiv.org/pdf/1801.10198.pdf#page=4)
    - auto-regressively extends partial output one token at a time
        - on iteration t: generates $\mathbf{y}_t$
        - feeds $y_{(1:t)}$ back as input for iteration $t+1$
        - question $\mathbf{x}$ is a prefix of $\mathbf{y}$
        $$\mathbf{y}' = \text{concat}(\mathbf{x}, \mathbf{y})$$
    - no cross-attention to the Encoder (because no Encoder)
        - just self-attention to incrementally generated $\mathbf{y}$

In a retrieval Transformer (non-parametric knowledge)

- Encoder-Decoder style Transformer
- at inference time
    - question $\mathbf{x}$ is sent to Retriever
        - returns K relevant items
        - the K relevant items, and $\mathbf{x}$, are input to the Encoder
- Once the Encoder finishes, the Decoder operates auto-regressively
    - just as above
    - but with Cross-Attention to Encoder output (retrieved knowledge)

Notation

Variable	Definition

n | maximum text length | | example: $n = 2048$ m | chunk of text length | | text of length n broken up into $l = \frac{n}{m}$ chunks of length m | | example: $m = 64$ $\mathcal{D}$ | Knowledge store | | Collection of items | | implemented as key/value pairs | | Item i has key N_i and value F_i | | N_i is a chunk of text; F_i is the following chunk of text | T | number of items in $\mathcal{D}$ | | $T = 2 * 10^{12}$ r | length of returned item | | $r = 2 * m = 128$ k | number of similar items returned | | size of set $\mathbf{z}$ | | example: $k = 40$ $\text{Ret}(C)$ | set $\mathbf{z}$ of returned itemes | | dimension $(k \times r)$ | $\mathbf{y}$ | The partially built output sequence | | - starts with question $\mathbf{x}$ | | - is extended by the Decoder

Model details: chunked data

The model works by breaking long text into *chunks* of length m.

Thus, the items in the Knowledge store are chunks ("passages" as we previously called them) not full documents.

Similarly, the question $\mathbf{x}$ and partially generated answer $\mathbf{y}$ are also broken up into chunks.

Retriever

The Knowledge Store is implemented as Key/Value Pairs

- the element i
  - key N^i is a chunk
  - value F^i is the chunk that immediately follows N_i

Each key (and query against the Key/Value pairs)

- is encoded by a BERT transformer (with averaging over "time" == tokens).

Lookup with with query q returns k items

- each item is $[N^i, F^i]$ where distance between query C and key N is among the k smallest
  - $d(C, N) = ||\text{BERT}(C) - \text{BERT}(N)||_2^2$
  - $\text{Ret}(C) = \left([N^1, F^1], \ldots, [N^k, F^k]\right)$
  - length of item is $r = 2 * m$

Generator

Just like a Standard LM with a "predict the next" token objective

- the generator *autoregressively* generates next output token $\mathbf{y_{(i)}}$
    - conditioned on all previous output tokens $\mathbf{y}_{(1:i-1)}$

In addition

- $\mathbf{y}_{(1:t-1)}$ is broken into $l = \frac{t-1}{m}$ chunks of length m
$$\mathbf{y}_{(1:t-1)} = C_1, \ldots, C_l$$
- a set of k items is retrieved for each chunk C_j
$$\text{Ret}(C_i)$$

So the generator is *also* conditioned on
$$\text{Ret}(C_i), \ldots, \text{Ret}(C_l)$$

The Generator maximizes the likelihood of the next output $\mathbf{y}_t$ conditioned on

- previously generated partial $\mathbf{y}_{(1:t-1)}$
- and items retrieved from the chunks accessible to $\mathbf{y}_{(1:t-1)}$
    - let u denote the index of the last chunk accessible to $\mathbf{y}_{(1:t-1)}$

$$p(\mathbf{y}_i \mid \mathbf{y}_{(1:i-1)}, \{\text{Ret}(C_{u'}) \mid u' < u\})$$

Fine point about "chunks accessible to $\mathbf{y}_{(1:t-1)}$"

- if $\mathbf{y}_{(t-1)}$ is not the *last* item in the chunk
- then the chunk containing $\mathbf{y}_{(t-1)}$ includes $\mathbf{y}_{t'}$ for $t' > t-1$
- can't use a chunk if it includes $\mathbf{y}_{t'}$ for $t' > t-1$ because it violates causality

The paper expresses this

- as log-likelihoods $\mathbb{L}$
    - log so we can use sum rather than product
- indexes elements of $\mathbf{y}$ using the chunk number u and offset j within chunk

$$i = (u-1) * m + j$$

$$\mathbb{L}\left(\mathbf{y}_i \mid \mathbf{y}_{(1:i-1)}, \{\text{Ret}(C_{u'}) \mid u' < u\}\right) \quad = \quad \sum_{u=1}^{l} \sum_{j=1}^{m} \mathbb{L}\left(\mathbf{y}_{((u-1)*m+j)} \mid \mathbf{y}_{(1:(u-1)}\right.$$

RETRO-fitting existing models

The authors have had success adapting non-retrieval models to use RETRO retrieval

- freeze weights of non-retrieval model
- train only
    - Chunked Cross Attention
    - Neighbor Encoder
    - this training is on dataset that is only 3% as big as the full training set

WebGPT (OpenAI) (https://openai.com/blog/webgpt)

paper (https://arxiv.org/abs/2112.09332)

WebGPT uses the Web as its source of World Knowledge.

In order to better be able to evaluate the truthfulness of answers

- specific passages (called *references*) are extracted from Web pages
- answer uses the references as support

Model summary: high-level (approximate)

Retriever

- **dynamic** non-parametric knowledge
    - constantly changing Web

Generator

- GPT style LLM (i.e., Decoder only)
- Takes question $\mathbf{x}$ and supporting references $\mathbf{z}$ returned by Retriever
    - answer extends the input

Model details

The novelty is how the Retriever is able to "browse the Web".

Basically

- a human demonstrates
    - how to use a browser to search the Web
    - in order to gather references $\mathbf{z}$ that are relevant for answering question $\mathbf{x}$
- the model learns how to imitate the human's behavior.

This technique is called *Behavioral Cloning*.

The Retriever is trained with examples that are human-created *demonstrations* of behavior.

The human's behavior (sequence of actions) is recorded as the human interacts with a Browser.

- Issue a query to the browser
- Extract relevant *references* (passages) from the query results
- End the Web search and move to generating an answer using the collected references

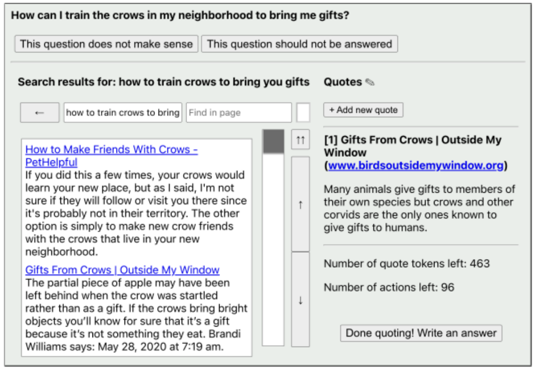
Here is a example of the behavioral actions from [WebGPT (https://openai.com/blog/webgpt)](https://openai.com/blog/webgpt)

Command	Effect

Search `<query>` | Send `<query>` to Bing API Click on link `<link ID>` | Follow the link Find in page `<text>` | Find next occurrence of `<text>` and scroll to it Quote: `<text>` | If `Find` successful: add it as a reference Scroll down | Navigate through page Scroll up | Top | Back | End: Answer | End browsing and move to answering phase End: `<Nonsense, Controversial>` | End browsing and skip answering phase

The behavior is encoded as text and, along with the question, forms an example.

Here is the interface (left) used by a human to record behavior and the encoded behavior (right)



(a) Screenshot from the demonstration interface.

```
♦Question
How can I train the crows in my neighborhood to bring me gifts?

♦Quotes
From Gifts From Crows | Outside My Window (www.birdsoutsidemywindow.org)
> Many animals give gifts to members of their own species but crows and
other corvids are the only ones known to give gifts to humans.

♦Past actions
Search how to train crows to bring you gifts
Click Gifts From Crows | Outside My Window www.birdsoutsidemywindow.org
Quote
Back

♦Title
Search results for: how to train crows to bring you gifts

♦Scrollbar: 0 - 11
♦Text
【0†How to Make Friends With Crows - PetHelpful†pethelpful.com】
If you did this a few times, your crows would learn your new place, but
as I said, I'm not sure if they will follow or visit you there since it's
probably not in their territory. The other option is simply to make new
crow friends with the crows that live in your new neighborhood.

【1†Gifts From Crows | Outside My Window†www.birdsoutsidemywindow.org】
The partial piece of apple may have been left behind when the crow was
startled rather than as a gift. If the crows bring bright objects you'll
know for sure that it's a gift because it's not something they eat.
Brandi Williams says: May 28, 2020 at 7:19 am.

♦Actions left: 96
♦Next action
```

(b) Corresponding text given to the model.

Encoded fields:

- Question `how can I train the crows in my neighborhood to bring me gifts ?`
- Past actions
    - Web query: `Search "how to train crows to bring you gifts"`
- Text
    - results returned by Web query
- Next action
    - prompt for LLM to complete, by predicting next action

The Retriever is trained with examples that are a prefix of a full demonstration

- learns how to extend the behavior with a new action
    - "predict the next" action

Since a machine can't "see" the screen

- the browser context and state is recorded as a written summary of the environment

Similarly, it can't "remember" the past actions

- these too are recorded as text

Using this textual encoding of the history of actions, the LLM tries to extend the behavior via a new action.

Generator

The Generator is trained to create answers

- that cite the references

Because the references are text (just like the questions)

- there is no preprocessing of the retrieved items necessary
    - compare to RETRO which needs to process retrieved passages through the Encoder
        - in order to facilitate Decoder-Encoder cross-attention

Questions

There are some questions as to the exact details

- what is the syntax of the prompt ?
    - identifying the parts: passage, question
- how many passages are used ?

Most significant: is the task **zero**-shot or **few**-shot

- zero-shot: prompt consists only of the particular question q (and passages)

  ```
  **Passage**:  <...>
  **Question**: <...>
  **Answer**:
  ```

  with the LLM expected to extend the text beyond the final `**Answer**:`
- few-shot:
  - the particular question and passage is preceded by $k > 0$ run-time examples
  - suggesting the task is to complete the answer based on references
  - e.g., here is an illustration of one of the k examples

    ```
    **Passage**:  <...>
    **Question**: <...>
    **Answer**:   <..>
    ```

    - suggesting the goal of completing the Answer based on references

Although I can't find the precise details

- the next section on Internet Augmented Language models is similar to WebGPT
- with more detail

so is suggestive of the details for WebGPT.

Other model characteristics (out of scope for this module)

There is a lot more to WebGPT than just the use of a dynamic, non-parametric knowledge store.

These are beyond the scope of the present topic, but we briefly describe some interesting characteristics/

Reward Model

There is a desire to produce answers that are helpful, truthful, non-harmful and high quality.

None of these are explicit objectives of a LLM.

The authors fine-tune the LLM towards this end.

The idea is to train a *reward model* to predict which of two answers is "better".

Given the reward model, the authors use *Reinforcement Learning with Human Feedback* to fine-tune the LLM in the direction of producing better answers.

The Human Feedback comes from

- having the initial LLM generate multiple answers to a question
- having a human rank the answers

A question and two ranked answers become an example used

- to train a Classifier
- to predict which answer is better.

This is called the *Reward Model*

Here is the interface with which a Human evaluates multiple answers

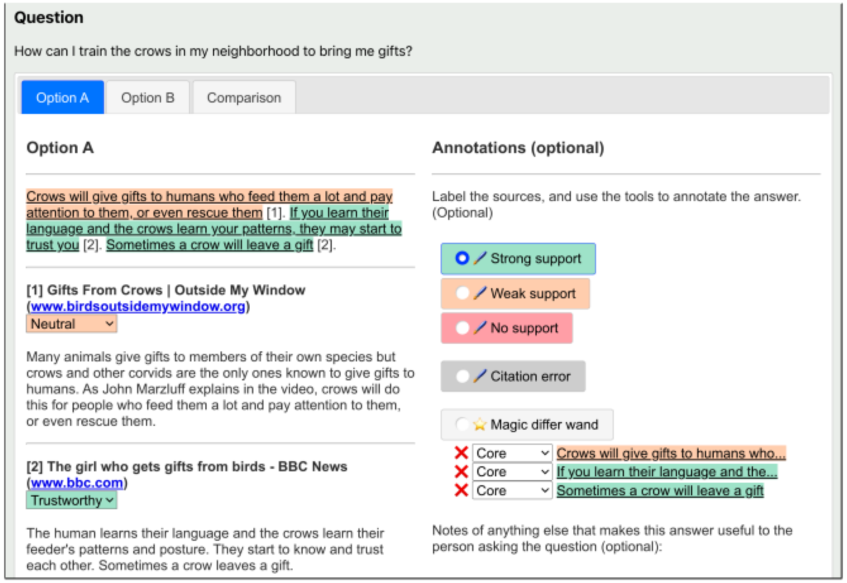


Figure 9: Screenshot from the comparison interface, showing the annotation tool.

For each of the comparison ratings, we used a 5-point Likert scale with the options “A much better”, “A better”, “Equally good”, “B better” and “B much better”.

Reinforcement Learning to fine-tune the LLM

In the improvement phase, the initial LLM

- generates an answer
- the answer's quality is predicted by the Reward Model

The LLM parameters are adjusted by *Reinforcement Learning*

-
    - parameters are adjusted so as to increase Reward

Rejection sampling

The model is asked to produce several answers

- each is evaluated by the Reward model
- the answer with highest reward is selected

Rejection sampling can be used either on

- the initial LLM (before Reinforcement Learning)
- after Reinforcement Learning

Internet Augmented Language Models (DeepMind) (https://arxiv.org/pdf/2203.05115.pdf)

This model is similar to WebGPT

- but developed by Google
- uses Google search rather than Bing

There is more detail in this paper than the one for WebGPT

- perhaps the answer to the open questions we had for WebGPT have similar answers to what we find here

Retriever

- Google as the search Engine
- Question q passed *verbatim* (unchanged) to the Search Engine
- Search Engine returns the URL of the top 20 results
- The results are converted to text and broken into paragraphs of 6 sentences
- The similarity of the retrieved paragraphs is compared to q
- The top 50 paragraphs form the set $\mathbf{z}$

We contrast the query used to that of WebGPT

- Google: unchanged from question q
- WebGPT: "learning to query" approach
    - Model trained to search the Web

A justification given for using the original question q as query

Apparently:

- most search engines perform some type of query transformation to improve user experience
- hidden from user

We can also compare how relevant passages are extracted

- Google: chunks, ranked by similarity to question
- WebGPT: learns to extract passages (via demonstrations)

Generator

k-shot Prompting ($k = 15$)

The LLM has not been trained for the particular task of Question Answering.

Thus, it needs to be *conditioned* on this task by being shown k examples of question/evidence/answer.

The format of each prototype example is

```
**Evidence**: <...>
**Question**: <...>
**Answers**:  <...>
```

The particular questions q is appended to the k prototype examples

- but without anything following `**Answer**:`
- the LLM will provide the answer by extending the prompt

Number of passages/Number of answers

Only a *single* passage is used as evidence at a time

- the model creates *multiple* answers from question q and *each* passage
- $a_{i,j}$ denotes answer j based on the evidence z_i: element i of $\mathbf{z}$

There are $50 * 4$ answers to each question q.

There is a *scoring function* that ranks each of the answers.

The answer with the highest score is chosen as the final answer

$$\mathbf{y} = \max_{a_{i,j}} f(q, z_i, j)$$

The authors experiment with different scoring functions.

The simplest: the answer with highest model probability

- recall:
  - LLM generates output one token at a time
  - Each token is drawn from a probability distribution
  - Can thus derive probability of $\mathbf{y}$ by

In [2]:

```
print("Done")
```

```
Done
```