CS310 Natural Language Processing
自然语言处理
Lecture 12 - Question Answering and Group Project Discussion

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

Overview

- **Question Answering (QA)**
  - What is QA?
  - Information Retrieval; Tf-idf
  - Retriever-based QA; Datasets
    - Answer Span Extraction
    - Retrieval-Augmented Generation
- Project Discussion

What is Question Answering?

- To build a system that **automatically** answer questions posed by human in natural language

"The Ultimate Question Of Life, The Universe, and Everything" →  → "42"

(from movie *Hitchhiker's Guide to the Galaxy*)

QA System focuses on *factoid* questions

- **factoid questions**: Questions that can be answered with simple facts expressed in short texts
- Ex. 1: Where is the Louvre Museum located?
- Ex. 2: What is the distance from Moon to Earth?

- One way: to directly ask a large language model (LLM)
  - Using prompts: “Q: What is the distance from Moon to Earth? A: ”
- **Problems**:
- LLMs hallucinate; not calibrated
- No access to proprietary/private/personal data: email, private documents, …

Current Solution to QA

- Two-stage **retriever/reader** model
- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents
- Stage 2. Reader algorithms: Either **extract** or **generate** an answer

Brief Overview of Information Retrieval (IR)

- **Information retrieval, IR**: Retrieval of all kinds of media based on user information needs. IR system ≈ **search engine**
- We focus on **ad hoc (临时) retrieval**: a user poses a query to an IR system, which then returns an ordered set of documents from some collection

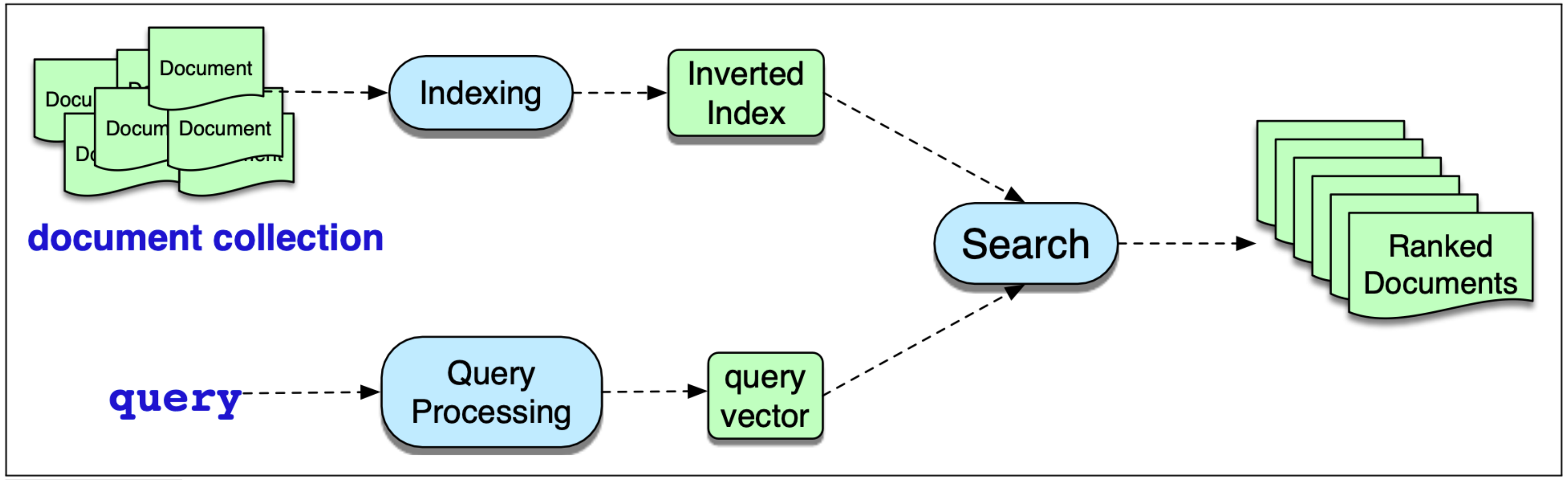


Figure 14.1 The architecture of an ad hoc IR system.

Query: a user's information need expressed as a set of **terms**

Term refers to a word/phrase in a collection of documents

Figure from SLP3, Ch 14

How to match a document a query?

- Compute a term weight for each document term
- Common method: **tf-idf** and BM25
  - **tf**: term frequency
  - **idf**: inverse document frequency
- tf-idf $\triangleq$ tf $\times$ idf   (product of two)

term t; document d

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **tf**: words that occur more often in a document are likely to be informative about the document's content
- Use the $\log_{10}$ of word frequency count rather than raw count
- Why? A word appearing 100 times doesn't make it 100 times more likely

Tf-idf

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

term t; document d

term occurs 0 times in document: tf = 0
term occurs 1 times in document: tf = 1
term occurs 10 times in document: tf = 2, …

- **document frequency** df_t of a term t is the number of documents it occurs in
- Terms that occur in only **a few** documents are useful for discriminating those documents from the rest of the collection;
- terms that occur across the entire collection aren't as helpful (*the, a, an, …*)
- **inverse document frequency** or **idf** is defined as:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

N: total number of documents
The fewer documents in which t occurs, the higher idf_t

Inverse document frequency example

- Some idf values for some words in the corpus of Shakespeare plays

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Extremely informative words that occur in only one play like *Romeo*

good or *sweet* tare completely non-discriminative since they occur in all 37 plays

Scoring with tf-idf

- We can score document d by the cosine of its vector $\vec{\boldsymbol{d}}$ with the query vector $\vec{\boldsymbol{q}}$:

$$\mathrm{score}(q,d) = \cos\left(\vec{\boldsymbol{q}}, \vec{\boldsymbol{d}}\right) = \frac{\vec{\boldsymbol{q}} \cdot \vec{\boldsymbol{d}}}{|\vec{\boldsymbol{q}}| \cdot |\vec{\boldsymbol{d}}|}$$

- in which $\vec{\boldsymbol{q}}$ and $\vec{\boldsymbol{d}}$ are vectors of query length n, whose values are the **tf-idf** values (normalized):

$$\vec{\boldsymbol{q}} = \frac{[\mathrm{tf}-\mathrm{idf}(t_1,q),\dots,\mathrm{tf}-\mathrm{idf}(t_n,q)]}{\sqrt{\sum_{t\in q}\mathrm{tf}-\mathrm{idf}^2(t,q)}}$$

$$\vec{\boldsymbol{d}} = \frac{[\mathrm{tf}-\mathrm{idf}(t_1,d),\dots,\mathrm{tf}-\mathrm{idf}(t_n,d)]}{\sqrt{\sum_{t\in d}\mathrm{tf}-\mathrm{idf}^2(t,d)}}$$

$$\Rightarrow$$

$$\mathrm{score}(q,d) = \sum_{t_i\in q}\frac{\mathrm{tf}-\mathrm{idf}(t_i,q)}{\sqrt{\sum_{t\in q}\mathrm{tf}-\mathrm{idf}^2(t,q)}}\cdot\frac{\mathrm{tf}-\mathrm{idf}(t_i,d)}{\sqrt{\sum_{t\in q}\mathrm{tf}-\mathrm{idf}^2(t,q)}}$$

Tf-idf scoring example

- A collection of 4 nano documents

Query:	sweet love
Doc 1:	Sweet sweet nurse! Love?
Doc 2:	Sweet sorrow
Doc 3:	How sweet is love?
Doc 4:	Nurse!

Using a variant of tf-idf cosine score, by dropping the idf term for the document (for better perf.)

$$\text{score}(q,d) = \sum_{t\in\mathbf{q}} \frac{\text{tf}_{t,q}\cdot\text{idf}_t}{\sqrt{\sum_{q_i\in q}\text{tf-idf}^2(q_i,q)}} \cdot \frac{\text{tf}_{t,d}\cdot\text{idf}_t}{\sqrt{\sum_{d_i\in d}\text{tf-idf}^2(d_i,d)}}$$

Query vector $\vec{\boldsymbol{q}} = (0.383, 0.924)$

				Query		
word	**cnt**	**tf**	**df**	**idf**	**tf-idf**	**n'lized** = tf-idf/$\lvert q\rvert$
sweet	1	1	3	0.125	0.125	0.383
nurse	0	0	2	0.301	0	0
love	1	1	2	0.301	0.301	0.924
how	0	0	1	0.602	0	0
sorrow	0	0	1	0.602	0	0
is	0	0	1	0.602	0	0
$\lvert q\rvert = \sqrt{.125^2 + .301^2} = .326$						

Tf-idf scoring example

Query vector $\vec{q} = (0.383, 0.924)$

Document 1					
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**× q**
sweet	2	1.301	0.163	0.357	**0.137**
nurse	1	1.000	0.301	0.661	0
love	1	1.000	0.301	0.661	**0.610**
how	0	0	0	0	0
sorrow	0	0	0	0	0
is	0	0	0	0	0

$|d_1| = \sqrt{.163^2 + .301^2 + .301^2} = .456$

Document 2					
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**×q**
sweet	1	1.000	0.125	0.203	**0.0779**
nurse	0	0	0	0	0
love	0	0	0	0	0
how	0	0	0	0	0
sorrow	1	1.000	0.602	0.979	0
is	0	0	0	0	0

$|d_2| = \sqrt{.125^2 + .602^2} = .615$

$\vec{d}_1 = (0.357, 0.661)$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.747}$

$\vec{d}_2 = (0.203)$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.0779}$

Therefore, d_1 is more relevant

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Efficient Implementation: Inverted Index

- The basic search problem in IR is to find all documents $d \in C$ that contain a term $q \in Q$
- Use the data structure **inverted index**: given a query term, returns a list of documents that contain the term
- Contains two parts: *dictionary* and *postings*

dictionary: a list of terms, each pointing to a postings list for the term (including document frequency)

how $\{1\}$	$\rightarrow$	$3\ [1]$
is $\{1\}$	$\rightarrow$	$3\ [1]$
love $\{2\}$	$\rightarrow$	$1\ [1] \rightarrow 3\ [1]$
nurse $\{2\}$	$\rightarrow$	$1\ [1] \rightarrow 4\ [1]$
sorry $\{1\}$	$\rightarrow$	$2\ [1]$
sweet $\{3\}$	$\rightarrow$	$1\ [2] \rightarrow 2\ [1] \rightarrow 3\ [1]$

posting list: a list of document IDs associated with each term (including term frequency etc.)

IR with Dense Vectors

- **Flaws of TF-IDF -- Vocabulary mismatch problem**: it only works if there is exact overlap of words between the query and document
- **Solution**: Using dense vectors to represent queries/documents
  dating back to Latent semantic indexing vectors, all the way to modern times via encoders like BERT

Present both query q and document d to a single encoder, allowing self-attention to see all tokens from both q and d

$$\boldsymbol{h} = \mathrm{BERT}(q; [\mathrm{SEP}]; d)\ \text{[CLS]}$$

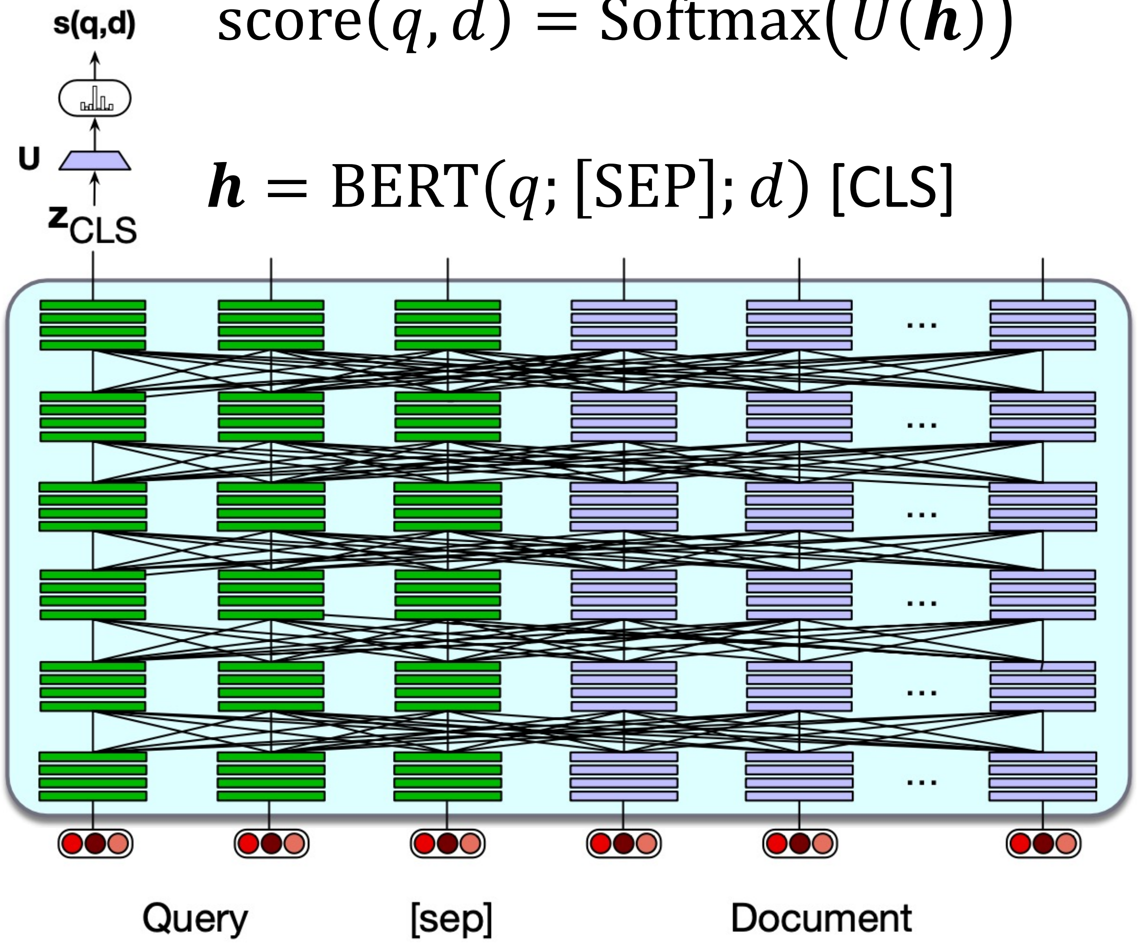
$$\mathrm{score}(q, d) = \mathrm{Softmax}\big(U(\boldsymbol{h})\big)$$

Predict the similarity score between q and d

Single BERT Encoder for IR

$$\text{score}(q, d) = \text{Softmax}\big(U(\boldsymbol{h})\big)$$

$$\boldsymbol{h} = \text{BERT}(q; [\text{SEP}]; d)\ [\text{CLS}]$$



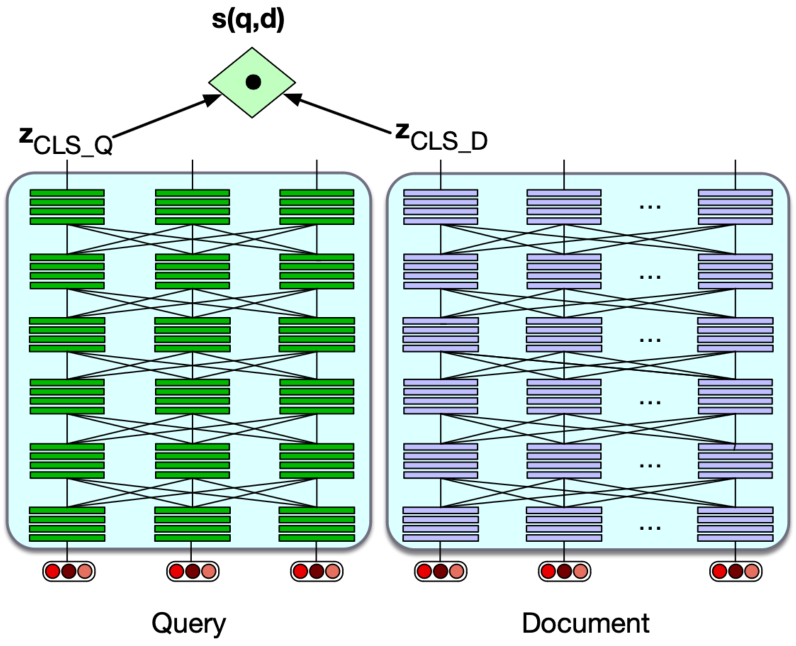
In practice, documents are broken up into smaller passages such as non-overlapping fixed-length chunks of ~100 tokens,

so that the q and d and can fit in the BERT 512-token window

Drawback: expense in computation!
Every time we get a query, have to pass every single document through a BERT encoder jointly with the new query!

More Efficient Way: Bi-Encoder

- Two separate encoder models:
  one to encode the query BERT_Q, and one to encode the document, BERT_D
- Encode each document and store the document vectors in advance
- When a query comes in, just encode this query, and compute the dot product between it and each candidate document



$$\boldsymbol{h}_q = \mathrm{BERT}_Q(q)\ \text{[CLS]}$$

$$\boldsymbol{h}_d = \mathrm{BERT}_D(d)\ \text{[CLS]}$$

$$\mathrm{score}(q,d) = \boldsymbol{h}_q \cdot \boldsymbol{h}_d$$

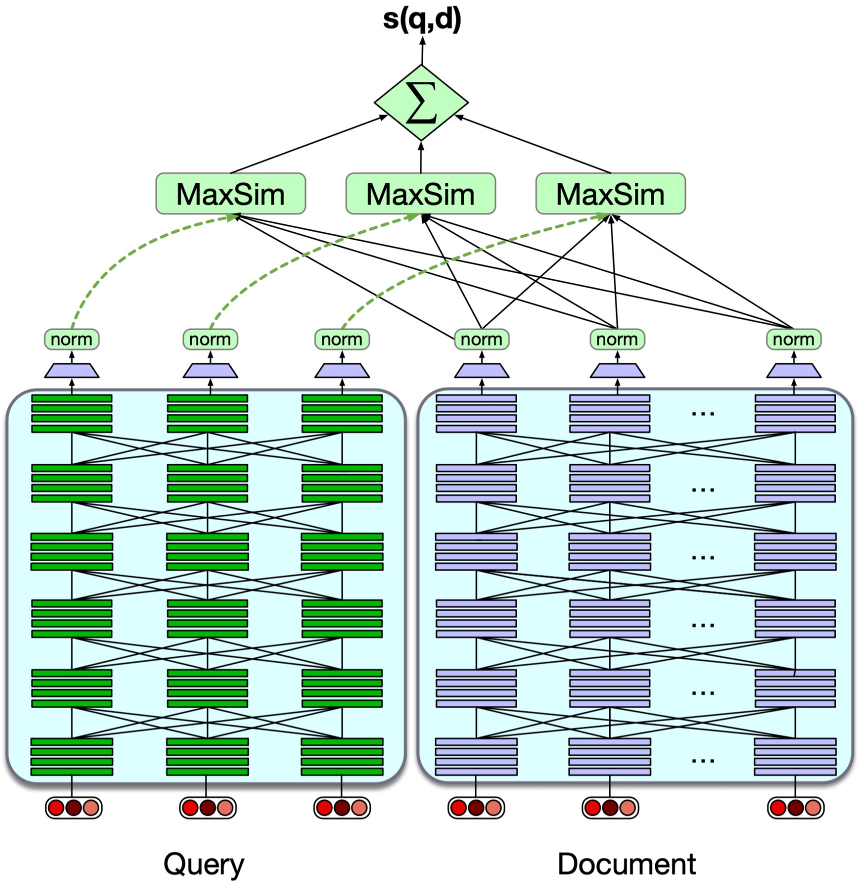
Cheaper in computation, but less accurate, since it does not take full advantage of the interaction between query tokens and document tokens

Alternative: Token-level similarity scores

- **ColBERT** (Khattab et al., 2021) computes the score between a query q and a document d as a sum of maximum similarity (MaxSim) between tokens in q and tokens in d

$$\text{score}(q,d) = \sum_{i=1}^{N} \max_{j=1}^{m} \mathbf{E}_{q_i} \cdot \mathbf{E}_{d_j}$$

More accurate than the bi-encoder method



Implementation Efficiency

- For dense vector-based IR, efficiency is also an important issue
- since every possible document must be ranked for its similarity to the query
- Bottle-neck: Finding the set of document vectors that have the highest dot product with a query vector -- **nearest neighbor search** problem
- Can be approximated with algorithms like Faiss (Johnson et al., 2017)

Current Solution to QA

- Two-stage **retriever/reader** model

- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents
- Stage 2. Reader algorithms: Either extract or generate an answer

Reader

- Extractor: **span extraction** ⇒ find spans of text that answer the question over the retrieved passages

- Generator: **retrieval-augmented generation** ⇒ Take a large pretrained LM, design the prompt based on the retrieved passage, and generate the answer token by token

Two-staged retriever/reader model

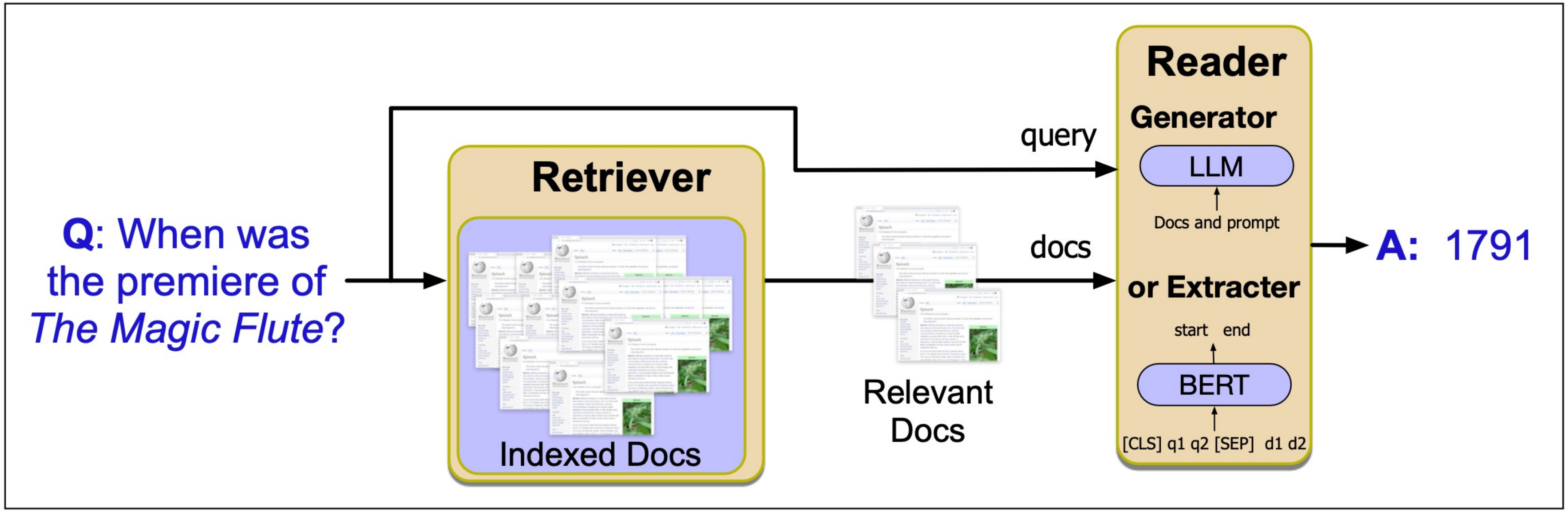


Figure from SLP3, Ch 14

Reader: Answer Span Extraction

Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
Q: "In what city and state did Beyoncé grow up?" A: "Houston, Texas"
Q: "What areas did Beyoncé compete in when she was growing up?" A: "singing and dancing"
Q: "When did Beyoncé release Dangerously in Love?" A: "2003"

Figure 14.11 A (Wikipedia) passage from the SQuAD 2.0 dataset (Rajpurkar et al., 2018) with 3 sample questions and the labeled answer spans.

Span labeling task: identify in the passage a span (continuous string of text) that constitutes an answer

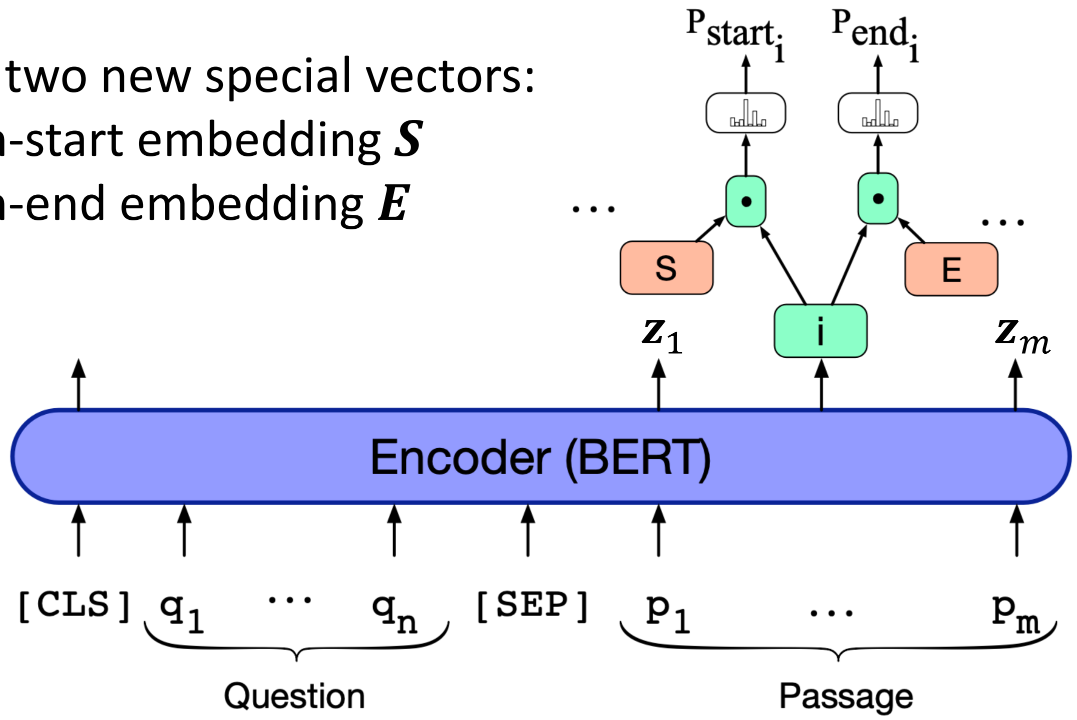
Span Labeling

- Given a question q of n tokens $q_1, \ldots, q_n$ and a passage p of m tokens $p_1, \ldots, p_m$
- **Goal**: compute the probability $P(a|q,p)$ of each possible span a is the answer
- Span a starts at position a_s and ends at a_e, then estimate the probability by:
- $P(a|q,p) = P_{\text{start}}(a_s|q,p)P_{\text{end}}(a_e|q,p)$
- For each token p_i in passage, compute two probabilities:
  - $P_{\text{start}}(i) \Rightarrow p_i$ is the start of answer span
  - $P_{\text{end}}(i) \Rightarrow p_i$ is the end of answer span
- Goal becomes:

$$\max_{i,j \in [1,m]; j \geq i} P_{\text{start}}(i) P_{\text{end}}(j)$$

Span Labeling

Add two new special vectors:
span-start embedding $\boldsymbol{S}$
span-end embedding $\boldsymbol{E}$



$$P_{\text{start}}(i) = \frac{\exp(\boldsymbol{S} \cdot \boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{S} \cdot \boldsymbol{z}_j)}$$

$$P_{\text{end}}(i) = \frac{\exp(\boldsymbol{E} \cdot \boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{E} \cdot \boldsymbol{z}_j)}$$

Goal: $\max_{i,j\in[1,m];j\geq i} P_{\text{start}}(i) P_{\text{end}}(j)$

$$\max_{i,j\in[1,m];j\geq i} \boldsymbol{S} \cdot \boldsymbol{z_i} + \boldsymbol{E} \cdot \boldsymbol{z_j}$$

The score for candidate span from position i to j

What if answer is not contained in passage?

- Many datasets contain (question, passage) pairs in which the answer is not contained in the passage
- Need a way to estimate this “none” probability
- Done by treating [CLS] token as the answer, i.e., a_s and a_e all point to [CLS]

Retrieval-based QA Datasets

- Reading comprehension datasets containing tuples of (*passage, question, answer*)
- Including *passage* eliminates the need for information retrieval
- A system can be trained to predict a span in passage as answer, given a question

- Stanford Question Answering Dataset (**SQuAD**) (Rajpurkar et al., 2016)
  - Over 150,000 questions
  - Passage from Wikipedia; SQuAD 2.0 includes unanswerable questions

Retrieval-based QA Datasets

- **HotpotQA** dataset (Yang et al., 2018): Showing crowd workers multiple context documents and asked to create questions that require reasoning
- Both SQuAD and HotpotQA are created by annotators who have first read the passage may make their questions easier to answer

- Datasets from questions that were not written with a passage in mind
- **TriviaQA** dataset (Joshi et al., 2017)   Trivia: 琐事，娱乐和消遣
  - 94K questions written by trivia enthusiasts, with supporting documents (Wikipedia and web)
  - 650K question-answer-evidence triples
  - Relatively complex, compositional questions
  - Requires more cross sentence reasoning

Retrieval-based QA Datasets

- **MS MARCO** (Microsoft Machine Reading Comprehension) (Nguyen et al., 2016)
  - 1 million real anonymized questions from Microsoft Bing query logs
  - with a human generated answer and 9 million passages
  - https://microsoft.github.io/msmarco/

- **Natural Questions** dataset (Kwiatkowski et al., 2019)
  - Anonymized queries to the Google search engine
  - Annotators are presented a query, along with a Wikipedia page from the top 5 search results
  - To annotate a paragraph-length **long** answer and a **short** span answer, or mark **null** if the text doesn't contain the paragraph.
  - https://ai.google.com/research/NaturalQuestions

Current Solution to QA

- Two-stage **retriever/reader** model

- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents
- Stage 2. Reader algorithms: Either extract or generate an answer

Reader
- Extractor: **span extraction** ⇒ find spans of text that answer the question over the retrieved passages
- Generator: **retrieval-augmented generation** ⇒ Take a large pretrained LM, design the prompt based on the retrieved passage, and generate the answer token by token

Reader: Retrieval-Augmented Generation (RAG)

- Cast the QA task as word prediction: feeding the LM a question and a token like "A: " -- suggesting the answer should come next

$$\underbrace{\text{[Q:]}}\ \underbrace{\texttt{Q: Who wrote the book ``The Origin of Species"?}}_{q}\ \underbrace{\texttt{A:}}_{\text{[A:]}}\ x_1, x_2, \ldots, x_n$$

Conditional generation that optimizes: $$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | [Q:]; q; [A]; x_{<i})$$

Problems: hallucination; no access to proprietary data

Retrieval-Augmented Generation (RAG)

- Idea: Conditioning on the retrieved passages as part of the prefix perhaps with some prompt like “Based on these text, answer this question”

Query q → Retrieved passages $R(q)$ → Append prompt text:

```
retrieved passage 1

retrieved passage 2

...

retrieved passage n
```

Based on these texts, answer this question: Q: Who wrote the book “The Origin of Species"? A:

Feed to LLM for generation

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | R(q); \text{prompt}; [Q:]; q; [A]; x_{<i})$$

RAG details

- Just like span-based extractor, RAG requires a successful retriever, in two-stage setting as well
- **Multi-hop** architecture may be needed: a query q is used to retrieve documents, which are then appended to original q for a *second* stage retrieval
- Detailed prompt engineering is needed
- When combining private data with public data, externally hosted LLMs may be concerned

Evaluation of Retrieval-based QA

- QA is commonly evaluated using **mean reciprocal rank**, or **MRR**

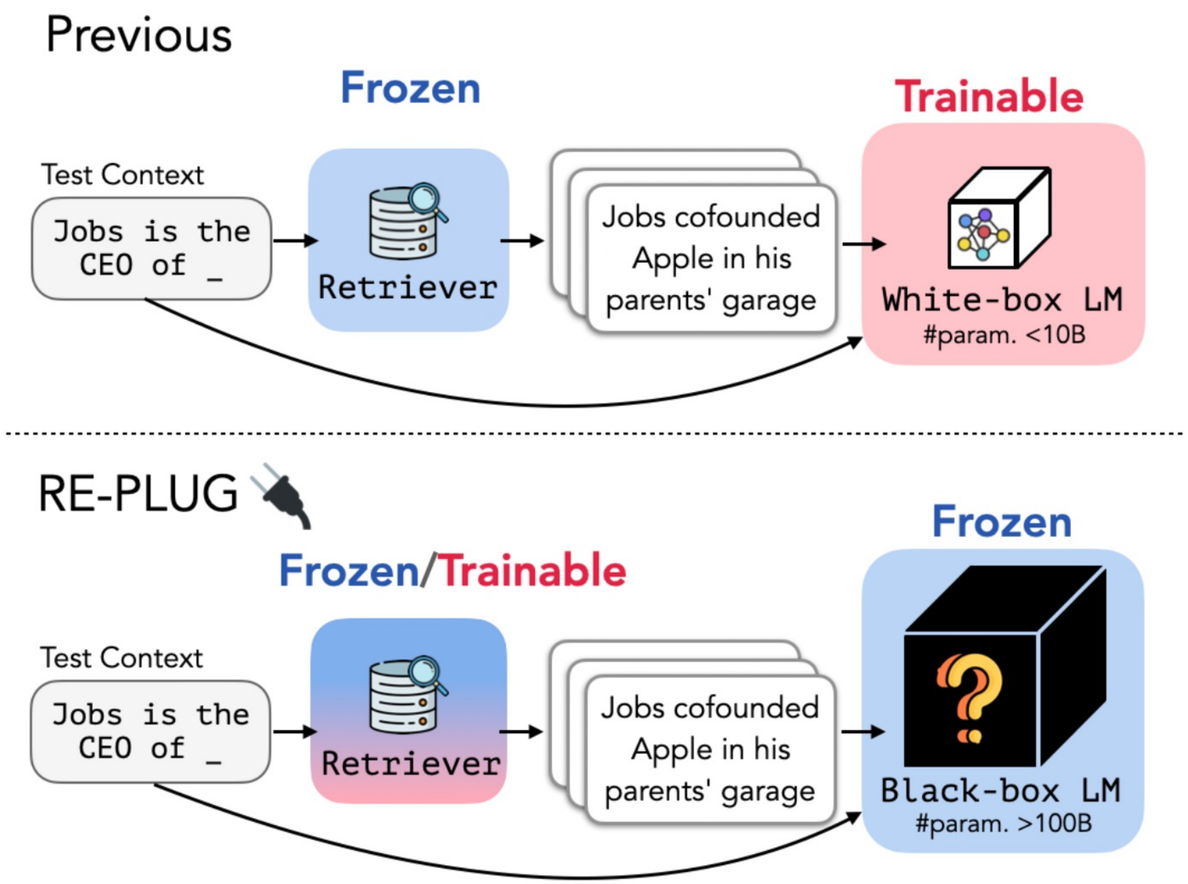
$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

- If the system returned 5 answers, but the first 3 are wrong, then the hightest-ranked correct answer is ranked 4th and thus the reciprocal rank is $\frac{1}{4}$

- Alternative methods:
- **Exact match**: The % of predicted answers that match the gold answer exactly
- **F-1 score**: average F-1 over all questions

RAG Example: RePLUG

Shi et al., 2023

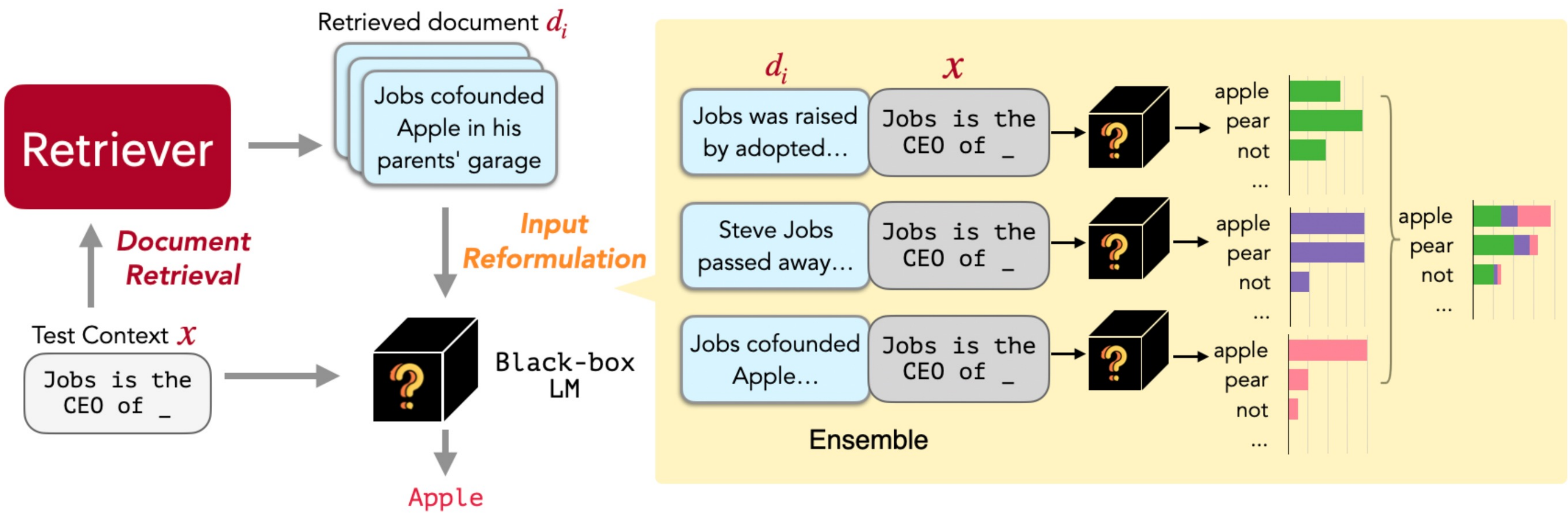


Treats the LM as a black box and augments it with a tuneable retrieval model

- Simply prepends retrieved documents to the input
- LM can be used to supervise the retrieval model

RAG Example: RePLUG

Shi et al., 2023



- Given an input context, REPLUG first retrieves a small set of relevant documents from an external corpus
- Then it prepends each document separately to the input context and ensembles output probabilities from different passes

Overview

- Question Answering (QA)
- **Project Discussion**
  - Do and Do-Nots
  - Trending Topics
  - Benchmarks
    - TriviaQA, Natural Questions, HotpotQA
    - GLUE, SuperGLUE

Project: What to do?

- Default project: **BERT + Fine-tuning on downstream tasks**
- Examples:
- BERT + Sequence Classification
  - Sentiment classification
  - Paraphrase detection
  - Semantic similarity etc.

Code template provided

- Or, BERT + QA on SQuAD, TriviaQA, Natural Questions etc.
- Or, BERT + Translation

BERT + Sequence Classification

Source code credit to: Stanford CS 224N Winter 2023 Default Final Project: Multitask BERT
https://github.com/gpoesia/minbert-default-final-project

- Primary Task: **Sentiment classification**
- Training dataset: Stanford Sentiment Treebank (SST) on movies
  - Train: 8545 lines of (sentence, score) pairs; score from 1 (neg) to 4 (pos)
  - Dev: 1102 lines
- Requirement
  - Finish the implementation of BERT (bert.py, skeleton provided, with six TODOs); Initialized from pretrained model
  - Fine-tune it on SST data (classifier.py, mostly implemented with two TODOs)
  - Extend and improve it in various ways:
    - Multi-task task through **paraphrase detection** and **semantic similarity regression** tasks (multitask_classifier.py, three new TODOs)
    - Different tasks correspond to different predict_xxx() functions in forward function

What to do with custom projects

- If you:
  - Have some research project that you’re excited about (and are possibly already working on)
  - You want to try to do something different
  - You want to see more of the process of defining a research goal, finding data and tools, and working out something you could do that is interesting, and how to evaluate it
- Then: Do the custom final project
- **Requirement**: must substantively involves both human language and neural networks

Project: What not to do?

- Train BIG models from scratch
  - Be realistic about the scale of compute you can do
  - You do not have the resources to train your own GPT-2 model from scratch
  - You probability do not have the resources to load a 7- to 11-B model (Llama-2, ChatGLM-3, Mistral-7B, T5-11B etc.)

Some trending topics

- Evaluating and improving models for something other than accuracy
  - Adaptation when there is domain shift
  - Evaluating the robustness of models in general
- Empirical work looking at what large pre-trained models have learned
- Get knowledge and good task performance without much data
- Bias, trustworthiness, and interpretability of large models
- Low resource languages or problems

Some trending topics

- Building small, performant models can be cool!
- Model pruning/quantization
  - QLoRA; Pruning; Compression: https://proceedings.mlr.press/v119/li20m/li20m.pdf; https://arxiv.org/pdf/2004.07320
  - Efficient Open-domain QA: https://efficientqa.github.io/ (within 6GB mem)
- Baby LM challenge: https://babylm.github.io/index.html
  - Efforts on optimizing pretraining given data limitations inspired by human development
  - 100M to 10M word text data

Grading: Project and Presentation

- Write-up quality

Try creative combos of model + data + method

It's okay your acc is low!

But do include analysis here!

Abstract Introduction	Prior related work	Model	Data	Experiments	Results	Analysis & Conclusion

- Focus on what you have done
  -- not on the amazing ChatGPT output showing that "look, it works zero-shot"
- Minimal 5 pages (template provided)

Important Dates

- In-class presentation of project: Week 16, Tuesday, June 4th, 2024
  - 7 minutes presentation + 3 minutes QA
- Project report due: Friday 11:59 PM, June 7th, 2024

References

- Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., & Deng, L. (2016). Ms marco: A human-generated machine reading comprehension dataset.
- Chen, D., Fisch, A., Weston, J., & Bordes, A. (2017). Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*.
- Joshi, M., Choi, E., Weld, D. S., & Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.
- Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535-547.
- Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.
- Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., ... & Yih, W. T. (2023). Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*.