The Information Retrieval Task

Information retrieval (IR)

User has an information need

And has some collection of documents

User wants to find a **relevant** document

- a document (or documents)
- in the collection
- that satisfy their need

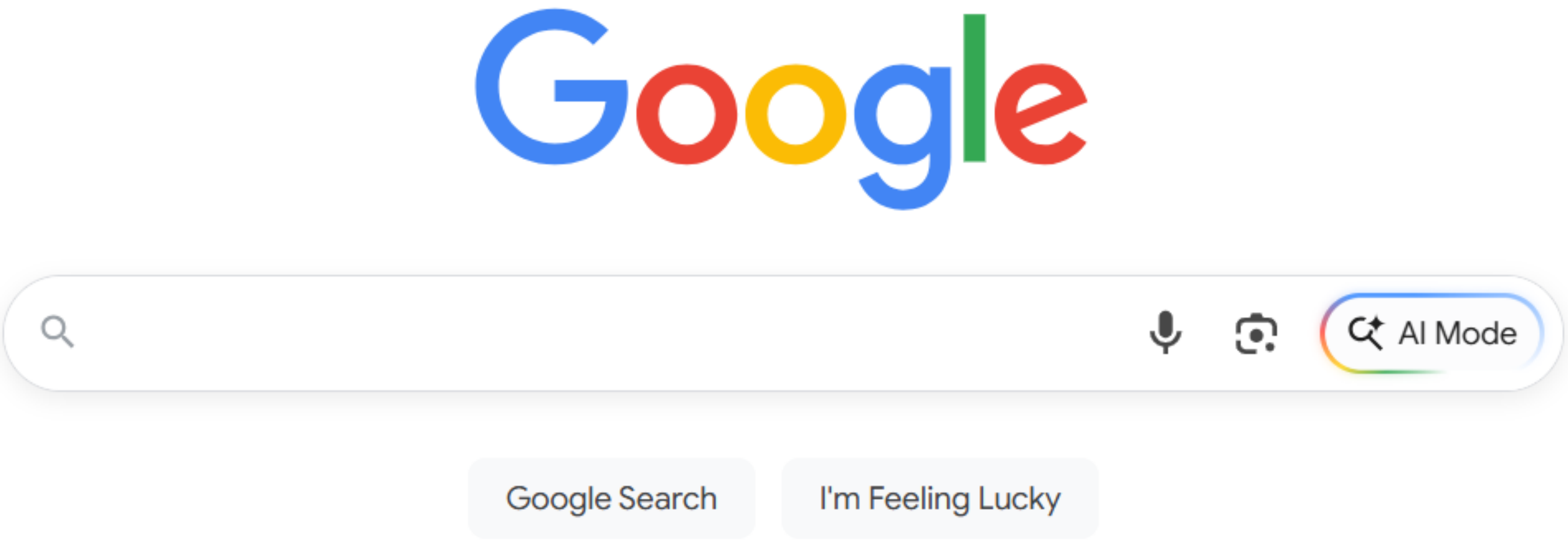
Simple factoid questions

Where is the Louvre Museum located?

Where does the energy in a nuclear explosion come from?

How to get a script l in latex?

Web search



Not just the web

Searching our email

Searching corporate documents

Searching personal medical records

The vector space model of IR

Gerard Salton, 1971

1. Represent each document as a vector of counts of the words it contains.
2. Represent a query as a vector too
3. Return the document whose vector is most **similar** to the query vector

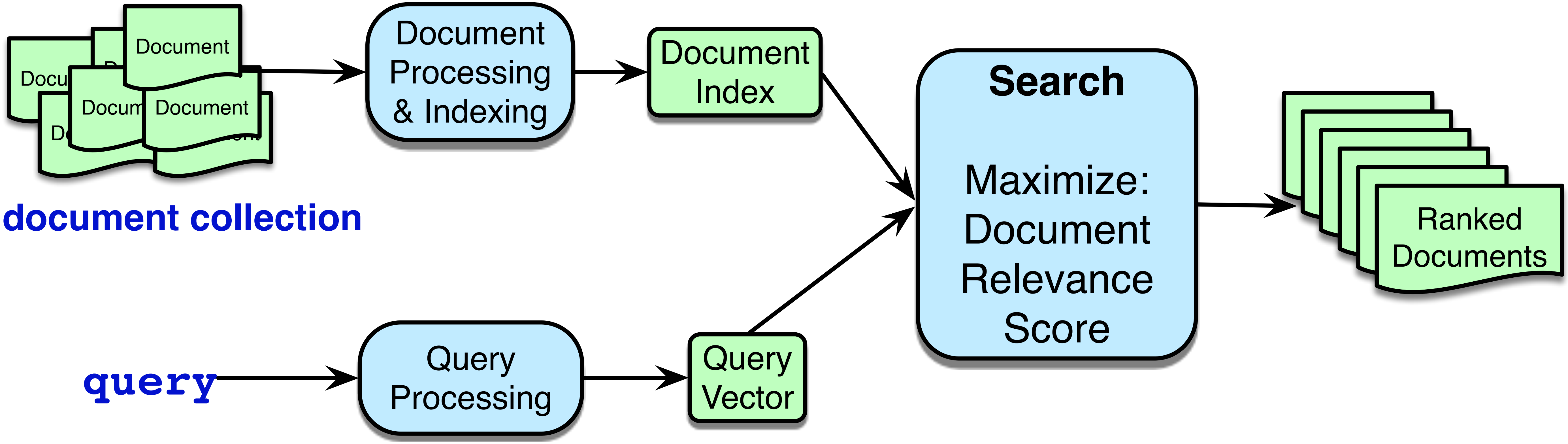
Ranked retrieval

The retriever returns top-k documents

These are ranked

We can show the user these, or some subset.

This task is called "ad hoc retrieval"



Document Relevance Score

Goal is to assign a score to each document for whether it meets the user's information need

Instead, we just approximate this by the textual similarity between the query and the document.

Two architectures for measuring query-doc relevance (= similarity)

Sparse retrieval

- represent query and doc as **vectors of word counts**
- weighted by tf-idf, BM25

Dense retrieval

- Use LLM to represent query and doc as **embeddings**

In both cases, similarity is usually the **cosine** between query and document representations

Information Retrieval in the LLM Age

If we have some information needs we don't need a document collection!

- Where is the Louvre Museum located?
- Where does the energy in a nuclear explosion come from?
- How to get a script l in latex?

Just prompt an LLM!

Just prompt an LLM!

AI Overview

The main Louvre Museum is located in **Paris, France, at the Musée du Louvre, 75001 Paris, France**. It is situated on the Right Bank of the Seine River in the city's 1st arrondissement, housed within the historic Louvre Palace.

- **Address:** Rue de Rivoli, 75001 Paris, France.

LLMs seem to store facts in the connections in their feedforward layers!

Information Retrieval in the LLM Age

But IR is still relevant for LLMs!

Retrieving relevant documents can still help solve problems that LLMs have with meeting information needs!

What are these problems?

LLMs Hallucinate

Hallucination: a response that is not faithful to the facts of the world.

In the legal domain LLMs were shown to hallucinate up to 88% of the time!

Dahl et al. (2024)

Can't use Proprietary Data

People need to ask questions about:

- personal email
- healthcare applications to medical records
- internal corporate documents
- legal documents discovery

These are (hopefully) not in the pretraining data of large language models!

Can't Handle Dynamic Data

LLMs can't answer questions about rapidly changing information like

- the sports match that happened last week
- the earthquake that happened this morning
- the latest Oscar winner

In general, data shifts over time!

Solution: RAG

Retrieval-Augmented Generation

1. Use IR to **retrieve** documents from some collection
2. Then use LLM to **generate** an answer conditioned on the documents

Information Retrieval and RAG

The Information Retrieval Task

Information Retrieval and RAG

Sparse retrieval: the vector model of IR

The vector space model of IR

Gerard Salton, 1971

1. Represent a document as a vector of counts of the words it contains.
2. Represent a query as a vector too
3. Return the document whose vector is most similar to the query vector

Bag-of-words model

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

Vector representation of that doc

adventure	1
and	3
fairy	1
genre	1
great	1
have	1
humor	1
I	5
it	6
satirical	1
seen	2
sweet	1
the	4
times	1
to	3
whimsical	1
would	1
yet	1
…	…

[1 3 1 1 1 1 1 5 6 1 2 1 4 1 3 1 1 1]

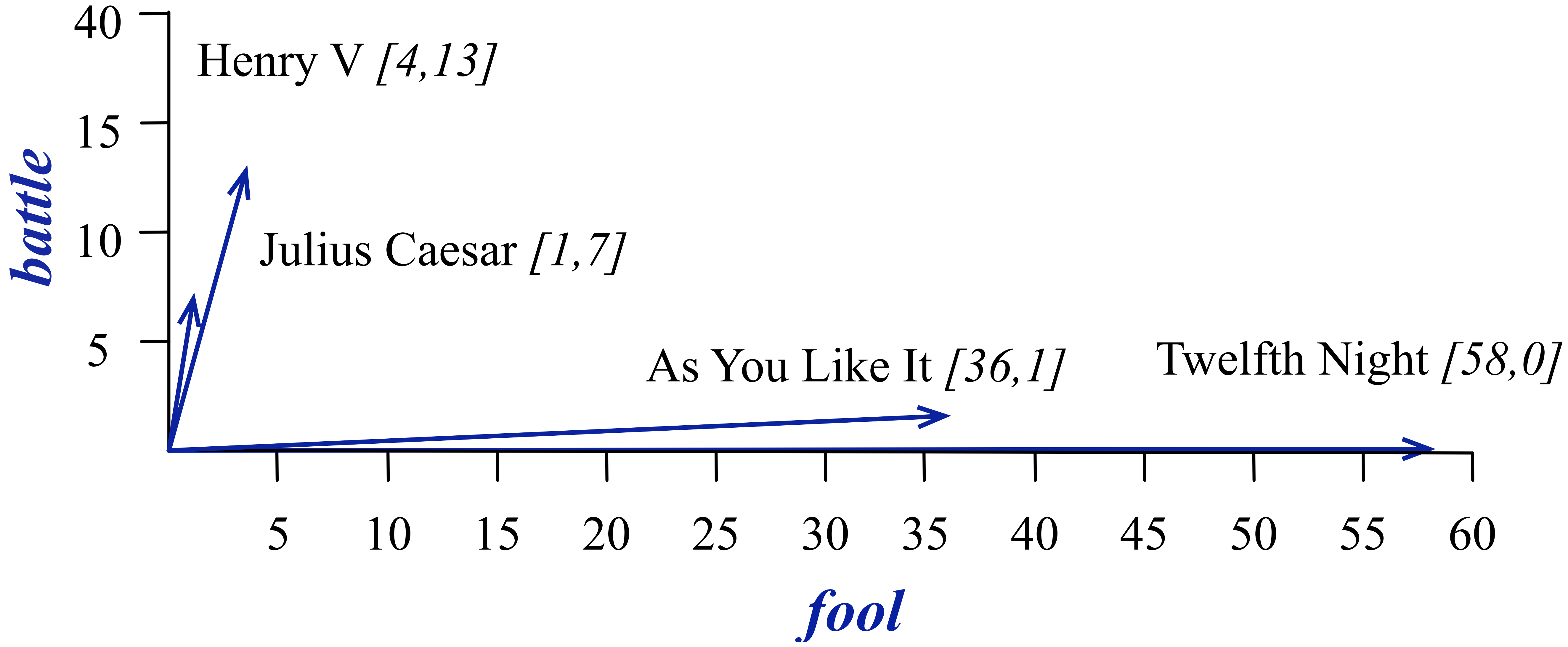
Term-document matrix

Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Visualizing document vectors

The two dimensional space [battle, fool]



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

The two comedies have similar vectors

But differ from Henry V and Julius Caesar

Comedies have more *fools* and *wit* and fewer *battles*.

Vector representations of queries and documents

Suppose we are looking for a witty fool play:

`Query = "fool wit"`

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Query
0
0
1
1

Choose the document that is most similar to the query

Which of $\mathbf{d_1}$, $\mathbf{d_2}$, $\mathbf{d_3}$, $\mathbf{d_4}$ is most similar to $\mathbf{q}$?

	d_1	d_2	d_3	d_4	q
	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**	**Query**
battle	1	0	7	13	0
good	114	80	62	89	0
fool	36	58	1	4	1
wit	20	15	2	3	1

Why dot product for similarity

The dot product tends to be high when the two vectors have large values in the same dimensions

Dot product can thus be a useful similarity metric between vectors

Similarity methods are variants of dot product

The dot product is $\mathbf{q} \cdot \mathbf{d}$

$$\textbf{score}\ (\mathbf{q},\mathbf{d}_4) = \mathbf{q} \cdot \mathbf{d}_4 =$$

	d_1	d_2	d_3	d_4	q
	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**	**Query**
battle	1	0	7	13	0
good	114	80	62	89	0
fool	36	58	1	4	1
wit	20	15	2	3	1

Problem with raw dot-product

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length $|\mathbf{d}| = \sqrt{\sum_{i=1}^{N} d_i^2}$

Long documents with many words have long vectors (since the counts are high in each dimension)

So dot product favors long documents

What we use instead to estimate word similarity: **cosine**

$$\text{cosine}(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|} = \frac{\sum_{i=1}^{N} q_i d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\sqrt{\sum_{i=1}^{N} d_i^2}}$$

Based on the definition of the dot product between two vectors a and b

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$

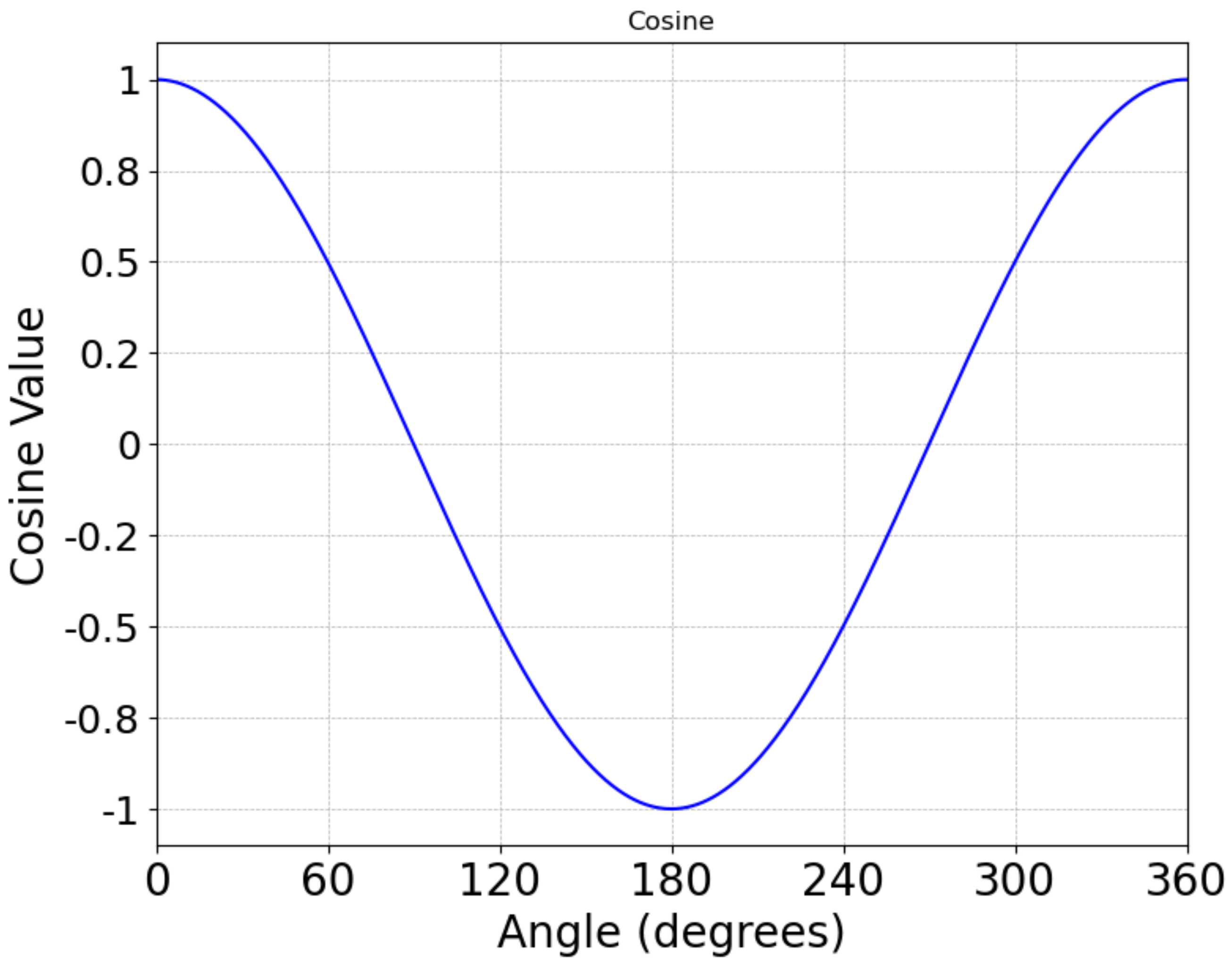
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

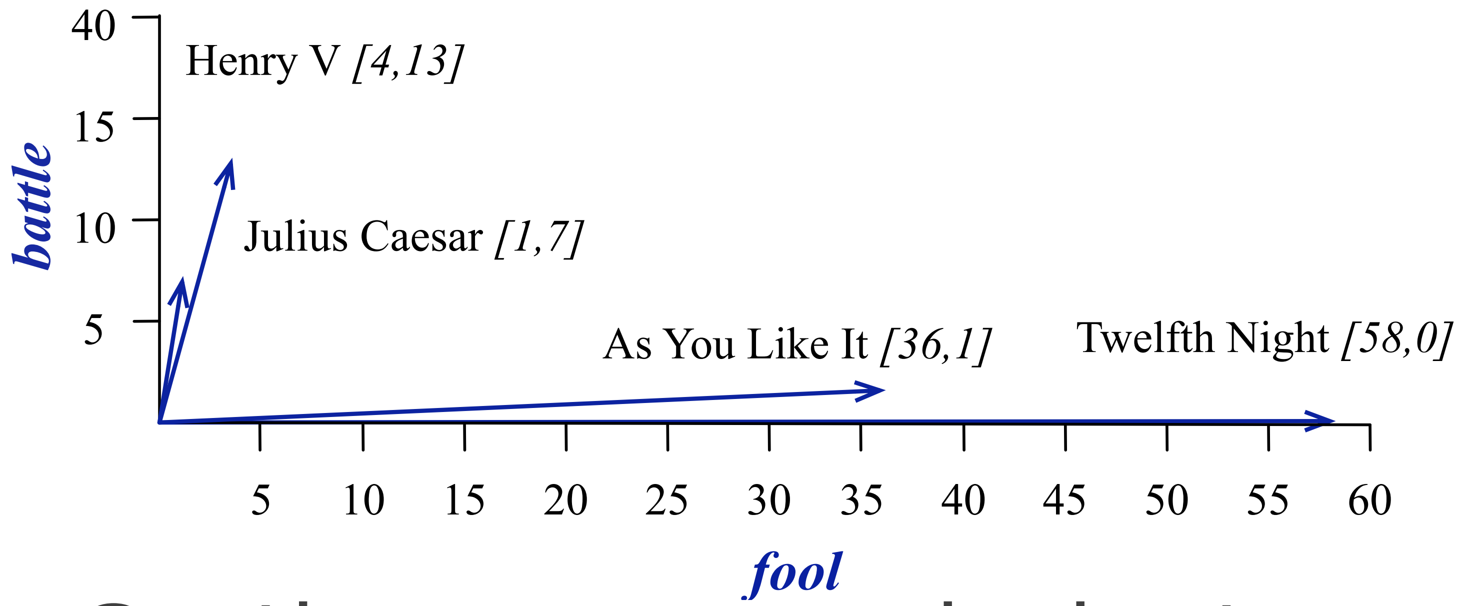
0: vectors are orthogonal

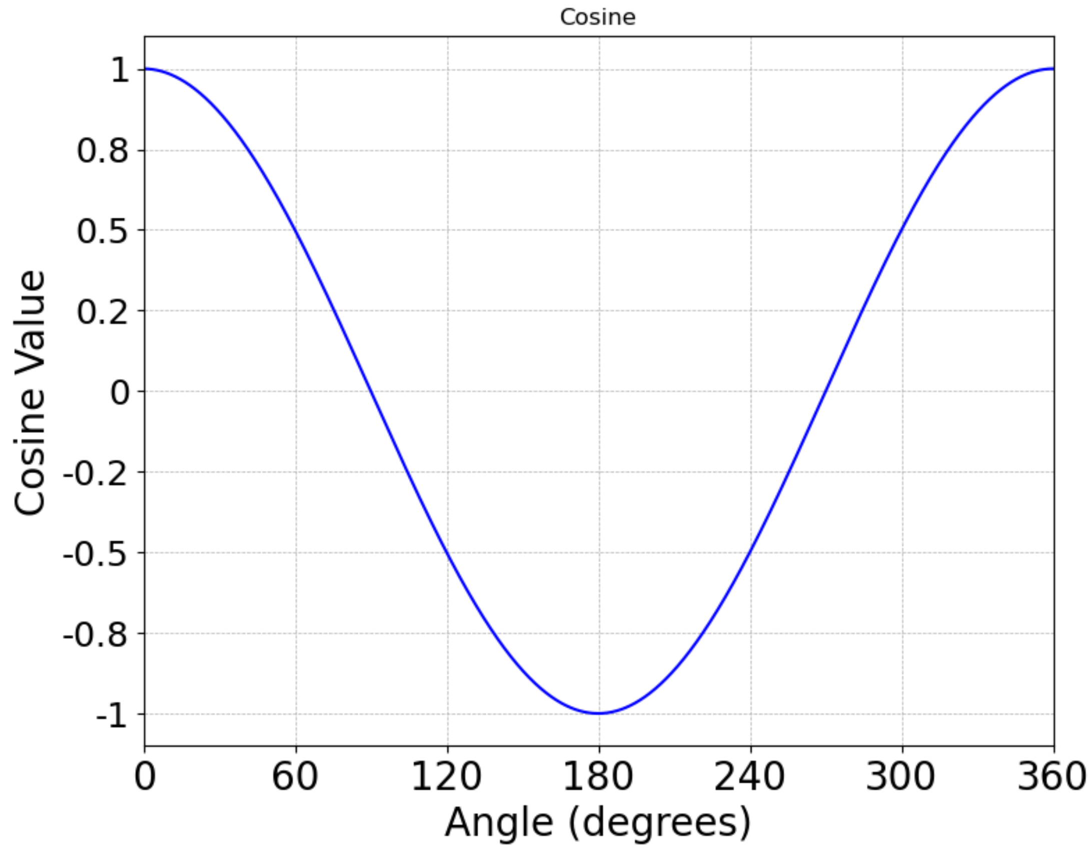


Cosine as a similarity metric

Since counts are non-negative,

Their vectors lie in the 1st quadra





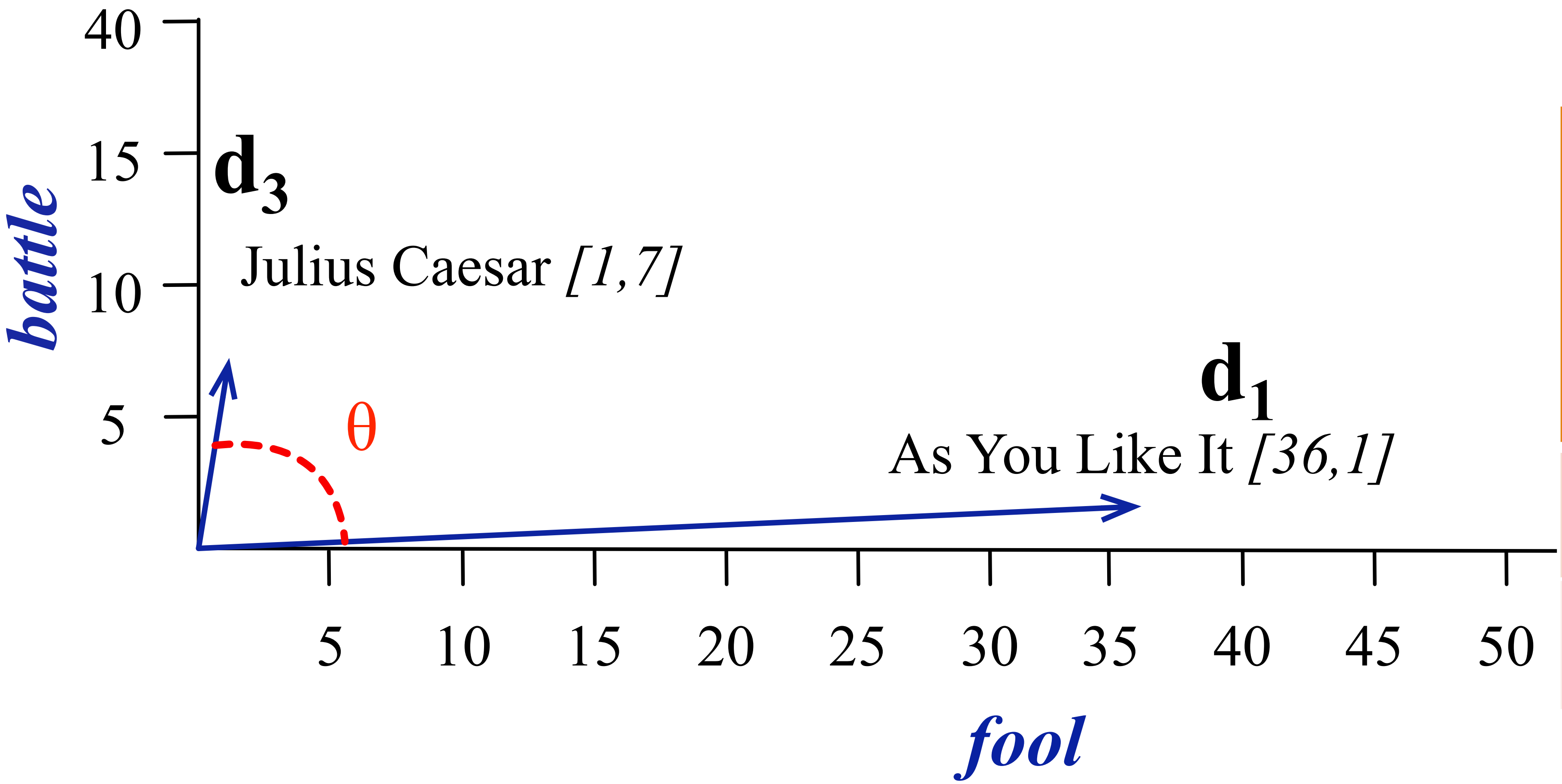
So the max angle between them is 90˚

So actually the cosine for word count vectors ranges from 0–1

$$\text{score}(q, d) = \cos(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{|\mathbf{q}||\mathbf{d}|}$$

Cosine example with two documents

$$\cos(\theta) =$$
$$\cos(d_3,d_1) = \frac{\mathbf{d_3} \cdot \mathbf{d_1}}{|\mathbf{d_3}| \cdot |\mathbf{d_1}|} =$$



	d_1: As You Like It	d_3: Julius Caesar
battle	1	7
fool	36	1

Information Retrieval and RAG

Sparse retrieval: the vector model of IR

Information Retrieval and RAG

TF-IDF

Raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *fool* appears a lot in some documents, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the document
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

tf-idf: tf-idf value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Words like "the" or "it" have very low idf

PMI: (Pointwise mutual information)

- $\text{PMI}(w_1, w_2) = log\frac{p(w_1,w_2)}{p(w_1)p(w_2)}$

See if words like "good" appear more often with "great" than we would expect by chance

Term frequency (tf) in the tf-idf algorithm

We could imagine using raw count:

$$\text{tf}_{t,d} = \text{count}(t,d)$$

But instead of using raw count, we usually squash a bit:

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

Document frequency (df)

df_t *is* the number of documents *t* occurs in.

(note this is not collection frequency: total count across all documents)

"*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

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

What is a document?

Could be a play or a Wikipedia article

But for the purposes of tf-idf, documents can be **anything**; we often call each paragraph a document!

Final tf-idf weighted value for a word

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.246	0	0.454	0.520
good	0	0	0	0
fool	0.030	0.033	0.0012	0.0019
wit	0.085	0.081	0.048	0.054

Information Retrieval and RAG

TF-IDF

Information Retrieval and RAG

TF-IDF: a worked example

Cosine

$$\text{score}(q, d) = \cos(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{|\mathbf{q}||\mathbf{d}|}$$

$$= \frac{\mathbf{q}}{|\mathbf{q}|} \cdot \frac{\mathbf{d}}{|\mathbf{d}|}$$

TF-IDF weighted cosine

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|} \cdot \frac{\mathbf{d}}{|\mathbf{d}|}$$

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

TF-IDF weighted cosine

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

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

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

TF-IDF nano-example

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Doc 3: How sweet is love?

Doc 4: Nurse!

TF-IDF nano-example

Query: sweet love

Word	Count	tf $1+\log_{10}(c)$	df	idf $\log_{10}N/df)$	tf-idf tf x idf	normalized
sweet	**1**					
nurse	0					
love	**1**					
how	0					
sorrow	0					
is	0					

TF-IDF nano-example

Query: sweet love

Word	Count	tf $1+\log_{10}(c)$	df	idf $\log_{10}(N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1				
nurse	0					
love	**1**	1				
how	0					
sorrow	0					
is	0					

TF-IDF nano-example

Query: sweet love

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Doc 3: How sweet is love?

Doc 4: Nurse!

Word						normalized
sweet						
nurse						
love						
how						
sorrow	0					
is	0					

TF-IDF nano-example

Query: sweet love

Word	Count	tf $1+\log_{10}(c)$	df	idf $\log_{10}(N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1	3	0.125		
nurse	0					
love	**1**	1	2	0.301		
how	0					
sorrow	0					
is	0					

TF-IDF nano-example

Query: sweet love

Word	Count	tf $1+\log_{10}(c)$	df	idf $\log_{10}(N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1	3	0.125	0.125	
nurse	0					
love	**1**	1	2	0.301	0.301	
how	0					
sorrow	0					
is	0					

TF-IDF nano-example

Query: sweet love

$|q| = \sqrt{(.125^2 + .301^2)} = .325$ $.125/.325=$

Word	Count	tf $1+\log_{10}(c)$	df	idf $\log_{10}N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1	3	0.125	0.125	0.383
nurse	0					
love	**1**	1	2	0.301	0.301	0.924
how	0					
sorrow	0					
is	0					

TF-IDF nano-example

Doc 2: Sweet sorrow

Word	Cnt	tf $1+\log_{10}(c)$	df	idf $\log_{10}N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1	3	0.125	0.125	
nurse	0					
love	0					
how	0					
sorrow	**1**	1	1	0.602	0.602	
is	0					

TF-IDF nano-example

Doc 2: Sweet sorrow

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

Word	Cnt	tf $1+\log_{10}(c)$	df	idf $\log_{10}N/df)$	tf-idf tf x idf	normalized
sweet	**1**	1	3	0.125	0.125	0.203
nurse	0					
love	0					
how	0					
sorrow	**1**	1	1	0.602	0.602	0.979
is	0					

Query: sweet love . **Doc 2**: Sweet sorrow

0.203 * 0.383 = **0.0779**

Word	Cnt	tf $1+\log_{10}(c)$	df	idf $\log_{10}N/df)$	tf-idf tf x idf	normalized	query
sweet	**1**	1	3	0.125	0.125	0.203	0.383
nurse	0					0	0
love	0					0	0.924
how	0					0	0
sorrow	**1**	1	1	0.602	0.602	0.979	0
is	0					0	0

Final cosine

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Doc 3: How sweet is love?

Doc 4: Nurse!

(details in chapter)

score(q,d1) = 0.747

score(q,d2) = 0.0779

- d1 has both terms, including 2 instances of *sweet*
- d2 is missing one of the terms

Information Retrieval and RAG

TF-IDF: a worked example

Information Retrieval and RAG

Efficiency: The Inverted Index

Goal: rank documents in **D** by their TF-IDF-weighted cosines with query **q**

Do we have to consider all documents in D?

No! **We can ignore all documents that don't have any query words!**

They will have a cosine of 0!

The Inverted Index

How do we efficiently find all documents that contain a query term q_i?

An index, that given a term, lists all the documents that have it.

```
giraffes,   35,72,245
wolverines,      104-110
numbered pages, 1-388
random page numbers, 8, 44-9, 70, 84, 213-28, 337
```

Which for historical reasons we call an inverted index!

Inverted Index

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

Two parts

Dictionary:	**Postings:**
how	→ 3
is	→ 3
love	→ 1→ 3
nurse	→ 1 → 4
sorrow	→ 2
sweet	→ 1→ 2 → 3

Inverted Index Creation

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

→

Term	Doc#
sweet	1
sweet	1
nurse	1
love	1
sweet	2
sorrow	2
how	3
sweet	3
is	3
love	3
nurse	4

→

1. Sort by term and document

Term	Doc#
how	3
is	3
love	1
love	3
nurse	1
nurse	4
sorrow	2
sweet	2
sweet	1
sweet	1
sweet	3

→

2. Create linked postings list

Dict: Postings:

how → 3
is → 3
love → 1→ 3
nurse → 1 → 4
sorrow → 2
sweet → 1→ 2 → 3

Inverted Index

Dict	Postings
how	→ 3
is	→ 3
love	→ 1 → 3
nurse	→ 1 → 4
sorrow	→ 2
sweet	→ 1 → 2 → 3

So far this just tells us which documents to grab

But we also need enough information to compute tf-idf:

For each term t in vocabulary:

- The df_t (document frequency) of word t

For each term t in each document d:

- The term frequency or count(t,d) of term t in doc d

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

Doc#

tf

df

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 $\{3\} \rightarrow 1\ [2] \rightarrow 2\ [1] \rightarrow 3\ [1]$

Information Retrieval and RAG

Efficiency: The Inverted Index

Information Retrieval and RAG

Evaluation of IR

Precision and Recall

We saw these already for classification

gold standard labels

system output labels		gold positive	gold negative
	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{precision} = \frac{tp}{tp+fp}$$

$$\textbf{recall} = \frac{tp}{tp+fn}$$

$$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$$

Precision: % of returned items that are correct

Recall: % of correct items that are returned

Precision and Recall for IR

User makes an information request

Every document in collection is either:

- **Relevant** to the user
- **Not relevant** to the user

The system retrieves a ranked set of documents

Precision for IR

Precision = % of **retrieved** documents that are **relevant**

System retrieves two kinds of documents

relevant documents

irrelevant documents

$$\textbf{Precision} = \frac{|\textbf{relevant retrieved}\text{ docs}|}{|\textbf{relevant retrieved}\text{ docs}| + |\textbf{irrelevant retrieved}\text{ docs}|}$$

Recall for IR

Recall = % of **relevant** documents that are **retrieved**

Recall = |retrieved **relevant** documents|

|all **relevant** document|

Precision and Recall aren't enough

This is **ranked** retrieval

- Given two ranked retrieval systems
- We want a metric that prefers the one that **ranks relevant documents higher**

We need to adapt precision and recall!

- to be sensitive to **where in the ranking** the relevant document occur

Rank	Judgment	$Precision_{Rank}$	$Recall_{Rank}$
1	R	1.0	.11

Rank-specific
precision and recall

Example:

- 25 documents
- 9 are relevant
- If we return all 25
  P=.36, R= 1.0
- Suppose we just return one (and it is relevant)?

Rank-specific precision and recall

Recall is non-decreasing

- Relevant docs increase recall
- Non-relevant docs don't

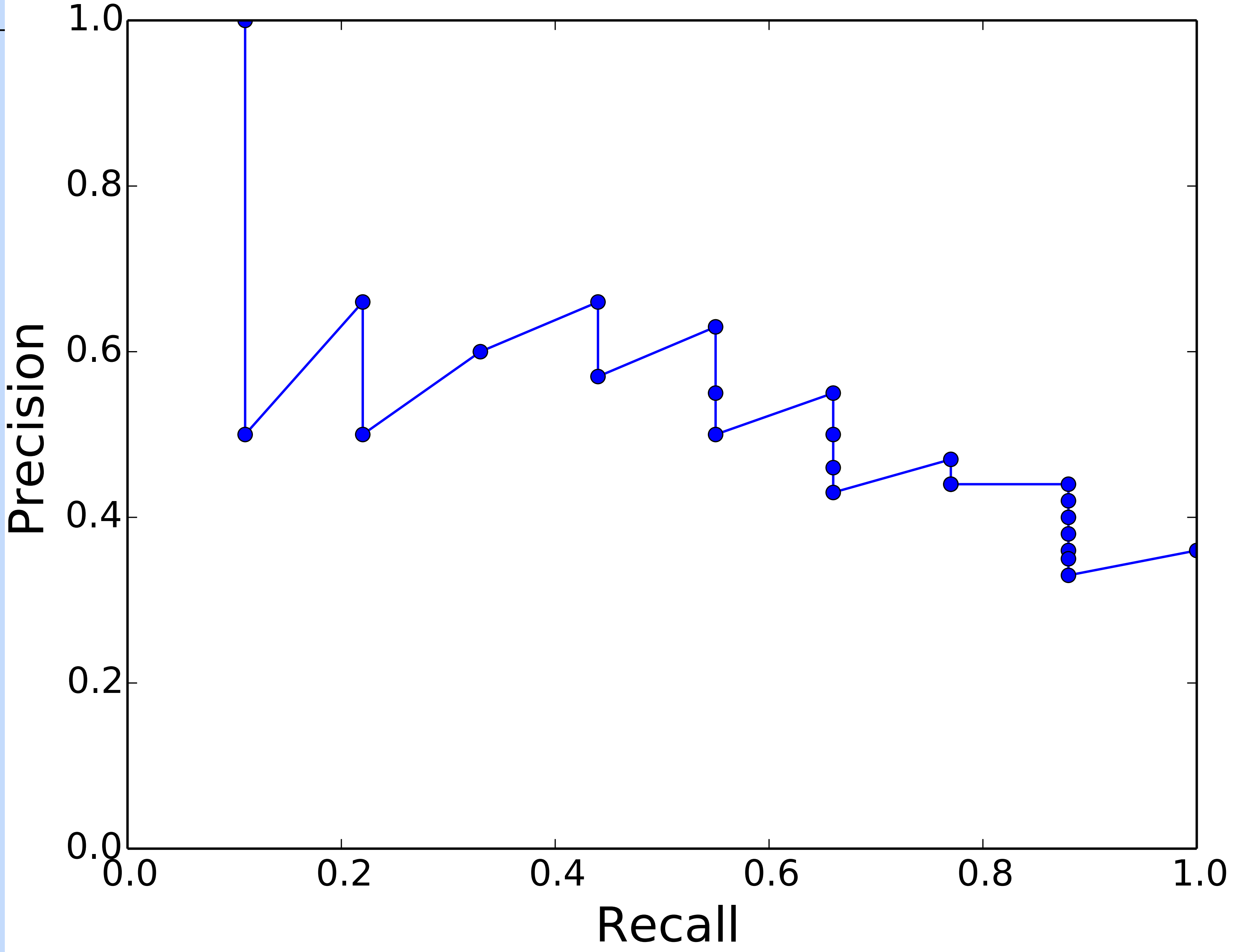
Precision jumps up and down

- increasing for relevant docs
- decreasing otherwise.

Rank	Judgment	$Precision_{Rank}$	$Recall_{Rank}$
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.41	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88
21	N	.38	.88
22	N	.36	.88
23	N	.35	.88
24	N	.33	.88
25	R	.36	1.0

The precision-recall curve (for one query)

Rank	Judgment	$Precision_{Rank}$	$Recall_{Rank}$
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.41	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88



Need a metric that aggregates over many queries

Two common approaches

- Mean Average Precision
- Interpolated Precision

Mean Average Precision

Descend through ranked items

Note precision only if item is **relevant**

- e.g. ranks 1, 3, 5, 6 but not 2 or 4:

Precision$_r$(d) "ranked precision"

precision at the rank doc *d* was found.

Rank	Judgment	Precision_{Rank}	Recall_{Rank}
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44

Average Precision

Descend through ranked items

Note precision only if item is **relevant**

Take the average of the ranked-precisions

$$\text{AP} = \frac{1}{|R_r|} \sum_{d \in R_r} \text{Precision}_r(d)$$

- R_r is the set of relevant documents at or above r
- $\text{Precision}_r(d)$ is precision measured at the rank at which document *d* was found.

Mean Average Precision

For a set of Q queries

Mean Average Precision (MAP):

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}(q)$$

Information Retrieval and RAG

Evaluation of IR

Dense Retrieval

Problem with classic IR

The **vocabulary mismatch problem**

- tf-idf cosine similarities measure query-doc similarity only if there is **exact word overlap** between query and doc!
- But query-writer can't know the exact words the doc might include!
  - Maybe query says "premiere" and document says "opening night"?

Dense retrieval

We still compare document vectors with query vectors to rank documents

But instead of representing query and documents with count vectors

- We represent both with dense **embeddings**!
- Here's a quick preview of embeddings; more details in future lectures

Sparse versus dense vectors

Count vectors (even if weighted by tf-idf)

- **big** (# of elements = |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

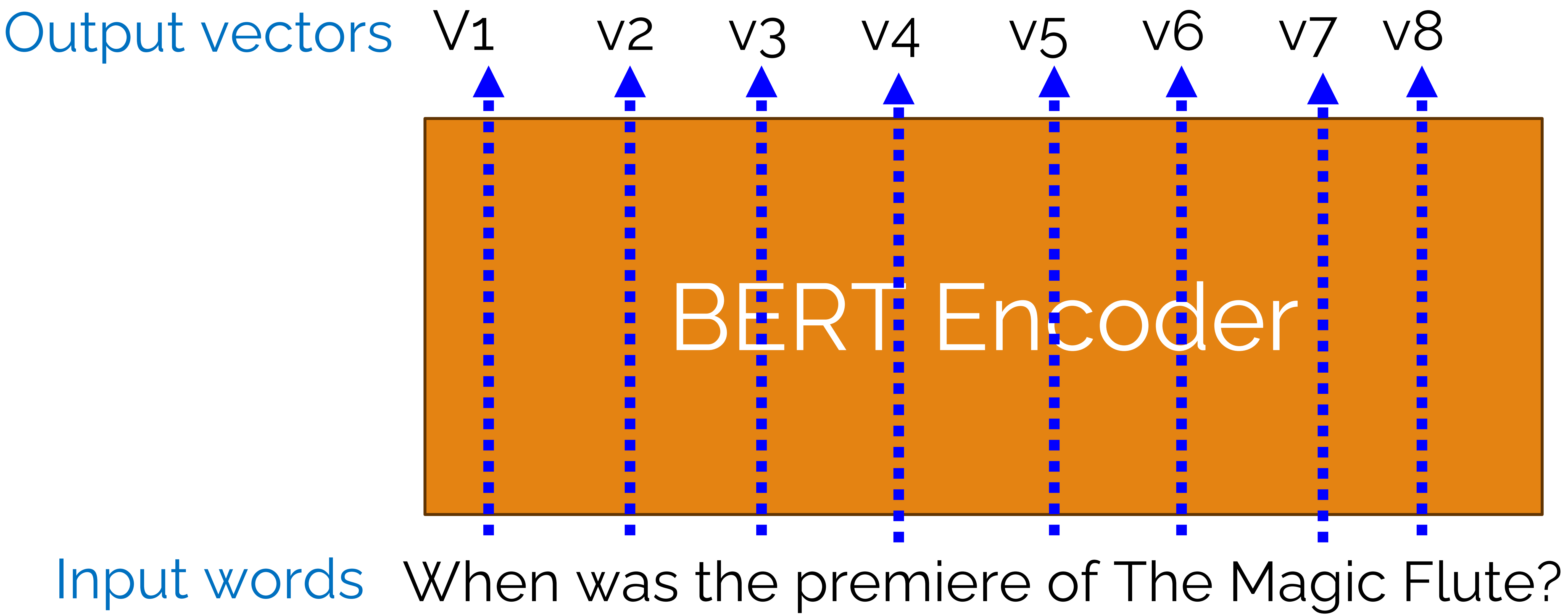
- **small** (# of elements 50-1000)
- **dense** (most elements are non-zero)

Dense retrieval

We use LLMs like BERT to compute **dense vectors** to represent the query and the document

These vectors are **contextual embeddings**

Dense embeddings for each word



Contextual embeddings

Each vector represents the meaning of a word in its context

V4 represents the meaning of:

- the word **premiere** in the context "When was the _____ of The Magic Flute"

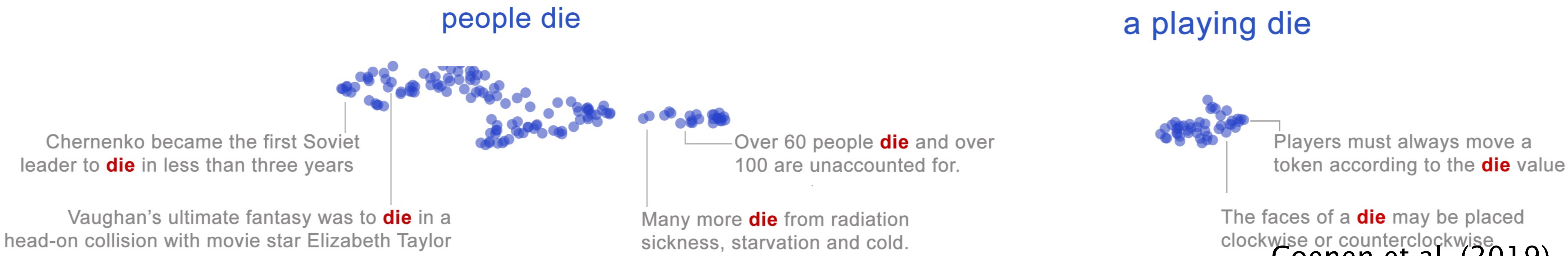
Contextual embeddings might be 1000 or 4000 dimensions.

Contextual **word embeddings**

Each word instance in context is represented as a single point in space

The meaning of a whole word is a region (a cloud of points) in 1000-dim space!

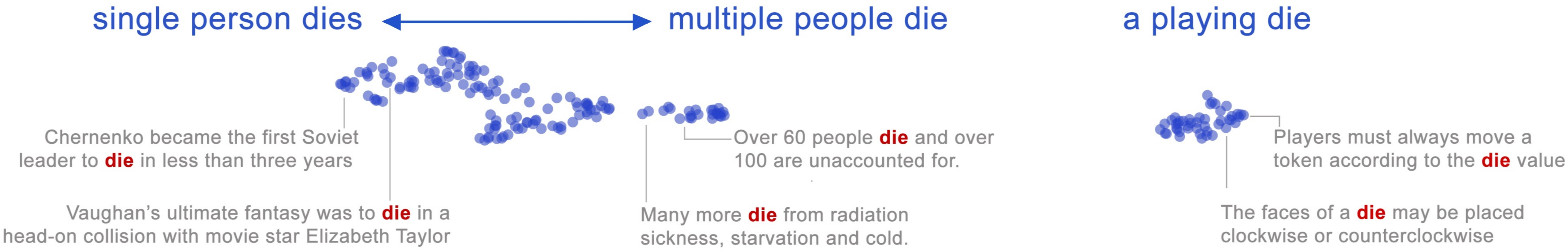
The word "die" in 2D, showing two regions for two senses



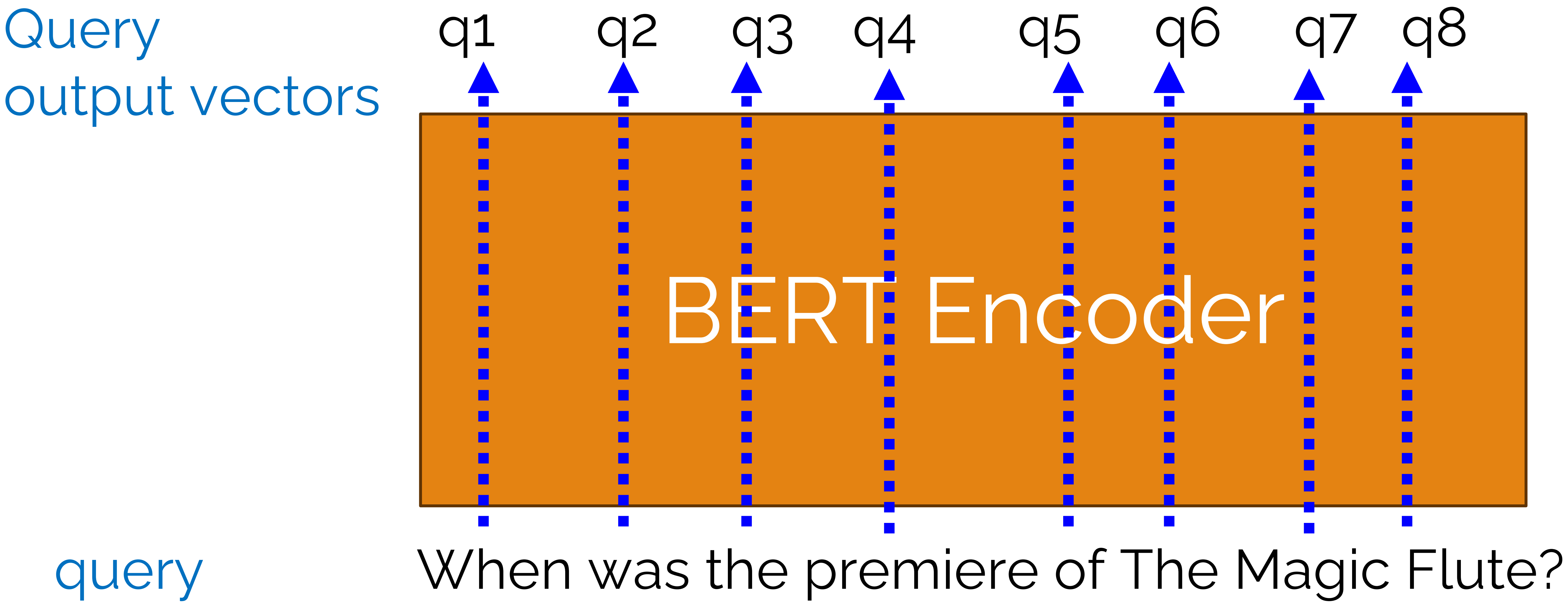
Coenen et al. (2019)

Contextual embeddings

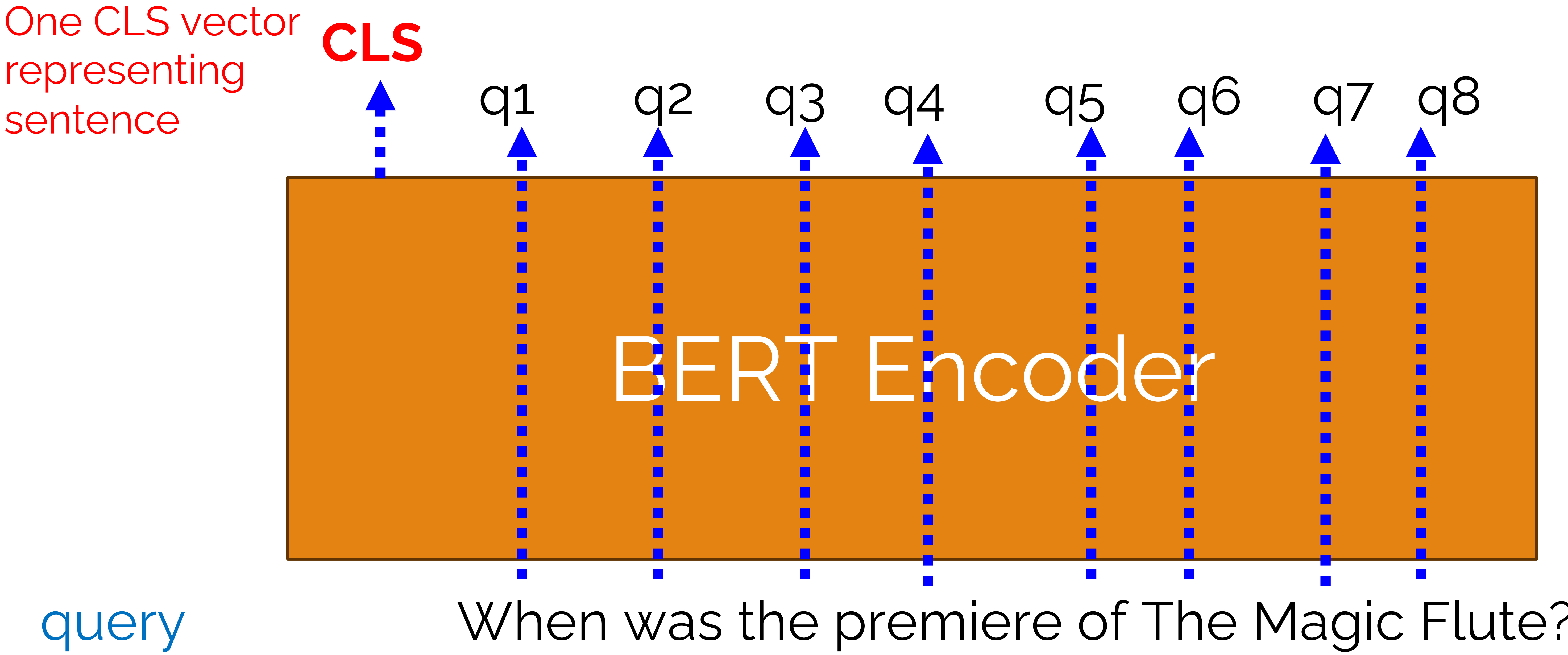
But not just two discrete senses



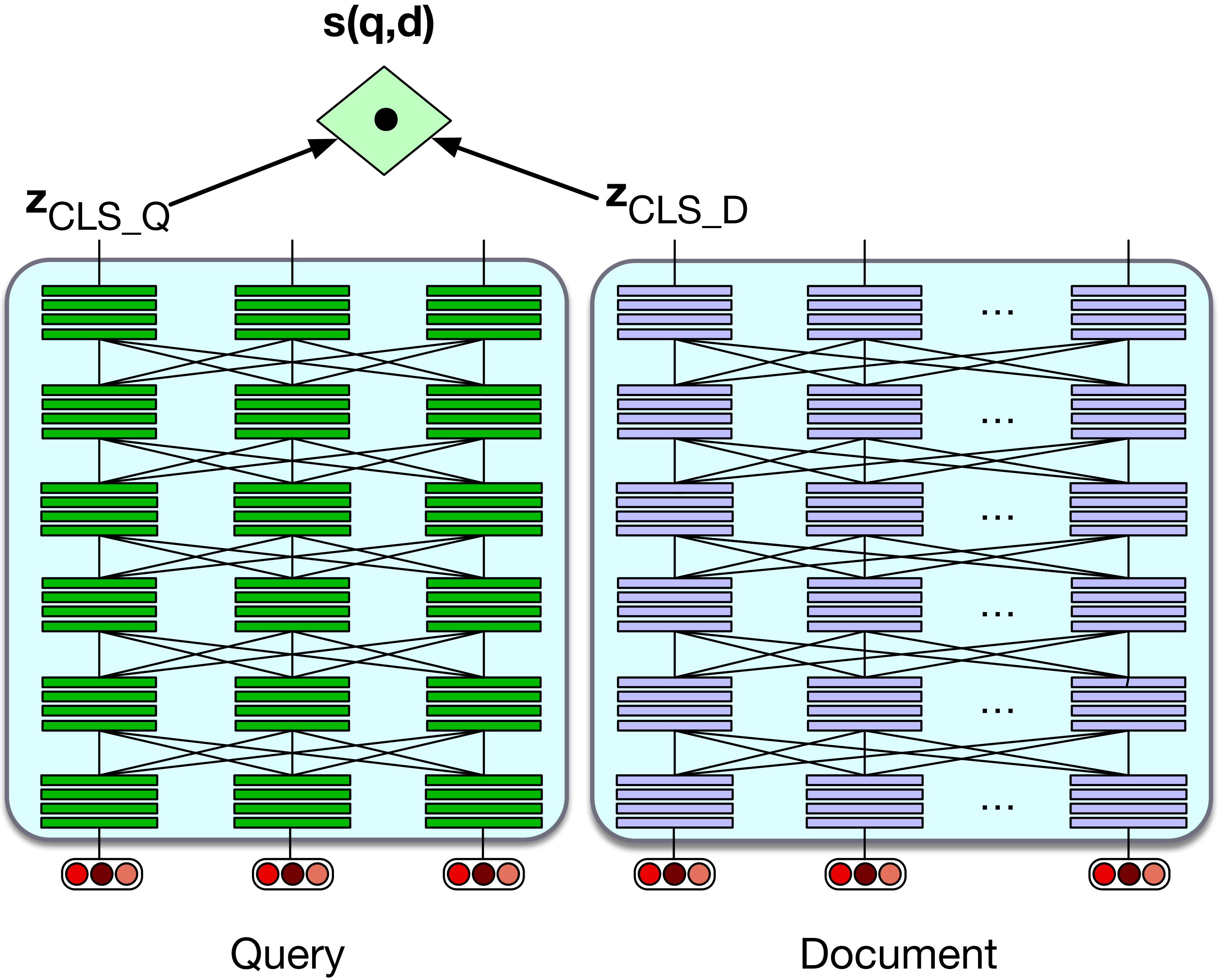
Dense vectors for each query



Dense vectors for each query



Simplest dense retrieval (**biencoder**)



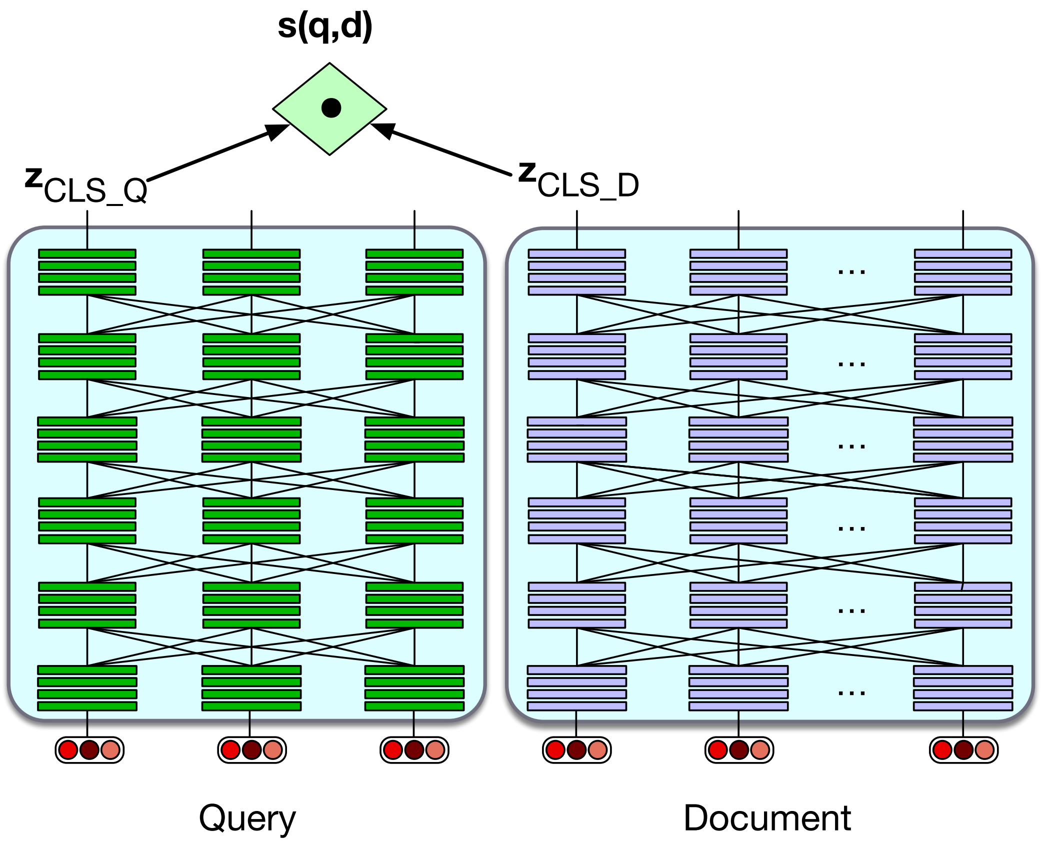
Score is dot product between query vector and document vector

$$\mathbf{z}_q = \mathrm{BERT}_Q(\mathrm{q})\,[\mathrm{CLS}]$$

$$\mathbf{z}_d = \mathrm{BERT}_D(\mathrm{d})\,[\mathrm{CLS}]$$

$$\mathrm{score}(q,d) = \mathbf{z}_q \cdot \mathbf{z}_d$$

Making dense retrieval efficient



Pre-encode all the document vectors in advance.

Encode query when it arrives

Efficiency in dense retrieval

We must rank **every document** for its similarity to the query!

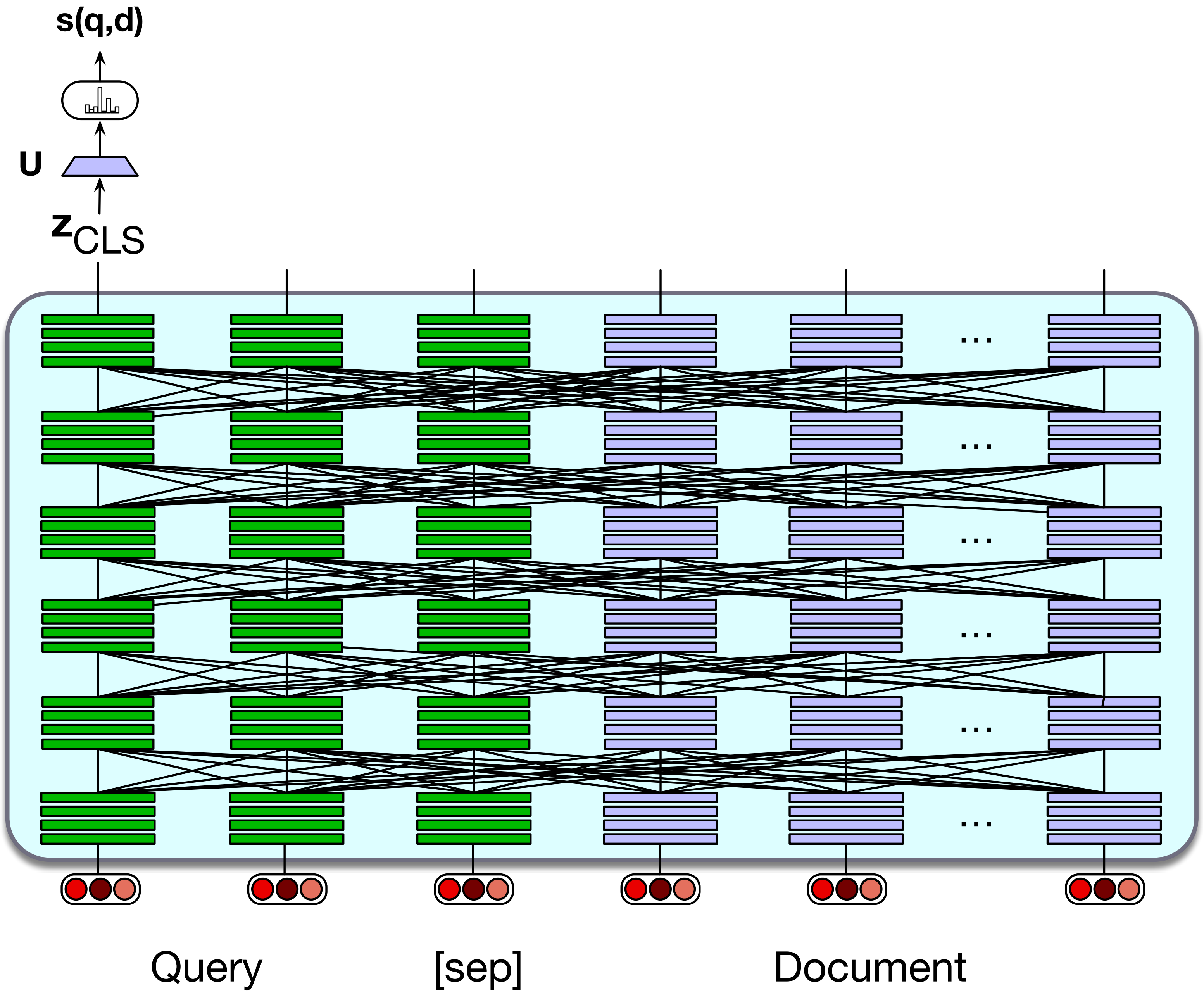
Efficiency for sparse word-count vectors: inverted index

Efficiency for dense retrieval:

- **nearest neighbor search**: finding the set of dense document vectors that have the highest dot product with a dense query vector.
- Approximate nearest neighbor algorithm **Faiss** (Johnson et al., 2017).
    - Approximates the doc vector by a smaller quantized vector

More on dense retrieval

Dense retrieval #2: Single encoder

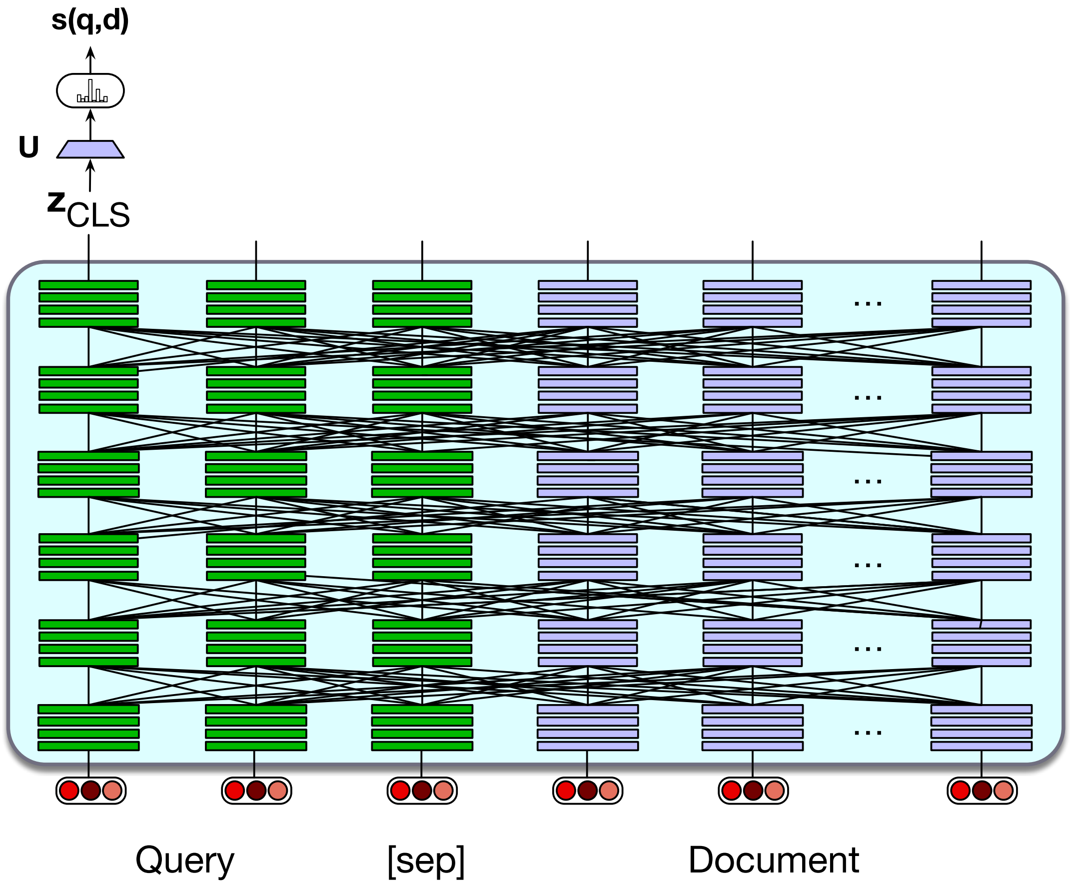


$$\mathbf{z} = \mathrm{BERT}(\mathrm{q}; \texttt{[SEP]}; \mathrm{d})\,\texttt{[CLS]}$$

$$\mathrm{score}(q, d) = \mathrm{softmax}(\mathbf{U}(\mathbf{z}))$$

More expensive than biencoder
But more accurate

Dense retrieval #2: Single encoder



System is run on passages (say 100 tokens) instead of whole documents

Training:

- BERT and linear layer U can then fine-tuned for relevance
- Creating a tuning dataset of relevant and non-relevant passages.

In-between dense retrieval methods

Use cheap methods (like BM25) as first pass relevance ranking for each document,

Then just rerank the top N ranked docs,

- Using expensive methods like the full BERT scoring

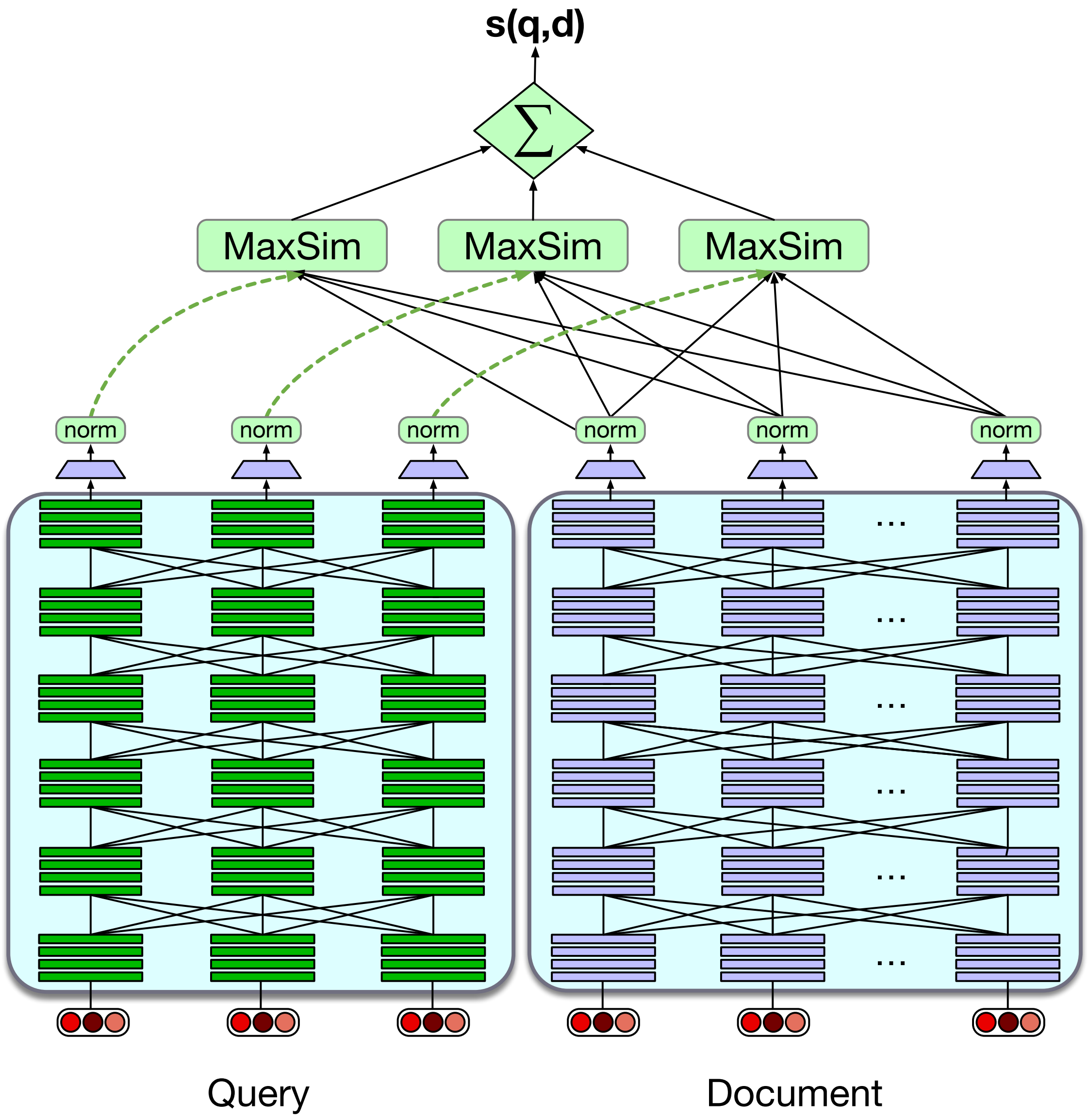
ColBERT

Precompute document but store each word vector

Then do maxsim between document and query words

Khattab and Zaharia (2020), Khattab et al (2021)

ColBERT



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

Training for dense retrieval

ColBERT and other models need to be trained

- To fine-tune the BERT encoders and train the linear layers (and the special [Q] and [D] embeddings)
- On datasets of triples $\langle q, d^{+}, d^{-} \rangle$
- Some datasets like MS MARCO Ranking have positive examples

Information Retrieval and RAG

Dense Retrieval

Information Retrieval and RAG

Retrieval-Augmented Generation

IR plays a central role in modern LLMs

LLMs can answer some information needs without a document collection!

- Where is the Louvre Museum located?
- Where does the energy in a nuclear explosion come from?
- How to get a script l in latex?

Just prompt an LLM!

Just prompt an LLM!

AI Overview

The main Louvre Museum is located in **Paris, France, at the Musée du Louvre, 75001 Paris, France**. It is situated on the Right Bank of the Seine River in the city's 1st arrondissement, housed within the historic Louvre Palace.

- **Address:** Rue de Rivoli, 75001 Paris, France.

LLMs seem to store facts in the connections in their feedforward layers!

But there are issues we mentioned earlier!

LLMs hallucinate, giving answers that aren't faithful to the facts of the world.

- They can make up medical, legal facts

LLMs can't use proprietary data

- Email, medical records, corporate docs

LLMs can't handle dynamic data

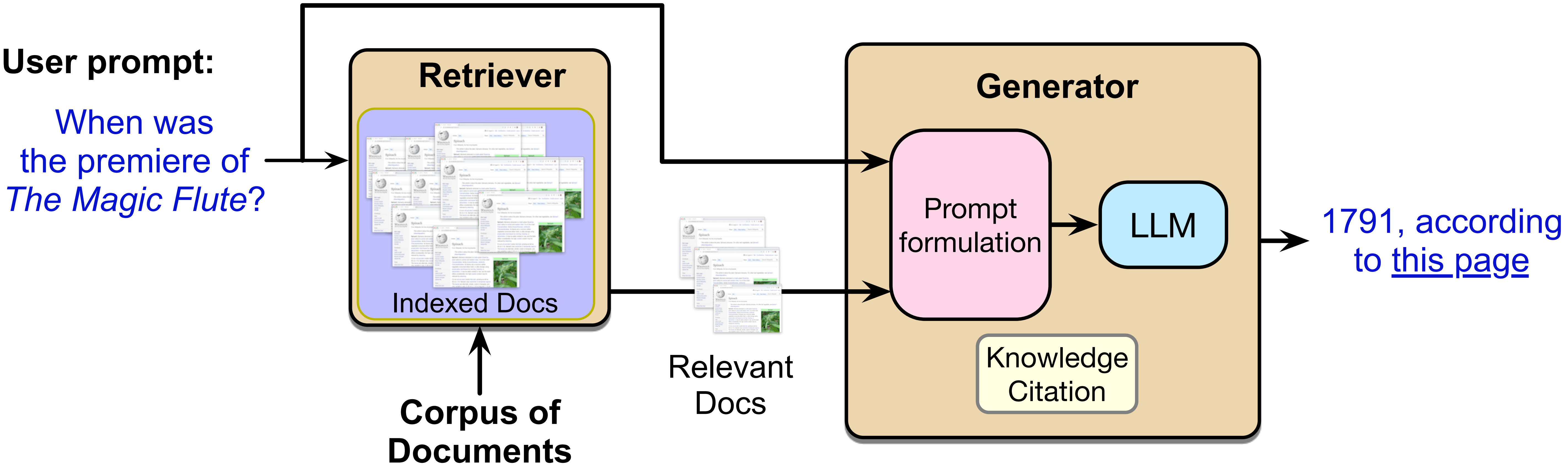
- Recent news, earthquakes, sports events

Solution: RAG

Retrieval-Augmented Generation

1. Use IR to **retrieve** documents from some collection
2. Then use LLM to **generate** an answer conditioned on the documents

Retrieval Augmented Generation (RAG)



Basic RAG

Given a document collection D and a user query q

- Call a retriever to return top *k* passages
- Create a prompt that includes *q* and the passages
- Call an LLM with the prompt

Schematic of a RAG Prompt

```
retrieved passage 1

retrieved passage 2

...

retrieved passage k

Based on these texts, answer this question: What year
was the premiere of The Magic Flute?
```

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | \mathrm{R}(q) \text{ ; Answer the following question... ; } q \text{ ;} x_{<i})$$

Extensions: Agent-based RAG

Instead of running RAG automatically on every user turn

Have a **retrieval agent**

System decides when to call the retrieval agent and for which document collection

Extensions: Training

Instruction-tune an LLM on a dataset of questions with retrieved passages and correct answers

Test-time compute: prompt an LLM to answer the question and simultaneously to generate reflections on which passages were useful

Extensions: Knowledge Citations

Q: Which films have Gong Li as a member of their cast?
A: The Story of Qiu Ju [1], Farewell My Concubine [2], The Monkey King 2 [3], Mulan [3], Saturday Fiction [3] ...

```
‘‘Write an answer for the given question
using only the provided search results
(some of which might be irrelevant) and
cite them properly...  Always cite for any
factual claim".
```

Information Retrieval and RAG

Retrieval-Augmented Generation

Information Retrieval and RAG

Question Answering datasets and evals

Two kinds of question answering datasets

Natural information-seeking questions

- Someone actually wanted to know the answer to this

Probing (testing) questions

- Exam-type questions for LLM evaluation

Natural Questions (Kwiatkowski et al., 2019),

anonymized English **queries** to the Google search engine and short and long **answers** (hand-created from Wikipedia)

"When are hops added to the brewing process?"

short answer: *the boiling process*

long answer: paragraph from the Wikipedia page on *Brewing*

MS MARCO (Microsoft Machine Reading Comprehension) collection of datasets,

1 million real anonymized English questions from Microsoft Bing query logs

human generated answer

9 million passages (Bajaj et al., 2016)

Probing dataset: MMLU

15908 knowledge and reasoning questions in 57 areas including medicine, mathematics, computer science, law, etc..

Sourced from exams for humans like GRE, AP

College Computer Science
Any set of Boolean operators that is sufficient to represent all Boolean expressions is said to be complete. Which of the following is NOT complete?
(A) AND, NOT
(B) NOT, OR
(C) AND, OR
(D) NAND

Information Retrieval and RAG

Question Answering datasets and evals