Embeddings

A walkthrough from Bag of Words to word2vec to Transformers to Context-dependent Embeddings

John Tan Chong Min

Bag of words

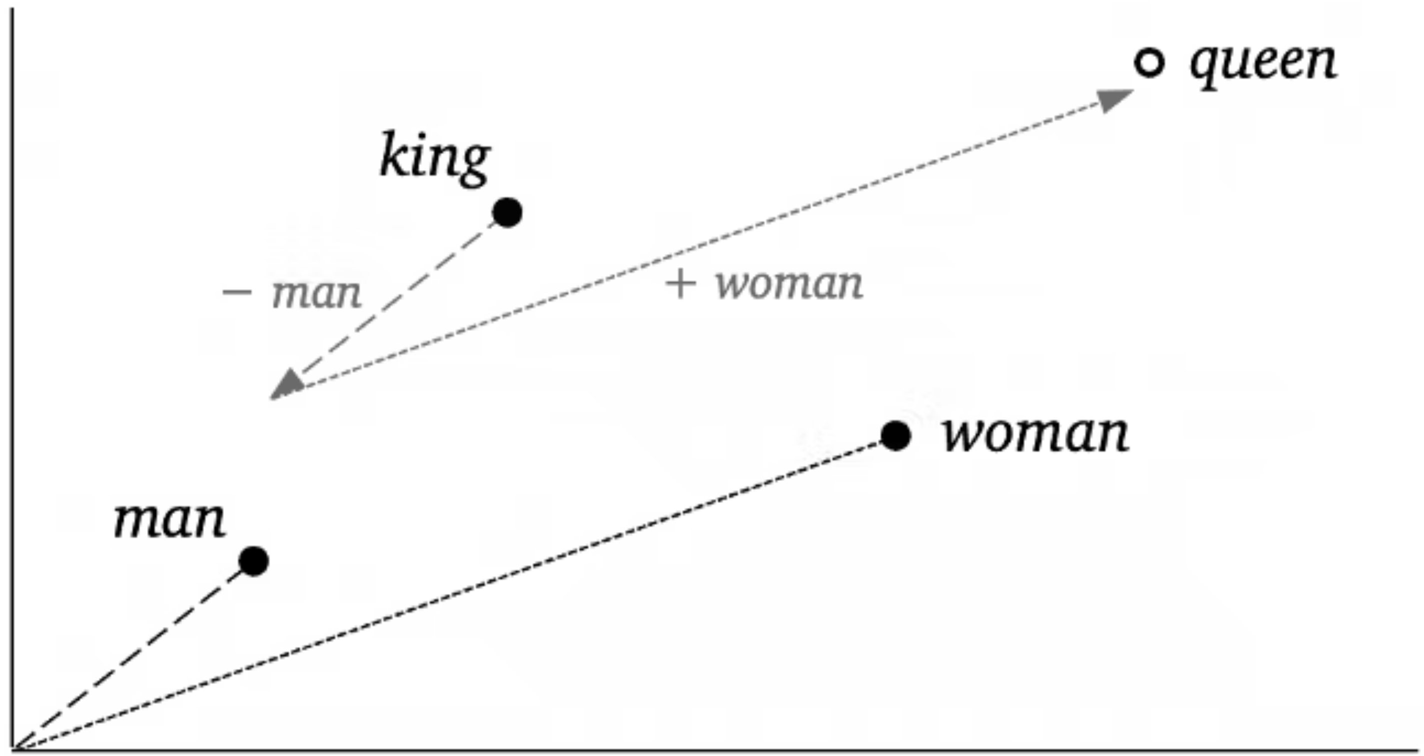
- Derive meanings of sentences by the occurrences of individual words
- Quite useful for detecting spam
- **BUT:**
    - Negative meanings and word order not factored in
    - Similar words not taken into account

	she	loves	pizza	is	delicious	a	good	person	people	are	the	best
She loves pizza, pizza is delicious	1	1	2	1	1	0	0	0	0	0	0	0
She is a good person	1	0	0	1	0	1	1	1	0	0	0	0
good people are the best	0	0	0	0	0	0	1	0	1	1	1	1

https://www.askpython.com/python/examples/bag-of-words-model-from-scratch

Can we represent words in continuous vectors?

Words as continuous vectors



Word2Vec

- Input: a large text corpora, V, d
  - V: a pre-defined vocabulary
  - d: dimension of word vectors (e.g. 300)
  - Text corpora:
    - Wikipedia + Gigaword 5: 6B
    - Twitter: 27B
    - Common Crawl: 840B
- Output: $f : V \to \mathbb{R}^d$

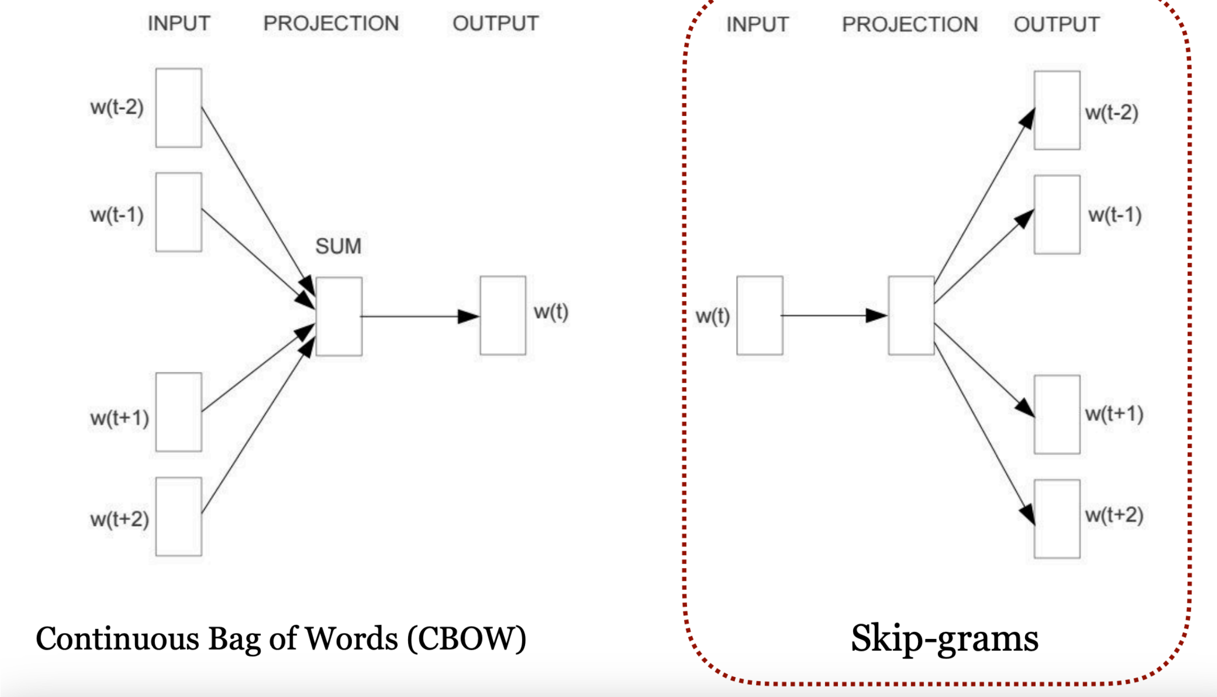
$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \qquad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \qquad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

https://courses.cs.washington.edu/courses/csep517/20wi/slides/csep517wi20-WordEmbeddings.pdf

Prediction of word from context / context from word

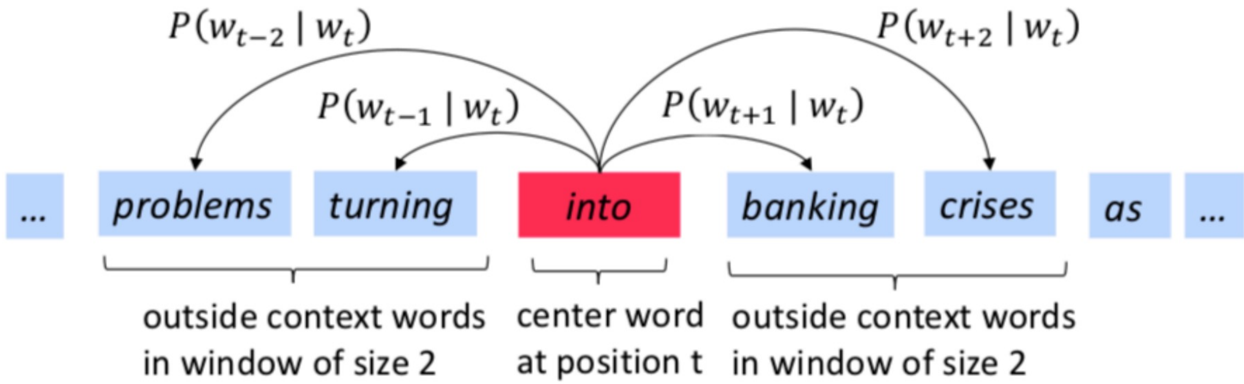
Intuition: A word is defined by its neighbours



https://courses.cs.washington.edu/courses/csep517/20wi/slides/csep517wi20-WordEmbeddings.pdf

Skip-gram

- The idea: we want to use words to **predict** their context words
- Context: a fixed window of size $2m$

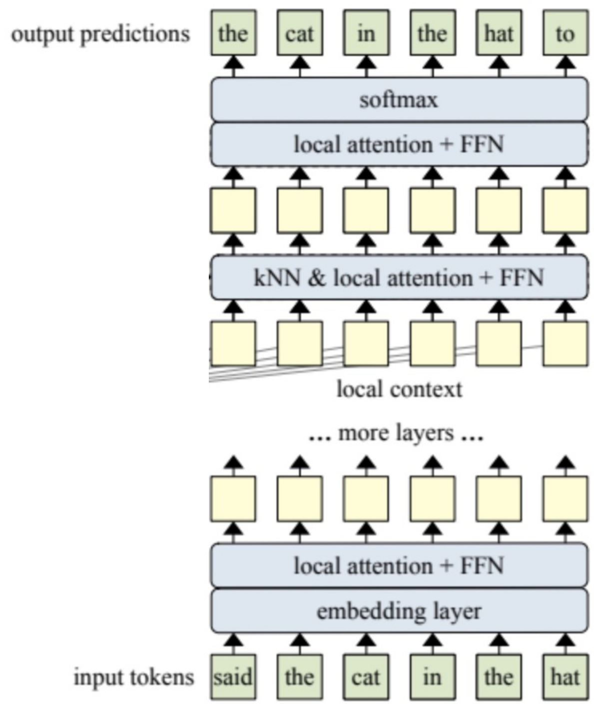


https://courses.cs.washington.edu/courses/csep517/20wi/slides/csep517wi20-WordEmbeddings.pdf

Can we learn embeddings via next-token prediction?

Transformer Embeddings

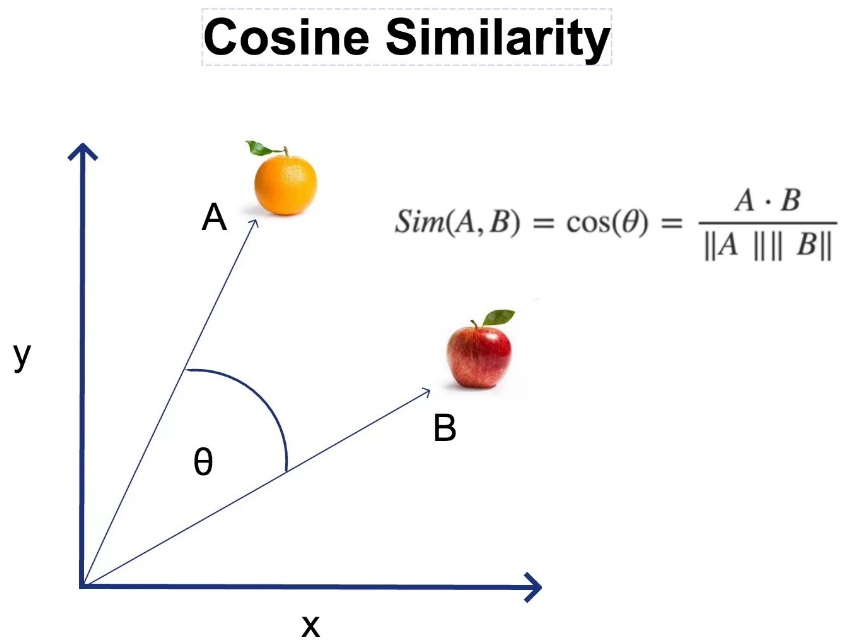
- Use next-token prediction to generate embeddings of tokens
- Tends to take similar words into account due to a similar statistical distribution when predicting next token
- Tokens more generalisable than words as it is based on frequently occurring characters
- Context-dependency on earlier tokens via attention can get **expressive token embeddings at the sentence level**



Memorising Transformers. Wu et. al. 2022.

Cosine Similarity

- Magnitude of OpenAI embeddings is 1
- Cosine similarity tells us the angle between two vectors
- Assumption: Vectors that point towards the same direction tend to be similar



https://aitechtrend.com/how-cosine-similarity-can-improve-your-machine-learning-models/

Learning Sentence-wise embeddings

- Naïve Rule
  - Neighbouring sentences are similar
  - Non-neighbouring sentences are not similar

- Start with Transformer pre-trained embeddings, and learn the embeddings between sentences via contrastive learning

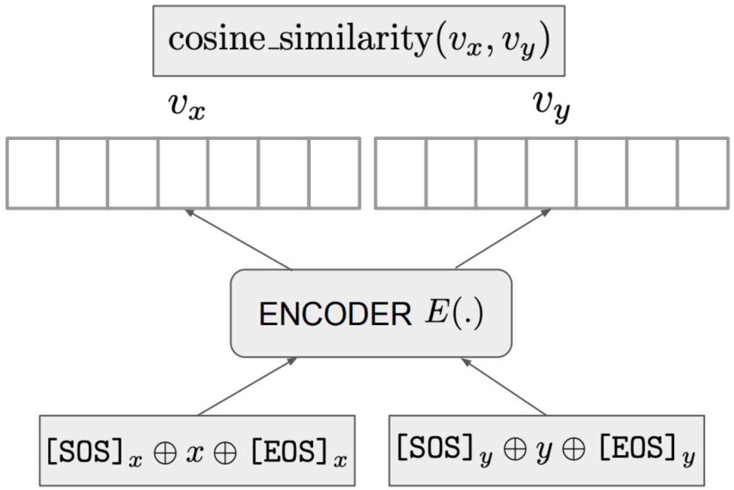


Figure 3. The encoder E maps inputs x and y, to embeddings, v_x and v_y independently. The similarity score between x and y is defined as the cosine similarity between these two embedding vectors.

Text and Code Embeddings by Contrastive Pre-Training. OpenAI. 2021.

Issues with just going by neighbouring sentences

Negation of values (text-embedding-ada-002)

```
[40]: model = 'text-embedding-ada-002'

[41]: np.dot(get_embedding('have', model), get_embedding('do not have', model))

[41]: 0.8617052588337162

[42]: np.dot(get_embedding('Jonathan was present', model), get_embedding('Jonathan was absent', model))

[42]: 0.9486850418244183

[43]: np.dot(get_embedding('present', model), get_embedding('absent', model))

[43]: 0.8297540600846347

[44]: np.dot(get_embedding('present', model), get_embedding('not present', model))

[44]: 0.8511365230483847
```

Cosine similarity goes from 0 to 1. 0 means embeddings are dissimilar, 1 means the embeddings are similar.
Here, it did not get negation right, as the two embeddings are grouped as similar even though they are opposites.

Seems to be fixed in recent OpenAI Embedding Models!

Negation of values (text-embedding-3-large)

```
[31]: model = 'text-embedding-3-large'

[32]: np.dot(get_embedding('have', model), get_embedding('do not have', model))

[32]: 0.537687984526523

[33]: np.dot(get_embedding('Jonathan was present', model), get_embedding('Jonathan was absent', model))

[33]: 0.7664889465115847

[34]: np.dot(get_embedding('present', model), get_embedding('absent', model))

[34]: 0.38757813415983455

[36]: np.dot(get_embedding('present', model), get_embedding('not present', model))

[36]: 0.48815064811640596
```

Shorter text fares better for embeddings

- Shorter text chunks have fewer similar tokens, leading to less similar embeddings
- Perhaps multiple hierarchies could be used for embedding text chunks

```
model = 'text-embedding-3-large'
```

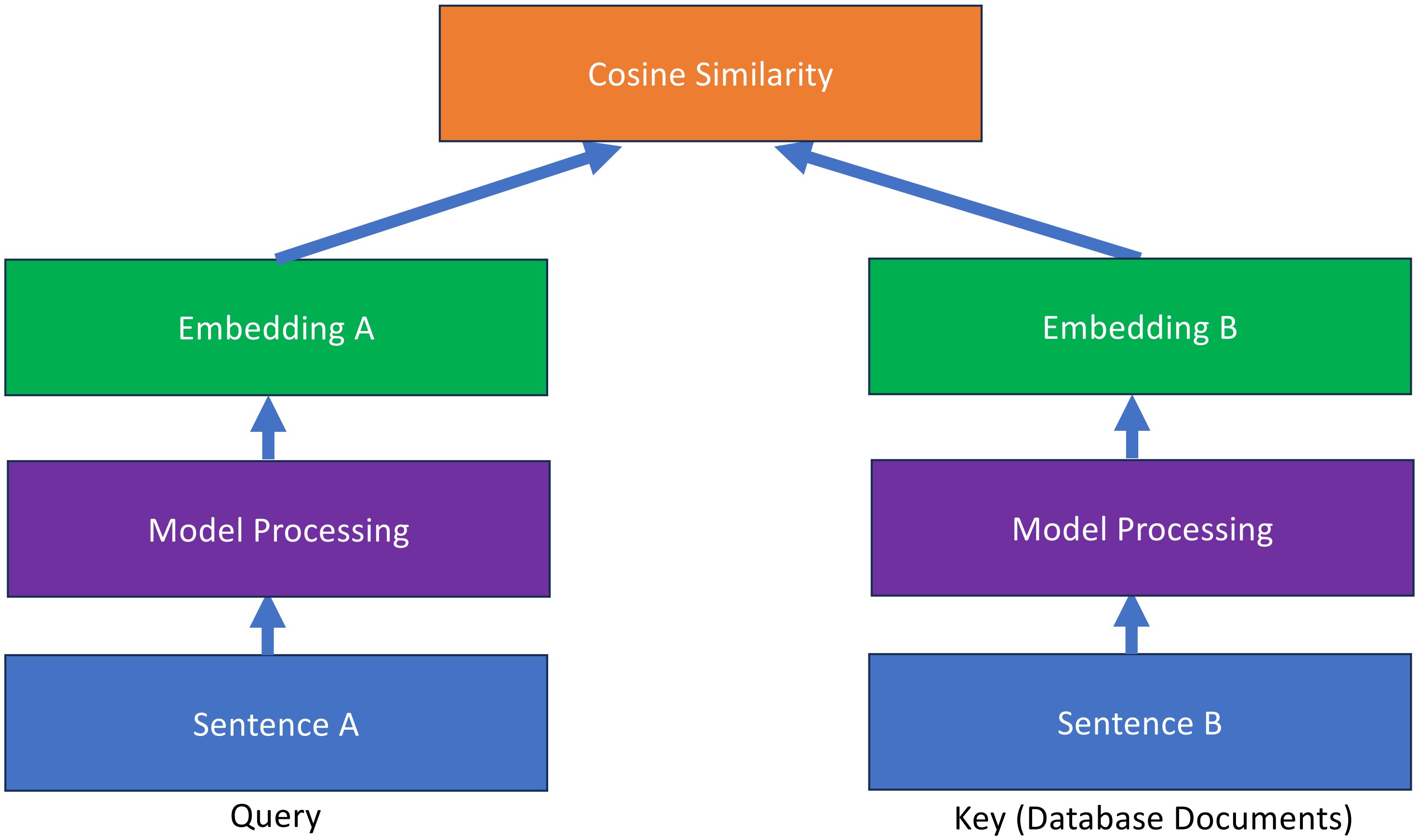
```
text1 = "John went to the supermarket. Peter went to the gym. Mary went to the garden."
text2 = "John went to the airport. Peter went to the gym. Mary went to the garden."
np.dot(get_embedding(text1, model), get_embedding(text2, model))
```

```
0.9297407654620802
```

```
text1 = "John went to the supermarket."
text2 = "John went to the airport."
np.dot(get_embedding(text1, model), get_embedding(text2, model))
```

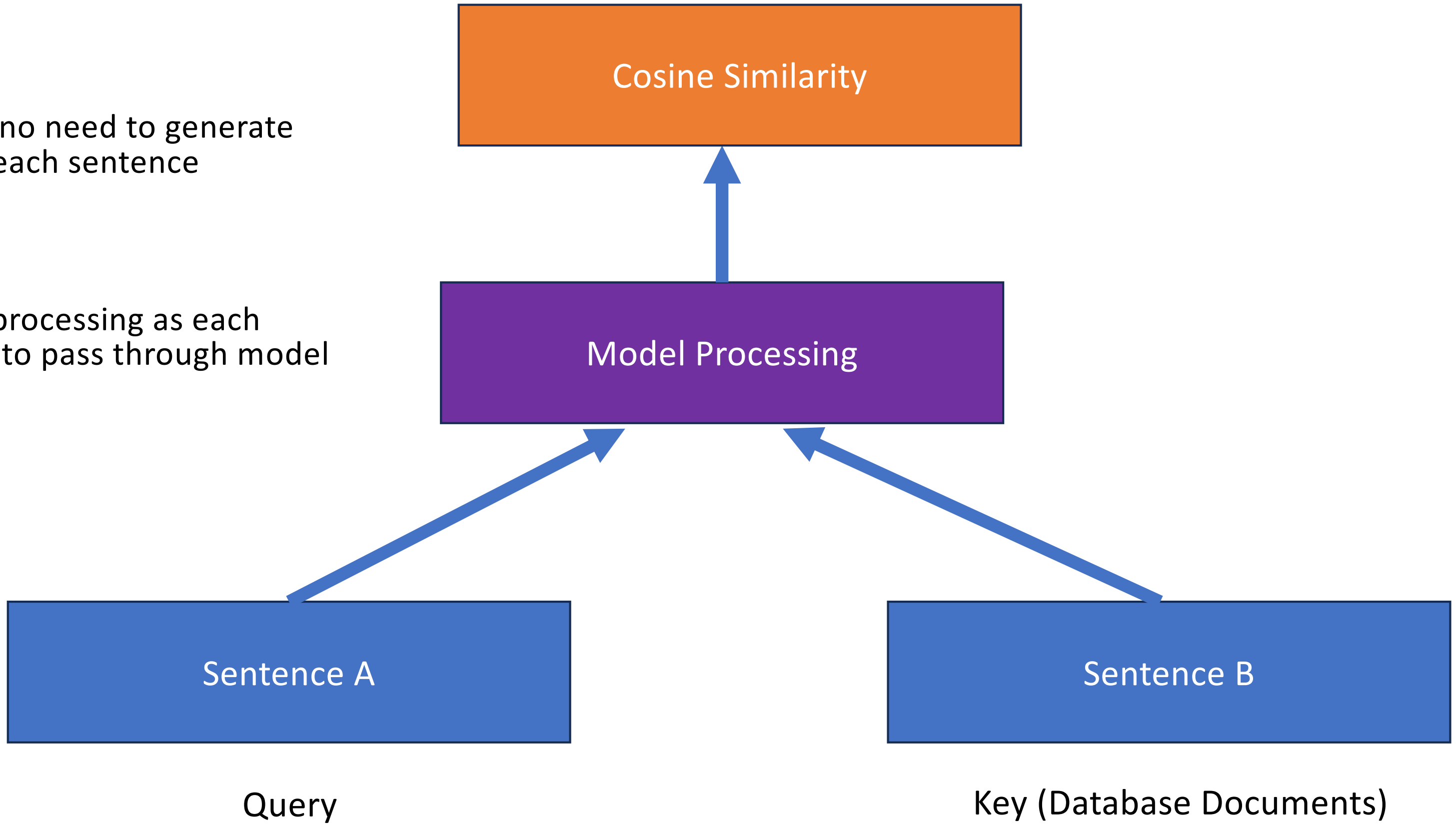
```
0.6328660633894473
```

How Retrieval Augmented Generation is typically done



How Similarity can be done with late interaction models

- Benefits
  - Less lossiness as no need to generate embeddings for each sentence
- Cons
  - More latency in processing as each query-key needs to pass through model
- Examples:
  - Cohere Reranker



Mapping of embedding space by generating hypothetical data

- Key intuition:
  - Embedding space only represents one slice of meaning
  - **Hypothetical Document Embeddings (HyDE)**: Try to change the meaning of query embedding by mapping it to various abstraction spaces

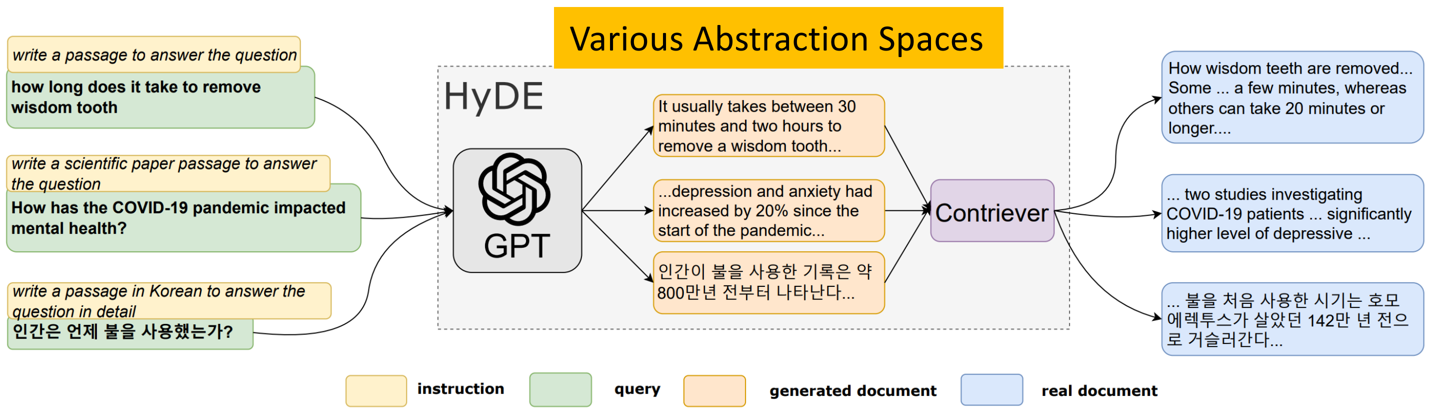
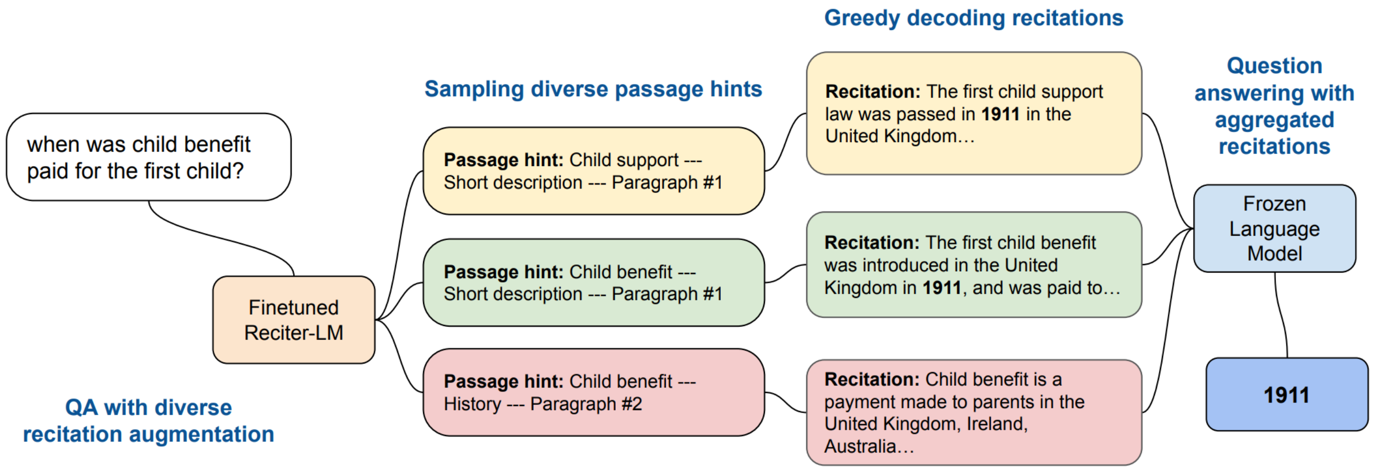


Figure 1: An illustration of the HyDE model. Documents snippets are shown. HyDE serves all types of queries without changing the underlying GPT-3 and Contriever/mContriever models.

Precise Zero-Shot Dense Retrieval without Relevance Labels. Gao et. al. 2022

Mapping of embedding space by hinting

- Hint the LLM differently for diverse outputs
- Due to context dependency, output generation will be skewed according to hint, allowing for more diverse sampling
- No external embedding – recitations directly aggregated into context of QA LLM



Recitation Augmented Language Models. Sun et al. 2023

My idea: Context-dependent embeddings

- Can we change the embedding meaning by just prepending a context to the text before embedding?
- Or how about modifying the text chunk itself based on context?
- Context can even be inferred from one of multiple categories based on use case

Approach 1: Prepending context

[140]:
```
model = 'text-embedding-3-large'
```

[141]:
```
np.dot(get_embedding('I went to the bank', model), get_embedding('I went to the river', model))
```

[141]: 0.5897037575817885

[142]:
```
np.dot(get_embedding('I went to the bank', model), get_embedding('I went to get money', model))
```

[142]: 0.8018812180559

[143]:
```
np.dot(get_embedding('Context: water. I went to the bank', model),
       get_embedding('Context: water. I went to the river', model))
```

[143]: 0.7513987620212337

[144]:
```
np.dot(get_embedding('Context: water. I went to the bank', model),
       get_embedding('Context: water. I went to get money', model))
```

[144]: 0.8789128046393934

Last one should be dissimilar
Embeddings have this issue that text with chunks of similar tokens have similar embeddings

Approach 2: Modify text based on context

```
def text_conversion(context, text):
    return chat(f'''Context is {context}.
Refine text based on context without changing the text's meaning.
Some parts of the text may have more meaning based on context, highlight those.
If unable to refine, output original text''', text)
```

```
text_conversion('water', 'I went to the bank')
```

```
'I went to the riverbank.'
```

```
I went to the bank
I went to the river
Embedding similarity 0.5897400164080131
I went to the riverbank.
I went to the river.
Embedding similarity 0.8759913799435554
```

Context: Water

```
I went to the bank
I went to get money
Embedding similarity 0.8018812180559
I went to the riverbank.
I went to withdraw money.
Embedding similarity 0.5029410724258297
```

Food for thought:
- We need not compare with just one contextual abstraction space.
- We can pick many potential abstraction spaces for query and key.
- We can pre-store these context-dependent text chunks beforehand to save compute!

```
I went to the bank
I went to the river
Embedding similarity 0.589747492674073
I visited the bank.
I went to the river.
Embedding similarity 0.4956905762475252
```

Context: Finance

```
I went to the bank
I went to get money
Embedding similarity 0.8019449821612953
I visited the bank.
I went to withdraw cash.
Embedding similarity 0.6819600807173446
```

Questions to Ponder

- How many dimensions are good for an embedding?

- Should we do similarity search using cosine similarity by embeddings, or by feeding everything into a Transformer to ask for similarity?

- Are there any limitations in the training process for embeddings, and how do we rectify them?
  - How do you think OpenAI solved the negation issue in embeddings?