Neural Information Retrieval

[DAT640] Information Retrieval and Text Mining

Weronika Lajewska
University of Stavanger

03.10.2022

In this module

1. Text representations for ranking
2. Dense retrieval

Text representations for ranking

Text representations for ranking

- Recap on ranking and text representations
- Transition from sparse to dense embeddings
- Static word embeddings
- Contextualized word embeddings

Recap on ranking

- **Ranking** is a core of the search engine. It calculates scores for documents using a *ranking algorithm* based on a *retrieval model* and generates a ranked list of documents for the user's query.
- **Retrieval function** estimates the relevance of documents in the collections w.r.t. the input query q (so that the highest-scoring ones can be returned as retrieval results)
- In principle, the relevance score is computed for each document in the collection. In practice, we're only interested in the top-k results for each query.

Recap on text representation: Discrete (sparse) representation

- Sparse representation has the length of the vocabulary and each element of the vector corresponds to a term in the vocabulary
- The *i*-th component of the vector encodes a discrete value, e.g. the presence/absence of a word (0/1) in the represented text (query or document)
- Sparse representations are based on the bag of words (BOW) model and the position of the word in the text is ignored in the representation
- Sparse representations can be efficiently stored in an inverted index

ID	term
1	discrete
2	sparse
3	document
4	text
5	word
6	this
	...

Vocabulary

$d =$ "this is sparse text"

$d = [t_6, OOV, t_2, t_4]$

$\vec{d} = \langle 0, 1, 0, 1, 0, 1, \dots \rangle$

Recap on text representation: Learning-to-Rank features

- Richer query and document representation can be built by exploiting additional relevance signals, such as PageRank, SPAM score, clicks, etc.
- Learning-to-Rank is implemented as a two-step retrieval with initial ranking (retrieving top-N candidate documents) and re-ranking (creating feature vectors and re-ranking top-N candidates)
- We distinguish three classes of features:
  - query features, e.g. query length, query type, number of named entities in the query
  - document features, e.g. URL length, number of inlinks, PageRank score
  - query-document features, e.g. retrieval score of a given document field

Recap on text representation: Learning-to-Rank features - cont.

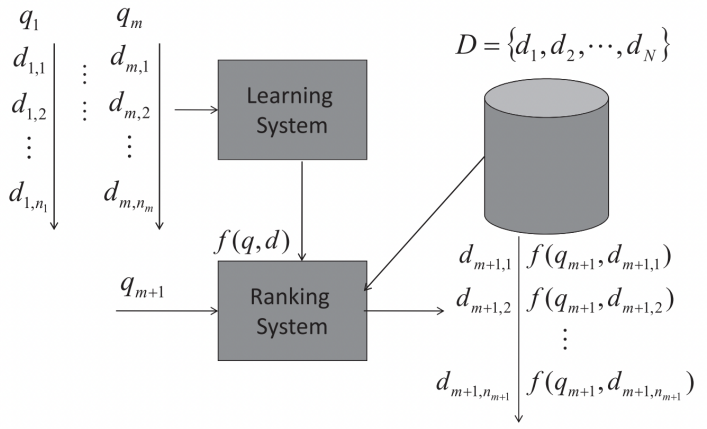


Figure: Learning to rank architecture for document retrieval[1]

[1] https://search.ieice.org/bin/pdf_link.php?category=D&lang=E&year=2011&fname=e94-d_10_1854&abst=

Text representations for ranking

- Recap on ranking and text representations
- Transition from sparse to dense embeddings
- Static word embeddings
- Contextualized word embeddings

Word-context matrix[2]

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Word-context matrix is a co-occurence matrix with W rows (words) and C columns (contexts), where f_{ij} gives the number of times word w_i occurs in context c_j
We distinguish different types of similarity:

- first-order co-occurence - words appears nearby each other, e.g., *write* is a first-order associate of *book* or *poem*
- second-order co-occurence - words have similar neighbors, e.g., *write* is a second-order associate of words like *say* or *remark*

[2] `https://web.stanford.edu/~jurafsky/slp3/`

Pointwise Mutual Information

- *Pointwise mutual information (PMI)* is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = log_2 \frac{P(x,y)}{P(x)P(y)}$$

- Then the PMI between a target word *w* and a context word *c* (first-order co-occurence) is defined as:

$$PMI(w,c) = log_2 \frac{P(w,c)}{P(w)P(c)}$$

- The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently

PPMI matrix - sparse representation

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous. Thus, it is more common to use Positive PMI (PPMI) defined as $PPMI(w, c) = max(PMI(w, c), 0)$.

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Figure: The PPMI matrix showing the association between words and context words[3]

[3] https://web.stanford.edu/~jurafsky/slp3/

Singular Value Decomposition (SVD)

- An *NxM* matrix X can be factorized into three matrices:

$$X = U\Sigma V^T$$

- where:
  - U (left singular vectors) is an *NxM* unitary matrix
  - Σ is an *MxM* diagonal matrix, where diagonal entries are eigenvalues (all positive, sorted from large to small values)
  - V^T (right singular vectors) is an *MxM* unitary matrix

PPMI + SVD

- By applying SVD to PPMI, keeping only top k eigenvalues in Σ, and setting the rest to 0 we can truncate the U and V^T, resulting in U_k and V_k^T
- U_k and V_k^T are new low-dimensional word vectors
- Both, U_k and V_k^T, can be treated as dense word embeddings

Text representations for ranking

- Recap on ranking and text representations
- Transition from sparse to dense embeddings
- Static word embeddings
- Contextualized word embeddings

Distributional semantics

You shall know a word by the company it keeps.

(J. R. Firth, A synopsis of linguistic theory (1957))

Distributional semantics

- Words that occur in the same context tend to have a similar meaning, e.g. book, volume, novel
- The meaning of words can be inferred by their usage together with other words in existing texts
- The *context* of a word is the set of words that appear within a fixed-size window. They are called *context-words*.

Distributional representation

- A word (token) can be represented as a vector of d dimensions
- d is much smaller that the size of the vocabulary
- This d-dimensional vector is called *distributional representation* or *word embedding*

Characteristics of word embeddings

- Components of word embeddings are rarely 0: they are real numbers, and can also have negative values
- Components of distributional representation are not mutually exclusive. More than one component can be "active" at the same time.

$\vec{w} =$

0.15	0.07	0.83	0.46	...	0.02

- Word embeddings are often referred to as *dense representations*

Word embeddings - visualization

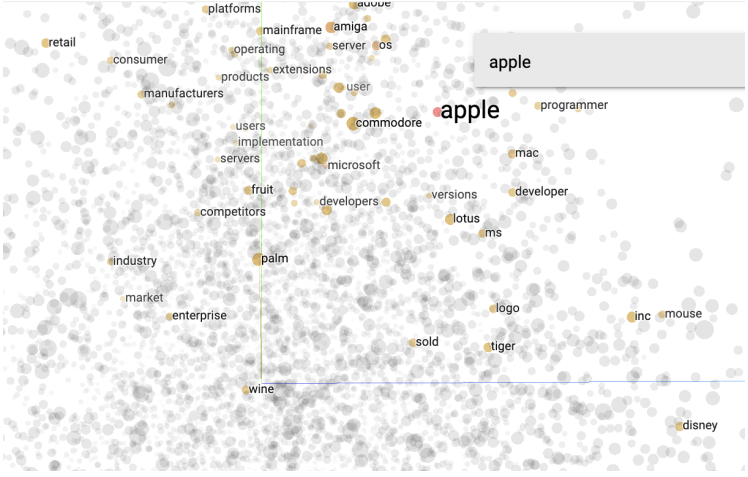
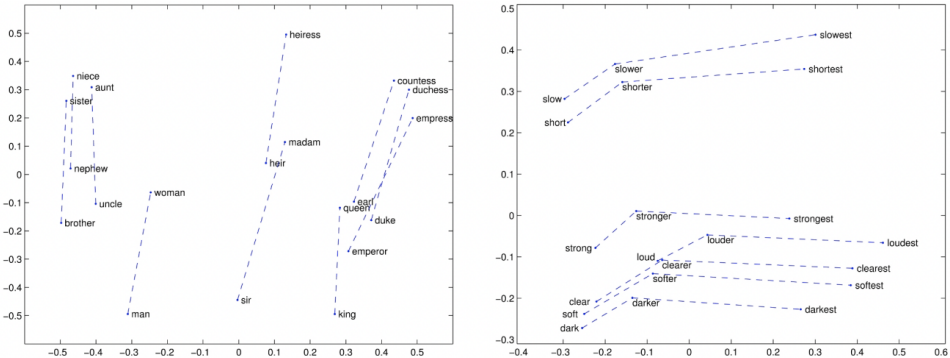


Figure: Tensorflow Embedding Projector [4]

[4] https://projector.tensorflow.org/

Analogy problem[5]



Analogy/relational similarity: Embeddings are able to capture relational meanings. Analogy problem can be generalized to the form *X is to Y as X* is to what?*.

[5] `https://web.stanford.edu/~jurafsky/slp3/`

Bias in word embeddings

- Embeddings reproduce the implicit biases and stereotypes that are latent in the text
- Bias visible in relational similarity:
  - *man:woman::king:queen*
  - *father:mother::doctor:nurse*
- Allocational harm - situation when system allocates resources unfairly to different groups
- Representational harm - situation when system demean or ignore some social groups
- Embeddings don't only reflect the statistics of their input, but they also amplify bias $\rightarrow$biased terms become more biased in embedding space than they were in the input text statistics
- Debiasing is still an open problem

Recap: n-gram Language Model

- Language modeling is a task of predicting a word given a context:

$$P(word|context)$$

- Formal definition: Given a sequence of words $x_1, x_2, ..., x_t$, a language model calculates the probability distribution of next word x_{t+1} over all words in vocabulary

$$P(x_{t+1}|x_t, x_{t-1}, ..., x_1)$$

- A *n-gram* is a chunk of *n* consecutive words
- A *n-gram language model* collects frequency statistics of different *n*-grams in a corpus, and uses these to calculate probabilities
- *Neural n-gram language model* predicts the co-occurrence probabilities
- Language models can be used to generate text by sampling from the probability distributions

word2vec: static word embeddings

- Static word embeddings are global representations of the words, i.e., there is a single fixed word embedding for each term in the vocabulary. Static word embeddings map terms with multiple senses into an average or most common sense representation based on the training data used to compute the vectors.
- Word embeddings are a by-product of neural language models
- We can use a neural language model to create a word embedding, by using the model to predict the probability of appearance of a context-word in a window around the given word

word2vec: skip-gram

- Intuition:
  - Treat the target word and a neighboring context word as positive examples
  - Randomly sample other words in the lexicon to get negative samples
  - Use logistic regression to train a classifier to distinguish those two cases
  - Use the learned weights as the embeddings
- Skip-gram trains a logistic regression classifier to compute the probability that two words are 'likely to occur nearby in text'. This probability is computed from the dot product between the embeddings for the two words.

word2vec: skip-gram

- Maximize the probability of true context words w_{t-m}, w_{t-m+1}, ..., w_{t-1}, w_{t+1}, ..., w_{t+m-1}, w_{t+m} for each target word w_t:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j}|w_t;\theta)$$

- Negative Log Likelihood:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j}|w_t;\theta)$$

word2vec: negative sampling

The skip-gram model tries to shift embeddings so that the target embeddings are closer to (have a higher dot product with) context embeddings for nearby words and further from (lower dot product with) context embeddings for noise words that don't occur nearby

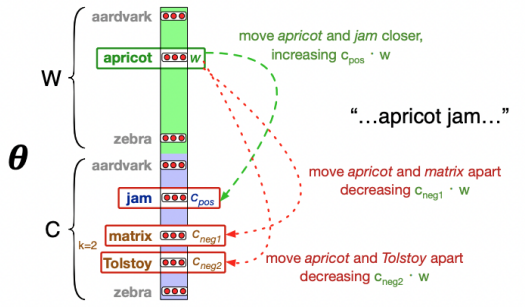


Figure: The skip-gram model [a]

[a] https://web.stanford.edu/~jurafsky/slp3/

word2vec: architecture

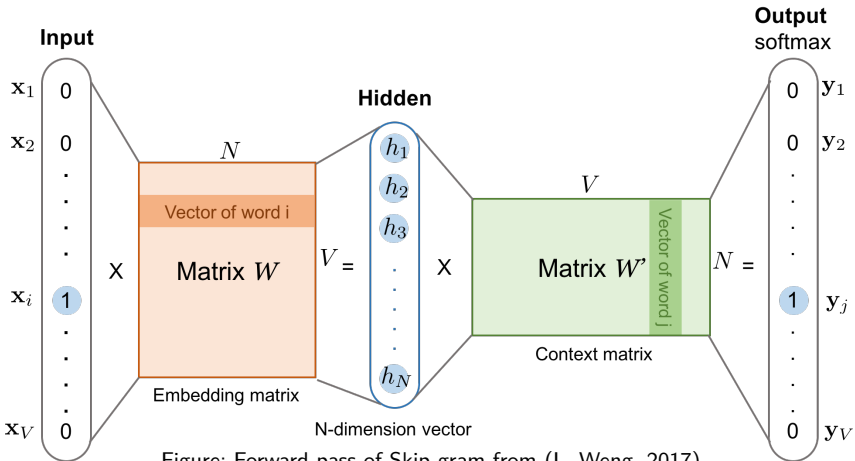


Figure: Forward pass of Skip-gram from (L. Weng, 2017).

Exercise

E9-1 Word embeddings with word2vec

Text representations for ranking

- Recap on ranking and text representations
- Transition from sparse to dense embeddings
- Static word embeddings
- Contextualized word embeddings

Contextualized word embeddings

- Contextualized word embeddings are local representations of the terms which benefit from the context. Each term in the vocabulary is associated with a different representation vector every time it appears in a document, depending on the surrounding tokens.
- The most popular contextualized word embeddings are learned with deep neural networks such as ELMO (Embeddings from Language Model), BERT (the Bidirectional Encoder Representations from Transformers), and RoBERTa (the Robustly Optimized BERT Approach)

Dense retrieval

Dense retrieval

- Dense retrieval overview
- Negative sampling
- Embedding index
- Single representation systems
- Multiple representations systems

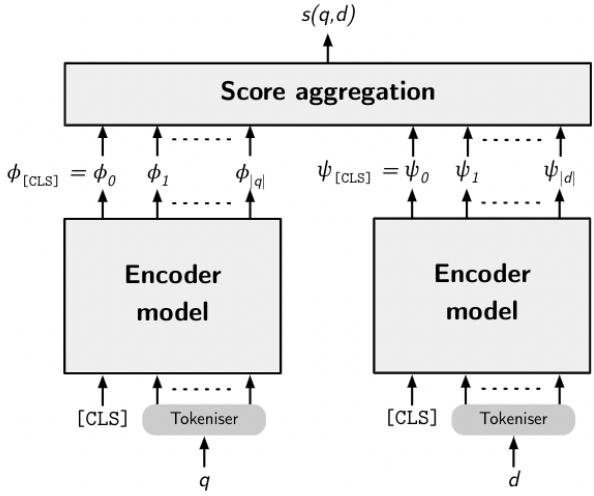
Recap on sparse retrieval models

- Advantages of BM25:
  - It is fast and effective
- Disadvantages of BM25:
  - Bag-of-words representation disregards word order
  - Context of words is not taken into consideration
  - It is an exact term-matching algorithm, meaning that it cannot leverage semantic similarity between words
  - Potential vocabulary mismatch

Dense Retrieval (Representation-focused Systems)

- Dense retrieval aims to match texts in a continuous representation space learned via deep neural networks
- Dense retrieval is characterized by fully learnable representation
- Dense retrieval calculates the relevance score using similarities in a learned embedding space
- Similarity function can be cosine or dot product
- The same representation model is used to encode a query and a document
- The effectiveness of dense retrieval resides in learning a good representation space that maps query and relevant documents together while separating irrelevant ones.

Representation-focused Systems[6]



[6] https://arxiv.org/pdf/2207.13443.pdf

Query and documents representations

- Query representation and document representations are learned independently
- Representations of all documents in the collection can be pre-computed and stored in advance
- During query processing, only the query representation is computed
- Representation-based systems are able to identify the relevant documents among all documents in the collection, not only the ones that are dependent on the query
- Dense retrieval takes into consideration the semantics of the terms in a query and is not limited to an exact match of terms

Dense Retrieval Ranking Architecture

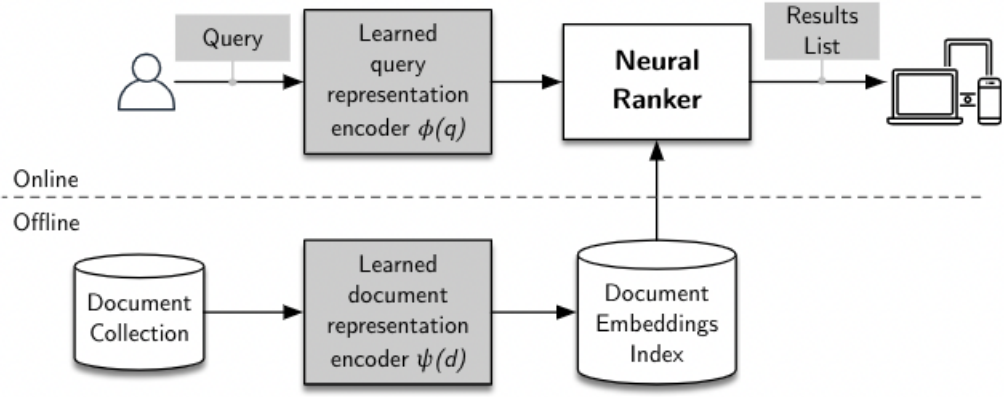


Figure: Dense retrieval architecture for representation-focused neural IR systems[7]

[7] https://arxiv.org/pdf/2207.13443.pdf

Main challenges

Question

How to train a model to distinguish between relevant documents and irrelevant documents?

Question

How to search through a massive index of document representations?

Main challenges

- How to train a model to distinguish between relevant documents and irrelevant documents? $\rightarrow$**Negative sampling**
- How to search through a massive index of document representations?

Dense retrieval

- Dense retrieval overview
- Negative sampling
- Embedding index
- Single representation systems
- Multiple representations systems

Negative sampling

- Most likely we have several relevant documents and millions of irrelevant ones in the collection
- The model needs some negative examples to be able to differentiate between correct matches and incorrect ones
- Choosing meaningful irrelevant documents is a non-trivial task

Fine-tuning representation-focused systems with contrastive learning

- The fine-tuning of a bi-encoder corresponds to learning an appropriate inner-product function suitable for the ranking task, i.e., for relevance scoring
- The goal is to learn a latent vector space for query and document representations where a query and its relevant documents are closer than the query and its non-relevant documents

Negative sampling

- *Random negatives* - any document from the corpus is considered irrelevant with equal probability
- *In-batch local negatives* - during training, the queries are randomly aggregated into batches of size *N*. If passage P1 is a good match for query Q1, it is a negative example for the remaining queries in the batch Q2, Q3, ..., QN.
- The main problem with the random negatives and in-batch local negatives is that they are not really informative for the model. They are easily distinguishable which leads to a weaker signal sent to the model.
- We need negative examples that are hard to distinguish from the relevant documents

Negative sampling - hard negatives

- Negative documents can be generated exploiting a classical or trained retrieval system
- Each query is given as input to the retrieval system. The non-relevant documents included in the top-N retrieved documents are treated as negatives.
- We assume to know the relevant documents for the query
- The retrieval system used to mine the negative documents can be a standard BM25, the currently neural model under training or another fine-tuned neural model

Negative sampling - hard negatives

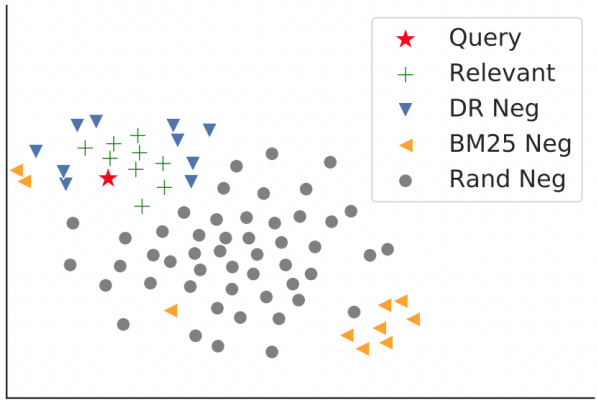


Figure: Representations of query, relevant documents, negative training instances from BM25 (BM25 Neg) or randomly sampled (Rand Neg), and testing negatives (DR Neg) in dense retrieval[8]

[8] https://arxiv.org/abs/2007.00808

Main challenges

- How to train a model to distinguish between relevant documents and irrelevant documents? $\rightarrow$**Negative sampling**
- How to search through a massive index of document representations? $\rightarrow$**Embedding index**

Dense retrieval

- Dense retrieval overview
- Negative sampling
- Embedding index
- Single representation systems
- Multiple representations systems

Embedding index

- *Embedding index* stores the pre-computed document embeddings and provides a search algorithm that, given a query embedding, efficiently finds the document embedding with the *maximum inner product* (MIP)
- There are different approaches for embedding index: flat index, nearest neighbor (NN) index, approximate nearest neighbor (ANN) index

Flat index

- Documents embeddings are stored explicitly in the flat index
- An exhaustive search is performed to find the document embedding with the maximum inner product
- The inner product search can be defined as:

$$\psi^* = \underset{\psi \in \Psi}{\text{argmax}} < \psi, \phi >$$

  where $\phi \in \mathbb{R}^l$ denotes a query embedding, $\Psi = \psi_1, ..., \psi_n$ denotes a set of *n* document embeddings, with $\psi_i \in \mathbb{R}^l$ for $i = 1, ..., n$, and ψ^* denotes the maximum inner product
- It's particularly inefficient for large document collections

Nearest neighbor index

- A common approach to improve the space and time efficiency of the flat index is to convert the maximum inner product search into a nearest neighbor (NN) search
- The goal of NN search is to find the document embedding ψ^+ defined as:

$$\psi^+ = \underset{\psi \in \Psi}{\text{argmax}} \, ||\phi - \psi||$$

- Many efficient index data structures exist for NN search. To leverage them with embedding indexes, MIP search between embeddings must be adapted to use the Euclidean distance and NN search.

Approximate nearest neighbor index

- The index data structures for exact NN search in low dimensional spaces have been very successful, but they are not efficient with high dimensional data, as in our case, due to the curse of dimensionality
- *Approximate nearest neighbor (ANN) search* is a natural compromise between search accuracy and search speed
- The ANN search approaches can be shared into three families: locality sensitive hashing, quantization and graph approaches

ANN search: Locality sensitive hashing approach

- Main idea: if two embeddings are close together, then after a "projection", using a hash function, these two embeddings will remain close together:
  - for any two embeddings that are close to each other, there is a high probabilityp1that they fall into the same hash bucket
  - for any two embeddings that are far apart, there is a low probability that they fall into the same hash bucket
- A set of random projections defines a family of hash function that can be used to build a data structure for ANN search
- The index is composed of hash tables, each containing concatenated random projections
- Main drawback: the index may require a large number of hash tables to cover most nearest neighbors

ANN search: Vector quantization approach

- Main idea: partitioning the input space Ψ (document embeddings) according to the data distribution:
  - Use k-means clustering algorithm to compute *k centroids* $\mu_1, ..., \mu_k$
  - Use the centroids to partition the input space. The set of centroids is called a *codebook* $M = \mu_1, ..., \mu_k$. A *vector quantiser* $q : \mathbb{R}^l \rightarrow \mathbb{R}^l$ maps a vector ψ to its closest centroid:

$$q(\psi) = \underset{\mu \in M}{\text{argmax}} \, ||\psi - \mu||$$

  - An *Inverted File* (IVF) index is built over *codebook* M and input space Ψ by storing the set of document embeddings in *k partitions* or *inverted lists* $L_1, ..., L_k$

$$L_i = \{\psi \in \Psi : q(\psi) = \mu_i\}$$

ANN search: Vector quantization approach - cont.

- At query processing time, we specify to search for the NN document embeddings in $p > 0$ partitions. If p=k, the search is exhaustive, but if $p < k$, the search is carried out in the partitions whose centroid is closer to the query embedding
- An IVF index does not improve the space consumption, since it still needs to store all document embeddings, but it can reduce the search time depending on the number of partitions processed for each query
- Main drawback: IVF indices can require a large number of centroids. This can be addressed with *product quantisation*.

ANN search: Graph approach

- Main idea: the distances between vectors in a dataset can be efficiently stored in a graph-based data structure called *kNN graph*
- In a kNN graph $G = (V, E)$, each input data$\psi \in \Psi$is represented as a node $v \in V$, and, for its k nearest neighbours, a corresponding edge is added in E
- A search for an approximate nearest neighbor to a given element ψ in a kNN graph can be performed with *greedy heuristic search*
- In greedy heuristic search we start from a predefined entry node, we visit the graph one node at a time, and we keep on finding the closest node to ψ among the unvisited neighbour nodes. The search terminates when there is no improvement in the current NN candidate.

Dense retrieval

- Dense retrieval overview
- Negative sampling
- Embedding index
- Single representation systems
- Multiple representations systems

Different approaches for representation-focused systems

- Single representation systems - queries and documents are represented with a single embedding
- Multiple representations systems - queries and/or documents are represented with more than one embedding

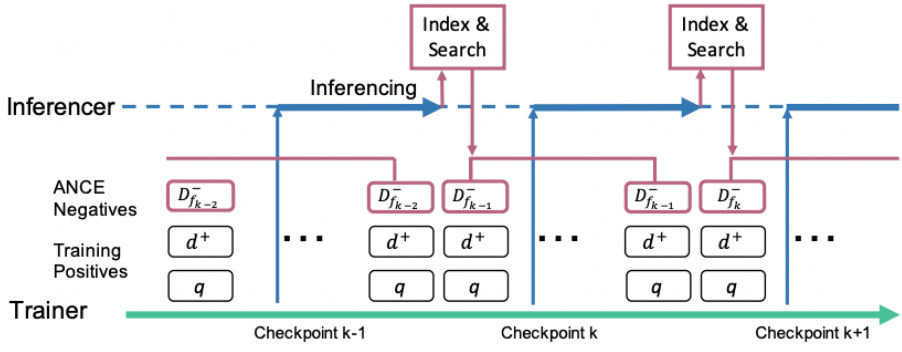
Single representation

- Query representations and document representations are computed in the same latent vector space
- The representation functions are computed through fine-tuned encoder-only sequence-to-sequence models such as BERT
- Using these single representations, the relevance score between a query and a document is computed as a dot product of their output embeddings (corresponding to [CLS] tokens returned by the encoders)
- There are several different single-representation systems, e.g., DPR, ANCE, STAR

ANCE

- Approximate nearest neighbor Negative Contrastive Learning (ANCE) is a contrastive learning mechanism for dense retrieval
- ANCE constructs global hard negatives using the being-optimized dense retrieval model retrieve from the entire corpus. It samples negatives from the top retrieved documents via the dense retriever model from the ANN index.
- ANCE uses an asynchronously updated ANN index of the collection representation. ANCE maintains an Inferencer that parallelly computes the document encodings with a recent checkpoint from the being optimized DR model, and refresh the ANN index used for negative sampling once it finishes, to keep up with the model training.
- ANCE can be used to train any dense retrieval model. The originally introduced system uses BERT Encoder, dot product similarity and negative log likelihood loss.

ANCE asynchronous training[9]



The Trainer learns the representation using negatives from the ANN index. The Inferencer uses a recent checkpoint to update the representation of documents in the corpus and once finished, refreshes the ANN index with most up-to-date encodings.

[9] https://arxiv.org/abs/2007.00808

Exercise

E9-2 Dense retrieval with ANCE

Dense retrieval

- Dense retrieval overview
- Negative sampling
- Embedding index
- Single representation systems
- Multiple representations systems

Dense Retrieval - multiple representations

- In contrast to single representation dense retrieval where a representation is assumed to incorporate the meaning of an entire text, multiple representations dense retrieval exploits more than a single embedding to represent a given text
- Multiple representations dense retrieval allows a richer semantic representation of the content
- Instead of using just the first output embedding (corresponding to the [CLS] token) to encode a document, this approach makes use of more output embeddings

Ranking pipeline architecture for multiple representation systems

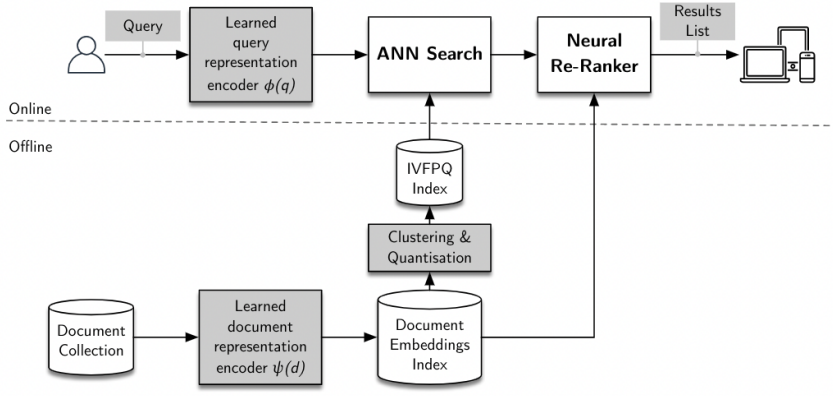


Figure: Ranking pipeline architecture for multiple representation systems[10]

[10] https://arxiv.org/pdf/2207.13443.pdf

ColBERT

- ColBERT uses all $1 + |d|$ output embeddings used to represent a document (of length d) - one output embedding per document token, including the [CLS] special token
- Similarly, query (of length q) is represented with $1 + |q|$ output embeddings
- ColBERT's late interaction scoring, also called *sum maxsim*, performs an all-to-all computation: each query embedding is dot-multiplied with every document embedding, and then the maximum computed dot products for each query embedding are summed up:

$$[\psi_0, \psi_1, ...] = Encoder(q)$$
$$[\phi_0, \phi_1, ...] = Encoder(d)$$
$$s(q, d) = \sum_{i=0}^{|q|} \max_{j=0,..,|d|} \psi_j \cdot \phi_j$$

ColBERT general architecture

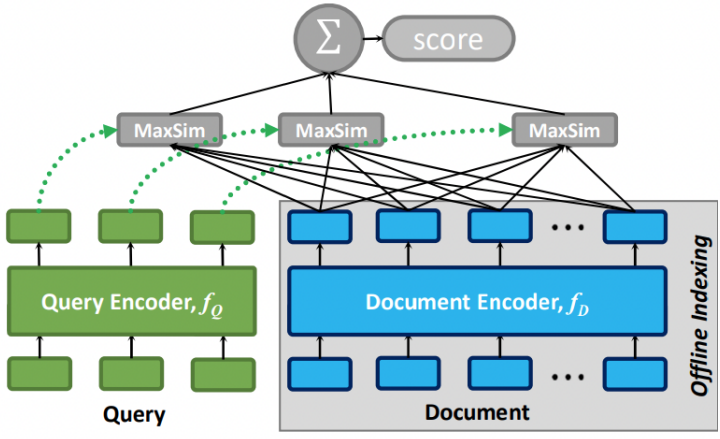


Figure: ColBERT architecture.[11]

[11] https://dl.acm.org/doi/abs/10.1145/3397271.3401075

Useful frameworks for dense retrieval

- FAISS framework for embedding indices (includes implementations of flat, LSH, IVF, PG indices)[12]
- Open-source search engines:
  - Elasticsearch[13]
  - Lucene[14]
  - Vespa[15]

[12] https://github.com/facebookresearch/faiss
[13] https://elastic.co
[14] https://lucene.apache.org4https://vespa.ai34
[15] https://vespa.ai

Summary

- Text representations for ranking
  - Recap on ranking and text representations
  - Transition from sparse to dense embeddings
  - Static word embeddings
  - Contextualized word embeddings
- Dense retrieval
  - Dense retrieval
  - Dense retrieval overview
  - Negative sampling
  - Embedding index
  - Single representation systems
  - Multiple representations systems

Reading

- *Lecture Notes on Neural Information Retrieval*, Nicola Tonellotto[16]
- *Vector Semantics and Embeddings*, Daniel Jurafsky and James H. Martin [17]

[16] https://arxiv.org/pdf/2207.13443.pdf
[17] https://web.stanford.edu/~jurafsky/slp3/