Embedding and Indexing Methods for RAG Application

This document aims at briefly introducing the different indexing and embedding methods that were used in the RAG project. Embedding refers to representing text (words, sentences, or documents) as dense numerical vectors that capture semantic meaning, allowing for similarity comparisons based on context. Indexing, on the other hand, is the process of organizing and storing documents in a way that enables efficient lookup, typically using keywords or token positions, as in inverted indexes. Not all these methods were tested in the framework, but enough code and documentation should be present to allow for their inclusion in the final RAG.

1 Word2Vec : Continuous Bag of Word and Skip-Gram

Word2Vec is a technique in natural language processing used to learn distributed representations of words, also known as word embeddings. Instead of representing words as sparse one-hot vectors, Word2Vec maps each word to a dense, continuous vector in a lower-dimensional space where semantically similar words are placed closer together. This approach captures both syntactic and semantic relationships between words and is foundational in modern NLP.

Word2Vec consists of two main model architectures: Continuous Bag of Words (CBOW) and Skip-gram. Both models are trained using a shallow neural network with one hidden layer, but they approach the learning task in opposite directions. The CBOW model predicts a target word based on its surrounding context words. It does this by representing each context word as a one-hot vector, projecting them into an embedding space, averaging the vectors, and feeding the result into a softmax layer to predict the most likely center word. On the other hand, the Skip-gram model reverses this task by predicting context words based on a single center word. For each context word, the model creates a separate training pair, treating it as an independent prediction problem

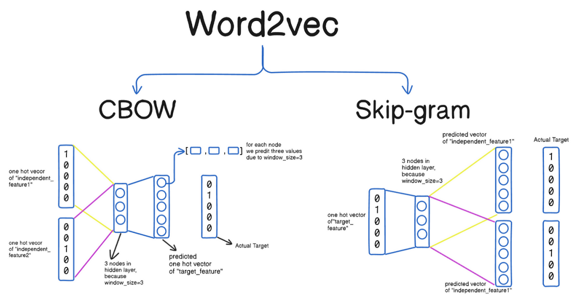
Word2vec
CBOW
Skip-gram
one hot vecor of "independent_feature1"
one hot vecor of "independent_feature2"
for each node we predit three values due to window_size=3
3 nodes in hidden layer, because window_size=3
predicted one hot vector of "target_feature"
Actual Target
predicted vector of "independent_feature1"
Actual Target
3 nodes in hidden layer, because window_size=3
one hot vector of"target_feature"
predicted vector of "independent_feature1"

Figure 1: Word2Vec methods, and difference between CBOW and Skip-Gram

2 Okapi BM25

Okapi BM25 is a ranking function used to estimate the relevance of documents to a given search query. It is based on the bag-of-words model, meaning it treats a document as a collection of individual terms, ignoring grammar and word order. BM25 improves upon traditional term-frequency approaches by introducing a probabilistic framework and tuning parameters that balance term frequency and document length. The main idea behind BM25 is that if a term appears more often in a document, it is more likely to be relevant—but with diminishing returns. That is, seeing a term 10 times is more informative than seeing it once, but seeing it 100 times isn't 10 times more useful than seeing it 10 times. BM25 applies a saturation function to the raw term frequency to account for this. The BM25 score of a document D regarding a query Q is calculated using the following formula:

$$\text{score}(D, Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl}}$$

where $f(q_i, D)$ is the frequency of the term q_i in the document D, $|D|$ the lengh of the document D (in terms of worlds) and $avgdl$ is the average document lengh in the collection. The variables k_1 and b are hyperparameters, usually with $b = 0.75$ and $k_1 \in [1.2\ ; 2.0]$.

The inverse document frequency component (IDF) in BM25 ensures that common terms, which appear in many documents, are given less weight than rare, more discriminative terms. Unlike TF-IDF, which uses a logarithmic IDF, BM25 uses a more robust form that also accounts for term saturation. BM25 is effective because it balances three important factors: how often a query term appears in a document, how common or rare the term is across the whole collection, and how long the document is relative to the average. This balance allows it to rank documents in a way that aligns well with human judgments of relevance, making it a strong baseline for information retrieval tasks, such as search engines and recommendation systems. However this method is reliant on exact term matching, meaning that synonyms and spelling mistakes will be considered as different terms.

3 BERT : Bidirectional Encoder Representations from Transformers

BERT is a language representation model that processes text using the Transformer encoder architecture. It operates by taking a sentence or a pair of sentences as input and representing every word in that input as a contextualized vector. This means each word is embedded based not only on its individual meaning but also on the words that appear around it—both before and after—thanks to the model's bidirectional attention mechanism.

The input to BERT starts with tokenization. Each word or subword is broken down into tokens using a method called WordPiece. Special tokens are added: [CLS] is placed at the beginning of the sequence to represent the overall meaning of the sentence or pair of sentences, and [SEP] is used to separate two sentences. BERT then converts these tokens into embeddings, which include token embeddings, positional embeddings (to preserve word order), and segment embeddings (to differentiate between two input sentences, if applicable). These combined embeddings are passed into the Transformer encoder.

Within the encoder, BERT uses self-attention to model relationships between all tokens. In each layer, every token attends to every other token in the sequence, and learns how much attention it should pay to each of them based on their content. This allows BERT to capture dependencies and nuances across entire sentences, not just local context. Multiple layers of self-attention and feedforward operations refine these representations, resulting in a final embedding for each token that reflects its full linguistic context.

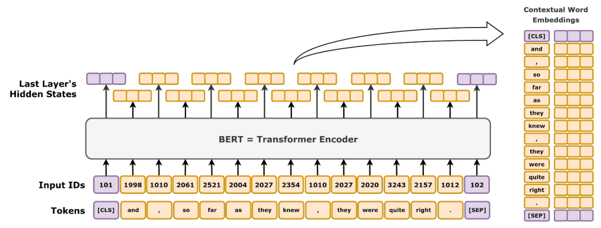
Contextual Word Embeddings
Last Layer's Hidden States
BERT = Transformer Encoder
Input IDs
101 1998 1010 2061 2521 2004 2027 2354 1010 2027 2020 3243 2157 1012 102
Tokens
[CLS] and , so far as they knew , they were quite right . [SEP]

Figure 2: BERT embedding

BERT is pretrained using two tasks. The first is masked language modeling, where some tokens in the input are randomly replaced with a [MASK] token, and the model is trained to predict the original tokens. This forces the model to learn from both the left and right context of each masked word. The second task is next sentence prediction, where the model receives two sentence segments and learns to predict whether the second follows the first in the original text. These objectives guide BERT to understand both word-level and sentence-level relationships.

Once pretrained, BERT can be adapted to downstream tasks by adding a small neural layer on top of the model and fine-tuning it using labeled data. For example, for a classification task, the [CLS] token's final embedding can be passed to a softmax layer to produce class probabilities. For token-level tasks like named entity recognition, each token's embedding can be used for individual classification.

Overall, BERT functions as a general-purpose language understanding model. Its key contribution is the ability to represent words in context, using deep bidirectional processing and attention mechanisms to capture meaning across entire sequences.

4 DPR : Dense Passage Retrieval

Dense Passage Retrieval (DPR) is a neural information retrieval method designed to improve search by moving beyond exact keyword matching. It uses dense vector representations, or embeddings, to encode both user queries and candidate passages, enabling retrieval based on semantic similarity rather than surface-level word overlap. The core idea is to train two separate encoders—one for questions and one for passages—usually based on a pre-trained transformer model such as BERT. These encoders transform the input text into fixed-size vectors within the same embedding space.

During training, DPR is typically optimized using a contrastive learning approach. For each question, the model is given a relevant (positive) passage and several irrelevant (negative) passages. It learns to produce embeddings such that the dot product (or cosine similarity) between the question and its relevant passage is higher than with the negatives. After training, the passage encoder is used to encode a large corpus of documents into embeddings, which are stored in a vector index. At retrieval time, the question encoder encodes a user's query, and the system retrieves the most similar passages by comparing the query embedding to all document embeddings using nearest neighbor search.

This dense retrieval approach allows DPR to capture semantic meaning and retrieve relevant content even when the exact terms in the question don't appear in the passage. As a result, DPR is especially powerful for open-domain question answering tasks, where it can find informative evidence from large text collections that keyword-based systems like BM25 may miss.

5 GloVe: Global Vectors for Word Representation

GloVe (Global Vectors for Word Representation) is a word embedding method that learns dense vector representations of words by leveraging global co-occurrence statistics from a large text corpus. The method is based on the idea that the meaning of a word can be inferred from the distribution of other words it co-occurs with.

First, a co-occurrence matrix X is constructed, where each entry X_{ij} records how often word j appears in the context of word i. The goal is to learn word vectors w_i and context word vectors $\tilde{w}_j$ such that their dot product, plus bias terms, approximates the logarithm of the observed co-occurrence:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log X_{ij}.$$

The learning objective minimizes the squared difference between the left and right sides of this relationship, weighted by a function $f(X_{ij})$ that limits the influence of extremely frequent co-occurrences. The cost function is:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) \left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where V is the vocabulary size, b_i and $\tilde{b}_j$ are bias terms, and $f(X_{ij})$ is the weighting function defined as:

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}} \right)^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

with typical values $x_{\max} \approx 100$ and $\alpha \approx 0.75$. This formulation encourages words that appear in similar contexts to have similar vector representations, capturing both semantic and syntactic relationships.