The chunks are converted into high-dimensional vectors using an embedding model.

How are the chunks converted into sentence vectors?

Sentences are first broken down into individual tokens, which are then represented as indices in a vocabulary (**using a one-hot representation**).

These index representations are then converted into vectors.

How are these embeddings initialized in the first place?

Initially, embeddings are **randomly initialized**.

As the model (Both LLM and Embedding model) continues to train, something interesting happens.

Tokens that are similar in meaning start to **cluster together** in the multidimensional space.

As a result, the similarity between the vectors of similar words becomes very high, **close to 1**.

This is how words are represented in multi-dimensional space.

How are sentences or chunks represented as vectors?

By **Changing the Final Linear layer in the Encoder transformer** model like BERT, we can get the sentence Vector or sentence embeddings.

**BERT**

The BERT stands for **Bidirectional Encoder Representations from Transformers**.

The Encoder layer is composed of two main components: **self-attention and feed-forward network layers.** These layers work together to help the model understand the entire sentence or chunk of text.

LLM (Decoder architecture) is an **autoregressive model**, which means the next token is predicted based on the current context.

By applying a causal mask in the attention layer, LLM obtains the Autoregressive property.

**When Causal masks are applied, the current token can only attend to previous tokens, not the next tokens in the sequence**, which helps LLM to predict the next token based on the current context.

But BERT is different, **In BERT the causal mask is not applied**, so the current token attends all the tokens in the sequence (chunks, query). **It attends previous tokens and the next tokens**. Tokens attend other tokens in both directions in the sequence. That’s why this transformer method got its name, Bidirectional Encoder Representation.

The Sentence embedding vectors have contextual information, positional information, and the relationship between tokens in the sequence.

The Output vectors of the BERT have rich information about the sequence. We use the mean pool method to combine all sentence vectors into a single vector. This sentence vector comprehensively represents the sequence/chunks/queries.

A screenshot of a graph

Description automatically generated

The encoder layer output matrix is the size of (N, 768), where N is chunks or query length (No of Tokens) and 768 is the model dimension. This Matrix is then multiplied by a linear layer weight matrix and averaged, resulting in a sentence vector of size 768 that effectively captures the information of the entire input.

How the search happens?

There 2 methods (**cosine similarity and Euclidean or Manhattan distance**) often used in finding chunks similar to the query.

Cosine similarity helps us to calculate the similarity between two vectors.

This cosine similarity ranges from **-1 to 1 ([-1,1])**.

**Where 1 means similar or identical vectors, 0 means they are orthogonal, and -1 means they are complete opposite vectors.**

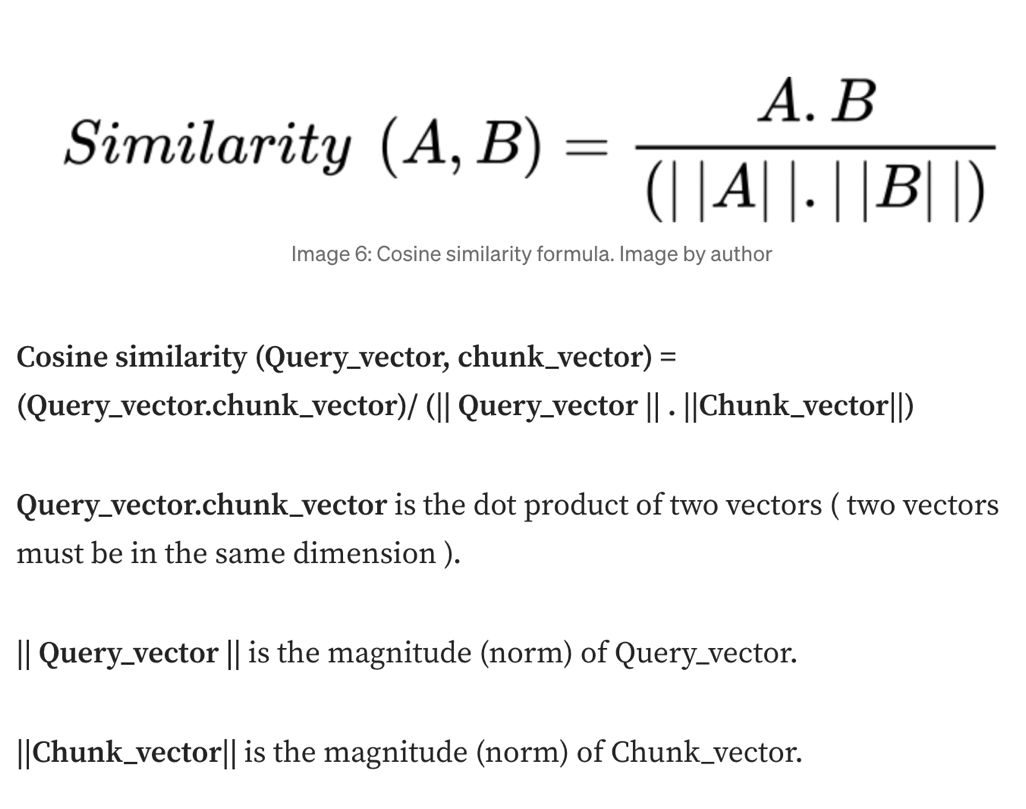
A graph of a vector line

Description automatically generated with medium confidence

**For two vectors to be considered similar, the angle between them (θ) needs to be close to 0 degrees.**

**The closer the angle gets to 0, the more similar the vectors are, and the cosine of that angle (cos(θ)) will approach 1.**

How is the Cosine similarity calculated for Multi-dimensional vectors?



Note: Euclidean distance or Manhattan distance helps us calculate the distance between two vectors in the Multidimensional space (Similar to KNN).

A smaller distance means the two vectors are close in multi-dimensional space.

Vector search methods

* Two of the methods: Naive Search, and HNSW.

**Naive search (flat)**

In this method, the query is searched against all the chunks in the vector index.

There will be N (number of chunks) number of similarity computations in this method.

Even though this method yields better results**, this method is inefficient, computationally expensive, and has high latency.**

This method is a kind of brute force to find all the queries nearest neighbors in the multi-dimensional space.

At the End, Top k high similarity chunks are retrieved and given to LLM as Input with Prompt.

**HNSW (Hierarchical Navigable small worlds)**

The Metric we usually care about in similarity search is Recall.

**Recall score means among all the true values, how many values are predicted as True.**

NSW and HNSW, **approximate Nearest neighbors**. It may not give all the nearest neighbors (Chunks), But it is computationally efficient and yields good results.

HNSW powers Qdrant, open-source vector database. And Qdrant only uses HNSW as a vector index algorithm.

HNSW is an Evolution of the **Navigable small worlds (NSW) algorithm** for the approximate nearest neighbors (Finding most similar chunks), which works based on the concept of **SIX Degrees of separation.**

What SIX Degrees of separation means is that people all over the world are all connected by six degrees of separation. This concept is Proven.

**Navigable Small World**

The NSW algorithm builds a graph that (Similar to social media connections) connects close vectors with each other but keeps the total number of connections small (To mimic the Six degrees of separation concept).

When a query is given, the process begins by **randomly selecting one chunk vector, also known as a node.**

The next step is to calculate the similarity score for this node.

From there, the process moves on to the connected nodes. Again, the similarity scores are calculated for these nodes. The node with the better similarity score is then selected, and its connected nodes are evaluated in the same way.

This process continues until no better similarity score can be found.

In other words, the algorithm keeps exploring connected nodes until it reaches a point where the similarity scores no longer improve.

This process repeated.

We repeat the search with randomly chosen starting points and keep the top k among all the visited nodes.

In the end, the Top K selected chunks are given to LLM to generate the augmented Response.

**HNSW (Hierarchical Navigable small worlds) is the combination of *Navigable small worlds* and *Skip list* data structure.**

In HNSW, there are multiple layers, The Top layer is sparse — a very minimum number of nodes will be there. The bottom layer is Dense.

When the Query is given, it searches against a randomly chosen node in the Top Layer (Layer 3 — Sparse), calculates cosine similarity, and moves to its connected nodes.

When it finds the Local best--high similarity scores compared to visited nodes (Each layer same NSW process), It Moves to the next layer (Layer 2).

In Layer 2, the cosine similarity is calculated to the node that is connected to the previous layer. Then the similarity scores are calculated for nodes that are connected, and when it finds the Local best, it moves to the Next Layer.

This will happen for all layers. Then Top k nodes are selected from visited nodes.

We repeat the search with randomly chosen starting points in the top layer and keep the top k among all the visited nodes. This is how the Nodes (chunks) are efficiently retrieved.

Response generation in RAG

The Retrieved Top K Chunks are given to LLM with Prompts.

Then, **Using Retrieved chunks, Query, and system Prompt, the LLM augments** the Input and Generate the response to the query.

The query is searched against nodes in the vector index and retrieves the top k similarity nodes. The retrieved Nodes are given to LLM with Prompt and Query to generate the response.