Creating Text Embedding Models

Creating text-embedding models in large language models" refers to the process of designing or using a model that converts text into **numeric vectors** (embedding's) that capture meaning. These vectors allow software systems to:

- compare semantic similarity
- perform search & retrieval
- cluster documents
- feed structured inputs into larger LLM pipelines

A text embedding model takes text (a word, sentence, or document) and produces a vector such as:

[0.12, -0.87, 0.45, ...]

These vectors represent the semantic meaning of the text.

Texts with similar meaning produce vectors that is close in vector space.

Examples:

"car" and "automobile" → close vectors

"dog" and "quantum physics" → far apart

How Embedding Models Relate to LLMs

Modern large language models (LLMs), like GPT, LLaMA, and others, internally use embedding's to represent text.

An embedding model is often: a separate small neural network trained specifically for embedding's or a part of a larger LLM, extracted or fine-tuned for embedding's. Many LLM providers offer embedding-specific versions of their main models:

OpenAI: text-embedding-3-large, text-embedding-3-small.

Meta: LLaMA model variants fine-tuned for embeddings

Google: models like Gecko (for embeddings)

Contrastive Learning

Contrastive learning in Large Language Models (LLMs) refers to training strategies where the model learns better representations of text by comparing examples—pulling similar text representations closer and pushing dissimilar ones apart.

While traditional LLM training uses next-token prediction (causal language modeling), contrastive learning is increasingly used to improve embedding's, alignment, reasoning, and robustness.

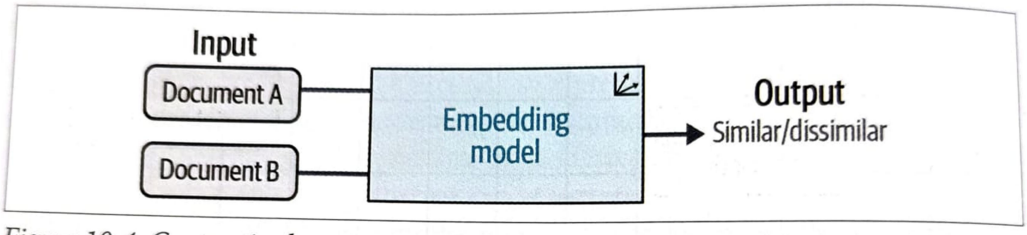

Figure 10-4. Contrastive learning aims to teach an embedding model whether documents are similar or dissimilar. It does so by presenting groups of documents to a model that are similar or dissimilar to a certain degree.

1. Improving Embeddings

LLMs often produce vector embedding's for tasks like search, retrieval, or clustering.
Contrastive learning helps embedding's capture semantic similarity.

Positive pairs might be:

- paraphrases
- question ↔ answer
- similar sentences from augmentation

Negative pairs might be:

- unrelated sentences
- contradictory sentences

This works for models like **E5**, **GTE**, **SimCSE**, and many retrieval-augmented LLMs.

2. Self-Supervised Contrastive Objectives

LLMs can generate their own training pairs. Examples:

SimCSE for LLMs

- Positive pair: two different dropout passes of the same sentence
- Negative pair: other sentences in the batch

This improves sentence embedding's without labels.

Contrastive Decoding

Models learn:

- anchor sentence = original
- positive = perturbed version with same meaning
- negative = perturbed version that changes meaning

3. Contrastive Learning for Instruction Following

LLMs can be aligned using **preference pairs**, e.g.:

- Good answer (positive)
- Bad answer (negative)

Contrastive loss pushes the model to prefer the positive.

This is used in:

- **Direct Preference Optimization (DPO)**
- **Contrastive Preference Optimization (CPO)**

These methods compare embedding's or log-probabilities of *good vs. bad* responses.

4. Contrastive Tuning for Robustness

Applied to make LLMs robust to:

- adversarial prompts
- hallucination
- paraphrased instructions

Example:
Contrast the model's output against a harmful or incorrect output.

5. Multimodal LLMs (e.g., CLIP-like training)

LLMs connected to image/audio/video encoders often use contrastive learning:

- Text encoder ↔ Image encoder
- Positive: caption matches an image
- Negative: mismatched pairs

@Why Contrastive Learning Helps LLMs…

Creates high-quality semantic embedding's

Improves retrieval-augmented generation (RAG) performance

Makes responses more consistent and aligned

Helps in preference optimization (LLM alignment)

Reduces hallucinations by encouraging consistent representations

SBERT

SBERT (Sentence-BERT) in Large Language Models

SBERT is a modification of BERT designed to produce sentence embedding's fixed-size vector representations that capture the semantic meaning of a sentence. While BERT (and many LLMs) can understand text well, they aren't optimized to produce efficient, meaningful sentence-level embedding's out of the box.

SBERT = BERT + Siamese network architecture + contrastive training

[Siamese network architecture]

A Siamese network architecture in the context of LLMs refers to using two identical transformer encoders (with shared weights) to convert two different text inputs into comparable embedding's, so that their similarity can be measured.

Instead, it is a training setup (dual-encoder) built on top of an LLM encoder.

A Siamese architecture for LLMs uses:

Text A → Encoder → Embedding A

Text B → Encoder → Embedding B

The same LLM encoder (same weights) processes A and B.

Then the embeddings are compared using:

cosine similarity

dot product

contrastive loss

triplet loss

So, a Siamese setup is basically a pair of identical LLM encoders used for similarity learning.

It was designed to:

Generate dense semantic vectors.

Enable semantic similarity, clustering, and retrieval.

Run much faster than BERT for pairwise comparisons.

How SBERT Works:

SBERT usually consists of:

Two identical transformer encoders (shared weights → "Siamese")

Sentences encoded into embeddings

A similarity function (cosine similarity)

Concept	SBERT	Modern LLMs
Purpose	Sentence embedding's	Text generation & understanding
Training	Contrastive, NLI	Next-token prediction
Output	Fixed semantic vectors	Probabilistic token distribution
Use case	Semantic similarity, retrieval, clustering	Chabot's, reasoning, generation
Relation	Used *with* LLMs in RAG	Needs SBERT-like training for embedding's

A solution to this overhead is to generate embeddings from a BERT model by averaging its output layer or using the `[CLS]` token. This, however, has shown to be worse than simply averaging word vectors, like GloVe.[4]

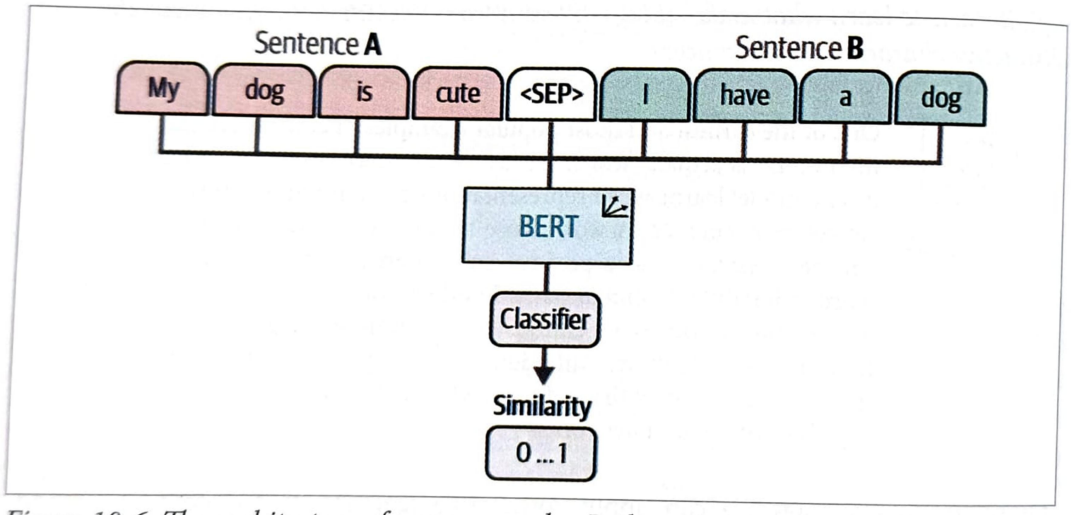

Figure 10-6. The architecture of a cross-encoder. Both sentences are concatenated, separated with a <SEP> token, and fed to the model simultaneously.

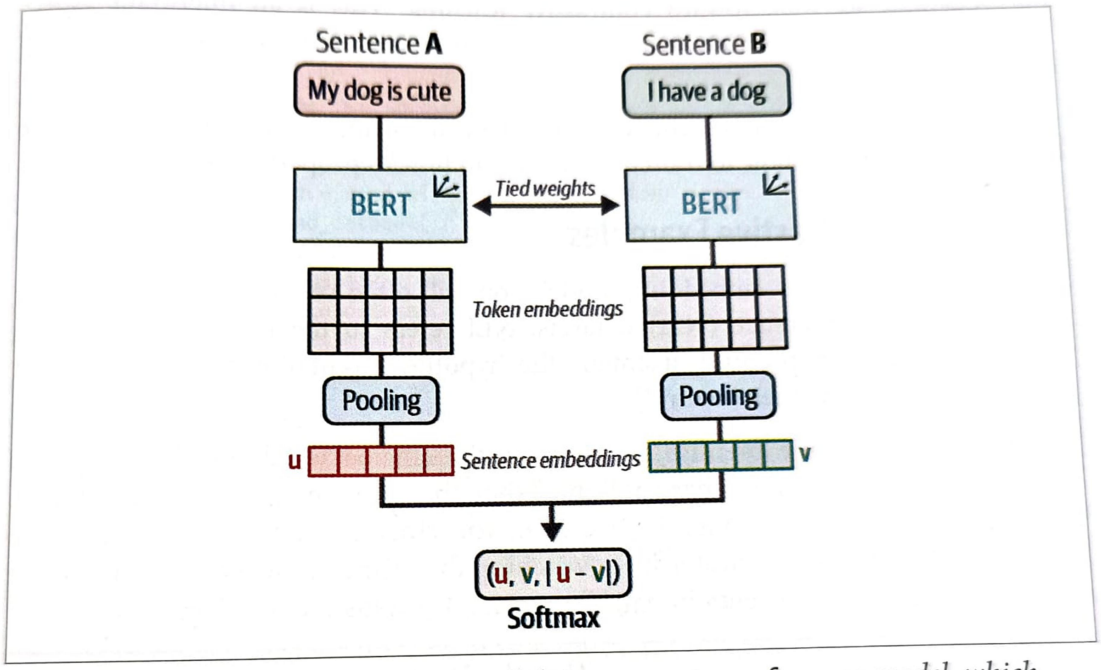

Figure 10-7. The architecture of the original `sentence-transformers` model, which leverages a Siamese network, also called a bi-encoder.

- The original sentence-transformers model uses a Siamese (bi-encoder) architecture.
- Training pairs of sentences involves loss functions that significantly affect model performance.
- During training, embedding's from each sentence are concatenated along with their difference.
- This combined embedding is then optimized using a **softmax classifier**.
- Bi-encoders are fast and create accurate sentence embedding's.
- Cross-encoders, although slower, usually provide better performance but do not generate embedding's.
- Like cross-encoders, bi-encoders use contrastive learning to optimize similarity or dissimilarity between sentence pairs.
- Through this optimization, the model learns what features make sentences similar or different.