Learning Embeddings: Part II

CS 594: Vector Databases

Abolfazl Asudeh, asudeh@uic.edu

Fall 2026

Outline

GloVe: Global Vectors for Word Representation

Contextual & Sentence Embeddings

Image Embeddings

Jointly Trained Embeddings: CLIP

GloVe: Global Vectors for Word Representation

Big picture: Word2Vec vs. GloVe

Both Word2Vec and GloVe learn one vector per word. They differ in how they use word co-occurrence information.

Word2Vec

- Trains a prediction task.
- Given a target word, predict nearby context words.
- Uses local context windows during training.
- The co-occurrence signal is implicit in the prediction objective.

GloVe

- Builds a global co-occurrence matrix first.
- Uses counts X_{ij} across the whole corpus.
- Directly fits word vectors to these counts.
- The co-occurrence signal is explicit in the objective.

The co-occurrence matrix

GloVe starts from a corpus and builds a word–context co-occurrence matrix.

Co-occurrence count

X_{ij} = number of times word j appears in the context of word i

The context is usually defined by a fixed-size window around each word. The matrix summarizes these windows over the corpus.

	data	science	spark
data	0	42	8
science	42	0	15
spark	8	15	0

Two embeddings per word

GloVe assigns two vectors to each vocabulary item.

Word vector

$w_i \in \mathbb{R}^d$

Vector for word i when it is the center word.

Context vector

$\tilde{w}_j \in \mathbb{R}^d$

Vector for word j when it appears in the context.

The model also learns scalar bias terms b_i and $\tilde{b}_j$.

After training, many implementations use $w_i + \tilde{w}_i$ as the final word embedding, or use only w_i.

What does GloVe fit?

For each observed pair (i, j), GloVe tries to make the dot product match the log co-occurrence count.

Model equation

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

Interpretation:

- If i and j co-occur often, $\log X_{ij}$ is large.
- The model increases $w_i^\top \tilde{w}_j$ for such pairs.
- Bias terms capture word frequency effects that should not all be encoded in the vector geometry.

GloVe loss function

GloVe minimizes a weighted least-squares objective over word–context pairs.

Objective

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

Here V is the vocabulary size. The weighting function $f(X_{ij})$ controls how much each co-occurrence count affects training.

Common choice:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max}, \\ 1 & \text{otherwise.} \end{cases}$$

Usually $\alpha = 3/4$ in the original GloVe paper.

Why use a weighting function?

Raw co-occurrence counts are highly skewed.

- Very frequent words produce many large counts.
- Very rare pairs can be noisy.
- Zero counts are not useful because $\log 0$ is undefined.

The weighting function gives small but nonzero counts some influence, while preventing very frequent pairs from dominating the loss.

Training signal: fit reliable co-occurrence structure, not every raw count equally.

What should students remember?

For a vector database course, the main points are high level.

- GloVe is a static word embedding method.
- Each word gets one dense vector after training.
- Unlike Word2Vec, GloVe explicitly uses a global co-occurrence matrix.
- The learned geometry reflects corpus-level word co-occurrence statistics.
- Once trained, GloVe vectors can be stored, indexed, and searched like other embeddings.

Limitation

The embedding for a word is fixed. The word “bank” has the same vector in “river bank” and “bank account.”

Contextual Embeddings

Big picture: static vs. contextual embeddings

Word2Vec and GloVe learn one vector per word.

$$E(\text{bank}) = w_{\text{bank}}$$

This is useful, but it ignores how the word is used in a sentence.

"I deposited money in the bank"

"We sat on the river bank"

In contextual embeddings, the vector depends on the surrounding text.

$$E(\text{bank}, \text{context})$$

Context changes the representation

The same word can produce different vectors in different sentences.

$$h^{(1)}_{\text{bank}} = E(\text{bank}, \text{“deposited money in the bank”})$$

$$h^{(2)}_{\text{bank}} = E(\text{bank}, \text{“sat on the river bank”})$$

Usually,

$$h^{(1)}_{\text{bank}} \neq h^{(2)}_{\text{bank}}$$

The representation is no longer only a property of the word. It is **a property of the word in context**.

How contextual embeddings are produced

A contextual model takes the whole sequence as input.

$$(x_1, x_2, \ldots, x_n) \longrightarrow \text{Contextual Encoder} \longrightarrow (h_1, h_2, \ldots, h_n)$$

Each output vector h_i corresponds to token x_i.
Unlike Word2Vec or GloVe, h_i is computed using the other tokens in the sequence.
Transformer-based models such as BERT use attention to combine information from the surrounding context.

From contextual tokens to one text vector

A contextual encoder produces one vector per token.

$$(x_1, x_2, \ldots, x_n) \longrightarrow (h_1, h_2, \ldots, h_n)$$

For vector search, we often want one vector for the whole sentence or document:

$$E(x_1, \ldots, x_n) \in \mathbb{R}^d$$

So we need a way to combine the token-level representations into one text embedding.

Using the [CLS] token

BERT-style models add a special token, [CLS], at the beginning of the sequence.

$$[\mathsf{CLS}], x_1, x_2, \ldots, x_n$$

The encoder produces

$$h_{\mathsf{CLS}}, h_1, h_2, \ldots, h_n$$

The [CLS] vector can be used as a representation of the whole sequence:

$$E(x_1, \ldots, x_n) = h_{\mathsf{CLS}}$$

Intuition: [CLS] has no word meaning of its own. Its output vector is allowed to collect information from the entire sequence.

Pooling token embeddings

Another approach is to combine the contextual token vectors directly.
Mean pooling:

$$E(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} h_i$$

Other choices include max pooling or learned pooling.

But pooling alone is not enough

BERT produces contextual representations, but it was not originally trained so that

$$\cos(E(s_1), E(s_2))$$

directly measures semantic similarity between two sentences.

For vector search, we want

$$E(s_1) \approx E(s_2)$$

when s_1 and s_2 have similar meaning.
Sentence embedding models such as **Sentence-BERT** explicitly train the embedding space for this purpose.

Static vs. contextual embeddings: summary

	Word2Vec / GloVe	Contextual embeddings
Input	word or word pairs	sequence of tokens
Word vector	fixed	depends on context
Output	word embeddings	token or text embeddings
VectorDB use	limited for text search	common for semantic search
Examples	Word2Vec, GloVe	BERT, Sentence-BERT

Image Embeddings

Image embeddings: basic idea

An image embedding model maps an image to a dense vector.

$$I \longrightarrow E_{\text{img}}(I) \in \mathbb{R}^d$$

The goal is that similar images map to nearby vectors.

$$\text{sim}(E_{\text{img}}(I_1), E_{\text{img}}(I_2))$$

How image embeddings are produced

An image encoder converts pixels into a representation.

$$I \longrightarrow \text{Image Encoder} \longrightarrow z_I$$

- The encoder can be a convolutional neural network or a Vision Transformer.
- The vector may come from the final hidden layer, a pooled representation, or a learned projection layer.

What does image similarity mean?

Image similarity depends on training. Two images may be similar because they have:

- similar low-level appearance,
- the same object category,
- similar semantic content,
- or similar labels for a task.

The embedding model defines what similarity means.

Separate image and text embeddings

Suppose we train one encoder for text:

$$T \longrightarrow E_{\text{text}}(T)$$

and another encoder for images:

$$I \longrightarrow E_{\text{img}}(I)$$

Even if both vectors have the same dimension, they are not necessarily comparable.

$$E_{\text{text}}(T), E_{\text{img}}(I) \in \mathbb{R}^d$$

does not imply that they live in the same semantic space.
So text-to-image search needs more than two independent embedding models.

Jointly Trained Embeddings: CLIP

CLIP: image and text in one space

CLIP learns two encoders jointly.

$$I \longrightarrow E_{\text{img}}(I)$$

$$T \longrightarrow E_{\text{text}}(T)$$

The goal is to place matching images and texts close to each other.

$$\text{sim}(E_{\text{img}}(I_i), E_{\text{text}}(T_i))$$

should be high for a matched pair.
For mismatched pairs, the similarity should be lower.

Training signal in CLIP

CLIP is trained on image-caption pairs.

$$(I_1, T_1), (I_2, T_2), \ldots, (I_m, T_m)$$

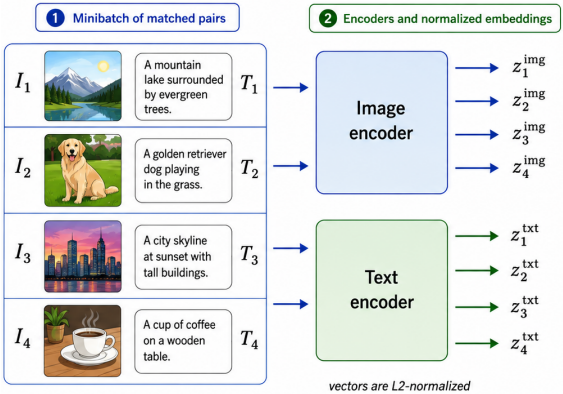
For a minibatch, the model compares every image with every text.

$$S_{ij} = \mathrm{sim}(E_{\mathrm{img}}(I_i), E_{\mathrm{text}}(T_j))$$

The diagonal entries are correct matches: S_{ii}.
The off-diagonal entries are mismatches: $S_{ij}, \quad i \neq j$.

CLIP applies a softmax over these similarities using a temperature parameter τ.
Smaller τ makes the softmax sharper; larger τ makes it softer.

Clip Contrastive Loss (AI-generated)



3 Similarity matrix and contrastive loss

Similarity matrix S

	T_1	T_2	T_3	T_4
I_1	S_{11} ✓	S_{12} –	S_{13} –	S_{14} –
I_2	S_{21} –	S_{22} ✓	S_{23} –	S_{24} –
I_3	S_{31} –	S_{32} –	S_{33} ✓	S_{34} –
I_4	S_{41} –	S_{42} –	S_{43} –	S_{44} ✓

Diagonal = matched pairs; off-diagonal = mismatches in the minibatch

$$\mathcal{L}_{\text{img}\to\text{txt}} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp(S_{ii}/\tau)}{\sum_{j=1}^{m}\exp(S_{ij}/\tau)}$$

$$\mathcal{L}_{\text{txt}\to\text{img}} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp(S_{ii}/\tau)}{\sum_{j=1}^{m}\exp(S_{ji}/\tau)}$$

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}\left(\mathcal{L}_{\text{img}\to\text{txt}} + \mathcal{L}_{\text{txt}\to\text{img}}\right)$$

τ: temperature parameter controlling the sharpness of the softmax

CLIP contrastive loss

For a minibatch of m matched image-text pairs,

$$(I_1, T_1), \ldots, (I_m, T_m),$$

CLIP computes all m^2 image-text similarities:

$$S_{ij} = \frac{E_{\text{img}}(I_i)^\top E_{\text{text}}(T_j)}{\|E_{\text{img}}(I_i)\| \|E_{\text{text}}(T_j)\|}.$$

CLIP contrastive loss (contiued)

For image I_i, the correct text is T_i:

$$L_{\text{img}\to\text{text}} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp(S_{ii}/\tau)}{\sum_{j=1}^{m}\exp(S_{ij}/\tau)}.$$

Similarly, CLIP predicts the correct image for each text:

$$L_{\text{text}\to\text{img}} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp(S_{ii}/\tau)}{\sum_{j=1}^{m}\exp(S_{ji}/\tau)}.$$

$$L_{\text{CLIP}} = \frac{1}{2}\left(L_{\text{img}\to\text{text}} + L_{\text{text}\to\text{img}}\right).$$

Why CLIP matters for vector databases

CLIP makes cross-modal vector search possible.
A text query can retrieve images:

$$q = E_{\text{text}}(\text{“a red sports car”})$$

$$\text{NN}\left(q, \{E_{\text{img}}(I_1), \ldots, E_{\text{img}}(I_n)\}\right)$$

The query vector and stored image vectors are comparable because they were trained in the same space.
This is different from using an image encoder and a text encoder trained separately.

Image embeddings vs. CLIP embeddings

	Image embedding	CLIP-style embedding
Input	image	image or text
Encoder	image encoder	image encoder + text encoder
Training	visual or task labels	matched image-text pairs
Vector space	image-only	shared image-text space
Query type	image query	image or text query
VectorDB use	image similarity search	cross-modal retrieval

References (word/contextual embedding)

1. Pennington, Socher, and Manning. *GloVe: Global Vectors for Word Representation*. EMNLP 2014.
2. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL, 2019.
3. Reimers, N. and Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP-IJCNLP, 2019.
4. Peters, M. E., et al. Deep Contextualized Word Representations. NAACL, 2018.

References (image embedding / clip)

1. He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. CVPR, 2016.
2. Dosovitskiy, A., et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ICLR, 2021.
3. Radford, A., et al. Learning Transferable Visual Models From Natural Language Supervision. ICML, 2021.

Thank you!