Embeddings

From Text to Meaningful Numbers

Recap: Why Chunking Matters

Before embeddings, we **split text into chunks** because:

- Models have **token limits**
- Smaller chunks → **more focused meaning**
- Better retrieval accuracy in search & RAG systems

Now let's see **how chunks become numbers**

What Is an Embedding?

An **embedding** is a **numeric representation of meaning**

- Text → Vector (list of numbers)
- Similar meanings → Similar vectors
- Different meanings → Distant vectors

Computers don't understand words
They understand **numbers**

Example (Conceptual)

Text chunks:

- `"Dog is a pet"`
- `"Cat is an animal"`
- `"Car needs fuel"`

Embeddings (simplified):

```
Dog → [0.12, 0.88, 0.31]
Cat → [0.10, 0.85, 0.29]
Car → [0.91, 0.02, 0.77]
```

Dog & Cat are close
Car is far away

What Embeddings Actually Represent

Each number in an embedding represents **abstract features**, like:

- Topic
- Context
- Intent
- Semantic meaning

These are **NOT human-readable**
You cannot say:

“Dimension 57 = happiness”

They work only in **mathematical space**

Think of Embeddings as Coordinates

Imagine a **meaning space**:

- Every sentence = a point
- Similar sentences = nearby points
- Different topics = far apart

Like a **map of meaning**

"Machine Learning basics"
"Intro to AI" → close
"Cooking pasta" → far

What Happens After Embedding?

Typical pipeline:

1. Chunk text
2. Convert chunks to embeddings
3. Store embeddings in a vector database
4. Embed user query
5. Find **most similar vectors**
6. Retrieve matching chunks

This is the core of **semantic search**

Why Not Compare Words Directly?

Keyword matching fails:

- "car" ≠ "automobile"
- "AI" ≠ "Artificial Intelligence"

Embeddings capture **meaning**, not spelling

That’s why:

“How to train a model” ≈ “Steps for ML training”

Measuring Similarity Between Embeddings

Once we have vectors, we need to answer:

How similar are these two meanings?

Common options:

- Euclidean distance
- Dot product
- **Cosine similarity** (most common)

Why Cosine Similarity Works

Cosine similarity measures:

Angle between vectors, not distance

Why this matters:

- Length of vector ≠ importance
- Direction = meaning

Same direction → similar meaning
Opposite direction → very different meaning

Visual Intuition

- Two vectors pointing the **same way** → cosine ≈ 1
- At 90° → cosine ≈ 0
- Opposite direction → cosine ≈ -1

In practice:

- Values closer to **1** = very similar
- Near **0** = unrelated

Cosine Similarity (Simple Formula)

```
cosine(A, B) = (A · B) / (|A| × |B|)
```

Libraries will handle this for you

What matters:
Angle = meaning similarity

Embedding Dimensions Explained

Embedding dimension =
How many numbers represent one chunk

Common sizes:

- **384**
- **768**
- **1536**

Each number adds more expressive power

Think of Dimensions Like Detail Level

Analogy: Image resolution

Dimensions	Like	Meaning
384	Low-res image	Fast, less detail
768	HD image	Balanced
1536	Ultra-HD	High detail, slower

384 Dimensions

Very fast
Low memory usage
Less semantic nuance

Good for:

- Small projects
- Fast search
- Prototypes
- Mobile / edge systems

768 Dimensions

Balanced accuracy
Moderate speed
Most common choice

Good for:

- Production RAG systems
- Knowledge bases
- Educational projects

1536 Dimensions

High semantic accuracy
Slower search
More storage required

Good for:

- Legal documents
- Medical data
- Research papers
- Large enterprise systems

Why the numbers

Model	Embedding Dimension
MiniLM	384
BERT-base	768
OpenAI `text-embedding-3-small`	1536
OpenAI `text-embedding-3-large`	3072
Word2Vec (classic)	100–300
GloVe	50, 100, 200, 300
Custom research models	Any N

Among these, 384, 768 and 1536 are widely used

Trade-Off: Speed vs Accuracy

Factor	Low Dimensions	High Dimensions
Speed	Fast	Slower
Memory	Low	High
Accuracy	Medium	High
Cost	Low	Higher

There is **no best size**
Only **best for your use case**

Practical Rule of Thumb

- Start with **768**
- Optimize later
- Increase only if:
  - Results feel vague
  - Meanings are very subtle
  - Domain is complex

Bigger ≠ Always better

Chunking + Embeddings Together

Why both matter:

- Chunking controls **context size**
- Embeddings control **semantic understanding**

Bad chunking → bad embeddings
Good chunking → meaningful embeddings

Common Beginner Mistakes

- Using very large chunks
- Assuming embeddings are interpretable
- Choosing max dimensions blindly
- Using keyword search instead of semantic search

Mental Model to Remember

- Chunking = **What to read**
- Embeddings = **How meaning is stored**
- Cosine similarity = **How meaning is compared**
- Dimensions = **How detailed meaning is**

Upcoming Topics

- Vector databases
- Approximate nearest neighbor (ANN)
- RAG architecture
- Embedding models in practice

Key Takeaways

- Embeddings turn text into numbers
- Similar meaning → similar vectors
- Cosine similarity compares direction
- Dimension size affects speed & accuracy
- Choose based on **use case**, not hype