Understanding Transformers

Part 1: The Encoder's Input Embedding Layer

Before a Transformer can process text, it needs to convert words into numbers. This is where **input embeddings** come in—the crucial first step that transforms language into a format neural networks can understand.

What Is an Embedding?

An **embedding** transforms words into dense numerical vectors that capture meaning and relationships—not just random numbers, but carefully structured representations.

Words with similar meanings end up close together in this numerical space, allowing the model to understand semantic relationships.

cat

[0.1, -0.5, 0.3, 0.2]

dog

[0.15, -0.45, 0.28, 0.25]

car

[0.9, 0.2, -0.1, -0.3]

Notice how "cat" and "dog" have similar vectors (related concepts), while "car" is quite different.

Why Do We Need Embeddings?

Simple Numbering Fails

Assigning cat=1, dog=2, car=3 doesn't capture meaning or show that "cat" and "dog" are related animals.

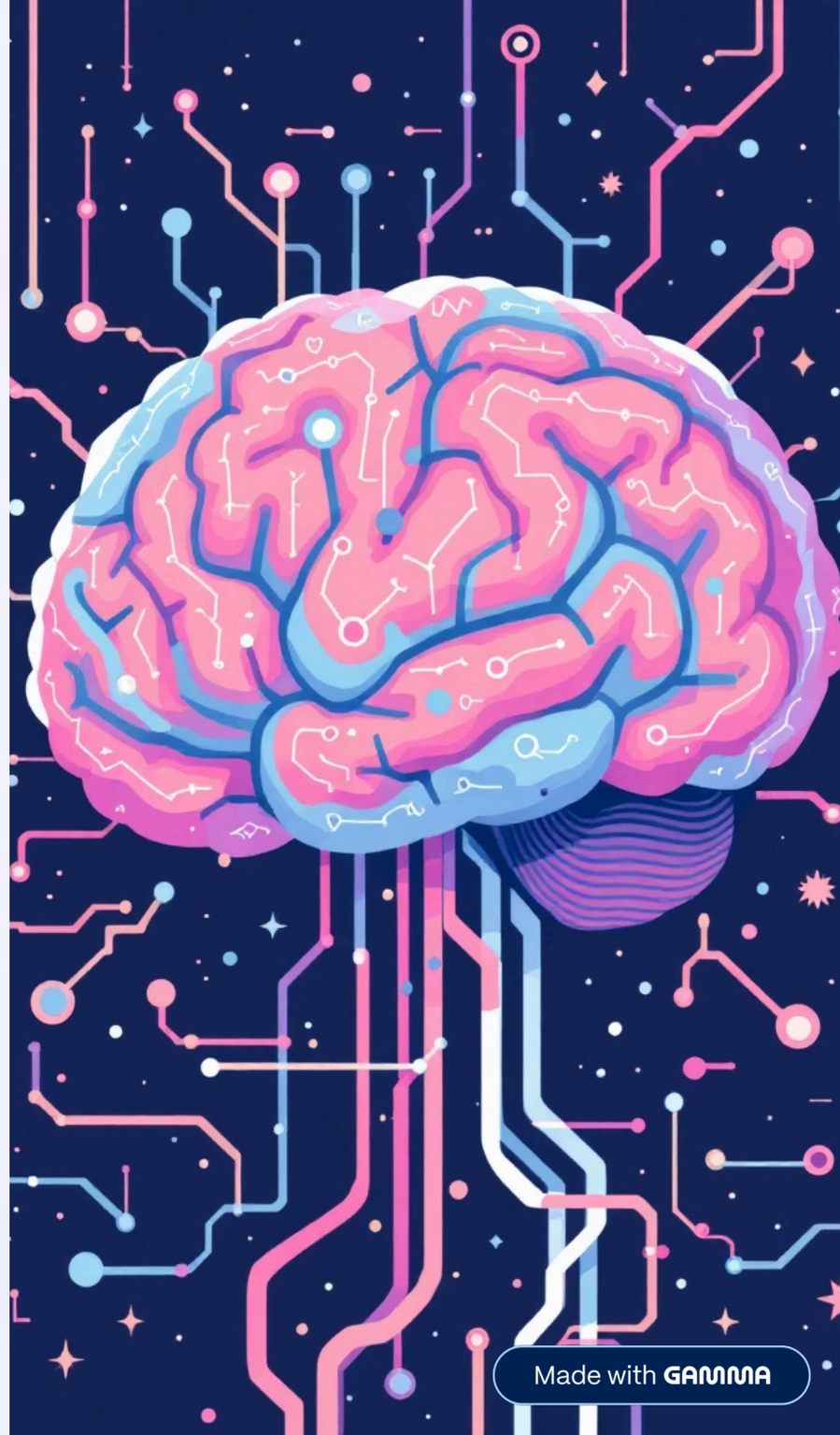
Embeddings Succeed

They preserve semantic relationships in numerical form, enabling the model to understand context and similarity.

Networks Need Numbers

Neural networks process only numerical data—embeddings bridge the gap between human language and machine learning.

Embeddings are the foundation that allows Transformers to work with text effectively, maintaining both structure and meaning.



How Transformers Use Embeddings

Let's walk through a practical example to see how text becomes numerical representation.

01

Tokenisation

The sentence "The cat sat on the mat" is broken into tokens: ["the", "cat", "sat", "on", "the", "mat"]

02

Lookup in Embedding Table

Each token is matched to its corresponding vector in a large embedding matrix

03

Vector Retrieval

From a matrix of shape (30,000 × 512), each word retrieves its unique 512-dimensional vector

Vocabulary Size

30,000 unique tokens (words and subwords)

Embedding Dimension

512 numbers per vector (representation depth)

Example: Input Embedding Matrix

For our sentence *"The cat sat on the mat"* containing 6 tokens, the embedding process creates a matrix:

1

Matrix Shape

(6 × 512)
6 words, each represented by 512 numbers

2

Each Row

One token's vector
[0.12, 0.33, -0.09, ..., 0.48] → "the"

3

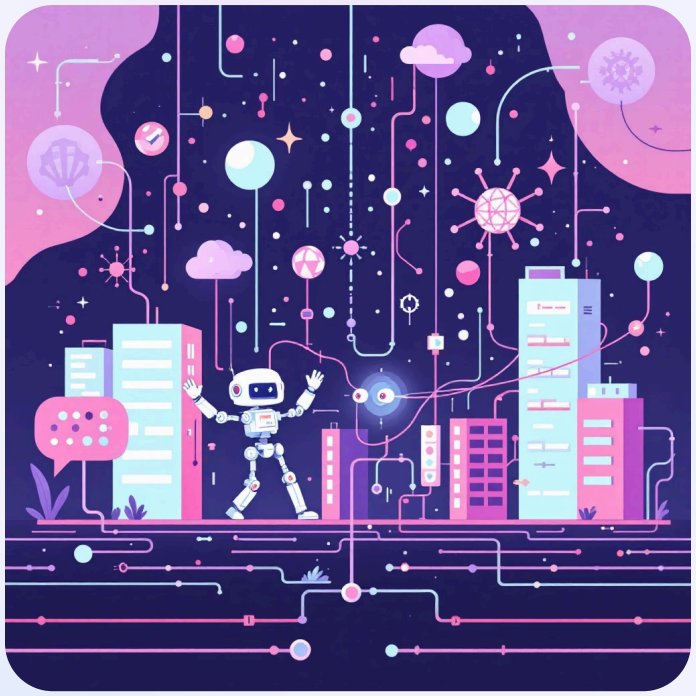
Numerical Form

The entire sentence is now processable by the neural network

```
[
  [0.12, 0.33, -0.09, ..., 0.48],  → "the"
  [0.29, -0.42, 0.18, ..., -0.12], → "cat"
  [0.11, 0.25, -0.07, ..., 0.22],  → "sat"
  ...
]
```

This matrix becomes the foundation for all subsequent Transformer operations.

Are Embeddings Fixed?



No—Embeddings Learn!

Embeddings are **trainable parameters** that improve during model training through gradient descent.

Initial State

Random or pre-trained vectors

During Training

Words in similar contexts move closer together in vector space

Final State

Embeddings capture meaning, grammar, and contextual usage

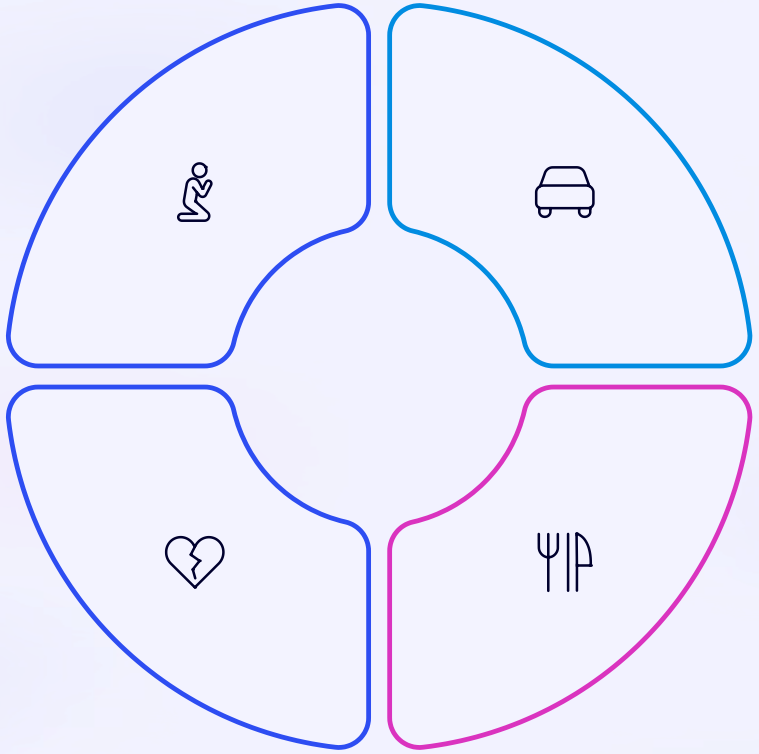
> This learning enables fascinating relationships: king – man + woman ≈ queen

Word Embeddings as a Semantic Map

Imagine all words positioned in a vast multi-dimensional space where distance = semantic difference.

Animals Cluster

"cat", "dog", "pet" group together—related concepts occupy nearby regions



Vehicles Cluster

"car", "truck", "bus" form their own neighbourhood

Emotions Cluster

"happy", "joy", "cheerful" are naturally grouped

Food Cluster

"apple", "banana", "fruit" share semantic proximity

While we visualise this in 2D or 3D, embeddings actually work in **512-dimensional space**—far more expressive and capable of capturing subtle relationships.

Final View: Embedding Layer Output

Input Sentence

> "The cat sat on the mat"

Becomes

6

Sequence Length

Number of tokens

512

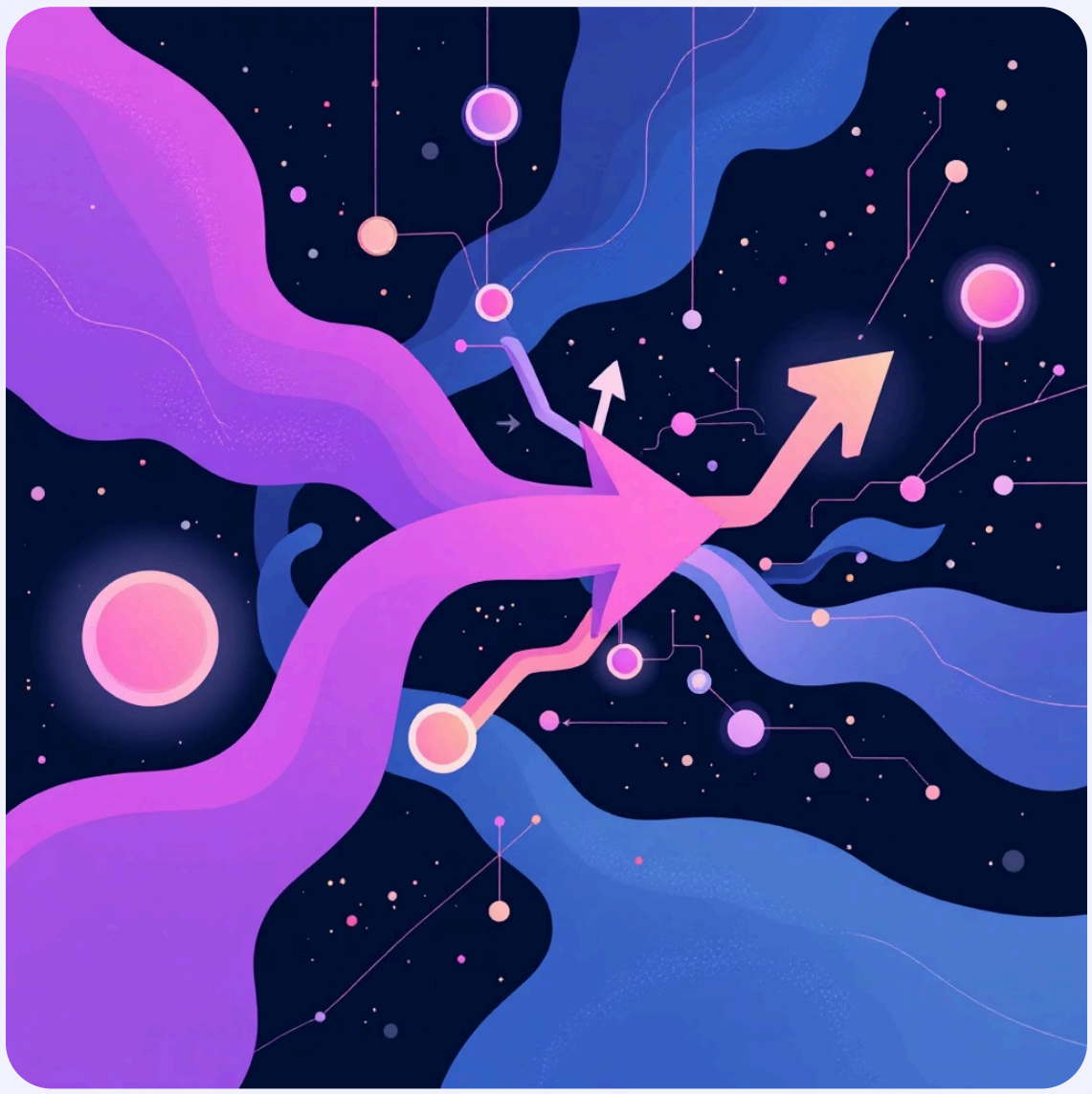
Embedding Dimension

Vector size per token

Output Shape

(6 × 512) matrix

This numerical representation is now ready for the next critical step in the Transformer architecture.



Next Step: Positional Encoding

The embedding layer has successfully converted text into numbers. Now we need to add information about *word order*—because right now, our model doesn't know if "cat sat on mat" or "mat on sat cat"!