Lecture 11: Positional Encoding

In the embedding layer, the same token ID always maps to the same vector, no matter where it appears in the sequence. So:

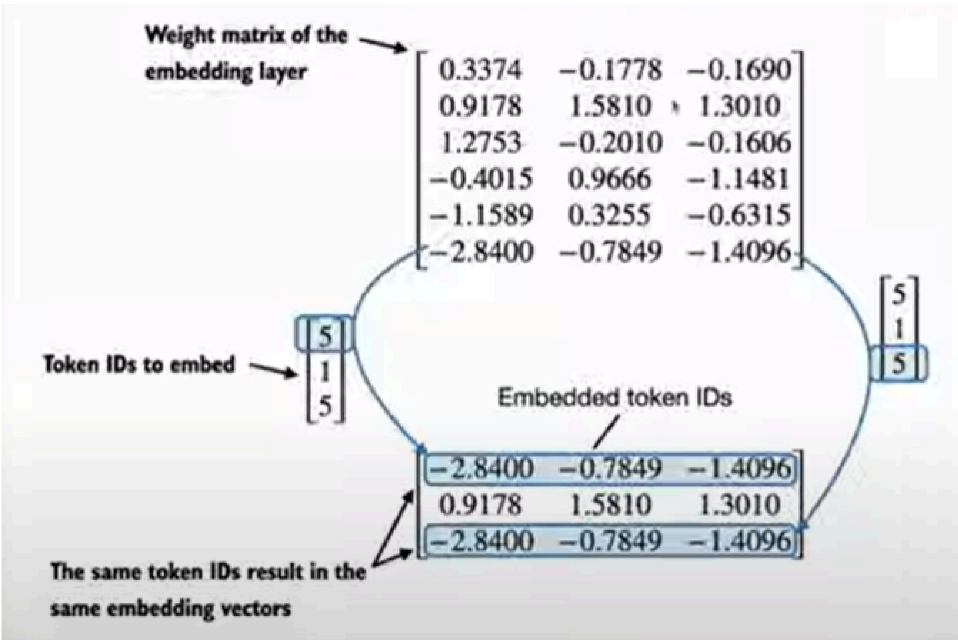
“The cat sat on the mat.”

and

“On the mat, the cat sat.”

Both contain the same words, and *each word gets the exact same embedding every time it appears*.

Nothing in the embedding layer says, “Ah yes, this is the *first* word now,” or “This is near a comma so I must behave differently.” Every instance of **“cat”** becomes the same vector.



That’s why models *need* positional encodings (positional embeddings, rotary embeddings, learned/absolute positions, etc.). Without positions, the model would see these two sentences as just an unordered bag of identical word vectors — and “cat sat mat” would look the same in any permutation.

With positional information added, the model can encode:

- “the cat” vs. “cat the”
- subject vs. object roles
- long-range structure like “On the mat, … sat.”

Two Types of Positional Embeddings

1. Absolute Positional Embeddings

For each position in input sequence, a unique embedding is added to the tokens’ embedding to convey its exact location.

These directly encode the *position number* of each token in the sequence:

token at position 0 gets one vector, position 1 gets another, and so on.

Think of it as telling the model:

“Hey, this word is the **3rd** word in the sentence. Don’t forget.”

Example

Sentence:

“The cat sat on the mat.”

Positions:

```
The  → position 0
cat  → position 1
sat  → position 2
on   → position 3
the  → position 4
mat  → position 5
```

The embedding for **“cat”** becomes:

```
token_embedding("cat") + absolute_pos_embedding(1)
```

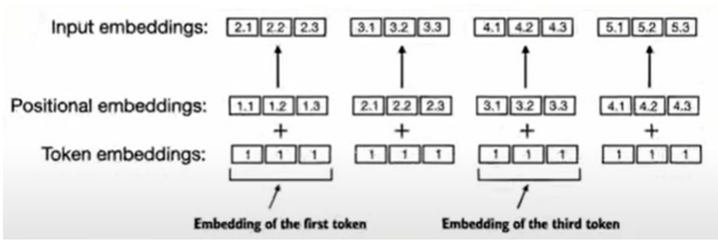
In the shuffled version:

“On the mat, the cat sat.”

Now **“cat”** is at position 4:

```
token_embedding("cat") + absolute_pos_embedding(4)
```

Same word, totally different positional vector → model knows *where* it is.



The positional vectors have the same dimension as the original token embeddings.

2. Relative Positional Embeddings

These encode the *distance* between tokens, not their absolute index.

Instead of “this is the 4th word,” the model gets:

“this word is **two tokens to the right** of that word.”

This is much more natural for language since meaning often depends on relationships, not fixed global positions.

Example

Take the phrase:

“The cat sat.”

If the attention head is looking at **“sat”**, the relative distances are:

```
The → distance -2
cat → distance -1
sat → distance 0
```

Now in the reordered sentence:

“Sat the cat.”

Distances are preserved *locally*:

```
Sat → distance 0
the → distance +1
cat → distance +2
```

The model learns patterns like “subjects are often one or two tokens before the verb”

instead of “subjects happen at index 1.”

This is why relative embeddings generalize better to:

- longer sequences
- reordered phrases
- variable-length inputs

Transformers like **T5**, **DeBERTa**, and **Llama** use variations of this.

Absolute encoding is suitable when fixed order of tokens is crucial, such as sequence generation. GPT-2 was trained using absolute encoding, and the original transformer paper too.

Relative encoding is suitable for tasks like language modeling over long sequences, where the same phrase can appear in different parts of the sequence.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$