Schritt 3: Positional Encoding — Wo stehst du im Satz?

Das Problem: Unsere Embedding-Matrix weiß nicht, in welcher Reihenfolge die Wörter stehen. „Katze sitzt auf Matte" und „Matte sitzt auf Katze" hätten die gleichen Vektoren!

Die Lösung: Jede Position im Satz bekommt ein festes Muster aus Sinus- und Cosinus-Wellen. Dieses Muster wird zum Embedding **addiert**.

Die Formel — Schritt für Schritt erklärt

$$PE(pos, 2i) = \sin\left(pos \,/\, 10000^{(2i \,/\, d)} \right)$$

$$PE(pos, 2i+1) = \cos\left(pos \,/\, 10000^{(2i \,/\, d)} \right)$$

pos = Position im Satz (0, 1, 2, ...) | i = welches Dimensions-Paar (0, 1, ...) | d = Embedding-Größe (bei uns: 4)

Das sieht kompliziert aus! Aber es sind nur **3 Zutaten**:

Zutat	Was es ist	Bei uns
pos	Welches Wort im Satz? (ab 0 gezählt)	0 = Die, 1 = Katze, 2 = sitzt, ...
i	Welches Dimensions-Paar?	i=0 → Dim 0 & 1, i=1 → Dim 2 & 3
$10000^{(2i/d)}$	Der „Teiler" — bestimmt wie schnell die Welle schwingt	i=0 → 10000^0 = **1**, i=1 → $10000^{0.5}$ = **100**

Analogie: Uhrzeiger

Jedes Dimensions-Paar ist wie ein **Uhrzeiger** mit unterschiedlicher Geschwindigkeit:

Dimensionen	Teiler	Geschwindigkeit	Wie eine Uhr
$\mathbf{d_1}$, $\mathbf{d_2}$ (i=0)	÷ 1	Schnell — ändert sich jede Position stark	Sekundenzeiger
$\mathbf{d_3}$, $\mathbf{d_4}$ (i=1)	÷ 100	Langsam — ändert sich kaum von Position zu Position	Stundenzeiger

Zusammen ergeben die Zeiger einen **einzigartigen Code** für jede Position — genau wie 14:30:05 eine einzigartige Uhrzeit ist.

Die Formel nachrechnen — mit Taschenrechner

> **Du brauchst:** Einen Taschenrechner (oder Handy) im **Radiant-Modus** (nicht Grad!)
> Prüfe: sin(1) sollte ≈ 0.841 ergeben, NICHT 0.017.

Unsere Teiler berechnen:

```
Dimension 0 & 1 (i=0):   Teiler = 10000^(2·0 / 4) = 10000^0 = 1
Dimension 2 & 3 (i=1):   Teiler = 10000^(2·1 / 4) = 10000^0.5 = √10000 = 100
```

> Die Teiler sind fest! Bei 4 Dimensionen immer: **1** und **100**.
> Bei 768 Dimensionen (GPT-2) gäbe es 384 verschiedene Teiler von 1 bis 10000.

Position 0 (Wort: „Die"):

```
d₁ = sin(0 / 1)   = sin(0)     = 0.000
d₂ = cos(0 / 1)   = cos(0)     = 1.000
d₃ = sin(0 / 100) = sin(0)     = 0.000
d₄ = cos(0 / 100) = cos(0)     = 1.000
```

Position 0 ist einfach — alles sin(0) oder cos(0)!

Position 1 (Wort: „Katze"):

```
d₁ = sin(1 / 1)   = sin(1)      = 0.841
d₂ = cos(1 / 1)   = cos(1)      = 0.540
d₃ = sin(1 / 100) = sin(0.01)   = 0.010 ← kaum Veränderung!
d₄ = cos(1 / 100) = cos(0.01)   = 1.000 ← fast gleich wie Position 0!
```

> **Siehst du den Uhren-Effekt?**
> d_1 und d_2 (schnelle Dimensionen): springen von 0.000/1.000 auf 0.841/0.540 — großer Unterschied!
> d_3 und d_4 (langsame Dimensionen): von 0.000/1.000 auf 0.010/1.000 — fast identisch!

Position 2 (Wort: „sitzt") — Rechne selbst!

✏️ Hier selbst ausfüllen

d_1 = sin(2 / 1) = sin(2) = ______
d_2 = cos(2 / 1) = cos(2) = ______
d_3 = sin(2 / 100) = sin(0.02) = ______
d_4 = cos(2 / 100) = cos(0.02) = ______

Die komplette Positional-Encoding-Tabelle

> **Drucke diese Seite aus!** Das ist deine zweite Nachschlagetabelle neben der Embedding-Tabelle.
> Für jede Position im Satz schlägst du hier die 4 Positions-Werte nach.

Position	Wort	d_1 sin(pos/1)	d_2 cos(pos/1)	d_3 sin(pos/100)	d_4 cos(pos/100)
0	Die	0.000	1.000	0.000	1.000
1	Katze	0.841	0.540	0.010	1.000
2	sitzt	0.909	-0.416	0.020	1.000
3	auf	0.141	-0.990	0.030	1.000
4	der	-0.757	-0.654	0.040	0.999
5	Matte	-0.959	0.284	0.050	0.999

> **Beachte:** Die Wörter in der Tabelle sind nur für unser Beispiel. Die Positions-Werte gelten für JEDES Wort an dieser Position! Position 1 hat IMMER [0.841, 0.540, 0.010, 1.000] — egal ob dort „Katze", „Hund" oder „Tisch" steht.

Embedding + Position = Finaler Input-Vektor

> Jetzt kommt der entscheidende Moment: Wir **addieren** die beiden Tabellen.
> Für jedes Wort: Nimm den Embedding-Vektor und addiere den Positions-Vektor, Zahl für Zahl.

Beispiel 1: „Die" (Position 0)

Embedding:	0.900	0.100	0.000	0.100
+				
Position 0:	0.000	1.000	0.000	1.000
Ergebnis:	**0.900**	**1.100**	**0.000**	**1.100**

Beispiel 2: „Katze" (Position 1)

Embedding:	0.000	0.900	0.100	0.200
+				
Position 1:	0.841	0.540	0.010	1.000
Ergebnis:	**0.841**	**1.440**	**0.110**	**1.200**

„sitzt", „auf", „der", „Matte" — Rechne selbst!

✏️ Hier selbst ausfüllen

sitzt Embedding:	0.000	0.100	0.900	0.000
+				
Position 2:	0.909	-0.416	0.020	1.000
Ergebnis:	______	______	______	______

auf Embedding: [	0.500	0.000	0.300	0.400]
+				
Position 3: [	0.141	-0.990	0.030	1.000]
Ergebnis: [	?	?	?	?]

der Embedding: [	0.900	0.100	0.000	0.100]
+				
Position 4: [	-0.757	-0.654	0.040	0.999]
Ergebnis: [	?	?	?	?]

Matte Embedding: [	0.000	0.000	0.000	0.900]
+				
Position 5: [	-0.959	0.284	0.050	0.999]
Ergebnis: [	?	?	?	?]

Das gleiche Wort, verschiedene Positionen: „Katze"

> **Warum ist das wichtig?** Hier siehst du, dass das Wort „Katze" je nach Position im Satz einen **anderen** finalen Vektor bekommt — obwohl das Embedding identisch ist!

Stell dir 3 verschiedene Sätze vor, in denen „Katze" an unterschiedlichen Positionen steht:

Satz	Position von „Katze"
Katze sitzt auf der Matte	**0** (erstes Wort)
Die **Katze** sitzt auf der Matte	**1** (zweites Wort)
Auf der Matte sitzt die **Katze**	**5** (letztes Wort)

Der Embedding-Vektor ist immer derselbe:

Katze Embedding: [`0.000` `0.900` `0.100` `0.200`]

Aber die Positions-Vektoren sind verschieden:

„Katze" an Position 0

__Katze__ sitzt auf der Matte

```
Embedding: [0.000, 0.900, 0.100, 0.200]
```

+

```
Position 0: [0.000, 1.000, 0.000, 1.000]
```

```
= [0.000, 1.900, 0.100, 1.200]
```

„Katze" an Position 1

Die __Katze__ sitzt auf der Matte

```
Embedding: [0.000, 0.900, 0.100, 0.200]
```

+

```
Position 1: [0.841, 0.540, 0.010, 1.000]
```

```
= [0.841, 1.440, 0.110, 1.200]
```

„Katze" an Position 5

Auf der Matte sitzt die __Katze__

```
Embedding: [0.000, 0.900, 0.100, 0.200]
```

+

```
Position 5: [-0.959, 0.284, 0.050, 0.999]
```

```
= [-0.959, 1.184, 0.150, 1.199]
```

Drei verschiedene Vektoren für dasselbe Wort!

Vergleiche die Ergebnis-Zeilen:
Position 0: [**0.000**, 1.900, 0.100, 1.200]
Position 1: [**0.841**, 1.440, 0.110, 1.200]
Position 5: [**-0.959**, 1.184, 0.150, 1.199]

Besonders $\mathbf{d_1}$ (schnelle Dimension) unterscheidet die Positionen klar: 0.000 vs. 0.841 vs. -0.959
Während $\mathbf{d_4}$ (langsame Dimension) fast gleich bleibt: 1.200 vs. 1.200 vs. 1.199

Warum geht die Bedeutung nicht verloren?

Der Transformer sieht **nie** den reinen Embedding-Vektor. Er wird von Anfang an mit den kombinierten Vektoren trainiert.

Stell dir vor, du hörst jemanden sprechen:

- Die **Stimme** sagt dir WAS gesagt wird (= Embedding)
- Der **Ort** im Raum sagt dir WOHER es kommt (= Position)

Du hörst beides gleichzeitig — Stimme + Raumklang — und dein Gehirn trennt das automatisch. Genauso lernt der Transformer, aus dem kombinierten Signal beides herauszulesen.

Für den Transformer **IST** [0.841, 1.440, 0.110, 1.200] einfach „Katze an der zweiten Stelle". Er muss nichts abziehen.

Die finale Input-Matrix für den Transformer

> Das ist die Matrix, die in die **Self-Attention**-Schicht geht! Jede Zeile enthält jetzt sowohl die Bedeutung als auch die Position des Wortes.

Pos	Wort	d_1	d_2	d_3	d_4
0	**Die**	**0.900**	**1.100**	**0.000**	**1.100**
1	**Katze**	**0.841**	**1.440**	**0.110**	**1.200**
2	**sitzt**				
3	**auf**				
4	**der**				
5	**Matte**				

↑ Fülle die leeren Felder mit deinen Ergebnissen von Seite 4 aus!

Übung: Sind „Die" und „der" jetzt noch ähnlich?

> Im Embedding-Schritt waren „Die" und „der" fast identisch (Skalarprodukt: 0.83). Jetzt haben sie **verschiedene Positions-Vektoren** bekommen. Sind sie immer noch ähnlich?

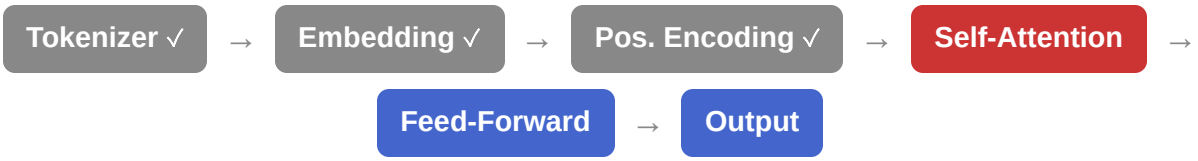
✏️ Hier selbst ausfüllen

	d_1	d_2	d_3	d_4	
Die (Pos 0)	0.900	1.100	0.000	1.100	
der (Pos 4)	_____	_____	_____	_____	
Multiplizieren					
Ergebnis					**Summe = ___**

> **Was erwartest du?**
> „Die" (Position 0) und „der" (Position 4) haben ähnliche Embeddings aber **verschiedene Positions-Vektoren**. Die Ähnlichkeit sollte sich verändert haben! Das ist gewollt — der Transformer soll wissen, dass sie an verschiedenen Stellen stehen.

Zusammenfassung: Was wir bisher gebaut haben

Schicht	Input	Output	Papier-Modell
Tokenizer	Text	Token-IDs	Kärtchen ausschneiden
Embedding	Token-IDs	Vektoren (nur Bedeutung)	In Tabelle nachschlagen
Pos. Encoding	Vektoren	Vektoren (Bedeutung + Position)	Positions-Werte addieren



> **Nächster Schritt: Self-Attention** — Das Herzstück des Transformers!
>
> Hier fragt jedes Wort: *„Welche anderen Wörter im Satz sind für mich relevant?"*
> „sitzt" wird lernen, auf „Katze" zu achten (wer sitzt?) und auf „Matte" (worauf?).
> Dafür berechnen wir **Query, Key und Value** — drei neue Vektoren pro Wort.