

# NUMERICAL ATTENTION EXAMPLE

Great — let's build a **simple, clear, fully-numerical self-attention example** with exactly the tokens you requested:

- Token 1: "I"
- Token 2: "learned"
- Token 3: "cooking"
- Embedding dimension: 3

We will compute **one self-attention head** from scratch.

Everything below uses tiny numbers so you can compute it by hand.

---

## Step 1 — Define the Embeddings (3-dimensional)

Let's pick small vectors:

$$x_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, x_{learned} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, x_{cooking} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Stack them into a matrix:

$$X = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Rows = tokens, columns = embedding features.

---

## Step 2 — Pick small projection matrices

To keep calculations simple, choose:

$$W^Q = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, W^K = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, W^V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note:  $W^V$  chosen simple so the values stay close to the embeddings.

---

## Step 3 — Compute Q, K, V

Compute:

$$Q = XW^Q, K = XW^K, V = XW^V$$

### Token 1: "I"

$$\begin{aligned} q_I = W^Q x_I &= \begin{bmatrix} 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 \\ 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} \\ k_I = W^K x_I &= \begin{bmatrix} 1 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 \\ 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \\ v_I = x_I &= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

---

### Token 2: "learned"

$$\begin{aligned} q_L = W^Q x_L &= \begin{bmatrix} 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 \\ 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 \\ 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ k_L = W^K x_L &= \begin{bmatrix} 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 \\ 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 \\ 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\ v_L &= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \end{aligned}$$

---

### Token 3: "cooking"

$$\begin{aligned} q_C = W^Q x_C &= \begin{bmatrix} 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ k_C = W^K x_C &= \begin{bmatrix} 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 \\ 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \\ v_C &= \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{aligned}$$

---

## Step 4 — Compute Attention Scores

### $q \cdot k$

We compute a  $3 \times 3$  score matrix.

**For token "I" (query  $q_I = [2, 0, 2]$ )**

$$q_I k_I = 2 \cdot 1 + 0 \cdot 1 + 2 \cdot 2 = 6$$

$$q_I k_L = 2 \cdot 1 + 0 \cdot 2 + 2 \cdot 1 = 4$$

$$q_I k_C = 2 \cdot 2 + 0 \cdot 1 + 2 \cdot 1 = 6$$

Scores for "I":

$$[6, 4, 6]$$

---

**For "learned" ( $q_L = [1, 1, 1]$ )**

$$q_L k_I = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 2 = 4$$

$$q_L k_L = 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 1 = 4$$

$$q_L k_C = 1 \cdot 2 + 1 \cdot 1 + 1 \cdot 1 = 4$$

Scores for "learned":

$$[4, 4, 4]$$

---

**For "cooking" ( $q_C = [1, 1, 1]$ )**

(same as learned)

$$[4, 4, 4]$$

---

## Step 5 — Scale

$$d_k = 3$$

$$\sqrt{3} \approx 1.732$$

"I"

$$[6, 4, 6] / 1.732 = [3.46, 2.31, 3.46]$$

**“learned”**

$$[4,4,4]/1.732 = [2.31, 2.31, 2.31]$$

**“cooking”**

(same)

$$[2.31, 2.31, 2.31]$$

---

## Step 6 — Softmax rows

**Token “I”**

$$\text{softmax}(3.46, 2.31, 3.46)$$

Exponentials:

- $e^{3.46}=31.8$
- $e^{2.31}=10.1$

$$\text{Sum} = 31.8 + 10.1 + 31.8 = 73.7$$

Weights:

$$[0.431, 0.137, 0.431]$$

---

**Token “learned”**

Row is uniform → softmax is uniform:

$$[0.333, 0.333, 0.333]$$

---

**Token “cooking”**

Same uniform:

$$[0.333, 0.333, 0.333]$$

---

## Step 7 — Weighted sum of values

Recall:

$$v_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, v_L = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, v_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

---

## Output for "I"

$$z_I = 0.431v_I + 0.137v_L + 0.431v_C$$

Compute:

$$\begin{aligned} 0.431v_I &= \begin{bmatrix} 0.431 \\ 0 \\ 0.431 \end{bmatrix} \\ 0.137v_L &= \begin{bmatrix} 0 \\ 0.137 \\ 0.137 \end{bmatrix} \\ 0.431v_C &= \begin{bmatrix} 0.431 \\ 0.431 \\ 0 \end{bmatrix} \end{aligned}$$

Add:

$$z_I = \begin{bmatrix} 0.431 + 0 + 0.431 \\ 0 + 0.137 + 0.431 \\ 0.431 + 0.137 + 0 \end{bmatrix} = \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \end{bmatrix}$$

---

## Output for "learned"

$$z_L = \frac{1}{3}(v_I + v_L + v_C)$$

Add values:

$$v_I + v_L + v_C = \begin{bmatrix} 1 + 0 + 1 \\ 0 + 1 + 1 \\ 1 + 1 + 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

Divide:

$$z_L = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}$$

---

## Output for "cooking"

Same as “learned” because the weights were uniform:

$$z_C = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}$$

---

## Final Self-Attention Outputs for 3 Tokens

$$z_I = \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \end{bmatrix}, z_{learned} = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}, z_{cooking} = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}$$

These are the new contextual embeddings after one self-attention head.

---



### Show Full transformer block (add residuals + layer norm + FFN) for the same example

Absolutely — let's build **the full Transformer block** using **your exact 3-token self-attention example**:

- Tokens: **I, learned, cooking**
- Embedding dimension: **3**
- We will reuse the **same attention outputs** we computed:

$$z_I = \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \end{bmatrix}, z_{learned} = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}, z_{cooking} = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}$$

Now we will complete **the full Transformer block**:

1. Add residual connection
2. Apply layer normalization
3. Apply the feed-forward network (FFN)
4. Add second residual
5. Apply final layer norm

Everything with small numbers so you can compute by hand.

---

# ◆ 1. Residual Connection (Add the original embedding X)

Original embeddings (from your example):

$$x_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, x_L = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, x_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Residual output =  $\mathbf{X} + \mathbf{Attention}(\mathbf{X})$

## Token I

$$r_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \end{bmatrix} = \begin{bmatrix} 1.862 \\ 0.568 \\ 1.568 \end{bmatrix}$$

## Token learned

$$r_L = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix} = \begin{bmatrix} 0.667 \\ 1.667 \\ 1.667 \end{bmatrix}$$

## Token cooking

$$r_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix} = \begin{bmatrix} 1.667 \\ 1.667 \\ 0.667 \end{bmatrix}$$

---

# ◆ 2. Layer Normalization (LN)

LayerNorm normalizes **each token vector** to mean 0, variance 1.

Instead of full decimals, we compute approximate normalized values for clarity.

---

## Token I — Normalize $r_I$

$$r_I = [1.862, 0.568, 1.568]$$

Mean:

$$\mu_I = (1.862 + 0.568 + 1.568)/3 = 1.333$$

Subtract mean:

$$[0.529, -0.765, 0.235]$$

Variance:

$$\sigma^2 \approx (0.529^2 + (-0.765)^2 + 0.235^2)/3 = 0.273$$

Std:

$$\sigma \approx 0.523$$

Normalized:

$$\text{LN}(r_I) \approx \begin{bmatrix} 1.01 \\ -1.46 \\ 0.45 \end{bmatrix}$$

---

## Token learned — Normalize $r_L$

$$r_L = [0.667, 1.667, 1.667]$$

Mean:

$$\mu_L = 1.333$$

Subtract mean:

$$[-0.666, 0.334, 0.334]$$

Std  $\approx 0.544$

Normalized:

$$\text{LN}(r_L) \approx \begin{bmatrix} -1.22 \\ 0.61 \\ 0.61 \end{bmatrix}$$

---

## Token cooking — Normalize $r_C$

$$r_C = [1.667, 1.667, 0.667]$$

Mean:

$$\mu_c = 1.333$$

Subtract:

$$[0.334, 0.334, -0.666]$$

Std  $\approx 0.544$

Normalized:

$$\text{LN}(r_c) \approx \begin{bmatrix} 0.61 \\ 0.61 \\ -1.22 \end{bmatrix}$$

---

## ◆ 3. Feed-Forward Network (FFN)

FFN has two layers:

$$\text{FFN}(x) = \max(0, xW_1 + b_1) W_2 + b_2$$

We choose **very simple** matrices so the math is easy:

$$W_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}, b_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$
$$W_2 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

💡 We keep values approximate so you can follow clearly.

---

## FFN for each token

---

**Token I** — input  $h_I = \text{LN}(r_I)$

$$h_I = \begin{bmatrix} 1.01 \\ -1.46 \\ 0.45 \end{bmatrix}$$

**1st layer:**

$$u_I = h_I W_1 = \begin{bmatrix} 1.01 + 0.45 \\ -1.46 + 0.45 \\ 1.01 \end{bmatrix} = \begin{bmatrix} 1.46 \\ -1.01 \\ 1.01 \end{bmatrix}$$

ReLU:

$$\max(0, u_I) = \begin{bmatrix} 1.46 \\ 0 \\ 1.01 \end{bmatrix}$$

**2nd layer:**

$$z_I^{FFN} = \begin{bmatrix} 1.46 - 1.01 \\ 0 \\ 1.46 + 0 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0 \\ 1.46 \end{bmatrix}$$

---

## Token learned

$$h_L = \begin{bmatrix} -1.22 \\ 0.61 \\ 0.61 \end{bmatrix}$$

1st layer:

$$u_L = \begin{bmatrix} -1.22 + 0.61 \\ 0.61 + 0.61 \\ -1.22 \end{bmatrix} = \begin{bmatrix} -0.61 \\ 1.22 \\ -1.22 \end{bmatrix}$$

ReLU:

$$[0, 1.22, 0]$$

2nd layer:

$$z_L^{FFN} = \begin{bmatrix} 0 - 0 \\ 1.22 \\ 0 + 1.22 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.22 \\ 1.22 \end{bmatrix}$$

---

## Token cooking

$$h_C = \begin{bmatrix} 0.61 \\ 0.61 \\ -1.22 \end{bmatrix}$$

1st layer:

$$u_c = \begin{bmatrix} 0.61 - 1.22 \\ 0.61 - 1.22 \\ 0.61 \end{bmatrix} = \begin{bmatrix} -0.61 \\ -0.61 \\ 0.61 \end{bmatrix}$$

ReLU:

$$[0, 0, 0.61]$$

2nd layer:

$$z_c^{FFN} = \begin{bmatrix} 0 - 0.61 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.61 \\ 0 \\ 0 \end{bmatrix}$$

---

## ◆ 4. Add 2nd Residual (LN output + FFN output)

$$\begin{aligned} o_I &= h_I + z_I^{FFN} \\ o_L &= h_L + z_L^{FFN} \\ o_C &= h_C + z_C^{FFN} \end{aligned}$$

Compute:

**Token I**

$$o_I = \begin{bmatrix} 1.01 + 0.45 \\ -1.46 + 0 \\ 0.45 + 1.46 \end{bmatrix} = \begin{bmatrix} 1.46 \\ -1.46 \\ 1.91 \end{bmatrix}$$

**Token learned**

$$o_L = \begin{bmatrix} -1.22 + 0 \\ 0.61 + 1.22 \\ 0.61 + 1.22 \end{bmatrix} = \begin{bmatrix} -1.22 \\ 1.83 \\ 1.83 \end{bmatrix}$$

**Token cooking**

$$o_C = \begin{bmatrix} 0.61 - 0.61 \\ 0.61 + 0 \\ -1.22 + 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.61 \\ -1.22 \end{bmatrix}$$

---

## ◆ 5. Final LayerNorm

We normalize each again (approx).

### Token I

$$\mu \approx 0.637, \sigma \approx 1.404$$
$$\text{LN}(o_I) \approx \begin{bmatrix} 0.59 \\ -1.49 \\ 1.00 \end{bmatrix}$$

### Token learned

$$\text{LN}(o_L) \approx \begin{bmatrix} -1.52 \\ 0.76 \\ 0.76 \end{bmatrix}$$

### Token cooking

$$\text{LN}(o_C) \approx \begin{bmatrix} -0.71 \\ 0.71 \\ -0.71 \end{bmatrix}$$



## Final Output of the Full Transformer Block

Token "I"	=	$\begin{bmatrix} 0.59 \\ -1.49 \\ 1.00 \end{bmatrix}$
Token "learned"	=	$\begin{bmatrix} -1.52 \\ 0.76 \\ 0.76 \end{bmatrix}$
Token "cooking"	=	$\begin{bmatrix} -0.71 \\ 0.71 \\ -0.71 \end{bmatrix}$

These vectors are the **final contextual embeddings** after:

- self-attention
- residual
- layer norm
- FFN
- residual
- layer norm

This is **exactly what a Transformer encoder block does** — just with much larger dimensions and many stacked layers.

---



## Continue the example to show last layers that include a linear layer and softmax after transformer block

Great — let's **complete the entire pipeline** by adding the **final linear layer + softmax** on top of your **full Transformer block outputs**.

We will continue **directly from the outputs you already computed** for the block:

$$\text{Token "I"} = \begin{bmatrix} 0.59 \\ -1.49 \\ 1.00 \end{bmatrix}, \text{Token "learned"} = \begin{bmatrix} -1.52 \\ 0.76 \\ 0.76 \end{bmatrix}, \text{Token "cooking"} = \begin{bmatrix} -0.71 \\ 0.71 \\ -0.71 \end{bmatrix}$$

These are the **final contextual embeddings** after:

- Self-attention
- Residual
- LayerNorm
- FFN
- Residual
- LayerNorm

Now we'll perform the **final prediction stage** (just like in GPT/BERT/etc.):

---

## ★ 1. Linear Layer (the "output projection")

For language modeling, we map each 3-dimensional vector to **vocab size** logits.

Let's assume a **tiny artificial vocabulary** of 4 tokens:

1. "I"
2. "learned"
3. "cooking"
4. "." (period)

So vocab size = 4.

Define a small weight matrix:

$$W^{out} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & -1 & 0 \\ 2 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{4 \times 3}$$

and zero bias for simplicity:

$$b^{out} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The output logits for each token are:

$$\ell = W^{out}h + b^{out}$$

---

## ★ 2. Compute Logits for Each Token

### Token "I"

$$h_I = \begin{bmatrix} 0.59 \\ -1.49 \\ 1.00 \end{bmatrix}$$

Multiply:

$$\ell_I = \begin{bmatrix} 1 \cdot 0.59 + 0 \cdot (-1.49) + 1 \cdot 1.00 \\ 0 \cdot 0.59 + 1 \cdot (-1.49) + 1 \cdot 1.00 \\ 1 \cdot 0.59 + (-1) \cdot (-1.49) + 0 \cdot 1.00 \\ 2 \cdot 0.59 + 0 \cdot (-1.49) + (-1) \cdot 1.00 \end{bmatrix} = \begin{bmatrix} 1.59 \\ -0.49 \\ 2.08 \\ 0.18 \end{bmatrix}$$

---

### Token "learned"

$$h_L = \begin{bmatrix} -1.52 \\ 0.76 \\ 0.76 \end{bmatrix}$$
$$\ell_L = \begin{bmatrix} -1.52 + 0.76 \\ 0.76 + 0.76 \\ -1.52 - 0.76 \\ 2(-1.52) - 0.76 \end{bmatrix} = \begin{bmatrix} -0.76 \\ 1.52 \\ -2.28 \\ -3.80 \end{bmatrix}$$

---

### Token "cooking"

$$h_c = \begin{bmatrix} -0.71 \\ 0.71 \\ -0.71 \end{bmatrix}$$
$$\ell_c = \begin{bmatrix} -0.71 + (-0.71) \\ 0.71 + (-0.71) \\ -0.71 - 0.71 \\ 2(-0.71) - (-0.71) \end{bmatrix} = \begin{bmatrix} -1.42 \\ 0 \\ -1.42 \\ -0.71 \end{bmatrix}$$

---

## ★ 3. Apply Softmax to Logits

Softmax produces a probability distribution over the 4 vocabulary tokens.

---

### Token "l"

Logits:

$$[1.59, -0.49, 2.08, 0.18]$$

Exponentials:

- $e^{2.08} = 8.01$
- $e^{1.59} = 4.90$
- $e^{-0.49} = 0.61$
- $e^{0.18} = 1.20$

Sum = 14.72

Softmax:

$$p_l \approx \begin{bmatrix} 0.333 \\ 0.041 \\ 0.544 \\ 0.082 \end{bmatrix}$$

Interpretation:

Token "l" is most likely predicting "**cooking**" next (in this toy vocabulary).

---

### Token "learned"

Logits:

$$[-0.76, 1.52, -2.28, -3.80]$$

Exponentials:

$$e^{1.52} = 4.57, e^{-0.76} = 0.47, e^{-2.28} = 0.10, e^{-3.80} = 0.02$$

Sum = 5.16

Softmax:

$$p_L \approx \begin{bmatrix} 0.091 \\ 0.885 \\ 0.019 \\ 0.004 \end{bmatrix}$$

Interpretation:

"learned" strongly predicts "**learned**" itself in this tiny demo (due to our simple matrices).

---

## Token "cooking"

Logits:

$$[-1.42, 0, -1.42, -0.71]$$

Exponentials:

$$[0.24, 1.00, 0.24, 0.49]$$

Sum = 1.97

Softmax:

$$p_C \approx \begin{bmatrix} 0.12 \\ 0.51 \\ 0.12 \\ 0.25 \end{bmatrix}$$

Interpretation:

"cooking" most strongly predicts "**learned**" under these toy weights.

---

## Final Output Probabilities After the Transformer Block

$p_I$	=	[0.333, 0.041, 0.544, 0.082]
$p_{learned}$	=	[0.091, 0.885, 0.019, 0.004]
$p_{cooking}$	=	[0.12, 0.51, 0.12, 0.25]

These are the **final predictions** after:

1. Self-Attention
2. Add & Norm
3. Feed-Forward
4. Add & Norm
5. Linear projection
6. Softmax

This completes **the full forward pass of a Transformer encoder layer with output head**, using your **custom 3-token numerical example**.



## Visualize the attention weights for each token.

Below is a **simple and clear visualization** of the **self-attention weights** for the **3-token example**:

- Token 1: **I**
- Token 2: **learned**
- Token 3: **cooking**

These weights come *directly* from the attention calculation we already performed.

## Final Attention Weights (Softmax Results)

From the earlier math, the attention-weight matrix was:

Query ↓ attends to →	I	learned	cooking
I	0.431	0.137	0.431
learned	0.333	0.333	0.333
cooking	0.333	0.333	0.333

This means:

- **"I"** strongly attends to **I** and **cooking**
- **"learned"** attends uniformly to all tokens
- **"cooking"** also attends uniformly



## Heatmap-Style Visualization

Think of darker color = higher weight.

	I	learned	cooking
I	[ 0.431	0.137	0.431 ]
learned	[ 0.333	0.333	0.333 ]
cooking	[ 0.333	0.333	0.333 ]

You can visualize intensity like this:

	I	learned	cooking
I			
learned			
cooking			

Legend:

- = strong attention ( $\approx 0.43$ )
- = medium ( $\approx 0.33$ )
- = weak ( $\approx 0.14$ )



## Diagram Visualization

	I	learned	cooking
Query: I	0.431	0.137	0.431

Query:			
learned	0.333	0.333	0.333
Query:			
cooking	0.333	0.333	0.333

---

## Intuition Behind the Weights

### ◆ Token: "I"

- It strongly attends to **itself** (0.431)
- It also strongly attends to **"cooking"** (0.431)
- It mostly ignores **"learned"** (0.137)

### ◆ Token: "learned"

- All weights equal (0.333)
- Meaning: "learned" doesn't show strong preference — it takes an average of the context.

### ◆ Token: "cooking"

- Also evenly distributed
- Learns an "average" context representation.



In this example, Which one is selected as next-token for language modelling?

Great question — let's determine **which next-token would be selected** by the model in *our exact numerical example*.

We already computed the **final softmax probabilities** for each token after the Transformer block + linear layer.

But in **language modelling**, the model predicts the **next token based on the LAST token's output distribution**.

Since our tiny example sentence was:

**I learned cooking**

...the next token prediction comes from the **final token's representation**, i.e., **the softmax for "cooking."**

---

## ★ Final Softmax for "cooking" (from earlier)

$$p_{\text{cooking}} = [0.12, 0.51, 0.12, 0.25]$$

And recall our tiny vocab was:

Index	Token
1	"I"
2	"learned"
3	"cooking"
4	":"

Mapping the probabilities:

Token	Probability
"I"	0.12
"learned"	<b>0.51</b>
"cooking"	0.12
":"	0.25

---

## 🎯 Next-token selected (argmax softmax)

The highest probability is **0.51**, corresponding to:

✅ **"learned"**

So in this toy example, the model predicts:

**Next token = "learned"**

---



# Why did this happen?

Because:

- The final token “cooking” had a **uniform attention pattern**, meaning its contextual representation was quite neutral.
- The output projection matrix we defined happens to map that embedding strongly toward the “learned” direction.
- In a real model with millions of learned parameters, patterns would be meaningful — our numbers are hand-chosen just for demonstration.



## Show a numeric multi-head example

Great — here is a **fully numeric, step-by-step multi-head attention example** using **two heads**, built on the same style as your earlier single-head examples.

No web search is required because this is purely mathematical.

This example is intentionally small so you can compute it by hand.



### Setup

We use the same 3 tokens:

1. **I**
2. **learned**
3. **cooking**

We assign **3-dimensional embeddings**:

$$x_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, x_L = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, x_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Stack into matrix:

$$X = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

We create **2 heads**, each with its own  $W^Q, W^K, W^V$ .

---

## HEAD 1 — Matrices

Define:

$$W_1^Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, W_1^K = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, W_1^V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So head1 is basically your earlier example.

---

## HEAD 1 — Compute Q,K,V

**Queries** ( $Q = X W_1^Q$ ) are identical to X:

$$Q_1 = X = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

**Keys:**

- For “I”:

$$k_I = W_1^K x_I = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

- “learned”:

$$k_L = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

- “cooking”:

$$k_C = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

**Values** are the embeddings (because  $W_1^V$  is identity):

$$v_I = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, v_L = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, v_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

---

## HEAD 1 — Attention Scores

(We already computed this earlier.)

Attention weights matrix:

Query ↓ → Key →	I	learned	cooking
I	0.431	0.137	0.431
learned	0.333	0.333	0.333
cooking	0.333	0.333	0.333

---

## ■ HEAD 1 — Output vectors

(Also computed earlier)

$$\text{head1}(I) = \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \end{bmatrix}, \text{head1}(L) = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}, \text{head1}(C) = \begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \end{bmatrix}$$


---

## 🎯 HEAD 2 — Different projection matrices

To make multi-head meaningful, head 2 must look at a different sub-space.

Define:

$$W_2^Q = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \end{bmatrix}, W_2^K = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, W_2^V = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Now head2 will behave very differently.

---

## 🔍 HEAD 2 — Compute Q,K,V

I'll compute them concisely:

**Queries**

$$q_I = W_2^Q x_I = \begin{bmatrix} 1 \cdot 1 + (-1) \cdot 0 + 0 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 0 + (-1) \cdot 1 \\ 1 \cdot 1 + 0 \cdot 0 + (-1) \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

$$q_L = \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}, q_C = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$


---

## Keys

$$k_I = W_2^K x_I = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

$$k_L = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

$$k_C = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

---

## Values

Compute quickly:

$$v_I = W_2^V x_I = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$v_L = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

$$v_C = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

---

## HEAD 2 — Compute Attention Scores

Query: I

$$q_I = [1, -1, 0]$$

$$q_I k_I = 1 \cdot 1 + (-1) \cdot 1 + 0 \cdot 2 = 0$$

$$q_I k_L = 1 \cdot 2 + (-1) \cdot 1 + 0 \cdot 1 = 1$$

$$q_I k_C = 1 \cdot 2 + (-1) \cdot 2 + 0 \cdot 1 = 0$$

Scores:

$$[0, 1, 0]$$

Softmax:

$$[0.212, 0.576, 0.212]$$

---

### Query: learned

$$q_L = [-1, 0, -1]$$

$$q_L k_I = -1 \cdot 1 + 0 \cdot 1 + -1 \cdot 2 = -3$$

$$q_L k_L = -1 \cdot 2 + 0 \cdot 1 + -1 \cdot 1 = -3$$

$$q_L k_C = -1 \cdot 2 + 0 \cdot 2 + -1 \cdot 1 = -3$$

Uniform  $\rightarrow$  softmax:

$$[0.333, 0.333, 0.333]$$

---

### Query: cooking

$$q_C = [0, 1, 1]$$

$$q_C k_I = 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 2 = 3$$

$$q_C k_L = 0 \cdot 2 + 1 \cdot 1 + 1 \cdot 1 = 2$$

$$q_C k_C = 0 \cdot 2 + 1 \cdot 2 + 1 \cdot 1 = 3$$

Softmax of [3,2,3] =

$$[0.422, 0.155, 0.422]$$

---

### ■ HEAD 2 — Output values

Compute weighted sums:

I

$$z_I^{(2)} = 0.212v_I + 0.576v_L + 0.212v_C$$

$$= 0.212 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + 0.576 \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} + 0.212 \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Compute:

$$z_I^{(2)} = \begin{bmatrix} 1.00 \\ 1.57 \\ 1.00 \end{bmatrix}$$

---

## learned

Uniform → mean of values:

$$z_L^{(2)} = \frac{v_I + v_L + v_C}{3} = \begin{bmatrix} 1.33 \\ 1.33 \\ 1.00 \end{bmatrix}$$

---

## cooking

$$z_C^{(2)} = 0.422v_I + 0.155v_L + 0.422v_C = \begin{bmatrix} 1.58 \\ 1.00 \\ 1.00 \end{bmatrix}$$

---

## ✿ Concat the Heads

Each head output is 3-dimensional.

Concatenation gives a **6-dimensional vector per token**:

### Token “I”

$$\text{Concat} = \begin{bmatrix} 0.862 \\ 0.568 \\ 0.568 \\ 1.00 \\ 1.57 \\ 1.00 \end{bmatrix}$$

### Token “learned”

$$\begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \\ 1.33 \\ 1.33 \\ 1.00 \end{bmatrix}$$

### Token “cooking”

$$\begin{bmatrix} 0.667 \\ 0.667 \\ 0.667 \\ 1.58 \\ 1.00 \\ 1.00 \end{bmatrix}$$

---

## Final Output Projection $W^O$

Pick a simple matrix to map 6 → 3:

$$W^O = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

This just adds the two heads component-wise.

### Token “I”

$$\begin{bmatrix} 0.862 + 1.00 \\ 0.568 + 1.57 \\ 0.568 + 1.00 \end{bmatrix} = \begin{bmatrix} 1.862 \\ 2.138 \\ 1.568 \end{bmatrix}$$

### Token “learned”

$$\begin{bmatrix} 0.667 + 1.33 \\ 0.667 + 1.33 \\ 0.667 + 1.00 \end{bmatrix} = \begin{bmatrix} 2.00 \\ 2.00 \\ 1.667 \end{bmatrix}$$

### Token “cooking”

$$\begin{bmatrix} 0.667 + 1.58 \\ 0.667 + 1.00 \\ 0.667 + 1.00 \end{bmatrix} = \begin{bmatrix} 2.247 \\ 1.667 \\ 1.667 \end{bmatrix}$$

---

## Final Multi-Head Attention Outputs

$$\text{Token “I”} = \begin{bmatrix} 1.862 \\ 2.138 \\ 1.568 \end{bmatrix}, \text{Token “learned”} = \begin{bmatrix} 2.00 \\ 2.00 \\ 1.667 \end{bmatrix}, \text{Token “cooking”} = \begin{bmatrix} 2.247 \\ 1.667 \\ 1.667 \end{bmatrix}$$

These vectors would then be passed into:

- Add & Norm
- Feed-Forward
- Add & Norm
- Output head (linear + softmax)

...completing the full Transformer block.