Multi-Head Self-Attention

⚡ PART 1 — Attention (Short Recap)

> Attention lets a model focus on the most relevant parts of the input when producing each part of the output.

Instead of using one fixed context vector (the bottleneck), the model:

1. Looks at **all encoder hidden states**.
2. Assigns **weights** to each based on relevance.
3. Creates a **weighted average (context vector)** to generate the next output.

Mini Formula:

$$e_{t,i} = \text{score}(s_t, h_i) \tag{1}$$
$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{2}$$
$$c_t = \sum_i \alpha_{t,i} h_i \tag{3}$$

→ The model "attends" to specific words at each output step.

Example: When translating "I love deep learning",

to generate "J'aime", the model focuses more on "love".

🧠 PART 2 — Self-Attention (Detailed)

Now let's extend that same idea — but **within a single sequence**.

1️⃣ What It Is

> Self-Attention (or Intra-Attention) is a mechanism where each token in a sequence attends to every other token in the same sequence — including itself — to compute richer contextual representations.

💬 In plain English:

Every word *looks at* every other word to understand **its meaning in context**.

2️⃣ Why We Need It

In text, the meaning of a word depends on others around it:

- “bank” in “river bank” vs. “credit bank”.

A simple word embedding can’t capture that dynamic context.

Self-Attention fixes this by updating each word’s representation based on *all the other words* in the sentence.

3️⃣ The Core Idea

For every input token, the model asks:

> “Which other words in this sentence should I pay attention to when representing this word?”

It then creates a **weighted combination** of all token representations.

4️⃣ How It Works — Step-by-Step

Let’s say our input sequence has n words, each represented by an embedding vector:

$$X = [x_1, x_2, ..., x_n]$$

We’ll compute **Self-Attention** as follows 👇

(a) Create Queries, Keys, and Values

Each input vector x_i is projected into three different spaces using learned weight matrices:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

- **Query (Q):** what the word is *looking for*
- **Key (K):** what information each word *has*

- **Value (V):** the actual word representation to pass on

Each word gets its own q_i, k_i, v_i.

(b) Compute Similarity (Attention Scores)

For every word i, compare its query q_i with *all* keys k_j (including itself) using dot product:

$\text{score}(i, j) = q_i \cdot k_j^T$

→ This tells how much *word i* should pay attention to *word j*.

(c) Scale and Normalize

To keep values stable, divide by $\sqrt{d_k}$ (dimension of key vector) and apply **softmax**:

$$\alpha_{i,j} = \text{softmax}\left(\frac{q_i \cdot k_j^T}{\sqrt{d_k}}\right)$$

Each row $\alpha_{i,*}$ now represents the attention distribution for word i.

(d) Weighted Sum to Get Output

Each new word representation is a weighted sum of all value vectors:

$z_i = \sum_j \alpha_{i,j} v_j$

So each z_i is a **contextualized embedding** — it now encodes meaning based on the entire sentence.

(e) Combine into Matrix Form

All together (in one elegant formula):

$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

5️⃣ Example Intuition

Sentence:

> “The animal didn’t cross the street because it was too tired.”

Which word does **“it”** refer to? → “animal”.

Self-Attention helps the model figure this out:

when computing the embedding for “it”, the model assigns **high attention weight to “animal”** and lower to others.

6️⃣ Why It’s Powerful

Feature	Self-Attention Advantage
Parallelizable	Unlike RNNs, all tokens are processed simultaneously (matrix ops).
Global context	Every word can directly access every other word’s info.
Long-range dependencies	Easily captures distant relationships (e.g., subject–verb across clauses).
Interpretability	Attention weights show which words influence each other.

7️⃣ Multi-Head Self-Attention

In practice, Transformers use **multiple attention heads** (e.g., 8 or 12).

Each head learns to focus on different relationships:

- one head might focus on syntax (subject–verb),
- another on semantics (adjective–noun).

Formally:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where,

$$(\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V))$$

8️⃣ Visual Intuition

```
"The"  → attends to → itself (slightly)
"animal"  → attends to → "it"
```

```
"it"  → attends to → "animal"
"tired"  → attends to → "too"
```

All words exchange information through the attention matrix.

9️⃣ Summary Table

Concept	Regular Attention	Self-Attention
Source	Encoder–Decoder (cross-sequence)	Within same sequence
Purpose	Aligns input and output words	Finds relationships among input words
Inputs	Query from decoder, Keys/Values from encoder	Q, K, V all from same sequence
Example use	Translation (Seq2Seq)	Transformers (BERT, GPT, etc.)

🧾 Key Takeaway

> Self-Attention is the mechanism that lets every token in a sequence interact with every other token directly — learning global relationships efficiently and in parallel.

It is:

- the **core building block** of the **Transformer**,
- the reason it **outperforms RNNs/LSTMs**, and
- the foundation for modern NLP models like **BERT, GPT, and T5**.

Multi-Head Self-Attention

🧠 Recall: What is Self-Attention?

Self-attention lets every token in a sequence **look at (attend to)** other tokens to gather context.

For example, in the sentence

> “The cat sat on the mat.”

the representation of “cat” might attend more to “sat” than to “mat” — because they are semantically related.

🧩 Multi-Head Self-Attention — The Core Idea

A **single** self-attention mechanism can capture **only one type of relationship** at a time (for example, syntactic or positional).

👉 But **Multi-Head Self-Attention** runs **several self-attention operations in parallel**, each learning a **different type of relationship** between words.

Each “head” can focus on a different aspect:

- Head 1 might learn *subject–verb* dependencies.
- Head 2 might learn *long-range context*.
- Head 3 might focus on *positional information*, etc.

⚙️ Step-by-Step Mechanics

Let’s denote:

- Input sequence: $X = [x_1, x_2, ..., x_n]$
- Each token x_i is a vector of dimension d_{model}.

1️⃣ Linear Projections

We project X into three different spaces using learned weight matrices:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where Q, K, and V are the **Query**, **Key**, and **Value** matrices.

Each attention head has its own W_Q, W_K, W_V.

2️⃣ Scaled Dot-Product Attention (for one head)

Each head computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- QK^T → measures similarity between tokens
- Divide by $\sqrt{d_k}$ to prevent large gradients
- Softmax → converts similarities to attention weights
- Multiply by V → weighted sum of values

3️⃣ Multiple Heads in Parallel

Instead of one head, we compute several (say h = 8):

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Each head has smaller dimension $d_k = d_{model}/h$.

4️⃣ Concatenate + Linear Projection

Finally, we concatenate all heads and pass them through another linear layer:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W_O$$

where W_O is a learnable matrix that mixes information from all heads.

🎯 Intuition Recap

Concept	Meaning
Self-Attention	Looks at all words to decide what matters for each token
Multi-Head	Runs multiple attention "views" in parallel to capture different types of relationships
Benefit	Better contextual understanding — richer and more expressive representations

---

🧩 Example

Sentence: "She ate the apple because she was hungry."

- Head 1 may focus on *coreference* — "she" ↔ "she"
- Head 2 may focus on *cause–effect* — "ate" ↔ "hungry"
- Head 3 may focus on *object relations* — "ate" ↔ "apple"

Each head’s output gets combined → the model gains multi-perspective understanding.