SCALER School of Technology × Priyansh Saxena

The Transformer: Self-Attention and Architecture

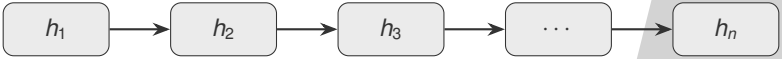
Self-Attention • Q/K/V • Multi-Head Attention • Positional Encoding • Architecture

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 7 — Session 1/2

9 June 2026

The Problem — Sequential Bottleneck



- RNNs process **one token at a time** — cannot parallelize
- Even LSTMs struggle beyond $\sim$200 tokens
- Attention helped, but still built **on top of** RNNs
- The question: *"What if we used ONLY attention?"*

2017: Vaswani et al. — *"Attention Is All You Need"*

Today’s Roadmap

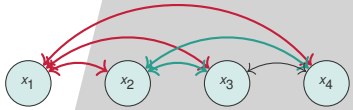
By the end: you’ll understand the architecture behind **GPT, BERT, and modern AI**.

Self-Attention — The Core Idea

- Every token attends to **every other token** simultaneously
- No sequential chain — all positions computed in **parallel**
- Each token asks: “which other tokens are relevant to me?”

RNN: token n must wait for tokens $1 \ldots n-1$

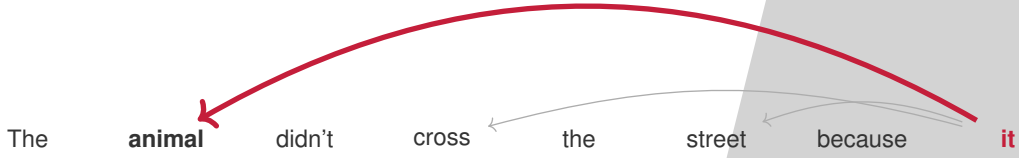
Self-Attention: all tokens processed at once



All-to-all connections

Self-Attention Example

*"The animal didn't cross the street because **it** was too tired."*



- "it" attends most strongly to "animal" (high attention weight)
- The model **learns** these attention patterns during training
- No explicit coreference resolution — emerges from data

Queries, Keys, and Values

Each token embedding is projected into three vectors:

- **Query** q: “What am I looking for?”
- **Key** k: “What do I offer?”
- **Value** v: “What info do I carry?”

$$q_i = W_Q \cdot x_i$$
$$k_i = W_K \cdot x_i$$
$$v_i = W_V \cdot x_i$$

Analogy: Library Search

You have a question (query).
Books have titles (keys).
Books contain content (values).

Match query to titles $\rightarrow$ read the best-matching content.

Scaled Dot-Product Attention

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

1. $QK^\top$ — pairwise similarity scores (dot products)
2. Divide by $\sqrt{d_k}$ — prevent softmax saturation
3. Softmax — normalize each row to a probability distribution
4. Multiply by V — weighted combination of value vectors

"Match queries to keys, then read the corresponding values."

Step-by-Step Computation

1. Compute $QK^\top$

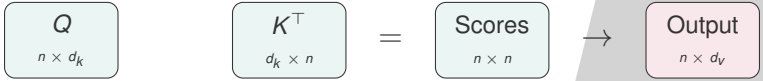
3×4 times 4×3
$= 3 \times 3$ score matrix

2. Scale & Softmax

Divide by $\sqrt{4} = 2$
Softmax each row

3. Weighted Sum

3×3 times 3×4
$= 3 \times 4$ output



Why Divide by $\sqrt{d_k}$?

Without scaling:

- Dot products grow proportionally to d_k
- Large values $\rightarrow$ softmax becomes very "peaked"
- Peaked softmax $\rightarrow$ gradients near zero (saturation)

With $\sqrt{d_k}$ scaling:

- Normalizes variance back to ~ 1
- Softmax stays in a "learnable" regime
- Healthy gradient flow

Same principle as Xavier initialization — control the scale of activations to keep gradients healthy.

Quick Check — Self-Attention & Q/K/V

1. In self-attention, where do Q, K, V come from?
2. What does $QK^\top$ compute?
3. Why do we apply softmax?

Quick Check — Self-Attention & Q/K/V

1. In self-attention, where do Q, K, V come from?
2. What does $QK^\top$ compute?
3. Why do we apply softmax?

Answers

1. Same input, **different learned weight matrices** W_Q, W_K, W_V
2. **Pairwise similarity** between all token pairs
3. Normalize to a **probability distribution** for the weighted average

Multi-Head Attention — Why One Head Isn't Enough

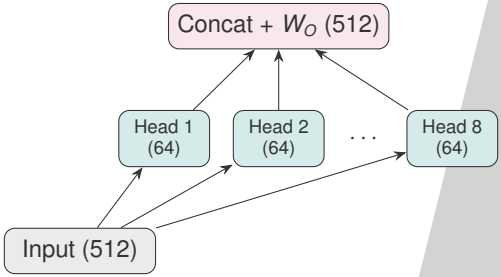
- A single attention head learns **one type** of relationship
- But language has **many** relationship types:

Syntax | Semantics | Coreference | Proximity

Solution: Run multiple attention heads in parallel.
Each head gets its own W_Q, W_K, W_V (smaller dimension).

Multi-Head Attention — How It Works

1. **Split:** d_{model} into h heads: $d_k = d_{\text{model}}/h$
2. **Attend:** Each head computes attention independently
3. **Concatenate:** All head outputs joined together
4. **Project:** $W_O \times \text{concat} \rightarrow d_{\text{model}}$



Same total compute as one big 512-dim attention head.

What Different Heads Learn

Specialization **emerges from training** — not programmed:

- **Head 1:** subject–verb agreement
- **Head 2:** adjective–noun proximity
- **Head 3:** pronoun $\rightarrow$ antecedent
- **Head 4:** punctuation boundaries

Like multiple reviewers reading the same essay — one checks grammar, one checks logic, one checks style.

Key Insight

8 heads $\times$ 64 dims each $=$ 512 total. Same parameter count, but 8 different "perspectives" on the input.

Coming Up Next

Part 2: Positional Encoding + Full Transformer Architecture

The Position Problem

- Self-attention is **permutation-invariant** — treats input as a **set**
- Without position info: “Dog bites man” = “Man bites dog”
- RNNs got position for free (sequential processing)
- We need to **explicitly inject** order information

Positional Encoding — Sinusoidal

Add a position-specific vector to each token embedding:

$$PE_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) \qquad PE_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right)$$

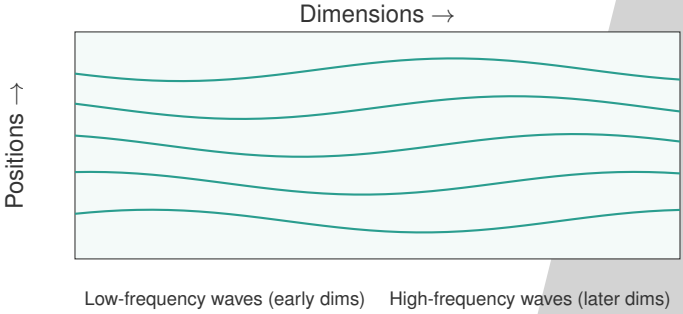
Properties:

- Each position gets a unique “fingerprint”
- Relative positions have consistent patterns
- Can generalize to longer sequences than seen in training

Analogy

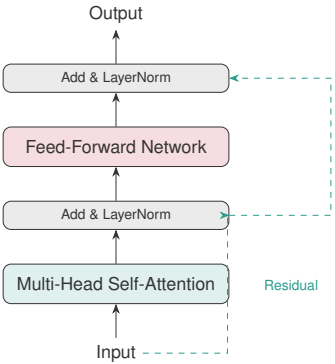
Numbering pages in a shuffled deck. Without numbers, you can’t reconstruct the order. Sin/cos = unique page number for each position.

Positional Encoding — Visualization



- Nearby positions have **similar** encodings (high dot product)
- Distant positions have **different** encodings (low dot product)
- The model can learn to extract relative position from these signals

Transformer Encoder Block



Stack $N = 6$ of these blocks. Residual connections = like ResNet!

Transformer Decoder Block

Three sub-layers (not two):

1. **Masked Self-Attention** — can't see future tokens
2. **Cross-Attention** — attend to encoder output
3. **Feed-Forward Network** — per-position transformation

Masking:

Set future positions to $-\infty$ before softmax

$\rightarrow$ softmax$(-\infty) = 0$

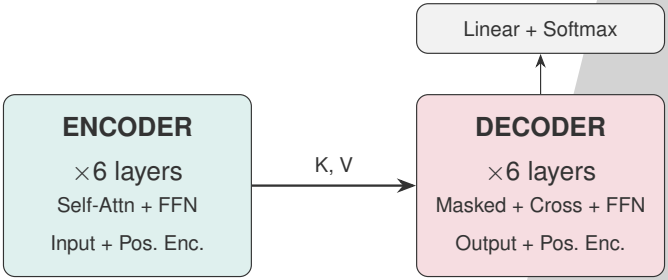
Position 5 cannot peek at position 6

Cross-Attention:

Decoder provides **queries**
Encoder provides **keys** and **values**

"Decoder asks questions, encoder has answers"

Full Transformer Architecture



Encoder reads input → decoder generates output one token at a time.

Why Transformers Beat RNNs

	RNN	LSTM	Transformer
Parallelizable	✗	✗	✓
Path length (any two tokens)	$O(n)$	$O(n)$	$O(1)$
Scales with compute	Limited	Limited	Excellent
Long-range dependencies	Poor	Good	Excellent
Memory (attention)	$O(1)$	$O(1)$	$O(n^2)$

Trade-off: $O(n^2)$ memory for attention — but GPUs handle it well for typical lengths.

Computational Considerations

Self-attention complexity: $O(n^2 \cdot d)$

- $n = 512$: manageable (standard BERT)
- $n = 2048$: GPT-3 context window
- $n = 10000+$: problematic $\rightarrow$ efficient attention research

"Attention Is All You Need" base model:
65M parameters, 8 GPUs, $\sim$12 hours (2017)

Why it still wins

RNN: n sequential steps (slow on GPU).

Transformer: 1 parallel step (fast on GPU).

Even with $O(n^2)$ memory, the parallelism wins for typical sequence lengths.

Final Quiz

1. What is the time complexity of self-attention w.r.t. sequence length?
2. Why does the decoder use masked attention?
3. Name one advantage AND one disadvantage of transformers vs. LSTMs.

Final Quiz

1. What is the time complexity of self-attention w.r.t. sequence length?
2. Why does the decoder use masked attention?
3. Name one advantage AND one disadvantage of transformers vs. LSTMs.

Answers

1. $O(n^2)$ — every token attends to every other token
2. Cannot peek at future tokens during autoregressive generation
3. **Advantage:** Parallel + $O(1)$ path. **Disadvantage:** $O(n^2)$ memory

Recap & Key Takeaways

- **Self-Attention:** Every token attends to every other — parallel, not sequential
- **Q/K/V:** Learned projections that compute relevance scores
- **Multi-Head:** Parallel attention for different relationship types
- **Positional Encoding:** Sin/cos signals inject order into a permutation-invariant model
- **Architecture:** Encoder-decoder with residual connections, layer norm, FFN

Next session: BERT and GPT — how to **use** transformers for real tasks
(Pre-training, fine-tuning, and the paradigm shift)

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn