Attention and the Transformer Architecture

Robert Minneker

2026-02-05

Sources

Content derived from: J&M Ch. 8 (Transformers)

Administrivia

- Midterm feedback form is out!
  - Fill out via Canvas path, instructions on Ed
- Please start A2 if you haven't already!

Learning Objectives

By the end of this lecture, you will be able to:

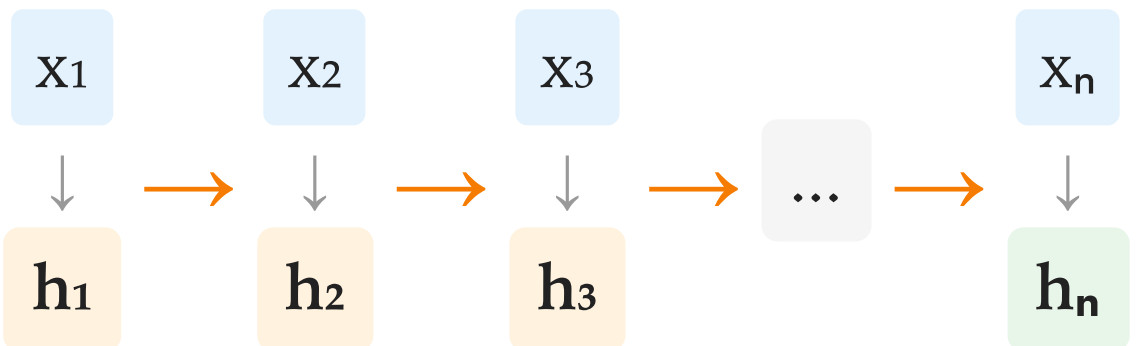
1. Explain why RNNs/LSTMs were replaced by attention-based architectures
2. Derive and compute scaled dot-product self-attention
3. Describe how multi-head attention captures diverse relationships
4. Trace information flow through a transformer block
5. Explain how causal masking enables autoregressive generation

Part 1: From Sequential to Parallel Processing

RNNs process sequences one token at a time

Recurrent hidden state update:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b)$$



- Each step depends on the previous—**no parallelization possible**
- Information must flow through the entire chain

Sequential processing creates three critical problems

1. Vanishing Gradients

Gradients shrink exponentially over long sequences

$\partial L/\partial h_1 \approx 0$ when n is large

2. Memory Bottleneck

All history compressed into fixed-size vector

$h \in \mathbb{R}^d$ for any sequence length

3. Training Speed

Cannot parallelize across time steps

$O(n)$ sequential operations

LSTMs and GRUs mitigated vanishing gradients but couldn't solve the parallelization problem.

Attention allows direct connections between all positions

RNN: Sequential Path

$x_1 \rightarrow \rightarrow \rightarrow x_n$

n steps to connect distant tokens

Attention: Direct Path

$x_1 \leftrightarrow x_n$

1 step to connect any tokens

Key insight: Every token can directly "look at" every other token in a single operation.

The transformer architecture revolutionized NLP in 2017

2017	→	2018	→	2019	→	2020	→	2022+
"Attention Is All You Need"		BERT, GPT		GPT-2, T5		GPT-3		ChatGPT, Claude, LLaMA

Vaswani et al. (2017): "Attention Is All You Need"

- Originally designed for machine translation
- Quickly became dominant for ALL NLP tasks
- Every modern LLM is based on the transformer architecture

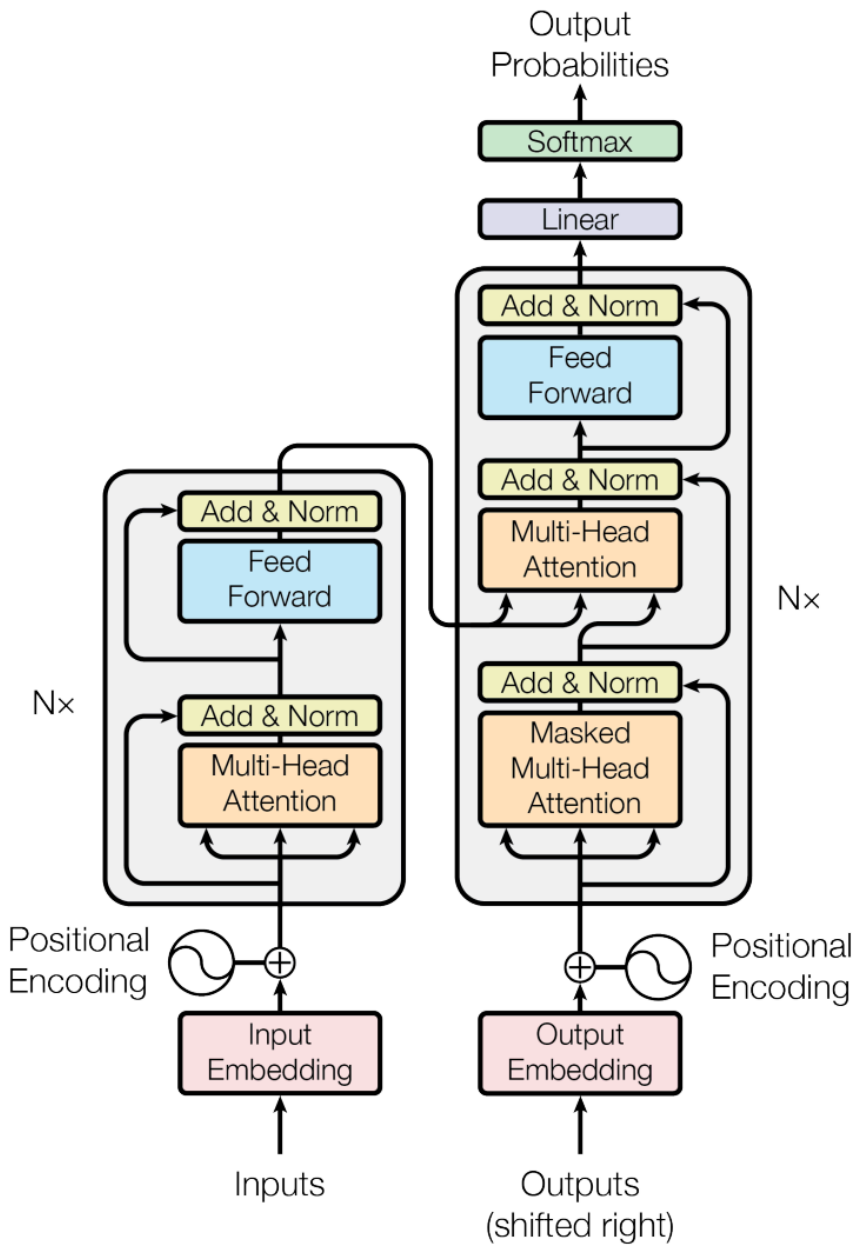


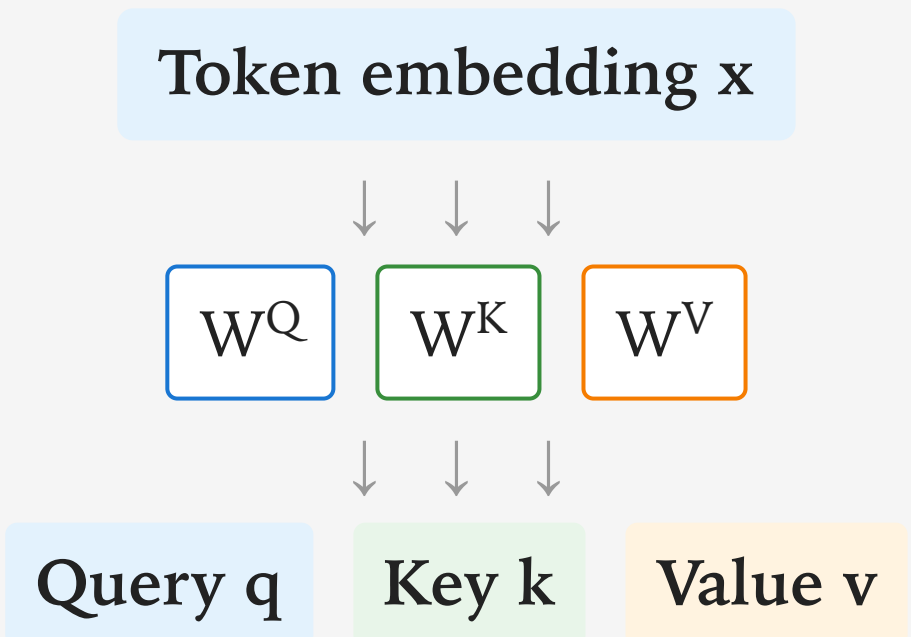
Figure 1: The Transformer - model architecture.

Part 2: Self-Attention Mechanism

Self-attention projects each token into query, key, and value vectors

For input embedding $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{x}_i W^Q, \quad \mathbf{k}_i = \mathbf{x}_i W^K, \quad \mathbf{v}_i = \mathbf{x}_i W^V$$



- $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are **learned** projection matrices

The QKV intuition: retrieval from a soft dictionary

Query (Q)

"What am I looking for?"

Represents the current token's information needs

Key (K)

"What do I contain?"

Represents what information this token offers

Value (V)

"What do I provide?"

The actual information to be retrieved

Analogy: Like a search engine where:

- Query = your search terms
- Key = document titles/metadata
- Value = document contents

Attention scores measure relevance via dot products

$$\text{score}(i, j) = \mathbf{q}_i \cdot \mathbf{k}_j = \mathbf{q}_i \mathbf{k}_j^\top$$

Computing Attention for "sat"

"The cat **sat** on the mat"

Token j	$\mathbf{q}_{\text{sat}} \cdot \mathbf{k}_j$	Interpretation
The	0.8	Low relevance
cat	**4.2**	High—subject of "sat"
on	1.5	Medium
the	0.6	Low relevance
mat	**3.8**	High—location of "sat"

Scaling prevents softmax saturation

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

Why divide by $\sqrt{d_k}$?

Without Scaling

`scores = [8.5, 42, 3.2, 42]`

`softmax ≈ [0, 0.55, 0, 0.45]`

Extreme values → near one-hot → vanishing gradients

With Scaling (d_k=64)

`scores/8 = [1.1, 5.3, 0.4, 5.2]`

`softmax ≈ [0.01, 0.49, 0, 0.49]`

Moderate values → smooth distribution → healthy gradients

The output is a weighted sum of value vectors

$$\text{output}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j$$

where $\alpha_{ij} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}\right)$

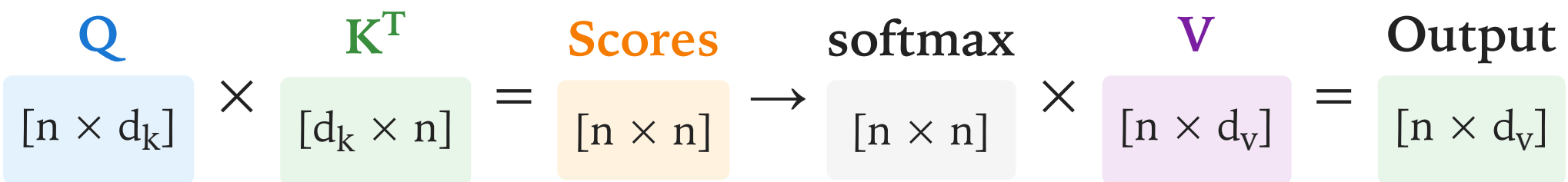
Output for "sat" = weighted combination of all values

$$\underset{\mathbf{v}_{\text{The}}}{0.05} + \underset{\mathbf{v}_{\text{cat}}}{\mathbf{0.40}} + \underset{\mathbf{v}_{\text{on}}}{0.15} + \underset{\mathbf{v}_{\text{the}}}{0.05} + \underset{\mathbf{v}_{\text{mat}}}{\mathbf{0.35}} = \mathbf{output}_{\text{sat}}$$

- High attention weight → that position contributes more
- Context is incorporated through this weighted aggregation

Matrix form: compute all attention outputs in parallel

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



- **One matrix multiply** computes all n^2 pairwise attention scores
- GPUs are highly optimized for matrix multiplication
- This enables massive parallelism

Part 3: Multi-Head Attention and Transformer Blocks

Multi-head attention runs several attention operations in parallel

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

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

Head 1	**Head 2**	**Head 3**	**Head 4**	**...**
Subject-verb	Adjective-noun	Coreference	Positional	Other patterns

- Each head has its **own** W^Q, W^K, W^V matrices
- Heads learn to specialize in different relationship types
- Typical: 8-16 heads with $d_k = d_{\text{model}}/h$

Different heads capture different linguistic relationships

Sentence: "The lawyer who the witness saw left"

Head A: Subject-verb

"left" attends strongly to "lawyer"
(captures who performed the action)

Head B: Relative clause

"saw" attends to "witness" and "lawyer"
(tracks nested clause structure)

Research finding: Attention heads in trained models often align with linguistic structure (“What Does BERT Look At? An Analysis of BERT’s Attention” Clark et al., 2019).

A transformer block combines attention with a feed-forward network

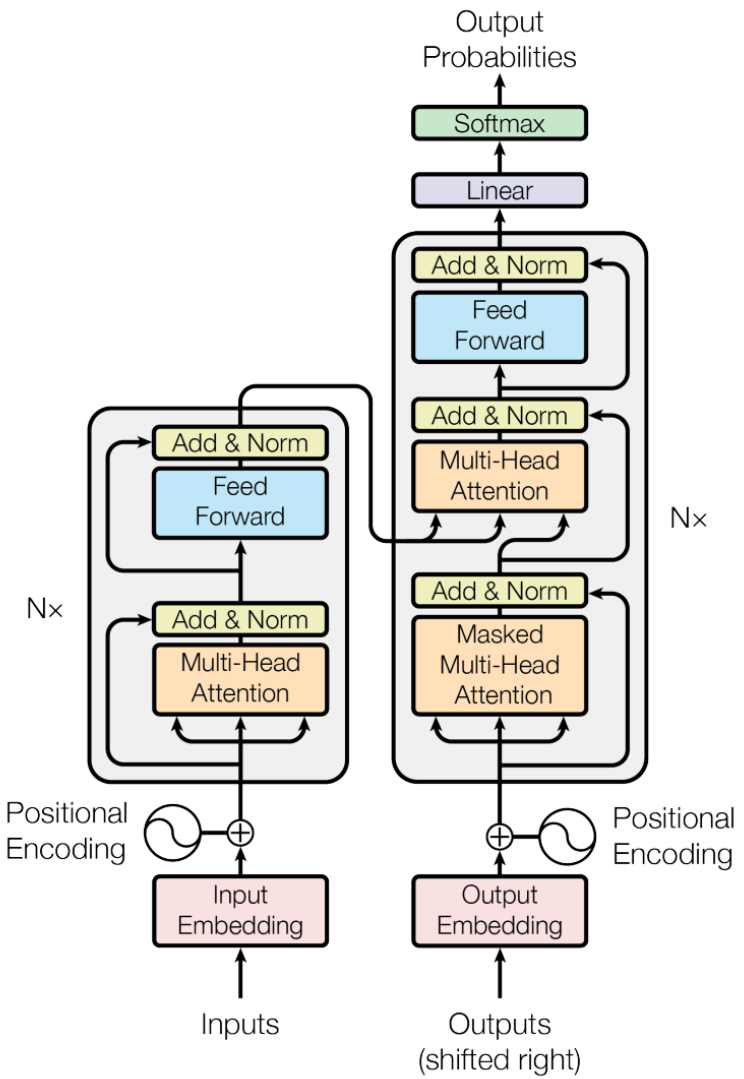


Figure 1: The Transformer - model architecture.

The feed-forward network processes each position independently

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

or with GELU (used in GPT-2+):

$$\text{FFN}(x) = \text{GELU}(xW_1)W_2$$

- **No interaction between positions**—applied identically to each token
- Expands to 4× dimension, then projects back

Residual connections and layer normalization stabilize training

Residual connection:

$$\text{output} = \text{sublayer}(x) + x$$

Layer normalization:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sigma + \epsilon} + \beta$$

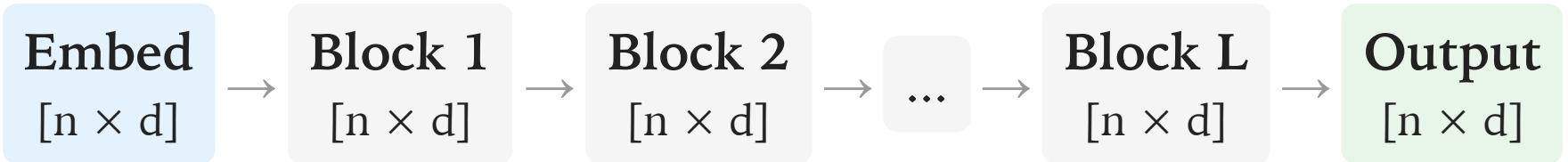
Why Residuals?

- Gradient flows directly through addition
- Enables training very deep networks
- Each layer learns a "delta"

Why LayerNorm?

- Normalizes activations per token
- Stabilizes training dynamics
- γ, β are learned parameters

Transformers maintain uniform dimensionality throughout



- Every token representation is d-dimensional throughout
- No reshaping between layers—just stack and go
- Simplifies architecture and enables easy layer scaling

Part 4: Causal Masking and Positional Encoding

Causal masking prevents attending to future tokens

For autoregressive generation (GPT, LLaMA, Claude):

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

where $M_{ij} = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{if } j > i \end{cases}$

	The	cat	sat	on
The	✓	−∞	−∞	−∞
cat	✓	✓	−∞	−∞
sat	✓	✓	✓	−∞
on	✓	✓	✓	✓

Causal masking enables the language modeling objective

Autoregressive factorization:

$$P(x_1, \ldots, x_n) = \prod_{t=1}^{n} P(x_t \mid x_1, \ldots, x_{t-1})$$

Predicting Each Token

```
P(The) → predict first token
P(cat | The) → see "The"
P(sat | The cat) → see "The cat"
```

- At each position, predict the next token using only past context
- Training and inference use the **same masking**—no distribution shift

Self-attention is permutation-invariant without position information

Problem: Attention treats input as a **bag of tokens**

$$\text{Attention}(\{x_1, x_2, x_3\}) = \text{Attention}(\{x_3, x_1, x_2\})$$

Without Position

`"dog bites man" ≡ "man bites dog"`

Same tokens = same representation!

With Position

`"dog bites man" ≠ "man bites dog"`

Position breaks the symmetry

Solution: Add positional information to embeddings

Positional encodings inject sequence order information

Sinusoidal (original transformer):

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

Sinusoidal (fixed)

- No learned parameters
- Can extrapolate to longer sequences
- Used in original transformer

Learned

- Embedding lookup by position
- More flexible
- Used in GPT, BERT

Rotary (RoPE)

- Rotation in embedding space
- Better length generalization
- Used in LLaMA, GPT-NeoX

Encoder-only vs. decoder-only architectures

Encoder-Only (BERT)

Bidirectional attention

- See all tokens at once
- Good for classification, NER
- Cannot generate text

Decoder-Only (GPT)

Causal attention

- See only past tokens
- Good for generation, chat
- Dominant for modern LLMs

Encoder-Decoder (T5)

Both + cross-attention

- Encoder sees all input
- Decoder generates output
- Good for translation, summarization

Part 5: Three Types of Attention in the Original Transformer

The original transformer uses three distinct attention mechanisms

1. Encoder Self-Attention

Bidirectional

Every token attends to every other token

2. Masked Self-Attention

Causal (unidirectional)

Tokens only attend to past positions

3. Cross-Attention

Encoder → Decoder

Decoder queries attend to encoder outputs

Each serves a different purpose in the encoder-decoder architecture.

Type 1: Bidirectional self-attention in the encoder

Used in: Encoder layers of transformer, BERT

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

Q, K, V all come from the same input

Encoder Input X → $Q = XW^Q$

Encoder Input X → $K = XW^K$

Encoder Input X → $V = XW^V$

- **No masking:** Every position can attend to every other position
- Captures full bidirectional context for understanding tasks

Bidirectional attention: full connectivity

	The	cat	sat	down
The	✓	✓	✓	✓
cat	✓	✓	✓	✓
sat	✓	✓	✓	✓
down	✓	✓	✓	✓

- All n^2 attention weights are computed
- Each token's output incorporates information from the entire sequence
- Ideal for **encoding** where full context is available

Type 2: Masked self-attention in the decoder

Used in: Decoder layers, GPT, LLaMA, Claude

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

Q, K, V all come from decoder input (with causal mask)

Decoder Input Y → $Q = YW^Q$, $K = YW^K$, $V = YW^V$ + Causal Mask M

- **Lower-triangular mask:** Position i can only attend to positions ≤ i
- Enables autoregressive generation

Masked attention: lower-triangular connectivity

	Le	chat	est	assis
Le	✓	✗	✗	✗
chat	✓	✓	✗	✗
est	✓	✓	✓	✗
assis	✓	✓	✓	✓

- The decoder is generating a French translation
- Each output token only sees previously generated tokens
- Prevents “cheating” by looking at future outputs during training

Type 3: Cross-attention connects decoder to encoder

Used in: Encoder-decoder models (original transformer, T5, BART)

Q from decoder, K and V from encoder

Decoder State Y → $Q = YW^Q$

Encoder Output Z → $K = ZW^K$, $V = ZW^V$

$$\text{CrossAttention}(Y, Z) = \text{softmax}\left(\frac{(YW^Q)(ZW^K)^\top}{\sqrt{d_k}}\right)(ZW^V)$$

Cross-attention: decoder queries encoder

English (Encoder): "The cat sat down"
French (Decoder): "Le chat est assis"

Decoder↓ Encoder→	**The**	**cat**	**sat**	**down**
Le	**0.85**	0.10	0.03	0.02
chat	0.08	**0.82**	0.05	0.05
est	0.05	0.15	**0.70**	0.10
assis	0.02	0.08	**0.50**	**0.40**

- Matrix is [decoder length × encoder length]—**not square!**
- "assis" attends to both "sat" and "down" (semantic alignment)

Comparing the three attention types

	Encoder Self-Attn	Masked Self-Attn	Cross-Attention
Query source	Encoder input	Decoder input	Decoder state
Key/Value source	Encoder input	Decoder input	Encoder output
Masking	None	Causal (lower-tri)	None
Matrix shape	[n × n]	[m × m]	[m × n]
Purpose	Understand input	Generate output	Connect both

Where n = encoder sequence length, m = decoder sequence length

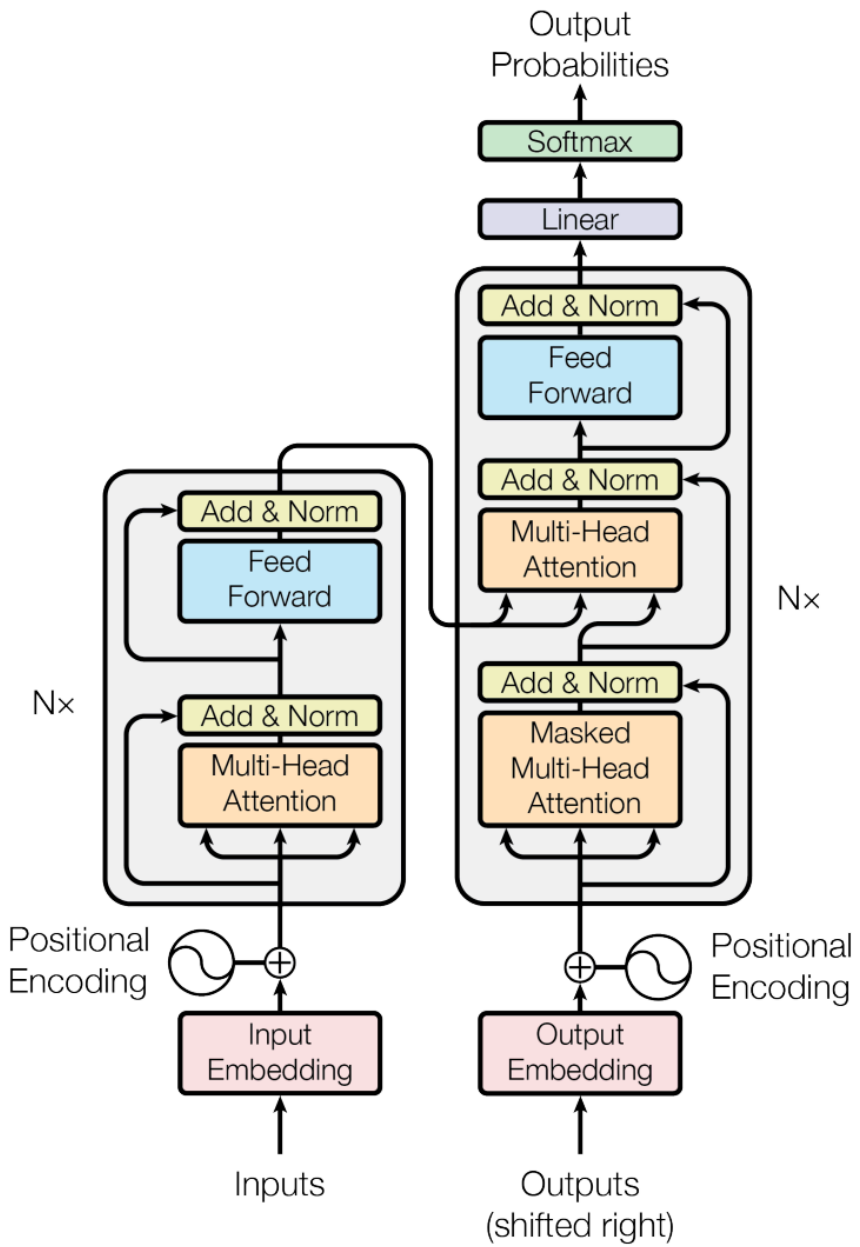


Figure 1: The Transformer - model architecture.

Modern architectures often simplify

Encoder-Only (BERT)

Uses: **Bidirectional self-attention only**

- Great for understanding/classification
- Cannot generate text autoregressively
- Examples: BERT, RoBERTa, ELECTRA

Decoder-Only (GPT)

Uses: **Masked self-attention only**

- Great for generation and chat
- Scales very well with compute
- Examples: GPT, LLaMA, Claude

Key insight: Decoder-only models have become dominant because they're simpler and scale better, while still being able to "understand" via in-context learning.

Summary: Day 1 Key Takeaways

1. **Attention replaces recurrence:** Direct connections between all positions enable parallelism and long-range dependencies
2. **QKV mechanism:** Query asks, key answers, value provides—scaled dot-product measures relevance
3. **Multi-head attention:** Multiple heads capture diverse relationship types in parallel
4. **Transformer blocks:** Attention + FFN with residuals and layer norm, stacked L times
5. **Causal masking:** Prevents attending to future tokens, enabling autoregressive generation
6. **Position encodings:** Break permutation invariance to preserve sequence order

Coming Up Next

LLM Training and Alignment

- Self-supervised pretraining at scale
- Instruction tuning (SFT)
- Preference alignment with RLHF
- Scaling laws and compute-optimal training

Reading:

- J&M Chapter 8 (Transformers)
- "Attention Is All You Need" (Vaswani et al., 2017)