Attention, Self-Attention, and Transformer Algebra

Mathematics of Deep Learning — Chapter 10

Yin Yang

Foundation Books Project

Roadmap

From Recurrence to Attention

Queries, Keys, and Values

Scaled Dot-Product Attention

Attention as a Weighted Graph

Masks for Padding and Causality

Positional Information

Multi-Head Attention

Cross-Attention

The Transformer Block

Encoder-Decoder vs. Decoder-Only

Cost and Applied Checks

Summary

From Recurrence to Attention

The Running Phrase

This chapter develops attention from one short phrase:

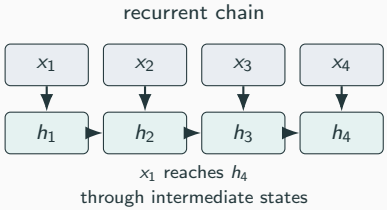
`not very good.`

- Last token `good` looks positive on its own.
- A sentiment model should also see the earlier `not`.
- Recurrence routes that information through intermediate states.
- Attention gives `good` a *direct* algebraic route to `not`.

The same three-token example will reappear across scores, masks, positions, heads, and full transformer blocks.

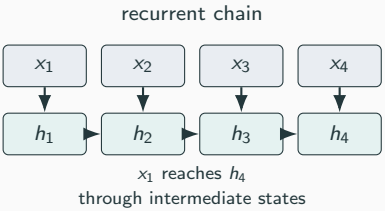
Three positions, one phrase — enough algebra to spot every mistake later models can make at scale.

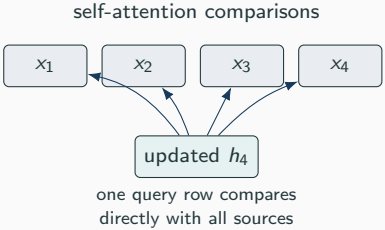
Recurrent Chain vs. Direct Comparison



(1) Recurrence is a chain of state updates.

Recurrent Chain vs. Direct Comparison





(1) Recurrence is a chain of state updates.

(2) Attention is an all-pairs comparison table.

Attention changes the graph: direct comparisons replace chained hops.

Sequence Matrix and Position Comparison

Stack T token embeddings as rows:

$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_T^\top \end{bmatrix} \in \mathbb{R}^{T \times d_{\text{model}}}.$$

- Subscript i marks *position*, not exponent.
- For `not very good`, $T = 3$ and $x_3^\top$ is `good`.
- A *position comparison* $S_{i,j}$ scores ordered pair (i, j): how compatible is reader i with source j?

Attention will form a table of these scores, normalize each row, and use the normalized weights to mix value vectors.

Direct comparison is a property of the architecture, not of any particular trained model.

Three Equations of Transformer Attention

The whole chapter unpacks the three lines

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}_{\mathrm{row}}(S), \qquad H = AV.$$

- $S \in \mathbb{R}^{T \times T}$: raw position-pair scores.
- $A \in \mathbb{R}^{T \times T}$: row-stochastic routing weights.
- $H \in \mathbb{R}^{T \times d_v}$: one mixed value vector per query row.

Masks, positional information, multi-head, cross-attention, and the transformer block all live around this score-normalize-mix pipeline.

The central object is the row-weight matrix A — a learned routing table over positions.

Queries, Keys, and Values

QKV: Three Learned Projections

Project the sequence matrix $X \in \mathbb{R}^{T \times d_{\text{model}}}$ three times:

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V.$$

- $W_Q, W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ — queries and keys share dimension d_k so dot products $q_i^\top k_j$ make sense.
- $W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ — value dim may differ; values are the things being mixed.
- Rows: $q_i^\top$ is query at i, $k_j^\top$ is key at j, $v_j^\top$ is value at j.

Names are useful, but W_Q, W_K, W_V are ordinary linear layers trained by backprop through the same loss as everything else.

Queries and keys must share d_k; the value dimension is free.

QKV Roles at a Glance

Object	Shape	Role in the layer
Q	$T \times d_k$	each row asks what one position should read
K	$T \times d_k$	each row is compared against queries
V	$T \times d_v$	each row is information available to be mixed
$QK^\top$	$T \times T$	raw position-pair scores
A	$T \times T$	row-stochastic routing weights
$H = AV$	$T \times d_v$	one mixed output row per query

For `not very good`: query q_3 is `good`, key k_1 is `not`. A large $q_3^\top k_1$ pulls v_1 into the updated representation h_3.

Reading attention by shape catches most implementation bugs before any arithmetic runs.

Scaled Dot-Product Attention

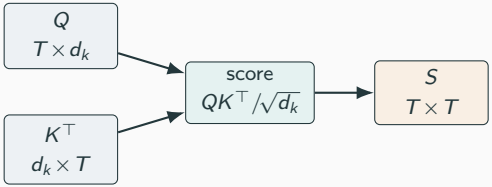
Score, Normalize, Mix

$$Q$$
$$T \times d_k$$

$$K^\top$$
$$d_k \times T$$

(1) Form queries and keys.

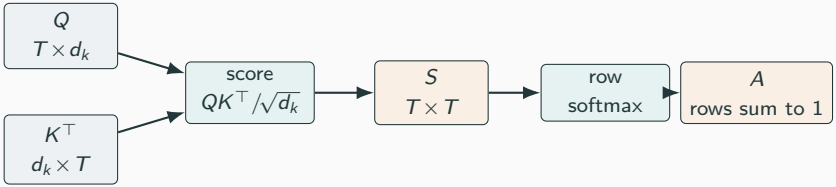
Score, Normalize, Mix



(1) Form queries and keys.

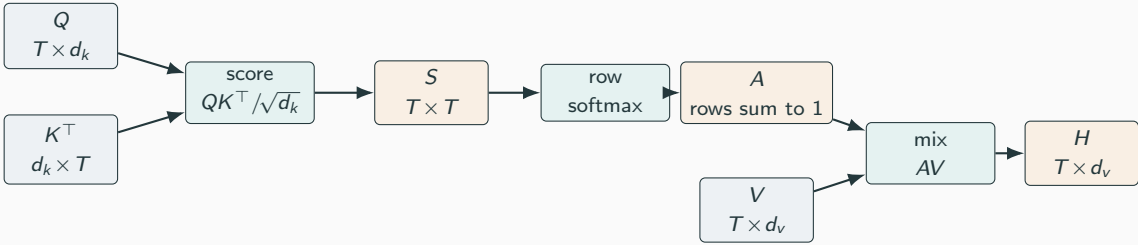
(2) Score every ordered pair, scaled by $\sqrt{d_k}$.

Score, Normalize, Mix



(1) Form queries and keys.
(2) Score every ordered pair, scaled by $\sqrt{d_k}$.
(3) Row-wise softmax: each row becomes a probability vector.

Score, Normalize, Mix



(1) Form queries and keys.
(2) Score every ordered pair, scaled by $\sqrt{d_k}$.
(3) Row-wise softmax: each row becomes a probability vector.
(4) Mix value rows by the row weights.

The $T \times T$ middle matrix records which source positions each query reads from.

Why Divide by $\sqrt{d_k}$?

Model query and key entries as zero-mean, unit-variance, independent coordinates. Then

$$\mathbb{E}[q^\top k] = 0, \quad \mathrm{Var}(q^\top k) = d_k, \quad \mathrm{Var}\left(\frac{q^\top k}{\sqrt{d_k}}\right) = 1.$$

- Without scaling, raw scores grow with d_k.
- Large scores push softmax toward one-hot rows.
- Sharp rows leak almost no gradient back through the layer.
- Numerical safety: subtract the row max before exponentiating.

The scale factor is an initialization argument, not a guarantee about trained weights — but the design pressure is real.

Tiny End-to-End Pass

Let $T = 3$, $d_k = d_v = 2$, identity projections, so

$$Q = K = V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad QK^\top = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

Row 3 scaled by $1/\sqrt{2}$ is approximately $(0.71, 0.71, 1.41)$:

$$\text{softmax}(0.71, 0.71, 1.41) \approx (0.25, 0.25, 0.50).$$

Mixed third output row:

$$h_3 \approx 0.25(1, 0) + 0.25(0, 1) + 0.50(1, 1) = (0.75, 0.75).$$

Identity projections only keep the arithmetic visible; trained W_Q, W_K, W_V would shape the row very differently.

One row of A is one probability distribution over source positions.

Hand Calculation on `not very good`

Suppose the score row at `good` is $s = (2, 0, 1)$. Then

$$\text{softmax}(2, 0, 1) \approx \frac{(7.39,\ 1.00,\ 2.72)}{11.11} \approx (0.67,\ 0.09,\ 0.24).$$

With scalar values $v_1 = -1$ (`not`), $v_2 = 0$, $v_3 = 2$:

$$h_3 \approx 0.67(-1) + 0.09(0) + 0.24(2) \approx -0.19.$$

- The negative value at `not` pulls the representation at `good` into negative territory.
- The architecture made this possible; learning made it useful.

Attention gives `not` a direct numerical handle on `good`.

Attention as a Weighted Graph

A Is a Directed Weighted Graph

Read A as a directed graph on positions $1, \ldots, T$:

$$A_{i,j} = \text{weight on edge } i \to j, \quad A_{i,j} \geq 0, \quad \sum_j A_{i,j} = 1.$$

- Row $A_{i,:}$ describes *where position i reads from*.
- Each output is a convex combination $h_i = \sum_j A_{i,j} v_j$.
- Different heads, layers, or tasks learn different graphs.

Plausible sentiment row for `good`: $A_{3,:} \approx (0.67, 0.09, 0.24)$ — mass on `not`.

Attention is a learned, soft, position-to-position routing table.

Attention Heatmaps: Useful, Not Sufficient

Query row	not	very	good	Interpretation
not	0.60	0.25	0.15	mostly self-focused
very	0.20	0.45	0.35	local context
good	0.67	0.09	0.24	reads negation strongly

Heatmaps are debug aids: they reveal padding leaks, future leaks, and collapsed heads. They are not full explanations of model reasoning — different internal mechanisms can yield the same prediction.

Attention weights are diagnostics, not causal accounts.

Masks for Padding and Causality

Adding a Mask Before Softmax

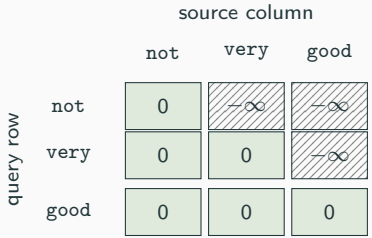
A mask $M \in (\mathbb{R} \cup \{-\infty\})^{T \times T}$ is added to scores *before* the softmax:

$$A = \text{softmax}_{\text{row}}(S + M), \quad M_{i,j} = \begin{cases} 0 & \text{allowed} \\ -\infty & \text{blocked} \end{cases}.$$

- $\exp(-\infty) = 0$, so blocked entries get $A_{i,j} = 0$ exactly.
- Padding mask: blocks fake positions added to align batch shapes.
- Causal mask: blocks future positions in a decoder.

Mask before softmax, not after $H = AV$ — otherwise the future has already leaked in.

Causal Mask on `not very good`



Lower triangular pattern: row i may read columns $j \leq i$ only. Hatched entries become weight 0 after the masked softmax.

A future leak during training looks like a working model that fails at prediction time.

Masked Row Arithmetic

Unmasked score row for `very` (position 2): $s = (2, 0, 1)$. Causal mask sets the third entry to $-\infty$:

$$(2, 0, -\infty) \xrightarrow{\text{softmax}} \frac{(e^2,\ e^0,\ 0)}{e^2 + e^0} \approx (0.88,\ 0.12,\ 0).$$

- Without the mask, `good` would receive nonzero weight from a position that should not yet see it.
- With the mask, that weight is *exactly* 0, not just small.
- The remaining row still sums to 1.

$-\infty$ in the score, not a small negative number, is what guarantees a zero weight.

Positional Information

Why Attention Needs a Position Signal

Pure self-attention treats X as an unordered set of rows: shuffling rows of X shuffles the output rows the same way.

Permutation equivariance. For a permutation matrix P_π,

$$\text{Attention}(P_\pi X W_Q, P_\pi X W_K, P_\pi X W_V) = P_\pi \ \text{Attention}(X W_Q, X W_K, X W_V).$$

- Score matrix transforms as $S' = P_\pi S P_\pi^\top$.
- Row-wise softmax preserves this conjugation.
- Therefore $A'V' = P_\pi A V$.

After mean-pooling, `not very good` and `good very not` give identical scores. Order is lost.

Without positions, attention cannot tell `not very good` from `good very not`.

Position-Aware Inputs

Combine token rows with position vectors before attention:

$$z_i = \Gamma(x_i, p_i), \qquad Z = X + P \text{ in the absolute additive case.}$$

- $P \in \mathbb{R}^{T \times d_{\text{model}}}$ holds one position vector per row.
- Original transformer: fixed sinusoidal P.
- Later: relative position biases, rotary position embeddings (RoPE) rotating q, k coordinates.
- Common purpose: break the permutation symmetry.

Token identity tells the model *what*; the position signal tells it *where*.

Multi-Head Attention

Multiple Comparison Spaces in Parallel

Run m attention computations side by side, each with its own projections:

$$H^{(r)} = \text{Attention}\big(XW_Q^{(r)},\, XW_K^{(r)},\, XW_V^{(r)}\big), \quad r = 1, \ldots, m,$$

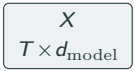
then merge with an output projection:

$$\text{MHA}(X) = \text{Concat}\big(H^{(1)}, \ldots, H^{(m)}\big)\, W_O.$$

- Each head explores a different learned subspace of comparisons.
- Heads may specialize, duplicate, or be hard to interpret.
- Concat shape: $T \times md_v$; $W_O \in \mathbb{R}^{md_v \times d_{\text{model}}}$.

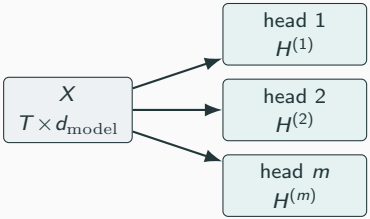
One attention is one comparison pattern; multi-head is several in parallel.

Multi-Head Flow



(1) Shared input X.

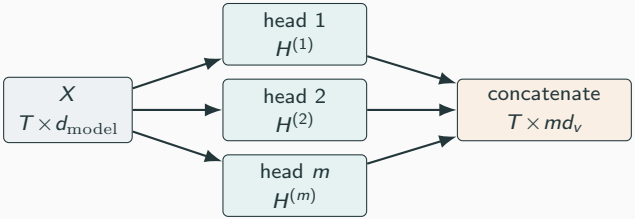
Multi-Head Flow



(1) Shared input X.

(2) m parallel attention computations.

Multi-Head Flow

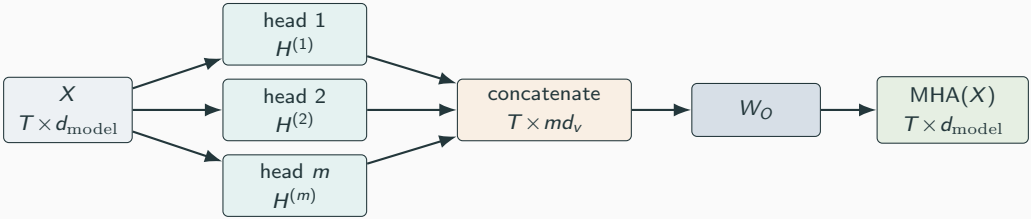


(1) Shared input X.

(2) m parallel attention computations.

(3) Concatenate head outputs along the feature axis.

Multi-Head Flow



(1) Shared input X.
(2) m parallel attention computations.
(3) Concatenate head outputs along the feature axis.
(4) W_O projects back to the model dimension.

Heads expand comparison capacity; W_O restores shape compatibility with the rest of the block.

Cross-Attention

Cross-Attention: Queries from One Sequence

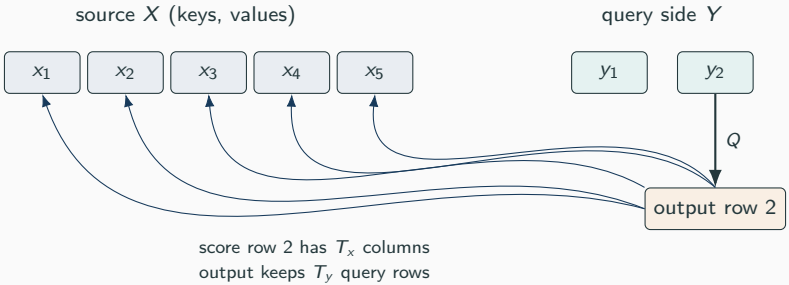
Self-attention takes Q, K, V from the same X. Cross-attention takes them from different sequences:

$$Q = YW_Q, \qquad K = XW_K, \qquad V = XW_V.$$

- $Y \in \mathbb{R}^{T_y \times d_{\text{model}}}$ controls the rows of A.
- $X \in \mathbb{R}^{T_x \times d_{\text{model}}}$ controls the columns of A and supplies values.
- Score shape: $T_y \times T_x$. Output shape: $T_y \times d_v$.
- Examples: decoder-to-encoder in translation; text-to-image tokens in vision-language models.

Queries pick the rows; keys and values pick the source.

Cross-Attention Picture



Same algebra as self-attention, with two sequences playing different roles.

The Transformer Block

Anatomy of One Block

Post-normalization variant:

$$U = \mathrm{LN}\big(X + \mathrm{MHA}(X)\big), \quad Y = \mathrm{LN}\big(U + \mathrm{FFN}(U)\big),$$

with position-wise feedforward

$$\mathrm{FFN}(u) = W_2\,\phi(W_1 u + b_1) + b_2.$$

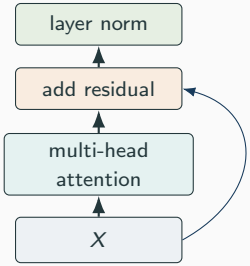
- Attention sublayer: communicates across positions.
- FFN sublayer: processes each row independently.
- Residual paths: each sublayer learns a *change* to the representation.
- Layer norm: rescales features within each token.

Communicate, then process row by row — then repeat.

Block Diagram

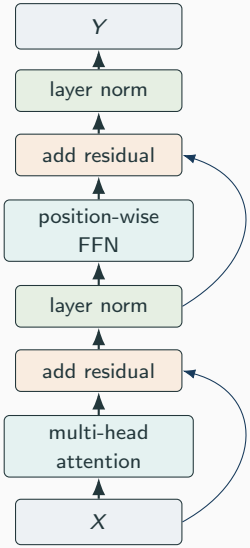


Block Diagram



mixes positions

Block Diagram



processes each row

mixes positions

Encoder-Decoder vs. Decoder-Only

Three Information Patterns

A transformer family is mostly a choice of source and mask:

$$A = \mathrm{softmax}_{\mathrm{row}}\left(\frac{QK^\top + M}{\sqrt{d_k}}\right).$$

Pattern	Query rows	Key/value rows	Allowed columns
Encoder self-attn	input X	same input X	all non-padding
Decoder-only self-attn	generated prefix	same prefix	$j \le i$ in row i
Encoder-decoder cross-attn	decoder states Y	encoder states X	non-padding source

Encoder models (BERT-style) read full inputs; decoder-only (GPT-style) keep the causal information set; encoder-decoder (T5-style) add a second source for text-to-text conditioning.

Family names label *mask and source*, not different algebra.

Causal Rows Are Prefix Functions

Claim. Under causal masking, output row i depends only on $x_1, \ldots, x_i$ — never on $x_{i+1}, \ldots, x_T$.

Why.

- Mask: $M_{i,j} = -\infty$ for $j > i$ gives $A_{i,j} = 0$.
- Mixed row i is a weighted sum of $v_1, \ldots, v_i$ only.
- Residual add, layer norm, position-wise FFN all act row by row.
- Stacking blocks preserves the prefix property by induction.

Consequence: causal transformer blocks are exactly the algebra needed inside an autoregressive factorization $\mathbb{P}(x_{i+1} \mid x_{\leq i})$.

Causal mask + row-wise sublayers = prefix-only outputs at every depth.

An Expressivity Note

Universal approximation (informal). For fixed maximum length and compact input domain, transformer architectures with positional encodings can approximate arbitrary continuous sequence-to-sequence functions to any desired accuracy.

- Without positions, only *permutation-equivariant* maps are reachable.
- This is an expressivity result, not a training guarantee.
- Says *nothing* about how many heads, layers, examples, or optimizer steps are needed.

"Can represent" is not "will learn" — expressivity is a ceiling, not a recipe.

Cost and Applied Checks

Quadratic Cost in Sequence Length

Full self-attention forms a $T \times T$ score matrix per head:

Length T	Scores T^2	Practical reading
16	256	tiny example
64	4,096	small paragraph or patch grid
256	65,536	moderate context
1024	1,048,576	>1M pairwise scores per head

Per head: $QK^\top$ costs $O(T^2 d_k)$; AV costs $O(T^2 d_v)$. Doubling T multiplies score count by four. Real memory also depends on batch, heads, precision, and saved activations.

Doubling the context is not twice the work — it is four times the score matrix.

Shape and Mask Audits

Before training a transformer block, audit the algebra:

- $QK^\top$ has the intended $T_{\text{query}} \times T_{\text{source}}$ shape.
- Padding columns receive attention weight exactly 0 after masking.
- Causal row i assigns weight 0 to every column $j > i$.
- A position signal is present whenever order matters.
- Heatmaps are inspected as diagnostics — never as full explanations.

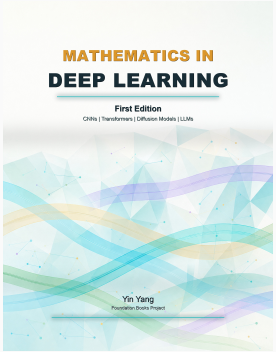
A correct loss cannot rescue a wrong score shape or a leaky mask.

Summary

Chapter in One Slide

- **Three equations.** $S = QK^\top/\sqrt{d_k}$, $A = \mathrm{softmax}_{\mathrm{row}}(S)$, $H = AV$ — score, normalize, mix.
- **QKV.** Learned projections; Q, K share d_k, V may differ.
- **Graph view.** A is a row-stochastic routing table; outputs are convex combinations of values.
- **Masks.** Padding and causal masks add to S before softmax; blocked entries become exactly 0.
- **Positions.** Self-attention is permutation equivariant; positional vectors break the symmetry.
- **Multi-head.** Several comparison spaces in parallel, concatenated, projected by W_O.
- **Cross-attention.** Queries from one sequence, keys and values from another.
- **Block & families.** Attention + residual + LN + FFN; encoder, decoder-only, encoder-decoder differ in mask and source.
- **Cost.** Scores grow as T^2. Audit shapes and masks before trusting losses.

Next: probability objectives, decoding, and the autoregressive language-modeling loop built on top of causal transformer blocks.



Mathematics in Deep Learning

Chapter overview slides for the book by Yin Yang.

Book page	`foundation-books.github.io/mathematics-in-deep-learning`
Code	`github.com/foundation-books/mathematics-in-deep-learning-code`
Paperback	`amazon.com/dp/B0H1ZZHDT2`
Hardcover	`amazon.com/dp/B0H1ZL11MJ`
Kindle	`amazon.com/dp/B0GX32M8SP`
License	CC BY-NC-ND 4.0

These free overview slides may be shared non-commercially with attribution and without modifications. Full explanations, exercises, and companion-code context are in the book and linked repository.