Data Science and AI for Business

Lecture 3b: The Mathematics of Attention and Transformers

Dr. (James) Yunying Huang

Fordham Gabelli School of Business

April 10, 2026

Where This Lecture Fits

In Lecture 3 you learned *what* attention and transformers do. Today we prove *why* every design choice works.

Topic	Lecture 3	Lecture 3b (Today)
Self-attention Q, K, V	✓	**Full matrix derivation**
Scaled dot-product $\div\sqrt{d_k}$	✓	**Variance proof**
Multi-head attention	✓	**Reshape trick + cost proof**
Positional encodings	✓	**Rotation property derived**
Layer normalization	✓	**Full derivation**
Residual connections	✓	**Gradient flow proof**
Permutation equivariance	—	**New: formal proof**
Linear attention collapse	—	**New: why FFN is essential**
Pre-Norm vs Post-Norm	—	**New: gradient analysis**

What you need: matrix multiplication, softmax, chain rule (all from Lectures 0–2).

Today's Roadmap

1. **Self-Attention: The Complete Matrix Story**
   - Attention as weighted regression; Q, K, V derivation; 4-token worked example
2. **Why Scale by $\sqrt{d_k}$: A Variance Proof**
3. **Formal Properties of Self-Attention**
   - Permutation equivariance; linear collapse theorem; why FFN is essential
4. **Multi-Head Attention: The Reshape Trick**
5. **Stabilizing Training: LayerNorm and Residuals**
6. **Positional Encodings: Sinusoidal, Learned, ALiBi**
7. **Practice Problems**

Lecture 3 gave us the intuition. Today we open every matrix multiplication and prove every claim.

Attention as Weighted Regression

Self-attention computes **data-dependent weighted averages**. The idea is familiar:

Recall from econometrics:

- OLS fitted value: $\hat{y}_i = \sum_j h_{ij} y_j$ where $H = X(X^\top X)^{-1} X^\top$ (hat matrix)
- Multinomial logit: $P(j \mid i) = \frac{\exp(u_{ij})}{\sum_{j'} \exp(u_{ij'})}$ (McFadden, 1973)

Self-Attention:

$$\mathbf{z}_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{v}_j, \qquad \alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j)}{\sum_{j'=1}^{T} \exp(\mathbf{q}_i^\top \mathbf{k}_{j'})}$$

The attention weights α_{ij} are exactly a multinomial logit over dot-product "utilities" $u_{ij} = \mathbf{q}_i^\top \mathbf{k}_j$: non-negative, sum to 1, data-dependent.

The Q, K, V Projection — Why Three Matrices?

Given an input matrix $X \in \mathbb{R}^{T \times d}$ (one row per token), we define three linear projections:

$$Q = XW^Q, \quad W^Q \in \mathbb{R}^{d \times d_k}, \quad Q \in \mathbb{R}^{T \times d_k} \qquad (\textbf{Queries}: \text{“what am I looking for?”})$$

$$K = XW^K, \quad W^K \in \mathbb{R}^{d \times d_k}, \quad K \in \mathbb{R}^{T \times d_k} \qquad (\textbf{Keys}: \text{“what do I advertise?”})$$

$$V = XW^V, \quad W^V \in \mathbb{R}^{d \times d_v}, \quad V \in \mathbb{R}^{T \times d_v} \qquad (\textbf{Values}: \text{“what do I carry?”})$$

Why not use X directly for all three?

- If $Q = K = X$, then $S = XX^\top$ is **symmetric**: token i attending to j equals j attending to i
- This is restrictive: “AAPL” should attend strongly to “revenue” (entity $\rightarrow$ attribute), but “revenue” might prefer to attend to “beat” (what happened)
- Separate W^Q, W^K projections allow **asymmetric** attention: $\mathbf{q}_i^\top \mathbf{k}_j \neq \mathbf{q}_j^\top \mathbf{k}_i$

Think of a database: the search query, the index key, and the retrieved content play different roles. Making them identical would defeat the purpose.

Worked Example Setup: 4 Tokens, d=8, d_k=4

Sentence: "AAPL revenue beat expectations" $(T = 4$ tokens$)$

Embedding matrix $X \in \mathbb{R}^{4\times 8}$ (toy binary features for tractability):

$$X = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \leftarrow \text{AAPL} \\ \leftarrow \text{revenue} \\ \leftarrow \text{beat} \\ \leftarrow \text{expectations} \end{matrix}$$

Projection matrices $W^Q, W^K, W^V \in \mathbb{R}^{8\times 4}$:

$$\underbrace{W^Q}_{8\times 4} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \underbrace{W^K}_{8\times 4} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \underbrace{W^V}_{8\times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dimension check: $\underbrace{X}_{4\times 8} \cdot \underbrace{W^Q}_{8\times 4} = \underbrace{Q}_{4\times 4}$ — inner dimensions match $(8 = 8)$. ✓

Computing $Q = XW^Q$ Step by Step

Each row of Q is a dot product of a row of X with each column of W^Q:

Row 1 (AAPL): $\mathbf{x}_{\text{AAPL}} = [1, 0, 1, 0, 1, 0, 0, 1]$

$q_{11} = 1{\cdot}1 + 0{\cdot}0 + 1{\cdot}0 + 0{\cdot}1 + 1{\cdot}0 + 0{\cdot}0 + 0{\cdot}0 + 1{\cdot}0 = \mathbf{1}$

$q_{12} = 1{\cdot}0 + 0{\cdot}1 + 1{\cdot}0 + 0{\cdot}0 + 1{\cdot}1 + 0{\cdot}0 + 0{\cdot}0 + 1{\cdot}0 = \mathbf{1}$

$q_{13} = 1{\cdot}0 + 0{\cdot}0 + 1{\cdot}1 + 0{\cdot}0 + 1{\cdot}0 + 0{\cdot}0 + 0{\cdot}1 + 1{\cdot}0 = \mathbf{1}$

$q_{14} = 1{\cdot}1 + 0{\cdot}0 + 1{\cdot}0 + 0{\cdot}0 + 1{\cdot}0 + 0{\cdot}1 + 0{\cdot}0 + 1{\cdot}1 = \mathbf{2}$

$\Rightarrow \mathbf{q}_{\text{AAPL}} = [1, 1, 1, 2]$

Full result (remaining rows computed similarly):

$$Q = XW^Q = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 0 & 2 \\ 1 & 0 & 2 & 1 \end{bmatrix} \begin{matrix} \leftarrow \text{AAPL} \\ \leftarrow \text{revenue} \\ \leftarrow \text{beat} \\ \leftarrow \text{expectations} \end{matrix}, \quad K = XW^K = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 2 & 2 & 0 & 1 \\ 0 & 0 & 2 & 2 \end{bmatrix},$$

$$V = XW^V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{bmatrix}$$

Note: Q and K are different projections of the same X — different "views" of each token. V carries the information to be aggregated.

The Score Matrix $S = QK^\top$

The score matrix computes **all pairwise attention scores** in one matrix multiplication:

$\underbrace{S}_{4\times4} = \underbrace{Q}_{4\times4} \cdot \underbrace{K^\top}_{4\times4}$, where $s_{ij} = \mathbf{q}_i^\top \mathbf{k}_j$ measures "how much should token i attend to token j?"

Entry $(1,2)$: $s_{\text{AAPL,rev}} = \mathbf{q}_{\text{AAPL}}^\top \mathbf{k}_{\text{rev}} = \begin{bmatrix}1\\1\\1\\2\end{bmatrix}^\top \begin{bmatrix}1\\1\\1\\2\end{bmatrix} = 1+1+1+4 = \mathbf{7}$

Entry $(2,1)$: $s_{\text{rev,AAPL}} = \mathbf{q}_{\text{rev}}^\top \mathbf{k}_{\text{AAPL}} = \begin{bmatrix}1\\1\\1\\1\end{bmatrix}^\top \begin{bmatrix}1\\1\\1\\1\end{bmatrix} = 1+1+1+1 = \mathbf{4}$

$$S = QK^\top = \begin{bmatrix} 5 & \mathbf{7} & 6 & 6 \\ \mathbf{4} & 5 & 5 & 4 \\ 5 & 7 & \mathbf{8} & 4 \\ 4 & 5 & 3 & \mathbf{6} \end{bmatrix}$$

Asymmetry: $s_{12} = 7 \neq s_{21} = 4$

AAPL attends much more to "revenue" than "revenue" attends to "AAPL." The separate W^Q, W^K projections create this asymmetry — exactly what we wanted.

Softmax: Row by Row

Each row of S is converted to a probability distribution via softmax. First, scale by $\sqrt{d_k} = \sqrt{4} = 2$:

$$\frac{S}{\sqrt{d_k}} = \frac{1}{2}\begin{bmatrix} 5 & 7 & 6 & 6 \\ 4 & 5 & 5 & 4 \\ 5 & 7 & 8 & 4 \\ 4 & 5 & 3 & 6 \end{bmatrix} = \begin{bmatrix} 2.50 & 3.50 & 3.00 & 3.00 \\ 2.00 & 2.50 & 2.50 & 2.00 \\ 2.50 & 3.50 & 4.00 & 2.00 \\ 2.00 & 2.50 & 1.50 & 3.00 \end{bmatrix}$$

Row 1 (AAPL as query): $\alpha_{1j} = \frac{e^{s_{1j}/2}}{\sum_{j'} e^{s_{1j'}/2}} = \frac{e^{s_{1j}/2}}{e^{2.5}+e^{3.5}+e^{3.0}+e^{3.0}} = \frac{e^{s_{1j}/2}}{12.18+33.12+20.09+20.09}$

$\Rightarrow [\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}] = [0.14,\ \mathbf{0.39},\ 0.24,\ 0.24]$

Full attention matrix $A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$:

$$A = \begin{bmatrix} 0.14 & \mathbf{0.39} & 0.24 & 0.24 \\ 0.19 & \mathbf{0.31} & \mathbf{0.31} & 0.19 \\ 0.11 & 0.31 & \mathbf{0.51} & 0.07 \\ 0.17 & 0.28 & 0.10 & \mathbf{0.46} \end{bmatrix}$$

AAPL → revenue
revenue → revenue, beat
beat → beat (self)
expectations → self

Each row sums to 1 — a probability distribution / portfolio allocation over tokens.

Output $Z = AV$: The Contextual Representation

The output for each token is a weighted average of the value vectors:

$$\underbrace{Z}_{4\times4} = \underbrace{A}_{4\times4} \cdot \underbrace{V}_{4\times4}$$

Row 1 (AAPL): $\mathbf{z}_{\text{AAPL}} = 0.14 \underbrace{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}}_{\mathbf{v}_{\text{AAPL}}} + 0.39 \underbrace{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}}_{\mathbf{v}_{\text{rev}}} + 0.24 \underbrace{\begin{bmatrix}2\\0\\2\\0\end{bmatrix}}_{\mathbf{v}_{\text{beat}}} + 0.24 \underbrace{\begin{bmatrix}0\\2\\0\\2\end{bmatrix}}_{\mathbf{v}_{\text{exp}}} = \begin{bmatrix}1.00\\1.00\\1.00\\1.00\end{bmatrix}$

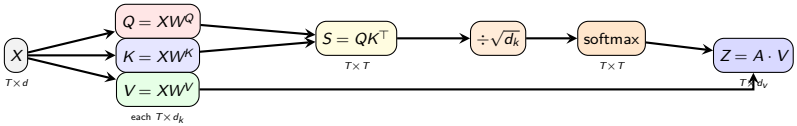
$$Z = AV = \begin{bmatrix} 1.00 & 1.00 & 1.00 & 1.00 \\ 1.12 & 0.88 & 1.12 & 0.88 \\ \mathbf{1.44} & 0.56 & \mathbf{1.44} & 0.56 \\ 0.65 & \mathbf{1.35} & 0.65 & \mathbf{1.35} \end{bmatrix} \begin{array}{l} \leftarrow \text{AAPL: balanced mix} \\ \leftarrow \text{revenue: slight beat-leaning} \\ \leftarrow \text{beat: action-enriched} \\ \leftarrow \text{exp: context-enriched} \end{array}$$

"beat" self-attends at 0.51, pulling in action-related features; "expectations" self-attends at 0.46, pulling in context. The same word in a different sentence produces a different **z** — that is what **contextual** means.

The Complete Self-Attention Formula

Everything in one equation, with dimensions annotated at every step:

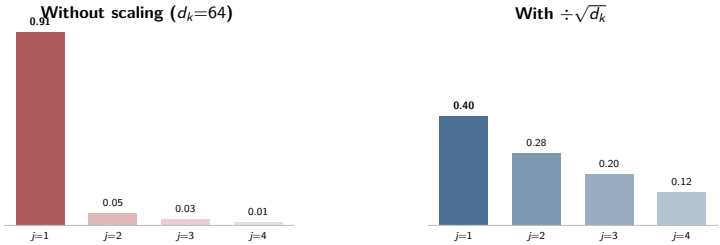
$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{\overbrace{Q}^{T\times d_k} \cdot \overbrace{K^\top}^{d_k\times T}}{\sqrt{d_k}}\right)}_{T\times T} \cdot \underbrace{V}_{T\times d_v} = \underbrace{Z}_{T\times d_v}$$



Computational complexity: $\mathcal{O}(T^2 d_k)$ — quadratic in sequence length T.
Three matrix multiplies and a softmax — that is the entire self-attention layer.

The Problem: Softmax Saturation

Without scaling, large d_k pushes dot products to extreme values, causing softmax to saturate:



Saturated softmax $\rightarrow$ **near-one-hot** $\rightarrow$ **vanishing gradients** $\rightarrow$ **model cannot learn.**
A portfolio that puts 91% weight on one stock cannot learn to diversify — same problem here.

Variance of Dot Products: The Proof

Setup. Let $\mathbf{q}, \mathbf{k} \in \mathbb{R}^{d_k}$ with entries $q_m, k_m \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$, and $\mathbf{q} \perp \mathbf{k}$.
The dot product is $\mathbf{q}^\top \mathbf{k} = \sum_{m=1}^{d_k} q_m k_m$.

Step 1 — Mean: $\mathbb{E}[q_m k_m] = \mathbb{E}[q_m]\,\mathbb{E}[k_m] = 0 \cdot 0 = 0$ (independence)
$\Rightarrow \mathbb{E}[\mathbf{q}^\top \mathbf{k}] = \sum_{m=1}^{d_k} \mathbb{E}[q_m k_m] = 0$

Step 2 — Variance of each term: $\mathrm{Var}(q_m k_m) = \mathbb{E}[q_m^2 k_m^2] - \big(\mathbb{E}[q_m k_m]\big)^2 = \mathbb{E}[q_m^2]\,\mathbb{E}[k_m^2] - 0 = 1 \cdot 1 = 1$

Step 3 — Variance of the sum: Since the d_k terms are independent:

$$\boxed{\mathrm{Var}(\mathbf{q}^\top \mathbf{k}) = \sum_{m=1}^{d_k} \mathrm{Var}(q_m k_m) = d_k}$$

Consequence: $\mathbf{q}^\top \mathbf{k} \sim \mathcal{N}(0,\, d_k)$, so $\mathrm{Std}(\mathbf{q}^\top \mathbf{k}) = \sqrt{d_k}$.
With $d_k = 64$: scores range over $\approx$ [-16, +16]. Softmax of values differing by ~ 32 is essentially one-hot.

The Fix: Divide by $\sqrt{d_k}$

After scaling:

$$\frac{\mathbf{q}^\top \mathbf{k}}{\sqrt{d_k}} \sim \mathcal{N}\left(0,\ \frac{d_k}{d_k}\right) = \mathcal{N}(0,1)$$

Scores now live in $\approx [-2, +2]$ regardless of d_k — softmax stays smooth, gradients flow.

Numerical comparison (our 4-token example, row 3 = "beat" as query):

	AAPL	revenue	beat	expectations
Raw scores s_j	5	7	8	4
Without scaling: softmax(s_j)	0.03	0.26	**0.70**	0.01
With $\div\sqrt{4}$: softmax($s_j/2$)	0.11	0.31	**0.51**	0.07

Without scaling, "beat" puts 70% weight on itself. With scaling, the distribution is smoother (51%), allowing the model to learn from context.

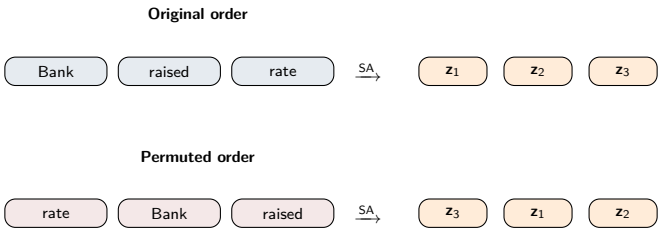
Compare to standardizing returns: $z_i = (r_i - \mu)/\sigma$ makes assets with different volatilities comparable. Dividing by $\sqrt{d_k}$ does the same thing — it normalizes dot products to unit variance so softmax stays well-conditioned.

Self-Attention is Permutation Equivariant

Claim: If we permute the rows of X by a permutation matrix P, the output rows are permuted in the same way:

$$\mathrm{SA}(PX) = P \cdot \mathrm{SA}(X)$$

Without positional encodings, self-attention treats the input as a **set**, not a sequence. Order does not matter.



Without positions, the model cannot distinguish "*Revenue exceeded costs*" from "*Costs exceeded revenue*" — hence the need for **positional encodings**.

Permutation Equivariance: The Proof

Let P be a permutation matrix. We need to show $\mathrm{SA}(PX) = P \cdot \mathrm{SA}(X)$.

Step 1 — Projections: $Q' = (PX)W^Q = P(XW^Q) = PQ$. Similarly $K' = PK$, $V' = PV$.

Step 2 — Score matrix: $S' = Q'(K')^\top = (PQ)(PK)^\top = PQK^\top P^\top = PSP^\top$
This permutes both rows and columns of S: entry (i, j) of S' equals entry $(\sigma^{-1}(i), \sigma^{-1}(j))$ of S.

Step 3 — Softmax: Softmax is applied row-by-row. Permuting rows/columns of S and then applying softmax row-by-row gives: $A' = \mathrm{softmax}(PSP^\top) = P\,\mathrm{softmax}(S)\,P^\top = PAP^\top$
(Permuting columns within a row before softmax just reorders the inputs; the output is the same values in permuted order.)

Step 4 — Output: $Z' = A'V' = (PAP^\top)(PV) = PA\underbrace{P^\top P}_{I}V = PAV = PZ$ ✓

$$\boxed{\mathrm{SA}(PX) = P \cdot \mathrm{SA}(X)}$$

Self-attention is **equivariant to input permutations** — it has no built-in notion of order.

Linear Self-Attention Collapse: Statement

Theorem. If we stack two self-attention layers **without nonlinearity** between them, the result is equivalent to a **single** self-attention layer with different weights.

So without the FFN (with its ReLU), adding layers does **not** increase expressiveness.

Setup. Consider two layers applied sequentially:

- Layer 1: $\mathbf{h}_i^{(1)} = \sum_{j=1}^{T} \alpha_{ij}^{(1)} V^{(1)} \mathbf{x}_j$
- Layer 2: $\mathbf{h}_i^{(2)} = \sum_{j=1}^{T} \alpha_{ij}^{(2)} V^{(2)} \mathbf{h}_j^{(1)}$

The attention weights $\alpha^{(1)}, \alpha^{(2)}$ are computed from their respective Q, K projections.

Stacking layers usually increases capacity. The collapse theorem says **linear** self-attention breaks that — depth buys nothing. The FFN's ReLU is what rescues depth.

Compare: composing two linear regressions gives another linear regression. You need nonlinear terms (polynomials, interactions) to gain expressiveness.

Linear Self-Attention Collapse: The Proof

Substitute Layer 1's output into Layer 2:

$$\mathbf{h}_i^{(2)} = \sum_{j=1}^{T} \alpha_{ij}^{(2)} V^{(2)} \mathbf{h}_j^{(1)} = \sum_{j=1}^{T} \alpha_{ij}^{(2)} V^{(2)} \left(\sum_{k=1}^{T} \alpha_{jk}^{(1)} V^{(1)} \mathbf{x}_k \right)$$

Rearrange (swap summation order):

$$= \sum_{k=1}^{T} \underbrace{\left(\sum_{j=1}^{T} \alpha_{ij}^{(2)} \alpha_{jk}^{(1)} \right)}_{\alpha_{ik}^*} \underbrace{V^{(2)} V^{(1)}}_{V^*} \mathbf{x}_k$$

Define:

- $\alpha_{ik}^* = \sum_j \alpha_{ij}^{(2)} \alpha_{jk}^{(1)}$ (product of attention weight matrices: $A^* = A^{(2)} A^{(1)}$)
- $V^* = V^{(2)} V^{(1)}$ (product of value matrices)

Then: $\mathbf{h}_i^{(2)} = \sum_{k=1}^{T} \alpha_{ik}^* V^* \mathbf{x}_k$

Two stacked linear attention layers = one layer with weights $A^* = A^{(2)} A^{(1)}$, $V^* = V^{(2)} V^{(1)}$

Note: α_{ik}^* still sums to 1 over k (product of two stochastic matrices is stochastic). So A^* is a valid attention matrix. **Depth buys nothing.**

The FFN Breaks the Collapse

The feed-forward network applied between attention layers prevents the collapse:

$\text{FFN}(\mathbf{x}) = \max(0,\ \mathbf{x}\, W_1 + \mathbf{b}_1)\, W_2 + \mathbf{b}_2$

Dimensions: $W_1 \in \mathbb{R}^{d \times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d}$, where typically $d_{ff} = 4d$.
The expansion–compression pattern: $d \to 4d \to d$.

Why the collapse proof fails with FFN:
After Layer 1 + FFN, the hidden state $\tilde{\mathbf{h}}_j^{(1)} = \text{FFN}(\mathbf{h}_j^{(1)})$ is a **nonlinear** function of $\mathbf{h}_j^{(1)}$. The substitution step becomes:
$\mathbf{h}_i^{(2)} = \sum_j \alpha_{ij}^{(2)}\, V^{(2)}\, \text{FFN}\Big(\sum_k \alpha_{jk}^{(1)}\, V^{(1)} \mathbf{x}_k\Big)$
The $\max(0, \cdot)$ inside FFN prevents factoring out $V^{(2)} V^{(1)}$. The two layers are **no longer collapsible**.

Parameter Accounting (BERT-base, per block)

Multi-head attention	$4d^2 = 4 \times 768^2$	$= 2.36$M
FFN	$2d \cdot d_{ff} = 2 \times 768 \times 3072$	$= 4.72$M

The FFN has **twice** the parameters of attention — it stores much of the model's learned knowledge.

Single Head = One Question at a Time

Consider an analyst reading: "*The board approved a $3B buyback despite declining margins.*"

To understand "approved," the analyst simultaneously asks:

1. **Who** approved? → "board" (subject–verb link)
2. **What** was approved? → "buyback" (verb–object link)
3. **How much?** → "$3B" (numerical binding)
4. **Under what conditions?** → "despite declining margins" (adversative)

A single attention head produces *one* attention distribution — it must compromise between all four questions. The result: mediocre attention on everything, strong attention on nothing.

Multi-head attention gives the model h separate heads, each asking a *different* question in parallel. Head 1 links subject–verb, Head 2 links verb–object, etc. No compromise needed.

Analogy: an equity analyst, a credit analyst, and a macro analyst all reading the same earnings call — each extracts different information from identical text.

Why Multiple Heads?

A single attention head computes one set of weights α_{ij} — one "question" per token. But each token might need to attend to different words for different reasons.

Multi-head attention runs h parallel attention operations, each with its own $W_\ell^Q, W_\ell^K, W_\ell^V$:

Head	Learned Pattern	Finance Example
Head 1	Entity–attribute linking	"AAPL" → "revenue"
Head 2	Sentiment modification	"beat" → "expectations"
Head 3	Temporal reference	"Q3" → "earnings"
Head 4	Negation scope	"not" → "sustainable"
Head 5	Numerical binding	"\$2.5B" → "revenue"
Head 6	Causal/adversative	"but" → "guidance"

Each head operates in a **lower-dimensional subspace** ($d_k = d/h$), but the h heads together cover the full d-dimensional space.

Multi-Head Attention: Full Definition

For h heads with $d_k = d/h$ dimensions per head:

Per-head computation ($\ell = 1, \ldots, h$):

$$\text{head}_\ell = \text{Attention}\Big(XW_\ell^Q,\ XW_\ell^K,\ XW_\ell^V\Big) \in \mathbb{R}^{T \times d_k}$$

$$\text{where } W_\ell^Q, W_\ell^K \in \mathbb{R}^{d \times d_k}, \quad W_\ell^V \in \mathbb{R}^{d \times d_k}$$

Concatenate and project:

$$\text{MultiHead}(X) = \underbrace{\text{Concat}(\text{head}_1, \ldots, \text{head}_h)}_{T \times (h \cdot d_k) = T \times d} \cdot \underbrace{W^O}_{d \times d} \in \mathbb{R}^{T \times d}$$

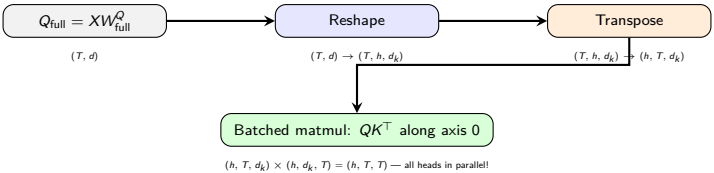
Dimension chain:

$$\underbrace{X}_{T \times d} \xrightarrow{W_\ell} \underbrace{Q_\ell, K_\ell, V_\ell}_{T \times d_k} \xrightarrow{\text{Attn}} \underbrace{\text{head}_\ell}_{T \times d_k} \xrightarrow{\text{Concat}} \underbrace{[\text{head}_1; \ldots; \text{head}_h]}_{T \times d} \xrightarrow{W^O} \underbrace{\text{Output}}_{T \times d}$$

The output has the **same shape** as the input ($T \times d$) — this is essential for stacking transformer blocks.

The Reshape Trick: Intuition

Naïvely, we would compute each head separately: h independent matrix multiplications. The **reshape trick** computes all heads in a **single batched operation**.



After reshaping, the h heads sit along a **batch dimension**. GPUs are built for batched matrix multiplies, so all heads run in parallel with no extra cost.

In code: `Q = Q.reshape(T, h, d_k).transpose(0, 1)` $\to$ shape `(h, T, d_k)`.

Multi-Head Worked Example: h=2 Heads

Using our $Q, K, V \in \mathbb{R}^{4\times 4}$ from the single-head example, now split into $h = 2$ heads with $d_k = 2$:

$$Q = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 0 & 2 \\ 1 & 0 & 2 & 1 \end{bmatrix} \xrightarrow{\text{split cols}} Q_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 2 \\ 2 & 1 \end{bmatrix}$$

Similarly: $K_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$, $K_2 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 0 & 1 \\ 2 & 2 \end{bmatrix}$, $V_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 2 & 0 \\ 0 & 2 \end{bmatrix}$, $V_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 2 & 0 \\ 0 & 2 \end{bmatrix}$

Head 1 scores: $S_1 = Q_1 K_1^\top = \begin{bmatrix} 2 & 2 & 4 & 0 \\ 2 & 2 & 4 & 0 \\ 3 & 3 & 6 & 0 \\ 1 & 1 & 2 & 0 \end{bmatrix}$ **Head 2 scores:** $S_2 = Q_2 K_2^\top = \begin{bmatrix} 3 & 5 & 2 & 6 \\ 2 & 3 & 1 & 4 \\ 2 & 4 & 2 & 4 \\ 3 & 4 & 1 & 6 \end{bmatrix}$

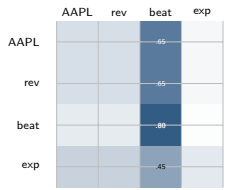
After scaling ($\div\sqrt{2}$) and softmax:

$$A_1 = \begin{bmatrix} .16 & .16 & \mathbf{.65} & .04 \\ .16 & .16 & \mathbf{.65} & .04 \\ .10 & .10 & \mathbf{.80} & .01 \\ .22 & .22 & \mathbf{.45} & .11 \end{bmatrix} \quad A_2 = \begin{bmatrix} .07 & .29 & .04 & \mathbf{.60} \\ .13 & .27 & .06 & \mathbf{.54} \\ .10 & \mathbf{.40} & .10 & \mathbf{.40} \\ .09 & .17 & .02 & \mathbf{.72} \end{bmatrix}$$

Head 1 focuses on "beat" (column 3). **Head 2** focuses on "expectations" (column 4).
Two different attention patterns from the same input.

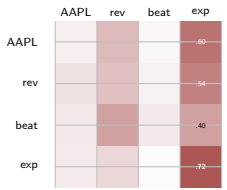
Multi-Head: Attention Heatmaps

Head 1 — "What happened?"



Head 1 is a **verb-focused** head: every token attends most to "beat."

Head 2 — "What is expected?"



Head 2 is a **context-focused** head: tokens attend to "expectations."

Head 1 asks "*What action occurred?*" (beat). Head 2 asks "*What was the benchmark?*" (expectations). A single head would have to compromise between these two questions.

Multi-Head: Concat and Output Projection

Head outputs:

$$Z_1 = A_1 V_1 = \begin{bmatrix} .16 & .16 & .65 & .04 \\ .16 & .16 & .65 & .04 \\ .10 & .10 & .80 & .01 \\ .22 & .22 & .45 & .11 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1.61 & 0.39 \\ 1.61 & 0.39 \\ 1.79 & 0.21 \\ 1.34 & 0.66 \end{bmatrix} \quad Z_2 = A_2 V_2 = \begin{bmatrix} 0.44 & 1.56 \\ 0.53 & 1.47 \\ 0.70 & 1.30 \\ 0.30 & 1.70 \end{bmatrix}$$

Concatenate (4×2 and $4 \times 2 \to 4 \times 4$):

$$\text{Concat}(Z_1, Z_2) = \begin{bmatrix} 1.61 & 0.39 & 0.44 & 1.56 \\ 1.61 & 0.39 & 0.53 & 1.47 \\ 1.79 & 0.21 & 0.70 & 1.30 \\ 1.34 & 0.66 & 0.30 & 1.70 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

Output projection: $\text{MultiHead} = \text{Concat}(Z_1, Z_2) \cdot W^O$, where $W^O \in \mathbb{R}^{4\times 4}$.
W^O is a **learnable** matrix that combines information from both heads. It allows the model to weight the head outputs adaptively — not just concatenate them.

Dimension check: Input $X \in \mathbb{R}^{4\times 4} \to$ Output $\in \mathbb{R}^{4\times 4}$ — same shape. ✓

Multi-Head is Free: The Cost Proof

Claim: Multi-head attention uses the **same** number of parameters and FLOPs as single-head.

Single-head ($d_k = d$): $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$ gives $3d^2$ params. With W^O: $4d^2$.

Multi-head (h heads, $d_k = d/h$):

- Each $W_\ell^Q, W_\ell^K, W_\ell^V \in \mathbb{R}^{d \times d/h}$: $\quad 3 \times d \times (d/h)$ per head
- h heads: $h \times 3d^2/h = 3d^2$ parameters
- Plus $W^O \in \mathbb{R}^{d \times d}$: d^2 parameters
- **Total:** $3d^2 + d^2 = 4d^2$ — **identical to single-head**

Same parameters. Same compute. More expressiveness.

For BERT-base ($d = 768$, $h = 12$): each head uses $d_k = 64$. Total attention parameters per block: $4 \times 768^2 = 2.36$M. Identical whether computed as 12 small heads or 1 large head.

Why Do We Need Normalization?

After multi-head attention and FFN, hidden representations can drift to extreme magnitudes across layers:

The problem: attention outputs are weighted averages of value vectors, then transformed by FFN. After stacking 12+ blocks, activations accumulate scale — some dimensions grow large, others shrink. This causes:

- Softmax saturation in downstream layers (same problem as unscaled dot products)
- Gradient magnitudes that vary wildly across dimensions
- Slow, unstable training that may diverge

The fix: normalize each token's representation to zero mean, unit variance *before* feeding it to the next sublayer. This is **Layer Normalization** (Ba et al., 2016).

Finance analogy

Before comparing returns across asset classes (equities, bonds, commodities), you standardize each to z-scores. LayerNorm does the same thing: it standardizes each token's feature vector so that different tokens and different layers operate on comparable scales.

Layer Normalization: Full Derivation

For a single token's representation $\mathbf{h} \in \mathbb{R}^d$, LayerNorm computes statistics **across features**:

$\hat{\mu} = \frac{1}{d}\sum_{j=1}^{d} h_j$ (mean across d features)

$\hat{\sigma} = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(h_j - \hat{\mu})^2}$ (standard deviation)

$\text{LN}(\mathbf{h}) = \frac{\mathbf{h}-\hat{\mu}}{\hat{\sigma}}$ (optionally: $\gamma \odot \frac{\mathbf{h}-\hat{\mu}}{\hat{\sigma}} + \beta$ with learnable $\gamma, \beta \in \mathbb{R}^d$)

Worked example: $\mathbf{h} = [2.0, -1.0, 0.5, 1.5]$ $(d = 4)$

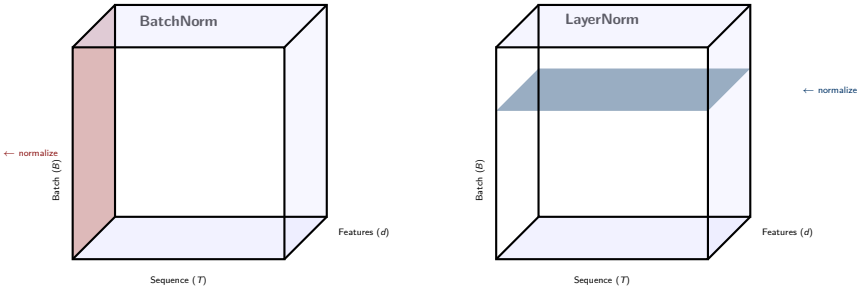
$\hat{\mu} = \frac{2.0+(-1.0)+0.5+1.5}{4} = \frac{3.0}{4} = 0.75$

$\hat{\sigma} = \sqrt{\frac{(2-0.75)^2+(-1-0.75)^2+(0.5-0.75)^2+(1.5-0.75)^2}{4}} = \sqrt{\frac{1.5625+3.0625+0.0625+0.5625}{4}} = \sqrt{1.3125} = 1.15$

$\text{LN}(\mathbf{h}) = \frac{[2.0,-1.0,0.5,1.5]-0.75}{1.15} = [\mathbf{1.09},\ \mathbf{-1.53},\ \mathbf{-0.22},\ \mathbf{0.65}]$

After normalization: mean $= 0$, std $= 1$. The representation is well-conditioned for the next layer.

LayerNorm vs BatchNorm: Why LayerNorm for Transformers



- **BatchNorm**: normalizes each feature across the batch. Depends on batch statistics — unstable with variable sequence lengths and small batches.
- **LayerNorm**: normalizes each token independently across its d features. No dependence on batch or sequence length — works for any T and B.

In econometric terms: LayerNorm $\approx$ **cross-sectional** standardization (per observation). BatchNorm $\approx$ **time-series** standardization (per feature).

Residual Connections: The Gradient Highway

A residual connection adds the input directly to the sublayer output:
$f_{\text{residual}}(\mathbf{x}) = \mathbf{x} + \text{sublayer}(\mathbf{x})$

Gradient analysis:
$\frac{\partial f_{\text{residual}}}{\partial \mathbf{x}} = I + \frac{\partial\, \text{sublayer}(\mathbf{x})}{\partial \mathbf{x}}$

Without residuals (stacking L layers):
$\frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{x}} = \prod_{\ell=1}^{L} \frac{\partial f_\ell}{\partial \mathbf{h}^{(\ell-1)}}$
If each $\left\| \frac{\partial f_\ell}{\partial \mathbf{h}} \right\| < 1$, the gradient **vanishes** exponentially: $\|\text{gradient}\| \approx c^L \to 0$.

With residuals:
$\frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{x}} = \prod_{\ell=1}^{L} \left(I + \frac{\partial f_\ell}{\partial \mathbf{h}^{(\ell-1)}} \right)$
Expanding: this is a sum of 2^L terms, and **one of them is always I** (the identity). The gradient always has a "highway" path of magnitude ≥ 1.

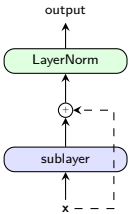
Residual connections guarantee: gradient magnitude ≥ 1, regardless of depth.

Transformers can stack 96+ layers (GPT-3) without gradient vanishing because of this.

Pre-Norm vs Post-Norm

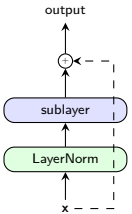
Two architectural variants for combining LayerNorm with residual connections:

Post-Norm (Vaswani et al., 2017):



$\text{output} = \text{LN}(\mathbf{x} + f(\mathbf{x}))$

Pre-Norm (modern standard):



$\text{output} = \mathbf{x} + f(\text{LN}(\mathbf{x}))$

Why Pre-Norm is preferred (Xiong et al., 2020):

In Pre-Norm, the residual path from input to output passes through **no normalization**. The gradient highway is **unobstructed**: $\frac{\partial}{\partial \mathbf{x}} = \boldsymbol{I} + \frac{\partial f(\text{LN}(\mathbf{x}))}{\partial \mathbf{x}}$.

In Post-Norm, the gradient must pass through LayerNorm, which can distort gradient magnitudes at initialization, causing training instability.

Most modern transformers (GPT-3, LLaMA, etc.) use **Pre-Norm**.

The Complete Transformer Block: Equations

One Pre-Norm transformer block (input $\mathbf{X} \in \mathbb{R}^{T \times d}$):

Sub-layer 1: $\mathbf{H}' = \mathbf{X} + \text{MultiHead}(\text{LN}(\mathbf{X})) \in \mathbb{R}^{T \times d}$

Sub-layer 2: $\mathbf{H}'' = \mathbf{H}' + \text{FFN}(\text{LN}(\mathbf{H}')) \in \mathbb{R}^{T \times d}$, where $\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$

Dimension trace:

$$\underbrace{\mathbf{X}}_{T\times d} \xrightarrow{\text{LN}} T\times d \xrightarrow{\text{MHA}} T\times d \xrightarrow{+\mathbf{X}} \underbrace{\mathbf{H}'}_{T\times d} \xrightarrow{\text{LN}} T\times d \xrightarrow{W_1} T\times d_{ff} \xrightarrow{\text{ReLU}} T\times d_{ff} \xrightarrow{W_2} T\times d \xrightarrow{+\mathbf{H}'} \underbrace{\mathbf{H}''}_{T\times d}$$

Parameters per block (BERT-base: $d = 768$, $h = 12$, $d_{ff} = 3072$):

Component	**Parameters**	**Count**
Multi-head attention (W^Q, W^K, W^V, W^O)	$4d^2$	2.36M
FFN ($W_1, \mathbf{b}_1, W_2, \mathbf{b}_2$)	$2d \cdot d_{ff} + d_{ff} + d$	4.72M
LayerNorm ($\times 2$)	$4d$	3.1K
Total per block		**7.09M**
$\times 12$ blocks		**85M**

Sinusoidal Positional Encodings: Mathematics

Since self-attention is permutation-equivariant, we must inject position information. The original Transformer uses sinusoidal functions:

$$\text{PE}(\text{pos}, 2k) = \sin\Big(\frac{\text{pos}}{10000^{2k/d}}\Big), \qquad \text{PE}(\text{pos}, 2k{+}1) = \cos\Big(\frac{\text{pos}}{10000^{2k/d}}\Big)$$

Each dimension pair $(2k, 2k{+}1)$ oscillates at a different frequency: $\omega_k = 1/10000^{2k/d}$.

Worked example ($d = 8$, positions 0–3):

pos	dim 0	dim 1	dim 2	dim 3	dim 4	dim 5	dim 6	dim 7
0	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000
1	**0.841**	0.540	0.100	0.995	0.010	1.000	0.001	1.000
2	**0.909**	-0.416	0.199	0.980	0.020	1.000	0.002	1.000
3	0.141	-0.990	0.296	0.955	0.030	1.000	0.003	1.000

Pattern: Low-index dimensions (0–1) oscillate rapidly, high-index dimensions (6–7) barely change. This creates a multi-scale "clock" encoding position at different resolutions.

Sinusoidal PE: The Rotation Property

Key property: $\text{PE}(\text{pos}+k)$ can be expressed as a **linear transformation** of $\text{PE}(\text{pos})$.

For each dimension pair $(2i, 2i{+}1)$, the shift by k positions is a **rotation matrix**:

$$\begin{bmatrix} \text{PE}(\text{pos}{+}k, 2i) \\ \text{PE}(\text{pos}{+}k, 2i{+}1) \end{bmatrix} = \begin{bmatrix} \cos(k\omega_i) & \sin(k\omega_i) \\ -\sin(k\omega_i) & \cos(k\omega_i) \end{bmatrix} \begin{bmatrix} \text{PE}(\text{pos}, 2i) \\ \text{PE}(\text{pos}, 2i{+}1) \end{bmatrix}$$

Proof: Apply the angle addition formulas:

$$\sin((\text{pos}{+}k)\omega) = \sin(\text{pos}\,\omega)\cos(k\omega) + \cos(\text{pos}\,\omega)\sin(k\omega)$$
$$\cos((\text{pos}{+}k)\omega) = \cos(\text{pos}\,\omega)\cos(k\omega) - \sin(\text{pos}\,\omega)\sin(k\omega)$$

Why this matters:

- The relative position offset k is encoded as a **linear** operation on the position vectors
- The model can learn to detect relative positions using simple dot products: $\text{PE}(\text{pos})^\top \text{PE}(\text{pos}+k)$ depends only on k, not on the absolute positions
- No additional parameters needed — the encoding is deterministic

Application: $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \text{PE}(i)$ adds position information to the token embedding before the first attention layer. The model learns to use the rotation structure through its W^Q, W^K projections.

Learned PE, RoPE, and ALiBi: Modern Alternatives

Learned Positional Encodings (BERT):

- $P \in \mathbb{R}^{N_{\max} \times d}$ is a learnable parameter matrix
- $\tilde{\mathbf{x}}_i = \mathbf{x}_i + P_i$
- Pros: can learn arbitrary position features. Cons: cannot extrapolate beyond $N_{\max}$

ALiBi — Attention with Linear Biases (Press et al., 2022):
Instead of modifying inputs, add a position-dependent **bias** directly to attention scores:

$$\alpha_i = \text{softmax}\Big(\mathbf{k}_{1:T}^\top \mathbf{q}_i + m \cdot [-i,\ -(i-1),\ \ldots,\ 0,\ -1,\ \ldots,\ -(T-i)]\Big)$$

where m is a head-specific slope. Nearby tokens get higher scores; distant tokens are penalized linearly.

ALiBi implements a **recency bias** — like an exponential moving average: recent tokens get more weight, distant tokens less.

Method	Where applied	Relative pos?	Extrapolate?	Used by
Sinusoidal	Input	Via rotation	Limited	Original Transformer
Learned	Input	No	No	BERT, GPT-2
RoPE	Q, K matrices	Yes	Good	LLaMA, GPT-NeoX
ALiBi	Attention scores	Yes	Excellent	BLOOM, MPT

Causal Masking for Autoregressive Models

In language modeling, a token must **not** attend to future tokens (it would be "cheating"):

$$\alpha_{ij,\text{masked}} = \begin{cases} \alpha_{ij} & \text{if } j \leq i \\ 0 & \text{if } j > i \end{cases}$$

Implementation: Add a mask matrix M before softmax:

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right), \quad M_{ij} = \begin{cases} 0 & j \leq i \\ -\infty & j > i \end{cases}$$

Example ($T = 4$, sentence: "AAPL revenue beat expectations"):

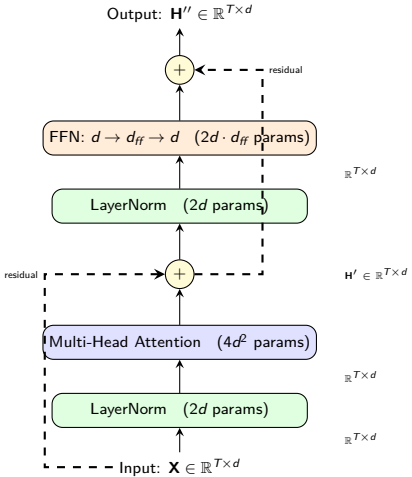
$$\frac{QK^\top}{\sqrt{d_k}} + M = \begin{bmatrix} 2.5 & -\infty & -\infty & -\infty \\ 2.0 & 2.5 & -\infty & -\infty \\ 2.5 & 3.5 & 4.0 & -\infty \\ 2.0 & 2.5 & 1.5 & 3.0 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 1.00 & 0 & 0 & 0 \\ 0.38 & 0.62 & 0 & 0 \\ 0.12 & 0.33 & 0.55 & 0 \\ 0.17 & 0.28 & 0.10 & 0.46 \end{bmatrix}$$

Row 1: "AAPL" can only see itself (weight 1.0).

Row 3: "beat" sees AAPL, revenue, and itself, but **not** "expectations."

This prevents **look-ahead bias** — same idea as in backtesting: a model predicting tomorrow's return cannot use tomorrow's data.

Complete Annotated Transformer Block



BERT-base: 12 of these blocks stacked. Total: $12 \times 7.09\text{M}$ + embeddings $\approx$ **110M parameters**.

Practice Problem 1: Compute Attention by Hand

Setup: 3 tokens ("margin", "pressure", "rising"), $d = 4$, $d_k = 2$.

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

Tasks:

1. Compute $Q = XW^Q$, $K = XW^K$, $V = XW^V$.
2. Compute the score matrix $S = QK^\top \in \mathbb{R}^{3\times 3}$.
3. Scale by $\sqrt{d_k} = \sqrt{2} \approx 1.41$ and apply softmax row by row.
4. Compute the output $Z = AV$.
5. Verify: does each row of A sum to 1? Is S symmetric?

You have 5 minutes. Hint: all entries are small integers before scaling.

Practice Problem 2: Variance and Scaling

1. If $d_k = 256$ and $q_m, k_m \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$:
   (a) What is $\text{Std}(\mathbf{q}^\top \mathbf{k})$?
   (b) A "typical" score is within ± 2 standard deviations. What range does this give?
   (c) Compute softmax$([30, -5, 2, -10])$. What does this look like?

2. After dividing by $\sqrt{d_k} = 16$:
   (a) What is the new standard deviation?
   (b) Compute softmax$([30/16,\ -5/16,\ 2/16,\ -10/16])$. Compare to above.

3. A colleague proposes dividing by d_k instead of $\sqrt{d_k}$.
   (a) What would $\text{Var}(\mathbf{q}^\top \mathbf{k}/d_k)$ be?
   (b) Would the attention distribution be **too uniform**? Why or why not?

Practice Problem 3: Understanding Attention Structure

Conceptual questions — no computation needed:

1. The score matrix $S = QK^\top$ is $T \times T$. Why is S generally **not symmetric** ($s_{ij} \neq s_{ji}$)? Give a finance example where asymmetric attention is essential.
2. After applying softmax, each row of A sums to 1. What does this mean for the output $\mathbf{z}_i = \sum_j \alpha_{ij}\mathbf{v}_j$? Why is this analogous to a portfolio allocation?
3. In a 2-head attention model, Head 1 attends mostly to "revenue" while Head 2 attends mostly to "guidance." If we used a single head instead, what would the attention pattern look like? Why is this a problem?
4. The linear collapse theorem says stacking two attention layers without FFN equals one layer. Why does adding ReLU in the FFN prevent collapse? Connect this to why linear regression cannot capture interaction effects.
5. Pre-Norm places LayerNorm **before** the sublayer; Post-Norm places it **after**. Draw the gradient path for each. Which provides an unobstructed highway to early layers?

Solution: Practice Problem 1

$Q = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 3 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}, \quad V = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$

$S = QK^\top = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 3 & 6 \\ 4 & 2 & 4 \\ 7 & 3 & 6 \end{bmatrix}, \quad S$ is **not symmetric** ($s_{01} = 3 \neq s_{10} = 4$). ✓

$S/\sqrt{2} = \begin{bmatrix} 3.54 & 2.12 & 4.24 \\ 2.83 & 1.41 & 2.83 \\ 4.95 & 2.12 & 4.24 \end{bmatrix}$

$A = \text{softmax}\left(S/\sqrt{2}\right) = \begin{bmatrix} 0.31 & 0.07 & 0.62 & (\text{Fed} \to \text{rates}) \\ 0.45 & 0.11 & 0.45 & (\text{cut} \to \text{Fed, rates}) \\ 0.64 & 0.04 & 0.32 & (\text{rates} \to \text{Fed}) \end{bmatrix}$. Each row sums to ≈ 1. ✓

$Z = AV = \begin{bmatrix} 0.31 & 0.07 & 0.62 \\ 0.45 & 0.11 & 0.45 \\ 0.64 & 0.04 & 0.32 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1.31 & 1.62 \\ 1.45 & 1.45 \\ 1.64 & 1.32 \end{bmatrix}$

"margin" attends most to "rising" (0.62). "rising" attends most to "margin" (0.64) — it wants to know *what* is rising. Note the asymmetry: "margin" → "rising" weight (0.62) ≠ "rising" → "margin" (0.64).

Solution: Practice Problems 2 and 3

Problem 2: (1a) Std $= \sqrt{256} = 16$. (1b) Typical range: ± 32. (1c) softmax$([30, -5, 2, -10]) = [1.00, 0.00, 0.00, 0.00]$ (near one-hot).
(2a) Std $= 1$. (2b) softmax$([1.875, -0.3125, 0.125, -0.625]) = [0.73, 0.08, 0.13, 0.06]$ — much smoother.
(3a) Var$(\mathbf{q}^\top \mathbf{k}/d_k) = d_k/d_k^2 = 1/d_k = 1/256 \approx 0.004$. The distribution is **too concentrated** around 0 — all scores nearly equal — making softmax nearly uniform. The model **cannot differentiate** between tokens.

Problem 3:

1. S is not symmetric because $Q \neq K$: $s_{ij} = \mathbf{q}_i^\top \mathbf{k}_j$ uses different projections for the query and key. Example: "revenue" queries for its modifiers ("strong"), while "strong" queries for what it modifies ("revenue"). The information needs differ.
2. $\mathbf{z}_i$ is a convex combination of value vectors — a portfolio with weights α_{ij} summing to 1. Each token's output is a "diversified portfolio" of information from all other tokens.
3. A single head would produce one compromise distribution, e.g., 30% on revenue, 30% on guidance, 40% elsewhere. Neither signal is strong. Two heads let each focus sharply on its target.
4. ReLU introduces element-wise nonlinearity: $\max(0, x)$ creates piecewise-linear regions that cannot be collapsed by matrix multiplication, just as $x_1 \cdot x_2$ interaction terms cannot be captured by a linear model $\beta_1 x_1 + \beta_2 x_2$.
5. Pre-Norm: gradient path $\mathbf{x} \to (+) \to$ output passes through no normalization. Post-Norm: gradient must pass through LN, which can distort magnitudes. Pre-Norm provides the unobstructed highway.

Key Takeaways

1. **Self-attention** = data-dependent weighted average via Q, K, V projections. Attention weights are multinomial logit probabilities. Three matrix multiplies + softmax.
2. **Scaling by** $\sqrt{d_k}$ normalizes dot product variance from d_k to 1, preventing softmax saturation.
3. **Permutation equivariance:** $\mathrm{SA}(PX) = P \cdot \mathrm{SA}(X)$. Self-attention is order-blind — positional encodings are essential.
4. **Linear collapse:** Without FFN nonlinearity, stacking attention layers gains nothing. ReLU in the FFN is what makes depth meaningful.
5. **Multi-head attention** = parallel specialized views at **zero extra cost** via the reshape trick. Each head learns different linguistic patterns.
6. **LayerNorm + residuals** stabilize deep training. Pre-Norm ensures unobstructed gradient flow. Residuals provide a gradient highway ($\|\partial/\partial \mathbf{x}\| \geq 1$).
7. **Sinusoidal PE** encodes position via rotation matrices; relative position is a linear operation. Modern alternatives (RoPE, ALiBi) offer better length extrapolation.

Questions?

Thank you!