Explainable Transformers via Graph-Based Operator Notation
A Beginner-Oriented, Concept-Only Guide

Anonymous

Abstract

We present a compact, pedagogy-first description of Transformers using a graph-based operator notation. Nodes denote operators, edges denote tensors, and double arrows mark the second operand of matrix multiplications. This article focuses on concepts (no code), pairing each formula with shape reasoning so a newcomer can translate diagrams into implementations after reading.[1] This article is intentionally written for readers without prior neural-network background: it builds intuition first with operator–tensor diagrams, so that by the end one can follow the end-to-end Transformer flow, and later dive into the mathematical and implementation details with confidence.

Keywords Transformer; attention; backpropagation; LayerNorm; softmax; diagram notation; pedagogy.

1 Introduction

Transformers have become the dominant sequence model in modern ML. However, many introductions assume prior exposure to neural networks and autodiff. We provide a beginner-friendly, concept-only path: (i) a concise section on graph notation and visual conventions, (ii) operator primitives with forward/backward summaries, (iii) multi-head attention (MHA) forward/backward, and (iv) assembly of a Transformer block (Input Embedding $\rightarrow$ MHA $\rightarrow$ MLP $\rightarrow$ Output Projection).

Learning Path and Promise We start from a visual, shape-first notation (Section 2), fix a small set of operator primitives with forward/backward summaries (Section 3), and then assemble the Transformer block from Input Encoding $\rightarrow$ MHA $\rightarrow$ MLP $\rightarrow$ Output Projection (Sections 4–5). By design, a reader who "doesn't yet know where to start" can still finish with a working mental model of the full Transformer flow, and later return to fill in mathematics, cost models, and implementation details without losing the big picture.

1.1 Contributions

(1) A consistent diagrammatic convention that separates operators from data. (2) Shape-first backprop summaries, including broadcasting and reduce axes. (3) A compact Transformer walkthrough suitable for first-time readers.

2 Graph Notation and Visual Conventions

How to Read This Section If you feel lost about where to begin, read this section as a map: each icon in the figures has a one-to-one meaning here (node type, edge role, and shape tags). Once you can read a single forward figure and a single backward figure fluently, the entire Transformer flow becomes a sequence of these same primitives reused in different places.

We use operator–tensor graphs where nodes are operators and edges carry tensors. The notation is identical in text and figures to avoid ambiguity and testing mismatches.

[1] A practical, code-first companion is intentionally excluded in this version.

2.1 Node Types (exactly as in figures)

- **MatMul** ⊙ : green circular node with a centered • (matrix multiplication).
- **Elementwise Add / Bias Add** ⊕ : red dashed circle with a centered + (broadcasting allowed).
- **Nonlinear / Normalization / Utility** LN, GL, DO, T, S, SM, CE: yellow rectangle with the operator label (LayerNorm, GELU, Dropout, Transpose, Softmax, Score+Mask, Cross-Entropy).
- **Reduction** Σ : small red circle with $\sum$ (axis reductions such as $\sum_{B,S}$ for bias gradients).

2.2 Edges and the Double-Arrow Convention

We draw one diagram per direction. In a forward diagram, all arrows carry forward values; in a backward diagram, all arrows carry gradients. We therefore use one unified arrow style:

- **Value flow:** single solid arrow $\rightarrow$ for all tensors in the current diagram.
- **Second MatMul operand:** double arrow $\Rightarrow$ into the MatMul node to mark the right-hand matrix (e.g., weights, possibly transposed).
- **Reductions:** a Σ node denotes axis reductions (e.g., $\sum_{B,S}$).

The double arrow removes ambiguity about which factor is on the right in $Z = XW$, and clarifies where transposes appear in backprop.

2.3 Tensor Names, Shapes, and Broadcasting

Edges are labeled with names and shapes, e.g., $[B, S, D]$, $[B, S, D_{\text{ff}}]$. Broadcast notation $\text{BC}_{B,S}(\cdot)$ indicates expansion (e.g., $\text{BC}_{B,S}(\tilde{\mathbf{b}}_{\text{up}}) : [D_{\text{ff}}] \rightarrow [B, S, D_{\text{ff}}]$).

2.4 MatMul Orientation and Backprop

For $Z = XW$ we mark W with the double arrow. Backprop uses

$$\frac{\partial L}{\partial X} = G\,W^{\top}, \qquad \frac{\partial L}{\partial W} = X^{\top}G,$$

with $G = \partial L/\partial Z$. In diagrams, needed transposes are shown with T before the gradient-accumulation MatMul.

2.5 Bias Add and Reductions

For $Y = A + B$ with broadcasting, the gradient w.r.t. B reduces across broadcast axes:

$$\frac{\partial L}{\partial B} = \sum_{\text{broadcast axes}} \frac{\partial L}{\partial Y}.$$

We render this with a Σ node labeled $\sum_{B,S}$ at bias-add sites.

2.6 (Optional) Cost Badges

You may attach small badges: `FLOPs=` inside MatMul nodes (e.g., $2mkn$) and `[shape | bytes]` on wide edges, to immediately surface compute/memory hot spots.

3 Operator Primitives

3.1 General Backprop Rule for $z = f(x, y)$

Let $g = \partial L/\partial z$. Then

$$\frac{\partial L}{\partial x} = \texttt{reduce\_like}\big(g \odot \tfrac{\partial f}{\partial x},\ x\big), \quad \frac{\partial L}{\partial y} = \texttt{reduce\_like}\big(g \odot \tfrac{\partial f}{\partial y},\ y\big), \tag{1}$$

where `reduce_like` sums over broadcast axes to match the input shape. For MatMul $Z = AB$ with upstream G, $\partial L/\partial A = GB^\top$ and $\partial L/\partial B = A^\top G$.

3.2 Elementwise Add / Bias Add

$Y = A + B$. Gradients reduce along broadcast axes.

3.3 Linear / Projection

$Y = XW + b$. $\partial L/\partial X = (\partial L/\partial Y)W^\top$, $\partial L/\partial W = X^\top(\partial L/\partial Y)$, $\partial L/\partial b = \sum_{B,S}(\partial L/\partial Y)$.

3.4 Softmax (Stable)

With $Y = \mathrm{softmax}(Z)$ along the last axis, $\partial L/\partial Z = (G - \langle G, Y\rangle) \odot Y$, where $G = \partial L/\partial Y$ and the inner product is along the softmax axis.

3.5 LayerNorm (LN)

Summarize forward $(\mu, \sigma, \hat{X})$ and use the standard derivative form with cached statistics.

4 Multi-Head Attention (MHA)

4.1 Forward

Given $Q, K, V \in \mathbb{R}^{[B,H,S,D_h]}$, scores $A = QK^\top/\sqrt{D_h}$, probabilities $P = \mathrm{softmax}(A)$ (with masking), outputs $O = PV$. Heads are merged (concat) and projected. Shapes: $A, P \in \mathbb{R}^{[B,H,S,S]}$, $O \in \mathbb{R}^{[B,S,D]}$.

4.2 Backward (Summary)

$dO \to dV = P^\top dO$, $dO \to dP = dO\, V^\top$, $dP \to dA = (dP - \mathrm{sum}(dP \odot P)) \odot P$, masking zeros gradients on masked entries, and $dA \to dQ = dA\, K/\sqrt{D_h}$, $dA \to dK = dA^\top Q/\sqrt{D_h}$.

5 Transformer Block (Concept-Only)

Pipeline: Input Embedding → MHA → MLP (FFN) → Output Projection, with residual connections and LayerNorm.
Input Embedding: token table $E \in \mathbb{R}^{V\times D}$ mapping $[B, S]$ to $[B, S, D]$; positional signal (absolute/learned or RoPE).
MLP: Linear–GELU–Linear (or SwiGLU), expansion factor α.
Output Projection: logits via $W_{lm} \in \mathbb{R}^{D\times V}$; optionally tie with E.

Figure 1: Forward pass of multi-head attention using the proposed legend.

6 Figures

7 Discussion and Limitations

Scope is conceptual; we omit code, datasets, and training details. Numerical stability notes (softmax, LN) are summarized rather than proven.

8 Conclusion

We deliberately prioritized a diagram-first, beginner-friendly path so that readers without a neural-network background can still form a correct end-to-end mental model of Transformers. From here, the same operator–tensor vocabulary extends naturally to stability tricks (e.g., log-sum-exp, pre-LN), cost models (FLOPs/bytes per edge), training regimes, and implementation details (fused kernels, KV caching), without losing the overall flow.