Attention as Bilinear Form

A Physicist's Guide to Transformer Attention

Tensor Calculus, Statistical Mechanics, and Differential Geometry

Baalateja Kataru
baalateja.k@gmail.com

January 2026

Abstract

If you've ever wondered what's really going on inside a transformer, this tutorial is for you. We're going to look at the attention mechanism through a physicist's lens—using the language of tensor calculus, bilinear forms, and statistical mechanics. Don't worry if that sounds intimidating; we'll build up the concepts step by step.

The punchline? That innocent-looking formula $\text{Attention}(Q, K, V) = \text{softmax}\left(QK^T/\sqrt{d_k}\right)V$ hides beautiful mathematical structure: the score computation is a bilinear form with a hidden metric tensor, the softmax is actually the Gibbs distribution from thermodynamics, and the whole thing implements an associative memory network with exponential storage capacity!

A companion Python library `attn-tensors` implements everything we discuss, with 400+ tests verifying our gradient derivations against JAX autodiff. Code at: https://github.com/bkataru-workshop/attn-as-bilinear-form.

Contents

1 Introduction

The attention mechanism, introduced by Bahdanau et al. [1] and refined in the Transformer architecture by Vaswani et al. [2], has become the foundation of modern deep learning. While typically presented in matrix notation optimized for implementation, the underlying mathematical structure reveals deep connections to classical physics and differential geometry.

This tutorial recasts the attention mechanism in the language of tensor calculus, making explicit the geometric and statistical mechanical structure hidden in the standard formulation. Our goals are:

1. **Tensor Calculus**: Express attention using index notation with proper contravariant/ covariant indices and Einstein summation convention
2. **Bilinear Forms**: Show that attention scores arise from a bilinear form with an implicit metric tensor
3. **Statistical Mechanics**: Interpret softmax as the Gibbs/Boltzmann distribution from thermodynamics
4. **Differential Geometry**: Understand the feature space as a Riemannian manifold
5. **Gradients**: Derive backpropagation formulas in index notation and verify against autodiff
6. **Efficiency**: Understand Flash Attention and linear attention through the geometric lens

1.1 Who Is This For?

This tutorial is aimed at:

- **ML practitioners** who want a deeper understanding of what attention is really doing
- **Physics students** curious about how their mathematical toolkit applies to deep learning
- **Researchers** looking for new perspectives on attention mechanisms

We assume familiarity with basic linear algebra and calculus. Prior exposure to index notation or differential geometry is helpful but not required—we'll introduce everything we need.

1.2 Notation Conventions

Throughout this document, we use the following conventions:

Definition (Index Notation).

- **Superscripts** denote contravariant (column vector) indices: v^a, Q^{ia}
- **Subscripts** denote covariant (row vector/dual) indices: g_{ab}, u_a
- **Einstein summation**: Repeated indices (one up, one down) are summed: $v^a u_a = \sum_a v^a u_a$
- **Kronecker delta**: $\delta^a_b = \begin{cases} 1 \text{ if } a{=}b \\ 0 \text{ otherwise} \end{cases}$

Intuition. Think of superscripts as "column vectors" and subscripts as "row vectors." When you contract (sum over) a matching pair, you're doing a dot product. The Einstein convention just saves us from writing summation signs everywhere.

The key index labels we use:

Index	**Meaning**
i	Query sequence position (n_q positions)

j, k	Key/value sequence position (n_k positions)
a, b, c	Feature/embedding dimensions (d_k or d_v)
h	Attention head index (H heads)
μ, ν	Pattern index in Hopfield networks

2 Part I: Foundations

2.1 Vectors and Dual Vectors

In physics, we distinguish between vectors and their duals (covectors). A vector v^a lives in a vector space V, while a covector u_a lives in the dual space V^*. The natural pairing between them is:

$$\langle u, v\rangle = u_a v^a \tag{1}$$

This is basis-independent. In a coordinate basis, this becomes the familiar dot product.

Intuition. In machine learning terms: a vector v^a is a column vector, and a covector u_a is a row vector. Their pairing $u_a v^a$ is just matrix multiplication of a row times a column, giving a scalar.

Definition (Metric Tensor). A **metric tensor** g_{ab} is a symmetric, positive-definite $(0, 2)$ -tensor that defines an inner product on the vector space:

$$\langle u, v\rangle_g = u^a g_{ab} v^b \tag{2}$$

The metric allows us to:
1. **Lower indices**: $v_a = g_{ab} v^b$ (convert vector to covector)
2. **Raise indices**: $v^a = g^{ab} v_b$ (convert covector to vector)

where g^{ab} is the inverse metric satisfying $g^{ac} g_{cb} = \delta^a_b$.

Remark. The metric tells us how to measure distances and angles in our space. Different metrics lead to different notions of “similarity”—this will be crucial for understanding attention.

2.2 Bilinear Forms

Definition (Bilinear Form). A **bilinear form** is a map $B : V \times W \to \mathbb{R}$ that is linear in both arguments:

$$B(\alpha u + \beta v, w) = \alpha B(u, w) + \beta B(v, w) \tag{3}$$

$$B(u, \alpha v + \beta w) = \alpha B(u, v) + \beta B(u, w) \tag{4}$$

In index notation with a matrix M_{ab}:

$$B(u, v) = u^a M_{ab} v^b \tag{5}$$

The connection to attention: the attention score between a query q and key k is precisely a bilinear form:

$$S = q^a g_{ab} k^b \tag{6}$$

where the metric g_{ab} encodes how we measure similarity in feature space.

Intuition. You can think of the metric g_{ab} as answering the question: "How should I weight different features when computing similarity?" The standard dot product weights all features equally, but we could do something more sophisticated.

2.3 Standard Metrics

Example (Euclidean Metric).

$$g_{ab} = \delta_{ab} \tag{7}$$

(identity matrix)

This gives the standard dot product: $\langle u, v\rangle = u^a \delta_{ab} v^b = u^a v_a$

Example (Scaled Euclidean Metric).

$$g_{ab} = \frac{1}{\sqrt{d_k}} \delta_{ab} \tag{8}$$

This is precisely the metric implicit in scaled dot-product attention! The $\frac{1}{\sqrt{d_k}}$ factor prevents dot products from growing too large in high dimensions.

Example (Learned Bilinear Metric).

$$g_{ab} = W_a^c W_{cb} = \left(W^T W\right)_{ab} \tag{9}$$

Parameterizing the metric as $W^T W$ ensures positive semi-definiteness. This generalizes standard attention to learnable similarity functions.

Remark. The scaling $\frac{1}{\sqrt{d_k}}$ has a statistical interpretation: if q^a and k^a are i.i.d. with zero mean and unit variance, then $\operatorname{Var}(q^a k_a) = d_k$. The scaling normalizes the variance to 1, keeping the softmax in a good operating regime. This is the "variance explosion" problem that the $\sqrt{d_k}$ scaling solves.

2.4 Einstein Summation (Einsum)

Now that we've introduced index notation, let's talk about how to implement it in code. The `einsum` function, available in NumPy, JAX, PyTorch, and other libraries, directly translates index notation to efficient tensor operations.

Definition (Einstein Summation Convention). In the **Einstein summation convention**, repeated indices are implicitly summed:

$$C^{ik} = A^{ij} B_j^{\ k} \quad \Leftrightarrow \quad C_{ik} = \sum_j A_{ij} B_{jk} \tag{10}$$

In code: `C = einsum('ij,jk->ik', A, B)`

The einsum string has a simple grammar:

- **Input specification** (left of `->`): Comma-separated indices for each input tensor

- **Output specification** (right of →): Indices in the output
- **Repeated indices** not in output are summed over

> *Intuition.* Think of einsum indices as a way to label the dimensions of tensors. When the same label appears in multiple places, those dimensions are "paired up" for multiplication. When a label is missing from the output, that dimension is summed over.

Example (Basic Einsum Patterns). Common operations in einsum:

Operation	Einsum	Index Notation
Dot product	`'a,a->'`	$s = u^a v_a$
Outer product	`'a,b->ab'`	$M_{ab} = u_a v_b$
Matrix multiply	`'ij,jk->ik'`	$C^{ik} = A^{ij} B_j{}^k$
Transpose	`'ij->ji'`	$B_{ji} = A_{ij}$
Trace	`'ii->'`	$s = A_i^i$
Batch matmul	`'bij,bjk->bik'`	$C^{bik} = A^{bij} B^{b}_{j\ k}$

2.4.1 Einsum for Attention

The attention mechanism maps beautifully to einsum. Here are the key operations:

Attention scores (query-key dot product):

$$S^{ij} = Q^{ia} K^{ja} / \sqrt{d_k} \tag{11}$$

```
S = einsum('ia,ja->ij', Q, K) / jnp.sqrt(d_k)
```

Attention output (weighted sum of values):

$$O^{ib} = A^{ij} V^{jb} \tag{12}$$

```
O = einsum('ij,jb->ib', A, V)
```

With bilinear metric:

$$S^{ij} = Q^{ia} g_{ab} K^{jb} \tag{13}$$

```
S = einsum('ia,ab,jb->ij', Q, g, K)
```

2.4.2 Multi-Head Einsum

Multi-head attention adds a head index h:

```
# Project to per-head queries: X^{hia} = X^{id} W_Q^{hda}
Q_h = einsum('id,hda->hia', X, W_Q)

# Per-head attention scores: S^{hij} = Q^{hia} K^{hja} / sqrt(d_k)
S = einsum('hia,hja->hij', Q_h, K_h) / jnp.sqrt(d_k)

# Per-head outputs: O^{hic} = A^{hij} V^{hjc}
O = einsum('hij,hjc->hic', A, V_h)

# Combine heads: Y^{id} = O^{hic} W_O^{hcd}
Y = einsum('hic,hcd->id', O, W_O)
```

Intuition. Einsum makes the summation indices explicit. When you see `'hia,hja->hij'`, you immediately know that `a` (the feature dimension) is being summed over, while `h`, `i`, `j` are preserved. This is exactly what the index notation tells us: $S^{hij} = \sum_a Q^{hia} K^{hja}$.

Remark. Einsum is not just notation—it's often faster than explicit loops and reshapes because it avoids creating intermediate arrays. The library can optimize the contraction order and fuse operations. For more on einsum, see Sankalp's excellent tutorial "Shape Rotation 101" [3].

3 Part II: The Attention Mechanism

Now let's see how all this machinery applies to attention.

3.1 Attention as Tensor Contraction

The attention mechanism operates on three inputs:

- **Queries** Q^{ia}: What we're looking for (shape: $n_q \times d_k$)
- **Keys** K^{ja}: What we're matching against (shape: $n_k \times d_k$)
- **Values** V^{jb}: What we retrieve (shape: $n_k \times d_v$)

Intuition. Think of it like a library search:

- The **query** is your question ("I want books about physics")
- The **keys** are the book titles/descriptions (what you match against)
- The **values** are the actual book contents (what you get back)

The mechanism proceeds in three steps:

3.1.1 Step 1: Score Computation

Compute pairwise similarity between all queries and keys:

Definition (Attention Scores).

$$S^{ij} = \frac{1}{\sqrt{d_k}} Q^{ia} K^{ja} \tag{14}$$

Or with explicit metric: $S^{ij} = Q^{ia} g_{ab} K^{jb}$

where $g_{ab} = \frac{1}{\sqrt{d_k}} \delta_{ab}$.

The contraction over the feature index a computes a scalar similarity for each query-key pair. This is a **bilinear form** evaluated for all pairs.

In code (JAX with einsum):

```python
# S^{ij} = Q^{ia} K^{ja} / sqrt(d_k)
S = jnp.einsum('ia,ja->ij', Q, K) / jnp.sqrt(d_k)
```

3.1.2 Step 2: Softmax Normalization

Convert scores to a probability distribution over keys:

Definition (Attention Weights).

$$A^{ij} = \frac{\exp(S^{ij})}{\sum_k \exp(S^{ik})} = \frac{\exp(S^{ij})}{Z^i} \tag{15}$$

where $Z^i = \sum_j \exp(S^{ij})$ is the **partition function** for query i.

The softmax is applied row-wise: each query i gets its own probability distribution over keys.

3.1.3 Step 3: Value Aggregation

Compute weighted sum of values:

Definition (Attention Output).

$$O^{ib} = A^{ij} V^{jb} \tag{16}$$

This contracts over the key index j, producing an output for each query.

Intuition. The output for each query is a weighted average of the values, where the weights are determined by how well each key matches the query. High-scoring keys contribute more to the output.

3.2 Full Attention in One Equation

Combining all steps:

Theorem (Scaled Dot-Product Attention).

$$O^{ib} = \frac{\exp(Q^{ia} g_{ac} K^{jc})}{\sum_k \exp(Q^{ia} g_{ac} K^{kc})} V^{jb} \tag{17}$$

Or in matrix notation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{18}$$

Complete implementation:

```python
def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    Full attention: O^{ib} = A^{ij} V^{jb}
    where A^{ij} = softmax_j(Q^{ia} K^{ja} / sqrt(d_k))
    """
    d_k = Q.shape[-1]

    # Step 1: Scores S^{ij} = Q^{ia} K^{ja} / sqrt(d_k)
    scores = jnp.einsum('ia,ja->ij', Q, K) / jnp.sqrt(d_k)

    # Apply mask if provided (for causal attention)
    if mask is not None:
        scores = jnp.where(mask, scores, -1e9)

    # Step 2: Weights A^{ij} = softmax_j(S^{ij})
    weights = jax.nn.softmax(scores, axis=-1)

    # Step 3: Output O^{ib} = A^{ij} V^{jb}
    output = jnp.einsum('ij,jb->ib', weights, V)

    return output
```

4 Part III: Statistical Mechanics

The softmax function is not merely a normalization trick—it is the **Gibbs distribution** from statistical mechanics, revealing deep connections to thermodynamics.

4.1 The Gibbs/Boltzmann Distribution

In statistical mechanics, a system with energy levels E_j at temperature T has probability:

Definition (Gibbs Distribution).

$$P(j) = \frac{\exp\left(-E_j/T\right)}{Z}, \quad \text{where } Z = \sum_j \exp\left(-E_j/T\right) \tag{19}$$

- T: Temperature (controls distribution sharpness)
- Z: Partition function (normalization)
- $\beta = 1/T$: Inverse temperature

For attention, we identify:
- **Scores as negative energies**: $S^{ij} = -E^{ij}$ (higher score = lower energy = preferred)
- **Temperature**: Standard attention uses $T = 1$

Thus:

$$A^{ij} = \frac{\exp\left(S^{ij}/T\right)}{\sum_k \exp\left(S^{ik}/T\right)} \tag{20}$$

Intuition. In physics, systems prefer low-energy states. In attention, we prefer high-scoring keys. The connection: if we define energy as negative score, then high scores become low energy, and softmax gives us the equilibrium distribution!

4.2 Temperature Effects

The temperature parameter controls how "peaked" the attention distribution is:

Theorem (Temperature Limits). As temperature varies:

$$\lim_{T\to 0} A^{ij} = \begin{cases} 1 \text{ if } j = \arg\max_k S^{ik} \\ 0 \text{ otherwise} \end{cases} \quad \text{(hard attention)} \tag{21}$$

$$\lim_{T\to\infty} A^{ij} = \frac{1}{n_k} \quad \text{(uniform attention)} \tag{22}$$

Proof. For the $T \to 0$ limit, consider scores S^{ij} with unique maximum at j^*. Then:

$$A^{ij} = \frac{\exp\left(S^{ij}/T\right)}{\sum_k \exp\left(S^{ik}/T\right)} = \frac{\exp\left(\left(S^{ij} - S^{ij^*}\right)/T\right)}{\sum_k \exp\left(\left(S^{ik} - S^{ij^*}\right)/T\right)} \tag{23}$$

As $T \to 0$, all terms with $S^{ik} < S^{ij^*}$ vanish exponentially, leaving only $j = j^*$.

For the $T \to \infty$ limit, $\exp(S/T) \to 1$ for all S, so all weights become equal. □

The $\frac{1}{\sqrt{d_k}}$ scaling in standard attention can be interpreted as setting an effective temperature $T = \sqrt{d_k}$ when using unscaled scores.

```python
def attention_temperature(Q, K, V, temperature=1.0):
    """Attention with explicit temperature control."""
    d_k = Q.shape[-1]
    scores = jnp.einsum('ia,ja->ij', Q, K) / jnp.sqrt(d_k)
    weights = jax.nn.softmax(scores / temperature, axis=-1)
    return jnp.einsum('ij,jb->ib', weights, V)
```

4.3 Entropy and Information

The **Shannon entropy** of attention weights measures how diffuse or focused the attention is:

Definition (Attention Entropy).

$$H^i = -\sum_j A^{ij} \log A^{ij} \tag{24}$$

- $H = 0$: Delta distribution (all attention on one key)
- $H = \log(n_k)$: Uniform distribution (equal attention on all keys)

The **normalized entropy** $H^i / \log(n_k) \in [0, 1]$ provides a scale-independent measure.

Intuition. Low entropy = focused attention (the model is "confident" about which keys to attend to). High entropy = diffuse attention (the model is spreading its attention broadly). Both can be useful depending on the task!

4.4 Free Energy

The **free energy** combines energy and entropy:

Definition (Free Energy).

$$F^i = -T \log Z^i = -T \log \sum_j \exp(S^{ij}/T) \tag{25}$$

At temperature T, the free energy satisfies:

$$F = \langle E \rangle - T \cdot H \tag{26}$$

where $\langle E \rangle$ is the expected energy and H is the entropy.

Theorem (Variational Principle). The attention weights minimize the free energy:

$$A^* = \arg\min_A [\langle E \rangle - TH(A)] \tag{27}$$

subject to $\sum_j A^{ij} = 1$ and $A^{ij} \geq 0$.

Proof. This is the standard derivation of the Gibbs distribution. The Lagrangian is:

$$\mathcal{L} = -\sum_j A_j S_j + T \sum_j A_j \log A_j + \lambda \left(\sum_j A_j - 1 \right) \tag{28}$$

Setting $\partial \mathcal{L} / \partial A_j = 0$:

$$-S_j + T(1 + \log A_j) + \lambda = 0 \tag{29}$$

$$A_j = \exp\big((S_j - \lambda - T)/T\big) \propto \exp\big(S_j/T\big) \tag{30}$$

Normalizing gives the softmax. □

Minimizing free energy balances:

1. **Low energy**: Attend to high-scoring keys
2. **High entropy**: Spread attention broadly (regularization)

This provides a principled way to understand attention with temperature.

5 Part IV: Differential Geometry

The feature space where queries and keys live can be understood as a Riemannian manifold, with the metric tensor defining geometry.

5.1 Riemannian Metrics

Definition (Riemannian Manifold). A **Riemannian manifold** (M, g) is a smooth manifold M equipped with a metric tensor $g_{ab}(x)$ at each point $x \in M$.

The metric defines:

- **Distances**: $ds^2 = g_{ab} dx^a dx^b$
- **Angles**: $\cos\theta = \frac{u^a g_{ab} v^b}{\|u\|_g \|v\|_g}$
- **Volumes**: $dV = \sqrt{\det g} dx^1 ... dx^n$

In attention, the feature space $\mathbb{R}^{d_k}$ is (implicitly) a Riemannian manifold with metric $g_{ab} = \frac{1}{\sqrt{d_k}} \delta_{ab}$.

5.2 Christoffel Symbols and Covariant Derivatives

For a general metric $g_{ab}(x)$ that varies with position, we need **covariant derivatives** to properly differentiate tensors:

Definition (Christoffel Symbols). The **Christoffel symbols of the second kind** are:

$$\Gamma^c_{ab} = \frac{1}{2} g^{cd} (\partial_a g_{bd} + \partial_b g_{ad} - \partial_d g_{ab}) \tag{31}$$

Definition (Covariant Derivative). For a contravariant vector v^b:

$$\nabla_a v^b = \partial_a v^b + \Gamma^b_{ac} v^c \tag{32}$$

For a covariant vector u_b:

$$\nabla_a u_b = \partial_a u_b - \Gamma^c_{ab} u_c \tag{33}$$

For the standard (constant) attention metric, $\Gamma^c_{ab} = 0$ and covariant derivatives reduce to ordinary derivatives. This is why we can get away without differential geometry in standard attention—but learned metrics would need these tools!

5.3 Natural Gradient Descent

When optimizing over parameter space, the **Fisher information matrix** provides a natural Riemannian metric:

Definition (Fisher Information).

$$F_{ij} = \mathbb{E}\left[\frac{\partial \log p(x|\theta)}{\partial \theta^i} \frac{\partial \log p(x|\theta)}{\partial \theta^j}\right] \tag{34}$$

Natural gradient descent uses this metric:

$$\Delta\theta^i = -\eta (F^{-1})^{ij} \frac{\partial L}{\partial \theta^j} \tag{35}$$

Proposition (Reparameterization Invariance). Natural gradient descent gives the same update in parameter space regardless of how we parameterize the model. Standard gradient descent does not have this property.

This is invariant under reparameterization and often converges faster than standard gradient descent.

6 Part V: Gradient Derivations

We now derive the gradients for backpropagation through attention, using index notation throughout.

6.1 Chain Rule in Index Notation

For a scalar loss L, the chain rule gives:

$$\frac{\partial L}{\partial Q^{kl}} = \frac{\partial L}{\partial O^{ib}} \frac{\partial O^{ib}}{\partial A^{mn}} \frac{\partial A^{mn}}{\partial S^{pq}} \frac{\partial S^{pq}}{\partial Q^{kl}} \tag{36}$$

We compute each factor.

6.2 Gradient Through Value Aggregation

Given $O^{ib} = A^{ij} V^{jb}$:

Lemma (Value Aggregation Jacobian).

$$\frac{\partial O^{ib}}{\partial A^{mn}} = \delta^i_m V^{nb} \tag{37}$$

$$\frac{\partial O^{ib}}{\partial V^{mn}} = A^{im} \delta^b_n \tag{38}$$

Proof. For the first:

$$\frac{\partial O^{ib}}{\partial A^{mn}} = \frac{\partial}{\partial A^{mn}} \left(A^{ij} V^{jb} \right) = \delta^i_m \delta^j_n V^{jb} = \delta^i_m V^{nb} \tag{39}$$

For the second:

$$\frac{\partial O^{ib}}{\partial V^{mn}} = \frac{\partial}{\partial V^{mn}} \left(A^{ij} V^{jb} \right) = A^{ij} \delta^j_m \delta^b_n = A^{im} \delta^b_n \tag{40}$$

□

Therefore:

$$\frac{\partial L}{\partial A^{mn}} = \frac{\partial L}{\partial O^{mb}} V^{nb} \tag{41}$$

In matrix form: $\partial L / \partial A = (\partial L / \partial O) V^T$

And:

$$\frac{\partial L}{\partial V^{mn}} = A^{im} \frac{\partial L}{\partial O^{in}} \tag{42}$$

In matrix form: $\partial L / \partial V = A^T (\partial L / \partial O)$

6.3 Gradient Through Softmax

The softmax Jacobian is the trickiest part:

Lemma (Softmax Jacobian). For $A^{ij} = \exp(S^{ij}) / \sum_k \exp(S^{ik})$:

$$\frac{\partial A^{ij}}{\partial S^{mn}} = \delta^i_m A^{ij}(\delta^j_n - A^{in}) \tag{43}$$

Proof. The softmax for row i depends only on scores in row i, so δ^i_m ensures we're in the same row.

For the diagonal case ($j = n$):

$$\frac{\partial A^{ij}}{\partial S^{ij}} = \frac{\exp(S^{ij}) \cdot Z - \exp(S^{ij}) \cdot \exp(S^{ij})}{Z^2} = A^{ij} - \left(A^{ij}\right)^2 = A^{ij}(1 - A^{ij}) \tag{44}$$

For the off-diagonal case ($j \neq n$):

$$\frac{\partial A^{ij}}{\partial S^{in}} = \frac{0 \cdot Z - \exp(S^{ij}) \cdot \exp(S^{in})}{Z^2} = -A^{ij}A^{in} \tag{45}$$

Combining: $\frac{\partial A^{ij}}{\partial S^{in}} = A^{ij}(\delta^j_n - A^{in})$ □

The gradient through softmax becomes:

Theorem (Softmax Gradient).

$$\frac{\partial L}{\partial S^{mn}} = A^{mn}\left(\frac{\partial L}{\partial A^{mn}} - \sum_j A^{mj}\frac{\partial L}{\partial A^{mj}}\right) \tag{46}$$

In compact form:

$$\frac{\partial L}{\partial S} = A \circ \left(\frac{\partial L}{\partial A} - \text{rowsum}\left(A \circ \frac{\partial L}{\partial A}\right)\right) \tag{47}$$

where $\circ$ is elementwise multiplication.

> *Intuition.* The subtraction of the weighted sum is crucial—it ensures the gradient respects the constraint that attention weights sum to 1. If we increase one weight, others must decrease.

6.4 Gradient Through Score Computation

Given $S^{ij} = \frac{1}{\sqrt{d_k}}Q^{ia}K^{ja}$:

Lemma (Score Jacobian).

$$\frac{\partial S^{ij}}{\partial Q^{kl}} = \frac{1}{\sqrt{d_k}}\delta^i_k K^{jl} \tag{48}$$

$$\frac{\partial S^{ij}}{\partial K^{kl}} = \frac{1}{\sqrt{d_k}}\delta^j_k Q^{il} \tag{49}$$

Therefore:

Theorem (Query Gradient).

$$\frac{\partial L}{\partial Q^{kl}} = \frac{1}{\sqrt{d_k}} \frac{\partial L}{\partial S^{kj}} K^{jl} \tag{50}$$

In matrix form: $\partial L/\partial Q = \frac{1}{\sqrt{d_k}} \cdot (\partial L/\partial S)K$

Theorem (Key Gradient).

$$\frac{\partial L}{\partial K^{kl}} = \frac{1}{\sqrt{d_k}} \frac{\partial L}{\partial S^{ik}} Q^{il} \tag{51}$$

In matrix form: $\partial L/\partial K = \frac{1}{\sqrt{d_k}} \cdot (\partial L/\partial S)^T Q$

6.5 Complete Backward Pass

Combining all the pieces:

Theorem (Attention Backward Pass). Given upstream gradient $\partial L/\partial O$:

1. $\partial L/\partial V = A^T(\partial L/\partial O)$
2. $\partial L/\partial A = (\partial L/\partial O)V^T$
3. $\partial L/\partial S = A \circ (\partial L/\partial A - \text{rowsum}(A \circ \partial L/\partial A))$
4. $\partial L/\partial Q = \frac{1}{\sqrt{d_k}}(\partial L/\partial S)K$
5. $\partial L/\partial K = \frac{1}{\sqrt{d_k}}(\partial L/\partial S)^T Q$

```python
def attention_backward(dL_dO, Q, K, V, A):
    """
    Manual backward pass for attention.

    Args:
        dL_dO: Upstream gradient, shape (n_q, d_v)
        Q, K, V: Forward pass inputs
        A: Attention weights from forward pass

    Returns:
        dL_dQ, dL_dK, dL_dV
    """
    d_k = Q.shape[-1]
    scale = 1.0 / jnp.sqrt(d_k)

    # Step 1: dL/dV = A^T @ dL/dO
    dL_dV = jnp.einsum('ij,ib->jb', A, dL_dO)

    # Step 2: dL/dA = dL/dO @ V^T
    dL_dA = jnp.einsum('ib,jb->ij', dL_dO, V)

    # Step 3: dL/dS = A * (dL/dA - rowsum(A * dL/dA))
    sum_term = jnp.sum(A * dL_dA, axis=-1, keepdims=True)
    dL_dS = A * (dL_dA - sum_term)

    # Step 4: dL/dQ = scale * dL/dS @ K
```

```
dL_dQ = scale * jnp.einsum('ij,ja->ia', dL_dS, K)

# Step 5: dL/dK = scale * dL/dS^T @ Q
dL_dK = scale * jnp.einsum('ij,ia->ja', dL_dS, Q)

return dL_dQ, dL_dK, dL_dV
```

7 Part VI: Multi-Head Attention

Multi-head attention runs multiple attention operations in parallel with different learned projections.

7.1 Structure

Definition (Multi-Head Attention). For H heads with projection matrices $W_Q^h, W_K^h, W_V^h, W_O^h$:

Projections (introducing head index h):

$$Q^{hia} = X^{ib} W_Q^{hba} \tag{52}$$

$$K^{hja} = X^{jb} W_K^{hba} \tag{53}$$

$$V^{hjc} = X^{jb} W_V^{hbc} \tag{54}$$

Per-head attention:

$$S^{hij} = \frac{1}{\sqrt{d_k}} Q^{hia} K^{hja} \tag{55}$$

$$A^{hij} = \text{softmax}_j\left(S^{hij}\right) \tag{56}$$

$$O^{hic} = A^{hij} V^{hjc} \tag{57}$$

Final output (sum over heads and head dimension):

$$Y^{id} = O^{hic} W_O^{hcd} \tag{58}$$

Intuition. Each head can learn to attend to different aspects of the input. One head might focus on syntax, another on semantics, another on nearby tokens. The output projection W_O learns to combine these perspectives.

7.2 Geometric Interpretation

Each head projects to a different subspace:

$$Q_h = X W_Q^h \in \mathbb{R}^{n \times d_k} \tag{59}$$

With $d_k = d_{\text{model}}/H$, each head operates on a d_k-dimensional subspace of the full d_{model}-dimensional space.

Different heads can specialize:

- **Syntactic heads**: Attend to grammatical structure
- **Semantic heads**: Attend to meaning similarity
- **Positional heads**: Attend to relative positions

7.3 Head Diversity

Well-trained multi-head attention should have **diverse** heads that capture different relationships. We can measure diversity using:

$$\text{diversity} = 1 - \text{avg pairwise cosine similarity between head attention patterns} \tag{60}$$

8 Part VII: Attention Variants

8.1 Causal Masking

For autoregressive models, we prevent position i from attending to future positions $j > i$:

Definition (Causal Mask).

$$M^{ij} = \begin{cases} 1 \text{ if } j \leq i \\ 0 \text{ otherwise} \end{cases} \tag{61}$$

Applied as: $S^{ij} \leftarrow S^{ij} + (1 - M^{ij}) \cdot (-\infty)$

After softmax, $\exp(-\infty) = 0$, so future positions get zero weight.

8.2 Learned Bilinear Attention

Instead of scaled dot-product, use a learned metric:

$$S^{ij} = Q^{ia} M^{ab} K^{jb} \tag{62}$$

where M^{ab} is a learnable parameter (or $M = W^T W$ for positive definiteness).

8.3 Relative Position Attention

Add relative position information to keys:

$$S^{ij} = \frac{1}{\sqrt{d_k}} Q^{ia} \left(K^{ja} + R^{(i-j)a} \right) \tag{63}$$

where R^{ka} is a learned embedding for relative position k.

9 Part VIII: Hopfield Networks and Attention

A remarkable connection exists between transformer attention and modern Hopfield networks.

9.1 Classical Hopfield Networks

Classical Hopfield networks store patterns ξ_μ in a weight matrix:

$$W_{ij} = \frac{1}{N} \sum_\mu \xi^i_\mu \xi^j_\mu \tag{64}$$

The energy function is:

$$E(x) = -\frac{1}{2} x^i W_{ij} x^j \tag{65}$$

Fixed points (local minima) correspond to stored patterns.

Problem: Capacity scales only as $M \approx 0.14N$ patterns—not great!

9.2 Modern Hopfield Networks

Modern Hopfield networks use an exponential energy:

Definition (Modern Hopfield Energy).

$$E(\xi) = -\mathrm{lse}(\beta \cdot K\xi) + \frac{1}{2} \|\xi\|^2 + \mathrm{const} \tag{66}$$

where $\mathrm{lse}(z) = \log \sum_\mu \exp(z_\mu)$ is the log-sum-exp (smooth maximum).

The update rule is:

Theorem (Hopfield Update = Attention).

$$\xi^{\mathrm{new}} = V^T \ \mathrm{softmax}(\beta K \xi) \tag{67}$$

This is exactly the attention mechanism with:
- Query: current state ξ
- Keys: stored patterns (rows of K)
- Values: stored patterns (rows of V, often $V = K$)
- Inverse temperature: β

Proof. Taking the gradient of the energy and setting to zero:

$$\nabla_\xi E = -K^T \ \mathrm{softmax}(\beta K \xi) + \xi = 0 \tag{68}$$

$$\xi = K^T \ \mathrm{softmax}(\beta K \xi) \tag{69}$$

With values V, this generalizes to $\xi^{\mathrm{new}} = V^T \ \mathrm{softmax}(\beta K \xi)$. □

9.3 Exponential Storage Capacity

Theorem (Exponential Capacity). Modern Hopfield networks can store exponentially many patterns:

$$M \approx \exp(d/2) \tag{70}$$

compared to $M \approx 0.14N$ for classical networks.

The key is the exponential separation provided by softmax: even small differences in scores lead to large differences in weights.

Intuition. This is why transformers are so powerful! Each attention layer is essentially a Hopfield network that can store and retrieve from an exponentially large pattern library. The keys are the "stored patterns" and the query retrieves a weighted combination of values.

Attention	**Hopfield**
Query q	Pattern to retrieve
Keys K	Stored patterns
Values V	Pattern outputs
Softmax	Update rule
Output o	Retrieved pattern

10 Part IX: Efficient Attention

Standard attention has $O(n^2)$ complexity, which is problematic for long sequences. Let's look at two major approaches to efficiency.

10.1 Flash Attention

Flash Attention achieves $O(n)$ memory (instead of $O(n^2)$) by computing attention block-wise and avoiding storage of the full attention matrix.

10.1.1 The Key Insight: Online Softmax

Softmax can be computed incrementally without storing all values:

Proposition (Online Softmax). Given running maximum m and sum of exponentials ℓ, we can incorporate new elements:

$$m' = \max(m, \max(x_{\text{new}})) \tag{71}$$

$$\ell' = \ell \cdot \exp(m - m') + \sum \exp(x_{\text{new}} - m') \tag{72}$$

This numerical trick is the foundation of Flash Attention.

10.1.2 Block-wise Computation

Theorem (Flash Attention Algorithm). Divide Q, K, V into blocks of size B. For each query block Q_i:

1. Initialize $O_i = 0$, $\ell_i = 0$, $m_i = -\infty$
2. For each key-value block (K_j, V_j):
   - Compute block scores: $S_{ij} = Q_i K_j^T / \sqrt{d_k}$
   - Update running max: $m_{\text{new}} = \max(m_i, \text{rowmax}(S_{ij}))$
   - Rescale previous: $O_i \leftarrow O_i \cdot \exp(m_i - m_{\text{new}})$
   - Accumulate: $O_i \leftarrow O_i + \exp(S_{ij} - m_{\text{new}}) V_j$
   - Update normalization: $\ell_i \leftarrow \ell_i \cdot \exp(m_i - m_{\text{new}}) + \text{rowsum}(\exp(S_{ij} - m_{\text{new}}))$
   - Update: $m_i \leftarrow m_{\text{new}}$
3. Normalize: $O_i \leftarrow O_i / \ell_i$

Complexity comparison:

- Standard attention: $O(n^2)$ memory for storing A
- Flash Attention: $O(n)$ memory, recomputes A during backward pass

10.2 Linear Attention

Linear attention approximates the exponential kernel to achieve $O(n)$ time complexity.

Definition (Kernel Attention). Using feature map φ:

$$K(q, k) \approx \varphi(q)^T \varphi(k) \tag{73}$$

Then:

$$o_i = \frac{\sum_j \varphi(q_i)^T \varphi(k_j) v_j}{\sum_j \varphi(q_i)^T \varphi(k_j)} = \frac{\varphi(q_i)^T \sum_j \varphi(k_j) v_j^T}{\varphi(q_i)^T \sum_j \varphi(k_j)} \tag{74}$$

The sums $\sum_j \varphi(k_j) v_j^T$ and $\sum_j \varphi(k_j)$ can be precomputed in $O(nd^2)$ time, then each query costs $O(d^2)$ instead of $O(nd)$.

Common feature maps:
- Random Fourier features
- ELU + 1: $\varphi(x) = \text{ELU}(x) + 1$
- Positive random features (Performers)

> *Intuition.* The catch? Linear attention is an approximation. It loses some expressiveness compared to full softmax attention. The trade-off between efficiency and quality depends on your application.

11 Part X: Worked Examples

11.1 Example 1: 2-Query, 3-Key Attention

Consider:

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad V = \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \tag{75}$$

Step 1: Scores ($d_k = 2$, so $\frac{1}{\sqrt{d_k}} = \frac{1}{\sqrt{2}} \approx 0.707$)

$$S = \frac{1}{\sqrt{2}} QK^T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \approx \begin{pmatrix} 0.707 & 0 & 0.707 \\ 0 & 0.707 & 0.707 \end{pmatrix} \tag{76}$$

Step 2: Softmax (row-wise)

For row 1: $\exp([0.707, 0, 0.707]) \approx [2.028, 1.000, 2.028]$

Sum $= 5.056$, so $A_1 \approx [0.401, 0.198, 0.401]$

Row 2 is symmetric: $A_2 \approx [0.198, 0.401, 0.401]$

$$A \approx \begin{pmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \end{pmatrix} \tag{77}$$

Step 3: Output

$$O = AV = \begin{pmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \tag{78}$$

$$O_1 = 0.401 \cdot (2, 0) + 0.198 \cdot (0, 2) + 0.401 \cdot (1, 1) = (1.203, 0.797) \tag{79}$$

$$O_2 = 0.198 \cdot (2, 0) + 0.401 \cdot (0, 2) + 0.401 \cdot (1, 1) = (0.797, 1.203) \tag{80}$$

$$O \approx \begin{pmatrix} 1.203 & 0.797 \\ 0.797 & 1.203 \end{pmatrix} \tag{81}$$

11.2 Example 2: Temperature Effects

For scores $S = [2, 1, 0]$, attention weights at different temperatures:

T	**Weights**	**Entropy**
0.25	$[0.997, 0.003, 0.000]$	0.02
0.5	$[0.876, 0.118, 0.006]$	0.42
1.0	$[0.665, 0.245, 0.090]$	0.80
2.0	$[0.474, 0.316, 0.211]$	1.02
∞	$[0.333, 0.333, 0.333]$	1.10

Lower temperature concentrates probability on the maximum score.

11.3 Example 3: Softmax Jacobian Verification

For $s = [1, 2]$:

$$a = \mathrm{softmax}([1, 2]) = \left[\frac{e^1}{e^1 + e^2}, \frac{e^2}{e^1 + e^2} \right] \approx [0.269, 0.731] \tag{82}$$

The Jacobian is:

$$\frac{\partial a_i}{\partial s_j} = a_i(\delta_{ij} - a_j) \tag{83}$$

$$J = \begin{pmatrix} a_1(1 - a_1) & -a_1 a_2 \\ -a_2 a_1 & a_2(1 - a_2) \end{pmatrix} = \begin{pmatrix} 0.197 & -0.197 \\ -0.197 & 0.197 \end{pmatrix} \tag{84}$$

Note: Rows sum to 0 (as expected since softmax outputs sum to 1).

12 Appendix A: Notation Reference

Symbol	Meaning
Q^{ia}	Query tensor, position i, feature a
K^{ja}	Key tensor, position j, feature a
V^{jb}	Value tensor, position j, feature b
S^{ij}	Attention scores
A^{ij}	Attention weights (probabilities)
O^{ib}	Output tensor
g_{ab}	Metric tensor
g^{ab}	Inverse metric tensor
δ^a_b	Kronecker delta
Γ^c_{ab}	Christoffel symbols
Z^i	Partition function for query i
H^i	Entropy for query i
F^i	Free energy for query i
T	Temperature
β	Inverse temperature ($1/T$)
h	Head index in multi-head attention
W_Q, W_K, W_V, W_O	Projection weight matrices

13 Appendix B: The attn-tensors Library

All derivations in this document have been verified against JAX autodiff. The code is available in the accompanying `attn_tensors` Python package.

13.1 Library Architecture

```
attn_tensors/
├── attention.py    # Core attention operations
├── bilinear.py     # Metric tensors and bilinear forms
├── einsum.py       # Einstein summation utilities
├── gradients.py    # Manual gradient implementations
├── softmax.py      # Temperature, entropy, Gibbs
├── multihead.py    # Multi-head attention
├── masking.py      # Causal/padding masks
├── hopfield.py     # Hopfield network view
└── backend.py      # JAX/MLX backend selection
```

13.2 Quick Start

```
# Install
git clone https://github.com/bkataru-workshop/attn-as-bilinear-form
cd attn-as-bilinear-form
uv sync

# Run tests
uv run pytest tests/ -v

# With MLX (Apple Silicon)
uv sync --extra mlx
```

13.3 Gradient Verification

```
from attn_tensors.gradients import verify_gradients
import jax.numpy as jnp

Q = jnp.array([[1., 0.], [0., 1.]])
K = jnp.array([[1., 0.], [0., 1.], [1., 1.]])
V = jnp.array([[2., 0.], [0., 2.], [1., 1.]])

results = verify_gradients(Q, K, V)
print(results)  # {'dL_dQ': True, 'dL_dK': True, 'dL_dV': True, 'all_correct': True}
```

13.4 Backend Support

```
from attn_tensors import get_backend, Backend

# Auto-detect best backend
backend = get_backend()  # MLX on Apple Silicon, JAX otherwise

# Check availability
from attn_tensors import is_mlx_available
if is_mlx_available():
    print("Using MLX acceleration")
```

14 Appendix C: Further Reading

For those who want to dive deeper:

Original Papers:

- Vaswani et al., "Attention Is All You Need" (2017) - The Transformer paper
- Ramsauer et al., "Hopfield Networks is All You Need" (2020) - The Hopfield connection
- Dao et al., "FlashAttention" (2022) - Efficient attention

Mathematical Background:

- Wald, "General Relativity" - Tensor calculus and differential geometry
- Amari, "Information Geometry" - Fisher information and natural gradients
- Kardar, "Statistical Physics of Fields" - Statistical mechanics

Code and Documentation:

- GitHub: https://github.com/bkataru-workshop/attn-as-bilinear-form
- Documentation: https://bkataru-workshop.github.io/attn-as-bilinear-form/

Einsum Resources:

- Sankalp, "Shape Rotation 101: An Intro to Einsum and Jax Transformers" - Excellent practical einsum tutorial
- Alex Riley, "A basic introduction to NumPy's einsum" - Clear einsum fundamentals
- Tim Rocktäschel, "Einstein Summation in Numpy" - Einsum internals

Bibliography

[1] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[2] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[3] Sankalp, “Shape Rotation 101: An Intro to Einsum and Jax Transformers.” 2024.