Attention as Bilinear Form: A Tensor Calculus Perspective on Transformer Attention

Baalateja Kataru
baalateja.k@gmail.com

January 22, 2026

Abstract

We present a rigorous formulation of the transformer attention mechanism using the language of tensor calculus, differential geometry, and statistical mechanics. The standard attention operation $\text{Attention}(Q, K, V) = \text{softmax}\left(QK^T / \sqrt{d_k}\right)V$ conceals rich mathematical structure: the score computation is a bilinear form with an implicit metric tensor, the softmax normalization is the Gibbs distribution from thermodynamics, and the entire mechanism implements a modern Hopfield network with exponential storage capacity. We provide complete derivations in index notation with Einstein summation convention, explicit gradient computations verified against automatic differentiation, and connections to Riemannian geometry through the Fisher information metric. A companion Python library `attn-tensors` implements all operations with JAX, achieving machine-precision agreement between manual gradients and autodiff. The geometric perspective reveals natural generalizations including learned metrics, temperature-controlled attention, and efficient approximations. Code and documentation are publicly available at https://github.com/bkataru-workshop/attn-as-bilinear-form.

1. Introduction

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

This work recasts attention using the language of:

1. **Tensor calculus**: Index notation with proper contravariant/covariant indices and Einstein summation
2. **Bilinear forms**: The score computation as an inner product with metric tensor
3. **Statistical mechanics**: Softmax as the Gibbs/Boltzmann distribution
4. **Differential geometry**: Feature space as a Riemannian manifold
5. **Associative memory**: Attention as modern Hopfield network retrieval

The geometric perspective is not merely pedagogical. It suggests natural generalizations (learned metrics, variable temperature), explains empirical phenomena (the $\frac{1}{\sqrt{d_k}}$ scaling), and connects attention to well-studied mathematical structures with known properties.

1.1. Contributions

Our specific contributions include:

- **Complete tensor formulation**: Attention expressed in index notation with explicit index contractions

- **Gradient derivations**: Full backpropagation formulas derived in index notation and verified against JAX autodiff
- **Statistical mechanics interpretation**: Temperature, entropy, and free energy for attention
- **Hopfield network connection**: Formal equivalence between attention and modern Hopfield updates
- **Reference implementation**: Python library with 400+ tests achieving machine-precision verification
- **Efficient attention analysis**: Flash Attention and linear attention through the geometric lens

1.2. Notation Conventions

Throughout this paper, we use the following index conventions:

Index	**Meaning**
i	Query sequence position ($i = 1, ..., n_q$)
j, k	Key/value sequence position ($j = 1, ..., n_k$)
a, b, c	Feature/embedding dimension ($a = 1, ..., d$)
h	Attention head index ($h = 1, ..., H$)
μ, ν	General tensor indices

Einstein summation convention: Repeated indices (one upper, one lower) are implicitly summed:

$$v^a u_a \equiv \sum_{a=1}^{d} v^a u_a \tag{1}$$

2. Mathematical Foundations

2.1. Vectors and Dual Vectors

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

$$\langle u, v \rangle = u_a v^a \tag{2}$$

This pairing is basis-independent. In machine learning terms, vectors are column vectors and covectors are row vectors.

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

$$\langle u, v \rangle_g = u^a g_{ab} v^b \tag{3}$$

The metric enables:
- **Lowering indices**: $v_a = g_{ab} v^b$ (vector $\rightarrow$ covector)
- **Raising indices**: $v^a = g^{ab} v_b$ (covector $\rightarrow$ vector)

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

The Kronecker delta δ^a_b equals 1 when $a = b$ and 0 otherwise.

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:

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

In index notation with matrix M_{ab}:

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

The key insight is that attention scores are bilinear forms:

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

where g_{ab} encodes how similarity is measured in feature space.

2.3. Standard Metrics for Attention

Example (Euclidean Metric).

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

This gives the standard dot product: $\langle u, v \rangle = u^a v_a$

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 with dimension.

Example (Learned Metric).

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

Parameterizing as $W^T W$ ensures positive semi-definiteness. This generalizes 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 $\mathrm{Var}(q^a k_a) = d_k$. The scaling normalizes the variance to 1, keeping the softmax in a good operating regime.

3. The Attention Mechanism

3.1. Tensor Formulation

The attention mechanism operates on three inputs:

Definition (Attention Inputs).

- **Queries** Q^{ia}: Shape (n_q, d_k), what we're looking for
- **Keys** K^{ja}: Shape (n_k, d_k), what we're matching against
- **Values** V^{jb}: Shape (n_k, d_v), what we retrieve

Note: The feature index a is shared between Q and K (for matching), while V can have different feature dimension b.

The mechanism proceeds in three steps, each a tensor contraction.

3.1.1. Step 1: Score Computation (Bilinear Form)

Definition (Attention Scores).

$$S^{ij} = \frac{1}{\sqrt{d_k}} Q^{ia} K^{ja} \tag{10}$$

Or with explicit metric:

$$S^{ij} = Q^{ia} g_{ab} K^{jb} \tag{11}$$

where $g_{ab} = \left(1/\sqrt{d_k}\right)\delta_{ab}$ is the scaled Euclidean metric.

The contraction over a computes a scalar similarity for each query-key pair. This produces an $\left(n_q \times n_k\right)$ score matrix.

Implementation in JAX:

```python
import jax.numpy as jnp

def attention_scores(Q, K):
    """Compute S^{ij} = Q^{ia} K^{ja} / sqrt(d_k)"""
    d_k = Q.shape[-1]
    return jnp.einsum('ia,ja->ij', Q, K) / jnp.sqrt(d_k)
```

3.1.2. Step 2: Softmax Normalization

Definition (Attention Weights).

$$A^{ij} = \frac{\exp(S^{ij})}{\sum_k \exp(S^{ik})} = \frac{\exp(S^{ij})}{Z^i} \tag{12}$$

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

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

Properties of attention weights:

- Non-negative: $A^{ij} \geq 0$
- Normalized: $\sum_j A^{ij} = 1$ for each i
- Differentiable everywhere

3.1.3. Step 3: Value Aggregation

Definition (Attention Output).

$$O^{ib} = A^{ij} V^{jb} \tag{13}$$

This contracts over the key index j, computing a weighted average of values.

The output has shape (n_q, d_v): one vector per query.

3.2. Complete Attention Operation

Theorem (Scaled Dot-Product Attention). The full attention operation is:

$$O^{ib} = \frac{\exp(Q^{ia} g_{ac} K^{jc})}{\sum_k \exp(Q^{ia} g_{ac} K^{kc})} V^{jb} \tag{14}$$

In matrix notation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{15}$$

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. Statistical Mechanics Interpretation

The softmax function is the Gibbs distribution from statistical mechanics, revealing thermodynamic structure in attention.

4.1. The Gibbs/Boltzmann Distribution

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

Definition (Gibbs Distribution).

$$P(j) = \frac{\exp(-E_j/T)}{Z}, \quad Z = \sum_j \exp(-E_j/T) \tag{16}$$

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

For attention, we identify:

Proposition (Attention as Gibbs Distribution). Attention weights are Gibbs probabilities with:
- Scores as negative energies: $S^{ij} = -E^{ij}$
- Temperature: $T = 1$ (standard) or $T = \sqrt{d_k}$ (for unscaled scores)

$$A^{ij} = \frac{\exp(S^{ij}/T)}{\sum_k \exp(S^{ik}/T)} \tag{17}$$

4.2. Temperature Effects

Theorem (Temperature Limits). As temperature varies:

$$\lim_{T\to 0} A^{ij} = \begin{cases} 1 \text{ if } j = \arg\max_k S^{ik} \\ 0 \text{ otherwise} \end{cases} \quad \text{(hard attention)} \tag{18}$$

$$\lim_{T\to\infty} A^{ij} = \frac{1}{n_k} \quad \text{(uniform attention)} \tag{19}$$

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

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

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. □

Temperature-controlled attention:

```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

Definition (Attention Entropy). The Shannon entropy of attention weights for query i:

$$H^i = -\sum_j A^{ij} \log A^{ij} \tag{21}$$

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

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

Proposition (Entropy Bounds). For any attention distribution:

$$0 \leq H^i \leq \log n_k \tag{22}$$

Equality holds on the left iff attention is a delta function, and on the right iff attention is uniform.

Computing entropy:

```
def attention_entropy(weights, eps=1e-12):
    """H^i = -sum_j A^{ij} log A^{ij}"""
    return -jnp.sum(weights * jnp.log(weights + eps), axis=-1)

def normalized_entropy(weights, eps=1e-12):
    """Normalized to [0, 1]"""
    n_k = weights.shape[-1]
    return attention_entropy(weights, eps) / jnp.log(n_k)
```

4.4. Free Energy

Definition (Free Energy). The Helmholtz free energy for query i:

$$F^i = -T \log Z^i = -T \log \sum_j \exp(S^{ij}/T) \tag{23}$$

This satisfies the fundamental relation:

$$F = \langle E \rangle - TH \tag{24}$$

where $\langle E \rangle = -\sum_j A^{ij} S^{ij}$ is the expected energy.

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

$$A^* = \arg\min_A [\langle E \rangle - TH(A)] \tag{25}$$

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:

$$\mathcal{L} = -\sum_j A_j S_j + T \sum_j A_j \log A_j + \lambda \left(\sum_j A_j - 1 \right) \tag{26}$$

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

$$-S_j + T\big(1 + \log A_j\big) + \lambda = 0 \tag{27}$$

$$A_j = \exp\big(\big(S_j - \lambda - T\big)/T\big) \propto \exp\big(S_j/T\big) \tag{28}$$

Normalizing gives the softmax. □

5. Gradient Derivations

We now derive the gradients for backpropagation through attention in index notation.

5.1. Chain Rule in Index Notation

For a scalar loss L, the chain rule gives:

$$\frac{\partial L}{\partial Q^{kl}} = \frac{\partial L}{\partial O^{ib}} \frac{\partial O^{ib}}{\partial A^{mn}} \frac{\partial A^{mn}}{\partial S^{pq}} \frac{\partial S^{pq}}{\partial Q^{kl}} \tag{29}$$

We compute each Jacobian factor explicitly.

5.2. Gradient Through Value Aggregation

Lemma (Value Aggregation Jacobian). For $O^{ib} = A^{ij}V^{jb}$:

$$\frac{\partial O^{ib}}{\partial A^{mn}} = \delta^i_m V^{nb} \tag{30}$$

$$\frac{\partial O^{ib}}{\partial V^{mn}} = A^{im}\delta^b_n \tag{31}$$

Proof. For the first:

$$\frac{\partial O^{ib}}{\partial A^{mn}} = \frac{\partial}{\partial A^{mn}}\big(A^{ij}V^{jb}\big) = \delta^i_m \delta^j_n V^{jb} = \delta^i_m V^{nb} \tag{32}$$

For the second:

$$\frac{\partial O^{ib}}{\partial V^{mn}} = \frac{\partial}{\partial V^{mn}}\big(A^{ij}V^{jb}\big) = A^{ij}\delta^j_m \delta^b_n = A^{im}\delta^b_n \tag{33}$$

□

Theorem (Value Gradient).

$$\frac{\partial L}{\partial V^{mn}} = A^{im} \frac{\partial L}{\partial O^{in}} \tag{34}$$

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

5.3. Gradient Through Softmax

The softmax Jacobian requires careful derivation.

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

$$\frac{\partial A^{ij}}{\partial S^{mn}} = \delta^i_m A^{ij}(\delta^j_n - A^{in}) \tag{35}$$

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 ($j = n$):

$$\frac{\partial A^{ij}}{\partial S^{ij}} = \frac{\exp(S^{ij}) \cdot Z - \exp(S^{ij}) \cdot \exp(S^{ij})}{Z^2} = A^{ij} - \left(A^{ij}\right)^2 = A^{ij}(1 - A^{ij}) \tag{36}$$

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

$$\frac{\partial A^{ij}}{\partial S^{in}} = \frac{0 \cdot Z - \exp(S^{ij}) \cdot \exp(S^{in})}{Z^2} = -A^{ij}A^{in} \tag{37}$$

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

Theorem (Score Gradient).

$$\frac{\partial L}{\partial S^{mn}} = A^{mn}\left(\frac{\partial L}{\partial A^{mn}} - \sum_j A^{mj}\frac{\partial L}{\partial A^{mj}}\right) \tag{38}$$

In compact form, defining $\overline{A}^{ij} = \partial L / \partial A^{ij}$:

$$\frac{\partial L}{\partial S} = A \circ \left(\overline{A} - \operatorname{rowsum}\left(A \circ \overline{A}\right)\right) \tag{39}$$

where $\circ$ denotes elementwise multiplication.

Proof. Using the chain rule and softmax Jacobian:

$$\frac{\partial L}{\partial S^{mn}} = \frac{\partial L}{\partial A^{ij}}\frac{\partial A^{ij}}{\partial S^{mn}} = \frac{\partial L}{\partial A^{ij}}\delta^i_m A^{ij}(\delta^j_n - A^{in}) \tag{40}$$

The δ^i_m forces $i = m$:

$$= \frac{\partial L}{\partial A^{mj}}A^{mj}(\delta^j_n - A^{mn}) \tag{41}$$

Expanding:

$$= \frac{\partial L}{\partial A^{mn}}A^{mn} - A^{mn}\sum_j \frac{\partial L}{\partial A^{mj}}A^{mj} \tag{42}$$

$$= A^{mn}\left(\frac{\partial L}{\partial A^{mn}} - \sum_j A^{mj}\frac{\partial L}{\partial A^{mj}}\right) \tag{43}$$

□

5.4. Gradient Through Score Computation

Lemma (Score Jacobian). For $S^{ij} = \left(1/\sqrt{d_k}\right)Q^{ia}K^{ja}$:

$$\frac{\partial S^{ij}}{\partial Q^{kl}} = \frac{1}{\sqrt{d_k}} \delta^i_k K^{jl} \tag{44}$$

$$\frac{\partial S^{ij}}{\partial K^{kl}} = \frac{1}{\sqrt{d_k}} \delta^j_k Q^{il} \tag{45}$$

Proof. For queries:

$$\frac{\partial S^{ij}}{\partial Q^{kl}} = \frac{1}{\sqrt{d_k}} \frac{\partial}{\partial Q^{kl}} (Q^{ia} K^{ja}) = \frac{1}{\sqrt{d_k}} \delta^i_k \delta^a_l K^{ja} = \frac{1}{\sqrt{d_k}} \delta^i_k K^{jl} \tag{46}$$

The derivation for keys is analogous. □

Theorem (Query and Key Gradients).

$$\frac{\partial L}{\partial Q^{kl}} = \frac{1}{\sqrt{d_k}} \frac{\partial L}{\partial S^{kj}} K^{jl} \tag{47}$$

$$\frac{\partial L}{\partial K^{kl}} = \frac{1}{\sqrt{d_k}} \frac{\partial L}{\partial S^{ik}} Q^{il} \tag{48}$$

In matrix form:

$$\partial L / \partial Q = \left(1/\sqrt{d_k}\right)(\partial L / \partial S) K \tag{49}$$

$$\partial L / \partial K = \left(1/\sqrt{d_k}\right)(\partial L / \partial S)^T Q \tag{50}$$

5.5. Complete Backward Pass Algorithm

Theorem (Attention Backward Pass). Given upstream gradient $\overline{O} = \partial L / \partial O$, the complete backward pass is:

1. **Value gradient**: $\overline{V} = A^T \overline{O}$
2. **Attention weight gradient**: $\overline{A} = \overline{O} V^T$
3. **Score gradient**: $\overline{S} = A \circ \left(\overline{A} - \text{rowsum}\left(A \circ \overline{A}\right)\right)$
4. **Query gradient**: $\overline{Q} = \left(1/\sqrt{d_k}\right) \overline{S} K$
5. **Key gradient**: $\overline{K} = \left(1/\sqrt{d_k}\right) \overline{S}^T Q$

Complete implementation:

```
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
```

5.6. Gradient Verification

All manual gradients are verified against JAX automatic differentiation:

```
import jax
from jax import random

def verify_gradients(Q, K, V, tol=1e-5):
    """Verify manual gradients match autodiff."""

    def loss_fn(Q, K, V):
        O = scaled_dot_product_attention(Q, K, V)
        return jnp.sum(O ** 2)  # Simple loss

    # Autodiff gradients
    auto_dQ, auto_dK, auto_dV = jax.grad(loss_fn, argnums=(0, 1, 2))(Q, K, V)

    # Manual gradients
    O, A = attention_with_weights(Q, K, V)
    dL_dO = 2 * O  # Gradient of sum(O^2)
    manual_dQ, manual_dK, manual_dV = attention_backward(dL_dO, Q, K, V, A)

    # Compare
    return {
        'dL_dQ': jnp.allclose(auto_dQ, manual_dQ, atol=tol),
        'dL_dK': jnp.allclose(auto_dK, manual_dK, atol=tol),
        'dL_dV': jnp.allclose(auto_dV, manual_dV, atol=tol),
    }

# Test
```

```python
key = random.PRNGKey(42)
Q = random.normal(key, (10, 64))
K = random.normal(random.split(key)[0], (20, 64))
V = random.normal(random.split(key)[1], (20, 64))

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

6. Multi-Head Attention

Multi-head attention runs H independent attention operations with different learned projections, then combines the results.

6.1. Tensor Formulation

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):

$$Q^{hia} = X^{ib} W_Q^{hba} \tag{51}$$

$$K^{hja} = X^{jb} W_K^{hba} \tag{52}$$

$$V^{hjc} = X^{jb} W_V^{hbc} \tag{53}$$

Per-head attention:

$$S^{hij} = \frac{1}{\sqrt{d_k}} Q^{hia} K^{hja} \tag{54}$$

$$A^{hij} = \operatorname{softmax}_j\left(S^{hij}\right) \tag{55}$$

$$O^{hic} = A^{hij} V^{hjc} \tag{56}$$

Output projection (sum over heads and head dimension):

$$Y^{id} = O^{hic} W_O^{hcd} \tag{57}$$

6.2. Parameter Count

For a transformer layer with model dimension d_{model} and H heads:

Parameter	**Shape**	**Count**
W_Q	$(d_{\text{model}}, d_{\text{model}})$	d_{model}^2
W_K	$(d_{\text{model}}, d_{\text{model}})$	d_{model}^2
W_V	$(d_{\text{model}}, d_{\text{model}})$	d_{model}^2
W_O	$(d_{\text{model}}, d_{\text{model}})$	d_{model}^2
Total		$4d_{\text{model}}^2$

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

6.3. Geometric Interpretation

Each head projects to a different subspace and computes attention there:

$$Q_h = XW_Q^h \in \mathbb{R}^{n \times d_k} \tag{58}$$

Different heads can specialize:

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

The output projection W_O learns to combine these perspectives.

7. Hopfield Networks and Attention

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

7.1. Classical Hopfield Networks

Classical Hopfield networks (1982) store M patterns ξ_μ in a weight matrix:

$$W_{ij} = \frac{1}{N} \sum_{\mu=1}^{M} \xi_\mu^i \xi_\mu^j \tag{59}$$

The energy function is:

$$E(x) = -\frac{1}{2} x^i W_{ij} x^j \tag{60}$$

The network dynamics minimize energy, converging to stored patterns. However, capacity is severely limited: $M \leq 0.14N$ patterns can be reliably stored.

7.2. Modern Hopfield Networks

Ramsauer et al. (2020) introduced an exponential energy function:

Definition (Modern Hopfield Energy).

$$E(\xi) = -\mathrm{lse}(\beta \cdot K\xi) + \frac{1}{2} \|\xi\|^2 + \mathrm{const} \tag{61}$$

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

The update rule that minimizes this energy is:

Theorem (Hopfield Update Equals Attention).

$$\xi^{\mathrm{new}} = V^T \ \mathrm{softmax}(\beta K \xi) \tag{62}$$

This is precisely 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:

$$\nabla_\xi E = -K^T \text{ softmax}(\beta K \xi) + \xi = 0 \tag{63}$$

$$\xi = K^T \text{ softmax}(\beta K \xi) \tag{64}$$

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

7.3. Exponential Storage Capacity

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

$$M \approx \exp(d/2) \tag{65}$$

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

The key is the exponential separation provided by softmax:

$$\text{softmax}(\beta x)_i \approx \begin{cases} 1 \text{ if } x_i = \max(x) \\ \exp(-\beta \Delta) \text{ if } x_i = \max(x) - \Delta \end{cases} \tag{66}$$

For large β, even small separation Δ gives clean retrieval.

7.4. Attention as Associative Memory

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

8. Efficient Attention Variants

8.1. 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.

8.1.1. Online Softmax Algorithm

The key insight is that softmax can be computed incrementally:

Proposition (Online Softmax). Given running maximum m and sum of exponentials ℓ, we can incorporate new elements without storing all values:

$$m' = \max(m, \max(x_{\text{new}})) \tag{67}$$

$$\ell' = \ell \cdot \exp(m - m') + \sum \exp(x_{\text{new}} - m') \tag{68}$$

8.1.2. Block-wise Computation

Theorem (Flash Attention). 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: $\ell_i \leftarrow \ell_i \cdot \exp(m_i - m_{\text{new}}) + \text{rowsum}(\exp(S_{ij} - m_{\text{new}}))$
   - Update: $m_i \leftarrow m_{\text{new}}$
3. Normalize: $O_i \leftarrow O_i / \ell_i$

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

8.2. Linear Attention

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

Definition (Kernel Attention). Using feature map φ:

$$K(q, k) \approx \varphi(q)^T \varphi(k) \tag{69}$$

Then:

$$o_i = \frac{\sum_j \varphi(q_i)^T \varphi(k_j) v_j}{\sum_j \varphi(q_i)^T \varphi(k_j)} = \frac{\varphi(q_i)^T \sum_j \varphi(k_j) v_j^T}{\varphi(q_i)^T \sum_j \varphi(k_j)} \tag{70}$$

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)

9. Worked Examples

9.1. Example 1: Complete Forward Pass

Consider the following inputs with $n_q = 2$, $n_k = 3$, $d_k = d_v = 2$:

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad V = \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \tag{71}$$

Step 1: Compute scores with $\sqrt{d_k} = \sqrt{2} \approx 1.414$

$$S = \frac{1}{\sqrt{2}} QK^T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \approx \begin{pmatrix} 0.707 & 0 & 0.707 \\ 0 & 0.707 & 0.707 \end{pmatrix} \tag{72}$$

Step 2: Apply softmax (row-wise)

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

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

For row 2: By symmetry, $A_2 \approx [0.198, 0.401, 0.401]$

$$A \approx \begin{pmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \end{pmatrix} \tag{73}$$

Step 3: Compute output

$$O = AV = \begin{pmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \tag{74}$$

$$O_1 = 0.401 \cdot (2, 0) + 0.198 \cdot (0, 2) + 0.401 \cdot (1, 1) = (1.203, 0.797) \tag{75}$$

$$O_2 = 0.198 \cdot (2, 0) + 0.401 \cdot (0, 2) + 0.401 \cdot (1, 1) = (0.797, 1.203) \tag{76}$$

$$O \approx \begin{pmatrix} 1.203 & 0.797 \\ 0.797 & 1.203 \end{pmatrix} \tag{77}$$

9.2. Example 2: Temperature Effects

For scores $s = [2, 1, 0]$, we compute softmax 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.

9.3. Example 3: Gradient Verification

We verify the softmax Jacobian for $s = [1, 2]$:

$$a = \text{softmax}([1, 2]) = \left[\frac{e^1}{e^1 + e^2}, \frac{e^2}{e^1 + e^2} \right] \approx [0.269, 0.731] \tag{78}$$

The Jacobian is:

$$\frac{\partial a_i}{\partial s_j} = a_i(\delta_{ij} - a_j) \tag{79}$$

$$J = \begin{pmatrix} a_1(1-a_1) & -a_1a_2 \\ -a_2a_1 & a_2(1-a_2) \end{pmatrix} = \begin{pmatrix} 0.197 & -0.197 \\ -0.197 & 0.197 \end{pmatrix} \tag{80}$$

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

10. Implementation and Validation

10.1. Library Architecture

The `attn-tensors` library implements all operations in JAX with the following structure:

```
attn_tensors/
├── attention.py    # Core attention operations
├── bilinear.py     # Metric tensors and bilinear forms
├── 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
```

10.2. Test Coverage

The library includes 400+ tests covering:

- Tensor shapes and contractions
- Gradient correctness (vs. JAX autodiff)
- Numerical stability (large scores, small temperatures)
- Edge cases (single key, identical keys, zero inputs)
- Property-based testing with Hypothesis

Example test output:

```
tests/test_attention.py .... 63 passed
tests/test_bilinear.py .... 45 passed
tests/test_gradients.py .... 89 passed
tests/test_softmax.py .... 52 passed
tests/test_multihead.py .... 41 passed
========================= 400+ tests passed =========================
```

10.3. Backend Support

The library supports multiple backends:

```python
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")
```

11. Conclusion

We have presented a comprehensive tensor calculus formulation of the attention mechanism, revealing its deep mathematical structure:

1. **Bilinear forms**: Attention scores arise from an inner product with implicit metric tensor
2. **Statistical mechanics**: Softmax is the Gibbs distribution; temperature controls attention sharpness
3. **Differential geometry**: The feature space is a Riemannian manifold with the metric defining similarity
4. **Associative memory**: Attention implements modern Hopfield network retrieval with exponential capacity

The geometric perspective is not merely pedagogical—it suggests natural generalizations (learned metrics, variable temperature) and connects attention to well-studied mathematical structures.

All derivations have been verified against automatic differentiation, with the reference implementation publicly available. The tensor notation, while initially unfamiliar to machine learning practitioners, provides a powerful and rigorous language for understanding and extending attention mechanisms.

11.1. Future Directions

Natural extensions include:

- **Riemannian optimization**: Natural gradient descent on attention parameters
- **Learned geometries**: End-to-end learning of the metric tensor
- **Geometric regularization**: Entropy penalties through the thermodynamic lens
- **Efficient approximations**: Geometric analysis of linear attention quality

12. Acknowledgments

We thank the JAX and MLX development teams for their excellent automatic differentiation libraries. This work builds on foundational research in differential geometry, statistical mechanics, and associative memory.

13. 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
O^{ib}	Output tensor
g_{ab}	Metric tensor
g^{ab}	Inverse metric
δ^a_b	Kronecker delta
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

14. Appendix B: Code Repository

Full source code, documentation, and examples:

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

14.1. 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
```

14.2. Basic Usage

```python
import jax.numpy as jnp
from attn_tensors import scaled_dot_product_attention
from attn_tensors.bilinear import scaled_euclidean_metric, bilinear_form_batch

# Standard attention
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.]])

output = scaled_dot_product_attention(Q, K, V)

# With explicit metric
g = scaled_euclidean_metric(d=2)
scores = bilinear_form_batch(Q, K, g)
```