Harmonic Loss: Computational Optimizations for Scalable Implementation

kyokopom

@Kyokopom

github.com/Kitsunp/Harmonic-loss

https://x.com/Kyokopom

September 1, 2025

Abstract

This document presents a comprehensive analysis of Harmonic Loss and two critical computational optimizations that enable practical deployment in large-scale language models. The first optimization eliminates prohibitively large intermediate tensors through algebraic expansion of the squared Euclidean distance. The second ensures numerical stability via log-space transformations using native model precision. These optimizations reduce memory complexity from $O(BSVN)$ to $O(BSV)$ while maintaining exact mathematical equivalence to the theoretical formulation of Baek et al. (2025).

1 Introduction

Harmonic Loss, introduced by Baek et al. (2025), represents a paradigm shift from inner product-based cross-entropy loss to distance-based probability computation. While theoretically elegant, the naive implementation presents computational challenges that render it impractical for modern language models. This document details two fundamental optimizations that bridge the gap between theory and practical implementation.

2 Mathematical Foundations

2.1 Formulation Comparison

Let $W \in \mathbb{R}^{V \times N}$ denote the unembedding matrix and $x \in \mathbb{R}^N$ the final hidden state, where V is vocabulary size and N is hidden dimension.

Loss Function Definitions

Cross-Entropy Loss:

$$\text{logits:} \quad y_i = w_i^T x \tag{1}$$

$$\text{probabilities:} \quad p_i = \frac{\exp(y_i)}{\sum_{j=1}^{V} \exp(y_j)} \tag{2}$$

$$\text{loss:} \quad \mathcal{L}_{CE} = -\log p_c \tag{3}$$

Harmonic Loss (Baek et al., 2025):

$$\text{distances:} \quad d_i = \|w_i - x\|^2 \tag{4}$$

$$\text{probabilities:} \quad p_i = \frac{d_i^{-n}}{\sum_{j=1}^{V} d_j^{-n}} \tag{5}$$

$$\text{loss:} \quad \mathcal{L}_H = -\log p_c \tag{6}$$

The harmonic exponent $n > 0$ controls the sharpness of the probability distribution, typically chosen as $n \approx \sqrt{N}$ following the theoretical analysis in Baek et al. (2025).

2.2 Key Theoretical Properties

[Finite Convergence (Baek et al., 2025)] Unlike cross-entropy loss requiring $y_c \to \infty$ for perfect classification, Harmonic Loss achieves $p_c \to 1$ through $d_c \to 0$, representing a finite convergence point.

[Scale Invariance] Harmonic Loss exhibits scale invariance: scaling all distances by $\alpha > 0$ leaves probabilities unchanged.

Under uniform scaling $d_i \mapsto \alpha d_i$:

$$p_i' = \frac{(\alpha d_i)^{-n}}{\sum_j (\alpha d_j)^{-n}} = \frac{\alpha^{-n} d_i^{-n}}{\alpha^{-n} \sum_j d_j^{-n}} = \frac{d_i^{-n}}{\sum_j d_j^{-n}} = p_i \tag{7}$$

3 The Computational Challenge

The direct implementation of Equation 4 creates severe scalability bottlenecks.

3.1 Complexity Analysis

4 Optimization 1: Algebraic Distance Expansion

4.1 Mathematical Foundation

The key insight lies in the algebraic expansion of squared Euclidean distance:

Fundamental Mathematical Identity

$$\boxed{\|w_i - x\|^2 = \|w_i\|^2 + \|x\|^2 - 2\langle w_i, x \rangle} \tag{8}$$

Algorithm 1: Naive Implementation (Computationally Prohibitive)

Input : $x \in \mathbb{R}^{B\times S\times N}$, $W \in \mathbb{R}^{V\times N}$, $n > 0$
Output: $p \in \mathbb{R}^{B\times S\times V}$, $\mathcal{L}$

$x_{expand} \leftarrow \text{expand}(x, \text{axis} = 2)$
// Shape becomes $[B, S, 1, N]$

$W_{expand} \leftarrow \text{expand}(W, \text{axis} = (0, 1))$
// Shape becomes $[1, 1, V, N]$

$\Delta \leftarrow x_{expand} - W_{expand}$
// CRITICAL: Creates $[B, S, V, N]$ tensor

$d \leftarrow \|\Delta\|^2_{\text{axis}=-1}$
// Euclidean distances: $[B, S, V]$

$p \leftarrow \frac{d^{-n}}{\sum_j d_j^{-n}}$
// Numerical instability for extreme values

Resource	Complexity	Example (B=4, S=512, V=50K, N=4K)
Peak Memory	$O(BSVN)$	~ 400 GB
Operations	$O(3 \cdot BSVN)$	$\sim 1.2 \times 10^{12}$ FLOPs
Memory Pattern	Random Access	GPU memory-bound
Parallelization	Poor	Element-wise operations

Complete Mathematical Derivation

Proof of Distance Expansion:
Starting from the definition of squared Euclidean norm:

$$\|w_i - x\|^2 = (w_i - x)^T(w_i - x) \tag{9}$$
$$= w_i^T w_i - w_i^T x - x^T w_i + x^T x \tag{10}$$
$$= w_i^T w_i - 2w_i^T x + x^T x \quad (\text{since } w_i^T x = x^T w_i) \tag{11}$$
$$= \|w_i\|^2 + \|x\|^2 - 2\langle w_i, x\rangle \tag{12}$$

Computational Transformation: This identity decomposes one memory-intensive operation into three efficient computations:

1. **Inner Products:** $W^T x$ via optimized GEMM
2. **Hidden Norms:** $\|x\|^2$ per sequence position
3. **Weight Norms:** $\|W\|^2$ per vocabulary entry (cacheable)

The final broadcasting step requires only $O(BSV)$ memory.

Algorithm 2: Optimization 1: Memory-Efficient Distance Computation

Input : $x \in \mathbb{R}^{B\times S\times N}$, $W \in \mathbb{R}^{V\times N}$
Output: $d \in \mathbb{R}^{B\times S\times V}$

Step 1: Compute Inner Products
$y \leftarrow x \cdot W^T$
// GPU-optimized GEMM: $2BSVN$ FLOPs
// Leverages Tensor Cores, high arithmetic intensity
Step 2: Compute Hidden State Norms
$x^2 \leftarrow \sum_{i=1}^{N} x_{:,:,i}^2$
// Per-position squared norms: shape $[B, S, 1]$
Step 3: Compute Weight Norms
$w^2 \leftarrow \sum_{i=1}^{N} W_{:,i}^2$
// Per-vocabulary squared norms: shape $[V]$
// Can be precomputed and cached across forwards
Step 4: Reconstruct Distances
$d \leftarrow x^2 + w^2 - 2y$
// Efficient broadcasting: $[B, S, 1] + [V] - 2[B, S, V]$
// No intermediate $[B, S, V, N]$ tensor created

4.2 Efficient Algorithm

5 Optimization 2: Log-Space Transformation

5.1 Numerical Stability Problem

Direct computation of $p_i = \frac{d_i^{-n}}{\sum_j d_j^{-n}}$ suffers from:

- **Overflow:** Small distances $d_i \to 0$ cause $d_i^{-n} \to \infty$
- **Underflow:** Large distances $d_i \gg 1$ cause $d_i^{-n} \to 0$
- **Precision Loss:** Extreme dynamic ranges exceed floating-point precision

5.2 Mathematical Equivalence

Define harmonic logits as:

$$z_i \equiv -n \log d_i \tag{13}$$

By convention in machine learning literature, we use z to denote logits (transformed activations before softmax). In this context, the harmonic logits z_i represent the log-space transformation of inverse distance probabilities.

[Log-Space Equivalence] The harmonic probabilities satisfy:

$$p_i = \frac{d_i^{-n}}{\sum_{j=1}^{V} d_j^{-n}} = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}} = \text{softmax}(z)_i \tag{14}$$

Substituting $z_i = -n \log d_i$:

$$\frac{e^{z_i}}{\sum_j e^{z_j}} = \frac{e^{-n \log d_i}}{\sum_j e^{-n \log d_j}} \tag{15}$$

$$= \frac{(e^{\log d_i})^{-n}}{\sum_j (e^{\log d_j})^{-n}} \tag{16}$$

$$= \frac{d_i^{-n}}{\sum_j d_j^{-n}} = p_i \tag{17}$$

5.3 Gradient Preservation

[Gradient Equivalence] Log-space transformation preserves exact gradients.

For harmonic loss $\mathcal{L} = -\log p_c$:

$$\frac{\partial \mathcal{L}}{\partial z_i} = p_i - \delta_{ic} \quad \text{(standard softmax gradient)} \tag{18}$$

$$\frac{\partial z_i}{\partial d_i} = -\frac{n}{d_i} \tag{19}$$

$$\therefore \frac{\partial \mathcal{L}}{\partial d_i} = (p_i - \delta_{ic}) \cdot \left(-\frac{n}{d_i}\right) = \frac{n(\delta_{ic} - p_i)}{d_i} \tag{20}$$

This exactly matches the direct harmonic loss gradient.

Algorithm 3: Optimization 2: Numerically Stable Log-Space Transformation (Simplified)

Input : $d \in \mathbb{R}^{B \times S \times V}$, $n > 0$, $\varepsilon > 0$
Output: $z \in \mathbb{R}^{B \times S \times V}$ (harmonic logits)

Step 1: Numerical Safety
$d_{safe} \leftarrow \text{clamp\_min}(d, \varepsilon)$
`// Prevent` $\log(0)$`; typically` $\varepsilon = 10^{-6}$
`// Modern GPUs handle this operation efficiently in native precision`

Step 2: Logarithmic Transformation
$z \leftarrow -n \cdot \log(d_{safe})$
`// Compute harmonic logits in native model precision`
`// Equivalent to` $\log(1/d^n)$ `but numerically stable`

6 Complete Production Implementation

7 Performance Analysis

7.1 Complexity Comparison

7.2 Practical Overhead

The optimized implementation introduces minimal overhead:

- **Dominant cost:** GEMM $W^T h$ (identical to cross-entropy)
- **Additional operations:** Norm computations $O(BSN + VN)$ and logarithms $O(BSV)$

Algorithm 4: Harmonic Loss: Complete Optimized Implementation

Input : $h \in \mathbb{R}^{B \times S \times N}$, $W \in \mathbb{R}^{V \times N}$, labels, $n > 0$, $\varepsilon > 0$
Output: $z \in \mathbb{R}^{B \times S \times V}$ (harmonic logits), $\mathcal{L}$

PHASE 1: Algebraic Distance Computation

Step 1.1: Standard Matrix Multiplication

$y \leftarrow h \cdot W^T$

// GEMM operation: $O(2BSVN)$ FLOPs

// Utilizes GPU Tensor Cores for acceleration

Step 1.2: Hidden State Norm Computation

$h^2 \leftarrow \sum_{k=1}^{N} h_{:,:,k}^2$

// Result shape: $[B, S, 1]$

// Keepdims=True for broadcasting compatibility

Step 1.3: Weight Norm Computation

$w^2 \leftarrow \sum_{k=1}^{N} W_{:,k}^2$

// Result shape: $[V]$

// Can be cached across multiple forward passes

Step 1.4: Distance Reconstruction

$d \leftarrow h^2 + w^2 - 2y$

// Broadcasting: $[B, S, 1] + [V] + [B, S, V] \rightarrow [B, S, V]$

// Uses identity: $\|w_i - x\|^2 = \|w_i\|^2 + \|x\|^2 - 2\langle w_i, x \rangle$

PHASE 2: Log-Space Stabilization

Step 2.1: Numerical Clamping

$d \leftarrow \text{clamp\_min}(d, \varepsilon)$

// Ensures numerical stability, prevents $\log(0)$

Step 2.2: Harmonic Logit Computation

$z \leftarrow -n \cdot \log(d)$

// Transform distances to harmonic logits

// Mathematically equivalent to harmonic probabilities

PHASE 3: Causal Language Model Loss

Step 3.1: Causal Shifting

if *labels* $\neq$ *null* **then**

  $z_{shift} \leftarrow z_{:,:-1,:}$

  // Remove last position for autoregressive prediction

  $labels_{shift} \leftarrow labels_{:,1:}$

  // Remove first position to align with shifted logits

  Step 3.2: Loss Computation

  $\mathcal{L} \leftarrow \text{CrossEntropy}(z_{shift}, labels_{shift})$

  // Standard cross-entropy over harmonic logits

  // Equivalent to original harmonic loss formulation

else

  // No loss computation needed for inference

return z, $\mathcal{L}$

Implementation	Memory	FLOPs	GPU Utilization
Naive Harmonic	$O(BSVN)$	$O(3 \cdot BSVN)$	Poor (element-wise)
Optimized Harmonic	$O(BSV)$	$O(2 \cdot BSVN)$	Excellent (GEMM)
Standard Cross-Entropy	$O(BSV)$	$O(2 \cdot BSVN)$	Excellent (GEMM)

- **Typical overhead:** $< 5\%$ wall-clock time compared to standard softmax
- **Memory footprint:** Practically identical to cross-entropy loss

8 Theoretical Guarantees Preserved

The optimizations maintain all theoretical advantages of Harmonic Loss (Baek et al., 2025):

1. **Finite Convergence:** Weights converge to finite class centers rather than diverging
2. **Interpretability:** Weight vectors align with class centroids in representation space
3. **Scale Invariance:** Robust to input and weight scaling transformations
4. **Faster Training:** Reduced grokking behavior on algorithmic tasks
5. **Better Generalization:** Improved data efficiency and reduced overfitting

9 Conclusion

We have presented two fundamental optimizations that transform Harmonic Loss from a theoretically compelling but computationally prohibitive approach into a practical alternative for large-scale language models:

1. **Algebraic Expansion:** Eliminates $O(BSVN)$ memory requirement through mathematical identity
2. **Log-Space Transformation:** Ensures numerical stability using native model precision

These optimizations achieve order-of-magnitude memory savings and significant computational speedups while maintaining all theoretical properties of the original formulation. The resulting implementation enables practitioners to leverage the interpretability and convergence advantages of Harmonic Loss in production systems.

10 Implementation Considerations

10.1 Precision Management

The log-space transformation is robust across precision formats including bfloat16 and fp16. Numerical stability is achieved through proper distance clamping rather than precision elevation, making the implementation simpler and more efficient.

10.2 Caching Opportunities

Weight norms w^2 can be precomputed and cached, especially beneficial when the same unembedding matrix is used across multiple forward passes or when implementing weight tying between embedding and unembedding layers.

10.3 Memory Layout

Careful attention to tensor broadcasting patterns ensures memory-efficient computation without creating unnecessary intermediate tensors. The use of `clamp_min` and native precision operations maximizes GPU efficiency.

References

David D. Baek, Ziming Liu, Riya Tyagi, and Max Tegmark. Harmonic loss trains interpretable ai models. *arXiv preprint arXiv:2502.01628v2*, 2025.