Theoretical Proposal: Hyperbolic Gated Delta Networks (HGDN) - Ultra-big memory of hyperbolic space

Vuk Rosić
vukrosic

1. Abstract

We propose **Hyperbolic Gated Delta Networks (HGDN)**, a theoretical Linear Transformer architecture designed to integrate non-Euclidean geometry with hardware-efficient parallel training. By reformulating the Gated Delta Rule as a sequence of manifold-constrained updates on a product of Poincaré balls, HGDN is theorized to exploit the exponential volume of hyperbolic space to eliminate memory collisions in ultra-long contexts.

2. The Core Intuition

Language and logic are inherently hierarchical (animal $\rightarrow$ mammal $\rightarrow$ dog), often branching like a tree. Euclidean space (used in Mamba2/DeltaNet) is "flat" and struggles to store many branches without them overlapping and blurring (memory collisions).

Hyperbolic space is naturally curved like a saddle, where the "available room" grows exponentially as you move away from the center. HGDN treats its hidden state as a point in this saddle-shaped space. When storing a new memory (a needle in the haystack), it pushes that memory toward the "edge" of the space where there is infinite room to keep it distinct. When it needs to forget, it pulls the state back toward the center. This "radial memory management" is hypothesized to make HGDN essentially collision-free for sequences exceeding 1 million tokens.

3. Mathematical Framework

3.1 Manifold Representation: The Product Poincaré Space

Instead of a flat Euclidean matrix, the HGDN state S_t is theorized to reside on a **product manifold** $\mathcal{M}$:

$$\mathcal{M} = \underbrace{\mathbb{B}_c^{d_k} \times \mathbb{B}_c^{d_k} \times \cdots \times \mathbb{B}_c^{d_k}}_{h \text{ times}}$$

Formula Breakdown:

1. **The Poincaré Ball** ($\mathbb{B}_c^d$): Defined as the set of points $\{x \in \mathbb{R}^d : c\|x\|^2 < 1\}$.
   - x: A point (vector) representing a specific memory or hidden state value.
   - $\mathbb{R}^d$: The standard d-dimensional space we use for calculations (vector x has d dimensions).
   - c: The **Curvature**. It determines how "sharp" the saddle shape is. If $c = 0$, it's flat Euclidean space.
   - $\|x\|^2 < 1/c$: This defines the boundary. The ball has a radius of $1/\sqrt{c}$. Points cannot "leave" this radius; instead, as they approach the edge, the space itself stretches to infinity.

Concept	What it is	Role in HGDN
Hyperbolic Space	The "True Reality"	The curved, infinite-volume geometry where memories live.
Saddle / Ball	Models (Maps)	Two different ways to map the same d-dimensional reality. The **Ball** is preferred because it fits neatly into $[-1, 1]$ coordinate
State Matrix (S_t)	Group of Vectors	A collection of h vectors. Each vector is a point (a "memory") tucked into its own Poincaré ball.

Analogy: If the "Hyperbolic Space" is the Earth, the **Ball** and the **Saddle** are just two different map projections (like Mercator vs. Globe). The **Vectors** (your hidden state) are the actual cities pinned on those maps. Everything here—the space, the maps, and the vectors—has d **dimensions**.

2. **The Product Manifold** ($\mathcal{M}$):
   - $\times$ **(Cartesian Product)**: This means "concatenation of independent spaces."
   - d_k: The dimensionality of each individual head (the "key/value dimension").
   - h **(The number of balls)**: Corresponds to the number of value-heads in the Linear Transformer.
     - **Why separate balls?**: One hyperbolic space is essentially one **tree**. By giving each head its own ball, we allow the model to learn a **"Forest of Trees."** This prevents a hierarchy in one head (like grammar rules) from colliding or interfering with a hierarchy in another head (like factual relationships). It ensures that each head can specialize in its own independent "semantic branch."

- **The Infinite Boundary**: While the ball looks bounded to our Euclidean eyes, the hyperbolic distance to the boundary is actually infinite. This provides a "bottomless" storage area; as we push memories toward the shell, they become mathematically farther apart from each other, even if they appear close in Euclidean coordinates.
- **Product Domain Advantage**: By treating each **head** (which corresponds to a row s_i in the state matrix) as an independent point in its own ball, we allow the model to learn a **high-dimensional product of trees**. This means HGDN doesn't just store one hierarchy, but hundreds of overlapping, independent hierarchies (e.g., one head for syntax, one for semantic clusters, one for temporal ordering).
- **Geometric Retrieval**: In standard Euclidean attention, deep branches of a tree "crowd" together because the space is too small. This causes **Branch Interference**: a query for a specific detail (a leaf) accidentally has high similarity with an unrelated detail in a neighboring branch, leading to a "blurry" or mixed retrieval.
  - **In HGDN**: The standard dot-product is conceptually replaced by seeking the point on the manifold that minimizes the **geodesic distance**. Because hyperbolic space expands exponentially at the edges, the mathematical "gap" between unrelated branches is massive. This ensures the model pulls exactly the right "needle" without any signal leakage from neighboring branches.

3.2 Tangent Space Parallel Training (TSPT)

The Problem: The "Associativity Gap"

In standard Linear Transformers (like Mamba or DeltaNet), we use an **Associative Scan** to calculate the hidden state for a million tokens in parallel.

What is an Associative Scan?
Normally, a memory S_t is calculated one step at a time: $S_t = S_{t-1} + \text{new memory}$. This is slow ($O(L)$).
An **Associative Scan** is a parallel algorithm that calculates everything in a **tree structure**:

1. **Round 1**: Calculate $(1+2), (3+4), (5+6), (7+8)$ all at once.
2. **Round 2**: Combine those results: $((1+2)+(3+4))$ and $((5+6)+(7+8))$.
3. **Round 3**: Combine the final two chunks into the grand total.

Instead of 8 sequential steps, it finishes in just 3 "rounds." This logarithmic speed-up is what allows Linear Transformers to process 1 million tokens while staying faster than standard Attention. This only works if your "addition" is **associative** ($A + B + C = (A + B) + C$).

Hyperbolic space is NOT associative. If you try to add memories directly on a curve, the order matters too much (rotating a globe 90° North then 90° East is different from 90° East then 90° North). This breaks the parallel scan. Without TSPT, the model would be 100× slower to train.

The Geometric Logic: Tangent Planes

To fix this, we use a "Locally Euclidean" trick. Imagine a large globe (the hyperbolic ball). If you zoom in on one tiny patch, that patch looks flat (like a piece of paper touching the globe). This flat paper is the **Tangent Space**.

TSPT works by "flattening" a chunk of data onto this paper, doing the fast math there, and then "wrapping" the result back onto the globe.

Step-by-Step Breakdown:

For each sequence chunk (e.g., 2048 tokens), we do the following:

1. **Projection (The "Log" Map)**:

$$K_{tan} = \text{Log}^{c}_{S_{[0]}}(K), \quad Q_{tan} = \text{Log}^{c}_{S_{[0]}}(Q)$$

   - $S_{[0]}$: The starting state (the "anchor point") for this chunk.
   - $\textbf{Log}^{c}_{S_{[0]}}$: The **Logarithm Map**. It "unrolls" the hyperbolic curve into a flat Euclidean plane centered at $S_{[0]}$.
   - K_{tan}, Q_{tan}: The Keys and Queries now live in a flat space where standard addition works.

2. **Euclidean Scan (The "Parallel Fast-Forward")**:

$$\Delta S_{tan} = \text{AssociativeScan}(Q_{tan}, K_{tan}, V)$$

   - We perform the standard Gated Delta Rule update. Because we are in the flat Tangent Space, we can use optimized GPU kernels (like Flash-Linear-Attention) to process the whole chunk in $O(L)$ time.
   - ΔS_{tan}: The total "movement" or memory update calculated in flat space.

3. **Manifold Mapping (The "Exp" Map)**:

$$S_{[end]} = \text{Exp}^{c}_{S_{[0]}}(\Delta S_{tan})$$

   - $\textbf{Exp}^{c}_{S_{[0]}}$: The **Exponential Map**. It takes the flat result ΔS_{tan} and "wraps" it back onto the hyperbolic manifold.

- $S_{[end]}$: The final, valid hyperbolic state that becomes the starting point for the next chunk.

Why this is a breakthrough: It allows us to keep the massive memory capacity of Hyperbolic space while keeping the 10× training speed of Euclidean models.

3.3 Stability: Hyperbolic Spectral Normalization (HSN)

To prevent the "Boundary Catastrophe" (numerical overflow as $\|x\| \to 1$), we propose a radial contraction after each state update:

$$S_t \leftarrow \tanh\left(\frac{R_{max}}{2} \cdot \frac{S_t}{\|S_t\|}\right)$$

This is intended to ensure the state matrix remains within a stable disk $\|x\| \leq 0.99$, guaranteeing differentiable gradients and training stability in FP16/BF16.

4. Summary

HGDN offers a novel synthesis of non-Euclidean geometry and parallel processing. By leveraging the exponential volume of hyperbolic space, it provides a theoretically collision-free memory for ultra-long contexts, while Tangent Space Parallel Training (TSPT) ensures it remains as fast as standard Linear Transformers on modern hardware.