Hyperbolic ViT on CIFAR-100: progressive ablation

September 8, 2025

Federico Forner

Abstract

Studiamo un ViT euclideo "tiny" su CIFAR-100 iperbolicizzando progressivamente i moduli (head, positional embeddings, residual, MLP, self-attention). Valutiamo Top-1/Top-5, costo computazionale (img/s, tempi), stabilità e curvature apprese. Il residual iperbolico con centro apprendibile fornisce il salto principale (+4pt Top-1 sulla baseline) a parità di parametri, al costo di training molto più lento. Discutiamo metodi originali come residual e attention, limiti (memoria, stabilità) e quando conviene applicare moduli iperbolici.

1. Introduction and Method

Obiettivo. Vogliamo studiare come cambiano le prestazioni di un ViT euclideo "tiny" quando introduciamo componenti iperboliche in modo progressivo. La motivazione è che la geometria iperbolica può rappresentare strutture gerarchiche e offre flessibilità: se il caso euclideo è più adatto, la curvatura appresa può tendere a 0.

Dataset e setup. Usiamo CIFAR-100 (100 classi), scelto perché "piatto" e quindi non ci aspettiamo un vantaggio automatico dalla sola head iperbolica. La baseline è un ViT tiny standard (12 blocchi, 3 head, embed dim 192, patch size 4), interamente euclideo. Le metriche principali sono Top-1/Top-5, gap (Top-1 $-$ Train acc), stabilità numerica, throughput (img/s) e tempo.

Pipeline. Applichiamo una ablation progressiva: Head $\rightarrow$ Positional Embeddings $\rightarrow$ Residual $\rightarrow$ Linear/MLP $\rightarrow$ Self-Attention. L'ordine serve a isolare l'effetto di ciascun blocco mantenendo stabilità. Per l'attenzione usiamo fine-tuning (vedi sotto).

Email: Federico Forner <forner.2154957@studenti.uniroma1.it>.

Machine Learning 2025, Sapienza University of Rome, 2nd semester a.y. 2024/2025.

Vincoli computazionali. L'addestramento è su Kaggle (P100 16GB). Per le fasi pesanti (Residual/Linear/Attention) usiamo gradient accumulation per evitare OOM. L'attenzione iperbolica è ottimizzata in due fasi: 10 epoche "attention-only" partendo dalla variante con Hyperbolic Linear, poi 10 epoche con tutto sbloccato (fine-tuning).

Nota. Nella letteratura sulle reti iperboliche esistono varie impostazioni. Qui manteniamo coerenza geometrica e introduciamo elementi originali (es. centri apprendibili nel residual e nell'attenzione). L'unico componente rimasto euclideo è il LayerNorm (mancano soluzioni iperboliche soddisfacenti).

Head iperbolica. Le classi sono prototipi nel ball di Poincaré; i logits sono $-d_{\mathbb{H}}(x, c)^2/(2\sigma^2) + b$, con σ apprendibile e curvatura dedicata. $d_{\mathbb{H}}$ è la distanza iperbolica.

Positional embedding iperbolici. Le posizioni sono learnable nel manifold; il token e la posizione si combinano con somma di Möbius (scriviamo $\oplus$, moltiplicazione scalare $\otimes$).

Residual iperbolico. Introdotto un centro $p = s \otimes x \oplus (1 - s) \otimes f(x)$ (con s apprendibile) e poi si calcola $x \oplus \gamma \otimes f(x)$ ma "centrando" l'operazione in p. In pratica, non sommiamo nell'origine ma in un punto intermedio appreso, che può spostarsi verso x o verso $f(x)$. Questo dettaglio strutturale originale porta qualche punto di Top-1 in più (a costo computazionale maggiore).

MLP iperbolico. Usiamo una HyperbolicLinear con due centri learnable distinti p_{in} e p_{out}:

2. Results and Discussion

Prima di tutto, riportiamo una tabella comparativa con i risultati nella versione in cui ho salvato i pesi, ossia quando il modello ha toccato la Top-1 più alta (oppure la Val loss più bassa). Per la Train acc considero il massimo tra i valori loggati ogni $\sim$100 iterazioni. Useremo le abbreviazioni: **hball** = curvatura della head iperbolica, **pball** = curvatura dei positional embedding, **Gap** = Top-1 $-$ Train acc.

Hyperbolic ViT on CIFAR-100: progressive ablation

Modulo	Top-1	Top-5	Train	TrLoss	ValLoss	Gap	img/s	Time	hball	pball
All Euclidean	53.10	79.04	96.09	0.1958	2.5203	42.99	1327.8	1h07m	—	—
Hyp. Head	48.63	75.79	92.97	0.2688	3.3771	44.34	1188.8	1h15m	1.33	—
Hyp. Pos. Embeddings	45.91	73.52	91.41	0.3832	3.3297	45.50	1137.6	1h19m	1.36	1.71
Hyp. Residual	57.39	82.99	96.88	0.1980	2.6093	39.49	194.1	7h36m	0.98	1.91
Hyp. Linear (and MLP)	53.92	82.23	75.00	1.2199	1.7604	21.08	60.1	23h44m	0.82	2.48
Hyp. Self Attention	45.76	76.94	68.75	1.9513	2.0386	22.99	12.4	21h05m	0.81	2.48

Table 1. Confronto tra varianti euclidee e iperboliche. **hball** = curvatura head; **pball** = curvatura positional; **Gap** = Top-1 - Train acc.

Varianti (head e residual). Head senza clipping: leggero aumento di accuracy (es. Val acc1 $\approx$ 50.75, acc5 $\approx$ 78.08; Val loss $\approx$ 3.3811), segno che il modello usa anche punti vicini al bordo del ball. Residual senza centro learnable (10 epoche): più veloce ma peggiore. Es.: *senza centro* Ep10: loss 2.5326, acc1 36.20, acc5 67.92, img/s 334.2; *con centro* Ep10: 2.3543, 38.34, 70.99, 195.1. A parità di parametri, la struttura conta.

Costo computazionale. Il throughput (img/s) scende e la durata sale con più "quota" iperbolica; dal residual in poi l'aumento è marcato, fino a diventare quasi insostenibile con l'attenzione. Conviene iperbolicizzare porzioni mirate e soprattutto in dataset con gerarchie. Possibile compromesso futuro: approssimare operazioni (exp/log/add/dist) per guadagnare velocità con lieve calo di accuracy.

Memoria e stabilità. Le operazioni iperboliche consumano più memoria; negli early stage ho disabilitato l'autocast per stabilità. Per evitare OOM ho usato gradient accumulation (più lento).

Accuratezza e gap. All'inizio l'accuracy cala (53.1 $\rightarrow$ 48.6 $\rightarrow$ 45.9), poi con il residual sale a 57.4 (+11 punti sullo stadio precedente, $\sim$+4 sulla baseline). Il solo tempo di training non spiega tutto: la geometria aiuta a organizzare meglio token/classi. I parametri sono quasi uguali: il boost è strutturale. I moduli pesanti tendono a dare il meglio più tardi (es. residual all'ultima epoca), quindi più epoche/tuning aiuterebbero. Il gap (Train vs Val) diminuisce dal residual in poi: meno overfitting. Linear e Attention "sotto tono" suggeriscono iperparametri non ottimali e training più breve (train loss alta $\Rightarrow$ underfitting).

Curvature. Le curvature apprese non sono vicine a 0 (euclideo), quindi la geometria è usata davvero. Trend: pball cresce (circa 1.7 $\rightarrow$ 2.5), hball cala (circa 1.3 $\rightarrow$ 0.8) quando aumentano i moduli iperbolici: con molte operazioni nel manifold i token sono già ben distribuiti e la head richiede meno curvatura.

Nota sull'attenzione. Solo 20 epoche di fine-tuning (partendo da Linear) per vincoli computazionali: risultati da prendere con cautela. Centroidi globali condivisi tra head/layer per ridurre parametri e concentrare il gradiente; non ho testato centroidi separati.

3. Conclusions

Strutture e architetture contano: dando più libertà geometrica (senza gonfiare i parametri) si può migliorare di non poco. Nel mio caso, su un dataset non gerarchico come CIFAR-100, ho superato la baseline euclidea di oltre 4 punti in Top-1 grazie ai moduli iperbolici, pagando però i ben noti costi di memoria/stabilità e soprattutto di velocità. Vale la pena usarli quando ci sono gerarchie e solo su pezzi mirati (nel mio studio, lo skip/residual è risultato particolarmente efficace), più che iperbolicizzare tutto indiscriminatamente.

A mia conoscenza, nessun lavoro su ViT iperbolici ha adottato un residual centrato in un *barycenter* adattivo p (calcolato come combinazione learnable tra x e $f(x)$) su cui poi effettuare la combinazione residua. Il *centro apprendibile* nel residual — inteso come barycenter tra x e $f(x)$ che *centra* la combinazione sul manifold — si è rivelato decisivo: a parità di capacità, sposta l'optimum consentendo combinazioni più espressive e stabili, traducendosi nel maggior guadagno osservato in Top-1.

References

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2020.

Ermolov, A., Mirvakhabova, L., Khrulkov, V., Sebe, N., and Oseledets, I. Hyperbolic vision transformers: Combining improvements in metric learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Ganea, O.-E., Bécigneul, G., and Hofmann, T. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Google Research. Vision transformer: Official repository.

https://github.com/google-research/vision_transformer, 2020.

Gülçehre, Ç., Denil, M., Malinowski, M., Razavi, A., Pascanu, R., Hadsell, R., and Lillicrap, T. Hyperbolic attention networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Kochurov, M. and contributors. Geoopt documentation: Stereographic math (poincaré ball) and Möbius operations. https://geoopt.readthedocs.io/en/latest/_modules/geoopt/manifolds/stereographic/math.html, 2023.

Kochurov, M., Karimov, R., and Kozlukov, S. Geoopt: Riemannian optimization in pytorch. https://arxiv.org/abs/2005.02819, 2020.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

PyTorch Contributors. Pytorch documentation: torch.nn.softplus. https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html, 2025.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. DOI: 10.5281/zenodo.4414861.