## A. Hyperbolic Neural Networks

**Linear layer.** Assume we have a standard (Euclidean) linear layer $\mathbf{x} \to \mathrm{M}\mathbf{x} + \mathbf{b}$. In order to generalize it, one needs to define the Möbius matrix by vector product:

$$\mathrm{M}^{\otimes_c}(\mathbf{x}) \coloneqq \frac{1}{\sqrt{c}} \tanh\left(\frac{\|\mathrm{M}\mathbf{x}\|}{\|\mathbf{x}\|}\operatorname{arctanh}(\sqrt{c}\|\mathbf{x}\|)\right)\frac{\mathrm{M}\mathbf{x}}{\|\mathrm{M}\mathbf{x}\|}, \tag{13}$$

if $\mathrm{M}\mathbf{x} \neq \mathbf{0}$, and $\mathrm{M}^{\otimes_c}(\mathbf{x}) \coloneqq \mathbf{0}$ otherwise. Finally, for a bias vector $\mathbf{b} \in \mathbb{D}_c^n$ the operation underlying the hyperbolic linear layer is then given by $\mathrm{M}^{\otimes_c}(\mathbf{x}) \oplus_c \mathbf{b}$.

**Concatenation of input vectors.** In several architectures (e.g., in siamese networks), it is needed to concatenate two vectors; such operation is obvious in Euclidean space. However, straightforward concatenation of two vectors from hyperbolic space does not necessarily remain in hyperbolic space. Thus, we have to use a generalized version of the concatenation operation, which is then defined in the following manner. For $\mathbf{x} \in \mathbb{D}_c^{n_1}$, $\mathbf{y} \in \mathbb{D}_c^{n_2}$ we define the mapping $\mathrm{Concat} : \mathbb{D}_c^{n_1} \times \mathbb{D}_c^{n_2} \to \mathbb{D}_c^{n_3}$ as follows.

$$\mathrm{Concat}(\mathbf{x}, \mathbf{y}) = \mathrm{M}_1^{\otimes_c}\mathbf{x} \oplus_c \mathrm{M}_2^{\otimes_c}\mathbf{y}, \tag{14}$$

where $\mathrm{M}_1$ and $\mathrm{M}_2$ are trainable matrices of sizes $n_3 \times n_1$ and $n_3 \times n_2$ correspondingly. The motivation for this definition is simple: usually, the Euclidean concatenation layer is followed by a linear map, which when written explicitly takes the (Euclidean) form of Equation (14).

**Multiclass logistic regression (MLR).** In our experiments, to perform the multiclass classification, we take advantage of the generalization of multiclass logistic regression to hyperbolic spaces. The idea of this generalization is based on the observation that in Euclidean space logits can be represented as the distances to certain *hyperplanes*, where each hyperplane can be specified with a point of origin and a normal vector. The same construction can be used in the Poincaré ball after a suitable analogue for hyperplanes is introduced. Given $\mathbf{p} \in \mathbb{D}_c^n$ and $\mathbf{a} \in T_{\mathbf{p}}\mathbb{D}_c^n \setminus \{\mathbf{0}\}$, such an analogue would be the union of all geodesics passing through $\mathbf{p}$ and orthogonal to $\mathbf{a}$.

The resulting formula for hyperbolic MLR for $K$ classes is written below; here $\mathbf{p}_k \in \mathbb{D}_c^n$ and $\mathbf{a}_k \in T_{\mathbf{p}_k}\mathbb{D}_c^n \setminus \{\mathbf{0}\}$ are learnable parameters.

$$p(y = k|\mathbf{x}) \propto \exp\left(\frac{\lambda_{\mathbf{p}_k}^c\|\mathbf{a}_k\|}{\sqrt{c}}\operatorname{arcsinh}\left(\frac{2\sqrt{c}\langle -\mathbf{p}_k \oplus_c \mathbf{x}, \mathbf{a}_k\rangle}{(1 - c\| - \mathbf{p}_k \oplus_c \mathbf{x}\|^2)\|\mathbf{a}_k\|}\right)\right).$$

For a more thorough discussion of hyperbolic neural networks, we refer the reader to the paper [11].

## B. Experiment details

**Omniglot.** As a baseline model, we consider the prototype network (ProtoNet). Each convolutional block consists of $3 \times 3$ convolutional layer followed by batch normalization, ReLU nonlinearity and $2 \times 2$ max-pooling layer. The number of filters in the last convolutional layer corresponds to the value of the embedding dimension, for which we choose 64. The hyperbolic model differs from the baseline in the following aspects. First, the output of the last convolutional block is embedded into the Poincaré ball of dimension 64 using the exponential map. Results are presented in Table 7. We can see that in some scenarios, in particular for one-shot learning, hyperbolic embeddings are more beneficial, while in other cases, results are slightly worse. The relative simplicity of this dataset may explain why we have not observed a significant benefit of hyperbolic embeddings. We further test our approach on more advanced datasets.

Table 7: Few-shot classification accuracies on Omniglot. In order to obtain Hyperbolic ProtoNet, we augment the standard ProtoNet with a mapping to the Poincaré ball, use hyperbolic distance as the distance function, and as the averaging operator we use the HypAve operator defined by Equation (10).

| | ProtoNet | Hyperbolic ProtoNet |
|---|---|---|
| 1-shot 5-way | 98.2 | **99.0** |
| 5-shot 5-way | 99.4 | 99.4 |
| 1-shot 20-way | 95.8 | **95.9** |
| 5-shot 20-way | **98.6** | 98.15 |

***mini*ImageNet.** We performed the experiments with two different backbones, namely the previously discussed 4-Conv model and ResNet18. For the former, embedding dim was set to 1024 and for the latter to 512. For the one-shot setting both models were trained for 200 epochs with Adam optimizer, learning rate being $5 \cdot 10^{-3}$ and step learning rate decay with the factor of 0.5 and step size being 80 epochs. For the 4-Conv model we used $c = 0.01$ and for ResNet18 we used $c = 0.001$. For 4-Conv in the five-shot setting we used the same hyperparameters except for $c = 0.005$ and learning rate decay step being 60 epochs. For ResNet18 we additionally changed learning rate to $10^{-3}$ and step size to 40.

**Caltech-UCSD Birds.** For these experiments we used the same 4-Conv architecture with the embedding dimensionality being 512. For the one-shot task, we used learning rate $10^{-3}$, $c = 0.05$, learning rate step being 50 epochs and decay rate of 0.8. For the five-shot task, we used learning rate

$10^{-3}$, $c = 0.01$, learning rate step of $40$ and decay rate of $0.8$.

**Person re-identification.** We use ResNet50 [14] architecture with one fully connected embedding layer following the global average pooling. Three embedding dimensionalities are used in our experiments: $32$, $64$ and $128$. For the baseline experiments, we add the additional classification linear layer, followed by the cross-entropy loss. For the hyperbolic version of the experiments, we map the descriptors to the Poincaré ball and apply multiclass logistic regression as described in Section 4. We found that in both cases the results are very sensitive to the learning rate schedules. We tried four schedules for learning $32$-dimensional descriptors for both baseline and hyperbolic versions. The two best performing schedules were applied for the $64$ and $128$-dimensional descriptors. In these experiments, we also found that smaller $c$ values give better results. We therefore have set $c$ to $10^{-5}$. Based on the discussion in 4, our hyperbolic setting is quite close to Euclidean. The results are compiled in Table 6. We set starting learning rates to $3 \cdot 10^{-4}$ and $6 \cdot 10^{-4}$ for $sch\#1$ and $sch\#2$ correspondingly and multiply them by $0.1$ after each of the epochs $200$ and $270$.

## C. Visualizations

For the visual inspection of embeddings we computed projections of high dimensional embeddings obtained from the trained few–shot models with the (hyperbolic) UMAP algorithm [26] (see Figure 6). We observe that different classes are neatly positioned near the boundary of the circle and are well separated.

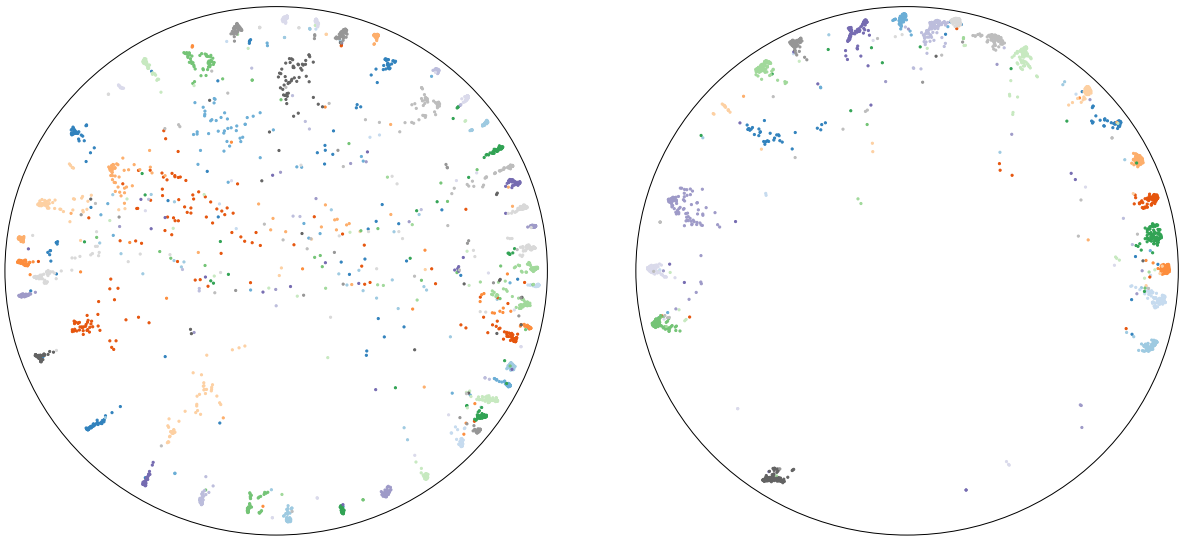

Figure 6: A visualization of the hyperbolic embeddings learned for the few–shot task. **Left**: 5-shot task on CUB. **Right**: 5-shot task on *Mini*ImageNet. The two-dimensional projection was computed with the UMAP algorithm [26].