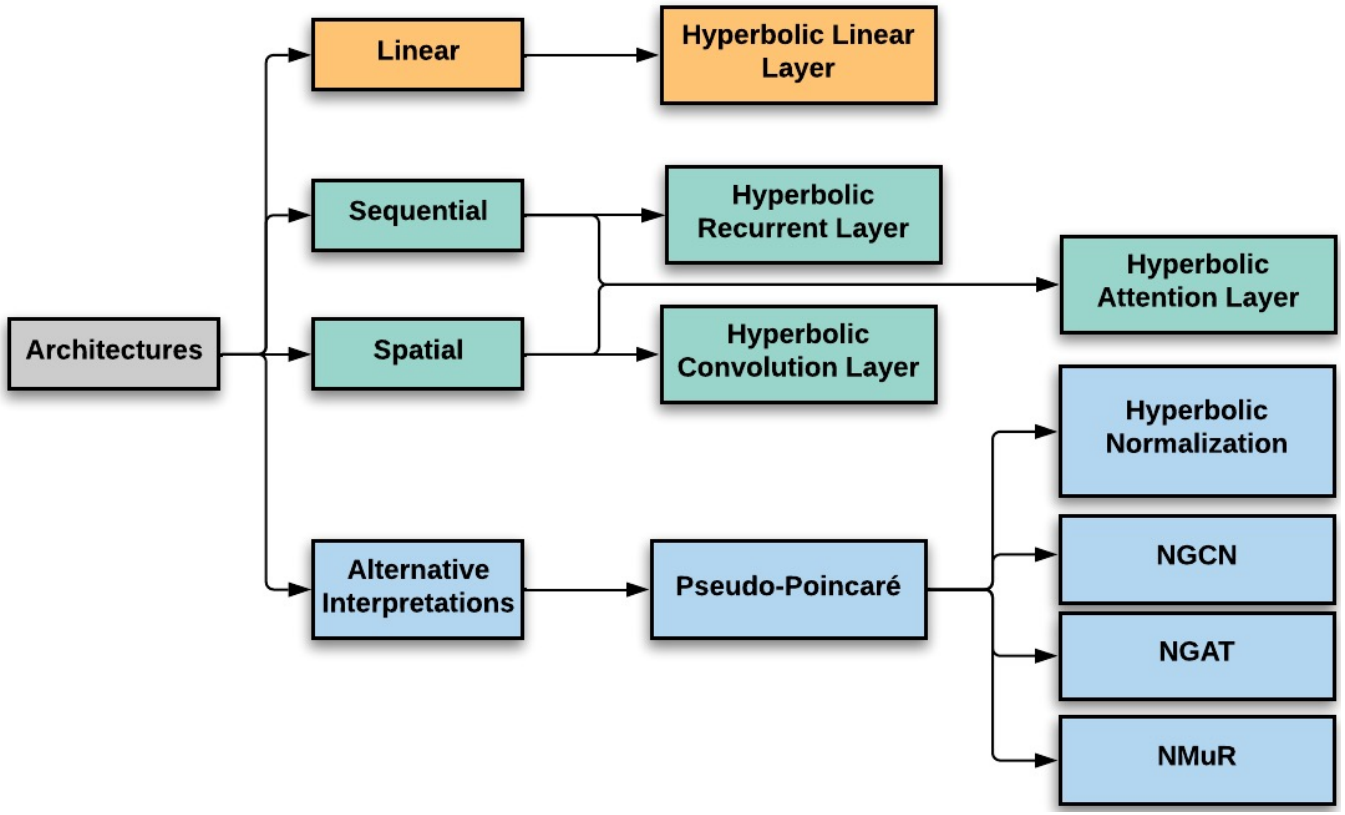
Part 3: Architectures



Requirements
Development of Neural Networks

Functional requirements for adopting a space into neural networks:

- ❑ Addition (by extension subtraction)
- ❑ Scalar Multiplication
- ❑ Matrix Multiplication
- ❑ Distance Metric
- ❑ Applying Function
- ❑ SoftMax - MLR

Hyperbolic Operations

Development of Neural Networks

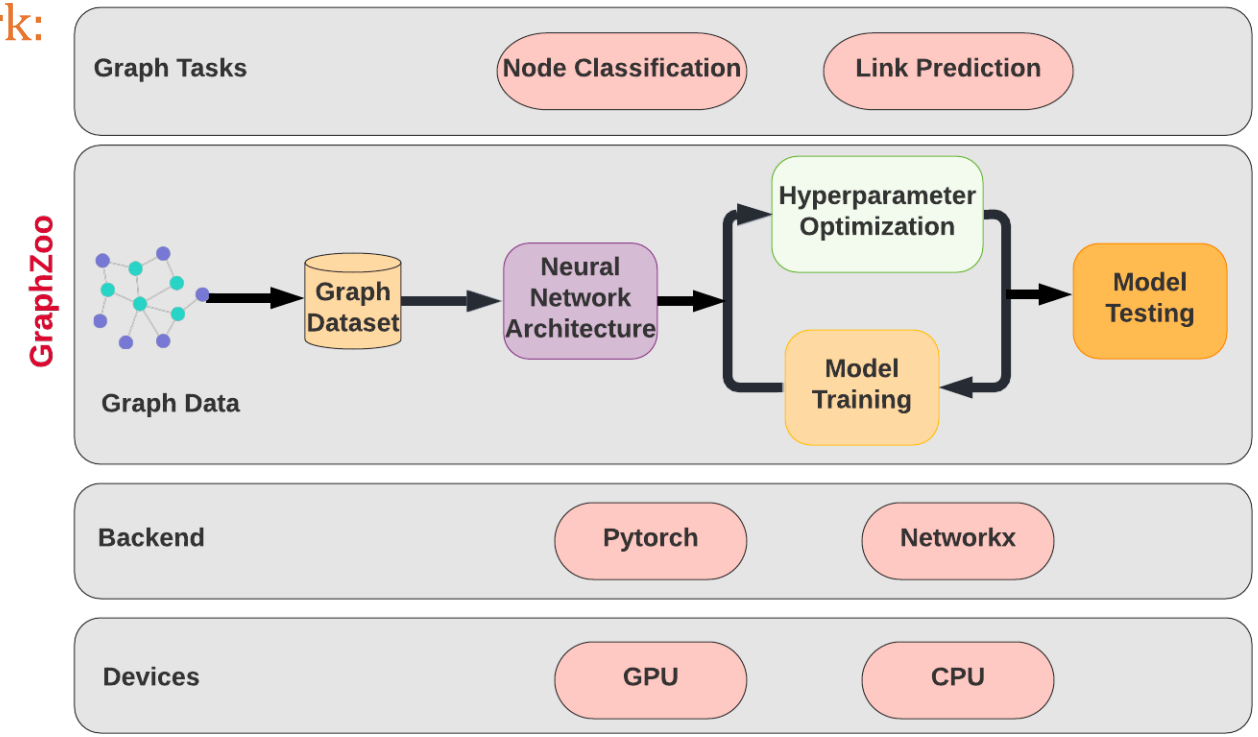
Functional requirements for adopting a space into neural networks:

- ✓ Addition (by extension subtraction) $$x \oplus_c a = \frac{(1 + 2ca \cdot x + c \parallel x \parallel^2)a + (1 - c \parallel a \parallel^2)x}{1 + 2ca \cdot x + c^2 \parallel a \parallel^2 \parallel x \parallel^2}$$
- ✓ Scalar Multiplication $$r \otimes_c x = exp_0^c(r log_0^c(x))$$
- ✓ Matrix Multiplication $$[M_1 \quad M_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = M_1 \otimes_c x_1 \oplus_c M_2 \otimes_c x_2$$
- ✓ Distance Metric $$d_{\mathbb{H}}(x, y) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c} \parallel -x \oplus_c y \parallel)$$
- ✓ Applying Function $$f^{\otimes_c}(x) = exp_0^c(f(Log_0^c(x)))$$
- ✓ SoftMax - MLR $$P(y = k|x) \propto exp\left(\frac{\lambda_{p_k}^c \|a_k\|}{\sqrt{c}} \sinh^{-1}\left(\frac{2\sqrt{c}\langle -p_k \oplus_c x, a_k \rangle}{(1 - c\|-p_k \oplus_c x\|^2)\|a_k\|}\right)\right)$$

Hyperbolic Neural Networks
GraphZoo Toolkit

Supports designing a hyperbolic neural network:

- Design a Manifold
- Activation Functions
- Layers
- Optimizers
- Loss function
- Model Architecture (based on task)



Link to Toolkit: https://github.com/reddy-lab/GraphZoo
(Implemented in Pytorch)

Hyperbolic Neural Networks

GraphZoo Toolkit

In this part of the tutorial, we will understand and show the implementation of the following parts:

- Manifolds; Poincaré Ball and Lorentz (same as Klein)
- Activation Functions; General Hyperbolic activation
- Loss function; Hyperbolic distance
- Optimizers; Reimannian SGD and Reimannian AdaM
- Layers; Linear, Recurrent, Convolution and Attention
- Model Architecture; Combine to form a full-fledged model

Hyperbolic Neural Networks

Designing the Manifold

ManifoldParameter

1. Defines the manifold type
2. Defines the curvature

Exponential and Logarithmic Maps

1. Defines the function of exponential map from tangential space to manifold.
2. Defines the function of logarithmic map from manifold to tangential space at a point.

Operations

1. Defines the manifold operations
2. The manifold operations include addition, scalar product, matrix vector product.

Distance

1. Defines the function of hyperbolic distance

Projections

1. Defines the projection operation to maintain the range of hyperbolic space.

Parallel Transport

1. Defines the function of parallel transport from one tangential space to another.

Egrad2Rgrad

1. Defines the function of transforming Euclidean gradient to Riemannian gradient.

Hyperbolic Neural Networks

Euclidean Model: Designing the Manifold

Distance

$$d_{\mathbb{R}}(x, y) = \| y - x \|$$

Operations

$$x \oplus_c y = x + y$$

$$[M_1 \quad M_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = M_1 x_1 + M_2 x_2$$

Parallel Transport

$$P^c_{0 \to x}(v) = x + v$$

Exponential and Logarithmic Maps

$$log^c_p(y) = y - p$$

$$exp^c_p(v) = p + v$$

Egrad2Rgrad

$$\delta_{\mathbb{R}}(p) = p$$

Hyperbolic Neural Networks

Poincaré Ball Model: Designing the Manifold

Distance

$$d_{\mathbb{H}}(x,y) = \frac{2}{\sqrt{c}}\tanh^{-1}(\sqrt{c} \parallel -x \oplus_c y \parallel)$$

Operations

$$x \oplus_c y = \frac{(1 + 2cy \cdot x + c \parallel x \parallel^2)y + (1 - c \parallel y \parallel^2)x}{1 + 2cy \cdot x + c^2 \parallel y \parallel^2 \parallel x \parallel^2}$$

$$[M_1 \quad M_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = M_1 \otimes_c x_1 \oplus_c M_2 \otimes_c x_2$$

Parallel Transport

$$P^c_{0 \to x}(v) = log^c_x\left(x \oplus_c exp^c_0(v)\right) = \frac{\lambda^c_0}{\lambda^c_x} v$$

Exponential and Logarithmic Maps

$$log^c_p(y) = \frac{2}{\sqrt{c}\lambda^c_p}\tanh^{-1}(\sqrt{c} \parallel -p \oplus_c y \parallel)\frac{-p \oplus_c y}{\parallel -p \oplus_c y \parallel}$$

$$exp^c_p(v) = p \oplus_c \left(\tanh\left(\sqrt{c}\frac{\lambda^c_x \parallel v \parallel}{2}\right)\frac{v}{\sqrt{c} \parallel v \parallel}\right)$$

Egrad2Rgrad

$$\delta_{\mathbb{H}}(p) = \frac{\delta_{\mathbb{R}}(p)}{\lambda^c_x(p)^2}$$

Hyperbolic Neural Networks

Poincaré Ball Model: Designing the Manifold

Distance

```
manifold.poincare.sqdist
```

Operations

```
manifold.poincare.mobius_add
manifold.poincare.mobius_matvec
```

Parallel Transport

```
manifold.poincare.ptransp
manifold.poincare.ptrans0
```

Exponential and Logarithmic Maps

```
manifold.poincare.expmap
manifold.poincare.expmap0
manifold.poincare.logmap
manifold.poincare.logmap0
```

Egrad2Rgrad

```
manifold.poincare.egrad2rgrad
```

Hyperbolic Neural Networks

Lorentz Model: Designing the Manifold

Distance

$$d_{\mathbb{L}}(x,y) = \frac{1}{c}\left(\cosh^{-1}\left(-c(\langle x,y\rangle_{\mathbb{L}})\right)\right)^2$$

$$\langle x,y\rangle_{\mathbb{L}} = -x_0 y_0 + \sum_{i=1}^{n} x_i y_i$$

Operations

$$x \oplus_c y = Exp_x^c\left(P_{0\to x}^c(Log_0^c y)\right)$$

$$M \otimes_c x = Exp_0^c(M \times Log_0^c(x))$$

Parallel Transport

$$\alpha = \frac{\langle Log_x^c(y), u\rangle_{\mathbb{L}}}{d_{\mathbb{L}}(x,y)}$$

$$P_{x\to y}^c(u) = \mathrm{u} - \alpha\left(Log_x^c(y) + Log_y^c(x)\right)$$

Exponential and Logarithmic Maps

$$\|v\|_{\mathbb{L}}^c = \sqrt{c\langle v,v\rangle_{\mathbb{L}}}$$

$$Exp_x^c(v) = \cosh(\|v\|_{\mathbb{L}}^c)x + \sinh(\|v\|_{\mathbb{L}}^c)\frac{v}{\|v\|_{\mathbb{L}}^c}$$

$$Log_x^c(y) = d_{\mathbb{L}}(x,y)\frac{y + \langle x,y\rangle_{\mathbb{L}} \times x \times c}{\|y + \langle x,y\rangle_{\mathbb{L}} \times x \times c\|_{\mathbb{L}}^c}$$

Hyperbolic Neural Networks

Lorentz Model: Designing the Manifold

Distance

```
manifold.lorentz.sqdist
manifold.lorentz.minkowski_dot
```

Exponential and Logarithmic Maps

```
manifold.lorentz.expmap
manifold.lorentz.expmap0
manifold.lorentz.logmap
manifold.lorentz.logmap0
```

Operations

```
manifold.lorentz.mobius_add
manifold.lorentz.mobius_matvec
```

Parallel Transport

```
manifold.lorentz.ptransp
manifold.lorentz.ptrans0
```

Hyperbolic Neural Networks
Activation Function

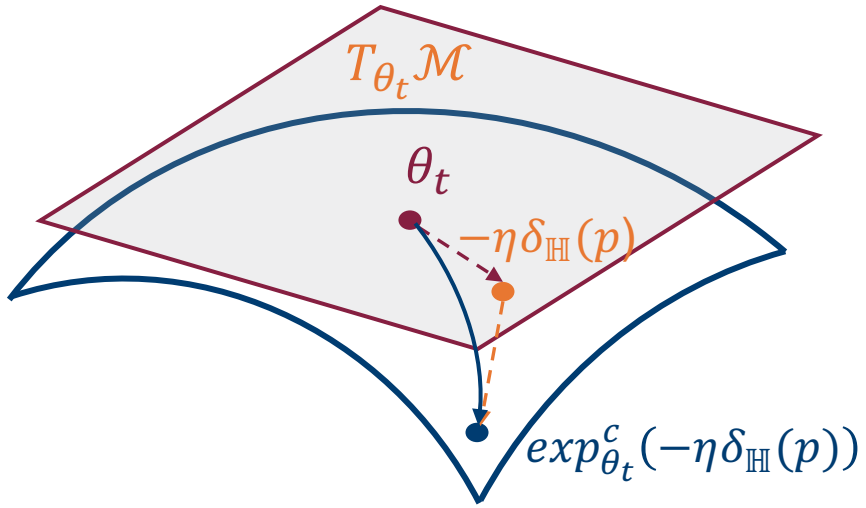
- Activation Functions; General Hyperbolic activation

Activation Function	Implementation
$f^{\otimes_c}(x) = exp_0^c(f(log_0^c(x)))$	`layers.HyperbolicActivation("poincare")` `layers.HyperbolicActivation("lorentz")`

Hyperbolic Neural Networks
Optimizer Functions

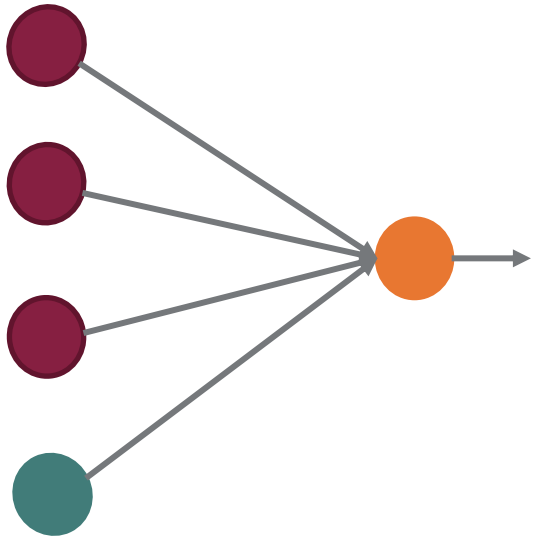
- Optimizers; RSGD and RAdaM

Optimizer Function	Implementation
$\theta_{t+1} = exp^c_{\theta_t}(-\eta\delta_{\mathbb{H}}(p))$	grad = manifold.egrad2rgrad(point, grad, c)



Hyperbolic Neural Networks

Layer: Linear Layer



Euclidean:

$w_1 x_1 + w_2 x_2 + w_3 x_3 + \mathrm{b}$

Hyperbolic: Replace with gyrovector operations

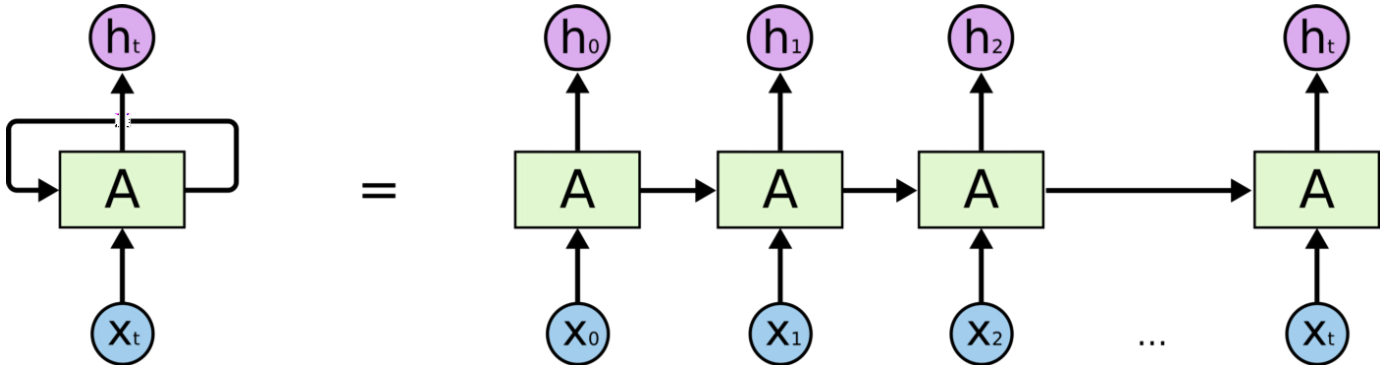
$w_1 \otimes_c x_1 \oplus_c w_2 \otimes_c x_2 \oplus_c w_3 \otimes_c x_3 \oplus_c \mathrm{b}$

Implementation
`Change "+" to manifold.mobius_add` `Change "matmul" to manifold.mobius_matvec`

$$\begin{bmatrix} w_1 & w_2 & w_3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ b \end{bmatrix}$$

Hyperbolic Neural Networks

Layer: Recurrent Layer



- Formulation is similar to Linear layer, but back-propagation happens over timesteps.
- The same replacement of operations to hyperbolic operations are needed.

Euclidean:

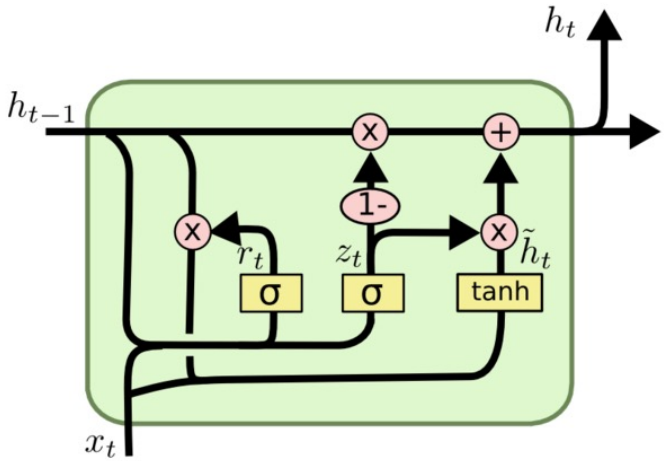
$$h_t = \phi(W h_{t-1} + U x_t + b)$$

Hyperbolic:

$$h_t = \phi^{\otimes_c}(W \otimes_c h_{t-1} \oplus_c U \otimes_c x_t \oplus_c b)$$

Hyperbolic Neural Networks

Layer: GRU Layer



Euclidean:

$$r_t = \sigma(W^r r_t + U^r x_t + b^r)$$
$$z_t = \sigma(W^z r_t + U^z x_t + b^z)$$
$$\tilde{h_t} = \phi(W(r_t \odot h_{t-1}) + U x_t + b)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t}$$

Hyperbolic:

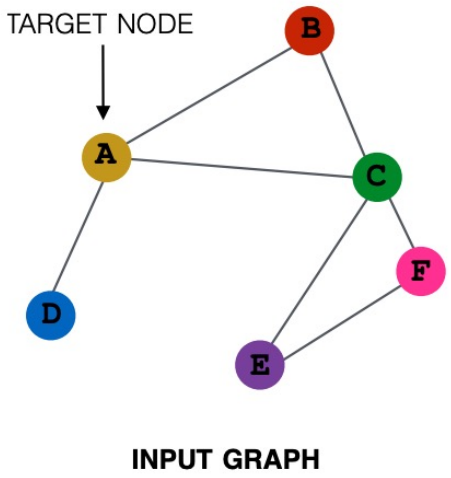
$$r_t = \sigma(log_0^c(W^r \otimes_c r_t \oplus_c U^r \otimes_c x_t \oplus_c b^r))$$
$$z_t = \sigma(log_0^c(W^z \otimes_c r_t \oplus_c U^z \otimes_c x_t \oplus_c b^z))$$

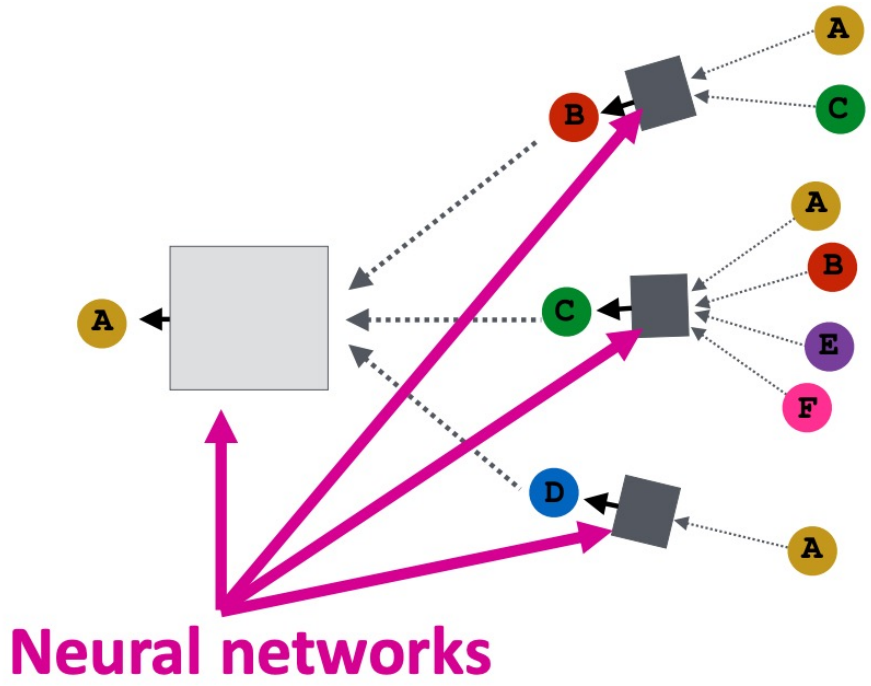
Through Möbius matrix associativity,

$$\tilde{h_t} = \phi_c^{\otimes}((W(diag(r_t)) \otimes_c h_{t-1}) \oplus_c U \otimes_c x_t \oplus_c b)$$
$$h_t = h_{t-1} \oplus_c diag(z_t) \otimes_c (-h_{t-1} \oplus_c \tilde{h_t})$$

Hyperbolic Neural Networks

Layer: Convolution Layer





Euclidean:

(feature transform)

$$h_i^{l,\mathbb{R}} = W^l x_i^{l-1,\mathbb{R}} + b^l$$

(neighborhood aggregation)

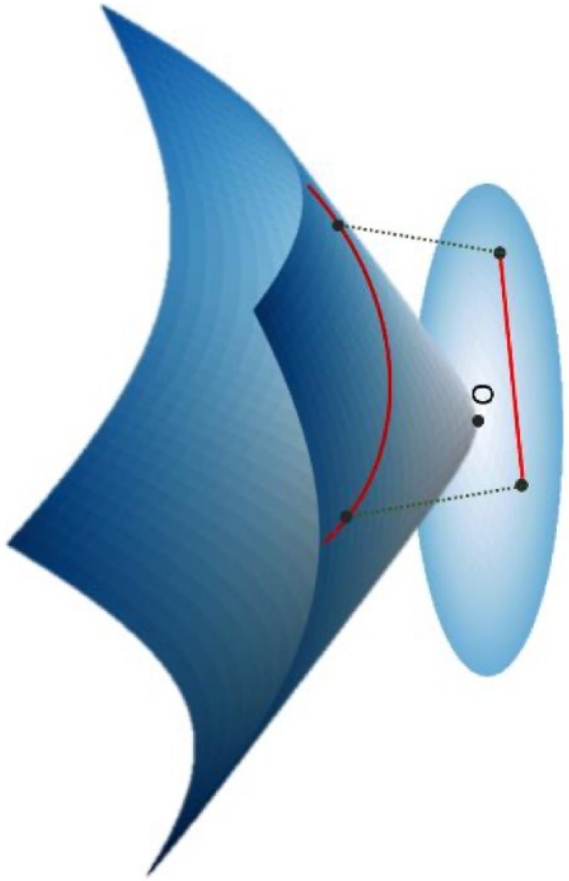
$$x_i^{l,\mathbb{R}} = \sigma\left(h_i^{l,\mathbb{R}} + \sum_{j \in \mathcal{N}(i)} w_{ij} h_j^{l,\mathbb{R}}\right)$$

Image Source: https://medium.com/neuralspace/graphs-neural-networks-in-nlp-dc475eb089de

Hyperbolic Neural Networks

Layer: Convolution Feature Transform

(feature transform)



Hyperbolic:

$$W^l \otimes_c x_i^{l-1,\mathbb{R}} = exp_0^c\left(W^l log_0^c\left(x_i^{l-1,\mathbb{R}}\right)\right)$$

Hyperbolic Transformation Notation

Map from tangent space to Poincaré Ball

Map from Poincaré Ball to tangent space

Implementation

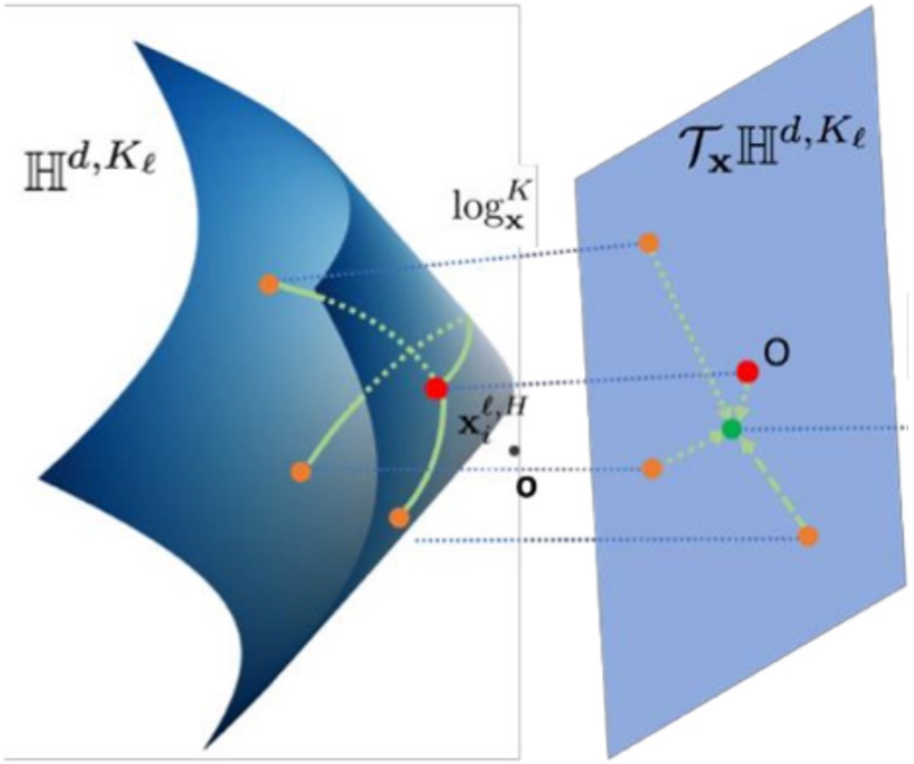
```
Change "+" to manifold.mobius_add
Change "matmul" to manifold.mobius_matvec
```

Image Source: Chami, Ines, et al. "Hyperbolic graph convolutional neural networks." Advances in neural information processing systems 32 (2019).

Hyperbolic Neural Networks

Layer: Convolution Feature Aggregation

(feature aggregation)



Hyperbolic:

$$w_{ij} = Softmax_{j\in\mathcal{N}(i)}\{MLP(log_0^c(x_i^{\mathbb{H}}) \,\|\, log_0^c(x_j^{\mathbb{H}}))\}$$

Attention of node i to node j

Concatenate in Tangent space because concatenation is not possible in hyperbolic space.

$$AGG(x^{\mathbb{H}})_i = exp_{x_i^{\mathbb{H}}}^c\left(\sum_{j\in\mathcal{N}(i)} w_{ij} log_{x_i^{\mathbb{H}}}^c(x_j^{\mathbb{H}})\right)$$

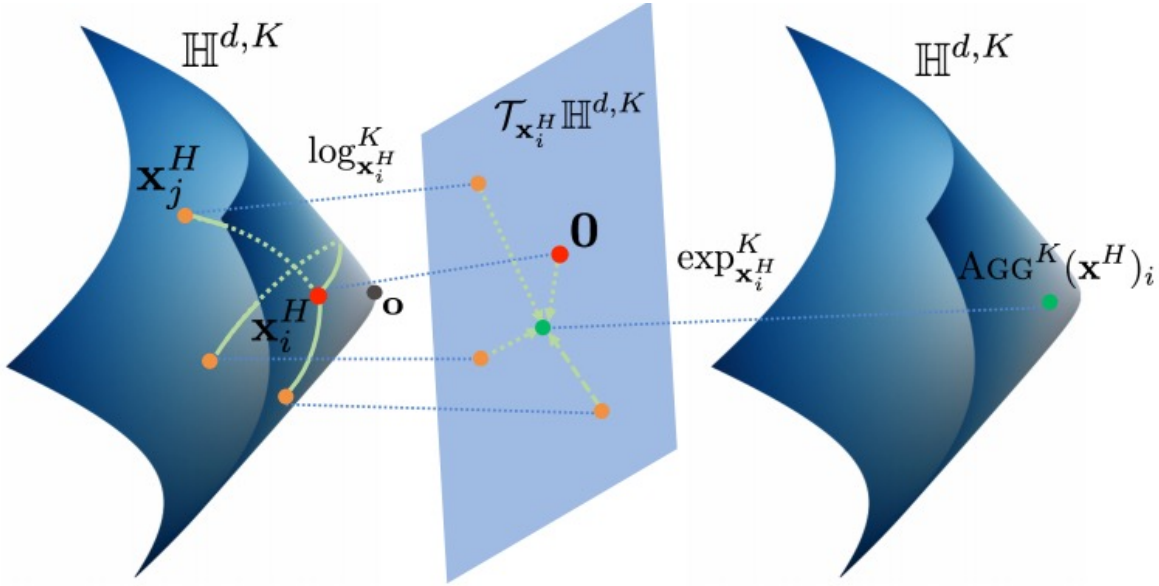
Implementation

1. `manifold.logmap0 over all inputs.`
2. `Concatenate after transformation.`
3. `Apply Linear layer and Softmax.`

Image Source: Chami, Ines, et al. "Hyperbolic graph convolutional neural networks." Advances in neural information processing systems 32 (2019).

Hyperbolic Neural Networks

Layer: Convolution Combined Pipeline



Feature Transform → Aggregate → Activation

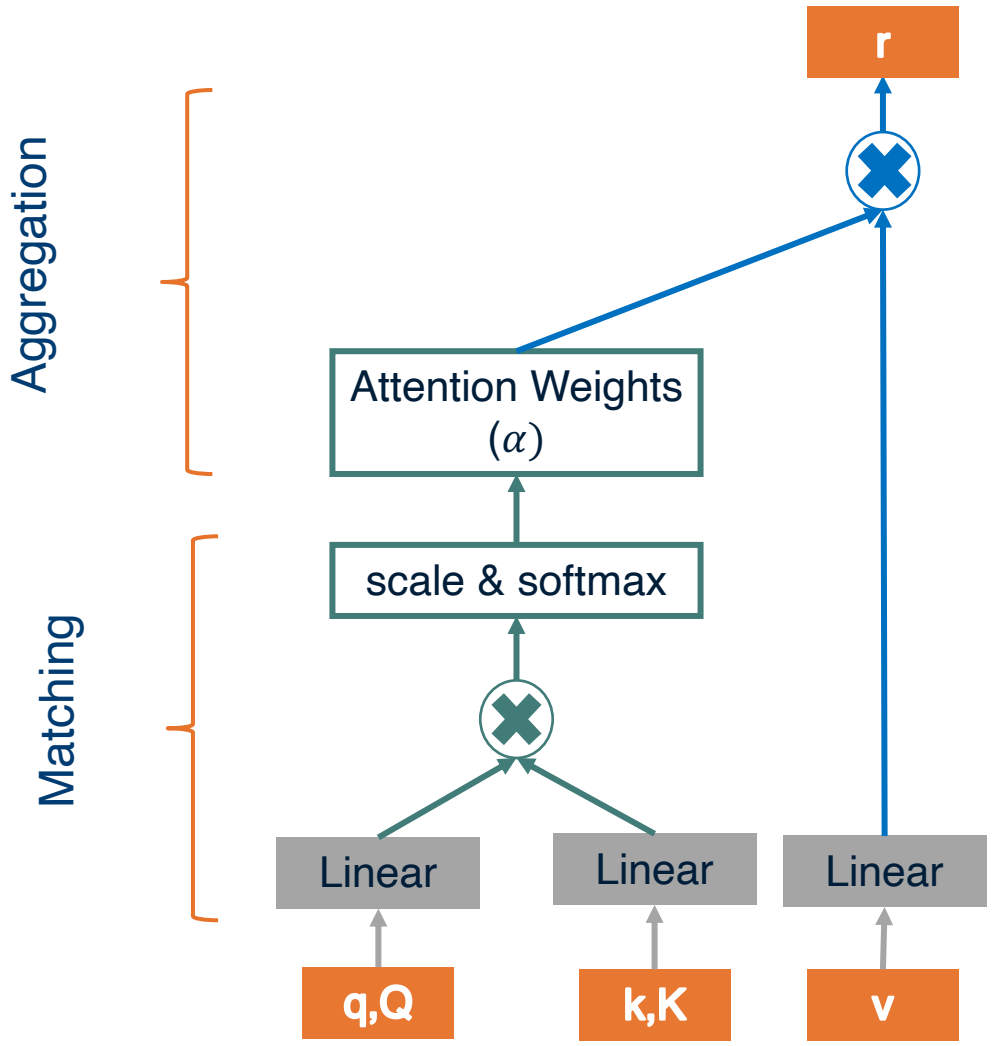
Implementation

1. Apply hyperbolic linear layer over all inputs in the hyperbolic space.
2. Aggregate all inputs with aggregation layer in tangent space at origin.
3. Transform them back to hyperbolic space and apply hyperbolic activation.

Image Source: Chami, Ines, et al. "Hyperbolic graph convolutional neural networks." Advances in neural information processing systems 32 (2019).

Hyperbolic Neural Networks

Layer: Attention Layer



Euclidean:

(matching; calculating attention weights)

$$\alpha(q_i, k_j) = \alpha_{ij} = \frac{q_i k_j}{\sqrt{d}}$$

(aggregation)

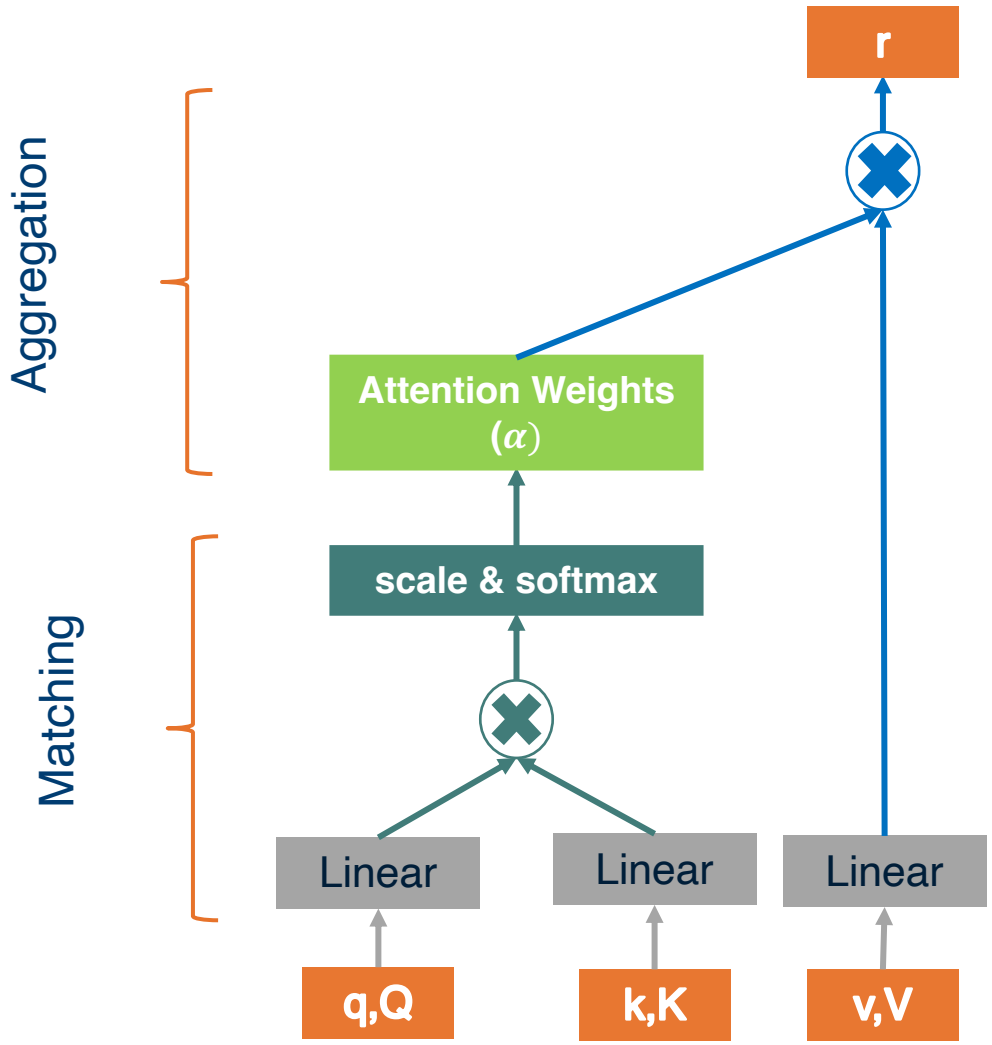
$$r_i\left(\{\alpha_{ij}\}_j, \{v_{ij}\}_j\right) = \sum_j softmax(\alpha_{ij}) v_{ij}$$

Challenge:

- No direct replacement for softmax in Poincaré ball (division replacement not defined)
- Inner product in Euclidean is used as a distance metric. Not so in hyperbolic space
- Calculating midpoint for aggregation is not well defined.

Hyperbolic Neural Networks

Layer: Attention Layer



Euclidean:

(matching; calculating attention weights)

$$\alpha(q_i, k_j) = \alpha_{ij} = \frac{q_i k_j}{\sqrt{d}}$$

(aggregation)

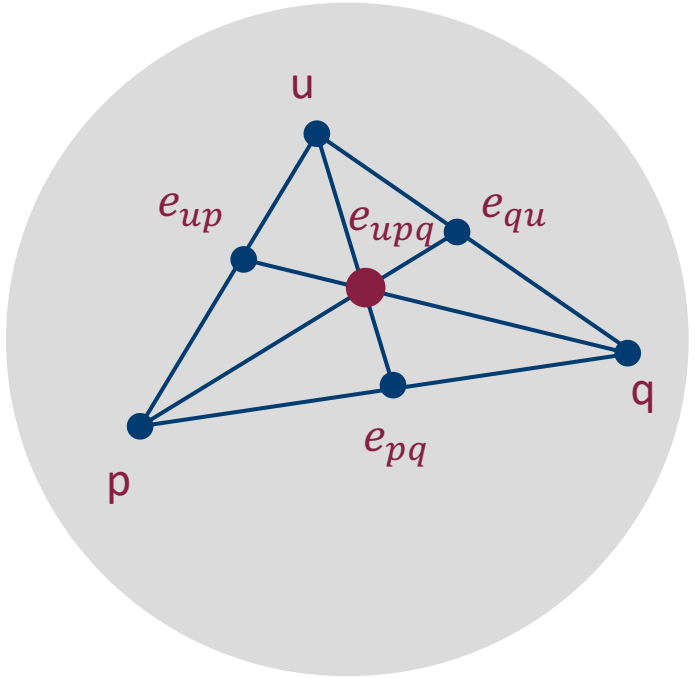
$$r_i\left(\{\alpha_{ij}\}_j, \{v_{ij}\}_j\right) = \sum_j softmax(\alpha_{ij}) v_{ij}$$

Solution:

- Use hyperbolic distance for matching
- Use Klein model and its definition of Einstein midpoint
- Softmax possible due to Euclidean nature of aggregation.

Hyperbolic Neural Networks

Layer: Attention Layer (Einstein Midpoint)



Lorentz Factor acts as weightage for calculating the mean; $\gamma(v) = \frac{1}{\sqrt{1-\|v\|^2}}$

$$e_{pq} = \frac{\gamma(p)p + \gamma(q)q}{\gamma(p) + \gamma(q)}$$

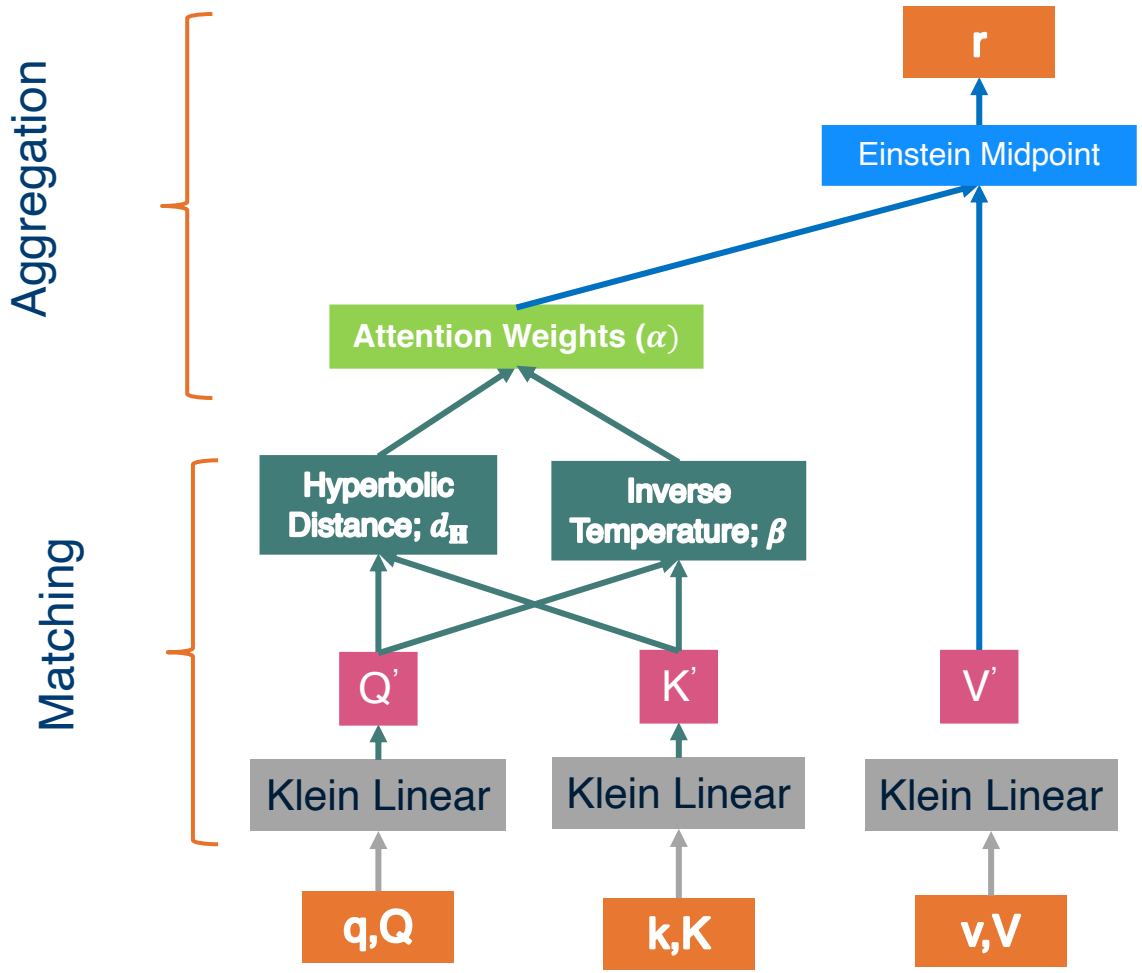
$$e_{qu} = \frac{\gamma(q)q + \gamma(u)u}{\gamma(q) + \gamma(u)}$$

$$e_{up} = \frac{\gamma(u)u + \gamma(p)p}{\gamma(u) + \gamma(p)}$$

$$e_{upq} = \frac{\gamma(u)u + \gamma(p)p + \gamma(q)q}{\gamma(u) + \gamma(p) + \gamma(q)}$$

Hyperbolic Neural Networks

Layer: Attention Layer



Hyperbolic:

(matching; calculating attention weights)

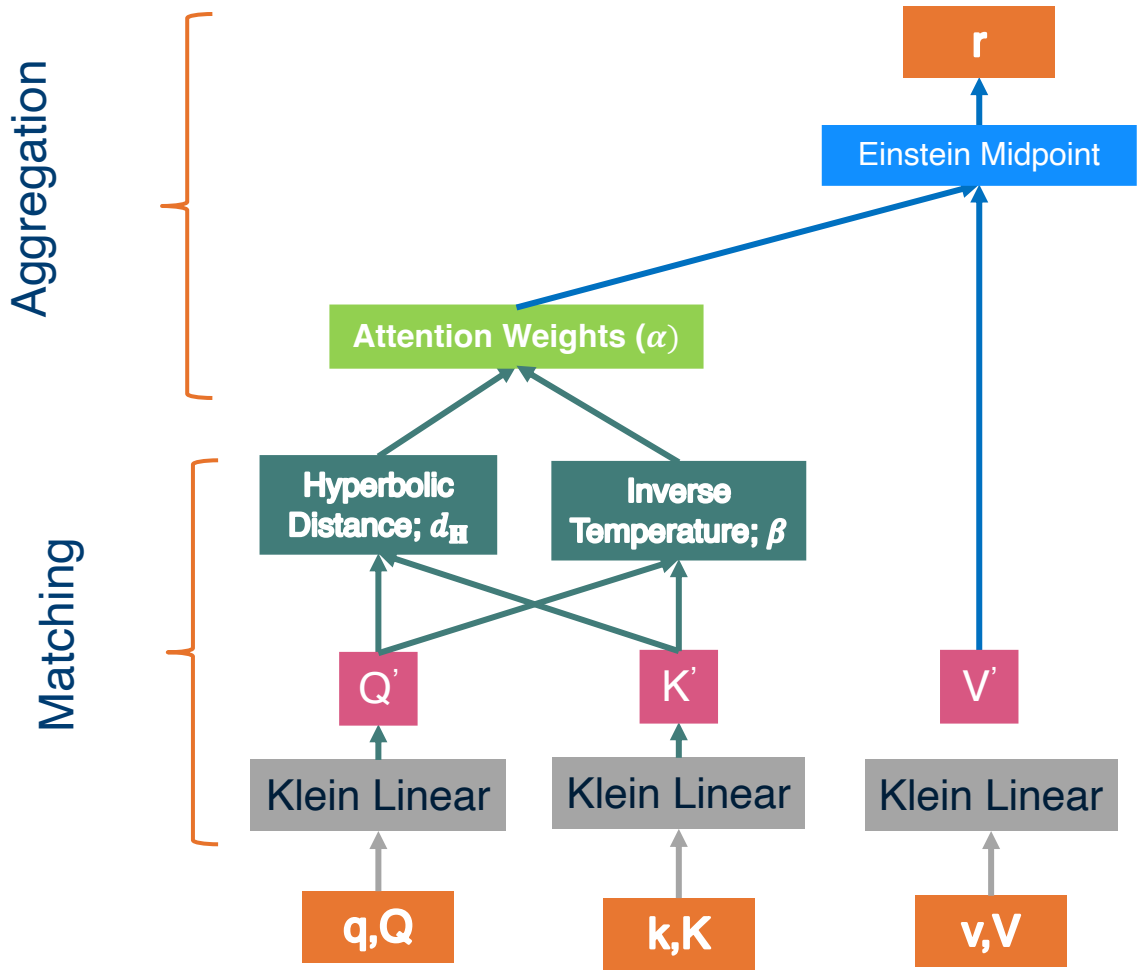
$$\alpha(q_i, k_j) = f\left(-\beta d_{\mathbb{H}}(q_i, k_j) - c\right)$$

(aggregation)

$$r_i\left(\{\alpha_{ij}\}_j, \{v_{ij}\}_j\right) = \sum_j \left[\frac{\alpha_{ij}\gamma(v_{ij})}{\sum_l \alpha_{il}\gamma(v_{il})}\right] v_{ij}; \gamma(v) = \frac{1}{\sqrt{1-\|v\|^2}}$$

Hyperbolic Neural Networks

Layer: Attention Layer



Implementation

1. Transform query, key and value with the Klein Linear layer.
2. Calculate hyperbolic distance (manifolds.lorentz.sqdist) and inverse temperature (manifolds.lorentz.beta) using the query and key.
3. Multiply them and apply a Linear layer with bias to get the attention weights.
4. Aggregate the attention weights and values using Einstein midpoint to get the final dense representation.

Hyperbolic Neural Networks

Practical Aspects

- Weights and biases can lie in either hyperbolic or Euclidean space, depending on the design choice.
  - Optimization algorithm (AdaM or RAdaM) needs to be chosen accordingly
- Applying incorrect operation to a corresponding manifold can cause instability in specific models.
- Exponential maps transform features non-linearly, activation may not be needed.
- Max pooling and Batch normalization are direct.
  - Mean pooling can be derived with gyro operations (Einstein midpoint for Lorentz manifold).

Hyperbolic Neural Networks
Practical Aspects

- In some cases one should use tangent space at origin or at a current point.
  - All transformation operations lead to loss of information, hence, it is preferred to operate in a tangent space at the exact point if possible. However, that is generally not feasible.
  - Hence, tangent space at a point is generally used when there is a source of operation, such as, root node for aggregation in convolution layers.
  - However, in most cases of transformations on singular points, tangent space at origin is sufficient.

Hyperbolic Neural Networks

Extensions of the hyperbolic operations

Due to the practical aspects of mixing Euclidean operations and hyperbolic operations, new approaches aim to unify the benefits into any one space.

- Move everything to hyperbolic space;

  Hyperbolic Neural Networks ++

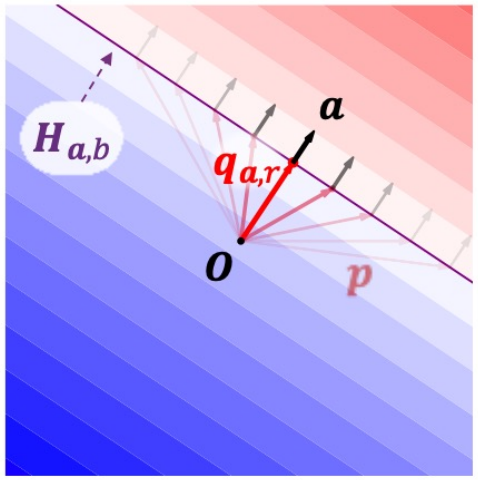
- Move everything to Euclidean space;

  Pseudo-Poincaré framework

Hyperbolic Neural Networks

Hyperbolic Networks ++

This is a new formalization of the Poincaré space. The current methods depend on the tangent space for several operations and the frequent back and forth mapping is both expensive and prone to a loss of data.



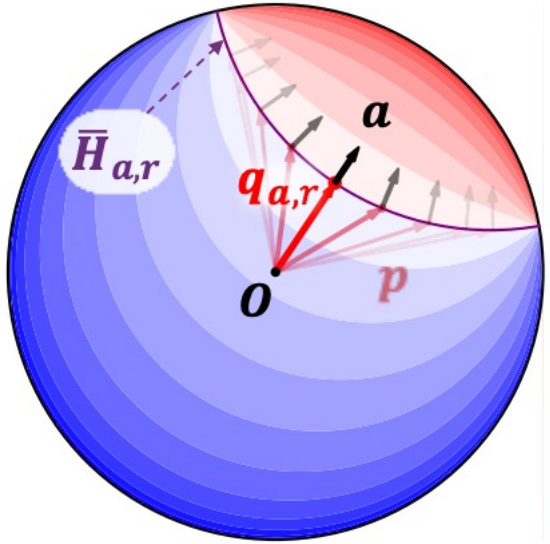


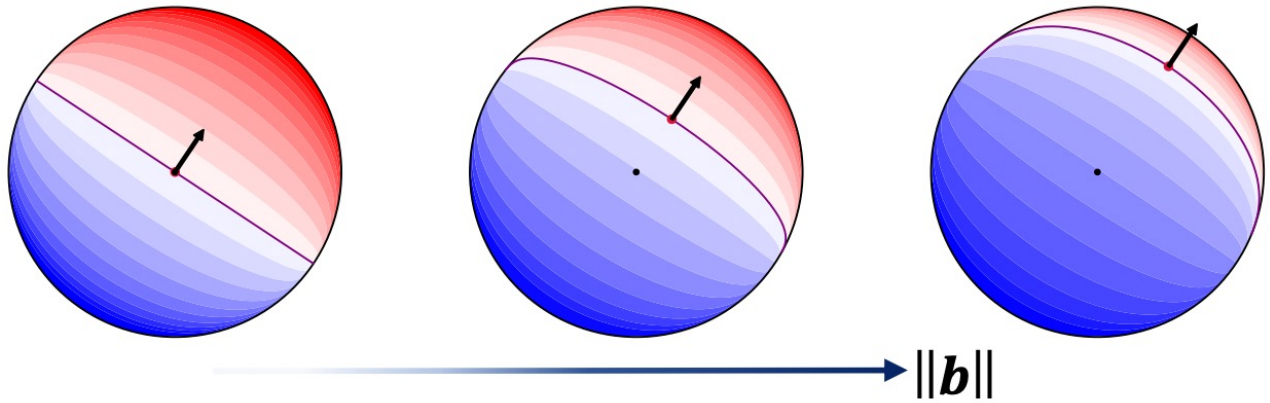
Image Source: Shimizu, Ryohei, Yusuke Mukuta, and Tatsuya Harada. "Hyperbolic neural networks++." ICLR (2021).

Hyperbolic Neural Networks

Hyperbolic Networks ++

Poincaré Ball

$$y = exp_0^c\left(W log_0^c(x)\right) \oplus_c b$$



Hyperbolic Neural Networks ++

$$y = \mathcal{F}^c(x; Z, r) = \mathrm{w}\left(1 + \sqrt{1 + c\|w\|^2}\right)^{-1}$$

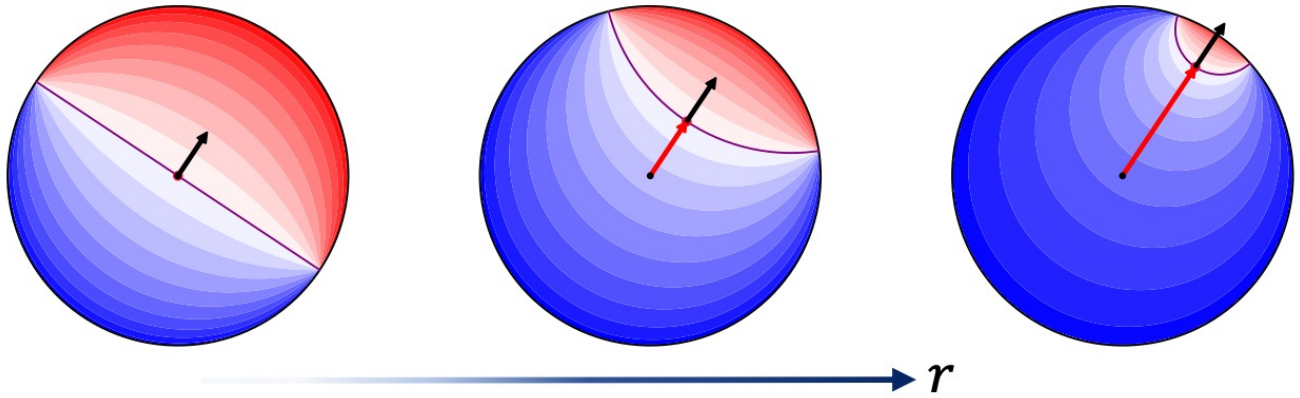


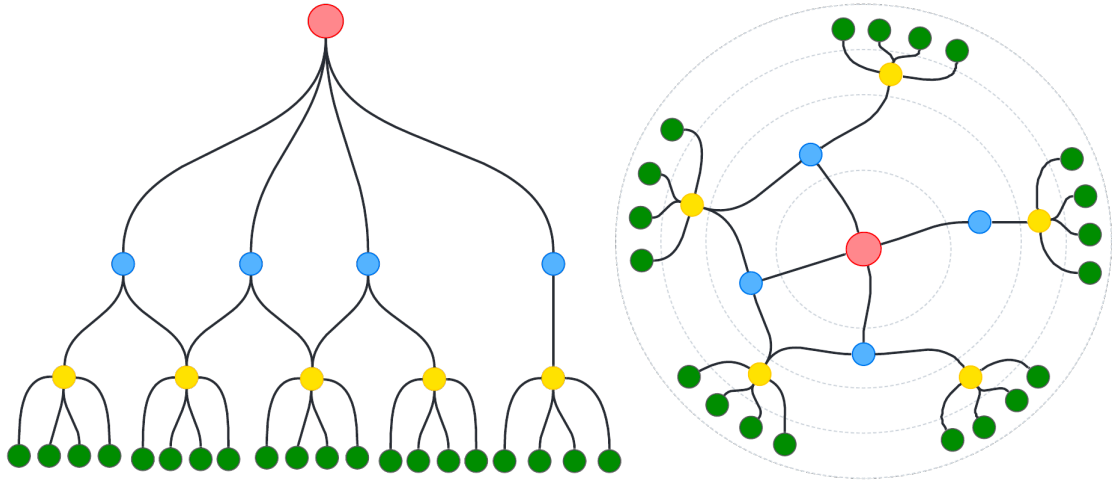
Image Source: Shimizu, Ryohei, Yusuke Mukuta, and Tatsuya Harada. "Hyperbolic neural networks++." ICLR (2021).

Hyperbolic Neural Networks

Pseudo-Poincaré framework

Pseudo-Poincaré space identifies the specific advantages of hyperbolic space and formulates it in the Euclidean space.

The primary advantage of Poincaré space is that it encodes hierarchical relations with exponential increase in volume. But it faces certain practical challenges;



Hyperbolic Neural Networks

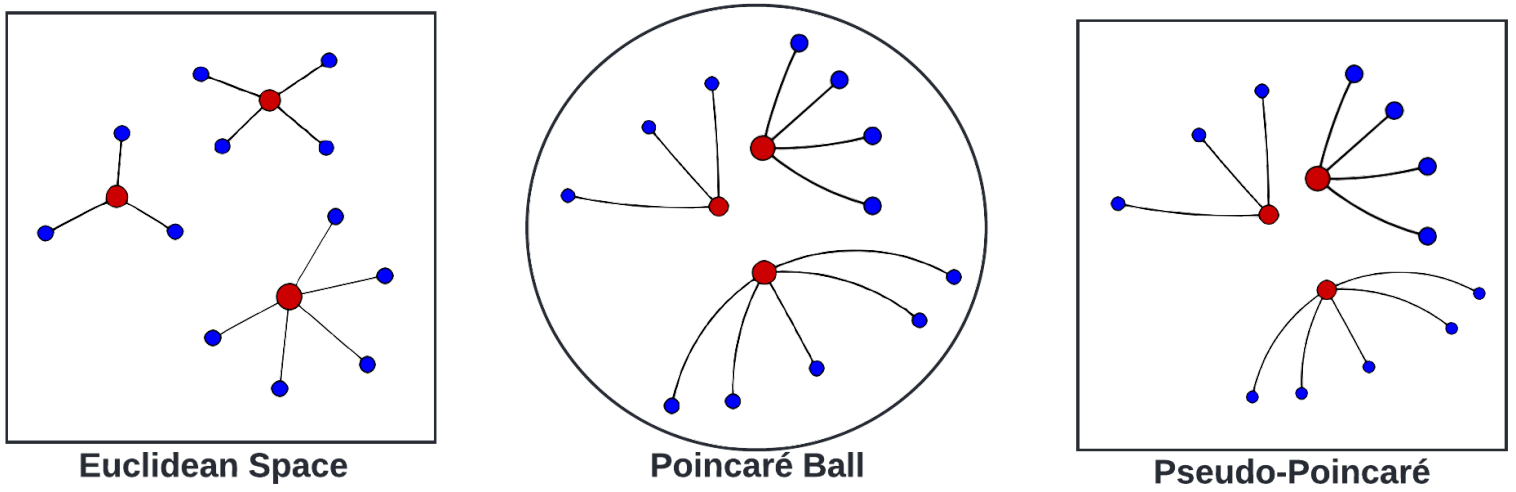
Pseudo-Poincaré framework

Challenges:

1. Non-scalability due to inverse hyperbolic functions.
2. Unstable training and gradient descent due to skips between hyperbolic and Euclidean space.
3. Non-closure of hyperbolic space.
4. Not-trivially extensible to complex Euclidean architectures.

Hyperbolic Neural Networks

Pseudo-Poincaré framework



Solution:

- Use hyperbolic mapping as a normalization function.
- The use of hyperbolic space can be limited to the input and output for expanding and contracting the representational space.

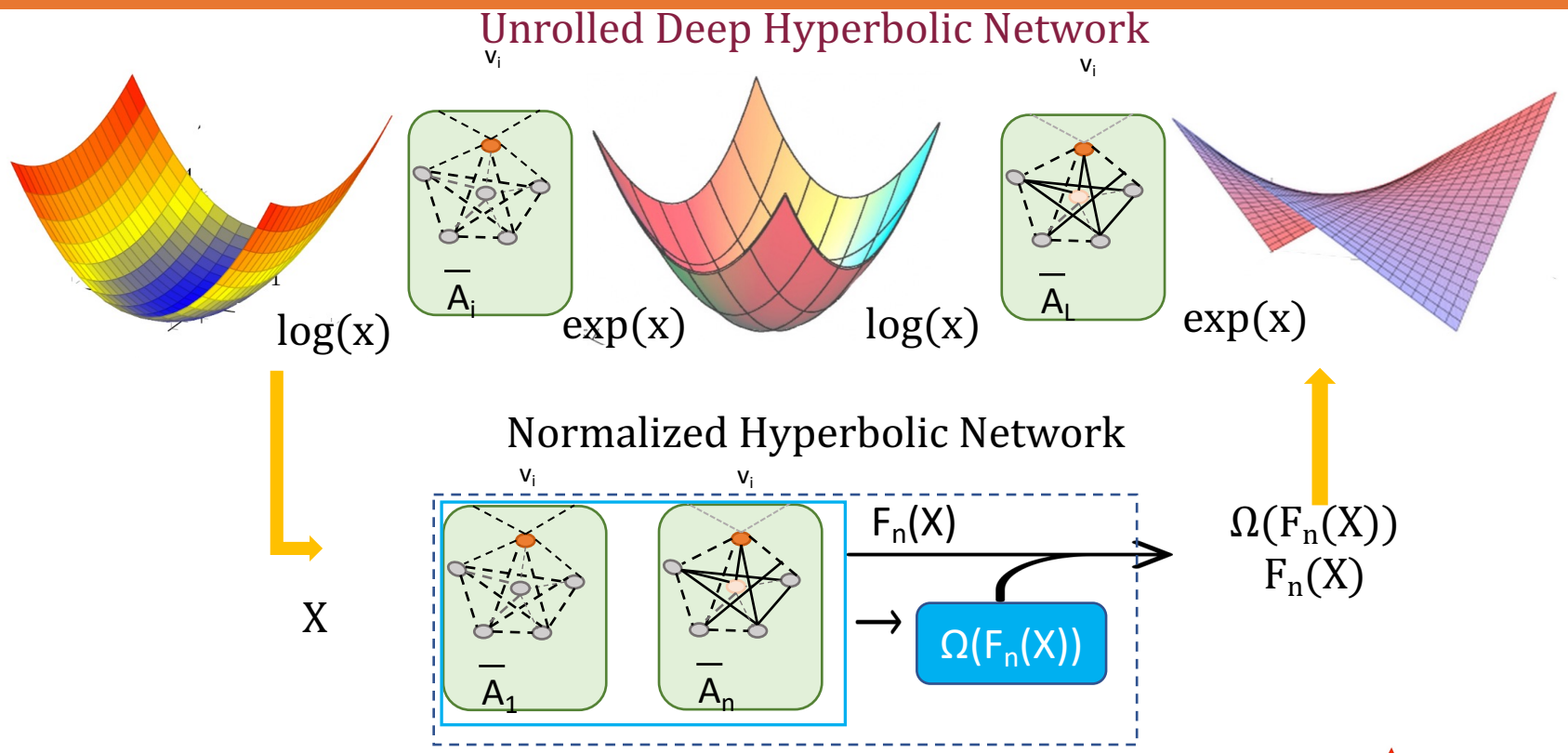
$$NF_n^{\otimes^c}(x) = \Omega\big(F_n(x)\big)F_n(x)$$

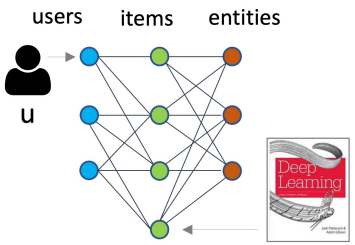
$$NGCN_n^{\otimes^c}(x) = \Omega\big(GCN_n(x)\big)GCN_n(x)$$

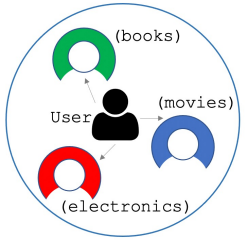
$$NGAT_n^{\otimes^c}(x) = \Omega\big(GAT_n(x)\big)GAT_n(x)$$

Hyperbolic Neural Networks

Pseudo-Poincaré: Hyperbolic Normalization



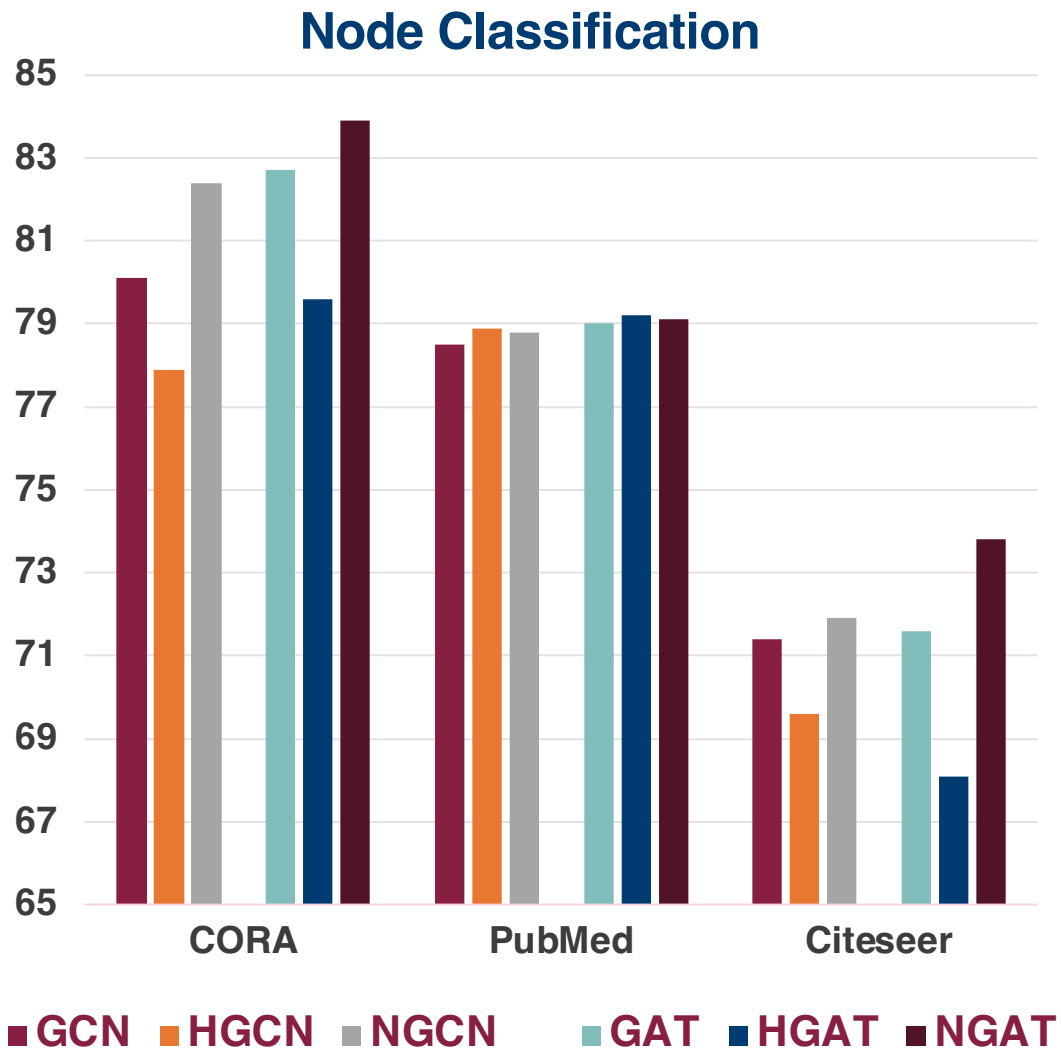


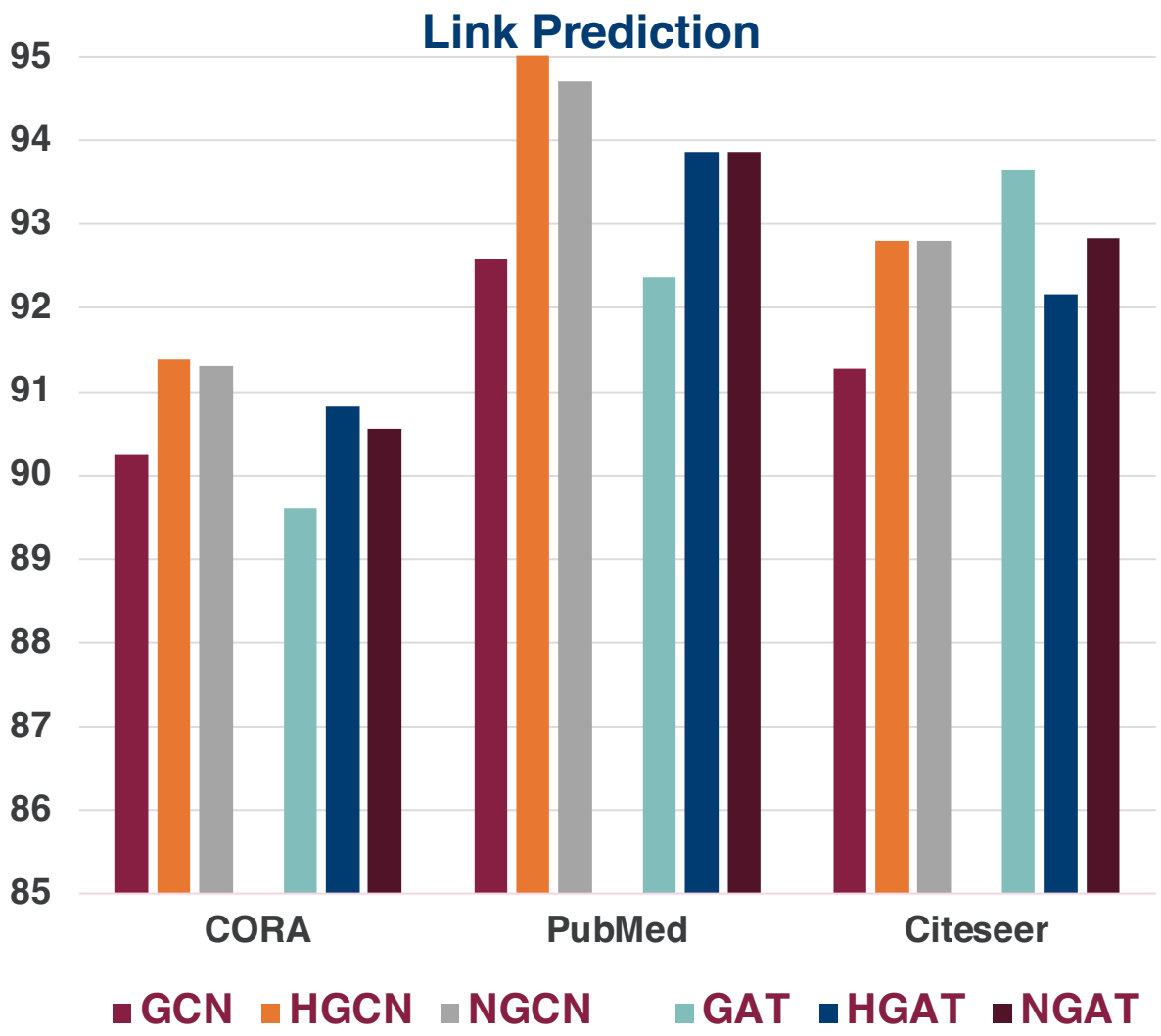


Multi-relational Graph | Poincaré Ball | Hyperbolic space

Hyperbolic Neural Networks

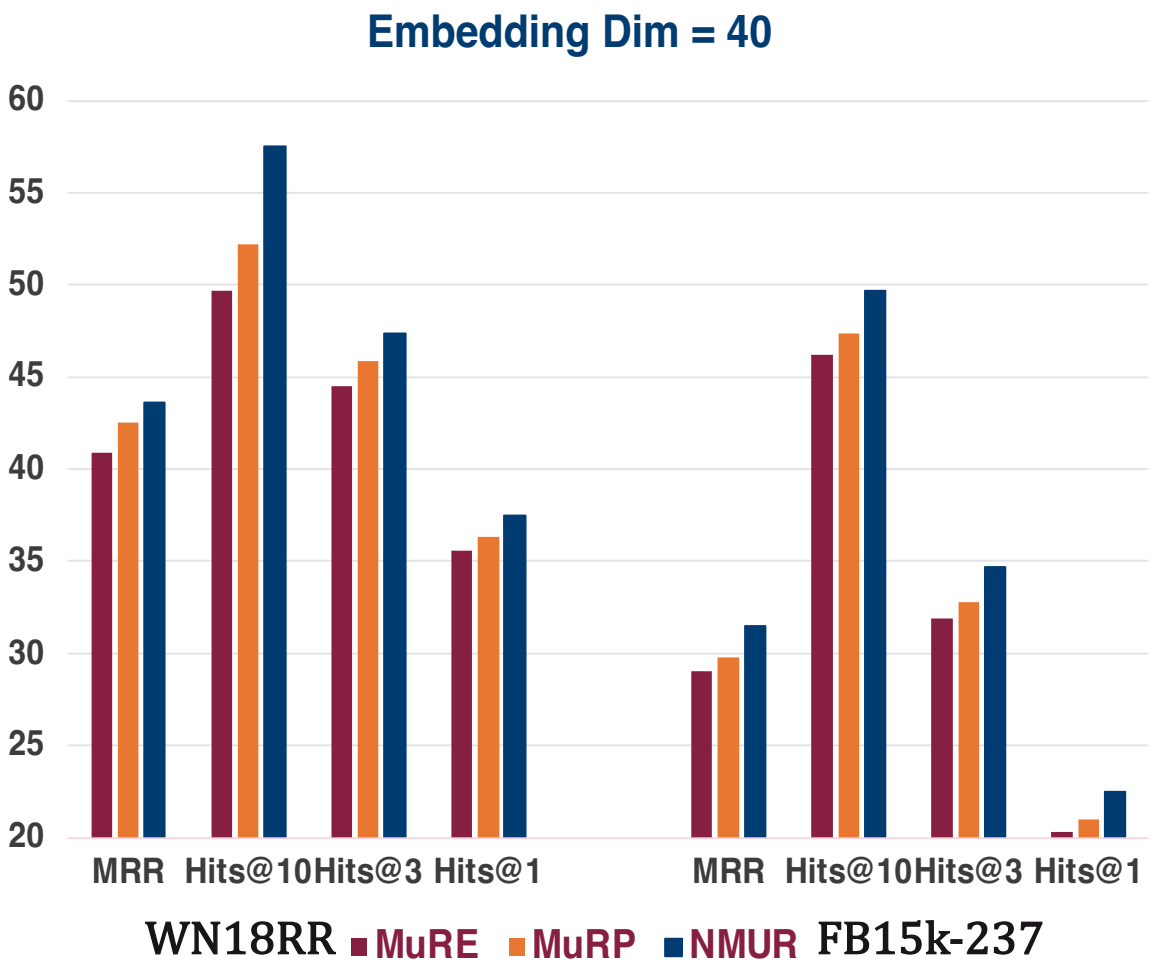
Pseudo-Poincaré: Results on Graph Processing tasks

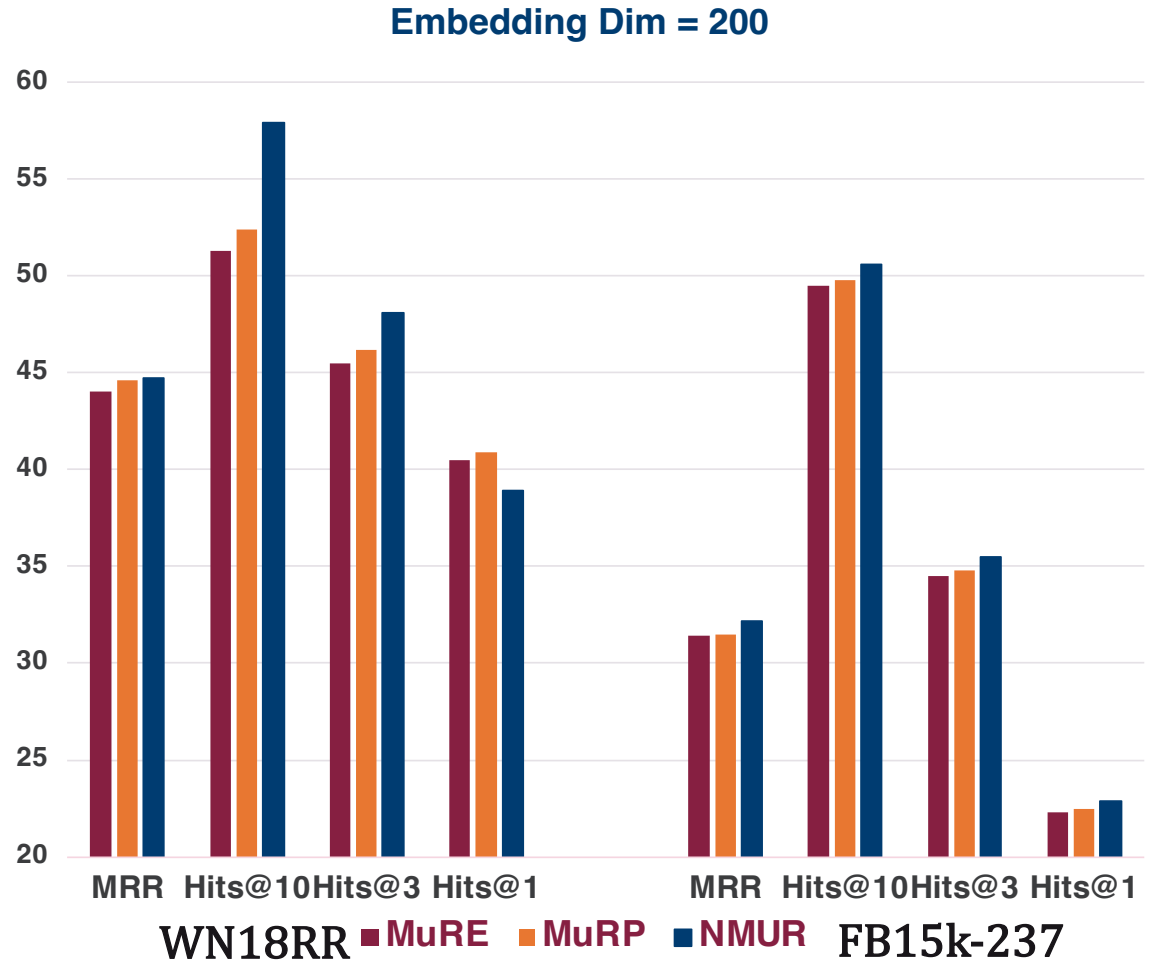




Hyperbolic Neural Networks

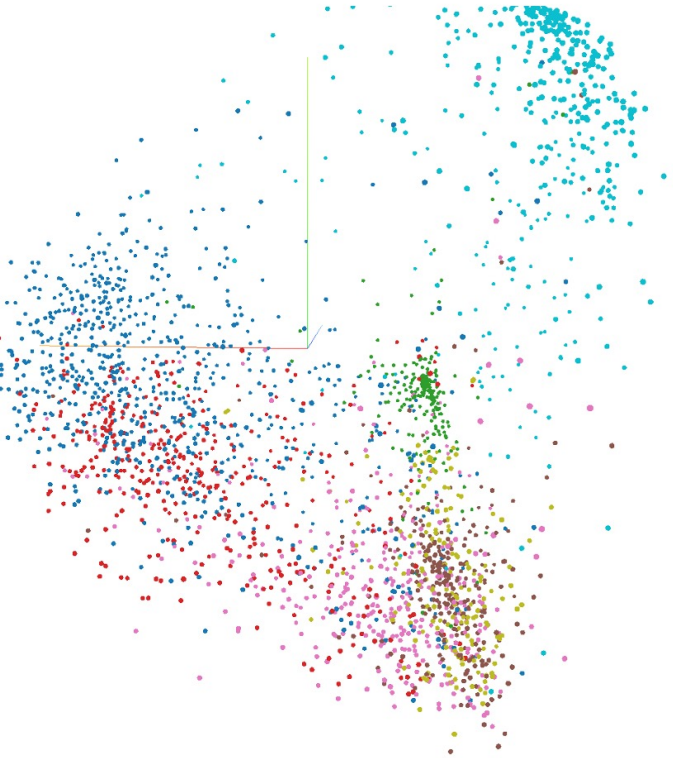
Pseudo-Poincaré: Multi-relational Representations





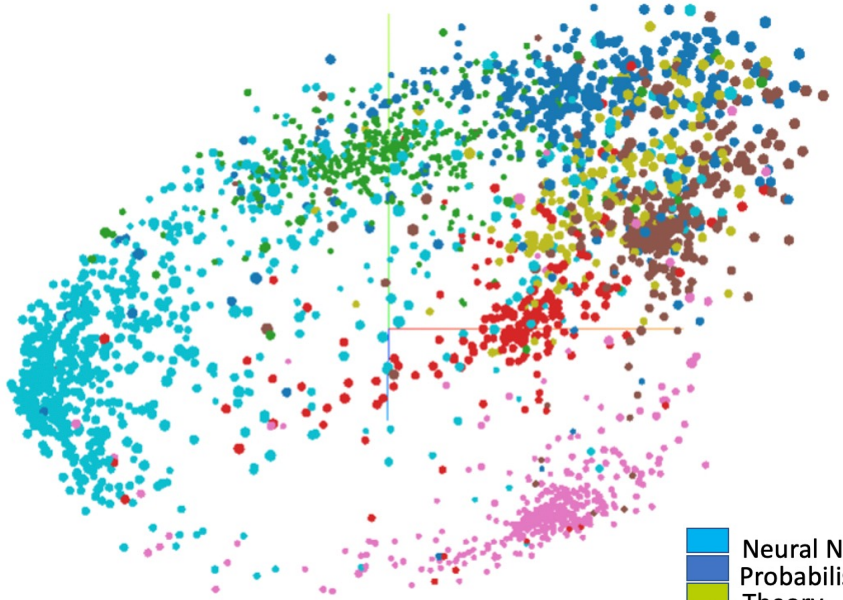
Hyperbolic Neural Networks

Pseudo-Poincaré: Multi-relational Representations

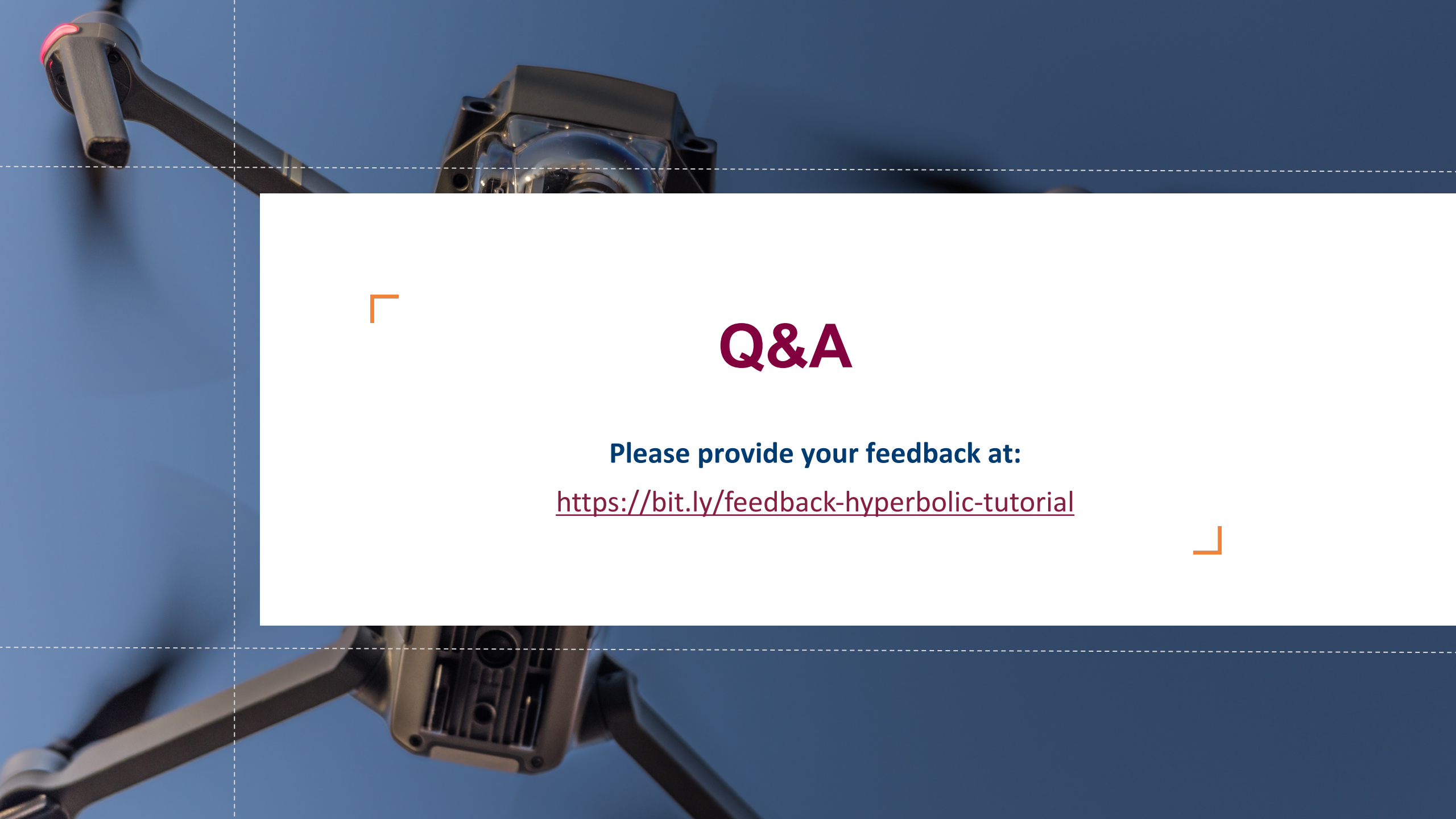


GAT

HGAT



NGAT (Ours)



Q&A

Please provide your feedback at:

https://bit.ly/feedback-hyperbolic-tutorial