Towards Scalable Hyperbolic Neural Networks using Taylor Series Approximations

Nurendra Choudhary, Chandan K. Reddy
Department of Computer Science, Virginia Tech, Arlington, VA, USA
nurendra@vt.edu, reddy@cs.vt.edu

Abstract

Hyperbolic networks have shown prominent improvements over their Euclidean counterparts in several areas involving hierarchical datasets in various domains such as computer vision, graph analysis, and natural language processing. However, their adoption in practice remains restricted due to (i) non-scalability on accelerated deep learning hardware, (ii) vanishing gradients due to the closure of hyperbolic space, and (iii) information loss due to frequent mapping between local tangent space and fully hyperbolic space. To tackle these issues, we propose the approximation of hyperbolic operators using Taylor series expansions, which allows us to reformulate the computationally expensive tangent and cosine hyperbolic functions into their polynomial equivariants which are more efficient. This allows us to retain the benefits of preserving the hierarchical anatomy of the hyperbolic space, while maintaining the scalability over current accelerated deep learning infrastructure. The polynomial formulation also enables us to utilize the advancements in Euclidean networks such as gradient clipping and ReLU activation to avoid vanishing gradients and remove errors due to frequent switching between tangent space and hyperbolic space. Our empirical evaluation on standard benchmarks in the domain of graph analysis and computer vision shows that our polynomial formulation is as scalable as Euclidean architectures, both in terms of memory and time complexity, while providing results as effective as hyperbolic models. Moreover, our formulation also shows a considerable improvement over its baselines due to our solution to vanishing gradients and information loss.

1 Introduction

Recent works on hyperbolic neural networks (HNNs) [14] have shown that a number of problems in fields such as biology, network science, e-commerce, and computer vision possess a non-Euclidean anatomy [7]. Consequently, this has led to the development of novel HNNs in the fields of graph analysis [8], e-commerce [11], computer vision [17], and reasoning [10]. These models, while superior to their Euclidean counterparts in performance, are only able to handle relatively smaller size datasets. The primary reason for this is their non-scalability on accelerated deep learning hardware such as GPUs and TPUs. The basic hyperbolic replacements of even simple operations (such as additions and multiplications) rely on computationally expensive and non-scalable hyperbolic functions (such as \( \tanh \), \( \cosh \), \( \tanh^{-1} \), and \( \cosh^{-1} \)). Additionally, the gyrovector operations [14] used in these models limit the hyperbolic vectors to the range of \( \tanh^{-1}(x) \in (-1, 1) \) (shown in Figure 1a). Note that here \( c^{-1} \) is multiplied on both sides, where \( c \) is the hyperbolic curvature. Due to this, the training of large hyperbolic models run into out-of-domain errors which are generally handled using domain clamping [8]. This leads to the issue of vanishing gradients in large HNNs. To tackle this problem, certain approaches [16, 23] have proposed the idea of feature clipping to limit the radius of hyperbolic manifolds. Furthermore, as shown in Figure 1b, a general HNN frequently uses the exponential and logarithmic maps to learn representations in the hyperbolic space and operate in the Euclidean space, respectively. While this approach is beneficial in adopting new Euclidean
architectures to the hyperbolic space, it also leads to a significant loss of information. To avoid this issue, certain methods propose to formulate all HNN operations in the hyperbolic space. But, this increases the memory and time complexity of existing formulation, subsequently, reducing its scalability for minor improvements in performance. This approach also limits the adoption of research on Euclidean components such as optimization methods and activation functions in HNNs.

(a) Restricted $\mathbb{H}_c$ in the exploration space of $\mathbb{R}$.

(b) Information loss due to frequent $\log_c$ and $\exp_c$ mapping to operate in the Euclidean space and represent in the hyperbolic space, respectively.

To tackle these issues, we investigate the gyrovector formulation of HNNs in the Poincaré ball to determine their key benefits and implementation challenges. Our insights into the problem show that the primary advantage of HNNs is their ability to utilize the hyperbolic trigonometric functions and exponentially expand the representation space with increasing radius of the Poincaré ball, which also corroborates their impressive performance gains on hierarchical datasets. However, the hyperbolic trigonometry also leads to computational inefficiency due to GPU non-scalability.

Hence, we aim to retain the benefits of the hyperbolic representational space while mathematically reformulating them for effective computation. To this end, we propose a mathematical approximation of the hyperbolic trigonometry functions using polynomial Taylor series expansions (PTSE) to define a computationally efficient set of HNN components. PTSE provides a pseudo-hyperbolic polynomial equivariant of the hyperbolic operations that is compatible with the current GPU hardware, thus, enabling model parallelization and computing acceleration. Furthermore, given its Euclidean nature, the PTSE formulation can also utilize the advancements of Euclidean network components such as the ReLU activation to tackle the issue of vanishing gradients, and also does not require the frequent mapping between tangent and hyperbolic spaces. It should be noted that using PTSE instead of gyrovector formulation will lead to an approximation loss. However, this loss can be managed by tuning the precision of PTSE according to the dataset and hardware. In our empirical evaluations, we observe that PTSE formulations have 85% lower latency than their gyrovector formulations. We also notice that the PTSE formulations are more scalable through empirical evidence of performance on multiple GPUs. Also, to evaluate the performance of PTSE formulation, we compare it against its hyperbolic and Euclidean counterparts on standard problems in the areas of graph analysis, natural language processing, and computer vision.

The main contributions of this paper are as follows:

- We reformulate the hyperbolic network components by utilizing the Taylor series expansions of hyperbolic trigonometric functions for achieving scalability and compatibility with accelerated deep learning hardware. We handle the problem of approximation loss by tuning the PTSE precision. To the best of our knowledge, no other work has explored such an alternative formulation.
- We show that PTSE formulations, while maintaining comparable performance, are 85% faster than their hyperbolic counterparts through empirical evaluation on standard problems in graph analysis and computer vision.
- We demonstrate the effect of precision and regularization factor in training the PTSE models through an ablation study. Also, we further show the inherent hyperbolic nature of PTSE formulation through experiments with both hyperbolic and Euclidean optimizers.

---

1The underlying arctanh and arcosh operations are not scalable on GPUs.
2Polynomial operations only contain tensor additions and multiplications which are natively supported by GPUs.
3Latency $\propto \frac{1}{T}$, where $T$ is the total time taken to train the model.
4Note that we experiment with problems in three areas in order to avoid any biases to a particular dataset/domain.
The rest of the paper is organized as follows: Section 2 describes the related research, Section 3 presents the hyperbolic operations with the proposed PTSE reformulation. Section 4 details our evaluation, Section 5 concludes the paper and Section 6 discusses the broader impact of our work.

2 Related Work

Hyperbolic Neural Networks. HNNs have recently gained prominence in several research problems involving hierarchical datasets. For the popular Poincaré ball model with a curvature $c$, Ganea et al. [14] introduced the essential gyrovector operations for training neural networks such as Möbius addition ($\oplus_c$), Möbius scalar multiplication ($\odot_c$), parallel transport ($P^c_x$), exponential map ($\exp_c^x$), and logarithmic map ($\log_c^x$). Building on this, HGCN [8] and HypE [10] models were developed for hyperbolic deep learning in graphs and knowledge graphs, respectively. In HAN [15], authors developed the Lorentz [33] gyrovector operations to model the attention mechanism for improvement on machine translation and visual question answering. HHAN [36] and HGAN [37] applied the framework for hierarchical aggregation and graph problems, respectively. In the e-commerce domain, ANTHEM [11] showed an improved performance over the baselines for product search. In computer vision, Hyperbolic Protonet [17] showed better performance over its Euclidean counterpart for image embeddings. While these methods show the impressive performance of HNNs, they are not scalable to practical problems with large datasets due to the underlying complexity of traditional HNN formulation.

Hyperbolic Formulations. In addition to the initial gyrovector operations [14], several new formulations have been developed to solve specific problems. Fully hyperbolic networks [9] and Hyperbolic networks++ [28] have formulated the gyrovector operations completely in the hyperbolic space to avoid the frequent mapping from tangent space. These works also cover their respective formulation’s extension to linear, convolution, and attention networks. Mathieu et al. [20] constructs Poincaré variational autoencoders through Gaussian generalizations on the hyperbolic space. Another line of research [16] provides a solution to vanishing gradients by limiting the radii of Poincaré ball model. While these methods are effective for their specific problems, they do not provide a complete solution to various limitations of a generalized hyperbolic learning model.

3 Taylor Series Approximation of Hyperbolic Operators

Our primary aim is to reformulate the hyperbolic gyrovector operations to (i) improve scalability, (ii) limit the problem of vanishing gradients and (iii) avoid the frequent mapping from the tangent space. We reformulate the essential gyrovector operations using Taylor series expansions with tunable precision and extend them to more complex neural network components such as layers, activation functions, and distance metrics.

3.1 Taylor Series Reformulation of Gyrovector Operations

For simplicity and ease of comparison across existing techniques, we show the reformulation of the popular Poincaré ball model of the hyperbolic space [14]. However, the technique can be trivially extended to other hyperbolic space models such as the Beltrami-Klein [30] and Lorentz [38] model. The Poincaré ball formulation primarily relies on the hyperbolic trigonometric functions of $\tanh$, $\tanh^{-1}$, and $\cosh^{-1}$. However, these functions are not scalable on GPUs and have limited range which further leads to information loss. Hence, we approximate the behavior of these functions using their polynomial Taylor series expansion (PTSE).

Definition 1. Hyperbolic trigonometric functions can be approximated with their PTSE of $n$ terms. The parameter $n$ is tunable to balance precision and computational costs, where higher value of $n$ implies lower approximation error but higher computational cost. The PTSE approximation

$^5$GPUs are generally faster for deep learning due to their ability to parallelize basic operations such as tensor addition and multiplication. But, hyperbolic functions are complex and their GPU parallelization is non-trivial.
Where $K$ is a polynomial function, $Mv$ is a scalable vector addition $(\oplus)$, and parallel transport $(P_{0\rightarrow x})$ are as follows:

$$x \oplus y = \frac{(1 + 2c(x, y) + c\|x\|^2)\|x\|^2 + (1 - c\|x\|^2)y}{1 + 2c(x, y) + c\|x\|^2\|y\|^2} \equiv K_x x + K_y y$$

$$\exp_x^c(v) = x \oplus \sum_{i=1}^{n} (-1)^{i-1} 2^{2i}(2^{2i} - 1)B_n x^{2i-1} \left(\frac{\sqrt{c}\|x\|^2}{(2i)!}\right)^{2i-2} \equiv K_x x + K_{\exp} v$$

$$\log_x^c(y) = \sum_{i=1}^{n} \frac{\lambda x^{2i-1}(-\sqrt{c}x \oplus y)^{2i-2}}{2i - 1} \equiv K_x x + K_y y$$

$$r \circ^\circ x = \frac{(-1)^{i-1} 2^{2i}(2^{2i} - 1)B_n \left(\sqrt{c}\|x\|^2\right)^{2i-2}}{(2i)!} \cdot \frac{\left(\sqrt{c}\|x\|^2\right)^{2i-2}}{2i - 1} r x \equiv K_{r \circ} r x$$

$$M \circ^\circ v = \sum_{i=1}^{n} (-1)^{i-1} 2^{2i}(2^{2i} - 1)B_n \left(\sum_{j=1}^{n} \frac{\left(\sqrt{c}\|x\|^2\right)^{2j-2}}{2j - 1} r x\right)^{2i-1} \equiv K_{M \circ} M v$$

$$P_{0\rightarrow x}^\circ(v) = \sum_{i=1}^{n} \frac{\|m\|^{2i-1}}{2i - 1} m \equiv K_x x + K_v v; \quad m = x \oplus \sum_{i=1}^{n} (-1)^{i-1} 2^{2i}(2^{2i} - 1)B_n \left(\sqrt{c}\|v\|^2\right)^{2i-2} v$$

In the above PTSE formulation, we note that the hyperbolic vector operations are reduced to their equivalent Euclidean operation scaled by polynomial functions\footnote{Additional details on the approximation errors of the PTSE reformulation are provided in Appendix A.} e.g., in $M \circ^\circ v = K_{M \circ} M v$, where $K_{M \circ}$ is a polynomial function, $x \oplus y = K_x x + K_y y$, where $K_x$ and $K_y$ are polynomial functions. Thus, we are able to approximate and replace hyperbolic trigonometric functions, which are non-scalable on GPUs/TPUs, with a scalable equivalent that uses the scalable vector addition and multiplication operation scaled by polynomial functions for hyperbolic gyrovector operations.\\

### 3.2 Extension to Components of Hyperbolic Neural Networks

In this section, we extend our PTSE gyrovector formulation to different HNN components including activation functions, layers, and optimizers. Note that, in the PTSE gyrovector formulation, we do not bound the operations in the hyperbolic space and hence the parameters of PTSE neural networks remain in the Euclidean vector space $\mathbb{R}^P$.

\footnote{Note that polynomial functions are scalable on GPUs and are computationally faster to compute than hyperbolic trigonometric functions.}
Hyperbolic Activation. We convert a Euclidean activation function $\varphi : \mathbb{R}^p \rightarrow \mathbb{R}^p$ to its PTSE hyperbolic equivalent $\tilde{\varphi} : \mathbb{R}^p \rightarrow \mathbb{R}^p$ as follows:

$$\varphi^\circ (x) = \sum_{i=1}^{n} \frac{(-1)^{i-1} 2^i (2^{2i} - 1) B_n(\sigma \left( \sum_{i=1}^{n} \frac{\| \sqrt{c} x \|^{2i - 2}}{2i - 1} \right))^{2i - 1} - x}{(2i)!} \varphi \left( \sum_{i=1}^{n} \frac{\| \sqrt{c} x \|^{2i - 2}}{2i - 1} \right)$$

Here, $\varphi$ can be any Euclidean activation function including sigmoid, tanh, and ReLU among others.

Hyperbolic Distance. We define two distance metrics $d_1(x, y)$ and $d_2(x, y)$ in the PTSE gyrovector space as equivariants to the L1-norm and L2-norm in the Euclidean space, respectively. $d_1$ is efficient as a loss function in hyperbolic networks due to its higher sparsity in prediction tasks, whereas, $d_2$ is applied when vector similarity is the primary goal, e.g., query-keyword matching to calculate the relevance weights in attention models. For vectors $x, y \in \mathbb{R}^p$, the scalar PTSE distance metrics are defined as follows:

$$d_1(x, y) = \frac{2}{\sqrt{c}} \sum_{i=1}^{n} (\sqrt{c} || x + y ||^{2i - 1})$$

$$d_2(x, y) = \text{PTSE} \left( \arccosh \left( 1 + \frac{\| x - y \|^2}{(1 - \| x \|^2)(1 - \| y \|^2)} \right) \right)$$

PTSE Linear Layer. For input features $x \in \mathbb{R}^p$, the PTSE formulation of the hyperbolic linear layer $f^\circ : \mathbb{R}^p \rightarrow \mathbb{R}^d$ is defined as $f^\circ(x) = w \odot x + b = K_{wx} wx + K_t b \equiv K_{(wx,b)}(wx + b)$, where $w \in \mathbb{R}^{d \times p}$ is a learnable weight matrix and $b \in \mathbb{R}^d$ is the bias vector. Note that our formulation is able to handle homogeneous system of solutions by removing the constraint $\|wx\| \neq 0$, which is necessary in previous formulations [14, 8].

PTSE GRU Layer. We can also extend the linear layer to formulate the gates, reset gate ($r_t$) and update gate ($z_t$), of the hyperbolic GRU layer as follows:

$$r_t \equiv \sigma^\circ (W^r \odot h_{t-1} \oplus U^r \odot x_t \oplus b^r)$$
$$z_t \equiv \sigma^\circ (W^z \odot h_{t-1} \oplus U^z \odot x_t \oplus b^z)$$

where $\sigma^\circ$ is the hyperbolic sigmoid activation function, $h_{t-1} \in \mathbb{R}^d$ is the output of the previous timestep, and $x_t \in \mathbb{R}^p$ is the input at current timestep. $W^r, W^z \in \mathbb{R}^{d \times d}$ are the learnable weight parameters of the reset and update gates, respectively for the previous timestep and $U^r, U^z \in \mathbb{R}^{d \times d}$ are the same for the input at current timestep. The formulation can be further simplified using the PTSE linear layer for computational efficiency [8].

Finally, the reset memory $\hat{h}_t$ and final output $h_t$ is calculated as follows:

$$\hat{h}_t = \varphi^\circ (W \text{diag}(r_t) \odot h_{t-1} \oplus U \odot x_t \oplus b)$$
$$h_t = h_{t-1} \oplus \text{diag}(z_t) \odot (-h_{t-1} \oplus \hat{h}_t)$$

where $\varphi$ is the appropriate hyperbolic activation function.

PTSE Convolution Layer. The hyperbolic convolution layer, as defined by [Chami et al. 8] in the context of graph neural networks, consists of two sequential operations: feature transformation and aggregation. We formulate the PTSE feature transformation at the $l$th layer, $h_{a,l}^\circ$, for input $x_a \in \mathbb{R}^p$ as a linear layer, i.e., $h_{a,l}^\circ = W^l \odot x_a \oplus b_l \equiv K_{(W^l, b_l)}(W^l x_a + b_l)$. The formulation of feature aggregation depends on the problem. In this section, we present the feature aggregation of convolution layers specifically in the context of hyperbolic graph neural networks that uses attention-based aggregation over the nodes’ neighborhood. However, note that the solution can be trivially extended to other problems as well, with a change in aggregation methodology, e.g., for computer vision, one could use max-pooling based spatial aggregation. Given a node $a$ in graph $G$ with a set of node neighbors $b \in \mathcal{N}(a)$, the PTSE feature aggregation at the $l$th layer is formulated as follows:

$$h_{a,l}^\circ = W^l \odot x_a \oplus b_l$$  \hspace{1cm} \text{Feature Transformation} \tag{12}$$

$$w_{ab}^l = \text{SOFTMAX}_{b \in \mathcal{N}(a)} \left( \text{MLP} \left( \frac{n}{2i - 1} \frac{\| \sqrt{c} \| h_{a,l}^\circ \|^{2i - 2}}{2i - 1} h_{b,l}^\circ \right) \right)$$

$$y_a^l = \sum_{i=1}^{n} \left( -1 \right)^{i-1} 2^i (2^{2i} - 1) B_n(\sqrt{c} || m ||^{2i - 1})$$
$$x_a^l = \sigma^\circ (y_a^l) \hspace{1cm} \text{Feature Aggregation and input to next layer} \tag{13}$$

\[8\]Details of the simplification and values of the polynomial functions are provided in Appendix C.
where \( W^l \in \mathbb{R}^{d \times p}, b^l \in \mathbb{R}^p, h^l \in \mathbb{R}^p \) are the weights, biases, and output of the feature transformation operation, \( x_i^{l-1} \in \mathbb{R}^p \) is the input from previous layer, \( w_{sb} \in \mathbb{R}^{d \times d} \) are the attention weights, \( y_i^{l} \in \mathbb{R}^d \) is the encoding after feature aggregation, and \( x_i^l \in \mathbb{R}^d \) is the output of PTSE convolution layer.

**PTSE Attention Layer.** The hyperbolic attention layer, as defined by Gulcehre et al. [15], involves the steps of hyperbolic matching and aggregation. The concept of matching in attention layer is meant to capture the similarity between the query and key. To capture this, we utilize \( d2x \) as the distance measure for matching. Given \( q_i, k_j, v_{ij} \in \mathbb{R}^p \) are the input query, key and value vectors, the output of the attention layer is calculated as follows:

\[
\alpha_{ij} = \sigma^2(-\beta d2x(q_i, k_j) - c); \quad \text{Calculate attention weights through PTSE matching.} \tag{14}
\]

\[
x_i = \frac{1}{\sum_j \alpha_{ij} \gamma(v_{ij})} \sum_j \alpha_{ij} \gamma(v_{ij}) v_{ij}; \quad \gamma(v_{ij}) = \frac{1}{\sqrt{1 - \|v_{ij}\|^2}}; \quad \text{Scaled output given in [15].} \tag{15}
\]

where \( \beta \) and \( c \) are the temperature and bias parameters that control the scale/variance and counteract the negative values of the distance metric, respectively. \( \gamma(v_{ij}) \) is the Lorentz factor [15] used to normalize the outputs of the attention layer.

Figure 2: PTSE formulation for learning. (a) shows the extended domain of PTSE learning, i.e., the range of real numbers \( \mathbb{R} \). PTSE does not need to clamp the gradients as there are no out-of-domain errors. (b) PTSE are able to operate on hyperbolic embeddings directly, thus, avoiding the mapping back and forth and limiting the information loss.

Thus, we have the PTSE formulations for the necessary layers, activation functions, and distance metrics which can be combined together in deep learning frameworks. Given their formulation, we see that the PTSE parameters are not restricted to \( \mathbb{H} \), and can explore the entire Euclidean space \( \mathbb{R} \) for solutions. Also, the PTSE layers do not need the frequent mapping between hyperbolic space and tangent space and can be computed in a single operation, i.e., \( f^2(x) = Kf(x) f(x) \). This tackles our major issues of closure (Figure 2a) and loss of information (Figure 2b) in the hyperbolic space. In our experimental study, we use these formulations to construct certain hyperbolic models and compare them against previous baselines in their respective research domains.

### 4 Experimental Study and Evaluation

In this section, we evaluate our model by addressing the following research questions (RQs):

- **RQ1:** How do models based on PTSE formulation compare to the models utilizing the existing hyperbolic formulation [14] and Euclidean formulation in terms of performance and computational complexity?
- **RQ2:** Can the PTSE formulation be extended to hyperbolic models in the domains of multi-relational reasoning, computer vision, and natural language processing?
- **RQ3:** How do the different components and design choice decisions affect PTSE model’s performance?

#### 4.1 Problem Settings

For the comparative analysis, we adopt the following problem settings for our experiments.

**Graph Prediction.** Given a graph \( G = (V \times E) \), with \( v_i \in V \) are the set of vertices with labels \( y_i \), and \( e_{ij} \in E \) is the Boolean adjacency matrix, where \( e_{ij} = 1 \) if a link exists between \( v_i \) and \( v_j \), and 0 otherwise. Based on this context, we formulate the problem of node classification, where the aim is to estimate a predictor model \( P_E \), parameterized by \( \theta \), such that for an input node \( v_i \), \( P_E(v_i | E) = y_i \),
and the problem of link prediction, where for an input node-pair \(v_i, v_j\) and incomplete adjacency matrix \(E^\text{in}\), the objective to estimate a predictor model \(P_\theta\) such that \(P_\theta(v_i, v_j | E^\text{in}) = e_{ij}\).

**Multirelational Reasoning.** Given a knowledge graph of triplets, \(h_i, r_{ij}, t_j \in K \mathcal{G}\), where \(h_i, r_{ij}, t_j \in \mathbb{R}^n\) are the head, relation and tail, respectively. In the task of reasoning, the goal is to predict tails \(t_j \in K \mathcal{G}\) such that they answer the query of head \(h_i\) transformed by relation \(r_{ij}\).

**Few-shot Image Classification.** Let us consider a dataset of images \(x_i \in X\) with corresponding labels \(y_i\). Additionally, we also have a backbone embedding model \(E_\phi\) trained on a large corpus \(\mathcal{C}\), where \(|X| \ll |\mathcal{C}|\). The goal here is to estimate a predictor model \(P_\theta\) in a limited number of training steps \(n\) such that \(P(x_i | X) = y_i\).

**Textual Entailment.** Given two sentences, premise \(s\) and a hypothesis \(h\), the goal here is to estimate a \(P_\theta\) such that \(P_\theta(p_i, h_i) = y_i\), where \(y_i = 1\) if \(p_i\) entails \(h_i\), and \(y_i = 0\), otherwise.

**Noisy Prefix Detection.** Given two sentences, \(s_i\) and \(s_j\), the goal here is to estimate a \(P_\theta\) such that \(P_\theta(s_i, s_j) = y_{ij}\), where \(y_{ij} = 1\) if \(s_i\) is a noisy prefix of \(s_j\), or \(y_{ij} = 0\) if \(s_i\) is just a random sentence.

### 4.2 Datasets and Baselines

The primary goal of our experimental study is to compare the underlying formulation criteria of different models. To this end, we select the standard datasets and benchmarks for hyperbolic networks in the areas of graph neural networks (GNN), multi-relational reasoning (MuR), computer vision (CV), and natural language processing (NLP). The datasets, baselines, and evaluation metrics used in our experiments are provided in Table 1. Additional details are provided in Appendix D.

| Domain | Dataset | Our models and baselines | Evaluation Metrics |
|--------|---------|--------------------------|--------------------|
| GNN    | Cora [25] | GCN [19], GAT [34] (Euc) | Accuracy (node classification) |
|        | Pubmed [27] | HGCN [8], HGAT [37] (Hyp) | Area under ROC (link prediction) |
|        | Citeseer [27] | TGCN, TGAT (PTSE, ours) | |
| MuR    | FB15K [5, 31] | MuRE [1] (Euc) | HITS@K |
|        | WN18RR [13] | MuRF [2] (Hyp) | (to compare precision in top-K results) |
|        |           | MuRT (PTSE, ours) | Mean Reciprocal Rank (to compare ranking) |
| CV     | MiniImageNet [26] | ProtoNet [29] (Euc) | Classification Accuracy |
|        | Caltech-UCSD-Birds [32] | HProtoNet [12] (Hyp) | |
|        |           | TProtoNet (PTSE, ours) | |
| NLP    | SNLI [6] | RNN [21] (Euc) | Prediction Accuracy |
|        | PREFIX-K% [14] | GRU [12] (Hyp) | |
|        |           | HRNN [14], HGRU [14] (Hyp) | |
|        |           | TRNN, TGRU (PTSE, ours) | |

**Implementation Details.** We conducted our experimental study on a multi-GPU setup of Quadro RTX 8000 with 48 GB of VRAM each. While our systems did have 48 GB of RAM, a reproduction of our experiments only requires around 12 GB and the actual requirement is much lower (≈ 8 GB) for PTSE formulation. All our models and corresponding baselines are implemented in PyTorch [24] to avoid cross-framework inaccuracies in comparison. The graph experiments also utilize the GraphZoo [33] framework. For the experiments involving computational complexity, the models have been run in isolation to remove any overhead due to memory management processes. Our models and corresponding baselines have all been tuned for best performance. An interesting point of note is on the optimizer choice between Riemannian Adam (RAdam) [4] and Euclidean Adam [18]. As expected, RAdam generally works better for HNNs and Adam for Euclidean NNs, but it is interesting to see that RAdam also works better for PTSE formulation, although the formulation is in the Euclidean space. This provides certain evidence of the inherent hyperbolic nature of PTSE formulation, even though the formulation lies in the Euclidean space. Another important practical detail is the use of weight regularization for PTSE layers. In the PTSE approximation errors, provided in Appendix A, we observe that \(\eta(f(x)) \propto \|f(x)\|\) and hence to maintain low values of \(f(x)\), we add an L1-regularization factor to our layers \(\lambda |f(x)|\), where \(\lambda > 0\), i.e., final output of our layers \(f_y = f^\theta(x) + \lambda |f(x)|\). In this case, \(\lambda\) works as a penalizing control that depends on the nature of \(f(x)\) and the corresponding precision \(n\), e.g., functions like \(\text{arcosh}(x)\) need lower \(\lambda\), whereas, \(\text{artanh}(x)\) need higher \(\lambda\), and also, low values of \(n\) need higher \(\lambda\) and vice-versa. [8]

---

[8] The code of our models and the baselines are provided at [https://github.com/Akirato/TSE-hyperbolic](https://github.com/Akirato/TSE-hyperbolic)
4.3 RQ1: Comparative Analysis of the PTSE formulation

To evaluate the performance of the PTSE formulation, we compare it against the Euclidean and hyperbolic formulation in the well-studied area of graph neural networks. We analyze the formulations on the problems of node classification and link prediction and utilize the standard benchmarks and evaluation criteria, as provided in Table [1].

Table 2: Performance comparison results between PTSE (ours) and the baselines on the tasks of node classification and link prediction in the graph domain. The columns present the evaluation metrics of Accuracy and ROC for node classification and link prediction, respectively. The cells with the best performance are highlighted in bold and the second-best performance is underlined.

| Models  | Cora Node Classification Accuracy (%) | Pubmed Node Classification Accuracy (%) | Citeseer Node Classification Accuracy (%) | Cora Link Prediction ROC (%) | Pubmed Link Prediction ROC (%) | Citeseer Link Prediction ROC (%) |
|---------|--------------------------------------|----------------------------------------|------------------------------------------|----------------------------|-------------------------------|----------------------------------|
| GCN     | 80.1                                 | 78.5                                   | 71.4                                     | 90.2                       | 92.6                          | 91.3                             |
| GAT     | 82.7                                 | 79.0                                   | 71.6                                     | 89.6                       | 92.4                          | 93.6                             |
| HGCN    | 77.9                                 | 78.9                                   | 69.6                                     | 91.4                       | 95.0                          | 92.8                             |
| HGAT    | 79.6                                 | 79.2                                   | 68.1                                     | 90.8                       | 93.9                          | 92.2                             |
| TGCN    | 77.6                                 | 80.8                                   | 68.8                                     | 91.2                       | 94.2                          | 92.3                             |
| TGAT    | 81.7                                 | 81.3                                   | 69.9                                     | 91.2                       | 94.4                          | 92.7                             |

**Performance Comparison.** The comparative results provided on the tasks of node classification and link prediction show that, despite the approximation errors, PTSE formulation is able to perform similar to the standard hyperbolic networks. In the case of node classification, we observe that TGAT is able to outperform the baselines, which is due to the extended exploration space of the solutions. In the case of link prediction, where message propagation is the primary focus, we observe that the performance of PTSE formulation and hyperbolic networks is similar, and hence, we conclude that approximation errors are not a limiting factor for performance.

Figure 3: Speed-up by parallelizing the process over multiple GPUs. Both Euclidean (GCN, GAT) and PTSE (TGCN, TGAT) formulations are able to leverage the multi-GPU setup and reduce the time per epoch and total time taken for convergence in both node classification and link prediction. We observe that, although both HNN and PTSE formulations are slower than Euclidean models on a single-GPU setup, PTSE formulations parallelize over multiple GPUs, to maintain their high performance while lowering their latency to similar times as Euclidean models.

**Computational Complexity.** From the results shown in Figure 3, we observe that both Euclidean and PTSE formulations are significantly faster and are able to speed-up when parallelized over multiple GPUs, in terms of both the time taken per epoch and the total time taken for model convergence. We also notice a slight dip in the time taken by hyperbolic methods, which is due to the parallelization of certain Euclidean operations in the HNNs. However, given the insignificant amount of dip for hyperbolic networks, we conclude that the PTSE formulation is significantly more parallelizable than its hyperbolic equivalent. From the comparative results on performance and computational complexity, we note that PTSE formulation, while similar in performance to hyperbolic networks, is much more scalable on deep learning hardware (GPUs in this case).

4.4 RQ2: Case study of PTSE applications

In this section, we study the implications of using the PTSE formulations in the domains of multi-relational reasoning, computer vision, and natural language processing. This study helps us understand the variety of applications that could leverage PTSE formulations.
Table 3: Performance comparison results between MuRT (ours) and the baselines on the task of multi-relational graph reasoning in the graph domain. The columns present the evaluation metrics of Hits@K (H@K) (%) and Mean Reciprocal Rank (MRR) (%). Best performing values are highlighted in bold and the second best performing ones are underlined.

| Datasets | Dim | Models | MRR | H@10 | H@3 | H@1 |
|----------|-----|--------|-----|------|-----|-----|
| WN18RR   | 40  | MuRE   | 40.9| 49.7 | 44.5| 35.2|
|          |     | MuRP   | 42.5| 52.2 | 45.9| 36.3|
|          |     | MuRT   | 42.2| 52.6 | 44.2| 38.3|
|          | 200 | MuRE   | 44.0| 51.3 | 45.5| 40.5|
|          |     | MuRP   | 44.6| 52.4 | 46.2| 40.9|
|          |     | MuRT   | 44.5| 52.8 | 46.0| 41.0|
| FB15K-237|     |        |     |      |     |     |
|          | MRR | H@10   | 29.0| 46.2 | 31.9| 20.3|
|          |     | H@3    | 29.8| 47.4 | 32.8| 21.0|
|          |     | H@1    | 29.6| 46.0 | 32.6| 22.2|
|          |     |        |     |      |     |     |

Multi-relational Reasoning. In the case of multi-relational reasoning, the goal is to retrieve and rank a set of candidates, based on their relevance to a query head entity and the corresponding relation. We compare our methods using the standard metrics of HITS@K, which determines the precision of the models in their top-K results, and MRR, which quantifies the ranking ability of the models. In our empirical results (presented in Table 3), we observe that our model MuRT is always able to provide results similar to its hyperbolic counterpart MuRP, but is significantly faster and more scalable than MuRP. Results in the areas of computer vision and natural language processing follow a similar trend and are presented in Appendix B.

Figure 4: Dependence of the performance of PTSE model (for node classification in the Pubmed dataset) on the following components: (a) regularization factor, (b) PTSE precision, and (c) Optimizer.

4.5 RQ3: Components of PTSE model

This experiment aims to analyze the components of PTSE model and understand the effect of different design choices including various values of precision $n$, optimizers (Adam vs RAdam), and PTSE-regularization, to the overall performance of PTSE models. As depicted in Figure 4, we observe that Accuracy is dependent on several design choices. We note that the regularization factor is necessary to appropriately weigh the penalty of high values, as both a high and low value of $\lambda$ decreases the model performance. Additionally, we note that the hyper-parameter precision works as expected, where higher precision always returns a better accuracy due to low approximation error, however, the advantage is negligible after a certain value ($n=3$ in our case). Furthermore, we also note that PTSE models perform better with RAdam optimizer, which provides further evidence of the inherent hyperbolic nature of the PTSE formulation.

5 Conclusion

In this paper, we present the PTSE formulation of the hyperbolic space and show that it is more scalable on GPUs/TPUs, compared to the original hyperbolic formulation. The primary reason for PTSE’s effectiveness is its substitution of non-scalable tangent hyperbolic functions with more efficient Taylor series expansions. Additionally, we also note that PTSE formulation avoids the information loss due to frequent mapping back and forth between the tangent space and hyperbolic space, and also, extends the explorability of the solution space from the range of hyperbolic vectors $\mathbb{H}_c$ to the range of all real vectors $\mathbb{R}$. This allows us to apply the benefits of hyperbolic networks on a wider range of applications, especially problems that require the use of large-scale datasets.
6 Broader Impact

The problem of scalability has remained a primary challenge in the development of HNNs, particularly in their applicability to practical problems. Our PTSE formulation is a significant step forward in that direction, and provides evidence that approximate solutions can emulate the gains of hyperbolic networks in a scalable framework. However, certain understanding of the dataset is required for smooth application of the PTSE formulation. One needs to identify the hierarchical nature of the underlying dataset (hyperbolicity can be used as a proxy), as non-hierarchical datasets would benefit more from Euclidean architectures. Furthermore, our method adds the tunable hyper-parameters of precision and \( \lambda \), that needs to be set appropriately, in order to achieve the best performance. However, certain preliminary experiments and an idea of the system precision could help in limiting the hyper-parameter space, thus reducing the effort to compute the precision and \( \lambda \) parameter. Improving the scalability of hyperbolic neural networks could also improve the collection of data from graph datasets, thus, improving the efficiency of bad faith actors in collecting private information or attacking existing information storage systems.

References

[1] Daniel S Abdi, Lucas C Wilcox, Timothy C Warburton, and Francis X Giraldo. A gpu-accelerated continuous and discontinuous galerkin non-hydrostatic atmospheric model. The International Journal of High Performance Computing Applications, 33(1):81–109, 2019.

[2] Gregor Bachmann, Gary Bécigneul, and Octavian Ganea. Constant curvature graph convolutional networks. In International Conference on Machine Learning, pages 486–496. PMLR, 2020.

[3] Ivana Balažević, Carl Allen, and Timothy Hospedales. Multi-relational poincar\’e graph embeddings. In Advances in Neural Information Processing Systems, 2019.

[4] Gary Bécigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=r1eiqi09K7.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26, 2013.

[6] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326, 2015.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine, 34 (4):18–42, 2017.

[8] Ines Chami, Zhitaoy Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. Advances in neural information processing systems, 32, 2019.

[9] Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Fully hyperbolic neural networks. arXiv preprint arXiv:2105.14686, 2021.

[10] Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. Self-supervised hyperboloid representations from logical queries over knowledge graphs. In Proceedings of the Web Conference 2021, pages 1373–1384, 2021.

[11] Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. Anthem: Attentive hyperbolic entity model for product search. WSDM, 22:21–25, 2022.

[12] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In NIPS 2014 Workshop on Deep Learning, December 2014, 2014.

[13] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[14] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. Advances in neural information processing systems, 31, 2018.
[15] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=rJxHsjRqFQ

[16] Yunhui Guo, Xudong Wang, Yubei Chen, and Stella X Yu. Free hyperbolic neural networks with limited radii. arXiv preprint arXiv:2107.11472, 2021.

[17] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 6418–6428, 2020.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR (Poster), 2015. URL http://arxiv.org/abs/1412.6980

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017. Conference Track Proceedings. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgI

[20] Emile Mathieu, Charline Le Lan, Chris J Maddison, Ryota Tomioka, and Yee Whye Teh. Continuous hierarchical representations with poincaré variational auto-encoders. Advances in neural information processing systems, 32, 2019.

[21] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Interspeech, volume 2, pages 1045–1048. Makuhari, 2010.

[22] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In ICml, 2010.

[23] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. Advances in neural information processing systems, 30, 2017.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 2019.

[25] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In AAAI, 2015. URL https://networkrepository.com

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3):211–252, 2015.

[27] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. AI Magazine, 29(3):93–106, 2008.

[28] Ryohei Shimizu, YUSUKE Mukuta, and Tatsuya Harada. Hyperbolic neural networks++. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=Ec85b0tUwbA

[29] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. Advances in neural information processing systems, 30, 2017.

[30] Li Sun, Zhongbao Zhang, Jiawei Zhang, Feiyang Wang, Hao Peng, Sen Su, and Philip S Yu. Hyperbolic variational graph neural network for modeling dynamic graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 4375–4383, 2021.

[31] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In Proceedings of the 2015 conference on empirical methods in natural language processing, pages 1499–1509, 2015.

[32] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. Advances in Neural Information Processing Systems, 30, 2017.

[33] Abraham A Ungar. Analytic hyperbolic geometry: Mathematical foundations and applications. World Scientific, 2005.
Appendix

This document is an appendix to the paper titled “Towards Practical Hyperbolic Neural Networks using Taylor Series Approximations”.

A Approximation Error of PTSE Expansions

As illustrated in figure 5, we note that approximation error is the least for low values of the function $f(x)$ and high values of precision $n$. However, the gains in precision also tend to saturate at a level of precision. Hence, we use a value-regularizer with a weight $\lambda$ to maintain low function values and tune the parameter of precision $n$ for efficient computation, faster convergence, and high performance.

B Case studies in Computer Vision and Natural Language Processing

From the results provided in Table 4, we observe that T-ProtoNet is able to perform significantly better than Euclidean ProtoNet and just slightly lower than the hyperbolic variant. This shows that T-ProtoNet is able to provide better results than Euclidean ProtoNet while being as scalable.

**Computer Vision.** From the results provided in Table 4, we observe that T-ProtoNet is able to perform significantly better than Euclidean ProtoNet and just slightly lower than the hyperbolic variant. This shows that T-ProtoNet is able to provide better results than Euclidean ProtoNet while being as scalable.

**Natural Language Processing.** From the results provided in Table 5, we observe that PTSE formulations are able to perform significantly better than Euclidean counterparts and just slightly lower than the hyperbolic variants. This shows that PTSE is able to provide better results than Euclidean formulation while being as scalable.
Figure 5: Plots of hyperbolic trigonometric functions and their respective PTSE formulations and the relative error between them at various precision values. We observe that the approximation error is the least for low values of the function $f(x)$ and high values of precision $\eta$. 

(a) Plot of the $\tanh$ (solid line) function and its PTSE formulation at different values of precision.

(b) Relative Error between $\tanh$ and the PTSE formulation at different values of precision ($n$ vs $\eta$).

(c) Plot of the $\text{artanh}$ (solid line) function and its PTSE formulation at different values of precision.

(d) Relative Error between $\text{artanh}$ and the PTSE formulation at different values of precision ($n$ vs $\eta$).

(e) Plot of the $\text{arcosh}$ (solid line) function and its PTSE formulation at different values of precision.

(f) Relative Error between $\text{arcosh}$ and the PTSE formulation at different values of precision ($n$ vs $\eta$).
Table 5: Performance comparison results between TSE formulation (ours) and the baselines on the task of textual entailment (SNLI) and noisy prefix detection (PREFIX-K%) in the NLP domain. The columns present the evaluation metric of Accuracy for the different datasets. Best performing cells are highlighted in bold and second best are underlined.

| Models | SNLI | PREFIX-10% | PREFIX-30% | PREFIX-50% |
|--------|------|------------|------------|------------|
| RNN    | 79.34 | 89.62      | 81.71      | 72.10      |
| GRU    | 81.52 |            |            |            |
| HRNN   | 78.21 | 96.91      | 87.25      | 62.94      |
| HGRU   | 81.19 |            |            |            |
| TRNN   | 80.19 | 94.32      | 82.53      | 72.87      |
| TGRU   | 81.41 | 96.98      | 87.40      | 75.85      |

C Simplification of PTSE HNN Components

Let us assume a hyperbolic linear layer $f^\diamond l(x)$ with weights $w$, inputs $x$ and bias $b$ is given by:

$$f^\diamond l(x) = w \odot x \oplus b$$

(16)

Simplifying this using Eqs. (10) and (5), we get

$$f^\diamond l(x) = K_{wx+b} = K_{wx+b}(wx + b)$$

(17)

Here, $wx + b$ is a Euclidean linear layer $f_l(x)$ and $K_{wx+b}$ is a computationally efficient polynomial function. Hence, the hyperbolic linear layer with non-scalable hyperbolic trigonometric functions can be converted to Euclidean equivariants scaled by a polynomial function, using PTSE formulation, which are scalable on GPUs.

D Datasets and Baselines

We select standard datasets, baselines, and evaluation metrics for each domain in the experiments; Graphs (GNNs), Multi-relational reasoning (MuR), Computer Vision (CV), and natural language processing (NLP). The datasets are summarized in Table 1 and described below:

- **Cora** (GNN) [25] contains 2708 publications classified into 7 classes and connected by citation links.
- **Pubmed** (GNN) [27] is collected from the Pubmed database and contains 19717 papers classified into 3 classes and connected by citation links.
- **Citeseer** (GNN) [27] contains 3312 annotated scientific publications classified into 6 classes and connected by citation links.
- **WN18RR** (MuR) [13] is a subset of the WordNet graph that connects words with semantic relations. WN18RR consists of 40,943 entities connected by 11 different semantic relations.
- **FB15k-237** (MuR) [5, 31] contains triple relations from the Freebase entity pairs, with 14,541 entities connected by 237 relations.
- **MiniImageNet** (CV) [26]: MiniImageNet dataset is the subset of ImageNet dataset with 600 examples per 100 of the sampled classes. From the experimental setup in [17], we set the training, validation, and test to consist of 64,16 and 20 classes, respectively.
- **Caltech-UCSD-Birds** (CV) [32]: The CUB dataset consists of 11,788 images annotated with 200 bird classes. From the experimental setup in [17], we set the training, validation, and test to consist of 100, 50, and 50 classes, respectively.
- **SNLI** (NLP) [6]: The SNLI datasets consists of a pair of sentences, the premise and hypothesis, annotated by whether the hypothesis can be inferred from the premise or not. The dataset consists of 570K, 10K, and 10K training, validation, and test pairs, respectively. Following the experimental setup in [14], we use the classes of “contradiction” and “neutral” class as negative samples, and “entailment” as positive samples.
- **PREFIX-K%** (NLP) [14]: According to the methodology provided in [14], we use the K values of 10,30, and 50, where for each sentence, we replace K% of the words of the prefix to get positive samples, and generate a random sentence of the same length as negative samples. With a word vocabulary size of 100, the training, validation, and test dataset contains 500K, 10K, and 10K samples, respectively.

The baseline methods (summarized in Table 1) are described below:
• **GCN (GNN)** [19]: Graph Convolution networks aggregate message from node’s k-hop neighborhoods using learnable convolution filters.

• **GAT (GNN)** [34]: Graph Attention networks utilize attention weights to aggregate messages from the node neighborhood towards the target root node.

• **HGCN (GNN)** [8]: The Hyperbolic Graph Convolution network utilizes hyperbolic gyrovector formulation to aggregate the neighborhood’s hyperbolic vectors at the root node’s local tangent space for further message propagation.

• **HGAT (GNN)** [37]: The Hyperbolic Graph Attention networks utilizes the Lorentz model of hyperbolic space to aggregate node neighborhood in the hyperbolic space using Lorentz factors and Einstein Midpoint.

• **MuRE (MuR)** [3]: Multi-relational Euclidean model learns the representations of knowledge graph entities and relations by formulating the tail entity as a translation of head entity by a relation vector.

• **MuRP (MuR)** [3]: Similar to MuRE, the multi-relation Poincaré model learns the representation of KG entities and relations, but in the hyperbolic Poincaré ball by leveraging the gyrovector operations to formulate the hyperbolic translation of head entity by a relation vector.

• **ProtoNet (CV)** [29]: Focused on the problem of few-shot classification, Prototypical networks learn representations of each class in the training set and learn a metric space for classification on the test set that can be formed by computing distances from the training class representations.

• **H-ProtoNet (CV)** [17]: H-ProtoNet is a hyperbolic variant of the ProtoNet model, formulated by essentially replacing the Euclidean mean operation with hyperbolic averaging (HypAve).

• **RNN (NLP)** [21]: Recurrent Neural Networks are widely used models for sequential encoding that can learn dependencies between different timesteps of sequential inputs.

• **GRU (NLP)** [12]: Gated recurrent networks are an optimization over RNNs to tackle the problem of exploding gradients using the reset gate and update gate.

• **HRNN (NLP)** [14]: Hyperbolic RNNs utilize the hyperbolic linear layer to formulate the forward passes of RNNs and RAdam to perform the backpropagation over time for weight updates.

• **HGRU (NLP)** [14]: Hyperbolic GRUs formulate the reset and update gate of the Euclidean GRUs in the hyperbolic space to learn hierarchical information.

### E Evaluation Metrics

The evaluation metrics for different problems (summarized in Table 1) are described below:

- **Accuracy in node classification**:
  The metric computes the ratio of the number of correct predictions of node classes and the total number of nodes in the test set, i.e.,
  \[
  \text{Accuracy} = \frac{\#\text{number of correctly classified nodes}}{\#\text{total nodes}}
  \]

- **Area under ROC in link prediction**:
  The metric is a quantification of the ability of classifiers to distinguish between classes on the basis of signal and noise. For a binary classification problem, this can be computed as:
  \[
  \text{AU-ROC} = \frac{1 + \text{TPR} - \text{FPR}}{2}, \quad \text{where} \quad \text{TPR} \text{ and } \text{FPR} \text{ are the true positive and false positive rates, respectively.}
  \]

- **HITS@K in multi-relation reasoning**:
  The metric is used to compute the precision of correct results in the the top K results of a query with head entity and relation, i.e.,
  \[
  \text{HITS@K} = \frac{1}{K} \sum_{i=1}^{K} f(e_i), \quad \text{where} \quad f(e_i) = 1, \text{ if } e_i \text{ is a correct result, and } f(e_i) = 0 \text{ otherwise.}
  \]

- **Mean reciprocal rank in multi-relation reasoning**:
  The metric is used to compute the rank of correct results retrieved set of answers, i.e.,
  \[
  \text{MRR} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{f(e_i)}, \quad \text{where} \quad n \text{ is the total number of samples, and } f(e_i) = 1, \text{ if } e_i \text{ is a correct result, and } f(e_i) = 0 \text{ otherwise.}
  \]

- **Accuracy in image classification**:
  The metric computes the ratio of the number of correct predictions of image classes and the total number of samples in the test set, i.e.,
  \[
  \text{Accuracy} = \frac{\#\text{number of correctly classified images}}{\#\text{total size of test set}}
  \]

- **Accuracy in textual entailment and noisy prefix detection**:
  The metric computes the ratio of the number of correctly classified sentence pairs of (premise, hypothesis), for textual entailment, and (noisy prefix, sentence), for noisy prefix detection, and the total number of samples in the test set, i.e.,
  \[
  \text{Accuracy} = \frac{\#\text{number of correctly classified sentence pairs}}{\#\text{total size of test set}}
  \]