Group 1, Team Boba: Encoding Twitter Network in Euclidean vs Hyperbolic Space

Rosa Garza
University of California, Los Angeles
United States
rgarza96@g.ucla.edu

Jessica Ho
University of California, Los Angeles
United States
jessicaho44@g.ucla.edu

Brian Tagle
University of California, Los Angeles
United States
taglebrian@gmail.com

Wenqi Zou
University of California, Los Angeles
United States
wenqizou625@g.ucla.edu

ABSTRACT

Embedding in Euclidean spaces has been extensively studied and applied to many different ML tasks. In contrast, research into hyperbolic embeddings is still new, experimental, and has not been used as widely as Euclidean embeddings. By embedding in a hyperbolic space rather a Euclidean one, the node embeddings retain latent hierarchical structures within the data with smaller dimensionality. In addition to making neural network operations more efficient, working with smaller dimensionality embeddings means 2D projections of the embeddings won't be an uninterpretable mass, as is often the case with the high dimensionality of learned Euclidean embeddings. In this work, we look specifically at a politicians-centered, multi-relational Twitter dataset through a GCN vs HGCN lens.

CCS CONCEPTS

• **Networks** → **Network structure**; • **Information systems** → **Data encoding and canonicalization**.

KEYWORDS

graph convolutional networks, embedding, hyperbolic space

ACM Reference Format:
Rosa Garza, Jessica Ho, Brian Tagle, and Wenqi Zou. 2022. Group 1, Team Boba: Encoding Twitter Network in Euclidean vs Hyperbolic Space . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Hyperbolic embeddings were first introduced by [3]. Informally, hyperbolic space can be thought of as a continuous tree. In [3], the authors explain the benefits of hyperbolic embeddings for modeling hierarchical structure over the more commonly used Euclidean embeddings. Hyperbolic space is one of constant negative curvature and if one considers a circle in this space, the area inside the circle grows exponentially with respect to the radius. In Euclidean space, however, a circle's radius grows quadratically. This difference causes embeddings in Euclidean space to require an arbitrarily large number of dimensions for an accurate representation of hierarchical data in comparison to an equivalent representation in hyperbolic space. The larger number of dimensions in Euclidean space also contributes to distortions in the embedding that do not represent the original data well.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PROBLEM

We hypothesize that Twitter's social network graph contains some hierarchical structure that will allow it to be modeled by a hyperbolic space better than a Euclidean space. For instance, Tweets can mention users other than the Tweet's author and be retweeted by many people. If we view the resulting graph as a tree (or group of trees), we expect the "branching factor" to grow exponentially at each level of a tree. This means that the space of the Tweet embeddings must also grow exponentially in order to avoid losing too much information. Therefore, Tweets and their connections should be better represented by the hyperbolic space, where the volume of a ball in the space increases exponentially.

Euclidean GCN has already proven to perform well on node classification and link prediction on a Twitter dataset in [5], whose details are in Fig. 1. For our experiments, we were only able to run on 3 out of the 4 datasets due to computational and resource limitations. These datasets are PureP, P50, and P20-50. PureP consists only of politicians while P50 is a superset of PureP that also contains a group of Twitter users keen on political affairs which was based on surpassing a number-of-followers threshold. Similarly, P20-50 is a superset of PureP that also contains a group of users moderately interested in politics.

We extend [5]'s model, TIMME, by running different hyperbolic GCNs [1] on the dataset provided by [5]. We should be able to produce comparable results for the aforementioned reasons if the dataset does indeed have a hierarchical nature within it. We also explore visualizations of hyperbolic embeddings, which should provide more clarity on the latent hierarchical structure(s) within [5]'s dataset than potentially high-dimensional learned Euclidean embeddings. Along with these visualizations, we compare distance metrics between disconnected nodes. This is important since one of the primary goals in embedding is to preserve distances between nodes in the new space. We expect the network's leafs to be a

	PureP	P50	P20~50	P+all
# User	583	5,435	12,103	20,811
# Link	122,347	1,593,721	1,976,985	6,496,107
# Labeled User	581	759	961	1,206
# Featured User	579	5,149	11,725	19,418
# Follow-Link	59,073	529,448	158,746	915,438
# Reply-Link	1,451	96,757	121,133	530,598
# Retweet-Link	19,760	311,359	595,030	1,684,023
# Like-Link	14,381	302,571	562,496	1,794,111
# Mention-Link	27,682	353,586	539,580	1,571,937

Table 1: Descriptive statistics of the three selected subsets of our dataset.

Figure 1: Table of [5]'s dataset. Original table by [5].

lot closer together when embedded in Euclidean space than in hyperbolic space, as Fig. 2 details.

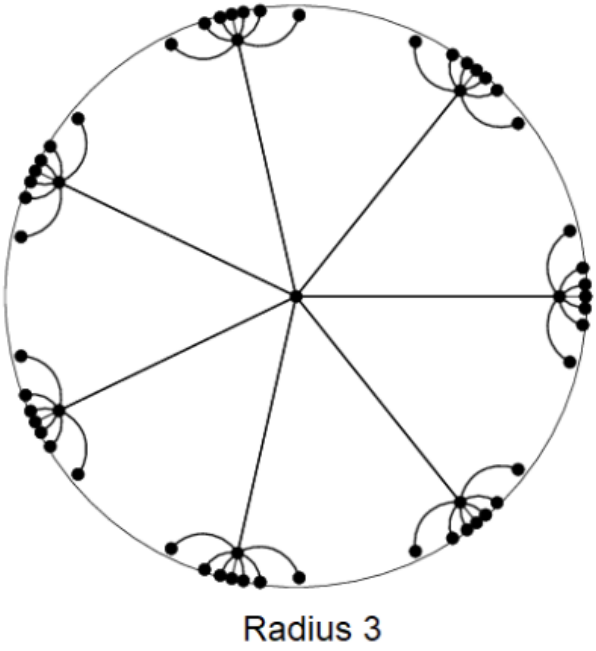

Figure 2: Tree hyperbolic embedding. The leaf nodes are very close to the space's boundary so subtrees are appropriately far apart and neighbors are close. Figure by [4].

Finally, we extend TIMME(-hierarchical), which learns weights for each edge type, by switching out its multi-relational GCN encoder with an HGCN.

3 RELATED WORK

3.1 Poincaré Embedding

[3] introduces the use of the Poincaré ball manifold for learning hyperbolic embeddings. This manifold is chosen because it is compatible with gradient descent based optimizations, which is important for learning an optimal embedding quickly. The Poincaré ball manifold is also unique in the fact that it maps all nodes to a sphere or ball of unit radius. Since hyperbolic space suits tree-like structures, embeddings can be trained at only two dimensions to create an intuitive visualization of a graph's hierarchy. Root nodes of a hierarchy will be near the center of the ball while leaf nodes will lay towards the edges of the ball.

3.2 HGCN

The use of hyperbolic embeddings was expanded upon with the introduction of Hyperbolic Graph Convolutional Neural Networks (HGCN) in [1]. Existing Graph Convolutional Networks (GCN) work on graph structured data and use neighborhood aggregation methods to pass information between nodes. Traditionally, they map the nodes to a Euclidean embedding to get some concrete mathematical insight into the similarities between different nodes. However, as discussed previously, Euclidean embedding can cause large distortions in hierarchical data and [3] only proposed a shallow embedding solution. HGCN, on the other hand, maps a graph's nodes into a hyperbolic embedding at each layer of the neural network before doing aggregation calculations in Euclidean space. Mapping back to Euclidean space is necessary because neighborhood aggregation in hyperbolic space has no closed form solution.) The authors also introduce a trainable curvature at each HGCN layer of a model to make the hyperbolic space better fit the real data. This is in contrast to the constant negative curvature that was used in the Poincaré embeddings from [3].

3.3 TIMME

Operating solely in Euclidean space, TIMME (Fig. 3) aims to predict Twitter users' ideologies by learning embeddings for users in a politicians-centered social network. (This is the same dataset we use for our experiments.) It achieved very high node classification accuracy scores by relying on how nodes were linked rather than the naively learned embeddings of noisy node features (i.e., user descriptions and Tweet contents).

4 METHOD

4.1 Pure HGCN

We added a data loader to [1]'s code for handling the Twitter data. As what will be mentioned in the experiment section, we performed two sets of experiments with [1]'s repo, which are node classification and link prediction. For link prediction, the current [1]'s code with our data loader sufficed. Node classification, on the other hand, can make use of all relations. Therefore we need to be able to produce the final adjacency matrix by combining adjacency matrices of all relations. We decided to do this in two way: a sum of adjacency matrices and a weighted sum of adjacency matrices. In our second approach, we made the weights learnable, in the hope that it would produce better results. To be more specific, we leave the

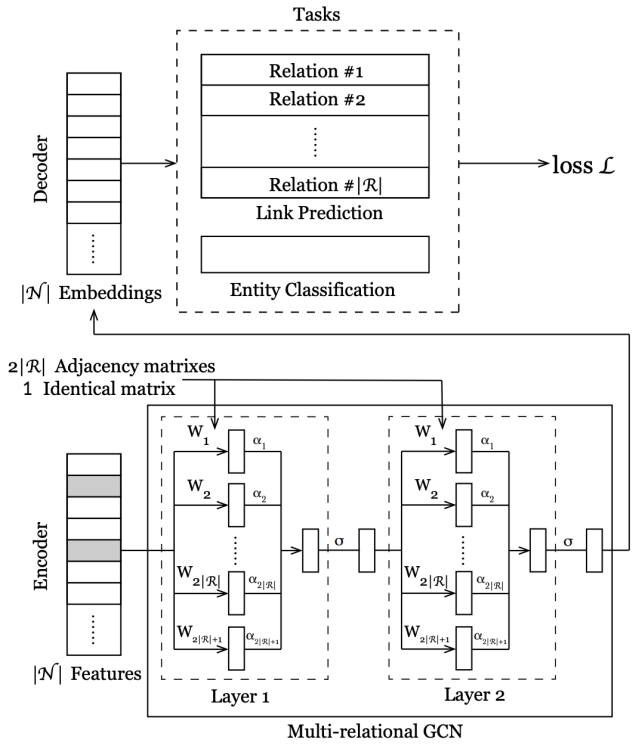

Figure 3: TIMME's general architecture. Its encoder is a multi-relational GCN which spits out learned node embeddings. Its decoder's structure depends on the downstream task but it's ultimately doing ideology detection. Figure by [5].

adjacency matrices for each relation as a list of adjacency matrices while passing them through the layers instead of concatenating them together as the first approach. Then during aggregation, each of the adjacency matrices is multiplied by its corresponding relation weight and all the matrices are added together.

4.2 TIMME+HGCN

Focusing on TIMME-hierarchical (the TIMME variant that learns the importance of each relation in making predictions), we swapped out the multi-relational GCN with an HGCN that projects onto a hyperboloid [2]. Unfortunately, we couldn't enable the HGCN to project onto a Poincaré Ball in time. For some reason, node classification scores would sometimes be nan. To prevent the model from dividing over 0 if it learns that the layers' curvatures should be 0, we clamp curvatures to -0.1. As for node features, we only got the resulting model to use user descriptions as node features; using the identity matrix or Tweet contents led to dimensionality issues which surely could be resolved given more time.

5 EXPERIMENTS

5.1 GCN vs HGCN

For experiments that's based on the [1] repo which contains the base HGCN model, we ran separate link prediction and node classification tasks. We ran link prediction tasks on booth the GCN and HGCN models for 3 types of Twitter relations: friend/follow, mention, and retweet. We selected these 3 relations since they have different degrees of hierarchy, with retweet being most tree-like and mention being the least tree-like. Because of this, we expected a bigger performance gap between the 2 models when performing retweet prediction than when performing mention prediction.

The node classification tasks aimed to predict the political affiliation (i.e., conservative or liberal) of each politician in the dataset. In node classifications, data associated with all relation types were used (in contrast to link prediction, where only one relation type is used). We performed node classification on PureP over 3 different models: GCN, HGCN, and HGCN with learnable weights (Table 1). We neglected to run node classification experiments on the larger datasets for reasons that are explained in the Discussion section. We present and discuss our results in the following subsections.

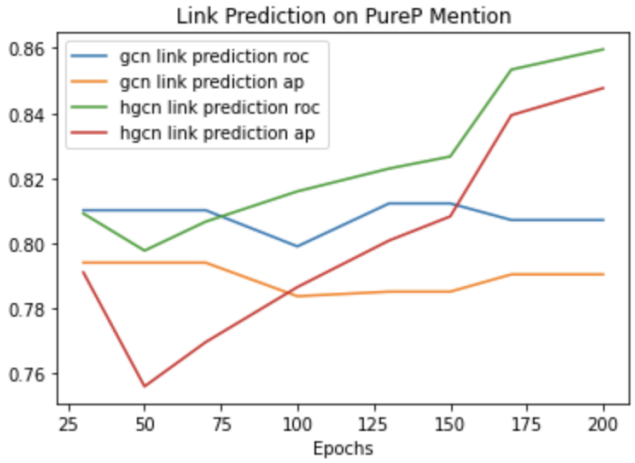

Figure 4: Link Prediction on Mention relation of PureP.

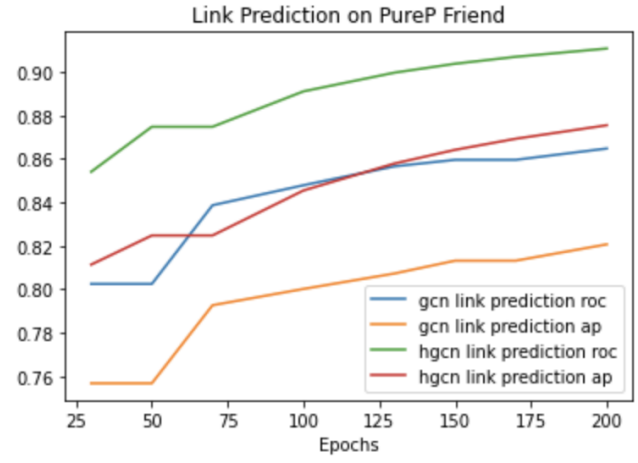

Figure 5: Link Prediction on Friend relation of PureP.

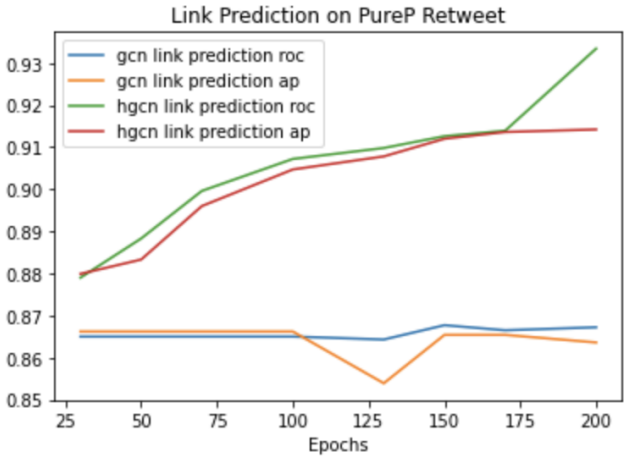

Figure 6: Link Prediction on Retweet relation of PureP.

5.1.1 Link Prediction across Relations. As demonstrated by Figs. 4 and 5, HGCN generally achieves better metrics than GCN. (It should be noted that the best results for either model saturated at around 200 epochs.) In the case of retweet and friend prediction, we observed that HGCN has a significant performance advantage over GCN. In addition, as discussed above, we expected the performance gap between HGCN and GCN to be bigger for retweet prediction than for mention prediction; Figs. 4 and 6 align with our expectation. One thing to note is that, during mention prediction (Fig. 4), GCN performed better than HGCN when the number of epochs was low. We think there are two possible reasons for this. Firstly, the mention relation is less hierarchical, so HGCN might need more of a warm-up period during training. Secondly, the model's loss function is naturally still unstable when the number of epochs is low.

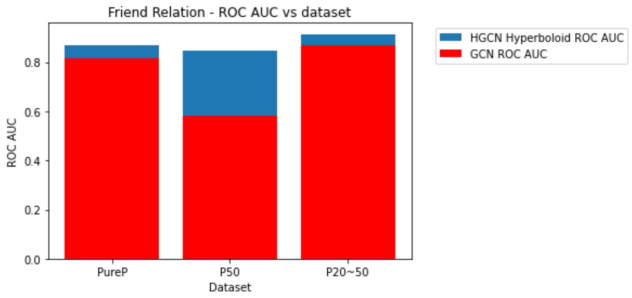

Figure 7: Link Prediction on Friend relation of Different Datasets.

5.1.2 Link Prediction across Datasets. In addition to performing link prediction on different relations, we also performed link prediction over different datasets. This helped us determine the effect of input size on HGCN and GCN performances. The datasets that we used were PureP, P50 and P20-50, and the relations we experimented with were "friend"/"follow" and "mention." According to

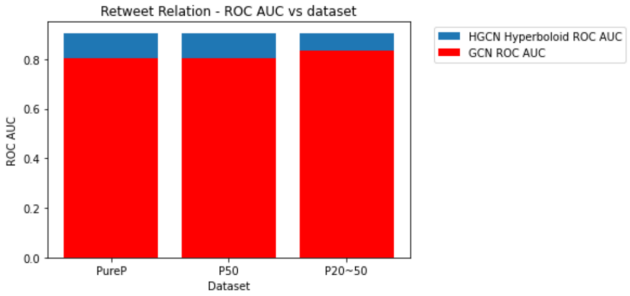

Figure 8: Link Prediction on Retweet relation of Different Datasets.

Model	GCN	Original **HGCN**	**HGCN** with learnable weights
Accuracy	0.9522	0.9449	0.8971
F1-score	0.9517	0.9455	0.8971

Table 1: Node classification accuracy across tested models on PureP.

Figs. 7 and 8, the performance differences between HGCN and GCN across different datasets were pretty minor. We believe this can be attributed to the fact that there are many overlapping nodes between the datasets.

5.1.3 Node Classification. We performed classification on PureP nodes with 3 different models: GCN, HGCN, and HGCN with learnable weights. GCN performed the best, followed by HGCN, then HGCN with learnable weights (Table 1). The performance differences between GCN and HGCN were fairly negligible. Perhaps HGCN with learnable weights performed the worst because the weights were being used naively in aggregation (i.e., it's just used in a weighted summation with corresponding adjacency matrices) which might have led to underfitting.

5.1.4 Measuring Embedding Spaces. As our project focuses on comparing the performance of modeling a Twitter data set when using a Euclidean space versus a hyperbolic space, we decided to look further into the embedding spaces themselves. Because the goal is to model a hierarchical data set with an embedding space of minimal dimensions, we decided to calculate the distance between the embeddings of two disconnected nodes in order to get a better understanding of how large the embedding spaces are and how the embeddings are arranged in the different spaces. Due to the differences in geometry between the Euclidean space and hyperbolic spaces, distances between nodes are calculated with the following equations.

For the Euclidean space, we calculated the distance between two nodes (X and Y) with the Euclidean distance formula.

$$d(X, Y) = (X - Y)^2$$

With the Hyperbolic space, we experimented with two different models for the space: hyperboloid and Poincaré ball. For the hyperboloid embeddings, we used the following equations:

$$d(X,Y)_L^K = \sqrt{K}(\text{arcosh} - \langle X\ ,\ Y\rangle_L / K)$$

For this calculation, we are taking the square root of K which is the inverse of the negative curvature. As for $-\langle X\ ,\ Y\rangle_L$, in this calculation we are taking the Minkowski inner product where $\langle X\ ,\ Y\rangle_L := -x_0y_0 + x_1y_1 + ... + x_dy_d$. With the inner product in Hyperbolic geometry, the hyperboloid model is defined as the Riemannian manifold. Due to this being a hyperbolic space, it is essentially calculating the shortest path in the manifold.

As for the Poincaré ball embeddings, we calculated the distance between two nodes with the following distance formula:

$$d(X,Y) = \text{arcosh}\,(1 + 2 * \frac{\|X-Y\|^2}{(\|1-X\|^2)(\|1-Y\|^2)})$$

In this calculation, we are essentially taking ArcCosh of 1 plus 2 times the quotient of the norm squared between the points X and Y and the result of the norm squared of $1-X$ times the norm squared of $1-Y$. The Poincaré ball manifold is well suited for gradient-based optimization [3] so this embedding space was also used and analyzed.

When further analyzing the embedding spaces, we looked at the distances between disconnected nodes in each space for each task: node classification and link prediction. Refer to Table 2 in order to see the distances calculated for each embedding space focused on the node classification task.

Embedding	Euclidean	Hyperboloid	Poincaré Ball
Minimum	1000	0.0	0.0
Maximum	nan	2.1913490	0.2274289
Average	nan	1.0034512	0.067873

Table 2: Distance between the embeddings of two disconnected node learned while training for node classification task over PureP.

Based on Table 2, we initially thought the Euclidean space is larger than the hyperbolic embedding spaces, which isn't what we were expecting. The reverse should hold true for the same number of dimensions (16) in all spaces since a hyperbolic space grows far faster than Euclidean space, as explained before. However, we can't look at the absolute values of these metrics since we didn't normalize our distance calculations and distance is defined depending on the embedding space. That is, 1000 units doesn't hold the same meaning in Euclidean space as in hyperbolic space.

The minimum distance of the Euclidean space between two nodes disconnected was 1000 in comparison to the hyperboloid and Poincaré Ball models for hyperbolic space, where they were 0. The 0s in the hyperbolic spaces make sense since the model should learn that nodes with many children should be root nodes and therefore at the center of the hyperbolic space. A minimum distance between 2 disconnected node embeddings indicates that there are multiple root nodes, which aligns with our datasets' context; the root nodes are likely popular politicians.

Another interesting observation lies in the average distance metrics. For Euclidean space, the average distance between 2 disconnected nodes was nan. The nan could be due to the fact that all the pairwise distances between disconnected nodes ended up being too large which led to a sort of precision error.

Lastly, we looked at the embeddings learned while training for link prediction (Table 3). Interestingly, all of the spaces' minimum distances were close to 0. In fact, the all distance metrics in Euclidean space were on a comparable scale with their counterparts in the hyperbolic spaces. Perhaps this is an indication that performing link prediction does a better job of representing the hierarchical nature of the data.

Embedding	Euclidean	Hyperboloid	Poincaré Ball
Minimum	1.821796e-05	0.0	1.684919e-05
Maximum	0.2983254	9.0161524	0.0826985
Average	0.0120677	5.2032113	0.0032204

Table 3: Distance between the embeddings of two disconnected node learned while training for link prediction task over PureP.

When comparing the results of the distances between the two spaces, we can get an idea of how using hyperbolic spaces is more efficient (based on the scale of our distance metrics in hyperbolic vs Euclidean space in Table 2) and greatly reduces the embedding space dimensions. That being said, it seems that the downstream task can greatly change the learned embeddings. For the node classification task, embedding in either model of hyperbolic space is better than embedding in Euclidean space. But for the link prediction task, all of manifolds seemed to do a good job of representing the data, given their comparable absolute value scales and how they clumped root nodes near the center of their spaces.

5.2 TIMME+HGCN

With the following parameters,

- epochs: 50
- number of hidden nodes: 16
- layers: 2
- bias: None
- node features: user descriptions

(and other default values), we trained a TIMME+HGCN model to do ideology detection while learning each relation's importance.

While TIMME+HGCN seems to do very well on PureP (Fig. 9), we note how volatile the accuracy and f1-score are after 15 epochs. This volatility is exacerbated over the bigger datasets. In Fig. 10, the accuracy and f1-score even steadily drop after 22 epochs. Pivoting to the link prediction results (which happen in the process of ideology detection so that the model may learn on a denser graph), Figs. 11 and 12 are representative of our results on all the datasets. We either get very high results that are comparable to TIMME-hierarchical's or we get a sudden dip in performance after some number of epochs.

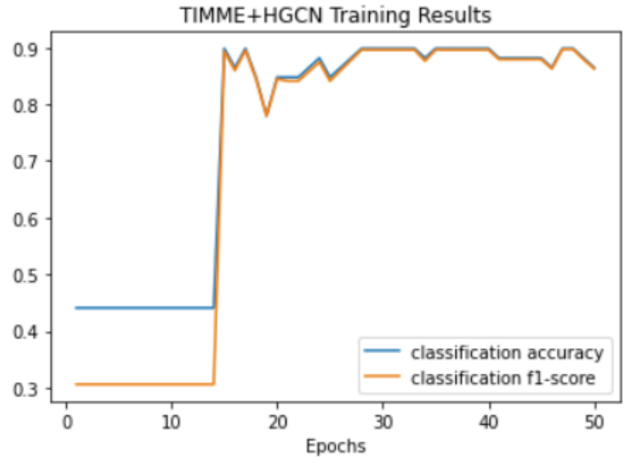

Figure 9: TIMME+HGCN's accuracy and f1-score on PureP.

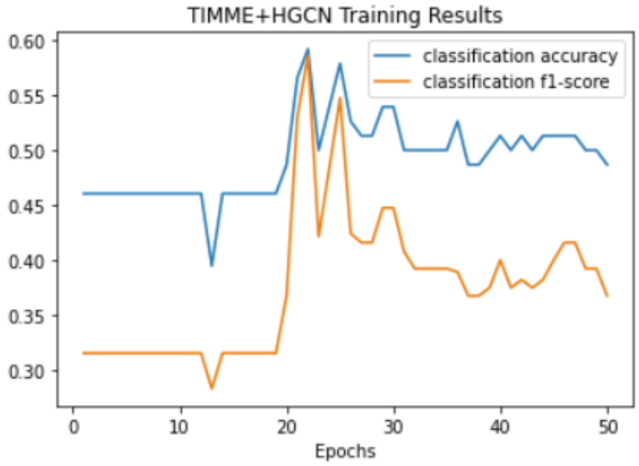

Figure 10: TIMME+HGCN's accuracy and f1-score on P50.

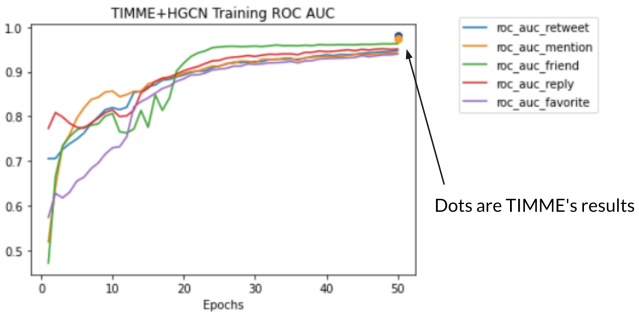

Figure 11: TIMME+HGCN's ROC AUC for each relation on P50.

To be clear, TIMME-hierarchical was able to achieve high results after just a few epochs; we placed their published results at 50 epochs because that's when our model would theoretically have converged by.

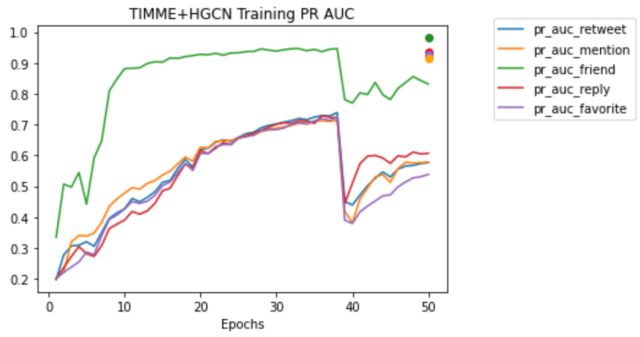

Figure 12: TIMME+HGCN's PR AUC for each relation on P20-50.

During training, the TIMME(-hierarchical) part of TIMME+HGCN learns the importance of each relation. The original TIMME-hierarchical almost consistently learned that the "friend" relation is most important when making predictions. Our TIMME+HGCN, on the other hand, did not consistently learn that any one relation is the most/least important (Fig. 13). That being said, all the values are very close to 2.0 (uniform distribution), including in [5]. So all the noisy perturbations from convergence could be preventing us from getting consistent results across datasets.

Retweet	Mention	Friend	Reply	Favorite	
0.20184161	0.20248325	0.19916178	0.19685115	0.19966224	PureP
Retweet	Mention	Friend	Reply	Favorite	
0.20042942	0.20015903	0.19952548	0.19922341	0.20066258	P50
Retweet	Mention	Friend	Reply	Favorite	
0.20023437	0.20009421	0.20013104	0.20041192	0.1991285	P20~50

Figure 13: TIMME+HGCN's learned relation importance.

6 DISCUSSION

We have met both success and failure. Our distance metrics (between the embeddings learned by GCN vs plain HGCN for 2 disconnected nodes) don't really give us much of an idea of how big the embedding spaces are since distance is defined differently depending on the space. But [4] mentions how we could normalize our distances by the sum of each node's distance to the origin. So that's something to look to in a future work.

For individual link prediction and node classification tasks, plain HGCN matches or outperforms all versions of the TIMME model on the smaller PureP dataset. However on the larger datasets, the vanilla HGCN model's performance stays around the same while the accuracy of the original TIMME-hierarchical model sees an extreme performance jump to nearly 100% accuracy on all tasks. Based on these results, it is clear that TIMME's focus on learning from links and its multitask (i.e., link prediction and node classification tasks)

learning approach are superior to the individual task approach that's used in vanilla HGCN.

However, it's still worth noting that plain HGCN performs better than TIMME on the smallest dataset. This is likely because there is an innate structure in a dataset of pure politicians that's more hierarchical than in a dataset that's diluted with common users. There are clearly politicians that are more popular and influential simply based on the fact that there is an existing hierarchy in American government and it is clearly mirrored in the Twitter data, which can be seen by our visualizations (Fig. 14). However, there is a clear breakdown of the hierarchy in the larger datasets. Although the politicians still have the same hierarchy, the other additional Twitter users do nothing but distort that.

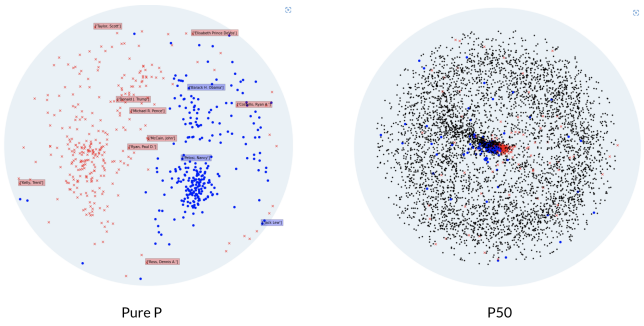

Figure 14: Plain HGCN's learned hyperbolic node embeddings for PureP and P50, projected to 2D. Each dot is a node embedding. Blue dots correspond to politicians labeled as liberal. Red dots correspond to politicians labeled as conversative. Black dots are unlabeled nodes (i.e., users with at least some interest in politics).

We don't perform the node classification task on P50 or P20-50 with any versions of the HGCN model. This is because those datasets only have ground truth labels for politicians. So the results would be identical to the PureP node classification task since the unlabeled nodes wouldn't contribute to the loss function and would be arbitrarily placed in the embedding space.

As for TIMME+HGCN, while it achieves fairly high classification metrics on PureP, the volatility of those metrics on all tested datasets is disconcerting. Perhaps projecting onto the Poincaré Ball model rather the hyperboloid model will yield a more stable convergence. That being said, the big dip in Fig. 12 implies that the model was learning that a Euclidean space is the more appropriate embedding space. At least, this seems to be the case for the bigger datasets, where hierarchical structures are diluted by all the non-politicians. Another potential flaw is that we only got TIMME+HGCN to work on users' Twitter descriptions for input, which is very noisy. Since we're leveraging TIMME's system of relying on graph connectivity, we should try to get TIMME+HGCN to accept the identity matrix as its nodes' features. That way, the model focuses on graph connectivity to make predictions and ignores noisy and naively-learned text.

REFERENCES

[1] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. 2019. Hyperbolic Graph Convolutional Neural Networks. https://doi.org/10.48550/ARXIV.1910.12933
[2] Brian Keng. 2018. Hyperbolic Geometry and Poincaré Embeddings. (jun 2018). https://bjlkeng.github.io/posts/hyperbolic-geometry-and-poincare-embeddings/
[3] Maximillian Nickel and Douwe Kiela. 2017. Poincaré Embeddings for Learning Hierarchical Representations. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/59dfa2df42d9e3d41f5b02bfc32229dd-Paper.pdf
[4] Frederic Sala, Chris De Sa, Albert Gu, and Christopher Ré. 2018. Representation tradeoffs for hyperbolic embeddings. In *International conference on machine learning*. PMLR, 4460–4469.
[5] Zhiping Xiao, Weiping Song, Haoyan Xu, Zhicheng Ren, and Yizhou Sun. 2020. TIMME. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM. https://doi.org/10.1145/3394486.3403275