Phase 1 - Experiment Findings

Introduction

For each dataset, I ran a series of “stress tests” similar to those in the original paper. While my results table doesn’t look exactly like Table 1 from the paper, it contains the same core information. I trained Poincaré embeddings using the same dimensional settings and tested them across different configurations. Then, I recorded both link prediction and reconstruction metrics.

Reconstruction measures how well the embedding captures the structure of the original training data, essentially showing how much information is preserved. It reflects the embedding’s ability to represent the data accurately.

Link prediction, on the other hand, tests how well the embedding generalizes by predicting held-out links based on what it learned during training. It evaluates whether the embedding can infer missing relationships in the graph or hierarchy.

To measure performance, I used Mean Rank (MR) and Mean Average Precision (MAP**)**. Mean Rank tells me the average position of the correct predictions in the ranked list—lower is better because it means the right answers show up closer to the top. MAP looks at both precision and relevance across ranks, so a higher MAP means the embedding consistently ranks the important predictions higher.

In all experiments I used a 0.8/0.2 train/test split and 100 epochs, as my preliminary experimental evaluations of metrics and runtime indicated it was optimal. I tested Poincare embeddings dimensions of 5,10,20,50,100 and 200 because these were the dimensions evaluated by the authors. Additionally, exploring a range of dimensions allows me to assess how different embedding sizes impact the model’s ability to capture data complexity. For each experiment I tested 3 levels of “stress tests” to see how dimensionality and stress levels affect the Poincare embedding. For each experiment I ran 6*3 = 18 models (6 different dimensions and 3 levels of stress tests). I ran a total of 3 experiments, so a total of 6*3*3=54 models. For every experiment I give a short explanation about the dataset used and about the technique / “stress test”.

Information about the datasets:

Experiment 1 – Data can be accessed through nltk library in python

Experiment 2 – Data can be accessed https://snap.stanford.edu/data/egonets-Facebook.html

Experiment 3 – I provide the dataset and the R code used to create it.

Experiment 1 – Mammals Wordnet dataset and edge flipping

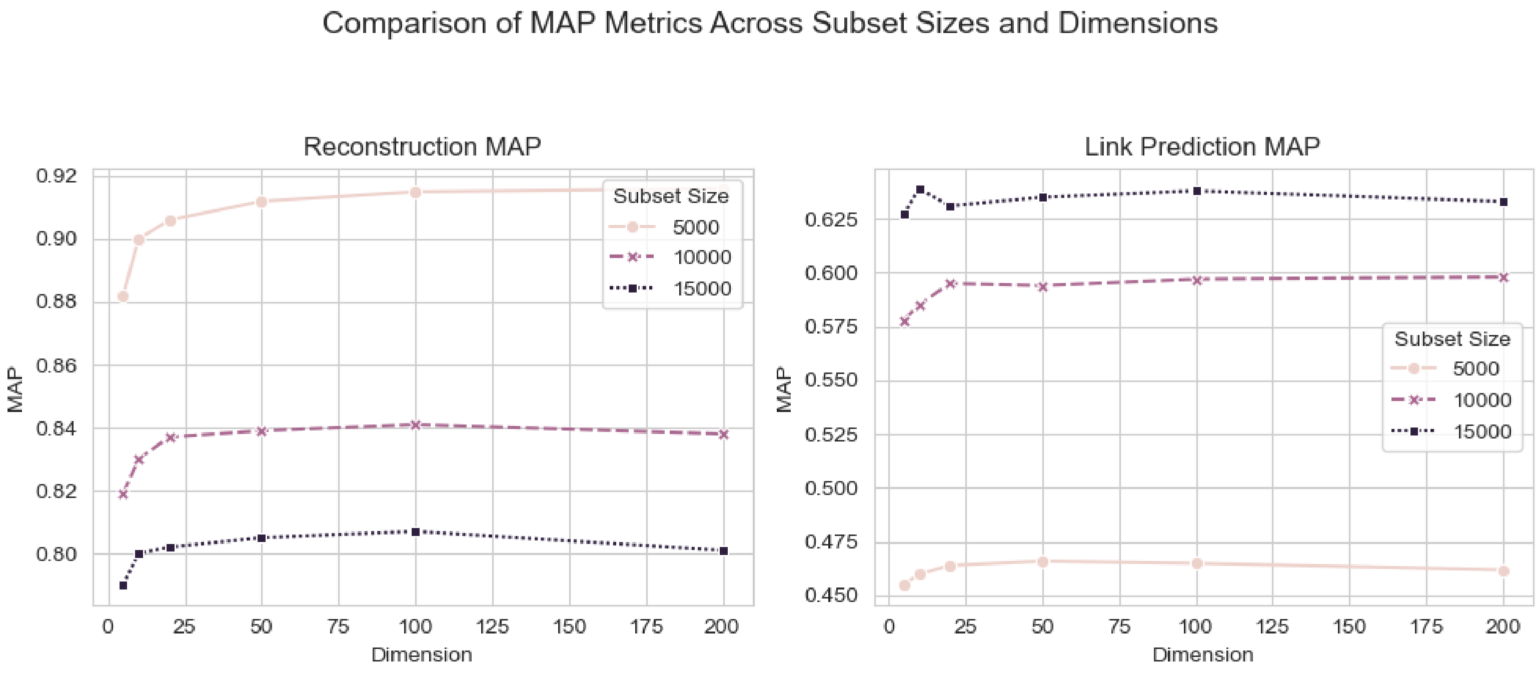
The mammals wordnet dataset is a subset of the popular wordnet hierarchy. It is a lexical database of semantic relations between words that links words into semantic relationships including synonyms, hyponyms and meronyms. It was developed by Princeton university and initially released in the mid-1980s. I decided to use this dataset because it is used in the paper (Figure 2) and we wanted to see how my run fares compared to the paper's authors. Additionally, I decided to use this dataset because it made the training feasible, at first I tried to run the experiments on the whole WordNet Hierarchy but it took forever. This dataset includes hierarchical relationships that represent the biological taxonomy of mammals. For example "canine" is a subset of "carnivore", which is a subset of "mammals" etc. The dataset forms a directed acyclic graph, where concepts are connected by relationships such as "is-a" / "is a kind of" / "is a part of" etc. I built a transitive closure for the mammal's subtree, which ended up having a total of 7051 hypernym pairs (edges). The stress test I decided to employ for this dataset was edge flipping. I wanted to simulate a real-world scenario where data is not always perfect, and sometimes labels might be reversed. For every stress test, I assigned a different flipping probability (p=0,0.1,0.3) on the training set. This causes a fraction of edges to get reversed, and this means that a child and ancestor swap roles, effectively creating incorrect edges. The following table displays the findings:

Embedding Dimension	Flip Probability	Reconstruction RANK	Reconstruction MAP	Link Prediction RANK	Link Prediction MAP
5	0	1.026	0.988	1.395	0.881
	0.1	1.122	0.953	1.83	0.781
	0.3	1.542	0.831	3.445	0.53
10	0	1.026	0.988	1.369	0.889
	0.1	1.112	0.957	1.707	0.804
	0.3	1.525	0.841	3.24	0.556
20	0	1.026	0.988	1.339	0.897
	0.1	1.115	0.955	1.722	0.795
	0.3	1.513	0.844	3.186	0.56
50	0	1.025	0.989	1.339	0.894
	0.1	1.107	0.958	1.665	0.809
	0.3	1.506	0.845	3.174	0.57
100	0	1.026	0.988	1.347	0.894
	0.1	1.111	0.957	1.659	0.809
	0.3	1.511	0.844	3.188	0.572
200	0	1.024	0.989	1.357	0.89
	0.1	1.108	0.958	1.724	0.8
	0.3	1.496	0.848	3.169	0.571

The most striking finding, in my opinion, is how little dimension impacts reconstruction metrics when there is no noise: even relatively small embeddings reconstruct the original data almost as well as much larger ones. In contrast, link prediction appears somewhat more sensitive to dimensionality, showing modest improvements as dimensions grow—from 5 up to around 20 or 50—and then plateauing or dipping slightly. Another key takeaway is the effect of noise injection: as flip probability increases (to 0.1 and especially 0.3), both reconstruction and link prediction metrics deteriorate markedly, this shows us how introducing flipped edges makes it harder for the Poincare embedding to learn consistent hierarchical structures. When looking at Dimension 5, and no flip probability (unaltered data) I get similar results to what the authors got in Figure 2.

Experiment 2 – Facebook Social circles and subset incrementation

The second dataset I decided to use for this second experiment was the social circles: Facebook dataset. This dataset has friends lists from Facebook, from 4039 users. This dataset captures the friendship connections between users and edges represent mutual friendships between users between the users’ networks. In this case, all friendship links in the dataset are bidirectional, and do not have an “is a” relationship like the dataset in experiment 1. In other words, the relationships are symmetric, if an edge exist between user X and Y then it implies the same relationship exists between Y and X. There is a total of 88234 edges in the dataset, I decided to “stress test” the Poincare embedding by using 3 different subset sizes of 5000,10000 and 15000(so different number of edges essentially). This test allows us to evaluate the robustness of the Poincare embedding across varying data volumes. By analyzing how the embedding performs with smaller, medium and larger subsets, we can identify strengths and limitations in capturing the underlying structures



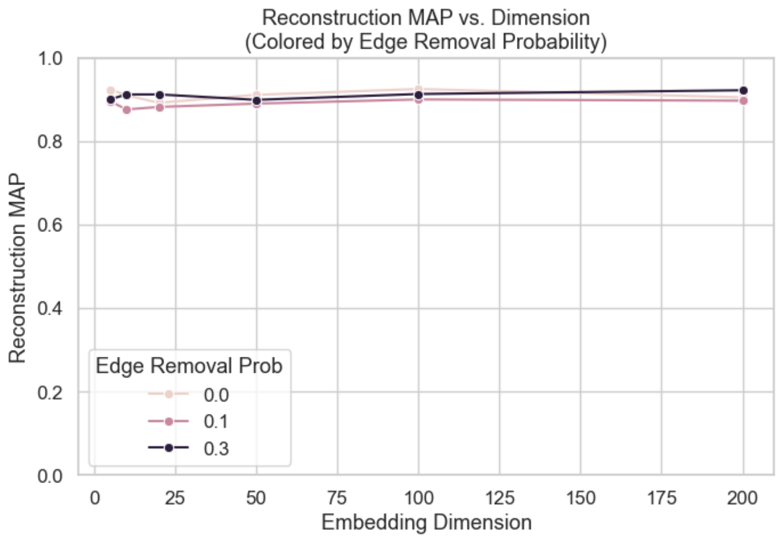
which are inherent in the social network.

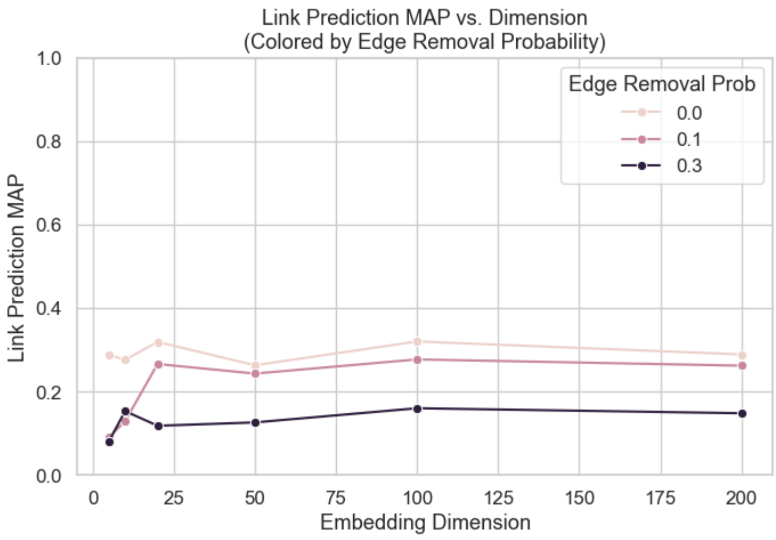
The results show a clear interplay between subset size, embedding dimension, and the performance metrics. When the subset size is smaller, reconstruction is relatively easy, since the network is simpler, leading to low rank and high MAP scores, but link prediction suffers due to limited data. As the subset size increases to 10,000 and 15,000 edges, reconstruction becomes more challenging, shown by slightly worse rank and MAP, but link prediction performance improves by a lot. This reveals a trade-off where memorizing a smaller network is easier, yet learning broader patterns for predicting new links is facilitated by more data. Increasing the embedding dimension from 5 to 20 usually gives us decent gains in both reconstruction and link prediction, as the model can capture more nuanced relational structures. Dimensions beyond 50 or 100 provide slight improvements, indicating diminishing returns. Notice that link prediction MAP jumps significantly when moving to larger subsets, suggesting that the Poincare embedding benefits from the additional data. Reconstruction MAP remains relatively high across all subsets but is ultimately constrained by the complexity of larger networks. For completeness I add the table which includes all of the metrics.

Subset Size	Embedding Dimension	Reconstruction RANK	Reconstruction MAP	Link Prediction RANK	Link Prediction MAP
5000	5	1.421	0.882	7.15	0.455
	10	1.333	0.9	6.829	0.46
	20	1.316	0.906	6.723	0.464
	50	1.296	0.912	6.584	0.466
	100	1.291	0.915	6.637	0.465
	200	1.296	0.916	6.742	0.462
10000	5	1.655	0.819	3.818	0.578
	10	1.631	0.83	3.914	0.585
	20	1.606	0.837	3.778	0.595
	50	1.599	0.839	3.818	0.594
	100	1.58	0.841	3.801	0.597
	200	1.592	0.838	3.861	0.598
15000	5	1.848	0.79	3.263	0.627
	10	1.766	0.8	3.217	0.639
	20	1.782	0.802	3.272	0.631
	50	1.75	0.805	3.206	0.635
	100	1.746	0.807	3.263	0.638
	200	1.765	0.801	3.304	0.633

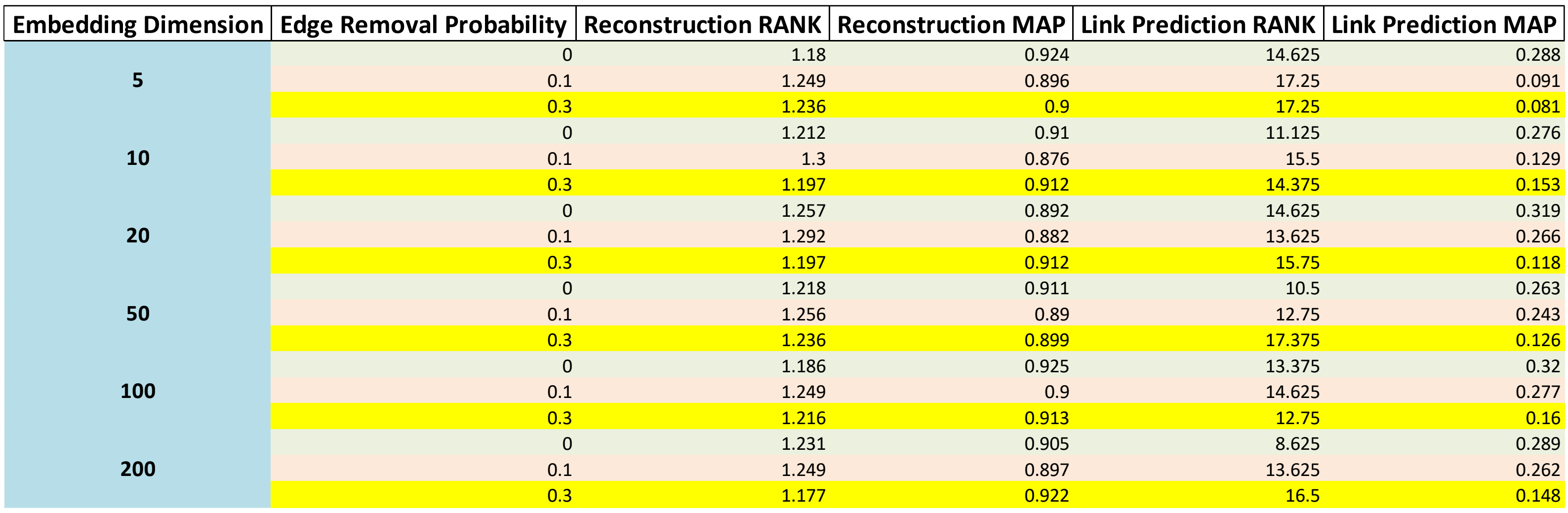
Experiment 3 – HebrewNet and Probabilistic Edge Removal

For the third experiment I used the hebrewnet dataset. It was created by Noam Ordan, Iris Eyal and Shuly Wintner from Haifa University. To work with the data, I focused on the two SQL files that were relevant for evaluating the hypernym-hyponym relationships for a given synset (hebrew_synset.sql to match between synsets and words and common_relation.sql to extract the hypernym-hyponym type of relations). I ensure that those synsets in hebrew_synset.sql that under the ‘word’ column were matched to “GAP!” (meaning they do not have a corresponding lexical unit in the language) were not taken into consideration at the moment of building the transitive closure since their appearance could cause hypernyms-hyponyms relationships to be misleading or incomplete, as they are not associated with real words, and my main purpose was to work with a dataset that reflects real words and their hierarchical connections. The dataset I created is based on the transitive closure of ״חפץ אובייקט״, The dataset has a total of 196 edges. Even though the dataset is very small, I believe it is still an interesting experiment since it can help us assess the Poincare embedding’s ability to capture hierarchical structures with limited data and provides insights into generalization and overfitting tendencies in such cases. Additionally, it demonstrates the potential of hyperbolic embeddings in low resource scenarios when data is scarce. For this experiment I decided to employ probabilistic edge removal. This experiment tests to see how removing connections between words in a hierarchy affect the ability of the Poincare embeddings to learn and represent the relationships. The edge removal is done randomly, based on a predetermined removal probability, in my case, I chose 0,0.1 and 0.3. I think that testing this edge removal method, on an already small dataset, makes sense because it allows us to examine the embedding’s robustness in extreme data-scarce conditions, like in the real world, where every single connection or edge might be very significant and important in defining the hierarchy. Here are the findings:





First of all, it is relatively easy to see that edge removal negatively impacts Link Prediction performance. We can see that for example when we have Embedding dimension of 5, Link prediction MAP drops from 0.288(p=0) to 0.081(p=0.3). Across dimensions, we can see that for lower dimensions(5,10) there’s a pretty big performance drop with edge removal. When looking at dimensions such as 20,50,100,200 we see that they have a similar performance between removal probability of 0 and 0.1. this suggests that higher dimensions can somehow fight against the low edge removal probability. On the other hand, we can see that if the removal probability is 0.3, there’s a drastic drop no matter the dimensionality, for link prediction. Note that Reconstruction is relatively stable across dimensions. In fact, the reconstruction MAP is relatively high, even with edge removal, which indicates that the model can memorize training edges in a way. We could say that at higher dimensions, having less edges (and higher removal probability, like 0.3) makes reconstruction easier for the Poincare embedding, as the embedding needs to do less for reconstruction. Link prediction is more sensitive, as I wrote above, dropping sharply with 0.3 probability of edge removal. I can conclude that a good Reconstruction MAP does not guarantee a good Link Prediction, which means that we apparently need enough edges to have some decent predictive performance. Again, I add all of the metrics as an excel table for completeness:



Embedding Dimension	Edge Removal Probability	Reconstruction RANK	Reconstruction MAP	Link Prediction RANK	Link Prediction MAP
5	0	1.18	0.924	14.625	0.288
	0.1	1.249	0.896	17.25	0.091
	0.3	1.236	0.9	17.25	0.081
10	0	1.212	0.91	11.125	0.276
	0.1	1.3	0.876	15.5	0.129
	0.3	1.197	0.912	14.375	0.153
20	0	1.257	0.892	14.625	0.319
	0.1	1.292	0.882	13.625	0.266
	0.3	1.197	0.912	15.75	0.118
50	0	1.218	0.911	10.5	0.263
	0.1	1.256	0.89	12.75	0.243
	0.3	1.236	0.899	17.375	0.126
100	0	1.186	0.925	13.375	0.32
	0.1	1.249	0.9	14.625	0.277
	0.3	1.216	0.913	12.75	0.16
200	0	1.231	0.905	8.625	0.289
	0.1	1.249	0.897	13.625	0.262
	0.3	1.177	0.922	16.5	0.148

Phase 2 – Customizing the model

In this phase I decided to perform a few changes to the original algorithm in order to hopefully make it work better on different types of data. The main idea here is to create a learnable curvature with a stable negative reparametrisation. This is a little bit different than what's defined in the paper, where we have a fixed k=-1 assumption. I decided to let the embedding space become more, or less, hyperbolic depending on the data. In the original paper, we have a fixed curvature of k=-1, so the embedding is a Poincare ball with a constant negative curvature. I implemented a trainable curvature via a reparametrisation of k, such that

k = -exp(α), where I keep α in the model and learn it by means of gradient descent. So basically, what I do is I can learn how to change and adapt to the specific dataset.

More clearly, in the original paper, we have the distance function:

$$d(u,v) = \operatorname{arcosh}\left(1 + 2\frac{\|u-v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}\right)$$

(eq 1 in the paper)

Now change it to be

$$\underset{\text{new}}{d(u,v)} = \operatorname{arcosh}\left(1 + 2\frac{\|u-v\|^2}{(1+\mathbf{K}\,\|u\|^2)\,(1+\mathbf{K}\,\|v\|^2)}\right)$$

I had to define k = -exp(α) because without that, I had a lot of nans and technical problems, and also we wanted to ensure that k is always negative. Also, although k = -exp(α) ensures the negativity, I still clamp the denominators to avoid numerical blow-ups if norms approach the boundary.

Again, we have to get a little bit technical with the code, but the idea here is to add k to the equation, learn it through alpha, and then keep on going. We gain some benefits, which include learning automatically the required hyperbolicity that best fits the current data being used. I argue that this is very useful in real-world datasets and hierarchies, where things are either shallows or steeper than the presupposed k=-1. I also gain the benefits of better performance, as different datasets have different ways of being "tree like". By letting K being learnable, I am reducing the risk enforcing a "geometric prior", so we are less of assess by assuming things about the data (assuming makes an ass of u and me!) – I hoped that this would give better link prediction & reconstruction metrics. Also, the change was relatively easy, I used "Monkey patching" in my code to make it work. Another thing is that autograd makes it very easy to compute the derivatives w.r.t alpha etc. On the other hand, the runtime was a lot longer than the original experiments – even though we kept all the parameters (epochs, train /test split, etc) like what I did in Phase 1. In this Phase I perform the exact same experiments, with the same data as we did in Phase 1, the difference is the change in the algorithms. By the way, If you want to see the K value we ended up with for every sub-experiment, please see the Jupyter notebooks. The variable is called kappa. Additionally, for every sub-experiment I initiate alpha to be 0, and set a learning rate of 0.001.

Experiment 1 – Mammals Wordnet dataset and edge flipping

Here are the results:

Embedding Dimension	Flip Probability	Reconstruction Rank	Reconstruction MAP	Link Prediction Rank	Link Prediction MAP
5	0	4.777	0.416	5.341	0.374
	0.1	7.29	0.342	6.697	0.342
	0.3	8.265	0.351	8.299	0.32
10	0	3.634	0.481	3.939	0.441
	0.1	6.109	0.437	5.791	0.411
	0.3	7.739	0.382	6.872	0.37
20	0	3.147	0.521	3.533	0.479
	0.1	6.585	0.426	5.946	0.409
	0.3	8.592	0.376	8.131	0.35
50	0	4.14	0.707	4.179	0.672
	0.1	6.39	0.445	5.892	0.42
	0.3	8.487	0.375	8.146	0.35
100	0	3.057	0.535	3.313	0.509
	0.1	5.716	0.501	5.23	0.476
	0.3	7.994	0.374	7.376	0.35
200	0	3.513	0.677	3.729	0.645
	0.1	6.236	0.472	6.045	0.444
	0.3	7.973	0.376	7.01	0.374

It is easy to see that the flip probability degrades the overall performance. As flip probability increases, all metrics in all cases seem to worsen. It could be argued that this is kind of expected, as reversing the edges in a hierarchy introduces contradictions that make both Link Prediction and Reconstruction harder. It also seems that dimensionality helps, but it does not seem to be helping in a monotonic way. What I mean by that is that it is not necessary that a higher dimension is better. In general, going from 5D to 50D tends to raise MAP, especially at the p=0 or p=0.1. However, there are exceptions. For example, at p=0.3, 10D does better than all other dimension in terms of reconstruction, while 200D is best at link prediction. In other words, there is not a single best dimension for all scenarios, there are some tradeoffs to be had at different noise levels. In short, I conclude that the altered algorithm is not that good, atleast when it comes to this specific experiment. Firstly, we see a consistent drop in performance with rising flip probability. Secondly, we see that that there's some dimensionality tradeoff. Medium – high dimensions often work relatively well at lower or moderate noise levels, but at very high noise the best might be smaller(10D) this depends on if we care about Link prediction or reconstruction.

<u>Comparison with Phase 1's Results:</u>

In short, the original algorithm consistently achieves better metrics for both link prediction and reconstruction, under every tested dimension and noise level. In the original Poincare, the ranks are typically in the 1-2 range, with very high MAP values, often above 0.8, with p=0. Even with p=0.3 the MAP scores often stay above 0.5. on the other hand, in the new model with trainable curvature, the ranks are significantly higher, all higher than 3, and we even have times where the rank is higher than 7 (when the noise is high). In short, at every dimension and noise level, the trainable curvature model I created shows worse ranks and MAPS, and this is sadly true across the board. At lower dimensions, such as 5 or 10, there is a very large gap between the trainable K and the original embedding. For example, at dimension 5 and no noise, the reconstruction map is about 0.99 in the original model, while in the updated model the reconstruction map is about 0.42. This is a huge gap between the models, and there is a similar gap at dimension 10. At 200D and p=0, the original model has a reconstruction of around 0.99 while the updated model has a reconstruction of around 0.68. While the trainable K model does better at 200D than at 5D, it never catches up to the original k=-1 baseline model. Let's talk about the edge flipping now, as we increase the flip probability, both methods degrade, but this makes sense since reversing edges injects contradictory hierarchical information. We note that at the fixed, k=-1, model, we see a modest jump in rank and a drop in MAP, but often the rank remains at around the 2 rank mark, and at around 0.5 – 0.8 MAP with p=0.3. On the other hand, the trainable K model tends to degrade and worsen more sharply in rank and drops towards 0.35 MAP at high noise levels. This is contrary to my original intuition, that learning the curvature might help with noisy hierarchical data. At least with the specific parameters, the updated model does not end up performing better than the original, fixed K=-1, under higher noise. At the end of this document, I will offer some explanations as to why I think my updated model does worse than the original one. One possible explanation though, is that the Wordnet Mammals dataset already favors k=-1, and forcing the model to learn a more "flexible" geometry can simply add noise and complexity without any real payoff.

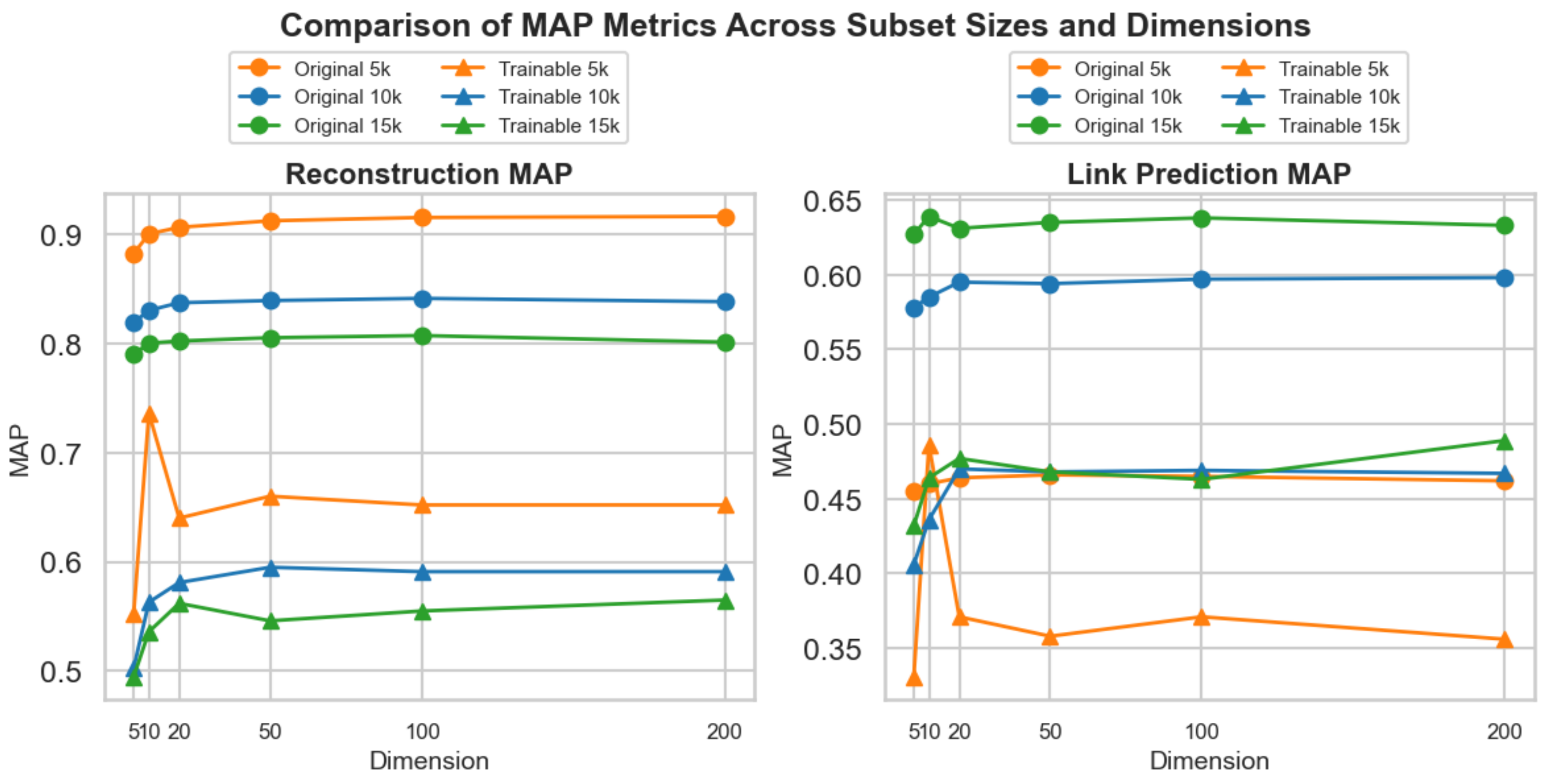
Experiment 2 – Facebook Social circles and subset incrementation

Here are our results:

Subset Size	Embedding Dimension	Reconstruction RANK	Reconstruction MAP	Link Prediction RANK	Link Prediction MAP
5000	5	5.614	0.553	11.057	0.331
	10	1.937	0.736	4.942	0.486
	20	5.395	0.64	11.21	0.371
	50	4.896	0.66	11.582	0.358
	100	5.199	0.652	11.677	0.371
	200	5.133	0.652	11.589	0.356
10000	5	5.917	0.503	8.309	0.406
	10	5.245	0.563	7.723	0.436
	20	4.97	0.581	7.51	0.47
	50	4.708	0.595	7.236	0.468
	100	4.688	0.591	7.253	0.469
	200	4.972	0.591	7.112	0.467
15000	5	6.017	0.495	7.515	0.432
	10	5.526	0.536	7.075	0.464
	20	4.911	0.562	6.664	0.477
	50	5.535	0.546	7.156	0.468
	100	5.174	0.555	6.997	0.463
	200	4.706	0.565	6.367	0.489

Comparison with Phase 1's Results:

Here is a graph for comparison with the original version:



Let's start with reconstruction:

The original Poincare embedding typically has MAP values around 0.88-0.92 for different dimensions on the small subset size of 5000. Then about 0.79 – 0.81 or so for the bigger subset size of 15000. On the other hand, the modified Poincare gives us lower values overall, sadly. About 0.55-0.74 on the 5000 subset and no more than 0.57 on the 15000 subset. What I'm trying to say is that the original Poincare, with K=-1, again, consistently shows higher reconstruction MAP across different dimensions and sample sizes. So even though this is a non-hierarchical facebook dataset, the original K=-1 curvature assumption is working surprisingly well.

For Link prediction it seems pretty similar in a way: The original model has an MAP is hovering around the 0.46 mark on the 5000 sized subset, and MAP of around 0.62-0.64 for biggest subset. This is, interestingly, higher than I might have expected from a method which assumes that the data hierarchical (here it is not hierarchical!), and it seems to outperform the modified poincare version with a learnable K in many different subset sizes and dimensions. The modified Poincare method is about the 0.33-0.49 range (over all dimensions and subset sizes), there are some improvements with more dimensions and more sample sizes but clearly there isn't any great success here. In summary, the original model, with K=-1, yields better MAP scores for both metrics, across nearly all dimensions and subset sizes, even though I would expect a flexible curvature to help with this non-hierarchical dataset.

From inspecting the facebook dataset results from Phase1 and Phase 2 I noticed a few things:

1 – The original Poincare consistently does better than my modified version in both reconstruction and link prediction, across all tested dimensions and subset sizes.

2 – Higher dimensions generally help the original Poincare model remain stable, or sometimes improve, while the modified curvature has more inconsistent patterns. but it never surpasses the original Poincare model metrics, except for when Subset=5000, dimension = 10, then the link prediction map is better for the modified Poincare model.

3 – The subset sizes somewhat affect both versions of Poincare, but the original is systematically better in terms of metrics.

4 – despite the lack of hierarchy in this specific dataset, the K=-1 curvature is still better in these experiments, which might suggest that the simpler, original model can be highly effective and fruitful if the dataset's structure is not too far away from being "hyperbolic".

Experiment 3 – HebrewNet and Probabilistic Edge Removal

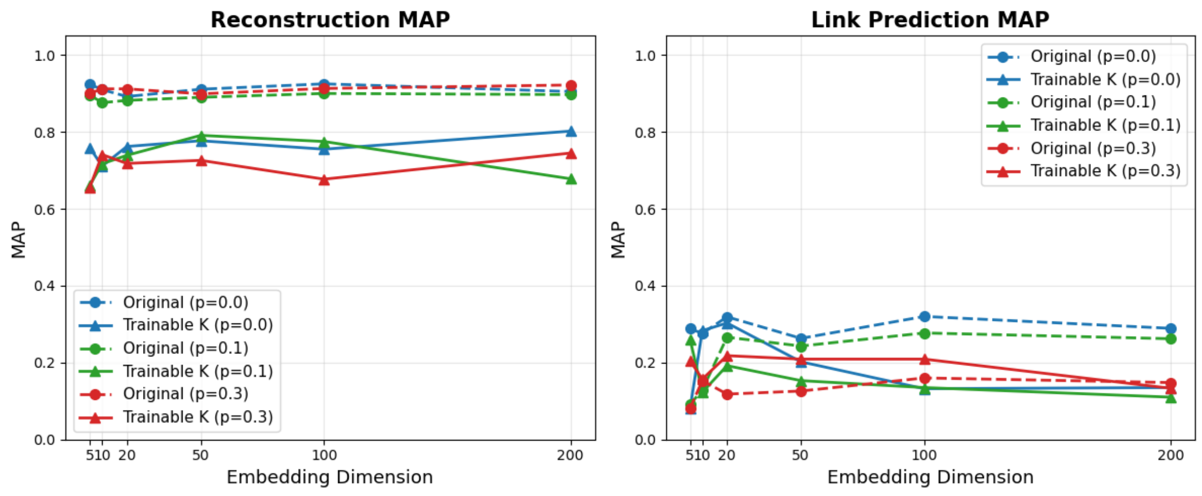
Here are the results:

Embedding Dimension	Edge Removal Probability	Reconstruction RANK	Reconstruction MAP	Link Prediction RANK	Link Prediction MAP
5	0	1.731	0.759	19.5	0.08
	0.1	4.628	0.661	8.375	0.261
	0.3	4.481	0.656	8.5	0.205
10	0	3.892	0.711	9.25	0.283
	0.1	3.417	0.717	11.5	0.123
	0.3	4.04	0.741	8.5	0.157
20	0	3.295	0.762	6.125	0.303
	0.1	4	0.739	12.75	0.192
	0.3	5.03	0.718	10.625	0.218
50	0	3.622	0.777	7.125	0.202
	0.1	3.227	0.791	7.625	0.153
	0.3	1.951	0.726	12.5	0.209
100	0	3.911	0.755	8	0.132
	0.1	1.694	0.775	9.5	0.135
	0.3	4.412	0.677	6.625	0.209
200	0	2.488	0.802	11.375	0.135
	0.1	4.161	0.678	10	0.11
	0.3	3.363	0.745	12.125	0.133

We easily see that we did not do as well as expected in reconstruction when compared to Phase 1, but in link prediction there is still hope. Let's compare:

<u>Comparison with Milestone 2's Results:</u>

Here is a graph comparing Phase 1's (Original) with Phase 2's results ("Trainable K"):



Let's look at reconstruction first: the original Poincare has a higher reconstruction MAP than the trainable K model I created, across all dimensions and probabilities. When looking at p=0, we see that the original Poincare ranges from 0.892 (20D) up to 0.925 (100D). On the other hand, the updated Poincare ranges from 0.711 (10D) up to 0.802 (200D). When p=0.1, the original ranges from 0.876-0.9, with the best map with 100D, while the trainable K Poincare ranges from 0.661-0.791 with the best being 50D. Again, the original outperforms the modified Poincare when looking at their best points. When p=0.3, the original model has very high MAP, with a peak at 200D (0.922 MAP) while the trainable K has a much lower MAP, with 10D having an MAP of 0.741 and 200D having an MAP of 0.745. The gap between the two models is still big if we're looking for example at 200D. What I'm trying to say is that the original model, with a fixed k=-1, consistently outperforms the modified Poincare, often by 0.1-0.2 regardless of dimension or noise level. Now let's look at link prediction: The performance depends strongly both on the dimensionality and the removal probability. Unlike reconstruction, there are crossover points where the trainable K Poincare can do better in some dimensions and noise levels. When looking at p=0, and Dimensionality =10, we can see that our updated model is slightly better, with a link prediction MAP of 0.283, while the original model has a link prediction MAP of 0.276. In all other p=0 cases, the original model is better, and often is significantly better. When p=0.1, and looking at dimensionality of 5, our model is much better than the original, we have a link prediction MAP of 0.261 while the original has 0.091. for the other dimensions, the original Poincare model is better. When p=0.3, the modified model is better in Dimensionality of 5,10,20,50,100. For example, at Dimensionality=5 we have 0.205, when the original model has 0.081. For dimensionality=20 we have a score of 0.218, and the original model has a link prediction MAP of 0.118. When the dimension of the embedding is 200, the original Poincare model has a link prediction MAP of 0.148 while my modified version has a score of 0.133, so the original is slightly better at that single point. In summary, these results show that for reconstruction, the original Poincare embedding consistently outperforms my modified version of the algorithm, regardless of dimensionality or noise level – which means that K=-1 is pretty good for preserving hierarchical edges in these experiments. For link prediction, the modified version has situational advantages, especially under high noise(p=0.3) and mid-range embedding dimensions.

Did I fail? Reasons why I might've failed and how to fix them:

Despite my initial intuitions that a trainable curvature would better adapt to the underlying dataset's geometry, the results show that the original Poincare model with K=-1 consistently outperforms our modified version in most of the experiments, I believe that there are several reasons as to why the results show such phenomena:

Addition of Training complexity: Introducing a trainable curvature parameter means that the model must optimize the alpha, which I wrote about in the introduction, alongside the embeddings. This increases the optimization complexity which might have given me sub-optimal local minima. This also means that I need a more careful hyperparameter tuning such as learning rates, regularization, epochs etc. to reach the full potential. I also found that a lot of numerical instabilities can arise near the Poincare ball boundary if the curvature becomes too negative or too close to zero, which made me perform clamping and reparameterization. These “real world” interventions and changes may have reduced the “theoretical” flexibility of my proposed model in practice.

Overfitting: Allowing the curvature to vary could lead the model to overfit to noise in the training set. In that case, while we do gain more freedom to reshape the embedding space, we might settle into configurations that do not generalize as well as the simple K=-1 approach given by the authors of the original paper.

Not enough tuning:

The approach that I proposed, with a trainable K, may require more fine tuning to actually tap into the full potential of this method. For example, bounding the curvature within a better range, or maybe scheduling the learning rate of alpha separately.

I believe that the factors I noted above explain why the modified Poincare method’s “theoretical” flexibility did not translate into conclusive and clear empirical advantages in the different data and experimental settings. I recommend that if someone were to take over the reins of this project, they should explore a smarter and more refined optimization schedule, maybe try different epochs, learning rates, initializations, maybe a stronger curvature regularization, or maybe a different clamping scheme. I hope that these recommendations will help the trainable curvature converge to geometries that genuinely outperform the fixed K=-1.