Representing hierarchy in graphs using Poincaré Embeddings with distributed frameworks

Rishi Hazra
14542
Indian Institute of Science
Systems Engineering
rishihazra@iisc.ac.in

Sourabh Balgi
14318
Indian Institute of Science
Systems Engineering
sourabhbalgi@iisc.ac.in

Abstract

Representation learning has become an invaluable approach for learning from symbolic data such as text and graphs. However, while complex symbolic datasets often exhibit a latent hierarchical structure, state-of-the-art methods typically learn embeddings in Euclidean vector spaces, which do not account for hierarchy. In the recent knowledge representation works, hyperbolic vector spaces have been shown to better encode the hierarchical structures. One of the recent work in this field is *Poincaré Embeddings*, new approach for learning hierarchical representations of symbolic data by embedding them into hyperbolic space or more precisely into an n-dimensional Poincaré ball. Due to the underlying hyperbolic geometry, Poincaré embedding allows us to learn parsimonious representations of symbolic data by simultaneously capturing hierarchy and similarity. Due to the large size of the amount of data in the knowledge graphs, Distributed training has become an integral part. In this work, We implement the Poincaré embeddings using Distributed frameworks like TensorFlow, PyTorch and Horovod. We explore, observe, compare their results and show that the training time can be improved by a factor or at-least two without compromising on the results.

1 Introduction

Many real-world datasets exhibit hierarchical structure, either explicitly in ontologies like WordNet, or implicitly in social networks (Adcock et al., 2013) and natural language sentences (Everaert et al., 2015). (Nickel et al., 2014) showed that linear embeddings of graphs can require a prohibitively large dimensionality to model certain types of relations. Although non-linear embeddings can mitigate this problem (Bouchard et al., 2015) in small dimension. However complex graph patterns can still require a computationally infeasible embedding dimensionality. When learning representations of such datasets, hyperbolic spaces have recently been advocated as alternatives to the standard Euclidean spaces in order to better represent the hierarchical structure (Nickel and Kiela, 2017), i.e., the Poincaré ball model, as it is well-suited for gradient-based optimization. .

In this paper: **(i)** We probe distributed TensorFlow setups to try and optimize the running time even as we maintain on par results.**(ii)** We explore the use of Horovod (Sergeev and Balso, 2018), a distributed training framework for TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2017). **(iii)** We investigate a multi-threading setup in PyTorch framework.

2 Related Work

Learning representations of symbolic data such as text, graphs and multi-relational data has become a central paradigm in machine learning and artificial intelligence. For instance, word embeddings such as WORD2VEC (Mikolov et al., 2013), GLOVE (Pennington et al., 2014) and FASTTEXT (Bojanowski et al., 2017) are widely used for tasks ranging from machine translation to sentiment analysis. Similarly, embeddings of graphs such as latent space embeddings (Hoff et al., 2001), NODE2VEC (Grover and Leskovec, 2016), and DEEPWALK (Perozzi et al., 2014) have found important applications for community detection and link prediction in social networks. Embeddings of multi-relational data such as RESCAL (Nickel et al., 2011), TRANSE (Bordes et al., 2013), and Universal Schema (Boratko et al., 2018) are being used for knowledge graph completion and information extraction.

3 Poincaré Embeddings

In this section, we give an overview of the Poincaré embeddings method from (Nickel and Kiela, 2017). A hyperbolic space is a non-Euclidean geometric space obtained by replacing Euclids parallel postulate with an alternative axiom. The parallel postulate asserts that for every line L and point P not on L, there is a unique line co-planar with P and L that passes through P and does not intersect L. In hyperbolic geometry, this axiom is replaced with the assertion that there are at least two such lines passing through P that do not intersect L (from which one can prove that there must be infinitely many such lines). In this geometry, some familiar properties of Euclidean space no longer hold; for example, the sum of interior angles in a triangle is less than 180 degrees. Like Euclidean geometry, hyperbolic geometry can be extended to d-dimensions. d-dimensional hyperbolic space is unique up to a curvature constant $\mathrm{K} < 0$ that sets the length scale. Without loss of generality, we assume $\mathrm{K} = 1$.

In a two dimensional hyperbolic space with constant curvature $\mathrm{K} = 1$, the length of a circle is given as $2\pi \sinh r$,while the area of a disc is given as $2\pi(\cosh r - 1)$. Since $\sinh r = \frac{1}{2}(\exp^r - \exp^{-r})$ and $\cosh r = \frac{1}{2}(\exp^r + \exp^{-r})$), both disc area and circle length grow exponentially with r. A similar construction is not possible in Euclidean geometry as circle length $(2\pi r)$ and disk area $(2\pi r^2)$ grow only linearly and quadratically, respectively with respect to r. Due to these properties, hyperbolic space has recently been considered to model complex networks.

There are several ways to model hyperbolic space within the more familiar Euclidean space. Of these, the Poincaré ball model is most suited for use with neural networks because its distance function is differentiable and it imposes a relatively simple constraint on the representations. Specifically, the Poincare ball model consists of points within the unit ball $\mathcal{B}^d$ is

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

Notice that, as $||\mathbf{u}||$ approaches 1, its distance to almost all other points increases exponentially. In order to learn representations $\Theta = \{\theta_i\}_{i=1}^n$ for a set of objects $\mathcal{S} = \{\mathrm{s}_i\}_{i=1}^n$, we must define a loss function $\mathcal{L}(\theta, \mathrm{d})$ that minimizes the hyperbolic distance between embeddings of similar objects and maximizes the hyperbolic distance between embeddings of different objects. Then we can solve the following optimization problem.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, \mathcal{L}(\Theta, d) \text{ s.t } ||\theta_i|| < 1 \,\forall\, \theta_{\mathrm{i}} \in \Theta \quad (2)$$

This paves the way for an efficient algorithm for computing the embeddings based on Riemannian optimization, which is easily parallelizable and scales to large datasets. This involves computing the Riemannian gradient (which is a scaled version of the Euclidean gradient) with respect to the loss, performing a gradient-descent step, and projecting any embeddings that move out of $\mathcal{B}^d$ back within its boundary.

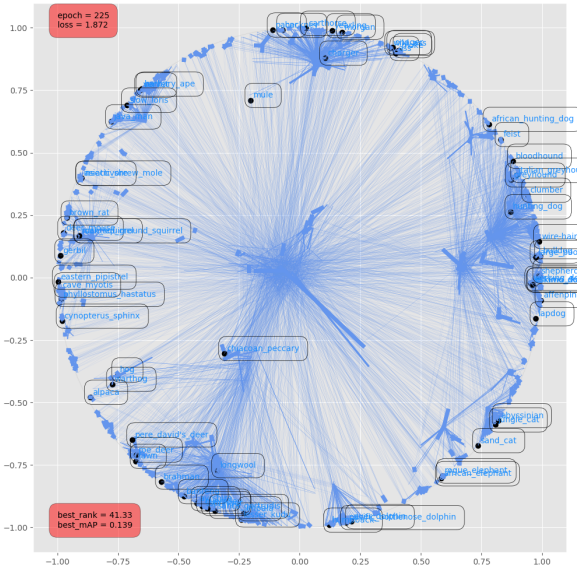
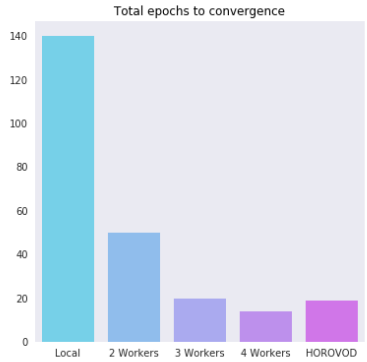


Figure 1: Poincaré Embeddings for Mammals subset. It can be observed that at epoch=225, the MAP=0.139 and rank=41.33. The MAP and rank gradually converge over many epochs; refer to Section A.1 of Supplemental Material.

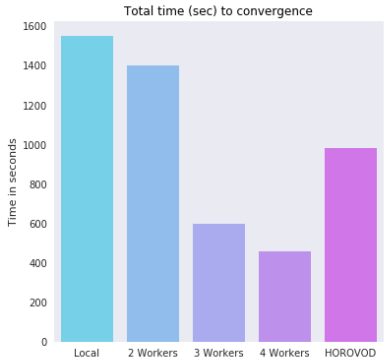
4 Experiments

4.1 WordNet

The WordNet (Miller, 1995) noun hierarchy is a collection of tuples $\mathcal{D} = \{\mathrm{x}, \mathrm{y}\}$ where each pair (x, y) denotes that x is a hypernym of y. We learned embeddings using the transitive closure D^+, which consists of 82,114 nouns and 743,241 hypernym-hyponym edges. In each case, we evaluated the embeddings by attempting to reconstruct the WordNet tree using the nearest



(a) Total epochs to convergence



(b) Total time (sec) to convergence

Figure 2: (a) shows the total number of epochs required for convergence in Asynchronous and HOROVOD setup for TensorFlow. It can be observed that with the increase in number of epochs, the total number of epochs also reduce (b) shows the comparison of the total time with respect to the number of worker nodes. With the increase in number of workers, the time for each epoch increases but the convergence occurs in lesser number of epochs as compared to the local setting, thus, reducing the overall time for convergence. It can be observed that HOROVOD setup with 2 CPU nodes performs better than Asynchronous setup with 2 CPU nodes.

Metric	**5D (FAIR)**	**5D (PyTorch Threading)**	**5D (HOROVOD PyTorch)**	**2D (PyTorch Threading)**	**2D (Asynchronous TensorFlow)**	**2D (HOROVOD TensorFlow)**
Rank	4.9	4.93	**4.91**	5.10	9.40	9.19
MAP	0.823	0.793	**0.81**	0.335	0.222	0.273

Table 1: Comparison of the *Rank* and *MAP* estimates for different setups.

neighbors of the nodes. For each node, we retrieved a ranked list of its nearest neighbors in embedding space and computed the *Mean Rank* of its ground truth children, i.e. among the set $\{\mathrm{d}(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}') \notin \mathcal{D}\}$. We also computed the *Mean Average Precision (MAP)*, which is the average precision at the threshold of each correctly retrieved child.

Two sets of experiments were performed: one with TensorFlow backend (on CPU) and other with PyTorch backend (on GPU) which we elucidate in the next section. Additionally, we implemented a Numpy version for the same; refer to Section A.2 for more details.

4.2 PyTorch Implementation

The baseline model from the github was used reproduce the baseline results from the origin paper for our experiments. The original code had to be modified to reproduce the original results. In the context of distributed settings, following 2 experiments were carried out.

- Multi-threading feature in PyTorch.
- Distributed training using Horovod with PyTorch.

Both the implementation are same with respect to the algorithms. However training mechanisms differ.

Burn-in (Chang et al., 2017) : It was observed in the original paper that initially decreasing the learning rate by a factor of 10 to 100 for burn-in period helps the learning of embeddings. The embeddings are randomly initialized from uniform distribution in the range [-0.0001, 0.0001]. Since the initial embeddings are very small, the smaller learning rate helps in better rearranging of embeddings. This initial orientation provides a better starting point for the learning with regular learning after the burn-in. The figures in the supplemental materials indicates the benefits of burn-in. 50 negative samples were used in all the experiments.

- **Multi-threading feature in PyTorch**: The original implementation from FAIR has a

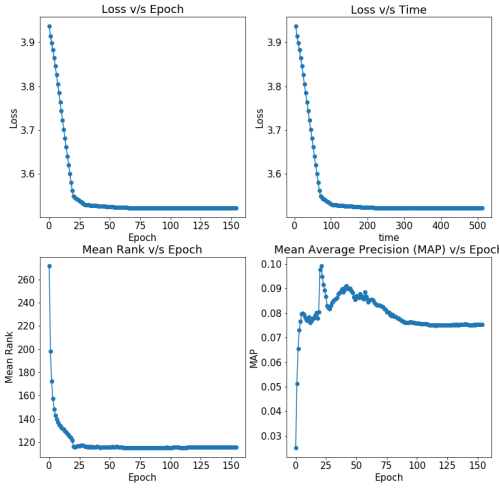


Figure 3: The given set of plots shows how the hyperbolic embeddings are learned in the **Horovod** over PyTorch implementation with 5D embeddings. The loss, MAP and Mean Rank are seen to converge gradually with increase in epochs.

multi-threading implementation to speed up the convergence. However each epoch takes around 4 seconds and the total number of epochs required to observe convergence is 500 epochs over the mammals subset of Wordnet.

- **Distributed training using HOROVOD with PyTorch** (Figure 3): 2 GPUs were used for distributed training in HOROVOD. The initial model has to be broadcasted across the GPUs for both the models to start at the same initial point. Because of the use of GPU for computation the per epoch time reduced to 1 second. The model convergence was observed at an even smaller number of epochs of 150 epochs as shown in (Figure 3). Horovod core principles are based on MPI concepts such as size, rank, local rank, allreduce, allgather and broadcast. It uses AllRingReduce mechanism for distributed training where as TensorFlow distributed training uses Synchronous Parallel distributed method.

With the distributed training on GPU using Horovod, we were able to observe a very drastic decrease in time for convergence of the model compared to the Multi-threaded implementation using only CPUs.

4.3 TensorFlow Implementation

With the TensorFlow, we explore three different distributed frameworks: Asynchronous, Drop Stale Synchronous and Horovod. Polynomial learning rate decay was used with initial learning rate being 0.001. 10 negative samples were used for each of the setups. We record the results of each of the setups and compare them with the local setting. The embeddings size was fixed to 2.

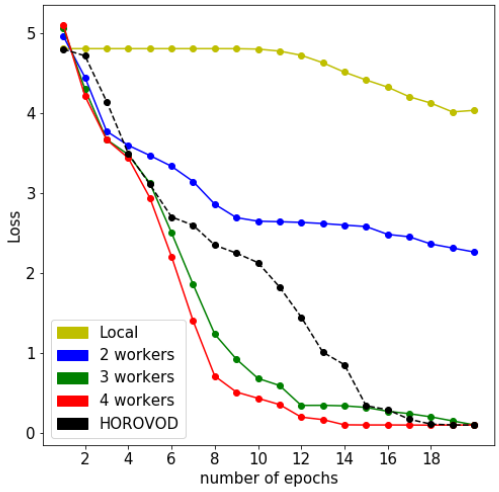


Figure 4: The given plot compares the decrease in Loss with the increase in number of workers in the Distributed TensorFlow setup. It can be observed that the the fall is loss gets steeper as the number of workers increases. For local settings, the loss takes a long time to decrease even slightly. It should be noted that the time taken per epoch also increases with increase in number of workers but the overall time for convergence decreases. The dotted line denotes the HOROVOD setup.

- **Distributed setup**: The distributed setup is implemented in Asynchronous and Drop Stale Synchronous. For the Asynchronous setting, we used 2, 3, 4 workers for our experiments (Figure 4). For 4 workers, the loss was seen to converge in 12 epochs as compared to 140 epochs for the local setting. It took 50 epochs and 20 epochs for 2 workers and 3 workers respectively, for convergence (Figure 2). It was also observed that the time per epoch was 10.71 seconds for local setting and 29.66 seconds for the distributed setup, although the total time for convergence reduced significantly in the distributed setting. The staleness for Drop Stale Synchronous setup

was fixed at 35 seconds.

- **HOROVOD setup**: The TensorFlow version of HOROVOD was run on CPU nodes, as opposed to, GPU nodes for the PyTorch version. The initial model was broadcasted across the CPUs for both the models to start at the same initial point. In our experiments, 2 CPU nodes were used. As seen in (Figure 4), in the HOROVOD setup, convergence occurs in the same number of epochs, with 2 CPU nodes, as that of Asynchronous setting with 3 CPU nodes. The per epoch time for HOROVOD setup was noted as 55 seconds and the total time for convergence was 985 seconds (Figure 2).

The CPU based codes were run on Turing cluster and the GPU based codes were run on GeForce GTX 1080.

5 Github links

- Github link to original implementation : (FAIR Poincaré Embedding code).
- Github link to our implementation : (Distributed Poincaré Embedding using TensorFlow, PyTorch and Horovod).

6 Conclusion and Future work

We experimented with the use of distributed setup in different frameworks to optimize the process of learning Poincaré Embeddings. We describe the difference in the training process with the use of distributed TensorFlow in CPU and distributed PyTorch in GPU. Additionally, we observed and concluded that Horovod proved to be a better distributed framework in the context of our experiments over both TensorFlow and PyTorch.

In the future work, we would like to explore the Gensim Implementation (Gensim Poincaré Embedding code) as it has been shown that the C++ implementation of the paper reduced the training time by a factor of 6.

We would also like to explore the Poincaré embedding with multi-relational knowledge bases to encode even higher levels of hierarchy. We would also like to explore the link prediction between any given entity pairs.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. 2013. Tree-like structure in large social and information networks. In *2013 IEEE 13th International Conference on Data Mining*, pages 1–10.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, Ryan Musa, Kartik Talamadupula, and Michael Witbrock. 2018. A systematic classification of knowledge, reasoning, and context within the ARC dataset. In *QA@ACL*, pages 60–70. Association for Computational Linguistics.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc.

Guillaume Bouchard, Sameer Singh, and Théo Trouillon. 2015. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI Spring Symposia*.

Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. 2017. Active bias: Training more accurate neural networks by emphasizing high variance samples. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1002–1012. Curran Associates, Inc.

M.B.H. Everaert, Marinus A C Huybregts, Noam Chomsky, Robert Berwick, and Johan Bolhuis. 2015. Structures, not strings: Linguistics as part of the cognitive sciences. *Trends in Cognitive Sciences*, xx.

Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA. ACM.

Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. 2001. Latent space approaches to social network analysis. *JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION*, 97:1090–1098.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Maximilian Nickel, Xueyan Jiang, and Volker Tresp. 2014. Reducing the rank in relational factorization models by including observable patterns. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1179–1187. Curran Associates, Inc.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 809–816, USA. Omnipress.

Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6338–6347. Curran Associates, Inc.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *In EMNLP*.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA. ACM.

Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799.

A Supplemental Material

A.1 PyTorch implementation

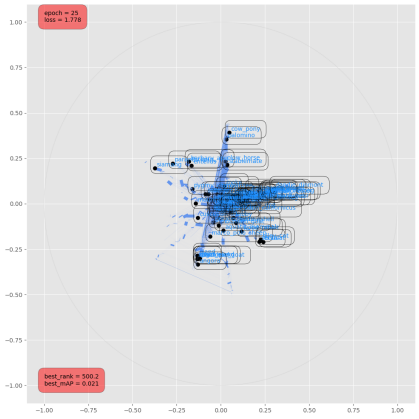


Figure 5: Epoch:25, Rank:500.2, MAP:0.021

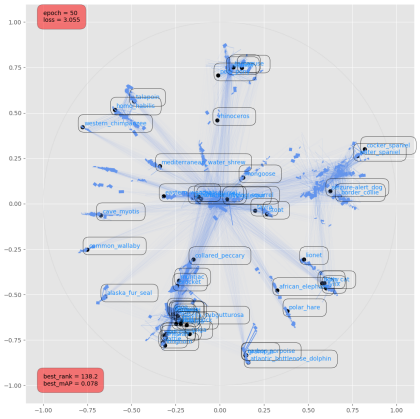


Figure 6: Epoch:50, Rank:138.2, MAP:0.078

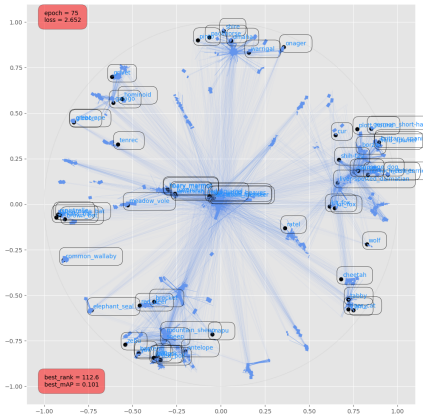


Figure 7: Epoch:75, Rank:112.6, MAP:0.101

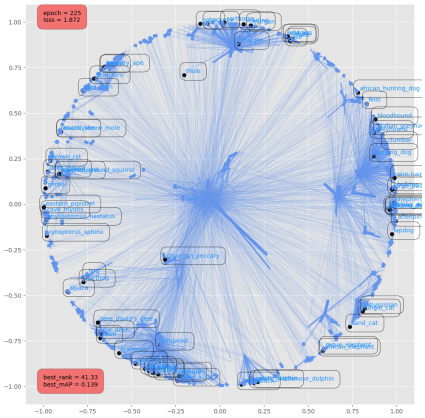


Figure 8: Epoch:225, Rank:112.6, MAP:0.101

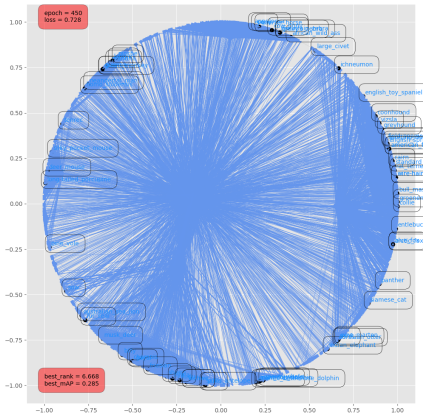


Figure 9: Epoch:450, Rank:6.68, MAP:0.285

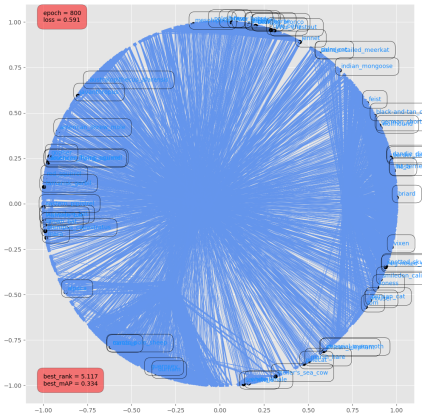


Figure 10: Epoch:800, Rank:5.11, MAP:0.285

Figure 11: shows how the Poincaré embeddings are learned with the increase in epochs. The *Rank* and *Mean Average Precision* are observed to converge to the baseline results. In 800 epochs, for 2D embeddings, *Rank* converges to 5.10 (baseline : 4.9) and MAP converges to 0.335 (baseline : 0.823). It may be noted that the embeddings were initialized in a uniform distribution in the interval [-0.0001, 0.0001]. 50 negative samples.

A.2 Numpy implementation

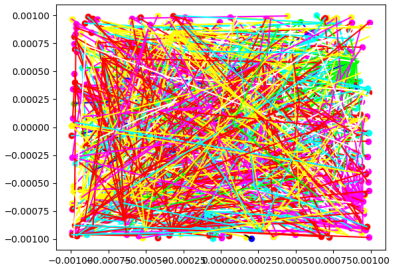


Figure 12

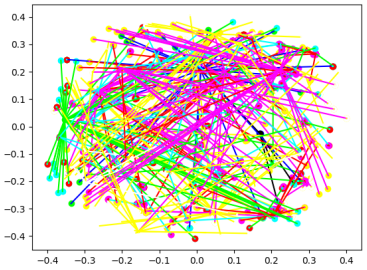


Figure 13

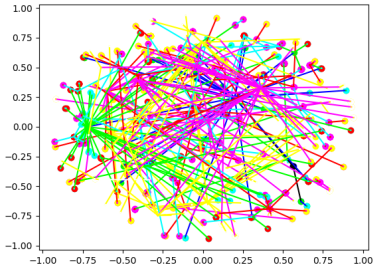


Figure 14

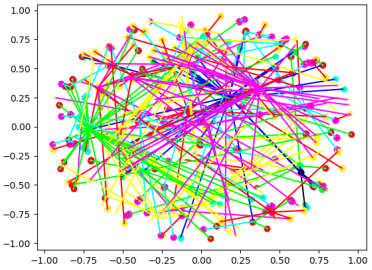


Figure 15

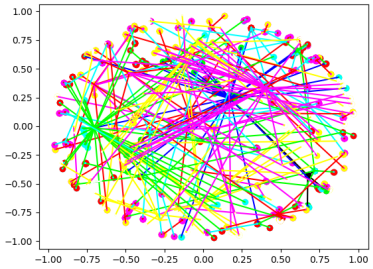


Figure 16

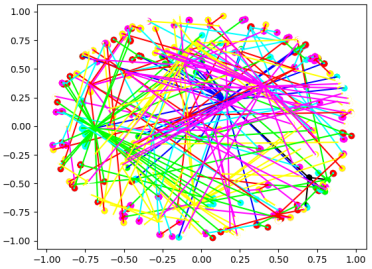


Figure 17

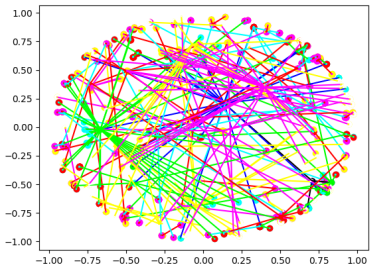


Figure 18

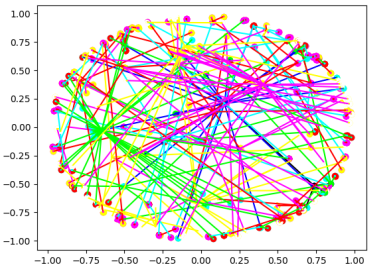


Figure 19

Figure 20: (c)shows how the Poincaré embeddings are learned with the increase in epochs in the Numpy setup for 8 levels of embeddings in WordNet. The different levels are plotted in different colours with all embeddings initialized with uniform distribution in the interval [-0.0001, 0.0001]. It can be observed that the same nodes get clustered with the increase in epochs.