progress_report

February 19, 2021

[18]:
```
from IPython.display import Image
```

1 The Plan

1.1 Phase 1

21.12.2020 - 21.02.2021

1. Explore existing literature on hyperbolic deep learning on graphs (HGCN, HGNN papers)
2. Understand the basic concepts of Diffential Geometry
3. Understand the code structure of existing solutions
4. Implement the hyperbolic graph convolution layer in PyTorch Geometric

you are here

5. Implement Hyperbolic Deep Graph Matching Consensus (and make it work)
6. Test the performance on the DBP15k dataset

1.2 Phase 2

21.02 - ???

1. Research existing datasets for graph matching and find out suitable geometry for each
2. Implement Spherical Deep Graph Matching Consensus (boils down to implementing the Spherical manifold)
3. Test the performance of the three versions of DGMC (Eucl., Hyp., Sph.) on these datasets
4. ?????

2 The Diary

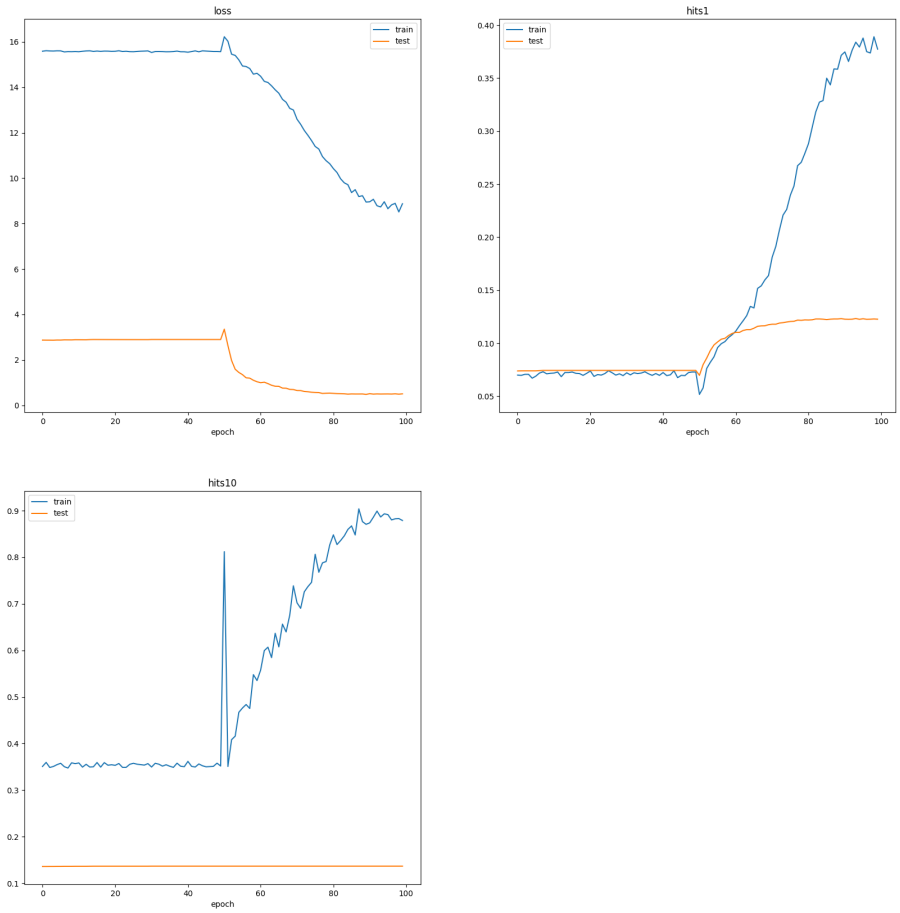
2.1 *17.02*

- Q: Are the weights in the hyperbolic layer Euclidian or Hyperbolic, in which way should they be optimized? A: The weights operate in the tangent space, hence they are Euclidian
- P: Implementing a layer with $in\_channels \neq out\_channels$ S: To rewrite *aggregate*, *message* and *update* methods

- I have run the HDGMC algorithm on the DBP15k dataset, the results are below

[25]:
```
Image('results_first_run.png')
```

[25]:



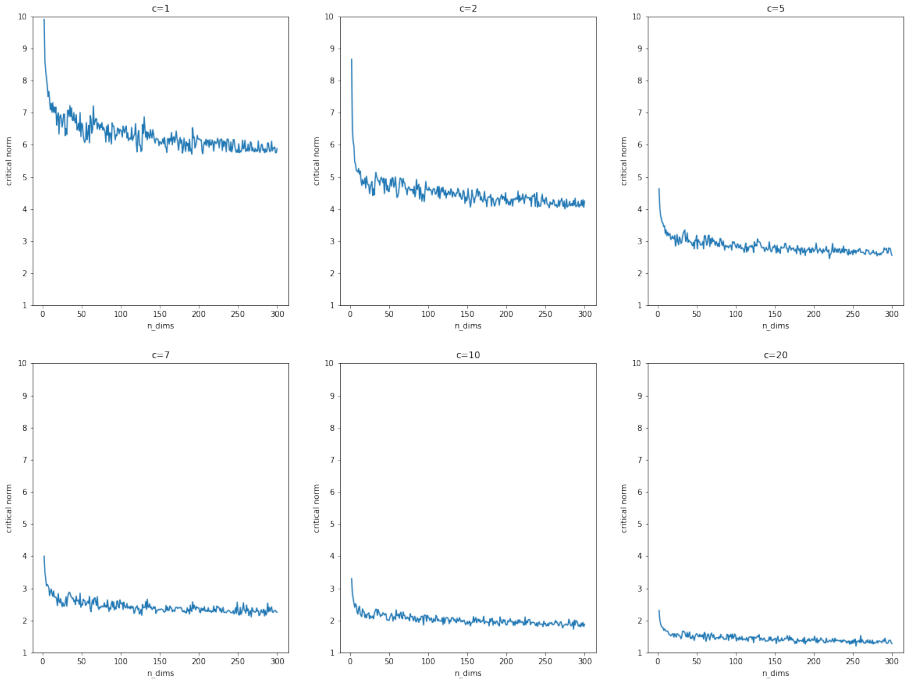
- This is weird. Need to check the code for errors.
- Q: Why is there *better than random* performance even after the first epoch? A: ...
- Q: Why does the improvement start only in the neighborhood consensus stage? H: Most of the embeddings are zero as a result of ReLU (no) H: Maybe the initial similarity matrix calculation is incorrect? (i think it is correct) N: Check if the embeddings remain on the hyperboloid. (no, they do not, see *18.02* entry) A: ...

2.2 *18.02*

- I have found out that for vectors with high Euclidian norm, *expmap0* method blows up and the result is not on the hyperboloid due to numerical reasons. At least its squared Minkowski norm is not equal to $-\frac{1}{K}$ as it should be. Maybe this is a result of the substraction of two big floating point numbers in *minkowski_dot*.
- Below are the plots which show for which norm the squared Minkowski norm starts going crazy for different numbers of dimensions and different curvatures.
- The first group of plots shows the dependence of the critical norm on the number of dimensions with fixed curvature, and the second group of plots is vice versa.

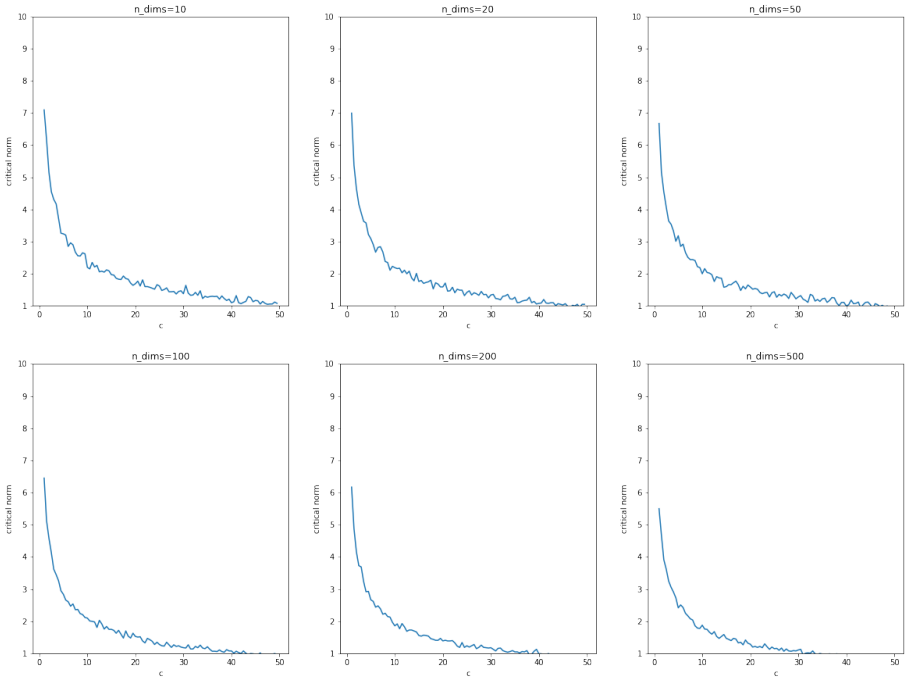
[29]:
```
Image('critical_norm1.png')
```

[29]:



[30]:
```
Image('critical_norm2.png')
```

[30]:


n_dims=10
n_dims=20
n_dims=50
n_dims=100
n_dims=200
n_dims=500
critical norm
c