progress_report

March 2, 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*

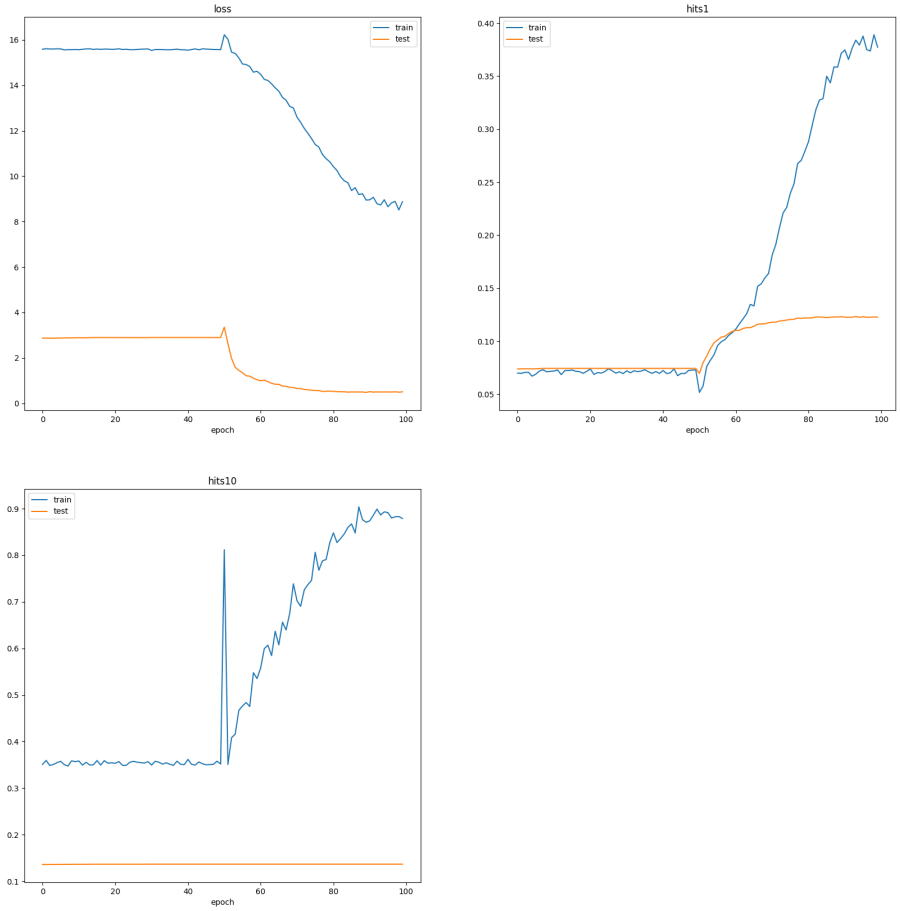
- Q: Are the weights in the hyperbolic layer Euclidian or Hyperbolic, in which way should they be optimized?
  A: The weights operate in the tangent space, hence they are Euclidian

- 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)

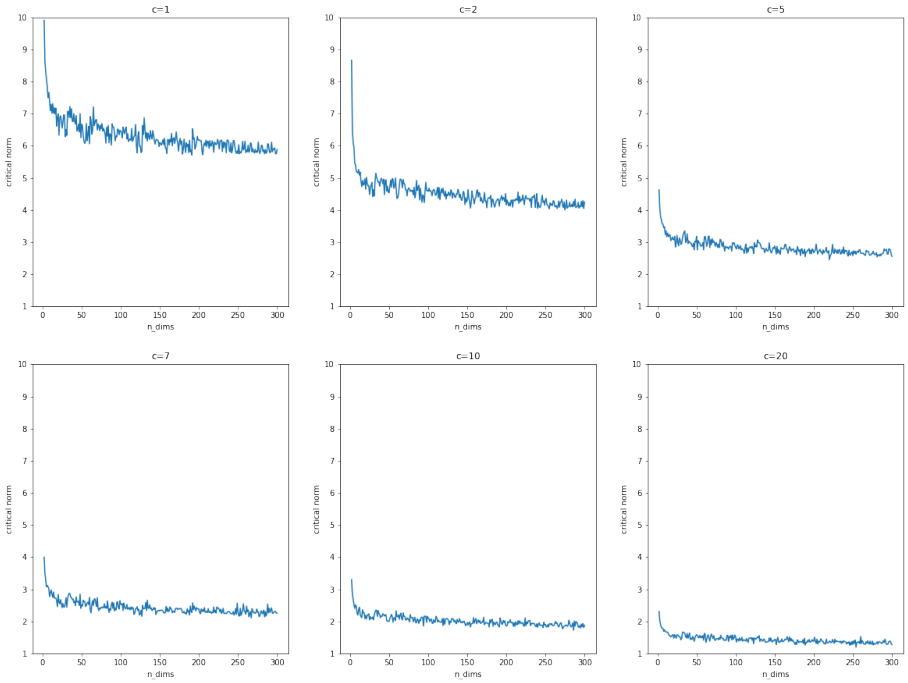
H: Maybe the initial similarity matrix calculation is incorrect? (i think it is correct)
N: Check if the embeddings remain on the hyperboloid. (no, they do not, see *18.02* entry)
A: ...

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.

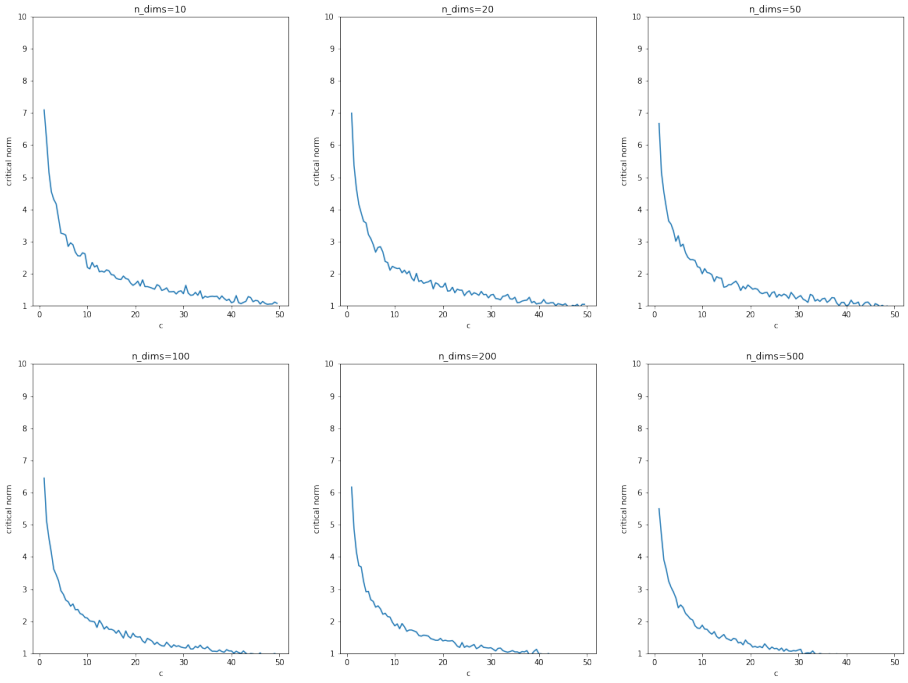
[29]:
```
Image('critical_norm1.png')
```

[29]:



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

[30]:



2.3 *19.02*

- Made a conclusion about the above graphs that the critical norm depends only on the curvature, not the number of dimensions, which is kinda intuitive
- Found out that if the initial embeddings are divided by 5, their *expmap0* does not explode and they end up on the hyperboloid after *expmap0*
- P: However, after passing them through the network, they are not on the hyperboloid. Tried lowering the norm of the weights in the network, it didn't work, tried changing the aggregation procedure from *sum* to *mean*, didn't work also. I have suspicions that the repeated applications of *expmap* cause this.
  S: The problem was that through DGMC, the outputs of different layers are concatenated, this crashes the hyperbolic procedure

- **IMPORTANT:** If I calculate *hits1* by just applying *expmap0* to the initial embeddings and then calculating the similarity via Minkowski scalar product, I get 0.55 *hits1*. This indicates that the network does not work (at least its first stage). After the second stage train *hits1* go up to 0.4.

2.4 *23.02*

- Tried checking if the embeddings stay on the hyperboloid through the forward pass in the experiments from the paper (yes, they do)
- Q: Why they do using the data from the paper and don't using the DBP15k dataset
- The main problem at the moment is that the gradient vanishes

2.5 *28.02*

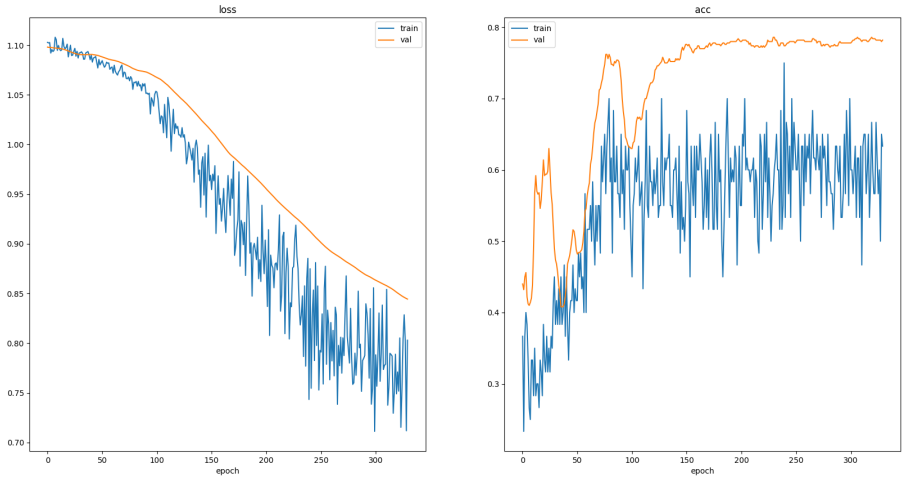
- Tried inserting backward hooks everywhere< found out that at some point the gradient norm drops by 8 orders of magnitude. The reason for this is unknown yet.
- The training dynamics in the paper's authors' experiments is kinda weird, needs some investigation.
- I got suspicious that in the authors' experiments the performance is gained only due to the training of the linear Euclidean decoder, checked the norms of gradients for both Hyperbolic encoder and Euclidean decoder: they are comparable, a proof of concept of sorts.
- I think some experiments on toy synthetic data should be done for a clearer view of where the problem might be.

2.6 *02.03*

- The training dynamics turned out to de strange due to low train size (60 nodes)
- Came up with another idea: instead of computing similarity via Minkowski dot, apply $proj\_tan0 \circ logmap0$ to the embeddings and then compute Euclidean similarity
- Found out a critical error in the similarity matrix computation, fixed it, the model does not train still :(
- Finally figured out what was wrong!

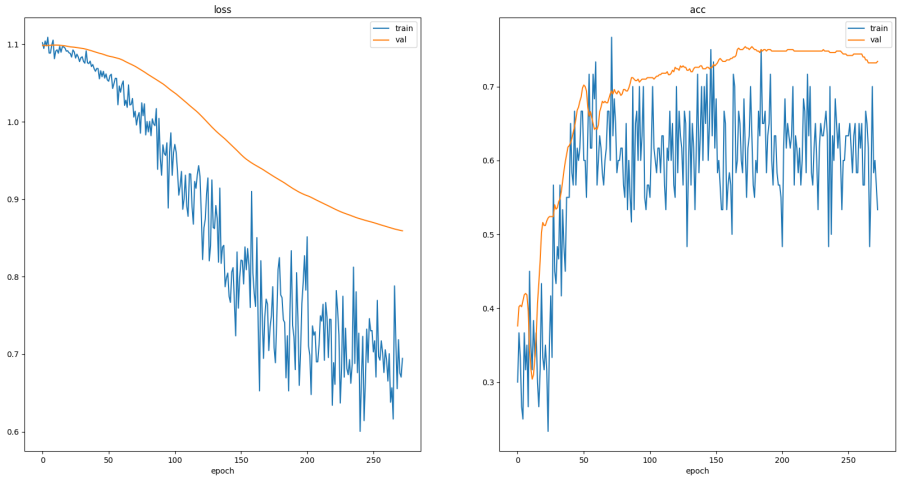
[31]:
```
# this is HGCN
Image('strange_hgcn_dynamicsHGCN.png')
```

[31]:



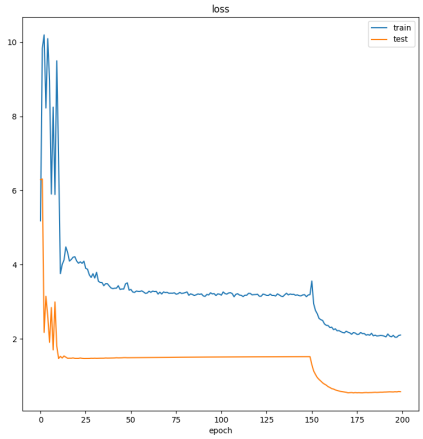
[32]:
```
# this is HGCN
Image('strange_hgcn_dynamicsMyHGCN.png')
```

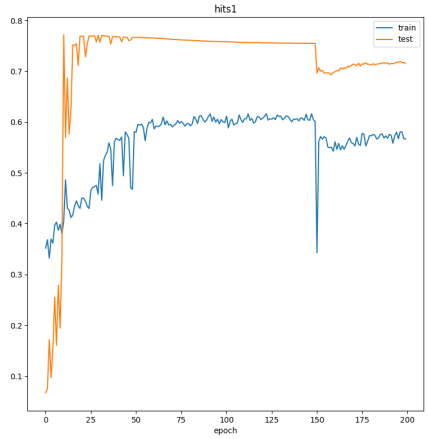
[32]:

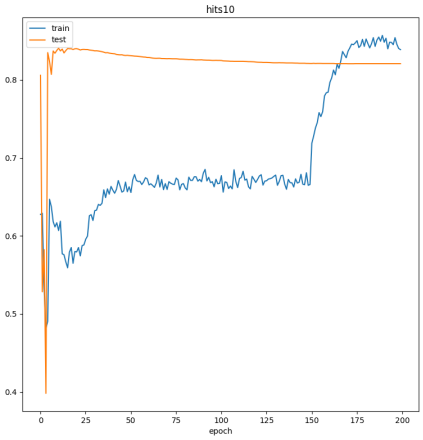


[33]:
```
# this is HDGMC
Image('results_initial_ver0.png')
```

[33]:


loss
train
test
epoch


hits1
train
test
epoch


hits10
train
test
epoch