MAP583

Jeremie Dentan – Ahmed Laftit – Dan Meller

Content

1. Introduction
2. First discussion: stability: modify the dataset:
   a) Presentation of the dataset
   b) Changing the probability q
   c) Changind the size of the dataset

3. Second discussion: performance: modify lap_pos_enc
   a) Without positional embedding
   b) Exponential factors
   c) Absolute value
   d) Homothety
   e) Increasing the dimension of the embedding

Introduction

- Two discussions
  - **Stability :** train a network and test it on another dataset

    => Generate new datasets on CPUs
  - **Performance :** modify the laplacian positional embedding (lap_pos_enc)

    => Train new networks on GPUs

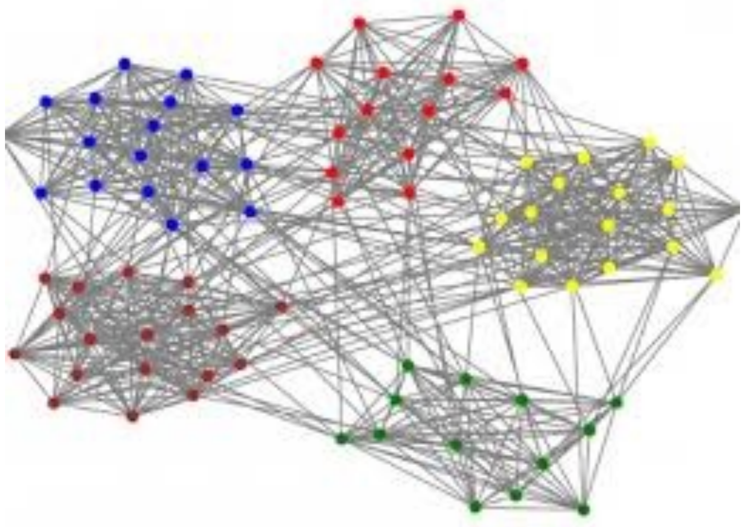
GPU	Our experiments	Paper’s experiments
Material	1 Nvidia Quatro 4000 8Go (~1000€)	4 Nvidia 1080Ti 11Go (~4x1000€)
Epochs	80 max	1000 max
Computation time	5h max per experiment	24h max per experiment

Matériel CPU	Datasets created	Computation time/dataset
2 Intel Xeon E-2174 3.80GHz	5	5h

Table 2: presentation of the CPU and computation time

1-a\ Stability: the PATTERN dataset

- 5 communities are generated with the same probabilities for each block
- We add a 6th community with a different extra-link probability
- Communities' average size is 20
- The goal is to classify the dataset and find which individuals belong to the 6th community

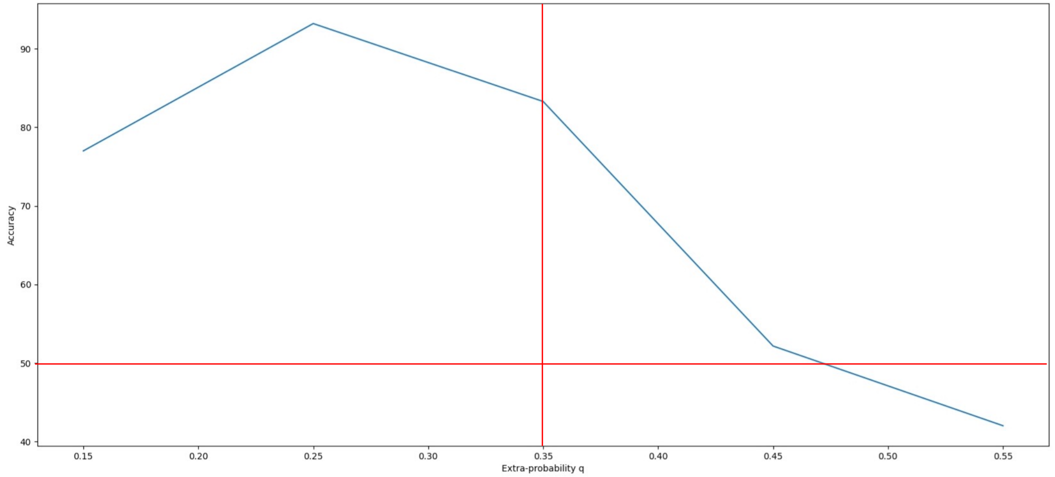


PATTERN: The graph pattern recognition task, presented in [75], aims at finding a fixed graph pattern P embedded in larger graphs G of variable sizes. For all data, we generate graphs G with 5 communities with sizes randomly selected between $[5, 35]$. The SBM of each community is $p = 0.5, q = 0.35$, and the node features on G are generated with a uniform random distribution with a vocabulary of size 3, *i.e.* $\{0, 1, 2\}$. We randomly generate 100 patterns P composed of 20 nodes with intra-probability $p_P = 0.5$ and extra-probability $q_P = 0.5$ (*i.e.*, 50% of nodes in P are connected to G). The node features for P are also generated as a random signal with values $\{0, 1, 2\}$. The graphs are of sizes 44-188 nodes. The output node labels have value 1 if the node belongs to P and value 0 if it is in G.

I-b\ Stabilité : faire varier q

- When $q < 0.35$, it's easier at first to find the different community
- However, when $|q_{extra} - q|$ is big, the model starts to lose accuracy.
- The 20 epochs model seems to generalize better than the 80 epochs one.

q	Model 20 epochs	Model 80 epochs
0.15	**<u>77.0046</u>**	69.1981
0.25	**<u>93.2172</u>**	92.2461
0.35 (original model)	83.2934	**<u>84.7762</u>**
0.45	**<u>52.1648</u>**	51.5637
0.55	**<u>42.0210</u>**	41.8587



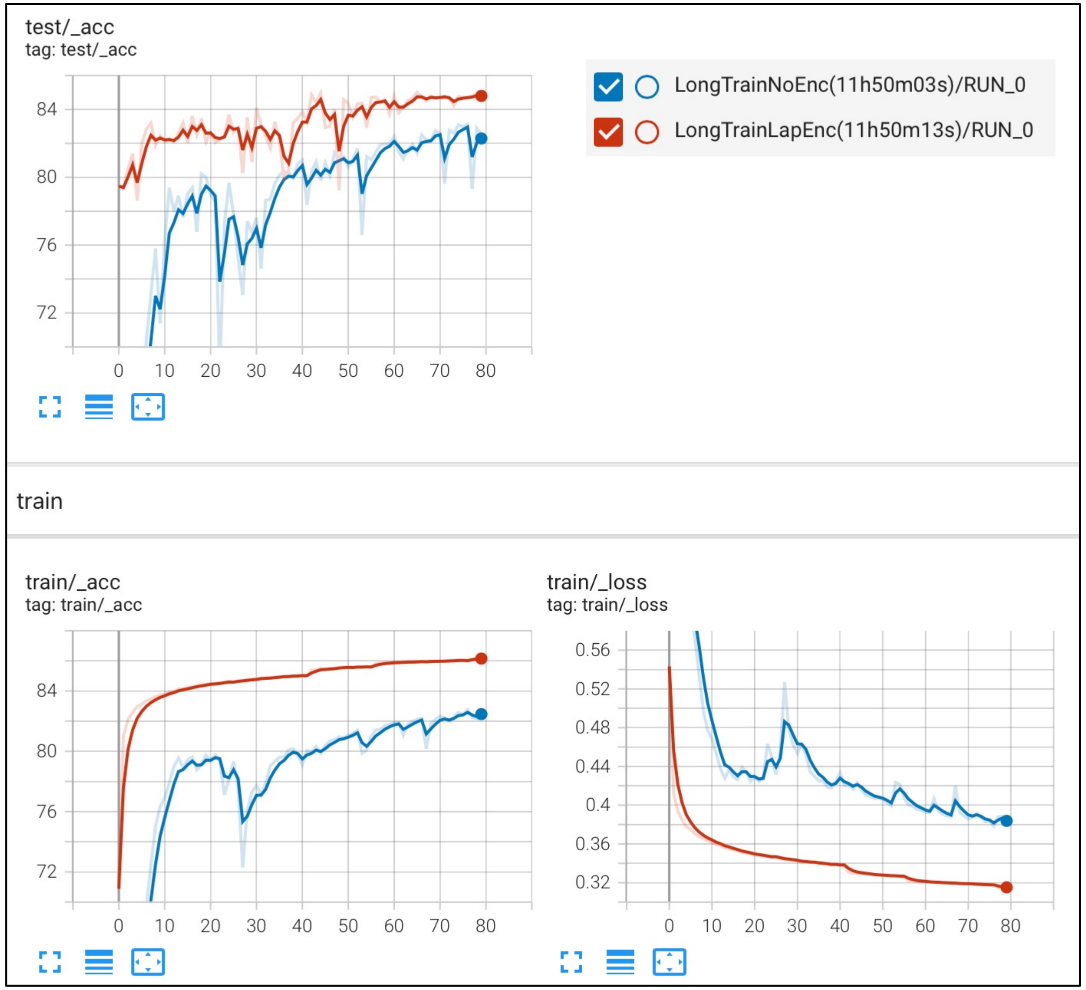
I-c\ Stability: size of the dataset

- As we could expect, the bigger the graph, the harder it is for the model
- The 20 epochs model seems to generalize better than the 80 epochs one

Communities sizes	Model 20 epochs	Model 80 epochs
Uniform between 5 and 35 (original)	83.2934	**84.7762**
Uniform between 35 and 65	**68.4562**	65.9456
Uniform between 65 and 95	**39.8145**	33.6427

2-a\ Performance – With or without embedding ?

- Much faster and precise training with the embedding.
- The gap is smaller and smaller with training
- About 20% accuracy gap for the first epoch.



2-b\ Performance – Exponential factors

- Exponential: give more weight to smaller eigenvalues :

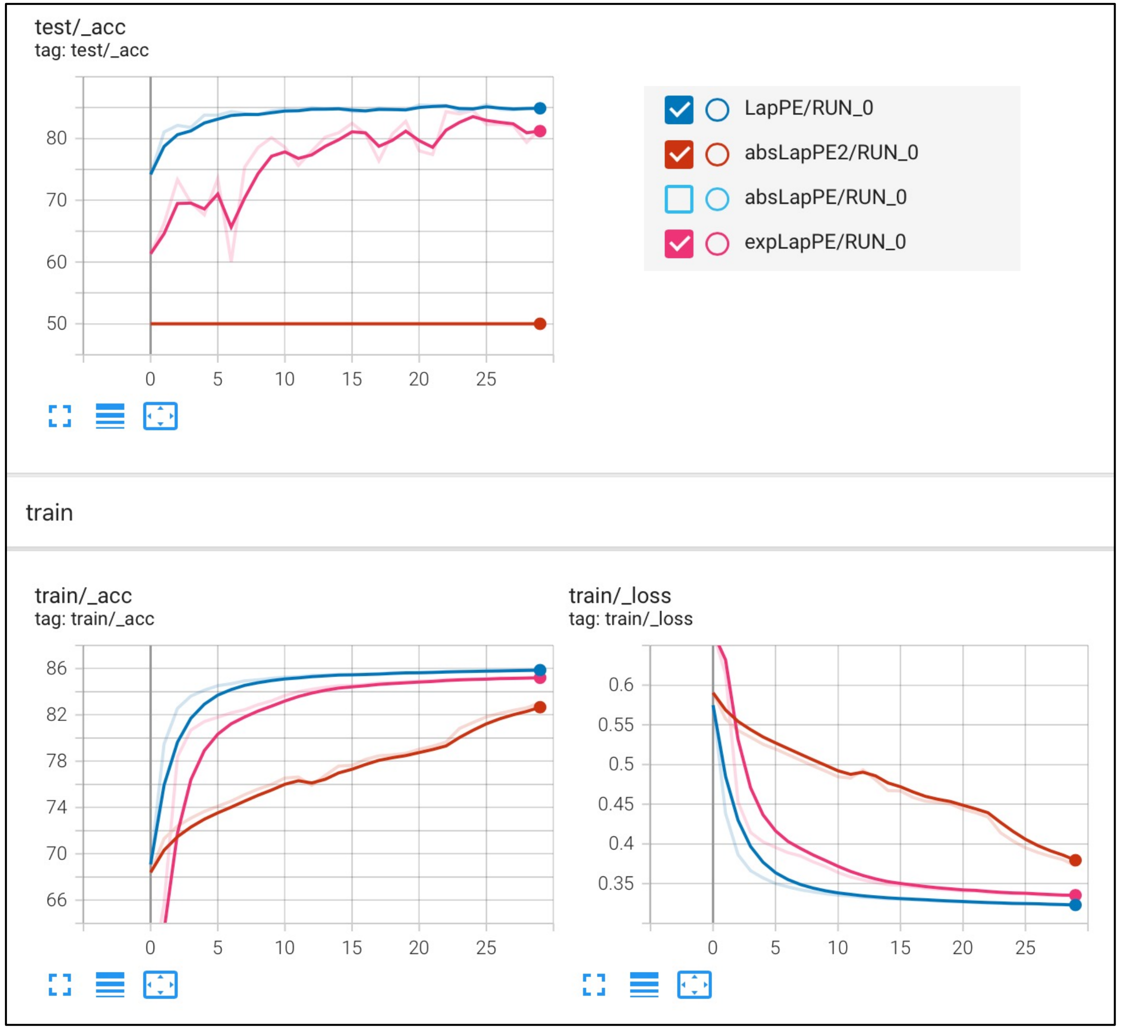
$$(U_1, ..., U_n) \Leftarrow (U_1 \exp^{-|\lambda_1|}, ..., U_n \exp^{-|\lambda_n|})$$

- Absolute value: increase sign symetry:

$$(U_1, ..., U_n) \Leftarrow (|U_1|, ..., |U_n|)$$

2-b\ Performance - Exponential factors

- Results:



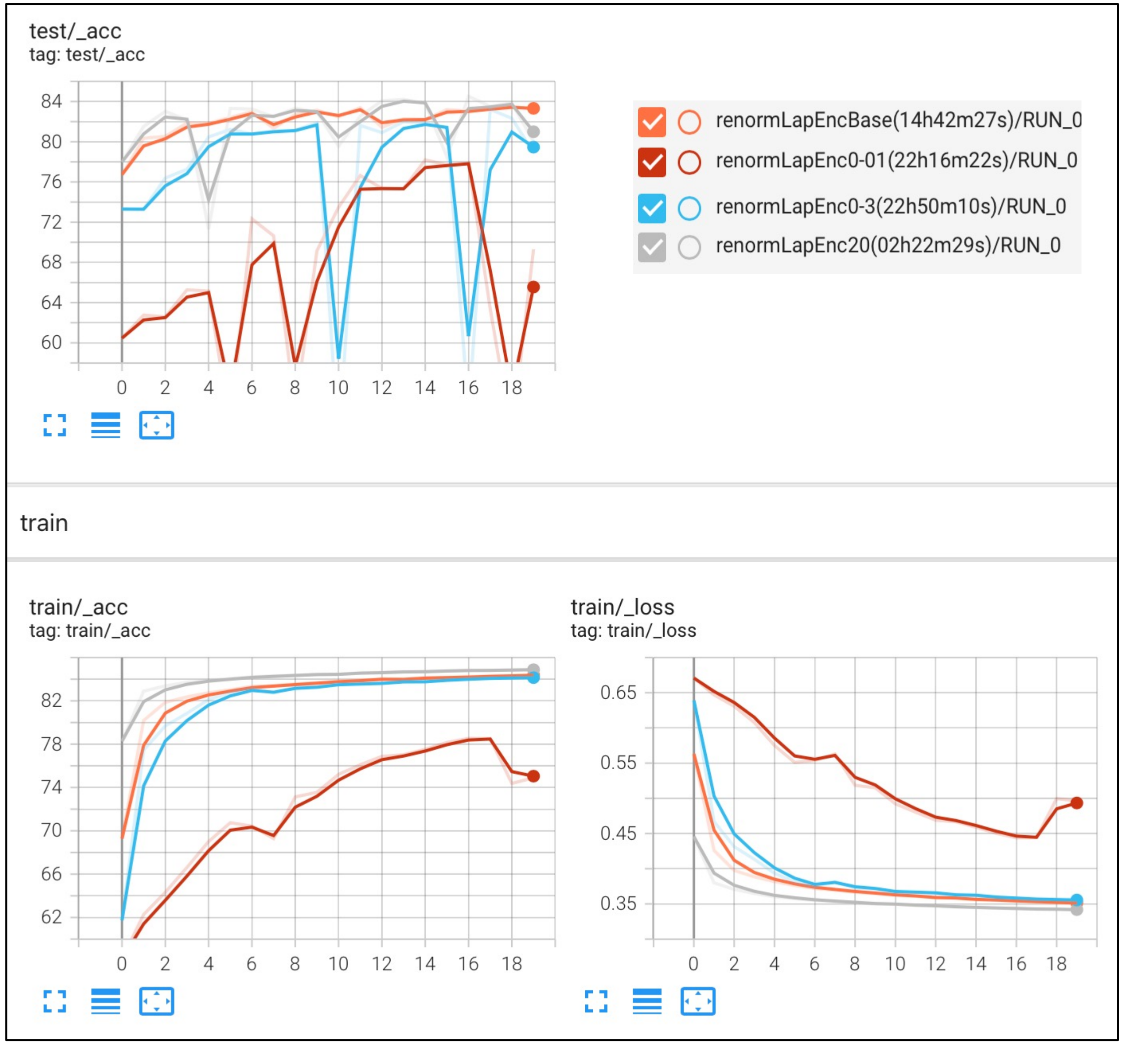
2-c\ Performance - Homothety

- Homothety :

$$(U_1, \ldots, U_n) \Leftarrow h \times (U_1, \ldots, U_n)$$

- In theory: no change due to the neural network

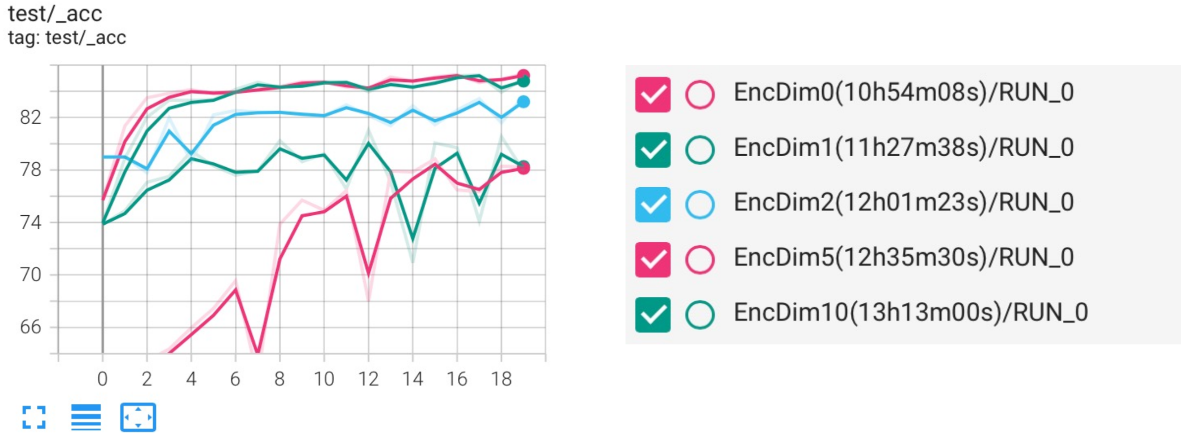
- In practice: difference since the signal is harder for the network to pick up information



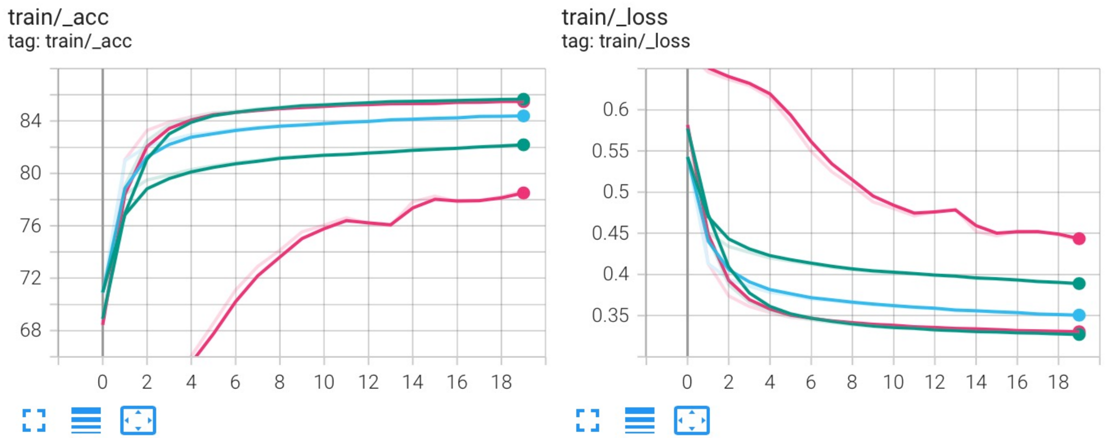
2-d\ Performance - Dimension

- The code of the paper uses the k=2 smallest eigenvalues of the Laplacian by default.

- Better performances can be achieved with k=5 in the long run. Few changes for k>5.



train



Thank you !