SCCGNN: any ideas?

Anton Savostianov[1] • **Francesco Tudisco**[1] • **Nicola Guglielmi**[1]

[1]Gran Sasso Science Institute, viale F.Crispi 7, L'Aquila, Italy, 67100, email: anton.savostianov@gssi.it

Abstract: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Keywords: NNs

I. Things

Let $\mathcal{K}$ be a simplicial complex where $\mathcal{V}_k(\mathcal{K})$ is the set of k-order simplices in $\mathcal{K}$, $\mathcal{K} = \mathcal{V}_0(\mathcal{K}) \cup \mathcal{V}_1(\mathcal{K}) \cup \mathcal{V}_2(\mathcal{K}) \cup \ldots$; let $m_k = |\mathcal{V}_k(\mathcal{K})|$. Let $B_k \in \mathrm{Mat}_{m_{k-1}\times m_k}$ be a boundary operator mapping simplex of order k to its border of the order $k-1$; by the fundamental lemma of topology $B_k B_{k+1} = 0$.

Given the lemma above, the homology group $\mathcal{H}_k = {}^{\text{im } B_k}\!/_{\ker B_{k+1}}$ is correctly defined; the elements of $\mathcal{H}_k$ correspond to k-dimensional holes in the simplicial complex $\mathcal{K}$. Through the harmonic representative it is convenient to exploit the isomorphism:

$$\mathcal{H}_k \cong \ker\left(B_k^\top B_k + B_{k+1}B_{k+1}^\top\right)$$

Operators $L_k^\downarrow = B_k^\top B_k$, $L_k^\uparrow = B_{k+1}B_{k+1}^\top$ and $L_k = L_k^\downarrow + L_k^\uparrow$ are referred as $k-$th order up-, down- and complete graph Laplacians.

Algebra of boundary operators admits an important full space decomposition:

Lemma 1 **(Hodge Decomposition, [Lim19, Thm. 5.2])** For the k-th order Hodge Laplacian $L_k \in \mathbb{R}^{m_k\times m_k}$, the following decomposition holds:

$$\mathbb{R}^{m_k} = \overbrace{\text{im } B_k^\top \oplus \underbrace{\ker L_k}_{\ker B_k} \oplus \text{im } B_{k+1}}^{\ker B_{k+1}^\top}$$

We also assume the case of weighted simplicial complex; in such situations the family of k weight functions are introduced: $w_k(\cdot) : \mathcal{V}_k(\mathcal{K}) \mapsto (0; +\infty)$ with corresponding diagonal weight matrices $W_k \in \mathrm{Mat}_{m_k\times m_k}$ where $[W_k]_{ii} = w_k(\sigma_i)$, $\sigma_i \in \mathcal{V}_k(\mathcal{K})$. Then the weighting scheme preserving the homology definition can be designed as follows:

$$B_k \to W_{k-1}^{-1} B_k W_k$$

Weighting does not change the dimensionality of the homology group and the remaining terms in the decomposition; isomorhism and the decomposition above hold in the weighted case as well.

here we need a proper discussion of the normalisation properties in the general case

II. SCCGNN

Graph Fourier Transform. Let L be a graph Laplacian (of any order); then L_k is simmetric positive semidefinite and has a complete set of orthonormal eigenvectors $U = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \boldsymbol{u}_n]$, $L_k = U\Lambda U^\top$. Then graph Fourier transform of a simplicial signal $\boldsymbol{x} \in C_k$ is defined as $\hat{\boldsymbol{x}} = U^\top \boldsymbol{x}$, [KW17], and the convolution on graphs:

$$\boldsymbol{x} \star \boldsymbol{y} = U\left(\left(U^\top \boldsymbol{x}\right) \odot \left(U^\top \boldsymbol{y}\right)\right) = U\,\mathrm{diag}\left(U^\top \boldsymbol{x}\right) U^\top \boldsymbol{y} \qquad \text{(Eqn. 1)}$$

exactly the same as in the discrete Fourier transform with a different basis instead of $e^{-2\pi ik/n}$

Then graph filter g_θ act as $g_\theta(L_k) = U g_\theta(\Lambda) U^\top$; since $g_\theta(\Lambda)$ is diagonal, every filter application $g_\theta(L_k)\boldsymbol{x}$ is a graph convolution. Hence, for each convolution of $\boldsymbol{x}$ with a kernel $\boldsymbol{w}$, one can find a graph filter

$$\operatorname{diag}\left(U^\top \boldsymbol{w}\right) = g_\theta(\Lambda) \tag{Eqn. 2}$$

Instead of the unpleasant and computationally demanding generic case of $g_\theta(L_k)$, one can use approximations by the polynomial filters of a smaller degree:

$$g_\theta(L_k) \approx \sum_{i=0}^{K} \theta_i T_i(\hat{L}_k) \approx \sum_{i=0}^{K} \alpha_i L_k^i \tag{Eqn. 3}$$

T_i are Chebyshev polynomials

Convolutional Layers. Let $\boldsymbol{x}_i \in C_1$; then a convolutional layer induced by the simplicial geometry is given by:

$$\boldsymbol{x}^{(n+1)} = \sigma\left(\sum_{i=0}^{K} \alpha_i^{(n)} L_1^i \boldsymbol{x}^{(n)}\right), \quad \text{where } \sigma = \text{reLU and } \alpha_i^{(n)} \in \mathbb{R}$$

where $\sigma(x) = x \cdot \chi(x)$. Note that $L_1^i = \left(B_1^\top B_1\right)^i + \left(B_2 B_2^\top\right)^i = L_1^{\downarrow i} + L_1^{\uparrow i}$; let $P_K(\boldsymbol{\alpha}, A) = \sum_{i=0}^{K} \alpha_i A^i$ be a matrix polynomial. Then, the convolution layer can be rewritten as

$$\begin{aligned} \boldsymbol{x}^{(n+1)} &= \sigma\left(\sum_{i=0}^{K} \alpha_i^{(n)} L_1^i \boldsymbol{x}^{(n)}\right) = \sigma\left(P_K(\boldsymbol{\alpha}^{(n)}, L_1)\boldsymbol{x}^{(n)}\right) = \\ &= \sigma\left(P_K(\boldsymbol{\alpha}^{(n)}, L_1^\downarrow)\boldsymbol{x}^{(n)} + P_K(\boldsymbol{\alpha}^{(n)}, L_1^\uparrow)\boldsymbol{x}^{(n)}\right) \end{aligned} \tag{Eqn. 4}$$

Equation above implies the same vector of polynomial coefficients for both up- and down-terms. In a more general case, one could consider a layer with two independent polynomials:

$$\boldsymbol{x}^{(n+1)} = \sigma\left(P_K(\boldsymbol{\alpha}^{(n)}, L_1^\downarrow)\boldsymbol{x}^{(n)} + Q_K(\boldsymbol{\beta}^{(n)}, L_1^\uparrow)\boldsymbol{x}^{(n)}\right) \tag{Eqn. 5}$$

Let us omit $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in the polynomial arguments for clarity (we will refer to the coefficients of P_K as α_i and Q_k as β_i respectively).

Assuming one can extend the dimension of the input for the hidden layers, $\boldsymbol{x}^{(n)} \in \mathbb{R}^{m_1 \times 1}$ and $\boldsymbol{x}^{(n+1)} \in \mathbb{R}^{m_1 \times H}$ and $\alpha_i^{(n)}, \beta_i^{(n)} \in \mathbb{R}^{1 \times H}$.

the order of mutliplication changes, $P_K(\boldsymbol{\alpha}^{(n)}, L_1)\boldsymbol{x}^{(n)} = \sum_{i=0}^{K} L_1^i \boldsymbol{x}^{(n)} \alpha_i^{(n)}$

Training, Batching, and Masking. Let $\boldsymbol{x} \in C_1$ be a given given flow on edges. We separate vector's entries into two parts — missing and complete: let $C \subset \overline{1..m_1}$ be a set of indices for the complete entries in $\boldsymbol{x}$ and $M \subset 1..m_1$ be a set of missing indices, $C \sqcup M = 1..m_1$. For each vector $\boldsymbol{x}$ we denote the vector of complete entries by $\boldsymbol{x}_c \in \mathbb{R}^{|C|}$ and the missing vector by $\boldsymbol{x}_m$ accordingly, $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_c \\ \boldsymbol{x}_m \end{bmatrix}$ modulo the permutation of entries.

Note that the missing part of the vector is missing and is unavailible during training. Instead, we assume that the vector $\widetilde{\boldsymbol{x}}$ that arrives as the input of SCCGNN is $\widetilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}_c \\ \widetilde{\boldsymbol{x}}_m \end{bmatrix}$ where $\widetilde{\boldsymbol{x}}_m = \phi$ where ϕ is a filler value for the missing entries, an initial guess. One may opt for $\phi = 0$ or $\phi = \text{median}\ (\boldsymbol{x}_c)$.

this is going to be a little shocking but bare with me

Def. 1 **(SCCGNN)** L-layer SCCGNN $f(\widetilde{\boldsymbol{x}}) = \boldsymbol{x}^{(L)}$ is defined as application of L consequetive convolutional layers with the activation function $\sigma(\cdot)$ starting from $\widetilde{\boldsymbol{x}}$:

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \widetilde{\boldsymbol{x}} \\ \boldsymbol{x}^{(1)} &= \sigma\left(P_K(\boldsymbol{\alpha}^{(0)}, L_1^\downarrow)\boldsymbol{x}^{(0)} + Q_K(\boldsymbol{\beta}^{(0)}, L_1^\uparrow)\boldsymbol{x}^{(0)}\right) \\ \dot{\boldsymbol{x}}^{(L)} &= \sigma\left(P_K(\boldsymbol{\alpha}^{(L-1)}, L_1^\downarrow)\boldsymbol{x}^{(L-1)} + Q_K(\boldsymbol{\beta}^{(L-1)}, L_1^\uparrow)\boldsymbol{x}^{(L-1)}\right) \end{aligned} \tag{Eqn. 6}$$

Problem

Train a SCCGNN $f(\widetilde{\boldsymbol{x}})$ composed of convolutional graph layers such that $f(\widetilde{\boldsymbol{x}})$ reconstructs the correct dependency on the missing values due to simplicial structure:

$$\|f(\widetilde{\boldsymbol{x}})_m - \boldsymbol{x}_m\|^2 \to \min \qquad \text{(Eqn. 7)}$$

without exploiting values from $\boldsymbol{x}_m$.

Remark (Batching and SGD without S) By the nature of the task $\boldsymbol{x}$ maybe the only data point available; this implies that data points cannot be grouped into batches and the optimisation with SGD loses the stochasticity (so, widely used `Adam` optimizer would run deterministic and highly dependent on the initial parameter values).

Naturally, one may try to sample entries of $\boldsymbol{x}$ to form a batch (in other words, pick up a subset of the edges and disregard all the others), but this brute force idea contradicts the interconnected nature of the graph neural network. One can find a substantial discussion on the proper subsampling for GNNs (which normally mean subsampling vertices with their neighbours via specifically constructed distribution, in other words), [HLK+20], but we propose the alternative batching and randomization.

Since the network has the access only to the known entries $\boldsymbol{x}_c$, it should be train to maintain correct values on the known entries (e.g. by minimizing $\|f(\widetilde{\boldsymbol{x}})_c - \boldsymbol{x}_c\|$) which has a global (but maybe not unique) optimum at $f(\widetilde{\boldsymbol{x}}) = \widetilde{\boldsymbol{x}}$ (identity). In order to avoid the trivial and incorrect sollution one may propose the following masking idea in order to simulate the same "missing/complete" partition on the known entries:

- let $\boldsymbol{e}$ be a binary vector such that $\boldsymbol{e}_m = 0$ and each entry of in the complete part $\boldsymbol{e}_c$ be $\sim \text{Bernoulli}\,(p)$;
- then $\boldsymbol{x} \odot \boldsymbol{e}$ is a subsampling where we "forget" about some of the known entries;
- let us train the network with a loss:

$$\|f(\widetilde{\boldsymbol{x}} \odot \boldsymbol{e})_c - \boldsymbol{x}_c\|^2 \qquad \text{(Eqn. 8)}$$

  so it avoids the identity and tries to keep the original known values intact;
- then the epoch is naturally formed by $C \ln m_1/p$ steps (such that each entry of $\boldsymbol{x}_c$ enters the training).

Generated data. We consider the simplicial complex $\mathcal{K}$ generated by the Delaunay triangulation of the unit square: the vertex set $\mathcal{V}_0(\mathcal{K})$ is formed by the corners of the square and $(m_0 - 4)$ uniformly sampled points inside the square; $\mathcal{V}_1(\mathcal{K})$ and $\mathcal{V}_2(\mathcal{K})$ are provided by the triangulation. In order to obtain a non-trivial $\mathcal{H}_1$, we exclude 2 random triangles from $\mathcal{V}_2(\mathcal{K})$, so $\dim \mathcal{H}_1 = 2$.

- weights of the edges chosen almost unit: $w_1(\sigma) = 1 + \mathcal{N}\left(0, \frac{1}{100}\right)$, modulo the exception for the unstable complex described below;
- weights of the vertices are cumulitive weights of the adjacent edges: $w_0(v) = \sum_{\substack{\sigma \in \mathcal{V}_1(\mathcal{K}) \\ v \in \sigma}} w_1(\sigma)$;
- weights of the triangles are following the min-rule: $w_2(\tau) = \min_{\substack{\sigma \in \mathcal{V}_1(\mathcal{K}) \\ \sigma \in \tau}} w_1(\sigma)$.

We use two setups for experiments: the stable $\mathcal{K}$, described above, and the unstable were two (randomly chosen) edges have a small weight $\varepsilon = 10^{-2}$. Defined instability agrees with both definition proposed by us in previous works: it takes a small edge perturbation to create another dimension in $\mathcal{H}_1$ and the conditionality of $L_1^{\uparrow}$ and L_1 is horrifying.

The target vector $\boldsymbol{x} \in C_1$ is produced as follows: we task SCCGNN to reconstruct some simple function of the local simplex structure. For instance, in the experiments below, we use the sum of the adjacent to the edge triangles:

$$x_\sigma = \sum_{\substack{\tau \in \mathcal{V}_2(\mathcal{K}) \\ \sigma \in \tau}} w_2(\tau) + \mathcal{N}\left(0, \frac{1}{100}\right) \qquad \text{(Eqn. 9)}$$

Technical details. We use 3-layer SCCGNN with $K = 3$ filter size (no more than cubic polynomial degree); the share of missing values in $\widetilde{\boldsymbol{x}}$ is $\nu = 0.25$; the filler value ϕ is chosen to be the median of known values.

The complex $\mathcal{K}$ is generated by the triangulation of 14 total nodes on the unit square; number of edges $m_1 = 35$, number of triangles $m_2 = 20$.

One epoch is composed of 20 batches, each batch has the entry presence probability of $p = \frac{1}{2}$; the training lasts no longer than 5000 epochs or untill the loss falls below 10^{-2}.

The loss is measured by MSE:

$$\mathcal{L} = \frac{1}{|C|} \left\| f(\widetilde{\boldsymbol{x}} \odot \boldsymbol{e})_c - \boldsymbol{x}_c \right\|^2 \qquad \text{(Eqn. 10)}$$

The accuracy on the missing value is measure by `r2score`:

$$r^2 = 1 - \frac{\frac{1}{|M|} \left\| f(\widetilde{\boldsymbol{x}})_m - \boldsymbol{x}_m \right\|^2}{\mathtt{Var}\left(f(\widetilde{\boldsymbol{x}})_m \right)} \qquad \text{(Eqn. 11)}$$

III. Flow leakage and First Experiments.

III.I Flow components in SCCGNN.

For the Hodge decomposition, Lemma 1, we define projectors on $\operatorname{im} B_1^\top$ and $\operatorname{im} B_2$ as $\Pi_1 = B_1^\top \left(B_1 B_1^\top \right)^+ B_1$ and $\Pi_2 = B_2 \left(B_2^\top B_2 \right)^+ B_2^\top$ respectively.

note that one can use stochastic Cholesky and `HeCS` as fast projectors here

Then

$$\begin{aligned} \boldsymbol{x}^{(n+1)} &= \sigma\left(P_K(\boldsymbol{\alpha}^{(n)}, L_1^{\downarrow}) \boldsymbol{x}^{(n)} + Q_K(\boldsymbol{\beta}^{(n)}, L_1^{\uparrow}) \boldsymbol{x}^{(n)} \right) = \\ &= \sigma\left(\gamma^{(n)} \boldsymbol{x}^{(n)} + P_{K-1}(L_1^{\downarrow}) L_1^{\downarrow} \boldsymbol{x}^{(n)} + Q_{K-1}(L_1^{\uparrow}) L_1^{\uparrow} \boldsymbol{x}^{(n)} \right) \end{aligned} \qquad \text{(Eqn. 12)}$$

Let $\boldsymbol{x}^{(n)} = \boldsymbol{y}^{(n)} + \boldsymbol{h}^{(n)} + \boldsymbol{z}^{(n)}$ such that $\boldsymbol{y}^{(n)} \in \operatorname{im} B_1^\top$, $\boldsymbol{z}^{(n)} \in \operatorname{im} B_2$ and $\boldsymbol{h}^{(n)} \in \ker L_1$. Then:

$$\boldsymbol{x}^{(n+1)} = \sigma\left(\gamma^{(n)} \boldsymbol{h}^{(n)} + \left[\gamma^{(n)} I + P_{K-1}(L_1^{\downarrow}) L_1^{\downarrow} \right] \boldsymbol{y}^{(n)} + \left[\gamma^{(n)} I + Q_{K-1}(L_1^{\uparrow}) L_1^{\uparrow} \right] \boldsymbol{z}^{(n)} \right) \qquad \text{(Eqn. 13)}$$

The last equation implies a certain structural problem in SCCGNN: the harmonic component $\boldsymbol{h}^{(n)}$ is "underrepresented" in the layer and is acted upon only by a constant $\gamma^{(n)}$ and activation function. Let us investigate the behaviour for test settings.

formally, that's not true: the action of σ on $\boldsymbol{y}^{(n)}$ and $\boldsymbol{z}^{(n)}$ components has non-zero projections on $\ker L_1$. Nevertheless, it looks suspicious

III.II Follow the flow.

We consider the setting of SCCGNN described above; note that in this experiment the output of each layer never widens, so each layer receives and return a vector of the size $m_1 \times 1$. We initialize the parameters of the convolutional filters randomly; SCCGNN is trained till the loss falls under the threshold. We record the accuracy (in r^2 terms) at the final point of training alongside with component-wise losses $\frac{1}{m_1} \left\| \Pi_1 \left(f(\widetilde{\boldsymbol{x}}) - \boldsymbol{x} \right) \right\|^2$, $\frac{1}{m_1} \left\| \Pi_2 \left(f(\widetilde{\boldsymbol{x}}) - \boldsymbol{x} \right) \right\|^2$ and $\frac{1}{m_1} \left\| \Pi_{\ker L_1} \left(f(\widetilde{\boldsymbol{x}}) - \boldsymbol{x} \right) \right\|^2$.

Both Figures III.1 and III.2 demonstrate final accuracies (on the left) and final component-wise losses (on the right) for several initialisations of SCCGNN in the case of topologically stable and unstable simplicial complexes $\mathcal{K}$ correspondingly. Final accuracies in neither stable nor unstable can be treated as particularly successfull ones (the optimal value of r^2 metrics is 1). However, the stable case significantly overperforms the unstable one in terms of the r^2-score whilst the training loss in all the cases is successfully reduced below the threshold.

More interestingly, the component-wise losses demonstrate an accentuated order: the edge-vertex part of the flow ($\operatorname{im} B_1^\top$) is trained significantly better than both harmonic and

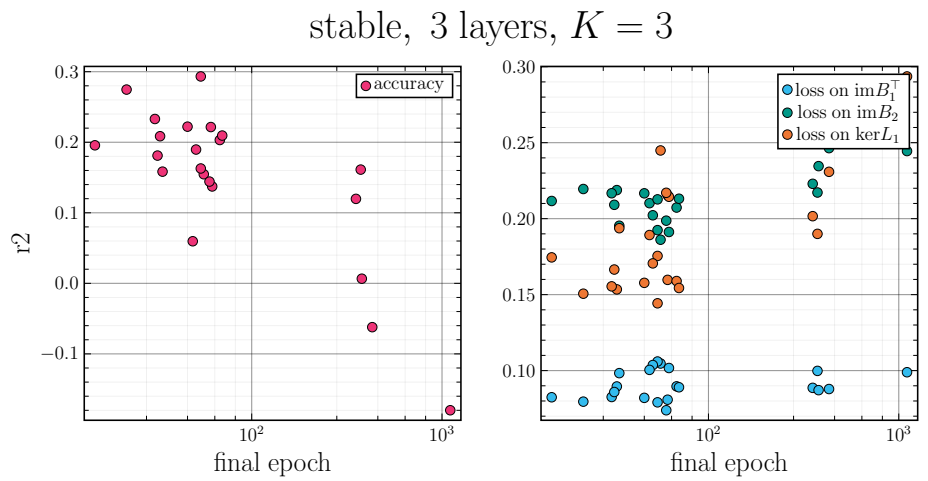

Figure III.1: Final accuracies (on the left) and component-wise losses (on the right) for 3-layer SCCGNN with cubic convolutional filter for several re-initializations. $\mathcal{K}$ is Deluanay-based simplicial complex with a stable weight scheme.

triangular parts. Additionally, the difference between the stable and unstable cases lies in the "order" of losses: the unstable case has the harmonic loss as the worst one (which we could have expected) whilst the triangular component underperforms in the stable case whic

I mean, more questions

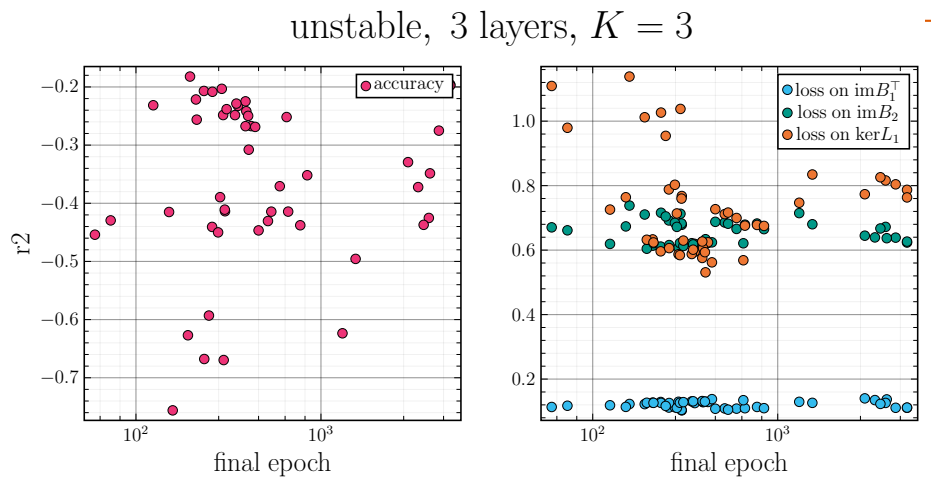

Figure III.2: Final accuracies (on the left) and component-wise losses (on the right) for 3-layer SCCGNN with cubic convolutional filter for several re-initializations. $\mathcal{K}$ is Deluanay-based simplicial complex with a unstable weight scheme.

Quick remarks.

- ♢ we can try to mitigate the problem with overall accuracy by actually widening the output of the layer (so the weights would be not scalars, but matrices), like in [EDS20];
- ♢ Figures III.1 and III.2 imply that we may benefit from the component-wise loss for the training (where you can add weights to the components depending on the conditionality of the initial complex);
- ♢ hypothetical switch to the component-wise loss would require fast projectors which seems to exploit our precious `HeCS` preconditioner and isn't is just the dream?
- ♢ everything is weird and unstable.

III. References

[EDS20] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial Neural Networks, December 2020.

[HLK+20] Yaochen Hu, Amit Levi, Ishaan Kumar, Yingxue Zhang, and Mark Coates. On Batch-size Selection for Stochastic Training for Graph Neural Networks. October 2020.

[KW17] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017.

[Lim19] Lek-Heng Lim. Hodge Laplacians on graphs, August 2019.

[YI23] Maosheng Yang and Elvin Isufi. Convolutional Learning on Simplicial Complexes, January 2023.