GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings

1. Proofs

Lemma 1. *Let* $\textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}$ *and* $\textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}$ *be Lipschitz continuous functions with Lipschitz constants* k_1 *and* k_2*, respectively. If, for all* $v \in \mathcal{V}$*, the inputs are close to the exact input,* i.e. $\|\tilde{\boldsymbol{h}}^{(\ell-1)}_v - \boldsymbol{h}^{(\ell-1)}_v\| \leq \delta$*, and the historical embeddings do not run too stale,* i.e. $\|\bar{\boldsymbol{h}}^{(\ell-1)}_v - \tilde{\boldsymbol{h}}^{(\ell-1)}_v\| \leq \epsilon$*, then the output error is bounded by*

$$\|\tilde{\boldsymbol{h}}^{(\ell)}_v - \boldsymbol{h}^{(\ell)}_v\| \leq \delta\, k_2 + (\delta + \epsilon)\, k_1\, k_2\, |\mathcal{N}(v)|.$$

Proof. By triangular inequality, it holds that $\|\bar{\boldsymbol{h}}^{(\ell-1)}_v - \boldsymbol{h}^{(\ell-1)}_v\| \leq \delta + \epsilon$. Since both $\textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}$ and $\textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}$ denote Lipschitz continuous functions with Lipschitz constants k_1 and k_2, respectively, it further holds that for any $\boldsymbol{x}, \boldsymbol{y}$:

$$\begin{aligned}\|\textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{y})\| &\leq k_1 \|\boldsymbol{x} - \boldsymbol{y}\| \\ \|\textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{y})\| &\leq k_2 \|\boldsymbol{x} - \boldsymbol{y}\|\end{aligned}$$

Furthermore, the Lipschitz constants for the aggregations $\sum_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}$, $\frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}$ and $\max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}$ are given as $|\mathcal{X}|$, 1 and 1, respectively. Then,

$$\begin{aligned}&\|\textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{h}}^{(\ell-1)}_v, \bigoplus_{w \in \mathcal{N}(v)} \textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}(\bar{\boldsymbol{h}}^{(\ell-1)}_w)) - \textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{h}^{(\ell-1)}_v, \bigoplus_{w \in \mathcal{N}(v)} \textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}(\boldsymbol{h}^{(\ell-1)}_w))\| \\ &\leq k_2\, (\delta + |\mathcal{N}(v)|\, (k_1\, (\delta + \epsilon))) = \delta\, k_2 + (\delta + \epsilon)\, k_1\, k_2\, |\mathcal{N}(v)|. \qquad \square\end{aligned}$$

Theorem 2. *Let* $\boldsymbol{f}^{(L)}_{\boldsymbol{\theta}}$ *be a L-layered GNN, containing only Lipschitz continuous* $\textsc{Message}^{(\ell)}_{\boldsymbol{\theta}}$ *and* $\textsc{Update}^{(\ell)}_{\boldsymbol{\theta}}$ *functions with Lipschitz constants* k_1 *and* k_2*, respectively. If, for all* $v \in \mathcal{V}$ *and all* $\ell \in \{1, \ldots, L-1\}$*, the historical embeddings do not run too stale,* i.e. $\|\bar{\boldsymbol{h}}^{(\ell)}_v - \tilde{\boldsymbol{h}}^{(\ell)}_v\| \leq \epsilon^{(\ell)}$*, then the final output error is bounded by*

$$\|\tilde{\boldsymbol{h}}^{(L)}_{v,j} - \boldsymbol{h}^{(L)}_{v,j}\| \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)}\, k_1^{L-\ell}\, k_2^{L-\ell}\, |\mathcal{N}(v)|^{L-\ell}.$$

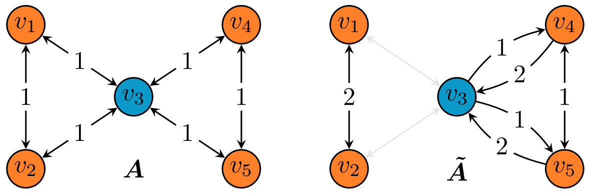
Proof. For layer $\ell = 1$, the inputs do not need to be estimated, *i.e.* $\delta^{(0)} = \|\tilde{\boldsymbol{h}}^{(0)}_v - \boldsymbol{h}^{(0)}_v\| = 0$, and, as a result, the output is *exact*, *i.e.* $\delta^{(1)} = \|\tilde{\boldsymbol{h}}^{(1)}_v - \boldsymbol{h}^{(1)}_v\| = 0$. With $\|\bar{\boldsymbol{h}}^{(1)}_v - \tilde{\boldsymbol{h}}^{(1)}_v\| \leq \epsilon^{(1)}$, it directly follows via Lemma 1 that the approximation error of layer $\ell = 2$ is bounded by $\|\tilde{\boldsymbol{h}}^{(2)}_v - \boldsymbol{h}^{(2)}_v\| \leq \epsilon^{(1)}\, k_1\, k_2\, |\mathcal{N}(v)| = \delta^{(2)}$. Recursively replacing

$$\delta^{(\ell)} = \delta^{(\ell-1)}\, k_2 + (\delta^{(\ell-1)} + \epsilon^{(\ell-1)})\, k_1\, k_2\, |\mathcal{N}(v)|$$

in $\|\tilde{\boldsymbol{h}}^{(L)}_v - \boldsymbol{h}^{(L)}_v\| \leq \delta^{(L-1)}\, k_2 + (\delta^{(L-1)} + \epsilon^{(L-1)})\, k_1\, k_2\, |\mathcal{N}(v)|$ (*cf.* Lemma 1) yields

$$\|\tilde{\boldsymbol{h}}^{(L)}_v - \boldsymbol{h}^{(L)}_v\| \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)}\, k_1^{L-\ell}\, k_2^{L-\ell}\, |\mathcal{N}(v)|^{L-\ell}. \qquad \square$$

Proposition 3. *Let $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)} \colon \mathcal{V} \to \mathbb{R}^d$ be a L-layered GNN as expressive as the WL test in distinguishing the L-hop neighborhood around each node $v \in \mathcal{V}$. Then, there exists a graph $\boldsymbol{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ for which $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)}$ operating on a sampled variant $\tilde{\boldsymbol{A}}$, $\tilde{a}_{v,w} = \begin{cases} \frac{|\mathcal{N}(v)|}{|\tilde{\mathcal{N}}(v)|}, & \text{if } w \in \tilde{\mathcal{N}}(v) \\ 0, & \text{otherwise} \end{cases}$, produces a non-equivalent coloring,* i.e. *$\tilde{\boldsymbol{h}}_v^{(L)} \neq \tilde{\boldsymbol{h}}_w^{(L)}$ while $c_v^{(L)} = c_w^{(L)}$ for nodes $v, w \in \mathcal{V}$.*



Proof. Consider the colored graph $\boldsymbol{A}$ and its sampled variant $\tilde{\boldsymbol{A}}$ as shown on the right. Here, it holds that $\boldsymbol{h}_{v_1}^{(1)} = \boldsymbol{h}_{v_4}^{(1)}$ while $\tilde{\boldsymbol{h}}_{v_1}^{(1)} \neq \tilde{\boldsymbol{h}}_{v_4}^{(1)}$. □

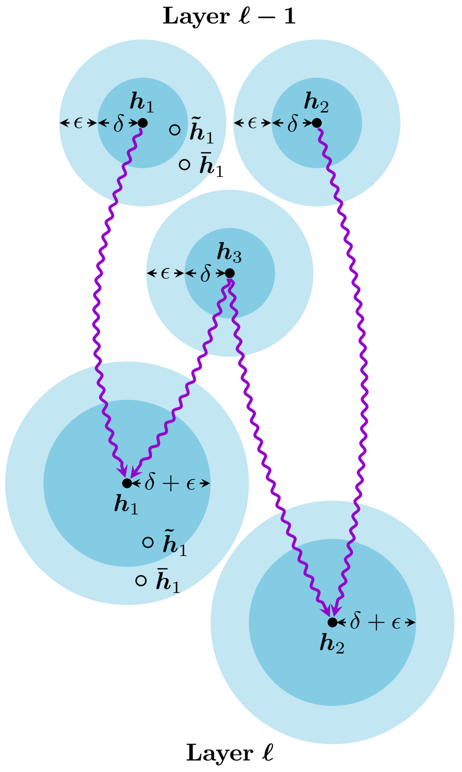
Lemma 4. *Let $\{\!\{\boldsymbol{h}_v^{(\ell-1)} \colon v \in \mathcal{V}\}\!\}$ be a countable multiset such that $\|\boldsymbol{h}_v^{(\ell-1)} - \boldsymbol{h}_w^{(\ell-1)}\| > 2(\delta+\epsilon)$ for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(\ell-1)} \neq \boldsymbol{h}_w^{(\ell-1)}$. If the inputs are close to the exact input,* i.e. *$\|\tilde{\boldsymbol{h}}_v^{(\ell-1)} - \boldsymbol{h}_v^{(\ell-1)}\| \leq \delta$, and the historical embeddings do not run too stale,* i.e. *$\|\bar{\boldsymbol{h}}_v^{(\ell-1)} - \tilde{\boldsymbol{h}}_v^{(\ell-1)}\| \leq \epsilon$, then there exist $\text{MESSAGE}_{\boldsymbol{\theta}}^{(\ell)}$ and $\text{UPDATE}_{\boldsymbol{\theta}}^{(\ell)}$ functions, such that*

$$\|\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\tilde{\boldsymbol{h}}_v^{(\ell-1)}) - \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{h}_v^{(\ell-1)})\| \leq \delta + \epsilon$$

and

$$\|\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{h}_v^{(\ell-1)}) - \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{h}_w^{(\ell-1)})\| > 2(\delta + \epsilon + \lambda)$$

for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(\ell-1)} \neq \boldsymbol{h}_w^{(\ell-1)}$ and all $\lambda > 0$.



Proof. Define $\boldsymbol{\phi} \colon \mathbb{R}^d \to \mathbb{R}^d$ as the Voronoi tessellation induced by exact inputs $\{\boldsymbol{h}_v^{(\ell-1)} \colon v \in \mathcal{V}\}$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{h}_v^{(\ell-1)} \quad \text{if} \quad \|\boldsymbol{x} - \boldsymbol{h}_v^{(\ell-1)}\| \leq \|\boldsymbol{x} - \boldsymbol{h}_w^{(\ell-1)}\| \quad \text{for all } v \neq w \in \mathcal{V} \tag{1}$$

Furthermore, we know that there exists $\text{Message}_{\boldsymbol{\theta}}^{(\ell)}$ and $\text{UPDATE}_{\boldsymbol{\theta}}^{(\ell)}$ functions so that $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$ is injective for all countable multisets (Zaheer et al., 2017; Xu et al., 2019; Morris et al., 2019; Maron et al., 2019). Therefore, it holds that $\|\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\tilde{\boldsymbol{h}}_v^{(\ell-1)})) - \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\boldsymbol{h}_v^{(\ell-1)}))\| = 0 \leq \delta$. Since $\{\!\{\boldsymbol{h}_v^{(\ell-1)} \colon v \in \mathcal{V}\}\!\}$ is countable and $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$ is injective, there exists a $\kappa > 0$ such that $\|\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\boldsymbol{h}_v^{(\ell-1)})) - \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\boldsymbol{h}_w^{(\ell-1)}))\| > \kappa$ for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(\ell-1)} \neq \boldsymbol{h}_w^{(\ell-1)}$. Due to the homogeneity of $\|\cdot\|$, it directly follows that there must exists $\alpha > 0$ so that

$$\|\alpha \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\boldsymbol{h}_v^{(\ell-1)})) - \alpha \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{\phi}(\boldsymbol{h}_w^{(\ell-1)}))\| > \alpha\,\kappa \geq 2(\delta + \epsilon + \lambda)$$

for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(\ell-1)} \neq \boldsymbol{h}_w^{(\ell-1)}$ and all $\lambda > 0$. □

Theorem 5. *Let $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)}$ be a L-layered GNN in which all $\text{MESSAGE}_{\boldsymbol{\theta}}^{(\ell)}$ and $\text{UPDATE}_{\boldsymbol{\theta}}^{(\ell)}$ functions fulfill the conditions of Lemma 4. Then, there exists a map $\phi \colon \mathbb{R}^d \to \Sigma$ so that $\phi(\tilde{\boldsymbol{h}}_v^{(L)}) = c_v^{(L)}$ for all $v \in \mathcal{V}$.*

Proof. Define $\phi \colon \mathbb{R}^d \to \Sigma$ as the Voronoi tessellation induced by exact outputs $\{\boldsymbol{h}_v^{(L)} \colon v \in \mathcal{V}\}$:

$$\phi(\boldsymbol{x}) = c_v^{(L)} \quad \text{if} \quad \|\boldsymbol{x} - \boldsymbol{h}_v^{(L)}\| \leq \|\boldsymbol{x} - \boldsymbol{h}_w^{(L)}\| \quad \text{for all } v \neq w \in \mathcal{V}$$

Since each GNN layer $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$ is injective for exact inputs, we know that such a function needs to exist (Xu et al., 2019; Morris et al., 2019). Therefore, it is sufficient to show that there exists a $\delta^{(L)} > 0$ so that $\|\tilde{\boldsymbol{h}}_v^{(L)} - \boldsymbol{h}_v^{(L)}\| \leq \delta^{(L)}$ and $\|\boldsymbol{h}_v^{(L)} - \boldsymbol{h}_w^{(L)}\| > 2\delta^{(L)}$ for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(L)} \neq \boldsymbol{h}_w^{(L)}$. Following upon Theorem 2, we know that $\|\tilde{\boldsymbol{h}}_v^{(1)} - \boldsymbol{h}_v^{(1)}\| = 0$. Due to Lemma 4, it holds that $\|\tilde{\boldsymbol{h}}_v^{(2)} - \boldsymbol{h}_v^{(2)}\| \leq \epsilon^{(1)}$. The next layer introduces an increased error, *i.e.* $\|\bar{\boldsymbol{h}}_v^{(2)} - \boldsymbol{h}_v^{(2)}\| \leq \epsilon^{(1)} + \epsilon^{(2)}$, and to compensate, we set $\lambda^{(2)} = \epsilon^{(2)}$ so that $\|\boldsymbol{h}_v^{(2)} - \boldsymbol{h}_w^{(2)}\| > 2\,(\epsilon^{(1)} + \epsilon^{(2)})$ for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(L)} \neq \boldsymbol{h}_w^{(L)}$. By recursively applying Lemma 4 with $\lambda^{(\ell)} = \epsilon^{(\ell)}$, it immediately follows that $\|\tilde{\boldsymbol{h}}_v^{(L)} - \boldsymbol{h}_v^{(L)}\| \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)} = \delta^{(L)}$, and $\|\tilde{\boldsymbol{h}}_v^{(L)} - \boldsymbol{h}_w^{(L)}\| > \sum_{\ell=1}^{L-1} 2\,\epsilon^{(\ell)}$ for all $v, w \in \mathcal{V}$, $\boldsymbol{h}_v^{(L)} \neq \boldsymbol{h}_w^{(L)}$. □

2. Algorithm

Our GAS mini-batch training algorithm is given in Algorithm 1:

Algorithm 1 GAS Mini-batch Execution

```
Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, input node features $\boldsymbol{H}^{(0)}$, number of batches $B$, number of layers $L$
$\{\mathcal{B}_1, \ldots, \mathcal{B}_B\} \leftarrow \textsc{Split}(\mathcal{G}, B)$
$\mathcal{V}_b \leftarrow \bigcup_{v \in \mathcal{B}_b} \mathcal{N}(v) \cup \{v\}$        $\forall b \in \{1, \ldots, B\}$
$\mathcal{G}_b \leftarrow \mathcal{G}[\mathcal{V}_b]$        $\forall b \in \{1, \ldots, B\}$
for $\mathcal{B}_b \in \{\mathcal{B}_1, \ldots, \mathcal{B}_B\}$ do
  for $\ell \in \{1, \ldots, L-1\}$ do
    $\boldsymbol{h}_v^{(\ell)} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}(\boldsymbol{h}_v^{(\ell-1)}, \{\!\{\boldsymbol{h}_w^{(\ell-1)} : w \in \mathcal{N}(v)\}\!\})$        $\forall v \in \mathcal{B}_b$
    $\textsc{Push}^{(\ell)}(\boldsymbol{h}_v^{(\ell)})$        $\forall v \in \mathcal{B}_b$
    $\boldsymbol{h}_w^{(\ell)} \leftarrow \textsc{Pull}^{(\ell)}(w)$        $\forall w \in \mathcal{V}_b \setminus \mathcal{B}_b$
  end for
  $\boldsymbol{h}_v^{(L)} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}}^{(L)}(\boldsymbol{h}_v^{(L-1)}, \{\!\{\boldsymbol{h}_w^{(L-1)} : w \in \mathcal{N}(v)\}\!\})$        $\forall v \in \mathcal{B}_b$
end for
```

3. GNN Operators

We briefly recap the details of all graph convolutional layers used in our experiments. We omit final non-linearities and edge features due to simplicity.

Graph Convolutional Networks (GCN) use a symmetrically normalized mean aggregation followed by linear transformation (Kipf & Welling, 2017)

$$\boldsymbol{h}_v^{(\ell)} = \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \boldsymbol{W} \boldsymbol{h}_w^{(\ell-1)},$$

where $c_{w,v} = \sqrt{\deg(w) + 1}\sqrt{\deg(v) + 1}$.

Graph Attention Networks (GAT) perform an anisotropic aggregation (Veličković et al., 2018)

$$\boldsymbol{h}_v^{(\ell)} = \sum_{w \in \mathcal{N}(v) \cup \{v\}} \alpha_{w,v} \boldsymbol{W} \boldsymbol{h}_w^{(\ell-1)},$$

where normalization is achieved via learnable attention coefficients

$$\alpha_{w,v} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\boldsymbol{a}^\top \left[\boldsymbol{W}\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{W}\boldsymbol{h}_w^{(\ell-1)}\right]\right)\right)}{\sum_{k \in \mathcal{N}(v) \cup \{v\}} \exp\left(\mathrm{LeakyReLU}\left(\boldsymbol{a}^\top \left[\boldsymbol{W}\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{W}\boldsymbol{h}_k^{(\ell-1)}\right]\right)\right)}.$$

Approximate Personalized Propagation of Neural Predictions (APPNP) networks first perform a graph-agnostic prediction of node labels, *i.e.* $\boldsymbol{h}_v^{(0)} = \mathrm{MLP}(\boldsymbol{x}_v)$, and smooth initial label predictions via propagation afterwards (Klicpera et al., 2019)

$$\boldsymbol{h}^{(\ell)} = \alpha\, \boldsymbol{h}^{(0)} + (1-\alpha) \sum_{w \in \mathcal{N} \cup \{v\}} \frac{1}{c_{w,v}} \boldsymbol{h}_w^{(\ell-1)},$$

where $\alpha \in [0,1]$ denotes the teleport probability and $c_{w,v}$ is defined as in GCN. Notably, the final propagation layers are non-trainable, and predictions are solely conditioned on node features (while gradients of model parameters are not).

Simple and Deep Graph Convolutional Networks (GCNII) extend the idea of APPNP to a trainable propgation scheme which leverages initial residual connections (Chen et al., 2020)

$$\boldsymbol{h}_v^{(\ell)} = \alpha \boldsymbol{W} \boldsymbol{h}_v^{(0)} + (1-\alpha) \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \boldsymbol{W} \boldsymbol{h}_w^{(\ell-1)},$$

and $\boldsymbol{W}$ makes use of identity maps, *i.e.* $\boldsymbol{W} \leftarrow (1-\beta)\boldsymbol{I} + \beta \boldsymbol{W}$ for $\beta \in [0,1]$.

Graph Isomorphism Networks (GIN) make use of sum aggregation and MLPs to obtain a maximally powerful GNN operator (Xu et al., 2019)

$$\boldsymbol{h}_v^{(\ell)} = \mathrm{MLP}_{\boldsymbol{\theta}} \left((1+\epsilon)\, \boldsymbol{h}_v^{(\ell-1)} + \sum_{w \in \mathcal{N}(v)} \boldsymbol{h}_w^{(\ell-1)} \right),$$

where $\epsilon \in \mathbb{R}$ is a trainable parameter in order to distinguish neighbors from central nodes.

Principal Neighborhood Aggregation (PNA) networks leverage mulitple aggregators combined with degree-scalers to capture graph structural properties (Corso et al., 2020)

$$\boldsymbol{h}_v^{(\ell)} = \boldsymbol{W}_2 \left[\boldsymbol{h}_v^{(\ell-1)}, \bigoplus_{w \in \mathcal{N}(v)} \boldsymbol{W}_1 \left[\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{h}_w^{(\ell-1)} \right] \right],$$

where

$$\bigoplus = \underbrace{\begin{bmatrix} 1 \\ s(\deg(v), 1) \\ s(\deg(v), -1) \end{bmatrix}}_{\text{Scalers}} \otimes \underbrace{\begin{bmatrix} \text{mean} \\ \text{min} \\ \text{max} \end{bmatrix}}_{\text{Aggregators}},$$

with $\otimes$ being the tensor product and

$$s(d, \alpha) = \left(\frac{\log(d+1)}{\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \log(\deg(v)+1)} \right)^{\alpha}$$

denoting degree-scalers.

4. PyGAS Programming Interface

To highlight the ease-of-use of our framework, we showcase the necessary changes to convert a common GCN architecture (Kipf & Welling, 2017) implemented in PyTorch Geometric (Fey & Lenssen, 2019) (*cf.* Listing 1) to its corresponding scalable version (*cf.* Listing 2). In particular, our model now inherits from `ScalableGNN`, which takes care of creating all history embeddings (accessible via `self.histories`) and provides an efficient concurrent history access pattern via `push_and_pull()`. Notably, the `forward()` execution method of our model now takes in the additional `n_id` parameter, which holds the global node index for each node in the current mini-batch. This assignment vector is necessary to push and pull the intermediate mini-batch embeddings to and from the global history embeddings.

Table 1. **Inter-/intra-connectivity ratio for real-world datasets with different mini-batch sampling strategies.** Utilizing METIS heavily minimizes inter-connectivity between mini-batches, which reduces history accesses and tightens approximation errors in return.

Sampling Scheme	CORA	CITESEER	PUBMED	COAUTHOR-CS	COAUTHOR-PHYSICS	AMAZON-COMPUTER	AMAZON-PHOTO	WIKI-CS
Random	1.33	1.24	3.17	6.81	9.94	9.05	5.61	5.85
METIS	0.14	0.02	0.52	2.77	2.26	2.27	1.03	1.12
	CLUSTER	PATTERN	REDDIT	PPI	FLICKR	YELP	ogbn-arxiv	ogbn-products
Random	36.64	51.02	6.58	6.79	1.82	6.74	3.02	26.18
METIS	1.57	1.61	2.80	1.27	1.07	2.52	0.48	1.94

```python
from torch_geometric.nn import GCNConv

class GNN(Module):
    def __init__(self, in_channels, hidden_channels, out_channels, num_layers):
        super(GNN, self).__init__()

        self.convs = ModuleList()
        self.convs.append(GCNConv(in_channels, hidden_channels))
        for _ in range(num_layers - 2):
            self.convs.append(GCNConv(hidden_channels, hidden_channels))
        self.convs.append(GCNConv(hidden_channels, out_channels))

    def forward(self, x, adj_t):
        for conv in self.convs[:-1]:
            x = conv(x, adj_t).relu()
        return self.convs[-1](x, adj_t)
```

Listing 1. **Full-batch** GCN (Kipf & Welling, 2017) **model within** PYTORCH GEOMETRIC (Fey & Lenssen, 2019)**.**

```python
from torch_geometric.nn import GCNConv
from torch_geometric_autoscale import ScalableGNN

class GNN(ScalableGNN):
    def __init__(self, num_nodes, in_channels, hidden_channels, out_channels, num_layers):
        super(GNN, self).__init__(num_nodes, hidden_channels, num_layers)

        self.convs = ModuleList()
        self.convs.append(GCNConv(in_channels, hidden_channels))
        for _ in range(num_layers - 2):
            self.convs.append(GCNConv(hidden_channels, hidden_channels))
        self.convs.append(GCNConv(hidden_channels, out_channels))

    def forward(self, x, adj_t, n_id):
        for conv, history in zip(self.convs[:-1], self.histories):
            x = conv(x, adj_t).relu()
            x = self.push_and_pull(history, x, n_id)
        return self.convs[-1](x, adj_t)
```

Listing 2. **Mini-batch** GCN (Kipf & Welling, 2017) **model within** PYTORCH GEOMETRIC (Fey & Lenssen, 2019) **and our proposed *PyGAS* framework.** 🟧 denotes lines that require changes, while 🟩 refers to newly added lines. Only minimal changes are required to auto-scale GCN (or any other model) to large graphs.

5. Addtional Ablation Studies

We report additional ablation studies to further strengthen the motivation of our GAS framework:

Table 2. **Ablation study for a 4-layer** GIN (Xu et al., 2019) **model on the** CLUSTER **dataset** (Dwivedi et al., 2020)**.** Combining both GAS techniques help in resembling full-batch performance for expressive models with highly non-linear message passing phases.

			Accuracy		
			Training	Validation	Test
	Full-batch Baseline		60.49	58.17	58.49
	Minimizing Inter-Connectivity	**Enforcing Lipschitz Continuity**			
GAS	✗	✗	55.66	54.86	55.15
	✓	✗	58.97	57.79	57.82
	✓	✓	**60.67**	**58.21**	**58.51**

Table 3. **Dataset statistics.**

	Dataset	Task	Nodes	Edges	Features	Classes	Label Rate
Small-scale	CORA	multi-class	2,708	5,278	1,433	7	5.17%
	CITESEER	multi-class	3,327	4,552	3,703	6	3.61%
	PUBMED	multi-class	19,717	44,324	500	3	0.30%
	COAUTHOR-CS	multi-class	18,333	81,894	6,805	15	1.64%
	COAUTHOR-PHYSICS	multi-class	34,493	247,962	8,415	5	0.29%
	AMAZON-COMPUTER	multi-class	13,752	245,861	767	10	1.45%
	AMAZON-PHOTO	multi-class	7,650	119,081	745	8	2.09%
	WIKI-CS	multi-class	11,701	215,863	300	10	4.96%
Large-scale	CLUSTER	multi-class	1,406,436	25,810,340	6	6	83.35%
	REDDIT	multi-class	232,965	11,606,919	602	41	65.86%
	PPI	multi-label	56,944	793,632	50	121	78.86%
	FLICKR	multi-class	89,250	449,878	500	7	50.00%
	YELP	multi-label	716,847	6,977,409	300	100	75.00%
	`ogbn-arxiv`	multi-class	169,343	1,157,799	128	40	53.70%
	`ogbn-products`	multi-class	2,449,029	61,859,076	100	47	8.03%

Minimizing Inter-Connectivity Between Batches. We make use of graph clustering methods (Karypis & Kumar, 1998; Dhillon et al., 2007) in order to minimize the inter-connectivity between batches, which minimizes history accesses and therefore increases closeness and reduces staleness in return. To evaluate this impact in practice, Tabel 1 lists the inter-/intra-connectivity ratio of all real-world datasets used in our experiments, both for randomly sampled mini-batches as well as for utilizing METIS partitions as mini-batches. Notably, applying METIS beforehand reduces the overall inter-/intra-connectivity ratio by a factor of 4 on average, which results in only a fraction of history accesses. Furthermore, most real-world datasets come with inter-/intra-connectivity ratios between 0.1 and 2.5, leading to only marginal runtime overheads when leveraging historical information, as confirmed by our runtime analysis.

Analysis of Gains for Obtaining Expressive Node Representations. Next, we highlight the impacts of minimizing the inter-connectivity between mini-batches and enforcing Lipschitz continuity of the learned function in order to derive expressive node representations. Here, we benchmark a 4-layer GIN model (Xu et al., 2019) on the CLUSTER dataset (Dwivedi et al., 2020), *cf.* Table 2. Notably, both solutions achieve significant gains in training, validation and test performance, and together, they are able to closely resemble the performance of full-batch training. However, we found that Lipschitz continuity regularization only helps in non-linear message passing phases, while it does not provide any additional gains for linear operators such as GCN (Kipf & Welling, 2017).

6. Datasets

We give detailed statistics for all datasets used in our experiments, *cf.* Table 3, which include the following tasks:

1. classifying academic papers in citation networks (CORA, CITESEER, PUBMED) (Sen et al., 2008; Yang et al., 2016)

2. categorizing computer science articles in Wikipedia graphs (WIKI-CS) (Mernyei & Cangea, 2020)
3. predicting active research fields of authors in co-authorshop graphs (COAUTHOR-CS, COAUTHOR-PHYSICS) (Shchur et al., 2018)
4. predicting product categories in co-purchase graphs (AMAZON-COMPUTER, AMAZON-PHOTO) (Shchur et al., 2018)
5. identifying community clusters in Stochastic Block Models (CLUSTER, PATTERN) (Dwivedi et al., 2020)
6. predicting communities of online posts based on user comments (REDDIT) (Hamilton et al., 2017)
7. classifying protein functions based on the interactions of human tissue proteins (PPI) (Hamilton et al., 2017)
8. categorizing types of images based on their descriptions and properties (FLICKR) (Zeng et al., 2020)
9. classifying business types based on customers and friendship relations (YELP) (Zeng et al., 2020)
10. predicting subject areas of ARXIV Computer Science papers (`ogbn-arxiv`) (Hu et al., 2020)
11. predicting product categories in an AMAZON product co-purchasing network (`ogbn-products`) (Hu et al., 2020)

References

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *ICML*, 2020.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *NeurIPS*, 2020.

Dhillon, I. S., Guan, Y., and Kulis, B. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR-W*, 2019.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open Graph Benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359—-392, 1998.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized PageRank. In *ICLR*, 2019.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In *NeurIPS*, 2019.

Mernyei, P. and Cangea, C. Wiki-CS: A wikipedia-based benchmark for graph neural networks. In *ICML-W*, 2020.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

Sen, G., Namata, G., Bilgic, M., and Getoor, L. Collective classification in network data. *AI Magazine*, 29, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. In *NeurIPS-W*, 2018.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Yang, Z., Cohen, W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. In *NIPS*, 2017.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT: Graph sampling based inductive learning method. In *ICLR*, 2020.