Principal Neighbourhood Aggregation for Graph Nets

Gabriele Corso[*]
University of Cambridge
gc579@cam.ac.uk

Luca Cavalleri[*]
University of Cambridge
lc737@cam.ac.uk

Dominique Beaini
InVivo AI
dominique@invivoai.com

Pietro Liò
University of Cambridge
pietro.lio@cst.cam.ac.uk

Petar Veličković
DeepMind
petarv@google.com

Abstract

Graph Neural Networks (GNNs) have been shown to be effective models for different predictive tasks on graph-structured data. Recent work on their expressive power has focused on isomorphism tasks and countable feature spaces. We extend this theoretical framework to include continuous features—which occur regularly in real-world input domains and within the hidden layers of GNNs—and we demonstrate the requirement for multiple aggregation functions in this setting. Accordingly, we propose Principal Neighbourhood Aggregation (PNA), a novel architecture combining multiple aggregators with degree-scalers (which generalize the sum aggregator). Finally, we compare the capacity of different models to capture and exploit the graph structure via a benchmark containing multiple tasks taken from classical graph theory, which demonstrates the capacity of our model.

1 Introduction

Graph Neural Networks (GNNs) have been an active research field for the last ten years with great advancements in graph representation learning [1, 2, 3, 4]. However, it is difficult to understand the effectiveness of new GNNs due to the lack of standardized benchmarks [5] and of theoretical frameworks for their expressive power.

In fact, most work in this domain has focused on improving the GNN architectures on a set of graph benchmarks, without evaluating the capacity of their network to properly characterize the graphs' structural properties. Only recently there have been significant studies on the expressive power of various GNN models [6, 7, 8, 9, 10, 11]. However, these have mainly focused on the capacity of distinguishing different graph topologies, with little work done on understanding their capacity to capture and exploit the underlying features of the graph structure.

Alternatively, some work focuses on generalizing convolutional neural networks (CNN) to graphs using the spectral domain, as first proposed by Bruna *et al.* [12]. To improve the efficiency of the spectral analysis and improve the performance of the models, Chebyshev polynomials were developed [13] and later generalized into Cayley filters [14] or replaced by wavelet transforms [15]. In our work, we look at the capacity of different GNN models to understand certain aspects of the spectral decomposition, namely the graph Laplacian and the spectral radius, as they constitute fundamental aspects of the graphs' spectral properties. Although the spectral properties and filters are not explicitly encoded in our architecture, a powerful enough spatial GNN should still be able to learn them effectively.

Previous work on tasks taken from classical graph theory focuses on evaluating the performance of GNN models on a single task such as shortest paths [16, 17, 18], graph moments [19] or trav-

[*]Equal contribution.

Preprint. Under review.

elling salesman problem [5, 20]. Instead, we took a different approach by developing a multi-task benchmark containing problems both on the node level and the graph level. In particular, we look at the ability of each GNN to predict single-source shortest paths, eccentricity, laplacian features, connectivity, diameter and spectral radius. Many of these tasks are based on algorithms using dynamic programming and, therefore, are known to be well suited for GNNs [17]. We believe this multi-task approach ensures that the GNNs are able to understand multiple properties simultaneously, which is fundamental for solving complex graph problems. Moreover, efficiently sharing parameters between the tasks suggests a deeper understanding of the structural features of the graphs. Furthermore, we explore the generalization ability of the networks by testing on graphs of larger sizes than those present in the training set.

We hypothesize that the aggregation layers of current GNNs are unable to extract enough information from the nodes' neighbourhoods in a single layer, which limits their expressive power and learning abilities. In fact, recent works show how different aggregators perform better on different tasks [16, 6], that GNNs do not excel at learning nodes' clustering themselves as they don't properly characterize their neighbourhood [21], and that they are unable to reliably find substructures [22].

We first prove mathematically the need for multiple aggregators by proposing a solution for the uncountable multiset injectivity problem introduced by [6]. Then, we propose the concept of degree-scalers as a generalization to the *sum* aggregation, which allow the network to amplify or attenuate signals based on the degree of each node. Combining the above, we design the proposed *Principal Neighbourhood Aggregation* (**PNA**) network and demonstrate empirically that using multiple aggregation strategies concurrently improves the performance of the GNN on graph theory problems.

Dehmamy *et al.* [19] have also found empirically that using multiple aggregators (mean, sum and normalized mean), which extract similar statistics from the input message, improves the performance of GNNs on the task of graph moments. In contrast, our work extends the theoretical framework by deriving the necessity to use complementary aggregators. Accordingly, we propose using different statistical aggregations to allow each node to understand the distribution of the messages it receives, and we generalize the *mean* aggregation as the first moment of a set of possible *n-moment* aggregations.

We present a consistently well-performing and parameter efficient encode-process-decode architecture [23] for GNNs. This differs from traditional GNNs by allowing a variable number of convolutions, vanquishing the limitation of GNNs to distributed local algorithms [24] described by [11].

Using this model, we compare the multi-task performances of some of the most diffused models in the literature (GCN [25], GAT [26], GIN [6] and MPNN [27]) with our PNA and a clear hierarchy arises. In particular, we observe that the proposed PNA, formed by the combination of various aggregators and scalers, significantly outperforms these baselines. The fact that this outperformance was consistent along all tasks with all the architectures experimented further supports our hypothesis.

2 Principal Neighbourhood Aggregation

In this section, we first explain the motivation behind using multiple aggregators concurrently. Then, we present the idea of degree-based scalers, linking to the prior related work of GNN expressiveness. Finally, we detail the design of graph convolutional layers which leverage the proposed Principal Neighbourhood Aggregation.

2.1 Proposed aggregators

One of the main concerns of this manuscript is the ability to understand a one-hop node neighbourhood using a single GNN layer. Doing so will reduce the effects of over-smoothing [28, 29, 30] and allow the depth of the network to focus on understanding the interactions of far-away nodes and support more complex latent state.

Most work in the literature uses only a single aggregation method, with *mean, sum* and *max* aggregators being the most used in the state-of-the-art models [6, 25, 27, 16]. In Figure 1, we observe how different neighbourhood aggregators fail to discriminate between different messages when using a single GNN layer.

We formalize our observations in the theorem given below.

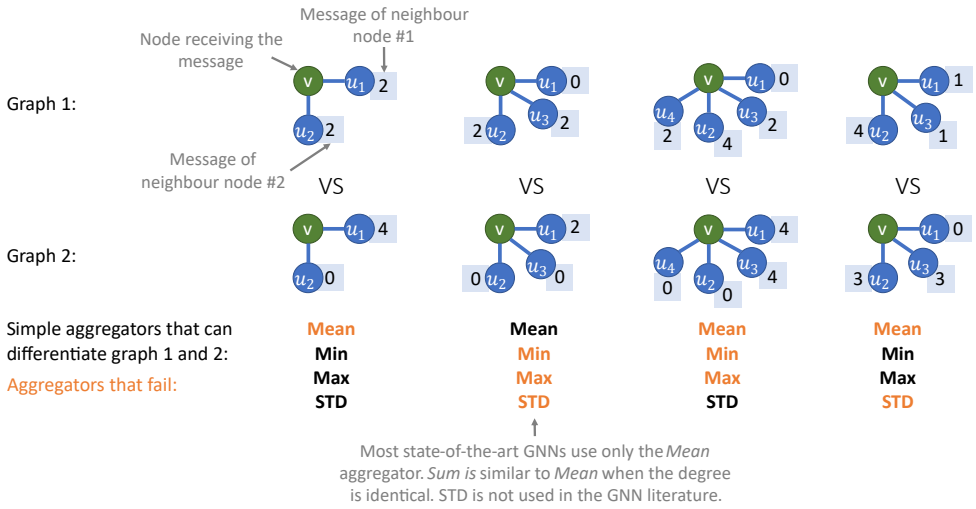


Figure 1: Examples where, for a single GNN layer and continuous input feature spaces, some aggregators fail to differentiate between neighbourhood messages. For these examples and using a single layer, we also observe that the aggregators are *complementary*: there is always at least one aggregator that can discriminate between the different neighbour messages.

Theorem 1 (Number of aggregators needed)**.** *In order to discriminate between multisets of size n whose underlying set is $\mathbb{R}$, at least n aggregators are needed.*

Proposition 1 (Moments of the multiset)**.** *The moments of a multiset (as defined in equation 4) constitute a valid example using n aggregators.*

We prove Theorem 1 in Appendix A and Proposition 1 in Appendix B. Note that unlike Xu *et al.* [6], we consider a continuous input features space; this better represents many real-world tasks where the observed values have uncertainty, and better models the latent node features within a neural network's representations. Using continuous features makes the multiset uncountable, and voids the injectivity proof of the *sum* aggregation presented by Xu *et al.* [6].

To make further progress, we redefine the concepts of aggregators and scalers. **Aggregators** are a continuous function of a multiset which compute a statistic on the neighbouring nodes, such as *mean*, *max* or *standard deviation*. The continuity is important with continuous input spaces, as small variations in the input should result in small variations of the aggregators' output. **Scalers** are applied on the aggregated value and perform either an *amplification* or an *attenuation* of the incoming messages, which is dependent on the number of messages being aggregated (usually the node degree). In this framework, we may re-express the *sum* aggregator as a *mean* aggregator followed by a linear-degree scaling (see Section 2.2).

Theorem 1 proves that the number of independent aggregators used is a limiting factor in the expressiveness of GNNs. To empirically demonstrate this, here we leverage four aggregators, namely *mean, maximum, minimum* and *standard deviation*. Furthermore, we note that this concept can be generalized to the *normalized moment* aggregator, which allows for variable numbers of aggregators and extracting advanced distribution information whenever the degree of the graph is high.

The following subsections will detail the aggregators we leveraged in our architectures. We also provide descriptions of a few additional aggregation functions of interest in Appendix D.

Mean aggregation $\mu(X^l)$ The most common message aggregator in the literature, wherein each node receives a weighted average or weighted sum of its incoming messages. Equation 1 presents, on the left, the general mean equation, and, on the right, the direct neighbour formulation, where X is any multiset, X^l are the nodes' features at layer l, $N(i)$ is the neighbourhood of node i

and $d_i = |N(i)|$. For clarity we use $\mathbb{E}[f(X)]$ where X is a multiset of size d to be defined as $\mathbb{E}[f(X)] = \frac{1}{d}\sum_{x \in X} f(x)$.

$$\mu(X) = \mathbb{E}[X] \quad , \quad \mu_i(X^l) = \frac{1}{d_i}\sum_{j \in N(i)} X_j^l \tag{1}$$

Maximum and minimum aggregations $\max(X^l),\ \min(X^l)$ Also often used in literature, they are very useful for discrete tasks, when extrapolating such tasks to unseen distributions of graphs [16] and for domains where credit assignment is important. Alternatively, we present the softmax and softmin aggregators in Appendix D, which are differentiable and work for weighted graphs, but don't perform as well on our benchmarks.

$$\max_i(X^l) = \max_{j \in N(i)} X_j^l \quad , \quad \min_i(X^l) = \min_{j \in N(i)} X_j^l \tag{2}$$

Standard deviation aggregation $\sigma(X^l)$ The standard deviation (STD or σ) is used to quantify the spread of neighbouring nodes features, such that a node can assess the diversity of the signals it receives. Equation 3 presents, on the left, the standard deviation formulation and, on the right, the STD of a graph-neighbourhood. *ReLU* is the rectified linear unit used to avoid negative values caused by numerical errors and ϵ is a small positive number to ensure σ is differentiable.

$$\sigma(X) = \sqrt{\mathbb{E}[X^2] - \mathbb{E}[X]^2}, \quad \sigma_i(X^l) = \sqrt{ReLU\left(\frac{1}{d_i}\sum_{j \in N(i)} {X_j^l}^2 - \left(\frac{1}{d_i}\sum_{j \in N(i)} X_j^l\right)^2\right) + \epsilon} \tag{3}$$

Normalized moments aggregation $M_n(X^l)$ The mean and variance being the first and second central moments of a signal ($n = 1, n = 2$), additional moments could be useful do better describe the signal, such as the skewness ($n = 3$) and the kurtosis ($n = 4$). This becomes more important when the degree of a node is high, because the four previous aggregators are then insufficient to describe the neighbourhood accurately. The central moments normalized by the standard deviation are presented in equation 4, for which we develop a corresponding node aggregator in Equation 5, where M_n is the desired moment of degree n. A *ReLU* can once again be used for all even values of n to avoid negative moments caused by numerical errors.

$$M_n(X) = \frac{\mathbb{E}\left[(X - \mu)^n\right]}{\sigma^n} \tag{4}$$

$$M_{n,i}(X^l) = \sigma_i^{-n}(X^l)\sum_{k=0}^{n}\left[\binom{n}{k}(-1)^{n-k}\left(\frac{1}{d_i}\sum_{j \in N(i)} {X_j^l}^k\right)\left(\frac{1}{d_i}\sum_{j \in N(i)} X_j^l\right)^{n-k}\right], \quad n \geq 3 \tag{5}$$

2.2 Degree-based scalers

Xu *et al.* [6] shows that the use of mean and max aggregators by themselves fails to distinguish between neighbourhoods with identical node features' distribution but differing cardinalities and the same applies to the other aggregators described above. They propose the *sum* aggregator to discriminate between such multisets. Redefining the *sum* aggregator as the composition of a *mean* aggregator and a scaling linear to the degree of each node $S_{\text{amp}}(d) = d$, allows us to generalise this property:

Theorem 2 (Injective functions on countable multisets)**.** *The mean aggregation composed with any scaling linear to an injective function on the neighbourhood size can generate injective functions on bounded multisets of countable elements.*

We formalize and prove Theorem 2 in Appendix C. Thus, the results proven in [6] about the *sum* aggregator become a particular case of this theorem, and we can use any kind of injective scaler to discriminate between multisets of various sizes.

Recent work shows that summation aggregation doesn't generalize well to unseen graphs [16], especially when they are of larger size. One reason is that a small change of the degree will cause the message and gradients to be amplified/attenuated exponentially (a linear amplification at each layer will cause an exponential amplification after multiple layers). Although there are different strategies to deal with this problem, we propose using a logarithmic amplification $S \propto \log(d+1)$ to reduce this effect. Note that the logarithm is injective for positive values, and d is defined non-negative.

Another motivation for using logarithmic scalers is to better describe the neighbourhood influence of a given node. Let's suppose we have a social network where nodes A, B and C have respectively 5 million, 1 million and 100 followers. On a linear scale, nodes B and C appear more similar than nodes A and B, however, this does not accurately model their relative influence. Hence, the logarithmic scale discriminates better between messages received by *influencer* and *follower* nodes.

We propose the logarithmic scaler S_{amp} presented in Equation 6, where δ is the average amplification in the training set, and d is the degree of the node receiving the message.

$$S_{\text{amp}}(d) = \frac{\log(d+1)}{\delta} \quad , \quad \delta = \frac{1}{|\text{train}|} \sum_{i \,\in\, \text{train}} \log(d_i + 1) \tag{6}$$

We may further generalize this scaler in Equation 7, where α is a variable parameter that is negative for attenuation, positive for amplification or zero for no scaling. Other definitions of $S(d)$ can be used—such as a linear scaling—as long as the function is injective for $d > 0$.

$$S(d, \alpha) = \left(\frac{\log(d+1)}{\delta} \right)^{\alpha}, \quad d > 0, \quad -1 \leq \alpha \leq 1 \tag{7}$$

2.3 Combined aggregation

Combining the aggregators and scalers presented in previous sections, we now propose the Principal Neighbourhood Aggregation (PNA). The PNA performs a total of twelve operations: four neighbour-aggregations with three scalers each, summarized in Equation 8. The aggregators are defined in Equations 1–3, while the scalers are defined in Equation 7, with $\otimes$ being the tensor product.

$$\bigoplus = \underbrace{\begin{bmatrix} I \\ S(D, \alpha = 1) \\ S(D, \alpha = -1) \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \mu \\ \sigma \\ \max \\ \min \end{bmatrix}}_{\text{aggregators}} \tag{8}$$

As mentioned earlier, higher degree graphs such as social networks could benefit from further aggregators (e.g. using the moments proposed in Equation 5). We insert the PNA operator within the framework of a message passing neural network [27], obtaining the following GNN layer:

$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i) \in E} M\left(X_i^{(t)}, X_j^{(t)} \right) \right) \tag{9}$$

where M and U are neural networks (for our benchmarks, a linear layer was enough). Further, U reduces the size of the concatenated message (in space $\mathbb{R}^{13F}$) back to $\mathbb{R}^F$ where F is the dimension of the hidden features in the network. As in the MPNN paper [27], we employ multiple towers to improve computational complexity and generalization performance.

Using twelve operations per kernel will require the usage of additional weights per input feature in the U function, which could seem to be just quantitatively—not qualitatively—more powerful than an ordinary MPNN with a single aggregator [27]. However, the overall increase in parameters in the

GNN model is modest and, as per our analysis, it is likely that usage of a single aggregation method is a potential limiting factor in GNNs.

This is comparable to convolutional neural networks (CNN) where a simple 3×3 convolutional kernel requires 9 weights per feature (1 weight per neighbour). Using a CNN with a single weight per 3×3 kernel will clearly reduce the computational capacity since the feedforward network won't be able to compute derivatives or the Laplacian operator. Hence, it is intuitive that the GNNs should also require multiple weights per node, as previously demonstrated in Theorem 1. We will demonstrate this observation empirically, by running experiments on baseline models with larger dimensions of the hidden features (and, therefore, more parameters).

3 Architecture

We compare various GNN layers, including PNA, on common architectures formed by $\mathcal{M}$ such layers, followed by three fully-connected layers for node labels and a set2set (S2S) [31] readout function for graph labels. For the following experiments[2] we used an architecture with a GRU after the aggregation function, and a variable number $\mathcal{M}-1$ of repeated middle convolutional layer. In particular, we want to highlight:

Gated Recurrent Units (GRU) [32] applied after the update function of each layer, as in [27, 33]. Their ability to retain information from previous layers proved effective when increasing $\mathcal{M}$.

Weight sharing in all the GNN layers from the second to $(\mathcal{M}-1)$-th (i.e. all but the first), makes the architecture follow an encode-process-decode configuration [3, 23]. This is a strong prior which works well on all our experimental tasks, with a parameter-efficient architecture that allows the model to have a variable number of layers, $\mathcal{M}$.

Variable depth, $\mathcal{M}$, decided at inference time (based on the size of the input graph and/or other heuristics). This is important when using models on distributions of graphs with a variety of different sizes. In our experiments, we have only used heuristics dependant on the number of nodes N ($\mathcal{M} = f(N)$). For the final architecture, we settled with $\mathcal{M} = \lfloor N/2 \rfloor$, where $\lfloor\ \rfloor$ is the floor operation and N the number of nodes in the graph. It would be interesting to test heuristics based on properties of the graph such as the diameter or an adaptive computation time heuristic [34] based on, for example, the convergence of the nodes' features [16]. We leave these analyses to future work.

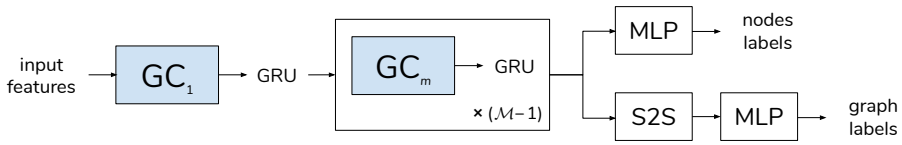


Figure 2: Layout of the architecture used, with a S2S layer for the graph labels and a multi-layer perceptron (MLP) at each output. When comparing different graph convolutions on the same graphs, the difference between the various models only lies in the type of graph convolution layer to use in place of GC_1 and GC_m.

This architecture layout (represented in Figure 2) was determined based on the combination of its downstream performance and parameter efficiency. We note that all attempted architectures have yielded similar comparative performance of GNN layers.

Skip connections after each GNN layer all feeding into the fully connected layers were also tried. They are known to improve learning in deep architectures [35], especially for GNNs where they reduce over-smoothing [8]. However, in presence of GRUs, they did not give significant performance improvements on our benchmarks.

3.1 Alternative Graph Convolutions

In this subsection, we present the details of the four graph convolution layers from existing models that we used to compare the performance of the PNA.

[2]The code for all the aggregators, scalers, models, architectures and dataset generation is available at `https://github.com/lukecavabarrett/pna` .

Graph Convolutional Networks (GCN) [25] use a form of normalized mean aggregator followed by a linear transformation and an activation function, as defined in Equation 10. Here, $\tilde{A} = A + I_N$ is the adjacency matrix with self-connections, W is a trainable weight matrix and b a learnable bias.

$$X^{(t+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X^{(t)}W + b\right) \tag{10}$$

Graph Attention Networks (GAT) [26] perform a linear transformation of the input features followed by an aggregation of the neighbourhood as a weighted sum of the transformed features, where the weights are set by an attention mechanism a, defined in Equation 11. Here, W is a trainable projection matrix. As in the original paper, we employ the use of multi-head attention.

$$X_i^{(t+1)} = \sigma\left(\sum_{(j,i)\in E} a\left(X_i^{(t)}, X_j^{(t)}\right) W X_j^{(t)}\right) \tag{11}$$

Graph Isomorphism Networks (GIN) [6] perform a sum aggregator between all the neighbours, followed by an update function U consisting in a multi-layer perceptron, as defined in Equation 12. Here, ϵ is a learnable parameter. As in the original paper, we use a 2-layer MLP for U.

$$X_i^{(t+1)} = U\left(\left(1+\epsilon\right)X_i^{(t)} + \sum_{j\in N(i)} X_j^{(t)}\right) \tag{12}$$

Message Passing Neural Networks (MPNN) [27] perform a transformation before and after an arbitrary aggregator defined in Equation 13, where M and U are neural networks and $\bigoplus$ is a single aggregator. In particular, we test models with *sum* and *max* aggregators, as they are the most used in literature. As with PNA layers, we found that linear transformations are sufficient for M and U and, as in the original paper [27], we employ multiple towers.

$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(X_i^{(t)}, X_j^{(t)}\right)\right) \tag{13}$$

4 Benchmarks

4.1 Random graph generation

Following previous work [16, 36], the benchmark contains undirected unweighted graphs of a wide variety of types (we provide, in parentheses, the approximate proportion of the graphs in the overall benchmark). Letting N be the total number of nodes:

- **Erdős-Rényi** [37] (20%): random probability of edge creation for each node
- **Barabási-Albert** [38] (20%): the number of edges for a new node is taken randomly from $\{1, 2, ..., N-1\}$
- **Grid** (5%): $m \times k$ 2d grid graph with $N = mk$ and m and k as close as possible
- **Caveman** [39] (5%): with m cliques of size k, with m and k as close as possible
- **Tree** (15%): generated with a power-law degree distribution with exponent 3
- **Ladder graphs** (5%)
- **Line graphs** (5%)
- **Star graphs** (5%)
- **Caterpillar graphs** (10%): with a backbone of size b (drawn from $\mathcal{U}[1, N)$), and $N - b$ pendent vertices uniformly connected to the backbone
- **Lobster graphs** (10%): with a backbone of size b (drawn from $\mathcal{U}[1, N)$), and p (drawn from $\mathcal{U}[1, N-b]$) pendent vertices uniformly connected to the backbone, and additional $N - b - p$ pendent vertices uniformly connected to the previous pendent vertices.

Additional randomness was introduced to the generated graphs by randomly toggling arcs, without strongly impacting the average degree and main structure. If e is the number of edges and m the number of 'missing edges' ($2e + 2m = N(N-1)$), the probabilities P_e and P_m of an existing and missing edge to be toggled are:

$$P_e = \begin{cases} 0.9 & e \leq m \\ 0.9 + 0.1\ \frac{e-m}{m} & e > m \end{cases} \qquad P_m = \begin{cases} 0.1\ \frac{e}{m} & e \leq m \\ 0.1 & e > m \end{cases} \tag{14}$$

After performing the random toggling, we discarded graphs containing singleton nodes, as these are in no way affected by the choice of aggregation. For the presented results we used graphs of small sizes (15 to 50 nodes) as they were already sufficient to demonstrate clear differences between the models.

4.2 Multi-task graph properties

The graph property tasks consist of a range of individual properties for each node and global properties of the entire graph. In the multi-task benchmark, we consider three node labels and three graph labels.

Node tasks

1. Single-source shortest-path lengths: length of the shortest path from a node to all the others, where the source node is specified via a one-hot vector. The labels of nodes outside the connected component of the source are set to 0. Note that, since the graph is unweighted, this task corresponds to performing a breadth-first search (BFS).
2. Eccentricity: for every node v, the longest shortest path from v to any other node within its connected component.
3. Laplacian features: LX where $L = (D - A)$ is the Laplacian matrix of the graph and X are the input node feature vectors.

Graph tasks

4. Connected: whether the graph is connected.
5. Diameter: the longest shortest path between any two nodes that share components.
6. Spectral radius: the largest absolute value of the eigenvalues of the adjacency matrix (always real since A is real and symmetric).

Input features As input features, the network is provided with two vectors of size N. The first represents a one-hot vector representing which node is the starting point for the single-source shortest-path tasks. The second is the feature vector X where each element is i.i.d. sampled as $X_i \sim \mathcal{U}[0, 1]$.

Apart from taking part in the Laplacian features task, this random feature vector also provides a "unique identifier" for the nodes in other tasks. This allows for addressing some of the problems highlighted in [7, 22]; e.g. the task of whether a graph is connected could be performed by continually aggregating the maximum feature of the neighbourhood and then checking whether they are all equal in the readout. Similar strengthening via random features was also concurrently discovered by [40].

4.3 Model training

While having clear differences, these tasks also share related subroutines, e.g. tasks (1, 2, 4, 5) can all be expressed via graph traversals, and the diameter is the maximum of all node eccentricities. While we do not give this sharing of subroutines as prior to the models as in [16], we expect models that have the capacity of understanding and exploiting the graph structure to pick up on these commonalities, efficiently share parameters and reinforce each other during the training.

Tasks are normalised by dividing each label by the maximum value of their label (among all nodes in node labels) in the training set; since all labels are non-negative, this results in all tasks having normalised labels between 0 and 1. This normalisation allows for a better equilibrium between the various tasks during the training and validation. The model's predictive power on the benchmark is calculated as the average of the mean squared errors (MSE) on the (normalised) tasks.

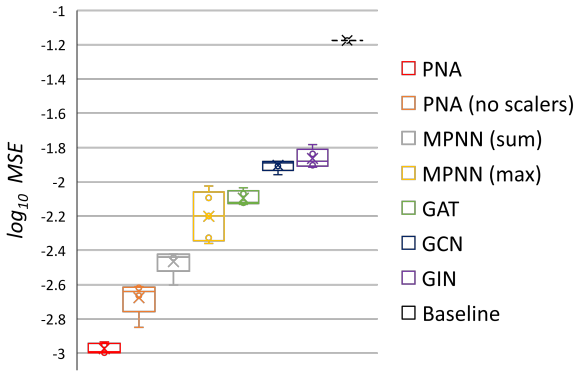
We trained the models using the Adam optimizer for a maximum of 10,000 epochs, using early stopping with a patience of 1,000 epochs. Learning rates, weight decay, dropout and other hyper-parameters such as the number of towers/attention heads were tuned on the validation set for each model. For each model, we run 10 training runs with different seeds and different hyper-parameters (but close to the tuned values) and report the five with least validation error.

5 Results and discussion

The multi-task results are presented in Figure 3a, where we observe that the proposed PNA model consistently outperforms state-of-the-art models, and in Figure 3b, where we note that the PNA performs consistently better on all tasks. The *baseline* represents the MSE from predicting the average of the training set for all tasks (or the variance of each task).

The trend of these multi-task results follows but amplifies differences in the average performances of the models when trained separately on the individual tasks, which suggests that the PNA model can better capture and exploit the common sub-units of these tasks. Further, PNA showed to perform the best on all architecture layouts that we attempted. We should note that the GIN architecture was the only one whose performance suffered when switching from an architecture without weight-sharing to the encode-process-decode architecture; in all other cases, the GIN model had performances in-between the MPNN and GAT models.

We note in Figure 3b that GCN, GAT and GIN are unable to estimate the Laplacian transformation of the features, and perform very close to the baseline for other node tasks. This shows a strong limitation of the models' capacity and can be attributed to the over-smoothing effect. This limitation is slightly reduced using skip-connections, but their general performance remains close to the baseline.



(a)

		Nodes tasks			Graph tasks		
Model	**Average score**	**1**	**2**	**3**	**4**	**5**	**6**
PNA	-3.13	-2.89	-2.89	-3.77	-2.61	-3.04	-3.57
PNA (no scalers)	-2.77	-2.54	-2.42	-2.94	-2.61	-2.82	-3.29
MPNN (sum)	-2.53	-2.36	-2.16	-2.59	-2.54	-2.67	-2.87
MPNN (max)	-2.50	-2.33	-2.26	-2.37	-1.82	-2.69	-3.52
GAT	-2.26	-2.34	-2.09	-1.60	-2.44	-2.40	-2.70
GCN	-2.04	-2.16	-1.89	-1.60	-1.69	-2.14	-2.79
GIN	-1.99	-2.00	-1.90	-1.60	-1.61	-2.17	-2.66
Baseline	-1.38	-1.87	-1.50	-1.60	-0.62	-1.30	-1.41

1. Single-source shortest-paths
2. Eccentricity
3. Laplacian features
4. Connected
5. Diameter
6. Spectral radius

(b)

Figure 3: Multi-task benchmarks for different GNN models using the same architecture and various near-optimal hyper-parameters. (a) Distribution of the $\log_{10}$MSE errors for the top 5 performances of each model. (b) Mean $\log_{10}$MSE error for each task and their combined average.

In order to demonstrate that the performance improvements of the PNA model are not due to the (relatively small) number of additional parameters it has compared to the other models (about 15%), we ran tests on all the other models with latent size increased from 16 to 20 features. The results of these models, presented in Table 1, suggest that even when baseline models are given 30% more parameters than the PNA, they are qualitatively less capable of capturing the graphs' structure.

Finally, we explored the extrapolation of the models to larger graphs, in particular, we trained models on graphs of sizes between 15 and 25, validated them with graphs between 25 and 30 and evaluated their performances on graphs between 20 and 50. This task presents many challenges, two of the most significant are: firstly, unlike in [16] the models are not given the step-wise supervision or trained on subroutines that can be extended. Secondly, the models have to cope with their architectures being extended to further hidden layers than trained on, which can sometimes cause problems with rapidly increasing feature scales.

Table 1: Average score of different models using feature sizes of 16 and 20, compared to the PNA with 16. "# params" is the total number of parameters in each architecture. We observe that, even with fewer parameters, PNA performs consistently better and the performance of the other models is not boosted by an increased number of parameters.

	Size 16		Size 20	
Model	# params	Avg score	# params	Avg score
PNA	8350	-3.13	-	-
MPNN (sum)	7294	-2.53	11186	-2.19
MPNN (max)	8032	-2.50	12356	-2.23
GAT	6694	-2.26	10286	-2.08
GCN	6662	-2.04	10246	-1.96
GIN	7272	-1.99	11168	-1.91

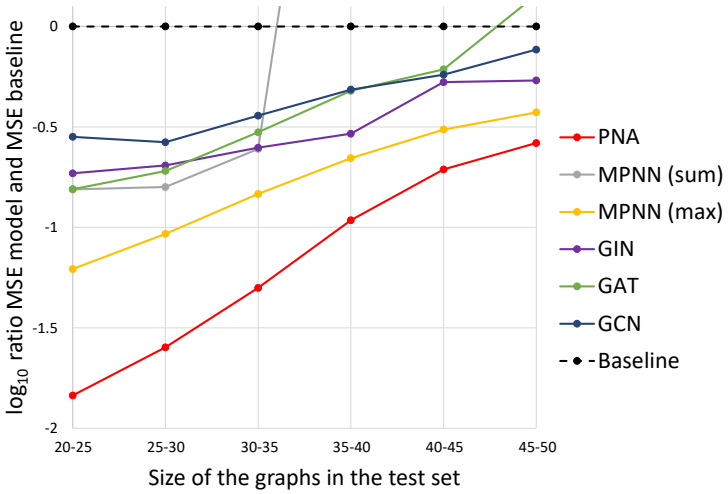


Figure 4: Multi-task $\log_{10}$ of the ratio of the MSE for different GNN models and the variance of the tasks (MSE of the baseline). The training is done with the same architectures on graphs of 15-25 nodes and the validation on 25-30 nodes.

Due to the aforementioned challenges, as expected, the performance of the models (as a proportion of the baseline performance) gradually worsens, with some of the models having feature explosions. However, the PNA model had once again consistently outperformed all the other models on all graph sizes. Our results also follow the findings in [16], i.e. that between single aggregators the *max* tends to perform best when extrapolating to larger graphs. For the PNA, we believe that it converges to a better aggregator combining the advantages of each operation.

6 Conclusion

We have extended the theoretical framework in which GNNs are analysed to continuous features and proven the need for multiple aggregators in such circumstances. We also generalize the sum aggregation by presenting degree-scalers and propose the use of a logarithmic scaling. Taking all of the above into consideration, we have presented a method, Principal Neighbourhood Aggregation, composed of multiple aggregators and degree-scalers. With the goal of understanding the capacity of GNNs to capture graph structures, we have proposed a novel multi-task benchmark and an encode-process-decode architecture for solving it. We believe that our findings constitute a step towards establishing a hierarchy of models w.r.t. their expressive power and, in this sense, the PNA model appears to outperform the prior art in GNN layer design.

References

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[2] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, Jul 2017.

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[5] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[7] Vikas K Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. *arXiv preprint arXiv:2002.06157*, 2020.

[8] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10943–10953, 2019.

[9] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[10] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. *arXiv preprint arXiv:1903.02541*, 2019.

[11] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

[12] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering.

[14] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. CayleyNets: Graph convolutional neural networks with complex rational spectral filters.

[15] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph wavelet neural network.

[16] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019.

[17] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

[18] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[19] Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems*, pages 15387–15397, 2019.

[20] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[21] Emmanuel Noutahi, Dominique Beaini, Julien Horwood, Sébastien Giguère, and Prudencio Tossou. Towards interpretable sparse graph representation learning with laplacian pooling.

[22] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*, 2020.

[23] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

[24] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, page 82–93, New York, NY, USA, 1980. Association for Computing Machinery.

[25] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[27] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[28] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[29] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*, 2019.

[30] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019.

[31] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[33] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[34] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

[37] P. Erdős and A Rényi. On the evolution of random graphs. pages 17–61, 1960.

[38] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, Jan 2002.

[39] Duncan J. Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology*, 105(2):493–527, 1999.

[40] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020.

[41] Joseph J. Rotman. *An Introduction to Algebraic Topology*, volume 119 of *Graduate Texts in Mathematics*. Springer New York.

[42] Karol Borsuk. Drei sätze über die n-dimensionale euklidische sphäre. *Fundamenta Mathematicae*, (20):177–190, 1933.

[43] Richard P. Stanley. *Enumerative Combinatorics Volume 2*. Cambridge Studies in Advanced Mathematics no. 62. Cambridge University Press, Cambridge, 2001.

[44] M Hazewinkel. *Encyclopaedia of mathematics. Volume 9, STO-ZYG.* Encyclopaedia of mathematics ; vol 9: STO-ZYG. Kluwer Academic, Dordecht, 1988.

A Proof for Theorem 1 (Number of aggregators needed)

Proof. Let S be the n-dimensional subspace S of $\mathbb{R}^n$ formed by all tuples $(x_1,\ x_2, \ldots, x_n)$ such that $x_1 \leq x_2 \leq \ldots \leq x_n$, and notice how S is the collection of the aforementioned multisets. We defined an aggregator as a continuous function from multisets to reals, which corresponds to a continuous function $g : S \to \mathbb{R}$.

Assume by contradiction that it is possible to discriminate between all the multisets of size n using only $n-1$ aggregators, viz. $g_1, g_2, \ldots, g_{n-1}$.

Define $f : S \to \mathbb{R}^{n-1}$ to be the function mapping each multiset X to its output vector $(g_1(X),\ g_2(X),\ \ldots\ , g_{n-1}(X))$. Since $g_1, g_2, \ldots, g_{n-1}$ are continuous, so is f, and, since we assumed these aggregators are able to discriminate between all the multisets, f is injective.

As S is a n-dimensional Euclidean subspace, it is possible to define a $(n-1)$-sphere C^{n-1} entirely contained within it, i.e. $C^{n-1} \subseteq S$. According to Borsuk–Ulam theorem [41, 42], there are two distinct (in particular, non-zero and antipodal) points $\vec{x}_1, \vec{x}_2 \in C^{n-1}$ satisfying $f(\vec{x}_1) = f(\vec{x}_2)$, showing f not to be injective; hence the required contradiction. □

Note: n aggregators are actually sufficient. A simple example is to use $g_1, g_2, \ldots, g_n$ where $g_k(X) =$ the k-th smallest item in X. It's clear to see that the multiset whose elements are $g_1(X),\ g_2(X),\ \ldots\ , g_n(X)$ is X, which can hence be uniquely determined by the aggregators.

B Proof for Proposition 1 (Moments of the multiset)

Proof. For $n < 3$, we can trivially uniquely determine the original multiset, so assume $n \geq 3$, and hence knowledge of μ, σ^2. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the multiset to be found, and define $R = \{r_1 = x_1 - \mu,\ r_2 = x_2 - \mu,\ \ldots,\ r_n = x_n - \mu\}$.

Notice how $\sum {r_i}^1 = 0$, and $\sum {r_i}^2 = n\sigma^2$, and for $2 < k \leq n$ we have $\sum {r_i}^k = n\sigma^k M_k(X)$, i.e. all the symmetric power sums $p_k = \sum {r_i}^k$ ($k \leq n$) are uniquely determined by the moments.

Additionally, e_k, the elementary symmetric sums of R, i.e. the sum of the products of all the sub-multisets of size k ($1 \leq k \leq n$), are determined as follow:

e_1, the sum of all elements, is equal to p_1; e_2, the sum of the products of all pairs in R, is $(e_1 p_1 - p_2)\,/2$; e_3,the sum of the products of all triplets, is $(e_2 p_1 - e_1 p_2 + p_3)\,/3$, and so on. Notice how $e_1, e_2, \ldots, e_n$ can be computed using the following recursive formula [43]:

$$\sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} \left(\prod_{j=1}^{k} r_{i_j} \right) = e_k = \frac{1}{k} \sum_{j=1}^{k} (-1)^{j-1} e_{k-j} p_j \quad , \quad e_0 = 1$$

Consider polynomial $P(x) = \Pi(x - r_i)$, i.e. the unique polynomial of degree n with leading coefficient 1 whose roots are R. This defines A, the coefficients of P, i.e. the real numbers $a_0, a_1, \ldots, a_{n-1}$ for which $P(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. Using Vieta's formulas [44]:

$$\sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} \left(\prod_{j=1}^{k} r_{i_j} \right) = (-1)^k \frac{a_{n-k}}{a_n}$$

which applied to P yield that a_{n-k} is equal to a_n (equal to 1 in P) divided by $(-1)^k$ multiplied by e_k. Hence A is uniquely determined, and so is P, being its coefficients a valid definition of it. By the fundamental theorem of algebra, P has n (possibly repeated) roots, which are the elements of R, hence uniquely determining the latter.

Finally, X can be easily determined adding μ to each element of R. □

Note: the proof above assume the knowledge of n. In the case that n is variable (as in GNNs) and so we have multisets of up to n elements an extra aggregator will be needed. An example of such aggregator is the mean multiplied by any injective scaler which would allow the degree of the node to be inferred.

C Proof for Theorem 2 (Injective functions on countable multisets)

Proof. Let χ be the countable input feature space from which the elements of the multisets are taken. Since χ is countable and the cardinality of multisets is bounded, let $Z : \chi \to \mathbb{N}^+$ be an injection from χ to natural numbers, and $N \in \mathbb{N}$ such that $|X| + 1 < N$ for all X.

Let's define an injective function s, and without loss of generality, assume $s(0), s(1), \ldots, s(N) > 0$ (otherwise for the rest of the proof consider s as $s'(i) = s(i) - \min_{j\in[0,N]} s(j) + \epsilon$ which is positive for all $i \in [0, N]$). $s(|X|)$ can only take value in $\{s(0), s(1), \ldots, s(N)\}$, therefore let us define $\gamma = \min\left\{\frac{s(i)}{s(j)} \mid i, j \in [0, N],\ s(i) \geq s(j)\right\}$. Since s is injective, $s(i) \neq s(j)$ for $i \neq j$, which implies $\gamma > 1$.

Let $K > \frac{1}{\gamma-1}$ be a positive real number and consider $f(x) = N^{-Z(x)} + K$.

$\forall x \in \chi, Z(x) \in [1, N] \Rightarrow N^{-Z(x)} \in [0, 1] \Rightarrow\ f(x) \in [K, K+1]$, so $\mathbb{E}_{x\in X}[f(x)] \in [K, K+1]$.

We proceed to show that the cardinality of X can be uniquely determined, and X itself can be determined as well, by showing that exist an injection h over the multisets.

Let us h as a function that scales the mean of f by an injective function of the cardinality:

$$h(X) = s(|X|)\, \mathbb{E}_{x\in X}[f(x)]$$

We want show that the value of $|X|$ can be uniquely inferred from the value of $h(X)$. Assume by contradiction $\exists\, X', X''$ multisets of size at most N such that $|X'| \neq |X''|$ but $h(X') = h(X'')$; since s is injective $s(|X'|) \neq s(|X''|)$, without loss of generality let $s(|X'|) > s(|X''|)$, then:

$$s(|X''|)(K+1) \geq s(|X''|)\, \mathbb{E}_{x\in X''}[f(x)] = h(X'') = h(X') = s(|X'|)\, \mathbb{E}_{x\in X'}[f(x)] \geq s(|X'|)\, K$$

$$\implies\ K \leq \frac{1}{\frac{s(|X'|)}{s(|X''|)} - 1} \leq \frac{1}{\gamma - 1}$$

which is a contradiction. So it is impossible for the size of a multiset X to be ambiguous from the value of $h(X)$.

Let us define d as the function mapping $h(X)$ to $|X|$.

$$h'(X) = \sum_{x\in X} N^{-Z(x)} = \frac{h(X)|X|}{s(|X|)} - K|X| = \frac{h(X)d(h(X))}{s(d(h(X)))} - Kd(h(X))$$

Considering the $Z(j)$-th digit i after the decimal point in the base N representation of $h'(X)$, it can be inferred that X contains i elements j, and, so, all the elements in X can be determined; hence h is injective over the multisets in X. □

Note: this proof is a generalization of the one by *Xu et al.* [6] on the *sum* aggregator.

D Alternative aggregators

Apart from the aggregators we described and used above, there are other aggregators that we have experimented with or that are used in literature, you can find some examples below. Domain-specific metrics could also be an effective choice.

Softmax and softmin aggregations As an alternative to *max* and *min*, *softmax* and *softmin* are differentiable and can be weighted in the case of weighted graphs or attention networks. They also allow an asymmetric message passing in the direction of the strongest signal. Equation 15 presents the direct neighbour formulation of the softmax and softmin operations, where X^l are the nodes' features at layer l with respect to node i and $N(i)$ is the neighbourhood of node i:

$$\text{softmax}_i(X^l) = \sum_{j\in N(i)} \frac{X_j^l\ \exp(X_j^l))}{\sum_{k\in N(i)} \exp(X_k^l)} \quad , \quad \text{softmin}_i(X^l) = -\text{softmax}_i(-X^l) \tag{15}$$