On the Unreasonable Effectiveness of Feature Propagation in Learning on Graphs with Missing Node Features

Anonymous Author(s)
Affiliation
Address
email

Abstract

While Graph Neural Networks (GNNs) have recently become the *de facto* standard for modeling relational data, they impose a strong assumption on the availability of the node or edge features of the graph. In many real-world applications, however, features are only partially available; for example, in social networks, age and gender are available only for a small subset of users. We present a general approach for handling missing features in graph machine learning applications that is based on minimization of the Dirichlet energy and leads to a diffusion-type differential equation on the graph. The discretization of this equation produces a simple, fast and scalable algorithm which we call Feature Propagation. We experimentally show that the proposed approach outperforms previous methods on seven common node-classification benchmarks and can withstand surprisingly high rates of missing features: on average we observe only around 4% relative accuracy drop when 99% of the features are missing. Moreover, it takes only 10 seconds to run on a graph with $\sim$2.5M nodes and $\sim$123M edges on a single GPU.

1 Introduction

Graph Neural Networks (GNNs) [6, 19, 21, 30, 40, 47] have been successful on a broad range of problems and in a variety of fields [13, 14, 17, 38, 43, 54, 60]. GNNs typically operate by a message-passing mechanism [3, 20], where at each layer, nodes send their feature representations ("messages") to their neighbors. The feature representation of each node is initialized to their original features, and is updated by repeatedly aggregating incoming messages from neighbors. Being able to combine the topological information with feature information is what distinguishes GNNs from other purely topological learning approaches such as random walks [22, 36] or label propagation [58], and arguably what leads to their success.

GNN models typically assume a fully observed feature matrix, where rows represent nodes and columns feature channels. However, in real-world scenarios, each feature is often only observed for a subset of the nodes. For example, demographic information can be available for only a small subset of social network users, while content features are generally only present for the most active users. In a co-purchase network, not all products may have a full description associated with them. With the rising awareness around digital privacy, data is increasingly available only upon explicit user consent. In all the above cases, the feature matrix contains missing values and most existing GNN models cannot be directly applied.

While classic imputation methods [29, 32, 55] can be used to fill the missing values of the feature matrix, they are unaware of the underlying graph structure. Graph Signal Processing, a field attempting to generalize classical Fourier analysis to graphs, offers several methods that reconstruct signals on

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

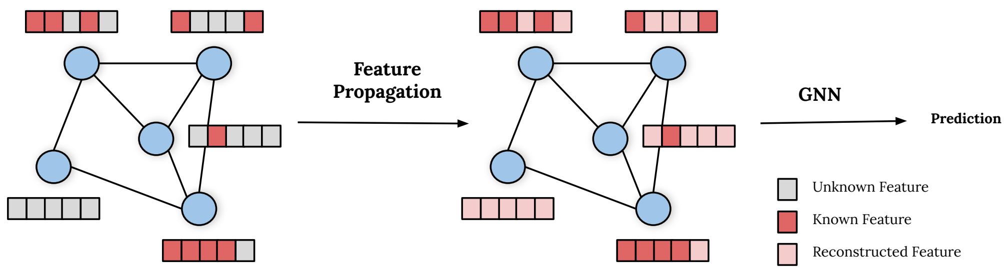

Figure 1: A diagram illustrating our Feature Propagation framework. On the left, a graph with missing node features. In the initial reconstruction step, Feature Propagation reconstructs the missing features by iteratively diffusing the known features in the graph. Subsequently, the graph and the reconstructed node features are fed into a downstream GNN model, which then produces a prediction.

graphs [34]. However, they do not scale beyond graphs with a few thousand nodes, making them infeasible for practical applications. More recently, SAT [10], GCNMF [44] and PaGNN [26] have been proposed to adapt GNNs to the case of missing features. However, they are not evaluated at high missing features rates ($> 90\%$), which occur in many real-world scenarios, and where we find them to suffer. Moreover, they are unable to scale to graphs with more than a few hundred thousand nodes. At the time of writing, PaGNN is the state-of-the-art method for node classification with missing features.

Contributions We present a general approach for handling missing node features in graph machine learning tasks. The framework consists of an initial diffusion-based feature reconstruction step followed by a downstream GNN. The reconstruction step is based on Dirichlet energy minimization, which leads to a diffusion-type differential equation on the graph. Discretization of this differential equation leads to a very simple, fast, and scalable iterative algorithm which we call Feature Propagation (FP). FP outperforms state-of-the-art methods on six standard node-classification benchmarks and presents the following advantages:

- **Theoretically Motivated**: FP emerges naturally as the gradient flow minimizing the Dirichlet energy and can be interpreted as a diffusion equation on the graph with known features used as boundary conditions. This contributes to the promising direction of building continuous-time models on graphs.
- **Robust to high rates of missing features**: FP can withstand surprisingly high rates of missing features. In our experiment, we observe on average around 4% relative accuracy drop when up to 99% of the features are missing. In comparison, GCNMF and PaGNN have an average drop of 53.33% and 21.25% respectively. This finding has important implications especially in scenarios where the cost of sampling (observing features on nodes) is high or sampling is not possible altogether.
- **Generic**: FP can be combined with any GNN model to solve the downstream task; in contrast, GCNMF and PaGNN are specific GCN-type models.
- **Fast and Scalable**: FP takes only around 10 seconds for the reconstruction step on OGBN-Products (a graph with $\sim$2.5M nodes and $\sim$123M edges) on a single GPU. GCNMF and PaGNN run out-of-memory on this dataset.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with $n \times n$ adjacency matrix $\mathbf{A}$ and a node feature vector[1] $\mathbf{x} \in \mathbb{R}^n$. The *graph Laplacian* is an $n \times n$ positive semi-definite matrix $\boldsymbol{\Delta} = \mathbf{I} - \tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix and $\mathbf{D} = \mathrm{diag}(\sum_j a_{1j}, \ldots, \sum_j a_{nj})$ is the diagonal degree matrix.

[1]For convenience, we assume scalar node features. Our derivations apply straightforwardly to the case of d-dimensional features represented as an $n \times d$ matrix $\mathbf{X}$.

Denote by $V_k \subseteq V$ the set of nodes on which the features are *known*, and by $V_u = V_k^c = V \setminus V_k$ the *unknown* ones. We further assume the ordering of the nodes such that we can write

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_u \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_{kk} & \mathbf{A}_{ku} \\ \mathbf{A}_{uk} & \mathbf{A}_{uu} \end{bmatrix} \quad \mathbf{\Delta} = \begin{bmatrix} \mathbf{\Delta}_{kk} & \mathbf{\Delta}_{ku} \\ \mathbf{\Delta}_{uk} & \mathbf{\Delta}_{uu} \end{bmatrix}.$$

Because the graph is undirected, $\mathbf{A}$ is symmetric and thus $\mathbf{A}_{ku}^\top = \mathbf{A}_{uk}$ and $\mathbf{\Delta}_{ku}^\top = \mathbf{\Delta}_{uk}$. We will tacitly assume this in the following discussion.

Graph feature interpolation is the problem of reconstructing the unknown features $\mathbf{x}_u$ given the graph structure G and the known features $\mathbf{x}_k$. The interpolation task requires some prior on the behavior of the features of the graph, which can be expressed in the form of an energy function $\ell(\mathbf{x}, G)$. The most common assumption is feature *homophily* (i.e., that the features of every node are similar to those of the neighbours), quantified using a criterion of *smoothness* such as the Dirichlet energy. Since in many cases the behavior of the features is not known, the energy can possibly be learned from the data.

Learning on a graph with missing features is a transductive learning problem (typically node-wise classification or regression using some GNN architecture) where the structure of the graph G is known while the labels and node features are only partially known on the subsets V_l and V_k of nodes, respectively (that might be different and even disjoint). Specifically, we try to learn a function $\mathbf{f}(\mathbf{x}_k, G)$ such that $f_i \approx y_i$ for $i \in V_l$. Learning with missing features can be done by a pre-processing step of graph signal interpolation (reconstructing an estimate $\tilde{\mathbf{x}}$ of the full feature vector $\mathbf{x}$ from $\mathbf{x}_k$) independent of the learning task, followed by the learning task of $\mathbf{f}(\tilde{\mathbf{x}}, G)$ on the inferred fully-featured graph. In some settings, we are not interested in recovering the features *per se*, but rather ensuring that the output of the *function* $\mathbf{f}$ on these features is correct – arguably a more 'forgiving' setting.

3 Feature Propagation

We assume to be given $\mathbf{x}_k$ and attempt to find the missing node features $\mathbf{x}_u$ by means of interpolation that minimizes some energy $\ell(\mathbf{x}, G)$. In particular, we consider the *Dirichlet energy* $\ell(\mathbf{x}, G) = \frac{1}{2}\mathbf{x}^\top \mathbf{\Delta} \mathbf{x} = \frac{1}{2}\sum_{ij} \tilde{a}_{ij}(x_i - x_j)^2$, where $\tilde{a}_{ij}$ are the individual entries of the normalized adjacency $\tilde{\mathbf{A}}$. The Dirichlet energy is widely used as a smoothness criterion for functions defined on the nodes of the graph and thus promotes homophily. Functions minimizing the Dirichlet energy are called *harmonic*; without boundary conditions, it is minimized by a constant function.

While the Dirichlet energy is convex and it is possible to derive its minimizer in a closed-form, as shown in Appendix A.2, its computational complexity makes it unfeasible for graphs with many nodes with missing features. Instead, we consider the associated *gradient flow* $\dot{\mathbf{x}}(t) = -\nabla \ell(\mathbf{x}(t))$ as a differential equation with boundary condition $\mathbf{x}_k(t) = \mathbf{x}_k$ whose solution at the missing nodes, $\mathbf{x}_u = \lim_{t\to\infty} \mathbf{x}_u(t)$, provides the desired interpolation.

Gradient flow. For the Dirichlet energy, $\nabla_{\mathbf{x}}\ell = \mathbf{\Delta}\mathbf{x}$ and the gradient flow takes the form of the standard isotropic heat diffusion equation on the graph,

$$\dot{\mathbf{x}}(t) = -\mathbf{\Delta}\mathbf{x}(t) \qquad \text{(IC)}\;\; \mathbf{x}(0) = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_u(0) \end{bmatrix} \qquad \text{(BC)}\;\; \mathbf{x}_k(t) = \mathbf{x}_k$$

where IC and BC stand for initial conditions and boundary conditions respectively. By incorporating the boundary conditions, we can compactly express the diffusion equation as

$$\begin{bmatrix} \dot{\mathbf{x}}_k(t) \\ \dot{\mathbf{x}}_u(t) \end{bmatrix} = -\begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{\Delta}_{uk} & \mathbf{\Delta}_{uu} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_u(t) \end{bmatrix} = -\begin{bmatrix} \mathbf{0} \\ \mathbf{\Delta}_{uk}\mathbf{x}_k + \mathbf{\Delta}_{uu}\mathbf{x}_u(t) \end{bmatrix}. \tag{1}$$

As expected, the gradient flow of the observed features is $\mathbf{0}$, given that they do not change during the diffusion.

The evolution of the missing features can be regarded as a heat diffusion equation with a constant heat source $\mathbf{\Delta}_{uk}\mathbf{x}_k$ coming from the boundary (known) nodes. Since the graph Laplacian matrix is

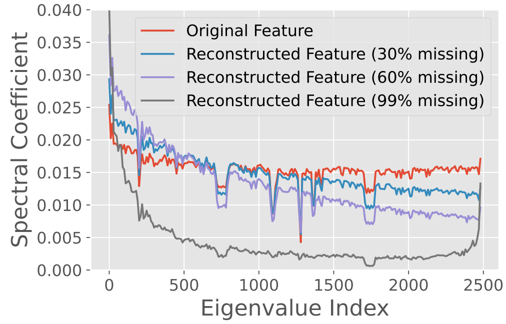

Figure 2: Graph Fourier transform magnitudes of the original Cora features (red) and those reconstructed by FP for varying rates of missing rates (we take the average over feature channels). Since FP minimizes the Dirichlet energy, it can be interpreted as a low-pass filter, which is stronger for a higher rate of missing features.

positive semi-definite, the Dirichlet energy ℓ is convex. Its global minimizer is given by the solution to the closed-form equation $\nabla_{\mathbf{x}_u}\ell = \mathbf{0}$ and by rearranging the final $|V_u|$ rows of Equation 1 we get the solution $\mathbf{x}_u = -\mathbf{\Delta}_{uu}^{-1}\mathbf{\Delta}_{ku}^{\top}\mathbf{x}_k$. This solution always exists as $\mathbf{\Delta}_{uu}$ is non-singular, by virtue of the following:

Proposition 3.1 (The sub-Laplacian matrix of an undirected connected graph is invertible)**.** *Take any undirected, connected graph with adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, and its Laplacian $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, with $\mathbf{D}$ being the degree matrix of $\mathbf{A}$. Then, for any principle sub-matrix $\mathbf{L}_u \in \mathbb{R}^{b\times b}$ of the Laplacian, where $1 \leq b < n$, $\mathbf{L}_u$ is invertible.*

Proof: See Appendix A.2. Also, while the proposition assumes that the graph is connected, our analysis and method generalize straightforwardly in the case of a disconnected graph as we can simply apply Feature Propagation to each connected component independently.

However, solving a system of linear equations is computationally expensive (incurring $\mathcal{O}(|V_u|^3)$ complexity for matrix inversion) and thus intractable for anything but only small graphs.

Iterative scheme. As an alternative, we can discretize the diffusion equation (1) and solve it by an iterative numerical scheme. Approximating the temporal derivative as forward difference with the time variable t discretized using a fixed step ($t = hk$ for step size $h > 0$ and $k = 1, 2, \ldots$), we obtain the *explicit Euler scheme:*

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - h\begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{\Delta}_{uk} & \mathbf{\Delta}_{uu}\end{bmatrix}\mathbf{x}^{(k)} = \left(\mathbf{I} - \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ h\mathbf{\Delta}_{uk} & h\mathbf{\Delta}_{uu}\end{bmatrix}\right)\mathbf{x}^{(k)} = \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ -h\mathbf{\Delta}_{uk} & \mathbf{I} - h\mathbf{\Delta}_{uu}\end{bmatrix}\mathbf{x}^{(k)}$$

For the special case of $h = 1$, we can use the following observation

$$\tilde{\mathbf{A}} = \mathbf{I} - \mathbf{\Delta} = \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix} - \begin{bmatrix}\mathbf{\Delta}_{kk} & \mathbf{\Delta}_{ku}\\ \mathbf{\Delta}_{uk} & \mathbf{\Delta}_{uu}\end{bmatrix} = \begin{bmatrix}\mathbf{I} - \mathbf{\Delta}_{kk} & -\mathbf{\Delta}_{ku}\\ -\mathbf{\Delta}_{uk} & \mathbf{I} - \mathbf{\Delta}_{uu}\end{bmatrix},$$

to write the iteration formula as

$$\mathbf{x}^{(k+1)} = \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \tilde{\mathbf{A}}_{uk} & \tilde{\mathbf{A}}_{uu}\end{bmatrix}\mathbf{x}^{(k)}. \tag{2}$$

The Euler scheme is the gradient descent of the Dirichlet energy. Thus, applying the scheme decreases the Dirichlet energy and results in the features becoming increasingly smooth. Iteration (2) can be interpreted as successive low-pass filtering. Figure 2 depicts the magnitude of the graph Fourier coefficients of the original and reconstructed features on the Cora dataset, indicating that the higher the rate of missing features, the stronger the low-pass filtering effect.

The following results shows that the iterative scheme with $h = 1$ always converges and its steady state is equal to the closed form solution. Importantly, the solution does not depend on the initial values $\mathbf{x}_u^{(0)}$ given to the unknown features.

Proposition 3.2. *Take any undirected and connected graph with adjacency matrix* $\mathbf{A} \in \{0,1\}^{n\times n}$*, and normalised Adjacency* $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$*, with* $\mathbf{D}$ *being the degree matrix of* $\mathbf{A}$*. Let* $\mathbf{x} = \mathbf{x}^{(0)} \in \mathbf{R}^n$ *be the initial feature vector and define the following recursive relation*

$$\mathbf{x}^{(k)} \quad = \quad \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \tilde{\mathbf{A}}_{uk} & \tilde{\mathbf{A}}_{uu} \end{bmatrix} \mathbf{x}^{(k-1)}.$$

Then this recursion converges and the steady state is given to be

$$\lim_{n\to\infty} \mathbf{x}^{(n)} = \begin{bmatrix} \mathbf{x}_k \\ -\mathbf{\Delta}_{kk}^{-1}\tilde{\mathbf{A}}_{uk}\mathbf{x}_k \end{bmatrix}.$$

Proof: See Appendix A.3.

Feature Propagation Algorithm. We can notice that the update in Equation 2 is equivalent to first multiplying the feature vector $\mathbf{x}$ by the original diffusion matrix $\tilde{\mathbf{A}}$, and then resetting the known features to their true value. This gives us Algorithm 1, an extremely simple and scalable iterative algorithm to reconstruct the missing features on a graph, which we refer to as *Feature Propagation* (FP). While $\mathbf{x}_u$ can be initialized to any value, in practice we initialize $\mathbf{x}_u$ to zero and find 40 iterations to be enough to provide convergence for all datasets we experimented on. At each iteration, the diffusion occurs from the nodes with known features to the nodes with unknown features as well as among the nodes with unknown features.

Algorithm 1 Feature Propagation

```
Input: feature vector x, diffusion matrix Ã
y ← x
while x has not converged do
    x ← Ãx          ▷ Propagate features
    x_k ← y_k       ▷ Reset known features
end while
```

4 Related Work

Label Propagation. The proposed algorithm bears some similarity with Label Propagation [58] (LP), which predicts a class for each node by propagating the known labels in the graph. Differently from our setting of diffusion of continuous node features, they deal with discrete label classes directly, resulting in a different diffusion operator. However, the key difference between them lies in how they are used. Importantly, LP is used to directly perform node classification, taking into account only the graph structure and being unaware of node features. On the other hand, FP is used to reconstruct missing features, which are then fed into a downstream GNN classifier. FP allows a GNN model to effectively combine features and graph structures, even when most of the features are missing. Our experiments show that FP+GNN always outperforms LP, even in cases of extremely high rates of missing features, suggesting the effectiveness of FP. Also, the derived scheme is a special case of Neural Graph PDEs [8], which are in turn related to the iterative scheme presented in [56].

Matrix completion. Several optimization-based approaches [7, 25] as well as learning-based approaches [29, 32, 55] have been proposed to solve the matrix completion problem. However, they are unaware of the underlying graph structure. Graph matrix completion [27, 33, 37, 46] extends the above approaches to make use of an underlying graph. Similarly, Graph Signal Processing offers several methods to interpolate signals on graphs. [34] prove the necessary conditions for a graph signal to be recovered perfectly, and provide a corresponding algorithm. However, due to the optimisation problems involved, most above approaches are too computationally intensive and cannot scale to graphs with more than ∼1,000 nodes. Moreover, the goal of all above approaches is to reconstruct the missing entries of the matrix, rather than solving a downstream task.

Extending GNNs to missing node features. SAT [10] consists of a Transformer-like model for feature reconstruction and a GNN model to solve the downstream task. GCNMF [44] adapts GCN [30] to the case of missing node features by representing the missing data with a Gaussian mixture model. PaGNN [26] is a GCN-like model which uses a partial message-passing scheme to only propagate observed features. While showing a reasonable performance for low rates of missing features, these methods suffer in regimes of high rates of missing features, and do not scale to large graphs.

Other related GNN works. Several papers investigate how to augment GNNs when no node features are available [12], as well as investigating the performance of GNNs with random features [1, 39]. Dirichlet energy minimization has been widely used as a regularizer in several graph-related tasks [49, 56, 59]. Discretizion of continuous diffusion on graphs has already been explored in [8] and [51]. Propagation on the graph has also been studied as a solution to the different problem of node regression on multi-relational graphs [4]. Other methods have investigated propagating node features [9, 18, 50], however not in the scenario of missing features. The boundary conditions given by the available features in FP's diffusion equation (enforced by resetting the known feature after each iteration in the algorithm) is what makes it different from other propagation approaches and makes it an effective solution to the missing features problem. While [9, 18, 50] assume to observe all features, and then modify all features, FP assumes to observe only a subset of the features and modifies only the unobserved ones.

5 Experiments and Discussion

Datasets. We evaluate on the task of node classification on several benchmark datasets: Cora, Citeseer and PubMed [41], Amazon-Computers, Amazon-Photo [42] and OGBN-Arxiv [24]. To test the scalability of our method, we also test it on OGBN-Products (2,449,029 nodes, 123,718,280 edges). We report dataset statistics in table 3 (Appendix).

Baselines. We compare to two strong feature-agnostic baselines: Label Propagation [58], which only makes use of the graph structure by propagating labels on the graph, and Graph Positional Encodings [15], which consist in computing the top k eigenvectors of the Laplacian matrix and treating them as node features in input to a GNN. We additionally compare to feature-imputation methods that are graph-agnostic, such as setting the missing features to 0 (Zero), a random value from a standard Gaussian (Random), or the global mean of that feature over the graph (Global Mean) [2]. We also compare to a simple graph-based imputation baseline, which sets a missing feature to the mean (of that same feature) over the neighbors of a node (Neighbor Mean). We additionally experiment with MGCNN [33], a geometric graph completion method which learns how to reconstruct the missing features by making use of the observed features and the graph structure. For all the above baselines, as well as for our Feature Propagation, we experiment with both GCN [30] and GraphSage with mean aggregator [23] as downstream GNNs. We also compare to recently state-of-the-art methods for learning in the missing features setting (GCNMF [44] and PaGNN [26]). For GCNMF we use the publicly available code.[3] We could not find publicly available code for PaGNN so use our own implementation for this comparison. We do not compare to other commonly used imputation based methods such as VAE [29] or GAIN [55], nor to the Transformer-based method SAT [10], as they have previously been shown to consistently underperform GCNMF and PaGNN [26, 44].

Experimental Setup. We report the mean and standard error of the test accuracy, computed over 10 runs, in all experiments. Each run has a different train/validation/test split (apart from OGBN datasets where we use the provided splits) and mask of missing features[4]. The splits are generated at random by assigning 20 nodes per class to the training set, 1500 nodes in total to the validation set and the rest to the test set, similar to [31]. For a fair comparison, we use the same standard hyperparameters for all methods across all experiments. We train using the Adam [28] optimizer with a learning rate of 0.005 for a maximum of 10000 epochs, combined with early stopping with a patience of 200. Downstream GNN models (as well as GCNMF and PaGNN) use 2 layers with a hidden dimension of 64 and a dropout rate of 0.5 for all datasets, apart from OGBN datasets where 3 layers and a hidden dimension of 256 are used. For OGBN-Arxiv we also employ the Jumping Knowledge scheme [53] with max aggregation. Feature Propagation uses 40 iterations to diffuse the features, as we found this to be enough to reach convergence on all datasets. We want to emphasize that we did not perform any hyperparameter tuning, and FP proved to perform consistently with any reasonable choice of hyperparameters. We use neighbor sampling [23] when training on OGBN-Products. All experiments are conducted on an AWS p3.16xlarge machine with 8 NVIDIA V100 GPUs with 16GB of memory each, and took around 4 GPU days in total to perform.

[2] If a feature is not observed for any of the node's neighbors, we set it to zero.

[3] https://github.com/marblet/GCNmf

[4] Each entry of the feature matrix is independently missing with a probability equal to the missing rate.

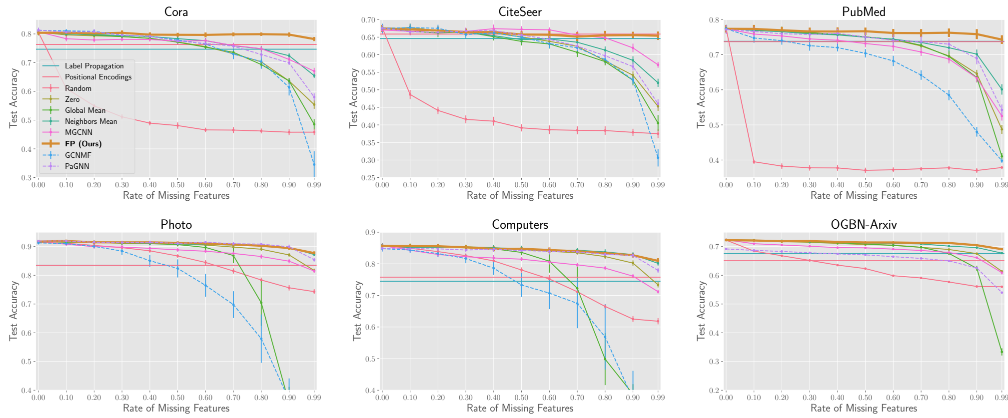

Figure 3: Test accuracy for varying rate of missing features on six common node-classification benchmarks. For methods that require a downstream GNNs, a 2-layer GCN [30] is used. On OGBN-Arxiv, GCNMF goes out-of-memory and is not reported.

Dataset	Full Features	50.0% Missing	90.0% Missing	99.0% Missing
Cora	80.39%	79.70%(-0.86%)	79.77%(-0.77%)	78.22%(-2.70%)
CiteSeer	67.48%	65.74%(-2.57%)	65.57%(-2.82%)	65.40%(-3.08%)
PubMed	77.36%	76.68%(-0.89%)	75.85%(-1.96%)	74.29%(-3.97%)
Photo	91.73%	91.29%(-0.48%)	89.48%(-2.46%)	87.73%(-4.36%)
Computers	85.65%	84.77%(-1.04%)	82.71%(-3.43%)	80.94%(-5.51%)
OGBN-Arxiv	72.22%	71.42%(-1.10%)	70.47%(-2.43%)	69.09%(-4.33%)
OGBN-Products	78.70%	77.16%(-1.96%)	75.94%(-3.51%)	74.94%(-4.78%)
Average	79.08%	78.11%(-1.27%)	77.11%(-2.48%)	75.80%(-4.10%)

Table 1: Performance of Feature Propagation (combined with a GCN model) for 50%, 90% and 99% of missing features, and relative drop compared to the performance of the same model when all features are present. On average, our method loses only 2.50% of relative accuracy with 90% of missing features, and 4.12% with 99% of missing features.

Dataset	GCNMF	PaGNN	Label Prop.	Pos. Enc.	FP (Ours)
Cora	34.54±2.07	58.03±0.57	74.68±0.36	76.33±0.26	**78.22**±0.32
CiteSeer	30.65±1.12	46.02±0.58	64.60±0.40	**65.87**±0.37	65.40±0.54
PubMed	39.80±0.25	54.25±0.70	73.81±0.56	73.70±0.29	**74.29**±0.55
Photo	29.64±2.78	85.41±0.28	83.45±0.94	83.45±0.26	**87.73**±0.27
Computers	30.74±1.95	77.91±0.33	74.48±0.61	75.77±0.47	**80.94**±0.37
OGBN-Arxiv	OOM	53.98±0.08	67.56±0.00	65.08±0.04	**69.09**±0.06
OGBN-Products	OOM	OOM	74.42±0.00	OOM	**74.94**±0.07

Table 2: Performance of GCNMF, PaGNN and FP(+GCN) with 99% of features missing, as well as Label Propagation and Positional Encodings (which are feature-agnostic). GCNMF and PaGNN perform respectively 58.33% and 21.25% worse in terms of relative accuracy in this scenario compared to when all the features are present. In comparison, FP has only a 4.12% drop.

Node Classification Results. Figure 3 shows the results for different rates of missing features (x-axis), when using GCN as a downstream GNN (results with GraphSAGE are reported in Figure 6 of the Appendix). FP matches or outperforms other methods in all scenarios. Both GCNMF and PaGNN are consistently outperformed by the simple Neighbor Mean baseline. This is not completely unexpected, as Neighbor Mean can be seen as a first-order approximation of Feature Propagation, where only one step of propagation is performed (and with a slightly different normalization of the diffusion operator). We elaborate on the relation between Neighbor Mean and Feature Propagation as well as on the results of the other baselines in Section A.5 of the Appendix. Interestingly, most methods perform extremely well up to 50% of missing features, suggesting that in general node features are redundant, as replacing half of them with zeros (*Zero* baseline) has little effect on the performance. The gap between methods opens up from around 60% of missing features, and is particularly large for extremely high rates of missing features (90% or 99%): FP is the only feature-aware method which is robust to these high rates on all datasets (see Table 2). Moreover, FP outperforms or matches Label Propagation and Positional Encodings on all datasets, even in the extreme case of 99% missing features. On some datasets, such as Cora, Photo, and Computers, the gap is especially significant. We conclude that reconstructing the missing features using FP is indeed useful for the downstream task. We highlight the surprising results that, on average, FP with 99% missing features performs only 4.12% worse (in relative accuracy terms) than the same GNN model used with no missing features, compared to 58.33% and 21.25% worse for GCNMF and PaGNN respectively.

Run-time analysis. Feature Propagation scales to extremely large graphs, as it only consists of repeated sparse-to-dense matrix multiplications. Moreover, it can be regarded as a pre-processing step, and performed only once, separately from training. In Figure 4 we compare the run-time to complete the training of the model for FP, PaGNN and GCNMF. The time for FP includes both the feature propagation step to reconstruct the missing features, as well as training of a downstream GCN model. FP is around 3x faster than PaGNN and GCNMF. The propagation step of FP takes only a fraction of the total running time, and the vast majority of the time is spent in training of the donwstream model. The feature propagation step takes only ~0.6s for Computers, ~0.8s for OGBN-Arxiv and ~10.5s for OGBN-Products using a single GPU. Both PaGNN and GCNMF go out-of-memory on OGBN-Products.

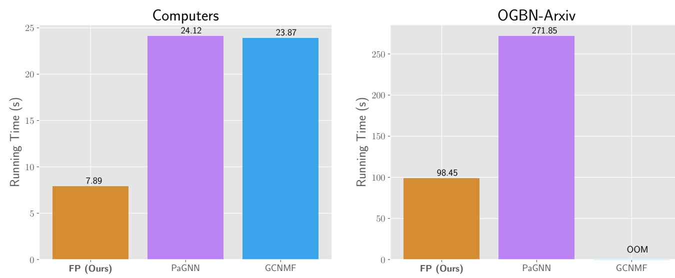

Figure 4: Run-time (in seconds) of FP, PaGNN and GCNMF. FP is 3x faster than both other methods. GCNMF goes out-of-memory (OOM) on OGBN-Arxiv.

6 Conclusion

We have introduced a novel approach for handling missing node features in graph-learning tasks. Our Feature Propagation model can be directly derived from energy minimization, and can be implemented as an efficient iterative algorithm where the features are multiplied by a diffusion matrix, before resetting the known features to their original value. Experiments on a number of datasets suggest that FP can reconstruct the missing features in a way that is useful for the downstream task, even when 99% of the features are missing. FP outperforms recently proposed methods by a significant margin on common benchmarks, while also being extremely scalable.

Limitations. While our method is designed for homophilic graphs, a more general learnable diffusion could be adopted to perform well in low homophily scenarios, as discussed in Section A.1. Feature Propagation is designed for graphs with only one node and edge type, however it could be extended to heterogenous graphs by having separate diffusions for different types of edges and nodes. Finally, Feature Propagation treats feature channels independently. To account for dependencies, diffusion with channel mixing should be used.

References

[1] R. Abboud, İ. İ. Ceylan, M. Grohe, and T. Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2112–2118, 2021.

[2] S. Abu-El-Haija, B. Perozzi, A. Kapoor, H. Harutyunyan, N. Alipourfard, K. Lerman, G. V. Steeg, and A. Galstyan. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning (ICML)*, 2019.

[3] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] E. Bayram. Propagation on multi-relational graphs for node regression. In R. M. Benito, C. Cherifi, H. Cherifi, E. Moro, L. M. Rocha, and M. Sales-Pardo, editors, *Complex Networks & Their Applications X*, pages 155–167, Cham, 2022. Springer International Publishing.

[5] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, 1994.

[6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[7] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[8] B. P. Chamberlain, J. R. Rowbottom, M. I. Gorinova, S. Webb, E. Rossi, and M. M. Bronstein. GRAND: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, 2021.

[9] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. D. Du, and J.-R. Wen. Scalable graph neural networks via bidirectional propagation. 2020.

[10] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. Tsang. Learning on attribute-missing graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 2020.

[11] F. R. K. Chung. *Spectral Graph Theory*. Number 92. American Mathematical Soc., 1997.

[12] H. Cui, Z. Lu, P. Li, and C. Yang. On positional and structural node features for graph neural networks on non-attributed graphs. *International Workshop on Deep Learning on Graphs (DLG-KDD)*, 2021.

[13] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Veličković. Traffic Prediction with Graph Neural Networks in Google Maps. 2021.

[14] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 2224–2232, 2015.

[15] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[16] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[17] P. Gainza, F. Sverrisson, F. Monti, E. Rodolà, M. M. Bronstein, and B. E. Correia. Deciphering interaction fingerprints from protein molecular surfaces. *Nature Methods*, 17(2):184–192, 2020.

[18] J. Gasteiger, A. Bojchevski, and S. Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

[19] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[20] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, 2017.

[21] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[22] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[23] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[24] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open Graph Benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[25] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[26] B. Jiang and Z. Zhang. Incomplete graph representation and learning via partial graph neural networks. *arXiv preprint arXiv:2003.10130*, 2021.

[27] V. Kalofolias, X. Bresson, M. M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *ArXiv preprint arXiv:1408.1717*, 2014.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[29] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations, ICLR*, 2014.

[30] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations, ICLR*, 2017.

[31] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[32] X. Liu, X. Zhu, M. Li, L. Wang, E. Zhu, T. Liu, M. Kloft, D. Shen, J. Yin, and W. Gao. Multiple kernel k-means with incomplete kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(5):1191–1204, 2020.

[33] F. Monti, M. M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3700–3710, 2017.

[34] S. K. Narang, A. Gadde, and A. Ortega. Signal processing techniques for interpolation in graph structured data. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5445–5449, 2013.

[35] K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[36] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[37] N. Rao, H.-F. Yu, P. K. Ravikumar, and I. S. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[38] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning (ICML)*, 2020.

[39] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM*, pages 333–341. SIAM, 2021.

[40] 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, 2008.

[41] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[42] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS*, 2018.

[43] J. Shlomi, P. Battaglia, and J.-R. Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[44] H. Taguchi, X. Liu, and T. Murata. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155–168, 2021.

[45] M. Thorpe, T. M. Nguyen, H. Xia, T. Strohmer, A. Bertozzi, S. Osher, and B. Wang. GRAND++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2022.

[46] R. van den Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. *International Conference on Learning Representations, ICLR*, 2018.

[48] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[49] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1168–1175, New York, NY, USA, 2008. Association for Computing Machinery.

[50] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 09–15 Jun 2019.

[51] L.-P. Xhonneux, M. Qu, and J. Tang. Continuous graph neural networks. In *International Conference on Machine Learning*, pages 10432–10441. PMLR, 2020.

[52] L.-P. A. C. Xhonneux, M. Qu, and J. Tang. Continuous graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[53] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018.

[54] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 974–983. Association for Computing Machinery, 2018.

[55] J. Yoon, J. Jordon, and M. van der Schaar. GAIN: Missing data imputation using generative adversarial nets. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5689–5698. PMLR, 2018.

[56] D. Zhou and B. Schölkopf. A regularization framework for learning from graph data. In *Workshop on Statistical Relational Learning (ICML)*, 2004.

[57] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[58] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. In *Technical Report CMU-CALD-02-107, Carnegie Mellon University*, 2002.

[59] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 912–919. AAAI Press, 2003.

[60] M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

A Appendix

A.1 Algorithm Discussion

Extension to Vector-Valued Features. Algorithm 1 extends seamlessly to vector-valued features by simply replacing the feature vector $\mathbf{x}$ with a $n \times d$ feature matrix $\mathbf{X}$, where d is the number of features. Multiplying the diffusion matrix $\mathbf{A}$ by the feature matrix $\mathbf{X}$ diffuses each feature channel independently. Interestingly, it would not be trivial to extend Equation 2 to vector-valued features without noticing its equivalence with Algorithm 1, as each node could have different missing features, leading to different sub-matrices $\tilde{\mathbf{A}}_{uk}$ and $\tilde{\mathbf{A}}_{uu}$ for each feature channel.

Learning. One significant advantage of FP is that it can be easily combined with any graph learning model to generate predictions for the downstream task. Moreover, FP is not aimed at merely reconstructing the node features. Instead, by only reconstructing the lower frequency components of the signal, it is by design very well suited to be combined with GNNs, which are known to mainly leverage these lower frequency components [50]. Our approach is generic and can be used for any graph-related task for missing features, such as node classification, link prediction and graph classification. In this paper, we focus on node classification.

Oversmoothing. Figure 2 shows that the more features are missing, the smoother the reconstruction produced by FP is. Despite this, FP does not suffer from oversmoothing [35], a term used when node representations converge to similar values. Oversmoothing is caused by repeated diffusion and occurs widely when stacking more than a few layers of the most popular GNNs such as GCN [30], GAT [47] or SGC [50]. However, the boundary conditions in the Feature Propagation diffusion equation prevent the reconstructed features from becoming overly smooth, even when using an extremely high number of diffusion steps. This has also been studied by CGNN [52] and GRAND++ [45], which require soft boundary conditions in the form of a source term to prevent oversmoothing, although not in the context of missing features.

When does Feature Propagation work? Since FP can be interpreted as a low-pass filter that smoothes the features on the graph, we expect it to be suitable in the case of homophilic graph data (where neighbors tend to have similar attributes), and, conversely, to suffer in scenarios of low homophily. To verify this, we experiment on the synthetic dataset from [2], which consists of 10 graphs with different levels of homophily. Figure 5 confirms our hypothesis: when the homophily is high, Feature Propagation with 99% of features missing performs similarly to the case when all the features are known. As the homophily decreases, the gap between the two widens to become extremely large in the case of zero homophily. In such scenarios, FP is only slightly better than

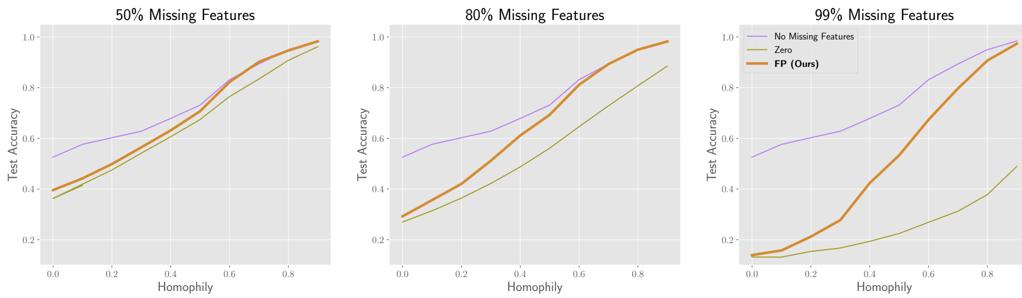

Figure 5: Test accuracy on the synthetic datasets from [2] with different levels of homophily. We use GraphSage as downstream model as it is preferable to GCN on low homophily data [57].

setting the missing features to zero (Zero baseline). This observation calls for a different kind of non-homogeneous diffusion dependent on the features that can potentially be made learnable for low-homophily data. We leave this as future work.

A.2 Closed-Form Solution for Harmonic Interpolation

Given the *Dirichlet energy* $\ell(\mathbf{x}, G) = \frac{1}{2}\mathbf{x}^\top \mathbf{\Delta}\mathbf{x}$, we want to solve for missing features $\mathbf{x}_u = argmin_{\mathbf{x}_u}\ell$, leading to the optimality condition $\nabla_{\mathbf{x}_u}\ell = \mathbf{0}$. From Eq. 1 we find $\nabla_{\mathbf{x_u}}\ell = \mathbf{0}$ to be the solution of $\mathbf{\Delta}_{uk}\mathbf{x}_k + \mathbf{\Delta}_{uu}\mathbf{x}_u = \mathbf{0}$. The unique solution to this system of linear equations is $\mathbf{x}_u = -\mathbf{\Delta}_{uu}^{-1}\mathbf{\Delta}_{uk}\mathbf{x}_k$. We show this solution always exists by proving $\mathbf{\Delta}_{uu}$ is non-singular (Proposition 3.1). The proof of this result follows from the following Lemma.

Lemma A.1. *Take any undirected and connected graph with adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, and normalised Adjacency $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, with $\mathbf{D}$ being the degree matrix of A. Let $\tilde{\mathbf{A}}_{uu}$ be the bottom right submatrix of $\tilde{\mathbf{A}}$ where $1 \leq b < n$. Then $\rho(\tilde{\mathbf{A}}_{uu}) < 1$ where $\rho(\cdot)$ denotes spectral radius.*

Proof. Define

$$\tilde{\mathbf{A}}_{up} = \begin{bmatrix} \mathbf{0}_u & \mathbf{0}_{uk} \\ \mathbf{0}_{ku} & \tilde{\mathbf{A}}_{uu} \end{bmatrix},$$

to be the matrix equal to $\tilde{\mathbf{A}}_{uu}$ in the lower right $b \times b$ sub-matrix and padded with zero entries elsewhere. Clearly $\tilde{\mathbf{A}}_{up} \leq \tilde{\mathbf{A}}$ elementwise and $\tilde{\mathbf{A}}_{up} \neq \tilde{\mathbf{A}}$. Furthermore, $\tilde{\mathbf{A}}_{up} + \tilde{\mathbf{A}}$ represents an adjacency matrix of some strongly connected graph and is therefore irreducible [5, Theorem 2.2.7]. These observations allow us to deduce that $\rho(\tilde{\mathbf{A}}_{up}) < \rho(\tilde{\mathbf{A}})$ [5, Corollary 2.1.5]. Note that $\rho(\tilde{\mathbf{A}}_{up}) = \rho(\tilde{\mathbf{A}}_{uu})$ as $\tilde{\mathbf{A}}_{up}$ and $\tilde{\mathbf{A}}_{uu}$ share the same non-zero eigenvalues. Furthermore, $\rho(\tilde{\mathbf{A}}) \leq 1$ as we can write $\tilde{\mathbf{A}} = \mathbf{I} - \mathbf{\Delta}$ and $\mathbf{\Delta}$ is known to have eigenvalues in the range $[0, 2]$ [11]. Combining these inequalities gives the result $\rho(\tilde{\mathbf{A}}_{uu}) = \rho(\tilde{\mathbf{A}}_{up}) < \rho(\tilde{\mathbf{A}}) \leq 1$. □

Proposition A.2 (The sub-Laplacian matrix of a undirected connected graph is invertible)**.** *Take any undirected, connected graph with adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, and its Laplacian $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, with $\mathbf{D}$ being the degree matrix of $\mathbf{A}$. Then, for any principle sub-matrix $\mathbf{L}_u \in \mathbb{R}^{b\times b}$ of the Laplacian, where $1 \leq b < n$, L_u is invertible.*

Proof. To prove $\mathbf{\Delta}_{uu}$ is non-singular it is enough to show 0 is not an eigenvalue. Note that $\mathbf{\Delta}_{uu} = \mathbf{I} - \tilde{\mathbf{A}}_{uu}$ so 0 is not an eigenvalue if and only if $\tilde{\mathbf{A}}_{uu}$ does not have an eigenvalue equal to 1, which follows from Lemma A.1. □

A.3 Closed-Form Solution for the Euler scheme

Proposition A.3. *Take any undirected and connected graph with adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, and normalised Adjacency $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, with $\mathbf{D}$ being the degree matrix of $\mathbf{A}$. Let $\mathbf{x} =$*

Dataset	Nodes	Edges	Features	Classes
Cora	2,485	5,069	1,433	7
CiteSeer	2,120	3,679	3,703	6
PubMed	19,717	44,324	500	3
Photo	7,487	119,043	745	8
Computers	13,381	245,778	767	10
OGBN-Arxiv	169,343	1,166,243	128	40
OGBN-Products	2,449,029	123,718,280	100	47

Table 3: Dataset statistics.

$\mathbf{x}^{(0)} \in \mathbf{R}^n$ *be the initial feature vector and define the following recursive relation*

$$\mathbf{x}^{(k)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \tilde{\mathbf{A}}_{uk} & \tilde{\mathbf{A}}_{uu} \end{bmatrix} \mathbf{x}^{(k-1)}.$$

Then this recursion converges and the steady state is given to be

$$\lim_{n\to\infty} \mathbf{x}^{(n)} = \begin{bmatrix} \mathbf{x}_k \\ -\mathbf{\Delta}_{kk}^{-1}\tilde{\mathbf{A}}_{uk}\mathbf{x}_k \end{bmatrix}.$$

Proof. The recursive relation can be written in the following form

$$\begin{bmatrix} \mathbf{x}_k^{(k)} \\ \mathbf{x}_u^{(k)} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_l & \mathbf{0}_{ku} \\ \tilde{\mathbf{A}}_{uk} & \tilde{\mathbf{A}}_{uu} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k^{(k-1)} \\ \mathbf{x}_u^{(k-1)} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_k^{(k-1)} \\ \tilde{\mathbf{A}}_{uk}\mathbf{x}_k^{(k-1)} + \tilde{\mathbf{A}}_{uu}\mathbf{x}_u^{(k-1)} \end{bmatrix}.$$

The first l rows remain the same so we can write $\mathbf{x}_k^{(k)} = \mathbf{x}_k^{(k-1)} = \mathbf{x}_k$ and consider just the convergence of the last u rows

$$\mathbf{x}_u^{(k-1)} = \tilde{\mathbf{A}}_{uk}\mathbf{x}_k + \tilde{\mathbf{A}}_{uu}\mathbf{x}_u^{(k-1)}.$$

We can look at the stationary behaviour by unrolling this recursion and taking the limit to find stationary state

$$\lim_{n\to\infty} \mathbf{x}_u^{(n)} = \lim_{n\to\infty} \tilde{\mathbf{A}}_{uu}^n \mathbf{x}_u^{(0)} + \left(\sum_{i=1}^{n} \tilde{\mathbf{A}}_{uu}^{(i-1)} \right) \tilde{\mathbf{A}}_{uk}\mathbf{x}_k.$$

Using Lemma A.1 we find $\lim_{n\to\infty} \tilde{\mathbf{A}}_{uu}^n \mathbf{x}_u^{(0)} = \mathbf{0}$ and the geometric series converges giving us the following limit

$$\lim_{n\to\infty} \mathbf{x}_u^{(n)} = \left(\mathbf{I}_u - \tilde{\mathbf{A}}_{uu}\right)^{-1} \tilde{\mathbf{A}}_{uk}\mathbf{x}_k = -\mathbf{\Delta}_{kk}^{-1}\tilde{\mathbf{A}}_{uk}\mathbf{x}_k.$$

□

A.4 Baselines' Implementation and Tuning

Label Propagation We use the label propagation implementation provided in Pytorch-Geometric [16]. Since the method is quite sensitive to the value of the α hyperparameter, we perform a gridsearch separately on each dataset over the following values: $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]$.

Positional Encodings We compute the laplacian eigenvectors using SciPy [48] sparse eigenvectors routines. We use the top twenty eigenvectors as positional encodings.

MGCNN We re-implement MGCNN [33] in Pytorch by taking inspiration from the authors' public TensorFlow code [5]. For simplicity, we use the version of the model with only graph convolutional layers and without an LSTM. For the matrix completion training process, we split the observed features into 50% input data, 40% training targets and 10% validation data. Once the MGCNN model is trained, we feed it the matrix with all the observed features to predict the whole feature matrix. This reconstructed features matrix is then used as input for a downstream GNN (as for the feature-imputation baselines).

[5]https://github.com/fmonti/mgcnn

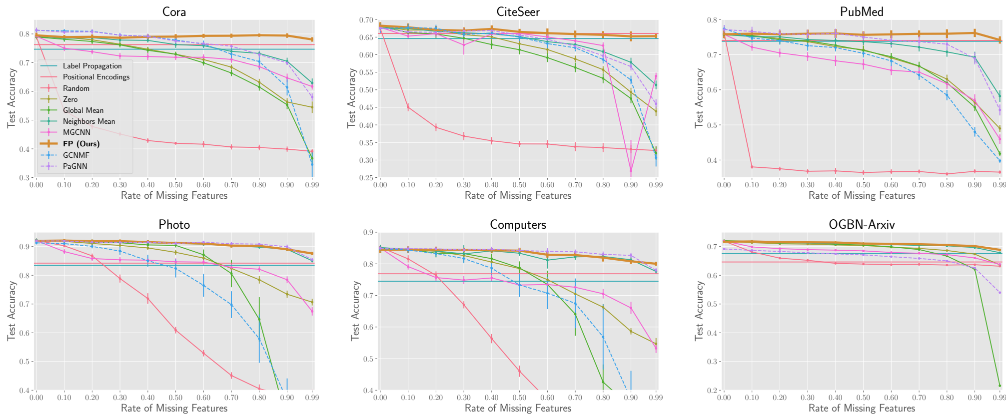

Figure 6: Test accuracy for varying rate of missing features on six common node-classification benchmarks. For methods that require a downstream GNNs, a 2-layer GraphSAGE [23] is used. On OGBN-Arxiv, GCNMF goes out-of-memory and is not reported.

A.5 Discussion Over Baselines' Performance

Neighborhood Averaging As for some intuition to why the simple Neighborhood Averaging performs competitively, let us assume to have a single feature channel and this feature to be homophilous over the graph. When a node has enough neighbors, the average of their features is a good estimate for the feature of the given node. However, as the rate of missing features increases, the feature may be present for only a few neighbors (or none at all), causing the estimate to have a higher variance. On the other hand, Feature Propagation allows information to travel longer distances in the graph by repeatedly multiplying by the diffusion matrix. Even if we do not observe the feature for any of a node's neighbors, it is still possible to estimate it from nodes further away in the graph. This can be observed empirically: the gap between Neighborhood Averaging and Feature Propagation becomes increasingly significant for higher rates of missing features.

Zero vs Random In models such as GCN and GraphSage, where node embeddings are computed as (weighted) average of neighbors embeddings, the effect of the Zero baseline is simply to reduce the norm of the average embeddings of all nodes (since all nodes have the same expected proportion of neighbors with missing features). On the other hand, the Random baseline corrupts this weighted average. More generally, while for a GNN model it could be relatively easy to learn to ignore features set to zero, and only focus on known (non-zero) features, it would be basically impossible for the model to do the same when setting the missing features to a random value.

However, we find Random to perform better than Zero when all features are missing. This is in line with findings in the literature [1, 39], where Random features have been shown to work well in conjunction with GNNs as they act as signatures for the nodes. On the other hand, if all nodes have all zero vectors, it becomes basically impossible to distinguish them. After applying a GNN, all nodes will still have very similar embeddings and the task performance will be close to a random guess.