CSIAM Trans. Appl. Math.
doi: 10.4208/csiam-am.SO-2024-0064

Vol. **x**, No. x, pp. 1-29
xx 2025

Interface Laplace Learning: Learnable Interface Term Helps Semi-Supervised Learning

Tangjun Wang[1], Chenglong Bao[2,3,*] and Zuoqiang Shi[2,3,*]

[1] *Department of Mathematical Sciences, Tsinghua University, Beijing 100084, P.R. China.*
[2] *Yau Mathematical Sciences Center, Tsinghua University, Beijing 100084,P.R. China.*
[3] *BIMSA Yanqi Lake Beijing Institute of Mathematical Sciences and Applications, Beijing 101408, P.R. China.*

Received 17 December 2024; Accepted 5 July 2025

Abstract. We introduce a novel framework, called Interface Laplace learning, for graph-based semi-supervised learning. Motivated by the observation that an interface should exist between different classes where the function value is non-smooth, we introduce a Laplace learning model that incorporates an interface term. This model challenges the long-standing assumption that functions are smooth at all unlabeled points. In the proposed approach, we add an interface term to the Laplace learning model at the interface positions. We provide a practical algorithm to approximate the interface positions using k-hop neighborhood indices, and to learn the interface term from labeled data without artificial design. Our method is efficient and effective, and we present extensive experiments demonstrating that interface Laplace learning achieves better performance than other recent semi-supervised learning approaches at extremely low label rates on the MNIST, FashionMNIST, and CIFAR-10 datasets.

AMS subject classifications: 68T01, 68T45
Key words: Graph-based semi-supervised learning, Laplace learning, interface, nonlocal model.

1 Introduction

The success of machine learning methods often depends on a large amount of training data. However, collecting training data can be labor-intensive, and is sometimes impossible in many application fields due to privacy or safety issues. To alleviate the dependency on training data, semi-supervised learning (SSL) [14, 49] has received great interest in recent years. Semi-supervised learning typically uses a large amount of unlabeled data, together with the labeled data, to improve model performance and generalization ability. The idea of combining labeled and unlabeled data has been widely used long before the

*Corresponding author. *Email addresses:* wangtj20@mails.tsinghua.edu.cn (T. Wang), clbao@mail.tsinghua.edu.cn (C. Bao), zqshi@tsinghua.edu.cn (Z. Shi)

term SSL was coined [8, 22]. By incorporating the geometric structure or data distribution of unlabeled data, SSL algorithms aim to extract more informative features, thereby enhancing the performance of machine learning models.

This paper focuses on a type of SSL method: graph-based SSL [38]. Graph-based SSL algorithms have received much attention as the graph structure can effectively encode relationships among data points, thereby allowing for full utilization of the information contained in unlabeled data. Graph-based SSL is based on the assumption that nearby nodes tend to have the same labels. In a graph, each sample is represented by a vertex, and the weighted edge measures the similarity between samples. One of the most widely used methods in Graph-based SSL is the Gaussian fields and harmonic functions algorithm [48], later commonly called Laplace learning. Laplace learning aims to minimize the graph Dirichlet energy with the constraint on labeled points, resulting in a harmonic function. Many variants of Laplace learning have been proposed. One way is to replace the hard label constraint with soft label regularization [2,5,25,44]. Another way is to generalize the ℓ^2 distance, which corresponds to Laplace learning, into ℓ^p distance, known as p-Laplace learning [9, 10, 19, 37, 45]. In the limit as p approaches infinity, p-Laplace learning is called Lipschitz learning [29].

However, it has been observed that Laplace learning and its variants exhibit poor performance when the label rate is low [12, 19, 34]. In these situations, the solutions tend to converge to non-informative, nearly constant functions with spikes near labeled points, significantly deteriorating the model's accuracy. To address the issue, several methods have been proposed, e.g. higher-order regularization [46], graph re-weighting [13, 35], spectral cutoff [6] and centered kernel [33]. While these methods have been explored, they are either much more computationally burdensome, or still perform poorly when the label rate is extremely low. Recently, Poisson learning [12] has shown promising results in this challenging scenario. Poisson learning replaces the Dirichlet boundary conditions in the Laplace equation with a source term in the Poisson equation, achieving a significant performance increase in classification tasks under extreme label rates.

Nonetheless, most existing methods assume that the solution should exhibit a certain degree of smoothness across all unlabeled points. In this work, we demonstrate that, ideally, there should exist an interface between two different classes. The solution should exhibit discontinuity on the interface, rather than being globally smooth. Interface problems are commonly encountered in fields like materials science [20], fluid dynamics [16, 36], and electromagnetic [32, 39], where the solution is expected to have clear boundaries between different regions or classes, rather than a smooth transition.

In this paper, we formulate the interface problem in graph-based SSL as solving the Laplace equation with jump discontinuity across the interface. By deriving the nonlocal counterpart of this system, we find that it is naturally related to Laplace learning, but with the addition of an explicit interface term. By accounting for the non-smoothness across the interface, our approach can more accurately model the underlying data distribution, resulting in improved performance compared to standard Laplace learning and its variants.

Our main contribution can be summarized as follows:

- We are the first to propose the concept of interface discontinuity in graph-based SSL, offering a new perspective and algorithmic design in this domain.
- We introduce a Laplace learning model that explicitly accounts for interface discontinuities, improving upon existing approaches.
- We develop a practical algorithm to approximate the interface and learn the interface term without deliberate design.
- On benchmark classification tasks with extremely low label rates, our method achieves state-of-the-art performance.

2 Motivation

2.1 Laplace learning and Poisson learning

SSL aims to infer the labels of unlabeled samples $\{x_{m+1},\cdots,x_n\}$ with the help of a labeled set $\{(x_1,y_1),\cdots,(x_m,y_m)\}$. For a classification problem with c classes, the label y_i is often represented as a one-hot vector in $\mathbb{R}^c$, where the l-th element is 1 if the sample belongs to class l, and the other elements are all zeros. Typically, the number of all samples n is much larger than the number of labeled samples m. Graph-based SSL methods construct a graph where the nodes correspond to the samples, $V=\{x_1,x_2,\cdots,x_n\}$. An edge e_{ij} is created between nodes x_i and x_j if the two samples are similar, and a non-negative weight w_{ij} is assigned to the edge, indicating the degree of similarity between x_i and x_j.

Among numerous graph-based SSL approaches, Laplace learning [48], also known as label propagation, is one of the most widely used methods. Laplace learning propagates labels to unlabeled nodes by solving the following Laplace equation:

$$\begin{aligned} &Lu(x_i)=0, \quad m+1\le i\le n,\\ &u(x_i)=y_i, \quad 1\le i\le m, \end{aligned}$$

where L is the graph Laplacian operator given by

$$Lu(x_i)=\sum_{j=1}^{n} w_{ij}\big(u(x_i)-u(x_j)\big).$$

The solution $u:V\to\mathbb{R}^c$ gives the prediction of size c for each vertex x_i. The label decision is then determined by the largest component in $u(x_i)$. In Laplace learning, labeled data is incorporated as Dirichlet boundary conditions $u(x_i)=y_i$.

Recently, Poisson learning [12] is proposed as an alternative to Laplace learning to deal with extremely low label rate. Poisson learning treats unlabeled data in the same

way as Laplace learning, but it differs in the way of handling labeled data. Poisson learning replaces the Dirichlet boundary condition with the Poisson equation and a given source term $y_i-\bar{y}$,

$$\begin{aligned} Lu(x_i)&=0, \qquad && m+1\le i\le n,\\ Lu(x_i)&=y_i-\bar{y}, && 1\le i\le m, \end{aligned}$$

where $\bar{y}=\sum_{j=1}^{m}y_j/m$. Surprisingly, such modification can result in huge improvement under extremely low label rates, e.g. 16.73% $\to$ 90.58% in MNIST [30] 1-label per class classification.

2.2 Interface discontinuity

Both Laplace learning and Poisson learning assume that the graph Laplacian of the target function $Lu(x_i)=0$ on all unlabeled points. However, we question whether this is the most appropriate way to model the underlying data distribution. To illustrate our idea, we will consider a synthetic 2-class classification example.

We uniformly sample 20,000 points $x_i=(a_i,b_i)\in\mathbb{R}^2$ from a unit disk. The decision boundary between the two classes is defined as the union of two half-circles

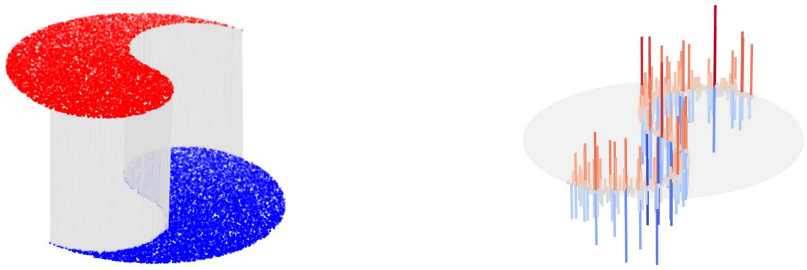
$$\{a<0, a^2+(b-0.5)^2=0.5^2\}\cup\{a\ge 0, a^2+(b+0.5)^2=0.5^2\}.$$

The ground truth label for each x_i is in $\{-1,+1\}\in\mathbb{R}$. A 3D visualization of this toy example is given in Fig. 1(a). To construct the similarity matrix $\mathbf{W}=(w_{ij})\in\mathbb{R}^{n\times n}$, we use a Gaussian kernel defined as

$$w_{ij}=\exp\left(-\frac{4\|x_i-x_j\|^2}{d_K(x_i)^2}\right), \tag{2.1}$$

where $d_K(x_i)$ is the distance between x_i and its K-th nearest neighbor. We choose $K=10$, and we sparsify the matrix $\mathbf{W}$ by truncating the weight for points farther than the K-th nearest neighbor to zero.

We will examine the problem from two different perspectives. Firstly, we assign the ground truth labels to the function $u(x_i)$ and then compute and plot $Lu(x_i)\in\mathbb{R}$ in



(a) Ground truth (b) $Lu(x_i)\neq 0$ on the interface

Figure 1: 3D visualization of a synthetic classification example.

Fig. 1(b). From the figure, it is clear that the Laplacian of the labeling function is nonzero along the interface between the two classes, while being zero in the interior of each class. This nonzero Laplacian near the decision boundary is due to the interface discontinuity in the labeling function. However, this interface discontinuity is ignored by both Laplace learning and Poisson learning, as they assume the function is harmonic almost everywhere, except at the labeled points. The assumption of harmonicity contradicts the true Laplacian behavior observed in the ground truth labeling function.

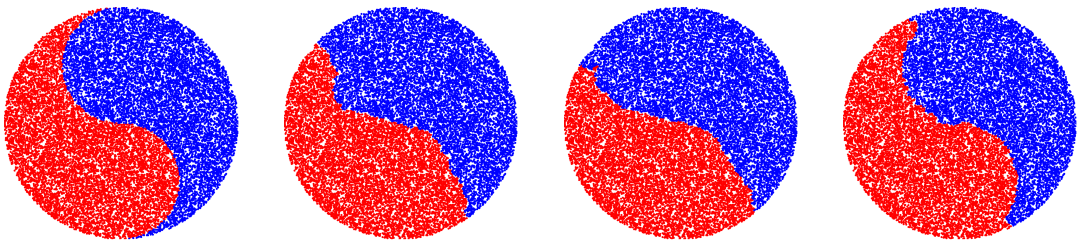
Secondly, we will show that adding an interface term to account for the interface discontinuity is helpful for classification. Specifically, we choose to modify the Laplace equation by introducing an interface term f_i that will be learned from the labeled data

$$\begin{aligned} Lu(x_i) &= 0, \quad i \notin \mathcal{I}, \\ Lu(x_i) &= f_i, \quad i \in \mathcal{I}. \end{aligned} \tag{2.2}$$

Here, $\mathcal{I}$ denotes the set of indices corresponding to the interface positions. This formulation allows us to explicitly model the nonzero Laplacian at the interface, in contrast to the assumptions made by standard Laplace and Poisson learning methods.

The theoretical basis for introducing this interface term to account for interface discontinuity will be provided in the next subsection. In this toy example, the set $\mathcal{I}$ is obtained by identifying the indices where the ground truth Laplacian, as shown in Fig. 1(b), is nonzero. The values of f_i will then be inferred from the labeled points using the algorithm detailed in Section 3. Finally, we use the solution u to Eq. (2.2) for classification.

We randomly select 25 labeled samples from each class, and use Laplace learning, Poisson learning, and our proposed method to classify the remaining data points. The classification results are provided in Fig. 2. We can observe that our method significantly outperforms both Laplace learning and Poisson learning in terms of classification accuracy. Moreover, the decision boundary obtained by our method is also much closer to the ground truth. Indeed, the comparison is unfair because our method utilizes additional information about the interface positions, which the other two methods do not have access to. However, this toy classification problem serves to verify our argument that incorporating an interface term to account for the discontinuity is both necessary and beneficial for improving classification performance.



(a) Ground truth (b) Laplace Acc=93.87% (c) Poisson Acc=91.17% (d) Ours Acc=98.43%

Figure 2: 2D visualization of a synthetic classification example.

2.3 Laplace equation with interface and associate nonlocal model

In this subsection, we will provide a theoretical verification of our approach from the perspective of nonlocal models. Nonlocal models [18] play a crucial role in many fields, such as peridynamical theory of continuum mechanics, nonlocal wave propagation and nonlocal diffusion process [1,3,7,17,26,40]. The terminology nonlocal is the counterpart of local operators. A linear operator L is considered local if the support set $\text{supp}\{L(f)\}\subset \text{supp}\{f\}$ for any function f. Differential operators, such as the Laplace operator Δ, are local operators.

As discussed in previous subsection, the interface problem in SSL can be modeled by the following Laplace equations with jump discontinuity on the interface Γ:

$$\begin{aligned} &\Delta u^+(x)=0, && x\in\mathcal{M}_1,\\ &\Delta u^-(x)=0, && x\in\mathcal{M}_2,\\ &u^+(x)-u^-(x)=1, && x\in\Gamma,\\ &\frac{\partial u^+}{\partial \mathbf{n}}(x)-\frac{\partial u^-}{\partial \mathbf{n}}(x)=0, && x\in\Gamma. \end{aligned} \tag{2.3}$$

Here Γ is a $(k-1)$-dimensional smooth manifold that splits a k-dimensional manifold $\mathcal{M}$ into two submanifolds $\mathcal{M}_1$ and $\mathcal{M}_2$, so that $\mathcal{M}_1\cap\mathcal{M}_2=\emptyset$ and $\mathcal{M}=\mathcal{M}_1\cup\Gamma\cup\mathcal{M}_2$. $\mathbf{n}$ is the outer normal of Γ.

To get nonlocal approximation, we introduce a rescaled kernel function

$$R_\delta(x,y)=C_\delta R\left(\frac{\|x-y\|^2}{4\delta^2}\right),$$

where $R\in C^2([0,1])$ is a non-negative compact function that is supported over $[0,1]$. $C_\delta=(4\pi\delta^2)^{-d/2}$ is a normalization factor for $x\in\mathbb{R}^d$. The rescaled functions $\bar{R}_\delta(x,y)$ and $\bar{\bar{R}}_\delta(x,y)$ are defined similarly, where

$$\bar{R}(r)=\int_r^{+\infty}R(s)ds,\quad \bar{\bar{R}}(r)=\int_r^{+\infty}\bar{R}(s)ds.$$

The role of kernel function is similar to the similarity matrix $\mathbf{W}$ in Eq. (2.1), where only nearest neighbors have nonzero weights. If we assume

$$\int_{\mathcal{M}_1}\bar{R}_\delta(y,s)d\mu_s=\int_{\mathcal{M}_2}\bar{R}_\delta(y,s)d\mu_s=:w_\delta(y),$$

then [43] gives a nonlocal model to approximate Eq. (2.3) as

$$\begin{aligned} &\int_{\mathcal{M}_1}R_\delta(x,y)\big(u_\delta^+(x)-u_\delta^+(y)\big)d\mu_y=\int_\Gamma\bar{R}_\delta(x,y)v_\delta(y)d\tau_y, && x\in\mathcal{M}_1,\\ &\int_{\mathcal{M}_2}R_\delta(x,y)\big(u_\delta^-(x)-u_\delta^-(y)\big)d\mu_y=-\int_\Gamma\bar{R}_\delta(x,y)v_\delta(y)d\tau_y, && x\in\mathcal{M}_2, \end{aligned} \tag{2.4}$$

where

$$v_\delta(y)=\frac{w_\delta(y)+\int_{\mathcal{M}_2}u_\delta^-(s)\bar{R}_\delta(y,s)d\mu_s-\int_{\mathcal{M}_1}u_\delta^+(s)\bar{R}_\delta(y,s)d\mu_s}{2\int_\Gamma\bar{\bar{R}}_\delta(y,s)d\tau_s}$$

is an approximation to the normal derivative up to a constant.

The solutions of the nonlocal model are proven to converge to the solutions of the local model with the rate of $\mathcal{O}(\delta)$ in H^1 norm.

Theorem 2.1 ([43]). *(1) (Well-Posedness) For any $f\in L^2(\mathcal{M})$, there exists a unique solution $(u_\delta^+,u_\delta^-)\in(H^1(\mathcal{M}_1),H^1(\mathcal{M}_2))$ to Eq.* (2.4).

(2) (Convergence) For any $f\in H^1(\mathcal{M}_1\cup\mathcal{M}_2)$, let $(u^+,u^-)\in(H^3(\mathcal{M}_1),H^3(\mathcal{M}_2))$ be the solution to Eq. (2.3)*, and (u_δ^+,u_δ^-) be the solution to Eq.* (2.4)*, then*

$$\|u^+-u_\delta^+\|_{H^1(\mathcal{M}_1)}+\|u^--u_\delta^-\|_{H^1(\mathcal{M}_2)}\le C\delta\big(\|u^+\|_{H^3(\mathcal{M}_1)}+\|u^-\|_{H^3(\mathcal{M}_2)}\big),$$

where the constant C only depends on $\mathcal{M}$ and Γ.

Nonlocal model formulation (2.4) provides theoretical justification of incorporating an interface term to account for interface discontinuity in our algorithm (2.2). On the left-hand side of the nonlocal model, we can identify the integral

$$\int_{\mathcal{M}_1}R_\delta(x,y)\big(u_\delta^+(x)-u_\delta^+(y)\big)d\mu_y$$

with the graph Laplacian operator

$$Lu(x_i)=\sum_{j=1}^n w_{ij}\big(u(x_i)-u(x_j)\big)$$

by discretizing the integral. On the right-hand side, since $\bar{R}_\delta(x,y)$ has compact support, the integral $\int_\Gamma\bar{R}_\delta(x,y)v_\delta(y)d\tau_y$ is nonzero only when x lies in a layer adjacent to the interface Γ with width 2δ. We can relate this adjacent layer of Γ to the interface positions $\mathcal{I}$ introduced earlier. This explains our choice of introducing $f_i=Lu(x_i)\neq 0$ for $i\in\mathcal{I}$.

While the nonlocal formulation provides theoretical justification for the interface term, it is not directly applicable to semi-supervised learning. The variables v_δ and (u_δ^+,u_δ^-) are coupled together, making it challenging to write Eq. (2.4) into a linear system. Furthermore, the interface Γ is not given explicitly, which makes the computation of the integral over Γ impossible. To address these limitations, we will make the interface terms learnable and provide a practical algorithm in subsequent section.

Remark 2.1. The assumptions

$$\int_{\mathcal{M}_1}\bar{R}_\delta(y,s)d\mu_s=\int_{\mathcal{M}_2}\bar{R}_\delta(y,s)d\mu_s,\quad\frac{\partial u}{\partial n}^+(x)=\frac{\partial u}{\partial n}^-(x)$$

for $x\in\Gamma$ are introduced to simplify the expression of the nonlocal model Eq. (2.4). However, these assumptions can be relaxed by introducing coefficients

$$\lambda_1\frac{\partial u}{\partial n}^+(x)=\lambda_2\frac{\partial u}{\partial n}^-(x),\quad \gamma_\delta(y)=\frac{\displaystyle\int_{\mathcal{M}_2}\bar{R}_\delta(y,s)d\mu_s}{\displaystyle\int_{\mathcal{M}_1}\bar{R}_\delta(y,s)d\mu_s}.$$

Correspondingly, the nonlocal model should be slightly modified to incorporate these coefficients. The exact form of the modified nonlocal model is provided in [43].

3 Method

In the previous section, from the premise that the function u should be discontinuous at the interface, we propose a formal algorithm

$$\begin{aligned} Lu(x_i)&=0,\quad i\notin\mathcal{I},\\ Lu(x_i)&=f_i,\quad i\in\mathcal{I}.\end{aligned}$$

However, two key questions remain unsolved:

- For $i\in\mathcal{I}$, how do we decide the value of interface term f?
- For general classification problems where the interface is not provided a priori, how do we determine $\mathcal{I}$?

In the subsequent two subsections, we will provide ideas to address these questions, and the final algorithm will be presented in Section 3.3.

3.1 Interface term learning

The idea is straightforward: We aim to learn an interface term such that the solution to the modified Laplace equation at the labeled points closely matches the given labels. To this end, we use the mean squared error (MSE) on the labeled points as the objective function

$$\mathcal{L}_{\text{MSE}}=\frac{1}{m}\sum_{i=1}^{m}\|u(x_i)-y_i\|_2^2.$$

In addition to the MSE loss, we find that a regularizer on the norm of f_i is beneficial for learning

$$\mathcal{L}_{\text{reg}}=\sum_{i=1}^{n}\|f_i\|_2^2.$$

The introduction of this regularization term is essential, as it (1) ensures the uniqueness of the minimizer of the objective function; (2) facilitates computational efficiency by justifying the later use of the Sherman-Morrison-Woodbury formula; and (3) suppresses

the magnitude of the interface term, thereby preventing overfitting. The final objective function is

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda \mathcal{L}_{\text{reg}},$$

where λ is a weighting factor, later referred to as the ridge parameter. A method for selecting an appropriate λ, based on the value of $\mathcal{L}_{\text{MSE}}$, will be presented in Section 3.3.

By minimizing this objective function, we can learn the interface term from label information. We will formulate the problem as a standard least squares (LS) problem, and the detailed optimization algorithm will be provided in Section 3.3.

3.2 Interface positions approximation

In the synthetic 2-class classification example presented in Section 2.2, the interface positions $\mathcal{I}$ are obtained by leaking label information and given as an additional input. However, in real-world classification tasks, the interface positions are generally not known a priori. Furthermore, compared to the two-dimensional and well-separated synthetic data, real-world datasets often possess high-dimensional data with much more sophisticated data distributions. Even if the interface between categories exists, it is impossible for us to know the exact locations of these interfaces.

Here, we approximate the interface positions by excluding the k-hop neighbors of training samples, where k-hop neighbors of a node v are those within distance k from v in the graph. The distance between two vertices in a graph is defined as the number of edges in a shortest path connecting them. We efficiently obtain the k-hop neighbors using an iterative approach, starting with the training samples (distance-0) and finding their direct neighbors (distance-1), then their neighbors (distance-2), and so on. We provide the Python implementation of this get_interface_idx() method in the following.

```python
# train_idx : numpy array, shape=[m]
# all_idx   : numpy array, shape=[n]
# W         : scipy sparse matrix, shape=[n, n]
# k         : k hop

import numpy as np

def get_interface_idx(train_idx, all_idx, W, k):
    if k == -1:
        return all_idx
    else:
        khop_idx = train_idx
        for _ in range(k):
            neighbor_idx = W[khop_idx].nonzero()[1]
            khop_idx = np.append(khop_idx, neighbor_idx)
            khop_idx = np.unique(khop_idx)
        interface_idx = np.setdiff1d(all_idx, khop_idx)
    return interface_idx
```

The reason we approximate the interface positions in this way is two-fold. First, our approach is based on two key assumptions: (1) $\mathcal{I}$ should lie near the class boundaries; and (2) the labeled points should be representative of their respective classes – i.e. located well within the interior of each class. The second assumption is particularly important in the extremely low label rate regime: if there is only one labeled point per class and it is not representative, good classification performance is unlikely. Based on these assumptions, it is natural to identify the interface points as those far from the labeled ones. Second, choosing $\mathcal{I}$ away from labeled points is also helpful to avoid overfitting. Empirically, when learning the interface term from extremely limited labeled data, directly adjusting f_i for indices i corresponding to labeled samples or their nearby neighbors can have a disproportionately large effect on minimizing the objective. As a result, such easily adjustable components are more prone to overfitting. A discussion of other possible approaches to approximate interface positions will be provided in 4.2.1.

3.3 Algorithm

For ease of writing, we introduce the following notations. $\mathbf{f}=[f_1,\cdots,f_n]^\top$ and $\mathbf{u}=[u(x_1),\cdots,u(x_n)]^\top$ both belong to $\mathbb{R}^{n\times c}$. $\mathbf{y}=[y_1,\cdots,y_m]^\top\in\mathbb{R}^{m\times c}$. The graph Laplacian matrix $\mathbf{L}=\mathbf{D}-\mathbf{W}\in\mathbb{R}^{n\times n}$, where $\mathbf{D}=\mathrm{diag}(d_i)$, $d_i=\sum_{j=1}^n w_{ij}$. $\mathbf{L}$ is related to the graph Laplacian operator L in that $\mathbf{Lu}=[Lu(x_1),\cdots,Lu(x_n)]^\top$. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

The objective is to solve the following optimization problem after we obtain the interface positions $\mathcal{I}$ using get_interface_idx():

$$\begin{aligned}&\underset{\mathbf{f}}{\operatorname{argmin}}\frac{1}{m}\sum_{i=1}^m\|u(x_i)-y_i\|_2^2+\lambda\sum_{i=1}^n\|f_i\|_2^2\\&\text{s.t.}\quad f_i=0,\quad i\notin\mathcal{I}.\end{aligned}\tag{3.1}$$

$\mathbf{u}$ and $\mathbf{f}$ are related by the Poisson equation $\mathbf{Lu}=\mathbf{f}$. Since we cannot directly write $\mathbf{u}=\mathbf{L}^{-1}\mathbf{f}$ because $\mathbf{L}$ is singular, we adopt the iterative solver in Poisson learning [12]. Specifically, we initialize $\mathbf{u}_0$ as an all-zero matrix in $\mathbb{R}^{n\times c}$ and writes the iteration step as

$$\mathbf{u}_{t+1}\ \leftarrow\ \mathbf{u}_t+\mathbf{D}^{-1}(\mathbf{f}-\mathbf{L}\mathbf{u}_t).\tag{3.2}$$

The stopping criterion is determined by another loop $\mathbf{p}_{t+1}=\mathbf{W}\mathbf{D}^{-1}\mathbf{p}_t$, where $\mathbf{p}_0\in\mathbb{R}^n$ is initialized as a vector with ones at the positions of all labeled vertices and zeros elsewhere. Once $\|\mathbf{p}_t-\mathbf{p}_\infty\|_\infty\le 1/n$, where $\mathbf{p}_\infty=\mathbf{W1}/(\mathbf{1}^\top\mathbf{W1})$ represents the invariant distribution, the iteration is stopped. The number of iterations is denoted as T, which is often around 200-300.

We can unroll the iteration Eq. (3.2) and write $\mathbf{u}=\mathbf{u}_T$ in terms of $\mathbf{f}$ directly

$$\mathbf{u}=\sum_{i=0}^{T-1}(\mathbf{D}^{-1}\mathbf{W})^i\mathbf{D}^{-1}\mathbf{f}=:\mathbf{A}\mathbf{f}.$$

Then the objective function in optimization problem (3.1) can be written as

$$\operatorname*{argmin}_{\mathbf{f}} \frac{1}{m}\sum_{i=1}^{m}\|(\mathbf{A}\mathbf{f})_i - y_i\|_2^2 + \lambda\sum_{i=1}^{n}\|f_i\|_2^2.$$

Since $f_i = 0$ for $i \notin \mathcal{I}$, we extract the interface positions of $\mathbf{f}$ and denote them as $\mathbf{f}_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}|\times c}$. $\mathbf{f}_{\mathcal{I}}$ is the interface term to be learned. Correspondingly, we extract the columns of $\mathbf{A}$ and denote them as $\mathbf{A}_{\mathcal{I}} \in \mathbb{R}^{n\times|\mathcal{I}|}$. Since the MSE loss only considers the m labeled points, we can further extract the rows of $\mathbf{A}_{\mathcal{I}}$ that correspond to the m training indices, denoted as $\tilde{\mathbf{A}}_{\mathcal{I}} \in \mathbb{R}^{m\times|\mathcal{I}|}$. This approach significantly reduces the space complexity because $m \ll n$. Finally, the optimization problem (3.1) becomes

$$\operatorname*{argmin}_{\mathbf{f}_{\mathcal{I}}} \frac{1}{m}\|\tilde{\mathbf{A}}_{\mathcal{I}}\mathbf{f}_{\mathcal{I}} - \mathbf{y}\|_F^2 + \lambda\|\mathbf{f}_{\mathcal{I}}\|_F^2. \tag{3.3}$$

It is in fact the well-known ℓ^2 regularized LS problem (also known as ridge regression) with m observations and $|\mathcal{I}|$ variables. It has an explicit solution

$$\mathbf{f}_{\mathcal{I}}^* = \left(\tilde{\mathbf{A}}_{\mathcal{I}}^\top\tilde{\mathbf{A}}_{\mathcal{I}} + m\lambda\mathbf{I}\right)^{-1}\tilde{\mathbf{A}}_{\mathcal{I}}^\top\mathbf{y}. \tag{3.4}$$

After we learn the interface term $\mathbf{f}_{\mathcal{I}}^*$, we can obtain the complete $\mathbf{f}^*$ by filling in the values of $\mathbf{f}_{\mathcal{I}}^*$ at the indices $i \in \mathcal{I}$, and setting the remaining elements to zero.

Notably, we employ the following methods to address complexity issues and solution non-uniqueness.

- In Eq. (3.4), we need to invert an $|\mathcal{I}|\times|\mathcal{I}|$ matrix, which is slow when $|\mathcal{I}|$ is large. A straightforward improvement comes from famous Sherman-Morrison-Woodbury formula, which performs inversion on $m\times m$ matrix instead

$$\mathbf{f}_{\mathcal{I}}^* = \tilde{\mathbf{A}}_{\mathcal{I}}^\top\left(\tilde{\mathbf{A}}_{\mathcal{I}}\tilde{\mathbf{A}}_{\mathcal{I}}^\top + m\lambda\mathbf{I}\right)^{-1}\mathbf{y}. \tag{3.5}$$

- $\mathbf{L}\mathbf{u} = \mathbf{f}$ does not have a unique solution. Thus, we enforce a zero mean on each column of $\mathbf{u}$ by subtracting $\bar{\mathbf{u}} = \sum_{i=1}^{n} u(x_i)/n$ along with each iteration step Eq. (3.2). Consequently, the matrix $\mathbf{A}$ should be slightly modified as

$$\mathbf{A} = \sum_{i=0}^{T-1}\mathbf{J}(\mathbf{D}^{-1}\mathbf{W}\mathbf{J})^i\mathbf{D}^{-1},$$

where $\mathbf{J} = \mathbf{I} - \mathbf{1}\mathbf{1}^\top/n$ is a projection matrix that removes the mean from each column.

Additionally, we provide a method for selecting the ridge parameter λ. By substituting $\mathbf{f}_{\mathcal{I}}$ in the optimization problem Eq. (3.3) with the closed-form solution $\mathbf{f}_{\mathcal{I}}^*$ given in Eq. (3.5), the first term, which corresponds to the MSE loss on labeled points, becomes

$$\frac{1}{m}\|\tilde{\mathbf{A}}_{\mathcal{I}}\mathbf{f}_{\mathcal{I}} - \mathbf{y}\|_F^2 = \frac{1}{m}\left\|\left(\mathbf{I} + \frac{1}{m\lambda}\tilde{\mathbf{A}}_{\mathcal{I}}\tilde{\mathbf{A}}_{\mathcal{I}}^\top\right)^{-1}\mathbf{y}\right\|_F^2 =: g(\lambda).$$

The function $g(\lambda)$, defined on $\lambda \in (0,\infty)$, is monotonically increasing and takes values in $(0,1)$. We observe that the final classification accuracy on unlabeled points depends

on the resulting value of $g(\lambda)$. A detailed ablation study on the relationship between classification accuracy and the MSE loss is provided in Section 4.3.2.

As a result, we propose to select a near-optimal λ by solving the equation $g(\lambda)=\mathcal{L}^*_{\text{MSE}}$, where $\mathcal{L}^*_{\text{MSE}}$ is a dataset-specific target MSE value. Since $g(\lambda)$ is monotonic, its root can be efficiently computed using, for example, the bisection method.

Although our derivation first obtains the closed-form solution $\mathbf{f}^*_{\mathcal{I}}$ of ridge regression and then substitutes it into the objective to define $g(\lambda)$, in practice this procedure can be viewed as a preprocessing step. Indeed, $\mathbf{f}^*_{\mathcal{I}}$ does not explicitly appear in the expression for $g(\lambda)$. In other words, we first determine λ by solving $g(\lambda)=\mathcal{L}^*_{\text{MSE}}$, and then use this value of λ to perform ridge regression and compute the solution $\mathbf{f}^*_{\mathcal{I}}$.

The final algorithm is provided in Algorithm 1. Notice that we use the iterative solver during inference because we need the predictions $u(x_i)$ for all samples, rather than only for training samples as in the training stage.

Algorithm 1. Interface Laplace Learning.

Input: $\mathbf{W}, \mathcal{L}^*_{\text{MSE}}, \mathbf{y}$, k-hop, iteration step T.

Preprocess:

1: $\mathcal{I}$ = get_interface_idx (train_idx, all_idx, $\mathbf{W}$, k).

2: Calculate $\mathbf{A}=\sum_{i=0}^{T-1}\mathbf{J}(\mathbf{D}^{-1}\mathbf{W}\mathbf{J})^i\mathbf{D}^{-1}$.

3: $\tilde{\mathbf{A}}_{\mathcal{I}}=\mathbf{A}[\text{train\_idx},\mathcal{I}]$.

4: Solve λ s.t. $g(\lambda)=(1/m)\|(\mathbf{I}+1/(m\lambda)\tilde{\mathbf{A}}_{\mathcal{I}}\tilde{\mathbf{A}}_{\mathcal{I}}^\top)^{-1}\mathbf{y}\|_F^2=\mathcal{L}^*_{\text{MSE}}$.

Training:

5: $\mathbf{f}^*_{\mathcal{I}}=\tilde{\mathbf{A}}_{\mathcal{I}}^\top(\tilde{\mathbf{A}}_{\mathcal{I}}\tilde{\mathbf{A}}_{\mathcal{I}}^\top+m\lambda\mathbf{I})^{-1}\mathbf{y}$.

6: $\mathbf{f}^*=\mathbf{f}^*_{\mathcal{I}}$, if $i\in\mathcal{I}$; $\mathbf{f}^*=0$, otherwise.

Inference:

7: $\mathbf{u}_0=\mathbf{0}$.

8: **for** $t=1,2,\ldots,T-1$ **do**

9: $\quad\mathbf{u}_{t+1}\leftarrow\mathbf{u}_t+\mathbf{D}^{-1}(\mathbf{f}^*-\mathbf{L}\mathbf{u}_t)$.

10: $\quad\mathbf{u}_{t+1}=\mathbf{u}_{t+1}-\overline{\mathbf{u}_{t+1}}$.

11: $\mathbf{u}=\mathbf{u}_T$.

Output:

12: $\ell(x_i)=\text{argmax}_{j\in\{1,\ldots,c\}}\{u_j(x_i)\}$.

4 Experiments

4.1 Classification at very low label rates

In this subsection, we conduct experiments to validate the effectiveness of our method under extreme label rates, with 1,2,3,4,5 labeled points per class, on the following real-world datasets: MNIST [30], FashionMNIST [41], and CIFAR-10 [28]. The MNIST and

FashionMNIST datasets each contain 70,000 images, while CIFAR-10 contains 60,000 images, all collected from 10 distinct classes. Rather than using raw images to build the similarity graph, we follow the approach of [12], which trains an autoencoder [27] to extract important features from the raw images. This generates a graph with higher quality for our method. The network architecture, loss function, and training procedure used for the autoencoder can be found in [12]. For fair comparison, we directly use the pre-computed features provided by [12] in our experiments.

After the features are extracted, we build a graph in the corresponding latent space. We use the Gaussian kernel (as defined in Eq. (2.1)) to compute the edge weights between nodes in the graph. The pre-processing procedures used to construct the graph are exactly the same as those described in [12]: We set $\mathbf{W} = (\mathbf{W} + \mathbf{W}^\top)/2$ for symmetry, and we set the diagonal entries of $\mathbf{W}$ to zero.

We compare our method with Laplace learning [48], random walk [44], weighted nonlocal Laplacian (WNLL) [35], multiclass MBO [23], centered kernel method [33], sparse label propagation [24], p-Laplace learning [21], Poisson learning [12] and variance-enlarged Poisson learning (V-Poisson) [47] in Table 1. A nearest neighbor classifier, which decided the label according to the closest labeled vertex with respect to the graph geodesic distance, is provided as a baseline. The results for all methods, excluding V-Poisson, are obtained using the GraphLearning Python package [11]. However, our implementation of the V-Poisson algorithm produces results that differ from the reported performance in [47]. We discuss this discrepancy further in Appendix B. In all experiments, we report the average accuracy and standard deviation across 100 random trials, with different labeled points selected each time. We test all methods on the same random permutations. Our method significantly outperforms others on various datasets. The code is publicly available at `https://github.com/shwangtangjun/Inter-Laplace`.

Time Complexity. The main computation burden is the calculation of the iteration matrix $\mathbf{A}$, which involves T matrix multiplications between a dense $m \times n$ matrix and a sparse $n \times n$ matrix. Matrix multiplication is a highly optimized operation on GPUs. It takes approximately 0.4 seconds to compute $\mathbf{A}$ on a single NVIDIA GeForce RTX 3090Ti GPU. As observed in Table 2, the time consumed by other parts of the algorithm is negligible in comparison.

Space complexity. The storage of the $m \times n$ dense matrix $\tilde{A}$ accounts for the majority of the memory requirements. On a single NVIDIA GeForce RTX 3090Ti GPU with 24 GB memory, our method can handle number of labeled samples m up to 10,000.

4.2 Ablation study on algorithm design

There are several components that contribute to the overall performance of our method, such as the MSE objective function, squared ℓ^2 regularization, interface positions approximation, and zero mean on each column. It is worthwhile to explore whether there exist superior alternatives to these components.

Table 1: Average accuracy scores over 100 trials with standard deviation on MNIST, FashionMNIST and CIFAR-10.

# Label Per Class		1	2	3	4	5
MNIST	Laplace [48]	16.73±7.41	28.04±10.04	42.98±12.18	54.90±12.79	66.94±12.06
	Nearest Neighbor	55.24±4.13	62.88±3.02	67.31±2.56	69.81±2.37	71.39±2.29
	Random Walk [44]	83.12±4.57	88.61±2.12	91.18±1.26	92.33±0.98	93.06±0.86
	MBO [23]	13.03±8.32	16.34±9.37	21.23±10.77	27.47±10.50	33.62±10.81
	WNLL [35]	55.32±13.61	84.86±5.89	91.34±2.80	93.68±1.57	94.60±1.17
	Centered Kernel [33]	20.43±2.18	25.94±3.05	30.73±3.52	34.63±4.13	37.84±4.07
	Sparse LP [24]	10.14±0.13	10.14±0.21	10.14±0.22	10.18±0.20	10.19±0.22
	p-Laplace [21]	65.93±4.89	75.72±2.83	80.54±1.99	83.02±1.71	84.54±1.56
	Poisson [12]	90.58±4.07	93.35±1.64	94.47±0.99	94.99±0.65	95.29±0.58
	V-Poisson [47]	90.68±4.89	93.98±1.80	94.88±0.94	95.20±0.66	95.38±0.56
	Inter-Laplace	**93.14**±3.82	**95.21**±1.02	**95.71**±0.64	**95.93**±0.47	**96.07**±0.40
FashionMNIST	Laplace [48]	18.77 ± 6.54	32.34 ± 8.98	43.44 ± 9.59	51.66 ± 7.50	57.38 ± 7.17
	Nearest Neighbor	43.98 ± 4.87	49.51 ± 3.19	53.00 ± 2.57	55.20 ± 2.37	56.95 ± 2.15
	Random Walk [44]	55.43 ± 4.97	62.01 ± 3.20	66.00 ± 2.61	67.93 ± 2.45	69.64 ± 2.07
	MBO [23]	11.27 ± 5.46	13.35 ± 6.24	15.76 ± 6.90	19.16 ± 7.85	22.63 ± 8.50
	WNLL [35]	45.31 ± 7.08	59.24 ± 4.27	65.61 ± 3.32	68.30 ± 2.75	70.35 ± 2.40
	Centered Kernel [33]	12.03 ± 0.37	13.35 ± 0.52	14.58 ± 0.81	15.95 ± 1.14	16.88 ± 1.05
	Sparse LP [24]	10.11 ± 0.18	10.17 ± 0.23	10.27 ± 0.25	10.26 ± 0.15	10.25 ± 0.17
	p-Laplace [21]	49.86 ± 5.13	56.91 ± 3.18	61.12 ± 2.48	63.46 ± 2.36	65.34 ± 2.03
	Poisson [12]	60.13 ± 4.85	66.57 ± 3.07	69.97 ± 2.50	71.37 ± 2.20	72.67 ± 1.96
	V-Poisson [47]	60.30 ± 5.64	66.70 ± 3.88	70.17 ± 2.97	71.44 ± 2.54	72.52 ± 2.14
	Inter-Laplace	**61.55** ± 5.08	**68.02** ± 3.45	**71.39** ± 2.65	**72.72** ± 2.34	**74.07** ± 1.92
CIFAR-10	Laplace [48]	10.50 ± 1.35	11.27 ± 2.43	11.55 ± 2.62	12.78 ± 3.81	13.88 ± 4.59
	Nearest Neighbor	30.06 ± 3.96	33.36 ± 2.95	35.21 ± 2.63	36.51 ± 2.32	37.70 ± 2.17
	Random Walk [44]	38.97 ± 4.94	45.55 ± 3.70	49.15 ± 3.47	51.75 ± 2.98	53.48 ± 2.29
	MBO [23]	11.07 ± 6.11	12.77 ± 6.76	14.01 ± 7.10	15.91 ± 7.15	17.43 ± 7.33
	WNLL [35]	17.67 ± 5.58	27.28 ± 7.06	34.98 ± 6.80	40.52 ± 5.93	44.78 ± 4.69
	Centered Kernel [33]	15.86 ± 1.81	17.71 ± 1.84	19.67 ± 2.15	21.53 ± 2.18	22.91 ± 2.42
	Sparse LP [24]	10.08 ± 0.10	10.07 ± 0.12	10.11 ± 0.22	10.03 ± 0.13	10.09 ± 0.15
	p-Laplace [21]	34.33 ± 4.65	40.33 ± 3.56	43.58 ± 3.10	45.99 ± 2.61	47.80 ± 2.17
	Poisson [12]	40.43 ± 5.48	46.63 ± 3.80	49.96 ± 3.84	52.39 ± 2.99	54.03 ± 2.35
	V-Poisson [47]	34.40 ± 4.85	36.73 ± 4.02	37.65 ± 3.60	38.41 ± 3.67	39.07 ± 3.65
	Inter-Laplace	**41.74** ± 6.11	**49.30** ± 4.13	**53.46** ± 3.75	**56.26** ± 3.25	**58.21** ± 2.48

Table 2: Wall time elapsed (seconds) for each stage, evaluated on MNIST with 1 label per class. A single NVIDIA 3090Ti GPU is used.

Preprocess				Training	Inference
get T	get_interface_idx()	get $\mathbf{A}$	get λ	get $\mathbf{f}^*$	get $\mathbf{u}$
0.05	0.0065	0.4	0.008	0.0004	0.06

4.2.1 Interface positions

In our algorithm, we remove k-hop neighbors of training samples to approximate the interface positions. To distinguish this method from others, we denote the resulting interface positions as $\mathcal{I}_{\text{khop}}$. To validate this choice, we test several other approaches:

(1) Ground truth. Similar to the method of getting interface positions in Section 2.2, we leak the label information and identify the indices of nonzero ground truth Laplacian values.

(2) Training. $\mathcal{I}_{\text{training}} = \{1,2,\ldots,m\}$.

(3) All. $\mathcal{I}_{\text{all}} = \{1,2,\ldots,n\}$.

(4) Random. For fair comparison, the size of random indices is chosen to be equal to $|\mathcal{I}_{\text{khop}}|$.

(5) Laplace-base. Firstly, we adopt Laplace learning [48] to get the prediction score $u(x_i)$ of each sample. This serves as a base method, not for direct classification, but for deciding the possible interface positions. We calculate the variance of each $u(x_i)$ and pick $|\mathcal{I}_{\text{khop}}|$ indices with the smallest variances. The base method is not limited to Laplace learning, as all classification methods can be possible options. We also test Poisson-base by adopting Poisson learning [12] as the base method.

(6) Geodesic. We define the geodesic distance between two vertices as the sum of edge weights along the shortest path connecting them. we calculate the geodesic distance of other nodes to the training nodes using Dijkstra's algorithm, and choose the farthest $|\mathcal{I}_{\text{khop}}|$ nodes.

We test each method on MNIST, FashionMNIST and CIFAR-10 with 1 labeled sample per class. The results are reported in Table 3. Let us discuss each option one by one: Surprisingly, the so-called "ground truth" option, which leaks label information, does not perform well in real-world classification tasks. This indicates that finding interface positions for high-dimensional multi-class classification is much more complex than for synthetic data. Using the training indices, which is similar to Poisson learning [12], indeed performs similarly to Poisson learning results in Table 1. The key difference is that

Table 3: Performance of different methods to approximate the interface positions with 1 label per class.

	MNIST	FashionMNIST	CIFAR-10
Remove k-hop	93.14 ± 3.82	61.55 ± 5.08	41.74 ± 6.11
Ground truth	90.33 ± 4.60	57.17 ± 5.19	38.45 ± 5.00
Training	90.53 ± 4.14	60.13 ± 4.87	40.55 ± 5.17
All	90.38 ± 4.34	59.96 ± 5.17	39.41 ± 5.34
Random	90.53 ± 4.46	59.89 ± 5.35	39.32 ± 5.43
Laplace-base	93.08 ± 3.54	60.99 ± 5.22	**41.84** ± 5.84
Poisson-base	**93.62** ± 3.47	**61.96** ± 5.88	41.69 ± 6.12
Geodesic	92.85 ± 3.87	61.34 ± 5.17	41.21 ± 6.18

Poisson learning uses a fixed source term, while our approach tries to learn the interface term. However, their performance is inferior to the k-hop removal method, suggesting that the given labeled points should be treated as the interior rather than the boundary. Treating all samples as the interface is even worse than learning on a random subset. This indicates that the solution $\mathbf{u}$, while exhibiting discontinuity on the interface, should still maintain a certain level of smoothness in the interior. The Laplace-base and Poisson-base methods are useful as they further improve the classification accuracy by about 0.4% on MNIST and FashionMNIST. This aligns with our intuition because the samples with the smallest prediction variance can be viewed as the hardest samples, and these are often located near the interface. However, in our proposed algorithm, we choose the k-hop removal approach because it is more efficient, easier to understand, and does not rely on other methods. The geodesic index approach, which uses a slightly more advanced version of graph distance, did not show improvements in the results. In conclusion, taking into account the implementation difficulty, classification performance, and execution efficiency, we choose to remove k-hop neighbors to approximate the interface positions.

4.2.2 MSE loss

In multi-class classification problems, cross-entropy loss (CE loss) is among one of the most popular choices of loss function. Specifically, the CE loss between the prediction $u(x_i)\in\mathbb{R}^c$ and integer-valued class label $y_i\in\mathbb{R}$ is defined as

$$\mathcal{L}_{\mathrm{CE}}=-\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp\big(u(x_i)_{y_i}\big)}{\sum_{j=1}^{c}\exp\big(u(x_i)_j\big)},$$

in which $u(x_i)_j$ denotes the j-th component of vector $u(x_i)\in\mathbb{R}^c$. The reason that we finally choose the MSE loss in our algorithm is two-fold:

- If we choose the CE loss, the optimization problem can no longer be formulated as a ridge regression problem and thus has no explicit solution. In this case, we use a gradient-based optimizer to minimize the loss function. Specifically, we adopt the L-BFGS method [31] with learning rate 0.1, initializing $\mathbf{f}_{\mathcal{I}}$ by sampling from $\mathcal{U}(-0.5,0.5)$. We find that 5 iterations are sufficient for convergence. Nonetheless, this approach requires approximately 0.4 seconds for training, which is about $1000\times$ slower than computing the explicit solution to the LS problem under the MSE loss.
- The results using the MSE loss are better than using the CE loss, as shown in Table 4. On MNIST and FashionMNIST datasets, the increase is marginal, but on CIFAR-10 we can see an increase of more than 1%.

4.2.3 Regularization norm

The squared ℓ^2 regularizer $\lambda|f|_2^2$ may be replaced by other regularization terms. Here, f denotes the vector obtained by flattening $\mathbf{f}$, for consistency of notation in this subsection.

Table 4: Comparison of mean squared error and cross-entropy loss performance.

# Label Per Class		1	2	3	4	5
MNIST	MSE	93.14 ± 3.82	95.21 ± 1.02	95.71 ± 0.64	95.93 ± 0.47	96.07 ± 0.40
	CE	93.01 ± 3.77	95.10 ± 1.13	95.67 ± 0.68	95.89 ± 0.50	96.00 ± 0.38
FashionMNIST	MSE	61.55 ± 5.08	68.02 ± 3.45	71.39 ± 2.65	72.72 ± 2.34	74.07 ± 1.92
	CE	61.12 ± 4.97	67.53 ± 3.54	71.17 ± 2.81	72.32 ± 2.53	73.89 ± 1.97
CIFAR-10	MSE	41.74 ± 6.11	49.30 ± 4.13	53.46 ± 3.75	56.26 ± 3.25	58.21 ± 2.48
	CE	40.43 ± 6.04	48.41 ± 4.31	52.20 ± 3.90	54.93 ± 3.23	57.20 ± 2.52

We test three alternatives: (1) ℓ^1 norm $|f|_1 = \sum |f_i|$; (2) unsquared ℓ^2 norm $|f|_2 = (\sum f_i^2)^{1/2}$; (3) ℓ^∞ norm $|f|_\infty = \max |f_i|$.

Similar to the squared ℓ^2 regularizer used in our method, these norms can also constrain the magnitude of the learned f. However, unlike ridge regression, optimization problems involving these norms generally do not admit explicit solutions. Moreover, since ℓ^p norms are non-differentiable at zero for $p \leq 1$ or $p = \infty$, gradient-based methods are not directly applicable to finding the corresponding minimizers.

To solve this issue, we use FISTA [4] (fast iterative shrinkage thresholding algorithm) to handle the optimization problem. Briefly, the iterative shrinkage thresholding algorithm (ISTA) performs a gradient descent step on the non-regularized objective, followed by a shrinkage or projection step depending on the regularizer. Specifically, the ℓ^1 norm corresponds to soft-thresholding, while the unsquared ℓ^2 and ℓ^∞ norms correspond to projections onto an ℓ^2 ball and an ℓ^1 ball, respectively. FISTA improves upon ISTA by introducing Nesterov acceleration to speed up convergence.

We report the performance of different regularization norms on the MNIST dataset with one label per class in Table 5. It can be observed that the ℓ^1 norm performs significantly worse than the others, while the unsquared ℓ^2 and ℓ^∞ norms yield comparable results to the squared ℓ^2 norm. However, due to the computational cost of the FISTA algorithm, which requires several seconds to converge (compared to an average of 0.0004 seconds for computing the closed-form solution with the squared ℓ^2 regularizer), we adopt the latter in our method. Nonetheless, exploring the effect of different regularization norms remains an interesting direction for future work.

Table 5: Comparison of different regularization norm on MNIST, 1 label per class.

$\|f\|_2^2$	$\|f\|_1$	$\|f\|_2$	$\|f\|_\infty$
93.14 ± 3.82	89.93 ± 5.14	93.11 ± 3.69	92.72 ± 3.85

4.2.4 Zero mean

In our algorithm, we add a mean subtraction step to ensure the uniqueness of solution, since the graph Laplacian matrix $\mathbf{L}$ is singular and admits infinite solutions. We manually

choose the solution with zero mean along each component. To show that the effectiveness of our method does not come solely from extracting the mean, we report the results of our algorithm with and without mean subtraction step in Table 6. It is observed that the mean subtraction step helps increase the performance by less than 0.5% on average. Notably, when we apply the same mean subtraction step to Poisson learning [12], the performance of Poisson learning with and without this step shows very little difference, with less than 0.05% difference.

Table 6: Comparison of with (w/) and without (w/o) zero mean technique.

# Label per class		1	2	3	4	5
MNIST	w/	93.14 ± 3.82	95.21 ± 1.02	95.71 ± 0.64	95.93 ± 0.47	96.07 ± 0.40
	w/o	92.93 ± 3.99	95.08 ± 1.15	95.62 ± 0.74	95.88 ± 0.49	95.99 ± 0.44
FashionMNIST	w/	61.55 ± 5.08	68.02 ± 3.45	71.39 ± 2.65	72.72 ± 2.34	74.07 ± 1.92
	w/o	61.08 ± 4.94	67.54 ± 3.44	71.06 ± 2.78	72.26 ± 2.44	73.62 ± 1.94
CIFAR-10	w/	41.74 ± 6.11	49.30 ± 4.13	53.46 ± 3.75	56.26 ± 3.25	58.21 ± 2.48
	w/o	41.25 ± 6.14	49.04 ± 4.20	53.08 ± 3.70	55.95 ± 3.20	57.94 ± 2.42

4.3 Ablation study on parameters

There are two parameters in our experiment: k in k-hop and the target MSE loss $\mathcal{L}^*_{\text{MSE}}$. In Table 1, we report the best results obtained through a grid search over the model parameters. The optimal k is provided in Table 7. We choose $\mathcal{L}^*_{\text{MSE}} = 0.20$ for MNIST dataset, and $\mathcal{L}^*_{\text{MSE}} = 0.35$ for FashionMNIST and CIFAR-10.

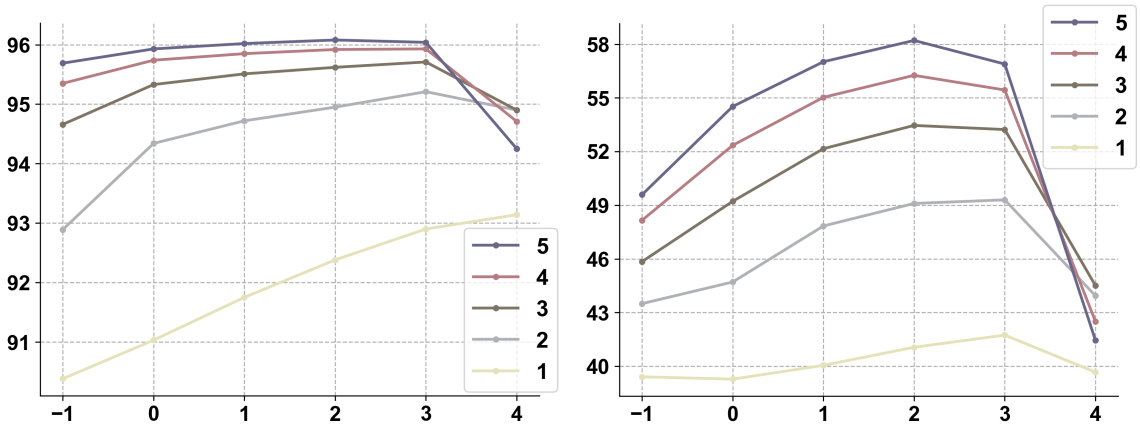
Table 7: Parameters k used to reproduce the results in Table 1.

# Label per class	1	2	3	4	5
MNIST	4	3	3	3	2
FashionMNIST	5	4	4	3	3
CIFAR-10	3	3	2	2	2

In this subsection, we will study the effect of these two parameters.

4.3.1 k-hop

We conduct experiments on MNIST and CIFAR-10, reporting the classification accuracy when k ranges from -1 to 4, and the number of labeled samples per class ranges from 1 to 5. Here, $k = -1$ means that all index are treated as the interface. For each value of k, we report the best performance with respect to the ridge regularization parameter λ. The results are presented in Fig. 3.



(a) MNIST (b) CIFAR-10

Figure 3: Ablation study on k-hop parameter. x-axis: k-hop. y-axis: accuracy (%). Each line corresponds to a different number of labeled samples per class.

From the results, it is evident that removing k-hop indices is crucial for classification accuracy. The best accuracy at different label rates significantly outperforms the corresponding accuracy when $k=-1$. Moreover, we have more observations on the choice of k.

- When k is too large, the performance collapses. This is because the remaining positions for training the interface term are too small. For example, on the CIFAR-10 dataset, when the labeled number per class is 1 and $k=4$, there are only 2730/60000 points beyond the k-hop neighborhood. When the labeled number per class is 5 and $k=4$, there are only 44/60000 points beyond the k-hop neighborhood. With such a small number of trainable parameters, it may be too challenging for the algorithm to learn a good interface term that generalizes well.
- On the same dataset, the optimal k decreases as the number of labeled samples per class increases. On MNIST, when there is 1 label per class, the optimal $k=4$. When we increase the label number to 4, $k=3$ gives the best result. This phenomenon meets our expectation, as we want to keep the number of remaining indices moderate. The former setting gives 38018/70000 remaining indices, while the latter gives 42059/70000 remaining indices, which is comparable.
- For different datasets with the same number of labeled samples per class, the optimal k is different. This is because the connectivity of the datasets varies. Although the similarity graph W is constructed with the same number of nearest neighbors $K=10$, the underlying data distributions for different datasets are disparate. Hence, even with the same k, the remaining number of nonzero indices after removing the k-hop indices can be very different. For example, when the label number per class is 1 and $k=4$, there are 38018/70000 points left for MNIST, but only 2730/60000

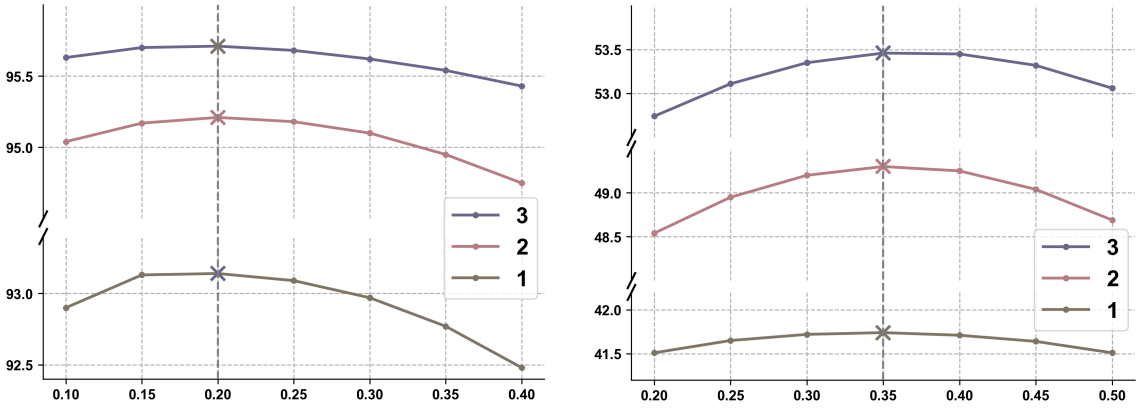
points left for CIFAR-10. Such distinctions definitely lead to the different optimal k values for different datasets: the better connectivity of CIFAR-10 suggests that optimal k is smaller.

4.3.2 $\mathcal{L}^*_{\mathbf{MSE}}$

Another important parameter is the target MSE loss $\mathcal{L}^*_{\text{MSE}}$, which guides the selection of the ridge parameter λ. In Fig. 4, we report the classification accuracy as a function of $\mathcal{L}^*_{\text{MSE}}$ under different numbers of labeled samples per class.

The results show that $\mathcal{L}^*_{\text{MSE}}$ is a dataset-specific parameter, in the sense that: (1) for each dataset, the optimal $\mathcal{L}^*_{\text{MSE}}$ yielding the best accuracy remains roughly the same across different label rates; (2) across different datasets, the optimal $\mathcal{L}^*_{\text{MSE}}$ varies – for example, approximately 0.20 for MNIST and 0.35 for CIFAR-10.

It is also noteworthy that the performance is fairly stable with respect to $\mathcal{L}^*_{\text{MSE}}$. For instance, on the CIFAR-10 dataset with one labeled sample per class, the accuracy remains above 41.5% when $\mathcal{L}^*_{\text{MSE}}$ ranges from 0.20 to 0.50. Considering that the range of $\mathcal{L}^*_{\text{MSE}}$ is $(0,1)$, we conclude that the accuracy is robust to the choice of $\mathcal{L}^*_{\text{MSE}}$.



(a) MNIST (b) CIFAR-10

Figure 4: Ablation study on $\mathcal{L}^*_{\text{MSE}}$. x-axis: $\mathcal{L}^*_{\text{MSE}}$. y-axis: accuracy(%). Each line corresponds to a different number of labeled samples per class.

4.4 Broader application scenarios

In this section, we extend the application of our method to more scenarios, including cases with unbalanced label distribution and higher label rates per class.

4.4.1 Unbalanced label distribution

To account for the issue of unbalanced label distribution, we consider the following setup: for the even-numbered classes $\{0,2,4,6,8\}$, we use 1 labeled sample per class, while for the odd-numbered classes $\{1,3,5,7,9\}$, we use 5 labeled samples per class.

Other methods, such as Poisson learning [12], deal with unbalanced training data in a post-processing way by introducing a factor $s_j = b_j/n_j$ to the prediction $u(x_i)$

$$\underset{j\in\{1,\dots,c\}}{\operatorname{argmax}}\{s_j u_j(x_i)\},$$

where b_j is the true proportion of class-j samples in the dataset, and n_j is the proportion of labeled points from class j among all labeled samples. On the contrary, our method naturally learns a more suitable interface term from the unbalanced distribution during the training stage. We adopt the same weighting factor s_j, and apply it as the weight for the MSE loss in our objective function,

$$\frac{1}{m}\sum_{i=1}^{m} s_{y_i}\|(A\mathbf{f})_i - y_i\|_2^2 + \lambda\sum_{i=1}^{n}\|f_i\|_2^2.$$

Here, y_i in the expression s_{y_i} refers to the integer-valued class label, not a one-hot encoded vector. We slightly abuse the notation for the sake of simplicity. This weighted loss approach is a widely applied technique when designing loss functions for handling unbalanced training data. Similar to Section 3.3, we can derive the explicit solution of the above objective function

$$\mathbf{f}_{\mathcal{I}}^{*} = \tilde{\mathbf{A}}_{\mathcal{I}}^{\top}\left(\tilde{\mathbf{A}}_{\mathcal{I}}\tilde{\mathbf{A}}_{\mathcal{I}}^{\top} + m\lambda\mathbf{S}^{-1}\right)^{-1}\mathbf{y},$$

where $\mathbf{S} = \operatorname{diag}(s_{y_i}) \in \mathbb{R}^{m\times m}$. The inference procedure remains unchanged from the previous algorithm, without the need for any additional post-processing steps.

The results are presented in Table 8. We compare our method to Poisson learning, as it provides an approach to handle unbalanced training data. As a sanity check, both methods outperform their counterparts where only 1 label per class is provided, indicating that the additional labeled samples for the odd-numbered classes are indeed helpful. Importantly, our method outperforms Poisson learning across all the datasets. This advantage stems from the fact that we incorporate the class imbalance directly into the objective function, allowing the interface term to learn from the unbalanced distribution, rather than relying on manual post-processing adjustments as in Poisson learning.

Table 8: Performance of unbalanced label distribution: even classes 1 label per class, odd classes 5 labels per class.

	MNIST	FashionMNIST	CIFAR-10
Poisson [12]	93.88 ± 2.35	66.47 ± 3.80	46.87 ± 4.16
Inter-Laplace	95.26 ± 2.07	68.16 ± 3.66	49.30 ± 4.72

4.4.2 Higher label rate

Although the motivation of interface La-place learning is to get accurate classification with low label rate, this approach remains effective in higher label rate scenarios. In experiments with 100 labeled samples per class, presented in Table 9, our method outperforms other approaches across the datasets tested. While the gains are modest on

Table 9: Performance of higher label rate: 100 labels per class.

	MNIST	FashionMNIST	CIFAR-10
Laplace [48]	96.83 ± 0.10	81.48 ± 0.39	63.88 ± 1.22
Nearest Neighbor	85.58 ± 0.44	71.06 ± 0.55	49.32 ± 0.49
Random Walk [44]	96.63 ± 0.11	81.37 ± 0.30	67.47 ± 0.51
MBO [23]	96.88 ± 0.18	74.72 ± 0.91	42.40 ± 0.78
WNLL [35]	96.25 ± 0.11	81.04 ± 0.30	67.04 ± 0.50
Centered Kernel [33]	88.66 ± 0.72	57.52 ± 2.24	57.30 ± 1.43
Sparse LP [24]	37.76 ± 1.12	28.12 ± 0.66	8.90 ± 0.22
p-Laplace [21]	93.54 ± 0.20	78.10 ± 0.39	63.46 ± 0.38
Poisson [12]	96.77 ± 0.09	80.41 ± 0.62	66.09 ± 0.57
V-Poisson [47]	95.89 ± 0.09	77.85 ± 0.52	41.67 ± 2.17
Inter-Laplace	$\mathbf{97.25} \pm 0.09$	$\mathbf{82.72} \pm 0.28$	$\mathbf{73.28} \pm 0.34$

simpler datasets like MNIST and FashionMNIST, the improvements become significant on more challenging dataset CIFAR-10. This result demonstrates the robustness of our approach in leveraging available labeled data effectively, regardless of the per-class label rate.

5 Conclusion

Inspired by the observation that interfaces exist between different classes, we propose a Laplace equation model with jump discontinuity and derive its nonlocal counterpart. Based on the nonlocal model, we introduce an interface term to enhance Laplace learning. We then design an effective algorithm to approximate the interface positions and learn the interface term. Experimental results verify that our method can accurately describe the data distribution and improve the performance of semi-supervised learning tasks. Our interface Laplace learning framework is general and may be extended to several important research areas such as node classification on graph-structured data and few-shot learning. Future work involves incorporating the interface concept into neural network architectures, such as graph neural networks, and exploring alternative approaches to approximate the interface position.

A Other tries

A.1 Dirichlet-type interface term

From the perspective of nonlocal models, we introduce the interface term f_i through the graph Laplacian, i.e. $Lu(x_i) = f_i$. However, as Laplace learning [48] uses a Dirichlet-type boundary condition to incorporate the label information, it is natural to test whether our

interface term can also be incorporated in a Dirichlet style. Specifically, we formulate the Dirichlet-type interface problem as the following:

$$\begin{aligned} Lu(x_i) &= 0, \quad i \notin \mathcal{I}, \\ u(x_i) &= f_i, \quad i \in \mathcal{I}. \end{aligned}$$

Similar to Subsection 3.3, we want to write $\mathbf{u}$ in terms of $\mathbf{f}$ in the form $\mathbf{u} = \mathbf{A}\mathbf{f}$. Denote $\mathcal{K}$ as the set of interior indices $i \notin \mathcal{I}$. $\mathbf{L}_{\mathcal{K}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{K}|}$ extracts the corresponding rows and columns from the graph Laplacian matrix $\mathbf{L}$. $\mathbf{W}_{\mathcal{K},\mathcal{I}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{I}|}$ extracts $\mathcal{K}$ rows and $\mathcal{I}$ columns from the similarity matrix $\mathbf{W}$. Notice that unlike singular matrix $\mathbf{L}$, $\mathbf{L}_{\mathcal{K}}$ is invertible. Thus, we can write the prediction $\mathbf{u}_{\mathcal{K}}$ on $i \in \mathcal{K}$ as

$$\mathbf{u}_{\mathcal{K}} = \mathbf{L}_{\mathcal{K}}^{-1}\mathbf{W}_{\mathcal{K},\mathcal{I}}\mathbf{f}_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{K}|\times c}.$$

The matrix inversion of $\mathbf{L}_{\mathcal{K}}$ is computationally expensive. However, as we only need the m rows corresponding to the labeled indices in $\mathbf{u}_{\mathcal{K}}$ during training, we can instead solve the following equation:

$$\tilde{\mathbf{L}}_{\mathcal{K}}^{-1} \cdot \mathbf{L}_{\mathcal{K}} = \tilde{\mathbf{I}}$$

to calculate only the m corresponding rows of $\mathbf{L}_{\mathcal{K}}^{-1}$. Here $\tilde{\mathbf{L}}_{\mathcal{K}}^{-1}$ and $\tilde{\mathbf{I}}$ indicates the rows that correspond to the m training indices of $\mathbf{L}_{\mathcal{K}}^{-1}$ and identity matrix $\mathbf{I}$, respectively. Finally, we can write

$$\tilde{\mathbf{A}}_{\mathcal{I}} = \tilde{\mathbf{L}}_{\mathcal{K}}^{-1}\mathbf{W}_{\mathcal{K},\mathcal{I}}.$$

We can then learn the interface term $\mathbf{f}^*$ in the same way as in Section 3.3. The algorithm with Dirichlet-type interface term is denoted as Inter-Laplace-D, and we compare its performance with Laplace learning, Poisson learning and our Inter-Laplace in Table 10.

The proposed Inter-Laplace-D method can significantly improve performance compared to Laplace learning. This further substantiate our viewpoint that the function u should exhibit discontinuities at category interfaces, while remaining smooth within their interiors.

However, the performance of Inter-Laplace-D does not quite match the results of Poisson learning and our Inter-Laplace method. The key difference is that Inter-Laplace imposes the interface term through $Lu(x_i) = f_i$, while Inter-Laplace-D imposes it through

Table 10: Performance of Dirichlet-type interface term with 1 label per class.

	MNIST	FashionMNIST	CIFAR-10
Laplace [48]	16.73 ± 7.41	18.77 ± 6.54	10.50 ± 1.35
Poisson [12]	90.58 ± 4.07	60.13 ± 4.85	40.43 ± 5.48
Inter-Laplace-D	84.68 ± 4.06	59.18 ± 5.23	37.19 ± 4.44
Inter-Laplace	**93.14** ± 3.82	**61.55** ± 5.08	**41.74** ± 6.11

$u(x_i)=f_i$. We argue that the $Lu(x_i)=f_i$ constraint in Inter-Laplace is a more theoretically-principled approach, as it is derived from nonlocal models. Additionally, this formulation may be a smoother way to incorporate the required discontinuities. By restricting the second derivative $Lu(x_i)$, rather than just the function value $u(x_i)$, Inter-Laplace can more effectively capture the desired discontinuities at category boundaries. This distinction in interface term formulation may help explain why Poisson learning outperforms Laplace learning by a significant margin.

A.2 Neural network parametrized interface term

In our algorithm, we have directly treated the interface term f_i as trainable parameters. However, an alternative approach could be to model f_i as the output of a neural network. Specifically, we could construct a neural network with the extracted feature x_i as input, and the network output $f_\theta(x_i)$ representing the interface term, where f_θ is the neural network with parameters θ. This neural network-based formulation means that an explicit solution is no longer possible, as neural networks are inherently non-convex. Nonetheless, we can choose to optimize the same objective function as before

$$\operatorname*{argmin}_\theta \frac{1}{m}\sum_{i=1}^{m}\|(A\mathbf{f}_\theta)_i-y_i\|_2^2+\lambda\sum_{i=1}^{n}\|f_\theta(x_i)\|_2^2,$$

where $\mathbf{f}_\theta=[f_\theta(x_1),\cdots,f_\theta(x_n)]^\top$. One may view the addition of neural networks as a way to incorporate feature information, since now the interface term $f_\theta(x_i)$ is dependent on the extracted feature x_i. This dependence on the input features can provide additional information to the learned interface term.

We conduct experiments on the MNIST dataset using 1 label per class. We construct a simple 2-layer multi-layer perceptron (MLP) with a hidden dimension of 64 and ReLU activation function to parametrize f_θ. The model is t rained for 1000 epochs using the Adam optimizer with a learning rate of 0.01. Without incorporating feature information, our method achieves an average accuracy of $93.14\pm3.82\%$. However, when using a neural network approach, the performance drops to $92.19\pm3.93\%$.

One possible explanation for this performance difference is that the feature information may already be sufficiently captured in the construction of the similarity matrix W. Incorporating the same information again through the neural network parametrization of f_θ could impose unnecessary restrictions on the learning process. Additionally, using a neural network introduces more hyper-parameters, such as the hidden dimension, choice of optimizer, learning rate, and network structure, which may need to be carefully tuned to achieve optimal performance.

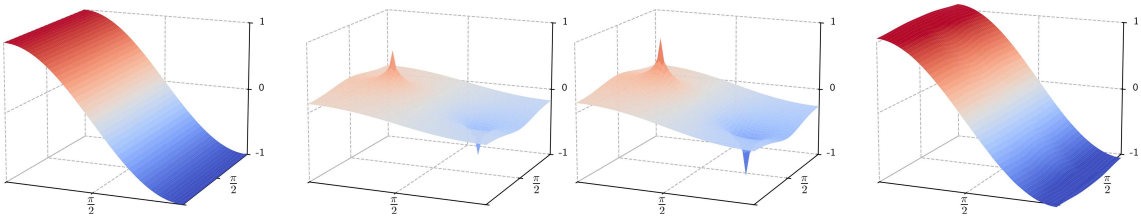
Given the slightly worse performance observed when using a neural network to parametrize the interface term f, we opt not to use a neural network in our approach. However, we hypothesize that feature information might be helpful in graph node classification tasks, such as the Cora dataset [42]. In these tasks, the graph edge information

is often considered orthogonal to the graph node information. Incorporating both the graph structure and the node features may lead to improved performance. We plan to explore this direction as part of our future work.

A.3 Synthetic regression

In Section 2.2, we provide a toy classification example. It is noteworthy that our approach can also be applied to regression problems naturally by setting scalar values as the labels. In this subsection, we provide a comparison of regression results for Laplace learning, Poisson learning and our method on synthetic data.

We uniformly sample a regular 50×50 grid in $(x,y)\in[0,\pi]\times[0,\pi]$. The target is set as $\cos(x)$. We manually pick two labeled points, $(\pi/4,\pi/2)$ and $(3\pi/4,\pi/2)$. The similarity matrix is constructed as same as Eq. (2.1). For Poisson learning and our method, we use $T=10{,}000$ to ensure convergence. We pick $\mathrm{k}=5$ and $\lambda=0.05$ in our method. The regression results and corresponding MSE are presented in Fig. 5. It is evident from the results that our method significantly outperforms the Laplace learning and Poisson learning approaches. While this particular regression problem does not exhibit a clear interface between distinct classes, it is still beneficial to assume that the underlying function is not harmonic almost everywhere.



(a) Ground truth (b) Laplace. MSE=0.3118 (c) Poisson. MSE=0.2598 (d) Ours. MSE=0.0028

Figure 5: A synthetic regression example.

B V-Poisson reproducibility issue

For the method Variance-enlarged Poisson learning (V-Poisson) proposed in [47], we can only find a zip file on OpenReview, containing only partially reproducible code. The provided Github repository in the paper is empty. So we reproduce the results of V-Poisson by ourselves.

- We use a different MNIST distance matrix from that used in V-Poisson. The authors of Poisson learning [12] provide a more fine-tuned distance matrix of MNIST in their GitHub repository two years after the original paper was published. However, since the exact training procedure for this updated distance matrix is not given, we decide to use the original distance matrix for experiments in the main

Table 11: Average accuracy scores over 100 trials with standard deviation on new MNIST.

# Label per class	1	2	3	4	5
Laplace [48]	17.74 ± 8.80	32.20 ± 12.23	49.47 ± 15.12	66.23 ± 12.80	76.45 ± 10.50
Nearest Neighbor	57.50 ± 4.53	65.87 ± 3.22	70.49 ± 2.51	73.24 ± 2.43	74.98 ± 2.26
Random Walk [44]	85.00 ± 4.28	90.21 ± 2.13	92.53 ± 1.46	93.56 ± 1.23	94.24 ± 0.95
MBO [23]	13.28 ± 8.66	17.10 ± 9.21	22.19 ± 10.77	28.01 ± 10.37	34.14 ± 11.67
WNLL [35]	66.12 ± 14.03	90.51 ± 4.45	94.66 ± 1.54	95.77 ± 0.89	96.20 ± 0.50
Centered Kernel [33]	19.52 ± 1.77	24.94 ± 2.70	29.86 ± 3.06	33.49 ± 3.22	36.72 ± 3.83
Sparse LP [24]	10.02 ± 0.14	9.97 ± 0.22	10.00 ± 0.14	9.94 ± 0.14	9.79 ± 0.13
p-Laplace [21]	69.04 ± 4.79	78.56 ± 2.94	83.16 ± 2.25	85.58 ± 2.04	87.07 ± 1.80
Poisson [12]	93.11 ± 3.87	95.20 ± 1.40	95.93 ± 0.71	96.22 ± 0.57	96.42 ± 0.35
V-Poisson [47]	93.27 ± 4.35	95.49 ± 1.61	96.18 ± 0.48	96.31 ± 0.37	96.43 ± 0.38
Ours	$\mathbf{94.91} \pm 3.83$	$\mathbf{96.34} \pm 1.20$	$\mathbf{96.72} \pm 0.46$	$\mathbf{96.80} \pm 0.50$	$\mathbf{96.94} \pm 0.36$

paper. Nonetheless, we also provide the performance comparison on this updated "new MNIST" distance matrix in Table 11. Our method still outperforms other approaches using this updated distance matrix.

- The code for CIFAR-10 is missing in the OpenReview zip file mentioned earlier. The results reproduced by ourselves are not as good as those reported in [47]. Missing of the original code makes it difficult to check the correctness of our reproduced results.

To clarify, we use the same off-the-shelf distance matrix for all methods in our experiments for a fair comparison .

C Experiments on self-supervised learned representations on CIFAR-10

Recent self-supervised learning techniques based on contrastive learning have achieved impressive performance in learning representations, even in fully unlabeled settings. However, our focus lies in the extremely low-label regime, where fewer than 5 labeled samples per class are available. In such scenarios, deep self-supervised learning methods that rely on pretrained embeddings often struggle when directly applied to classification tasks. To illustrate this point, we conduct the following experiment using SimSiam [15], a representative self-supervised model: (1) train a linear classifier directly on SimSiam features using 1 labeled sample per class (2) construct a similarity matrix $\mathbf{W}$ from SimSiam features and apply our Inter-Laplace algorithm.

As shown in Table 12, while SimSiam produces strong representations, the linear classifier performs poorly under scarce supervision. In contrast, our graph-based method

Table 12: Classification accuracy (%) on CIFAR-10 with 1 labeled point per class, using SimSiam [15] feature. Mean accuracy and std among 100 random trails.

Linear classifier	Inter-Laplace
37.73 ± 4.98	73.78 ± 7.31

significantly improves classification by effectively propagating the limited label information through the data graph. In one word, self-supervised models can provide good representations, but struggle to learn effective classifiers under extremly low label rate. Therefore, rather than positioning our method as a competitor to deep self-supervised learning techniques, we view it as complementary:

- When strong representations are available (e.g. via contrastive learning), they can be used to construct better similarity graphs, thereby improving label propagation under our framework.
- Our method is orthogonal to how features are obtained – it can operate on raw distances, pretrained embeddings, or explicitly given graph structures.

Acknowledgements

This work was supported by National Natural Science Foundation of China (NSFC) (Grant No. 92370125).

References

[1] M. Alfaro and J. Coville, *Propagation phenomena in monostable integro-differential equations: Acceleration or not?*, J. Differential Equations, 263:5727–5758, 2017.

[2] R. K. Ando and T. Zhang, *Learning on graph with Laplacian regularization*, in: Proceedings of the 20th International Conference on Neural Information Processing Systems, MIT Press, 25–32, 2007.

[3] Z. P. Bažant and M. Jirásek, *Nonlocal integral formulations of plasticity and damage: Survey of progress*, in: Perspectives in Civil Engineering: Commemorating the 150th Anniversary of the American Society of Civil Engineers, ASCE, 21–52, 2003.

[4] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM J. Imaging Sci., 2:183–202, 2009.

[5] M. Belkin, I. Matveeva, and P. Niyogi, *Regularization and semi-supervised learning on large graphs*, in: Learning Theory. COLT 2004. Lecture Notes in Computer Science, Vol. 3120, Springer, 624–638, 2004.

[6] M. Belkin and P. Niyogi, *Using manifold stucture for partially labeled classification*, in: Advances in Neural Information Processing Systems 15, MIT Press, 953–960, 2002.

[7] S. Blandin and P. Goatin, *Well-posedness of a conservation law with non-local flux arising in traffic flow modeling*, Numer. Math., 132:217–241, 2016.

[8] A. Blum and T. Mitchell, *Combining labeled and unlabeled data with co-training*, in: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, ACM, 92–100, 1998.
[9] N. Bridle and X. Zhu, *p-voltages: Laplacian regularization for semi-supervised learning on high-dimensional data*, in: Eleventh Workshop on Mining and Learning with Graphs (MLG2013), ACM, 2013.
[10] T. Bühler and M. Hein, *Spectral clustering based on the graph p-Laplacian*, in: Proceedings of the 26th Annual International Conference on Machine Learning, PLMR, 81–88, 2009.
[11] J. Calder, *GraphLearning Python package*, 2022. `https://doi.org/10.5281/zenodo.5850940`
[12] J. Calder, B. Cook, M. Thorpe, and D. Slepcev, *Poisson learning: Graph based semi-supervised learning at very low label rates*, in: Proceedings of the 37th International Conference on Machine Learning, PMLR, 119:1306–1316, 2020.
[13] J. Calder and D. Slepčev, *Properly-weighted graph Laplacian for semi-supervised learning*, Appl. Math. Optim., 82:1111–1159, 2020.
[14] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-supervised learning*, IEEE Trans. Neural Netw., 20:542–542, 2009.
[15] X. Chen and K. He, *Exploring simple Siamese representation learning*, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, 15750–15758, 2021.
[16] P. Constantin, *Some open problems and research directions in the mathematical study of fluid dynamics*, in: Mathematics Unlimited – 2001 and Beyond, Springer, 353–360, 2001.
[17] K. Dayal and K. Bhattacharya, *A real-space non-local phase-field model of ferroelectric domain patterns in complex geometries*, Acta Mater., 55:1907–1917, 2007.
[18] Q. Du, *Nonlocal Modeling, Analysis, and Computation*, SIAM, 2019.
[19] A. El Alaoui, X. Cheng, A. Ramdas, M. J. Wainwright, and M. I. Jordan, *Asymptotic behavior of ℓ_p-based Laplacian regularization in semi-supervised learning*, in: Conference on Learning Theory, PMLR, 879–906, 2016.
[20] H. Emmerich, *The Diffuse Interface Approach in Materials Science: Thermodynamic Concepts and Applications of Phase-Field Models*, in: Lecture Notes in Physics Monographs, Vol. 73, Springer Science & Business Media, 2003.
[21] M. Flores, J. Calder, and G. Lerman, *Algorithms for ℓ_p-based semi-supervised learning on graphs*, arXiv:1901.05031, 2019.
[22] S. Fralick, *Learning to recognize patterns without a teacher*, IEEE Trans. Inf. Theory, 13:57–64, 1967.
[23] C. Garcia-Cardona, E. Merkurjev, A. L. Bertozzi, A. Flenner, and A. G. Percus, *Multiclass data segmentation using diffuse interface methods on graphs*, IEEE Trans. Pattern Anal. Mach. Intell., 36:1600–1613, 2014.
[24] A. Jung, A. O. Hero III, A. Mara, and S. Jahromi, *Semi-supervised learning via sparse label propagation*, arXiv:1612.01414, 2016.
[25] F. Kang, R. Jin, and R. Sukthankar, *Correlated label propagation with application to multi-label learning*, in: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 2, IEEE, 1719–1726, 2006.
[26] C.-Y. Kao, Y. Lou, and W. Shen, *Random dispersal vs. non-local dispersal*, Discrete Contin. Dyn. Syst., 26:551–596, 2010.
[27] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv:1312.6114, 2022.
[28] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, Technical Report TR-2009, University of Toronto, 2009.
[29] R. Kyng, A. Rao, S. Sachdeva, and D. A. Spielman, *Algorithms for Lipschitz learning on graphs*,

in: Conference on Learning Theory, PMLR, 1190–1223, 2015.
[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE, 86:2278–2324, 1998.
[31] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Program., 45:503–528, 1989.
[32] A. Mahan, *Reflection and refraction at oblique incidence on a dielectric-metallic interface as a boundary value problem in electromagnetic theory*, J. Opt. Soc. Am., 46(11):913–926, 1956.
[33] X. Mai, *A random matrix analysis and improvement of semi-supervised learning for large dimensional data*, J. Mach. Learn. Res., 19:1–27, 2018.
[34] B. Nadler, N. Srebro, and X. Zhou, *Semi-supervised learning with the graph Laplacian: The limit of infinite unlabelled data*, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., 22:1330–1338, 2009.
[35] Z. Shi, S. Osher, and W. Zhu, *Weighted nonlocal Laplacian on interpolation from sparse data*, J. Sci. Comput., 73:1164–1177, 2017.
[36] W. Shyy and R. Narayanan, *Fluid Dynamics at Interfaces*, Cambridge University Press, 1999.
[37] D. Slepcev and M. Thorpe, *Analysis of p-Laplacian regularization in semisupervised learning*, SIAM J. Math. Anal., 51:2085–2120, 2019.
[38] Z. Song, X. Yang, Z. Xu, and I. King, *Graph-based semi-supervised learning: A comprehensive review*, IEEE Trans. Neural. Netw. Learn. Syst., 34:8174–8194, 2022.
[39] E. Stephan, *Solution procedures for interface problems in acoustics and electromagnetics*, in: Theoretical Acoustics and Numerical Techniques. International Centre for Mechanical Sciences, Vol. 277, Springer, 291–348, 1983.
[40] J. L. Vázquez, *Nonlinear diffusion with fractional Laplacian operators*, in: Nonlinear Partial Differential Equations. Abel Symposia, Vol. 7, Springer, 271–298, 2012.
[41] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*, arXiv:1708.07747, 2017.
[42] Z. Yang, W. Cohen, and R. Salakhudinov, *Revisiting semi-supervised learning with graph embeddings*, in: International Conference on Machine Learning, PMLR, 40–48, 2016.
[43] Y. Zhang and Z. Shi, *A nonlocal model of elliptic equation with jump coefficients on manifold*, Commun. Math. Sci., 19:1881–1912, 2021.
[44] D. Zhou and B. Schölkopf, *Learning from labeled and unlabeled data using random walks*, in: Pattern Recognition. DAGM 2004, Lecture Notes in Computer Science, Vol. 3175, Springer, 237–244, 2004.
[45] D. Zhou and B. Schölkopf, *Regularization on discrete spaces*, in: Pattern Recognition, DAGM 2005, Lecture Notes in Computer Science, Vol. 3663, Springer, 361–368, 2005.
[46] X. Zhou and M. Belkin, *Semi-supervised learning by higher order regularization*, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR, 15:892–900, 2011.
[47] X. Zhou, X. Liu, H. Yu, J. Wang, Z. Xie, J. Jiang, and X. Ji, *Variance-enlarged Poisson learning for graph-based semi-supervised learning with extremely sparse labeled data*, in: The Twelfth International Conference on Learning Representations, ICLR, 1310–1328, 2024.
[48] X. Zhu, Z. Ghahramani, and J. D. Lafferty, *Semi-supervised learning using Gaussian fields and harmonic functions*, in: Proceedings of the 20th International Conference on Machine Learning (ICML-03), AAAI Press, 912–919, 2003.
[49] X. Zhu and A. B. Goldberg, *Introduction to Semi-Supervised Learning*, in: Synthesis Lectures on Artificial Intelligence and Machine Learning, Springer, 2009.