Beltrami Flow and Neural Diffusion on Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose a novel class of graph neural networks based on the discretized *Beltrami flow*, a non-Euclidean diffusion PDE. In our model, node features are supplemented with positional encodings derived from the graph topology and jointly evolved by the Beltrami flow, producing simultaneously continuous feature learning, topology evolution. The resulting model generalizes many popular graph neural networks and achieves state-of-the-art results on several benchmarks.

1 Introduction

The majority of graph neural networks (GNNs) are based on the message passing paradigm [30], wherein node features are learned by means of a non-linear propagation on the graph. Multiple recent works have pointed to the limitations of the expressive power [82, 97, 59, 54, 19, 9, 7] of this approach [30] and the related oversmoothing [62, 64] and bottleneck phenomena [1, 95] that render such approaches inefficient, especially in deep GNNs. Multiple alternatives have been proposed, among which are higher-order methods [54, 7] and decoupling the propagation graph from the input one by modifying the topology, often referred to as *graph rewiring*. Topological modifications can take different forms such as graph sampling [32], kNN [44], using the complete graph [88, 1], latent graph learning [91, 36], or multi-hop filters [94, 75]. However, there is no agreement in the literature on when and how to modify the graph, and a single principled framework for doing so.

A somewhat underappreciated fact is that GNNs are intimately related to diffusion equations, a connection that was exploited in the early work of Scarselli et al. [78]. Diffusion PDEs have been historically important in computer graphics [85, 11, 51, 66], computer vision [13, 17, 6], and image processing [67, 84, 92, 87, 26, 12], where they created an entire trend of variational and PDE-based approaches. In machine learning and data science, diffusion equations underpin such popular manifold learning methods as eigenmaps [5] and diffusion maps [20], as well as the family of PageRank algorithms [65, 14]. In deep learning, differential equations are used as models of neural networks [18, 25, 96, 73, 100] and for physics-informed learning [74, 22, 77, 21, 83, 47].

Main contributions In this paper, we propose a novel class of GNNs based on the discretized non-Euclidean diffusion PDE in joint positional and feature space, inspired by the *Beltrami flow* [84] used two decades ago in the image processing literature for edge-preserving image denoising. We show that the discretization of the spatial component of the Beltrami flow offers a principled view on positional encoding and graph rewiring, whereas the discretization of the temporal component can replace GNN layers with more flexible adaptive numerical schemes. Based on this model, we introduce Beltrami Neural Diffusion (BLEND) that generalizes a broad range of GNN architectures and shows state-of-the-art performance on many popular benchmarks. In a broader perspective, our approach explores new tools from PDEs and differential geometry that are less well known in the graph ML community.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

2 Background

Beltrami flow Kimmel et al. [41, 84, 40] considered images as embedded 2-manifolds (parametric surfaces) of the form $\mathbf{z}(\mathbf{u}) = (\mathbf{u}, \alpha\mathbf{x}(\mathbf{u})) \subseteq \mathbb{R}^{d+2}$ where $\alpha \geq 0$ is a scaling factor, $\mathbf{u} = (u_1, u_2)$ are the 2D *positional coordinates* of the pixels, and $\mathbf{x}$ are the d-dimensional *color* or *feature coordinates* (with $d = 1$ or 3 for grayscale or RGB images, or $d = k^2$ when using $k \times k$ patches as features [12]). The embedding of the manifold induces a local distance structure: the distance between two nearby points $\mathbf{u}$ and $\mathbf{u} + \mathrm{d}\mathbf{u}$ is given by $\mathrm{d}\ell^2 = \mathrm{d}\mathbf{u}^\top \mathbf{G}(\mathbf{u})\mathrm{d}\mathbf{u} = \mathrm{d}u_1^2 + \mathrm{d}u_2^2 + \alpha^2 \sum_{i=1}^d \mathrm{d}x_i^2$, where $\mathbf{G} = \mathbf{I} + \alpha^2(\nabla_\mathbf{u}\mathbf{x}(\mathbf{u}))^\top \nabla_\mathbf{u}\mathbf{x}(\mathbf{u})$ is a 2×2 matrix called the *Riemannian metric*. The fact that the distance is a combination of the positional component (distance between pixels in the plane, $\|\mathbf{u} - \mathbf{u}'\|$) and color component (distance between the colors of the pixels, $\|\mathbf{x}(\mathbf{u}) - \mathbf{x}(\mathbf{u}')\|$) is crucial as it allows edge-preserving image diffusion by evolving the manifold according to the following PDE:

$$\frac{\partial \mathbf{z}(\mathbf{u}, t)}{\partial t} = \Delta_\mathbf{G}\mathbf{z}(\mathbf{u}, t), \qquad \mathbf{z}(\mathbf{u}, 0) = \mathbf{z}(\mathbf{u}), \quad t \geq 0, \tag{1}$$

and boundary conditions as appropriate. Here $\Delta_\mathbf{G}$ is the *Laplace-Beltrami operator*, a non-Euclidean version of the Laplacian defined on the manifold. Equation (1), referred to as *Beltrami flow*, can be obtained as the gradient flow of the Euler-Lagrange equations associated with a functional called the *Polyakov action* in particle physics [70] (roughly speaking, the Polyakov action is a non-Euclidean version of the Dirichlet energy measuring the smoothness of the manifold, the Euler-Lagrange equations express a first-order optimality condition, and the gradient flow is a continuous analogy of gradient descent for functionals).

When dealing with images, the evolution of the first two components of $(z_1, z_2) = \mathbf{u}$ is a nuisance amounting to the reparametrization of the manifold and can be ignored. For grayscale images (the case when $d = 1$ and $\mathbf{z} = (u_1, u_2, x)$), this is done by projection along the dimension z_3, in which case the Beltrami flow takes the form of an inhomogeneous diffusion equation of x,

$$\frac{\partial x(\mathbf{u}, t)}{\partial t} = \frac{1}{\sqrt{\det \mathbf{G}(\mathbf{u}, t)}} \operatorname{div}\left(\frac{\nabla x(\mathbf{u}, t)}{\sqrt{\det \mathbf{G}(\mathbf{u}, t)}}\right) \qquad t \geq 0. \tag{2}$$

The *diffusivity*

$$a = \frac{1}{\sqrt{\det \mathbf{G}}} = \frac{1}{\sqrt{1 + \alpha^2 \|\nabla x\|^2}} \tag{3}$$

determining the speed of diffusion at each point, can be interpreted as an *edge indicator*: diffusion is weak across edges where $\|\nabla x\| \gg 1$. The result is an adaptive diffusion [67] popular in image processing due to its ability to denoise images while preserving their edges. For cases with $d > 1$ (multiple color channels), equation (2) is applied to each channel separately; however, the metric $\mathbf{G}$ couples the channels and results in their gradients becoming aligned [39].

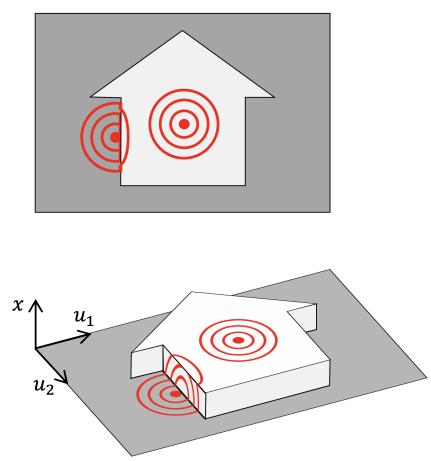

Figure 1: Two interpretations of the Beltrami flow: position-dependent bilateral kernel (top) and a Gaussian passed on the manifold (bottom).

Special cases In the limit case $\alpha = 0$, equation (2) becomes the simple homogeneous isotropic diffusion $\frac{\partial}{\partial t}x = \operatorname{div}(\nabla x) = \Delta x$, where $\Delta = \frac{\partial^2}{\partial u_1^2} + \frac{\partial^2}{\partial u_2^2}$ is the standard Euclidean Laplacian operator. The solution is given in closed form as convolution of the initial image and a Gaussian kernel with time-dependent variance, $x(\mathbf{u}, t) = x(\mathbf{u}, 0) \star \frac{1}{(4\pi t)^{d/2}} e^{-\|\mathbf{u}\|^2/4t}$ and can be considered a simple linear low-pass filtering. In the limit $t \to \infty$, the image becomes constant and equal to the average color.[1]

Another interpretation of the Beltrami flow is passing a Gaussian *on the manifold* (see Figure 1, bottom), which can locally be expressed as non-linear filtering with the *bilateral kernel* [87] dependent

[1] Assuming appropriate boundary conditions.

on the joint positional and color distance (Figure 1, top),

$$x(\mathbf{u},t) = \int_{\mathbb{R}^2} x(\mathbf{x},0) e^{-\|\mathbf{u}-\mathbf{v}\|^2/4t} e^{-\alpha^2\|\mathbf{x}(\mathbf{u})-\mathbf{x}(\mathbf{v})\|^2/4t} \mathrm{d}\mathbf{v}. \tag{4}$$

For $\alpha = 0$, the bilateral filter (4) reduces to a simple convolution with a time-dependent Gaussian.

3 Discrete Beltrami flow on graphs

We now develop the analogy of Beltrami flow for graphs. With this approach, we consider a graph to be a *discretization* of a continuous structure (manifold), and show that the evolution of the feature coordinates in time amounts to message passing layers in GNNs, whereas the evolution of the positional coordinates amounts to graph rewiring used in some GNN architectures.

3.1 Graph Beltrami flow

Let $\mathcal{G} = (\mathcal{V} = \{1,\dots,n\}, \mathcal{E})$ be an undirected graph, where $\mathcal{V}$ and $\mathcal{E}$ denote node and edge sets, respectively. We further assume node-wise d-dimensional features $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1,\dots,n$. Denote by $\mathbf{z}_i = (\mathbf{u}_i, \alpha\mathbf{x}_i)$ the embedding of the graph in a joint space $\mathcal{C} \times \mathbb{R}^d$, where $\mathcal{C}$ is a d'-dimensional space with a metric $d_{\mathcal{C}}$ representing the node coordinates (for simplicity, we will assume $\mathcal{C} = \mathbb{R}^{d'}$ unless otherwise stated). We refer to $\mathbf{u}_i$ and $\mathbf{x}_i$ as the *positional* and *feature coordinates* of node i, respectively, and arrange them into the matrices $\mathbf{U}$, $\mathbf{X}$, and $\mathbf{Z}$, of sizes $n \times d'$, $n \times d$, and $n \times (d' + d)$.

The *graph Beltrami flow* is a discrete diffusion equation of the form

$$\frac{\partial \mathbf{z}_i(t)}{\partial t} = \sum_{j:(i,j)\in\mathcal{E}'} a(\mathbf{z}_i(t), \mathbf{z}_j(t))(\mathbf{z}_j(t) - \mathbf{z}_i(t)) \qquad \mathbf{z}_i(0) = \mathbf{z}_i; \quad i = 1,\dots,n; \quad t \geq 0 \tag{5}$$

which can be considered as a discretization of equation (2): $\mathbf{g}_{ij} = \mathbf{z}_j - \mathbf{z}_i$ and $\mathbf{d}_i = \sum_{j:(i,j)\in\mathcal{E}} \mathbf{g}_{ij}$ are the discrete analogies of the gradient $\nabla\mathbf{z}$ and divergence $\mathrm{div}(\mathbf{g})$, and $a(\mathbf{z}_i, \mathbf{z}_j)$ is the diffusivity controlling the diffusion strength between nodes i and j assumed to be normalized: $\sum_{j:(i,j)\in\mathcal{E}'} a(\mathbf{z}_i, \mathbf{z}_j) = 1$. The graph $(\mathcal{V}, \mathcal{E}')$ used for diffusion can be interpreted as the numerical stencil for the discretization of the continuous Laplace-Beltrami operator in (2). Note that $\mathcal{E}'$ can potentially be different from the input one, $\mathcal{E}$ (referred to as 'rewiring'). As discussed in the next section, most GNNs use $\mathcal{E}' = \mathcal{E}$ (input graph is used for diffusion, no rewiring). Alternatively, the positional coordinates of the nodes can be used to define a new graph topology either with $\mathcal{E}(\mathbf{U}) = \{(i,j) : d_{\mathcal{C}}(\mathbf{u}_i, \mathbf{u}_j) < r\}$, for some radius $r > 0$, or using k nearest neighbours. This new rewiring can be precomputed using the input positional coordinates (i.e., $\mathcal{E}' = \mathcal{E}(\mathbf{U}(0))$) or updated throughout the diffusion (i.e., $\mathcal{E}'(t) = \mathcal{E}(\mathbf{U}(t))$).

Equation (5) can be rewritten compactly in matrix-vector notation,

$$\left(\tfrac{\partial}{\partial t}\mathbf{U}(t), \tfrac{\partial}{\partial t}\mathbf{X}(t)\right) = \left(\mathbf{A}(\mathbf{U}(t), \mathbf{X}(t)) - \mathbf{I}\right)(\mathbf{U}(t), \mathbf{X}(t)), \;\; \mathbf{U}(0) = \alpha\mathbf{U}; \;\; \mathbf{X}(0) = \alpha\mathbf{X}; \;\; t \geq 0, \tag{6}$$

where we emphasize the evolution of both the positional and feature components, coupled through the matrix-valued function $\mathbf{A}$

$$a_{ij}(t) = \begin{cases} a((\mathbf{u}_i(t), \mathbf{x}_i(t)), (\mathbf{u}_j(t), \mathbf{x}_j(t))) & (i,j) \in \mathcal{E}(\mathbf{U}(t)) \\ 0 & \text{else.} \end{cases}$$

representing the diffusivity. The graph Beltrami flow produces an evolution of the joint positional and feature coordinates, $\mathbf{Z}(t) = (\mathbf{U}(t), \mathbf{X}(t))$. In Section 3.3 we will show how the evolution of the feature coordinates $\mathbf{X}(t)$ results in feature diffusion or message passing on the graph, the core of GNNs. Note that in this formulation the diffusion does not mix different channels. Similarly to the continuous case, the evolution can be interpreted as a gradient flow (we refer to the Supplementary Materials for the proof and definitions):

Theorem 1. *Under structural assumptions on the diffusivity, graph Beltrami flow (5) is the gradient flow of the discrete Polyakov functional.*

3.2 Numerical solvers

Explicit vs implicit schemes Equation (5) is solved numerically, which in the simplest case is done by replacing the continuous time derivative $\frac{\partial}{\partial t}$ with forward time difference:

$$\frac{\mathbf{z}_i^{(k+1)} - \mathbf{z}_i^{(k)}}{\tau} = \sum_{j:(i,j)\in\mathcal{E}(\mathbf{U}^{(k)})} a\left(\mathbf{z}_i^{(k)}, \mathbf{z}_j^{(k)}\right)(\mathbf{z}_j^{(k)} - \mathbf{z}_i^{(k)}). \tag{7}$$

Here k denotes the discrete time index (iteration) and τ is the time step (discretization parameter). Rewriting (7) compactly in matrix-vector form leads to the *explicit* or *forward Euler scheme*:

$$\mathbf{Z}^{(k+1)} = (\mathbf{A}^{(k)} - \mathbf{I})\mathbf{Z}^{(k)} = \mathbf{Q}^{(k)}\mathbf{Z}^{(k)}, \tag{8}$$

where $a_{ij}^{(k)} = a(\mathbf{z}_i^{(k)}, \mathbf{z}_j^{(k)})$ and the matrix $\mathbf{Q}^{(k)}$ (diffusion operator) is given by

$$q_{ij}^{(k)} = \begin{cases} 1 - \tau \sum\limits_{l:(i,l)\in\mathcal{E}} a_{il}^{(k)} & i = j \\ \tau a_{ij}^{(k)} & (i,j) \in \mathcal{E}(\mathbf{U}^{(k)}) \\ 0 & \text{otherwise} \end{cases}$$

The solution to the diffusion equation is computed by applying the scheme (8) multiple times in sequence, starting from some initial $\mathbf{Z}^{(0)}$. The scheme (8) is called 'explicit' because the update $\mathbf{Z}^{(k+1)}$ is done directly by the application of the diffusion operator $\mathbf{Q}^{(k)}$ on $\mathbf{Z}^{(k)}$ (as opposed to *implicit schemes* of the form $\mathbf{Z}^{(k)} = \mathbf{Q}^{(k)}\mathbf{Z}^{(k+1)}$ arising from backward time differences and requiring inversion of the diffusion operator [93]).

Multi-step and adaptive schemes Higher-order approximation of temporal derivatives amount to using intermediate fractional steps, which are then linearly combined. Runge-Kutta (RK) [76, 45], ubiquitously used in numerical analysis, is a classical family of explicit numerical schemes, including Euler as a particular case. The Dormand-Prince (DOPRI) [24] is an RK method based on fifth- and fourth-order approximations, the difference between which is used as an error estimate guiding the time step size [80].

Adaptive spatial discretization and rewiring Many numerical PDE solvers also employ adaptive spatial discretization. The choice of the stencil (mesh) for spatial derivatives is done based on the character of the solution at these points; in the simulation of phenomena such as shockwaves it is often desired to use denser sampling in the respective regions of the domain, which can change in time. A class of techniques for adaptive rewiring of the spatial derivatives are known as Moving Mesh (MM) methods [33]. Interpreting the graph $\mathcal{E}'$ in (5) as the numerical stencil for the discretization of the continuous Laplace-Beltrami operator in (2), we can regard rewiring as a form of MM.

3.3 Relation to graph neural networks

Equation (7) has the structure of many GNN architectures of the 'attentional' type [10], where the discrete time index k corresponds to a (convolutional or attentional) layer of the GNN and multiple diffusion iterations amount to a deep GNN. In the diffusion formalism, the time parameter t acts as a continuous analogy of the layers, in the spirit of neural differential equations [18]. Typical GNNs amount to explicit single-step (Euler) discretization schemes, whereas our continuous interpretation can exploit more efficient numerical schemes.

GNNs as instances of graph Beltrami flow The graph Beltrami framework leads to a family of graph neural networks that generalize many popular architectures (see Table 1). For example, GAT [90] can be obtained as a particular setting of our framework where the input graph is fixed ($\mathcal{E}' = \mathcal{E}$) and only the feature coordinates $\mathbf{X}$ are evolved. Equation (8) in this case becomes

$$\mathbf{x}_i^{(k+1)} = \mathbf{x}_i^{(k)} + \tau \sum_{j:(i,j)\in\mathcal{E}} a\left(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}\right)(\mathbf{x}_j^{(k)} - \mathbf{x}_i^{(k)}) \tag{9}$$

and corresponds to the update formula of GAT with a residual connection and the assumption of no non-linearity between the layers. The role of the diffusivity is played by a learnable parametric

attention function, which is generally *time-dependent*: $a(\mathbf{z}_i^{(k)}, \mathbf{z}_j^{(k)}, k)$. This results in separate attention parameters per layer k, which can be learned independently. Our intentionally simplistic choice of a *time-independent* attention function amounts to *parameter sharing* across layers. We will show in Section **??** that this leads to a smaller model that is less likely to overfit.

Another popular architecture MoNet [58] uses linear diffusion of the features with weights dependent on the structure of the graph expressed as 'pseudo-coordinates', which can be cast as attention of the form $a(\mathbf{u}_i, \mathbf{u}_j)$. Transformers [89] can be interpreted as feature diffusion on a fixed complete graph with $\mathcal{E}' = \mathcal{V} \times \mathcal{V}$ [10]. Positional encoding (used in Transformers as well as in several recent GNN architectures [9, 27]) amounts to attention dependent on both $\mathbf{X}$ and $\mathbf{U}$, which allows the diffusion to adapt to the local structure of the graph; importantly, the positional coordinates $\mathbf{U}$ are *precomputed*. Similarly, DeepSets [99] and PointNet [72] architectures can be interpreted as GNNs applied on a graph with no edges ($\mathcal{E}' = \emptyset$), where each node is treated independently of the others [10]. DIGL [44] performs graph rewiring as a pre-processing step using personalized page rank as positional coordinates, which are then fixed and not evolved. In the point cloud methods, DGCNN [91] and DGM [36], the graph is constructed based on the *feature coordinates* $\mathbf{X}$ and rewired adaptively (in our notation, $\mathcal{E}' = \mathcal{E}(\mathbf{X}(t))$).

Method	**Evolution**	**Diffusivity**	**Graph** $(\mathcal{V}, \mathcal{E}')$	**Discretization**
ChebNet	Features $\mathbf{X}$	Fixed a_{ij}	Fixed $\mathcal{E}$	Explicit fixed step
GAT	Features $\mathbf{X}$	$a(\mathbf{x}_i, \mathbf{x}_j)$	Fixed $\mathcal{E}$	Explicit fixed step
MoNet	Features $\mathbf{X}$	$a(\mathbf{u}_i, \mathbf{u}_j)$	Fixed $\mathcal{E}$	Explicit fixed step
Transformer	Features $\mathbf{X}$	$a((\mathbf{u}_i, \mathbf{x}_i), (\mathbf{u}_j, \mathbf{x}_j))$	Fixed $\mathcal{E} = \mathcal{V} \times \mathcal{V}$	Explicit fixed step
DeepSet/PointNet	Features $\mathbf{X}$	$a(\mathbf{x}_i)$	Fixed $\mathcal{E} = \emptyset$	Explicit fixed step
DIGL*	Features $\mathbf{X}$	$a(\mathbf{x}_i, \mathbf{x}_j)$	Fixed $\mathcal{E}(\mathbf{U})$	Explicit fixed step
DGCNN/DGM*	Features $\mathbf{X}$	$a(\mathbf{x}_i, \mathbf{x}_j)$	Adaptive $\mathcal{E}(\mathbf{X})$	Explicit fixed step
Beltrami	Positions $\mathbf{U}$ +Features $\mathbf{X}$	$a((\mathbf{u}_i, \mathbf{x}_i), (\mathbf{u}_j, \mathbf{x}_j))$	Adaptive $\mathcal{E}(\mathbf{U})$	Explicit adaptive step / Implicit

Table 1: GNN architectures interpreted as particular instances of our framework. *Attentional variant.

Graph rewiring Multiple authors have recently argued in favor of decoupling the input graph from the graph used for diffusion. Such rewiring can take the form of graph sampling [32] to address scalability issues, data denoising [44], removal of information bottlenecks [1], or larger multi-hop filters [75]. The graph construction can also be made differentiable and a task-specific rewiring can be learned [91, 36]. The statement of Klicpera et al. [44] that 'diffusion improves graph learning', leading to the eponymous paradigm (DIGL), can be understood as a form of diffusion on the graph connectivity independent of the features. Specifically, the authors used as node positional encoding the Personalized PageRank (PPR), which can be interpreted as the steady-state of a diffusion process

$$\mathbf{U}_{\mathrm{PPR}} = \sum_{k \geq 0} (1-\beta)\beta^k \mathbf{\Delta}_{\mathrm{RW}}^k = (1-\beta)(\mathbf{I} - \beta \mathbf{\Delta}_{\mathrm{RW}})^{-1}, \tag{10}$$

where $\mathbf{\Delta}_{\mathrm{RW}}$ is the random walk graph Laplacian and $\beta \in (0, 1)$ is a parameter such that $1 - \beta$ represents the restart probability. The resulting positional encoding of dimension $d = n$ can be used to rewire the graph by kNN sampling, which corresponds to using $\mathcal{E}' = \mathcal{E}(\mathbf{U}_{\mathrm{PPR}})$ in our framework.

Numerical schemes All the aforementioned GNN architectures can be seen as an explicit discretization of equation (5) with fixed step size. On the other hand, our continuous diffusion framework offers an additional advantage of employing more efficient numerical schemes with adaptive step size. Graph rewiring of the form $\mathcal{E}' = \mathcal{E}(\mathbf{U}(t))$ can be interpreted as adaptive spatial discretization (moving mesh method).

3.4 Extensions

Non-Euclidean geometry There are multiple theoretical and empirical arguments [52, 16] in favor of using hyperbolic spaces to represent real-life 'small-world' graphs (in particular, scale-free networks can be obtained as kNN graphs in such spaces [8]). Our framework allows using a non-Euclidean metric $d_{\mathcal{C}}$ for the positional coordinates $\mathbf{U}$ (Figure 2). We show that hyperbolic positional encodings allow for a significant reduction in model size with only a marginal degradation of performance in the experimental results in Section 5.

Time-dependent diffusivity The diffusivity function a which we assumed time-independent and which lead to parameter sharing across layers (i.e., updates of the form $\mathbf{Z}^{(k+1)} = \mathbf{Q}(\mathbf{Z}^{(k)}, \boldsymbol{\theta})\mathbf{Z}^{(k)}$) can be made time-dependent of the form $\mathbf{Z}^{(k+1)} = \mathbf{Q}(\mathbf{Z}^{(k)}, \boldsymbol{\theta}^{(k)})\mathbf{Z}^{(k)}$, where $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{(k)}$ denote shared and layer-dependent parameters, respectively.

Onsager diffusion As we noted, the Beltrami flow diffuses each channel separately. A more general variant of diffusion allowing for feature mixing is the *Onsager diffusion* [63] of the form $\frac{\partial}{\partial t}\mathbf{Z}(t) = \mathbf{Q}(\mathbf{Z}(t))\mathbf{Z}(t)\mathbf{W}(t)$, where the matrix-valued function $\mathbf{W}$ acts across the channels. GCN [43] can be regarded a particular setting thereof, with update of the form $\mathbf{X}^{(k+1)} = \mathbf{A}\mathbf{X}^{(k)}\mathbf{W}$.

MPNNs Finally, we note that the Beltrami flow amounts to linear aggregation with non-linear coefficients, or the 'attentional' flavor of GNNs [10]. The more general message passing flavor [30] is possible using a generic non-linear equation of the form $\frac{\partial}{\partial t}\mathbf{Z}(t) = \boldsymbol{\Psi}(\mathbf{Z}(t))$.

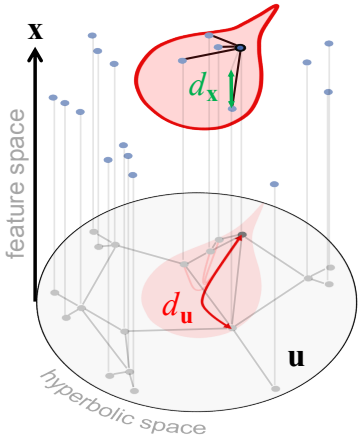

Figure 2: Graph Beltrami flow with hyperbolic positional coordinates.

4 BLEND: Beltrami Neural Diffusion

Beltrami Neural Diffusion (BLEND) is a novel class of graph neural network architectures derived from the graph Beltrami framework. We assume an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes and d-dimensional node-wise features represented as a matrix $\mathbf{X}_{\text{in}}$. We further assume a d'-dimensional positional encoding $\mathbf{U}_{\text{in}}$ of the graph nodes. BLEND architectures implement a learnable joint diffusion process of $\mathbf{U}$ and $\mathbf{X}$ and runs it for time T, to produce an output node embeddings $\mathbf{Y}$,

$$\mathbf{Z}(0) = (\phi(\mathbf{U}_{\text{in}}), \psi(\mathbf{Z}_{\text{in}})) \qquad \mathbf{Z}(T) = \mathbf{Z}(0) + \int_0^T \frac{\partial \mathbf{Z}(t)}{\partial t} \mathrm{d}t \qquad \mathbf{Y} = \psi(\mathbf{Z}(T)),$$

where ϕ, ψ are learnable positional and feature encoders and ξ is a learnable decoder (possibly changing the output dimensions). $\frac{\partial \mathbf{Z}(t)}{\partial t}$ is given by the graph Beltrami flow equation (6), where the diffusivity function (attention) a is also learnable. The choice of attention function depends on the geometry of the positional encoding and for Euclidean encodings we find the scaled dot product attention [88] performs well, in which case

$$a(\mathbf{z}_i, \mathbf{z}_j) = \mathrm{softmax}\left(\frac{(\mathbf{W}_K \mathbf{z}_i)^\top \mathbf{W}_Q \mathbf{z}_j}{d_k}\right) \tag{11}$$

where $\mathbf{W}_K$ and $\mathbf{W}_Q$ are learned matrices, and d_k is a hyperparameter.

5 Experimental results

In this section, we compare the proposed Beltrami framework to popular GNN architectures on standard node classification benchmarks and provide a detailed study of the choose of the positional encoding space. Additional experiments and details are shown in the Supplementary Materials.

Datasets In our experiments, we use the following datasets: Cora [57], Citeseer [79], Pubmed [60], CoauthorCS [81], Amazon, Computer, and Photo [56], and OGB-arxiv [35]. Since many works using the first three datasets rely on the Planetoid splits [98], we included them Table 2, together with a more robust evaluation on 100 random splits with 20 random initializations [81].

Baselines We compare to the following GNN architectures: GCN [43], GAT [90], MoNet [58] and GraphSAGE [32], and recent ODE-based GNN models: CGNN [96], GDE [69], GODE [100], and two versions of LanczosNet [49]. We use two variants of our method: using fixed input graph (BLEND) and using kNN graph in the positional coordinates (BLEND-knn).

Method	CORA	CiteSeer	PubMed	Coauthor CS	Computer	Photo	ogb-arxiv
GCN	81.5±1.3	**71.9** ±1.9	77.8±2.9	91.1±0.5	82.6±2.4	91.2±1.2	71.74±0.29
GAT	81.8±1.3	71.4±1.9	78.7±2.3	90.5±0.6	78.0±19	85.7±20	**73.01**±0.19*
GAT-ppr	81.6±0.3	68.5±0.2	76.7±0.3	91.3±0.1	85.4±0.3	90.9±0.3	—
MoNet	81.3±1.3	71.2±2.0	78.6±2.3	90.8±0.6	83.5±2.2	91.2±2.3	—
GS-mean	79.2±7.7	71.6±1.9	77.4±2.2	91.3±2.8	82.4±1.8	91.4±1.3	71.49±0.27
GS-maxpool	76.6±1.9	67.5±2.3	76.1±2.3	85.0±1.1	—	90.4±1.3	—
CGNN	81.4±1.6	66.9±1.8	66.6±4.4	92.3±0.2	80.3±2.0	91.4±1.5	58.70±2.50
GDE	78.7±2.2	71.8±1.1	73.9±3.7	91.6±0.1	82.9±0.6	92.4±2.0	56.66±10.9
BLEND	**84.8**±0.9	**75.9**±1.3	**79.5**±1.4	**92.9**±0.2	**86.9**±0.6	**92.9**±0.6	**72.56**±0.1
BLEND-kNN	**82.5**±0.9	**73.4**±0.5	**80.9**±0.7	**92.3**±0.1	**86.7**±0.6	**93.5**±0.3	—

Table 3: Performance (test accuracy±std) of different GNN models using random splits. *OGB GAT reference has 1.5M parameters vs ours 70K.

5.1 Node Classification

In this experiment, our goal is to assess the performance of BLEND. We followed the experimental methodology of [81] using 20 random weight initializations for datasets with fixed Planetoid splits and 100 random splits otherwise. Where available, results from [81] are reported. Hyperparameters with the highest validation accuracy were chosen and results are reported on a test set that is used only once. Hyperparameter search used Ray Tune [50] with a thousand random trials using an asynchronous hyperband scheduler with a grace period and half life of ten epochs. The code to reproduce our results is included with the submission and will be released publicly following the review process. Experiments ran on AWS p2.8xlarge machines, each with 8 Tesla V100-SXM2 GPUs.

Implementation details For all datasets excepting ogb-arxiv, adaptive explicit Dormand-Prince scheme was used as the numerical solver; for ogb-arxiv, we used the Runge-Kutta method. For the two smallest datasets (Cora and Citeseer) we performed direct backpropagation through each step of the numerical integrator. For the larger datasets, to reduce memory complexity, we use Pontryagin's maximum principle to propagate gradients backwards in time [71]. For the larger datasets, kinetic energy and Jacobian regularization [28, 37] was employed. The regularization ensures the learned dynamics is well-conditioned and easily solvable by a numeric solver, which reduced training time. We use constant initialization for the attention weights, $\mathbf{W}_K, \mathbf{W}_Q$, so training starts from a well-conditioned system that induces small regularization penalty terms [28].

The space complexity of BLEND is dominated by evaluating attention (11) over edges and is $\mathcal{O}(|\mathcal{E}'|(d+d'))$ where $\mathcal{E}'$ is the edge set following rewiring and d is dimension of features and d' is the dimension of positional encoding. The runtime complexity is $\mathcal{O}(|\mathcal{E}'|(d+d'))(E_b+E_f)$, split between the forward and backward pass and can be dominated by either depending on the number of function evaluations (E_b, E_f).

Method	CORA	CiteSeer	PubMed
GCN	81.9±0.8	69.5±0.9	79.0±0.5
GAT	82.8±0.5	71.0±0.6	77.0±1.3
MoNet	82.2±0.7	70.0±0.6	77.7±0.6
GS-maxpool	77.4±1.0	67.0±1.0	76.6±0.8
Lanczos	79.5±1.8	66.2±1.9	78.3±0.3
AdaLanczos	80.4±1.1	68.7±1.0	78.1±0.4
CGNN	81.7±0.7	68.1±1.2	80.2±0.3
GDE	**83.8**±0.5	72.5±0.5	79.9±0.3
GODE	83.3±0.3	72.4±0.6	80.1±0.3
BLEND	**84.2**±0.6	**74.4**±0.7	**80.7**± 0.7
BLEND-kNN	83.1±0.8	**73.3**±0.9	**81.5**±0.5

Table 2: Performance (test accuracy±std) of different GNN models using Planetoid splits.

Tables 2–3 summarize the results of our experiments. BLEND outperforms other GNNs in most of the experiments, showing state-of-the-art results on some datasets. Another important point to note is that compared GNNs use different sets of parameters per layer, whereas in BLEND, due to our choice of a time-independent attention, parameters are shared. This results in significantly fewer parameters: for comparison, the OGB versions of GCN, SAGE and GAT used in the ogb-arxiv experiment require 143K, 219K and 1.63M parameters respectively, compared to only 70K in BLEND.

5.2 Positional encoding

In the second experiment, we investigated the impact of varying the dimensionality and underlying geometry of the positional encodings, in order to showcase the flexibility of our framework. We used

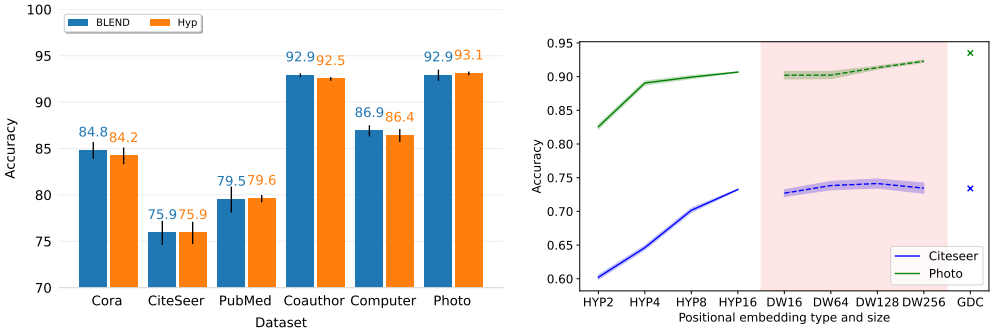

Figure 3: Left: performance comparison between Euclidean and hyperbolic positional embeddings. Right: results of positional embeddings ablation. Hyperbolic or Euclidean embeddings with $d' = 16$ allow to obtain performances comparable to euclidean embeddings with $d' = n >> 16$.

three forms of encoding: *DIGL* PPR embeddings of dimension n [44], *DeepWalk* embeddings [68] of dimensions 16–256, and *hyperbolic* embeddings in the Poincare ball [15, 61] of dimension 2–16. Positional encodings are calculated as a preprocessing step and input as the $\mathbf{U}$ coordinates to BLEND. For our experiments, we calculated DeepWalk node embeddings using PyTorch Geometric's Node2Vec implementation with parameters $p = 1$, $q = 1$, context size 20, and 16 walks per node. We trained with walk length ranging between 40 and 120 and took the embeddings with the highest accuracy for the link prediction task. Shallow hyperbolic embeddings were generated using the HGCN [16] implementation with the default parameters provided by the authors.

Figure 3 (left) compares the performance of the best model using DIGL n-dimensional positional encodings against the best performing d-dimensional hyperbolic positional encodings with d tuned over the range 2-16. The average performance with DIGL positional encodings is 85.48, compared to 85.28 for hyperbolic encodings. Only in one case the DIGL encodings outperform the best hyperbolic encodings. Figure 3 (right) show the change in performance with the dimension d' of the positional encoding using a fixed hyperparameter configuration. As expected, we observe monotonic increase in performance as function of d'. Importantly most of the performance is captured by either 16 dimensional hyperbolic or positional encodings, with a small additional margin gained for going up to $d' = n$, which is impractical for larger graphs.

6 Related work

Image processing, computer vision, and graphics. Following the Perona-Malik scheme [67], PDE-based approaches flourished in the 1990s with multiple versions of anisotropic [92] and non-Euclidean [84] diffusion used primarily for image denoising. The realization of these ideas in the form of nonlinear filters [87, 12] was adopted in the industry. PDE-based methods were also used for low-level vision tasks including inpainting [6] and image segmentation [13, 17]. In computer graphics, fundamental solutions ('heat kernels') of diffusion equations were used as shape descriptors [85]. The closed-form expression of such solutions using the Laplace-Beltrami operator served as inspiration for some of the early approaches for GNNs [34, 23, 43, 46].

Neural differential equations The interpretation of neural networks as discretized differential equations ('neural ODEs') [18] was an influential recent result with multiple follow-up works [25, 55, 28, 53, 38]. In deep learning on graphs, this mindset has been applied to GNN architectures [2, 69] and continuous message passing [96]. Continuous GNNs were also explored in [31] who, similarly to [78], addressed the solutions of fixed point equations. Ordinary Differential Equations on Graph Networks (GODE)[100] approach the problem using the technique of invertible ResNets. Finally, [77] used graph-based ODEs to generate physics simulations.

Physics-inspired learning Using deep learning to solve PDEs has been explored by [74]. Neural networks appeared in [47] to accelerate PDE solvers with applications in the physical sciences. These have been applied to problems where the PDE can be described on a graph [48]. [4] consider the problem of predicting fluid flow and use a PDE inside a GNN. These approaches differ from ours

in that they solve a given PDE, whereas we use the notion of discretizing PDEs as a principle to understand and design GNNs.

7 Conclusion

We developed a new class of graph neural networks based on the discretization of a non-Euclidean diffusion PDE called Beltrami flow. We represent the graph structure and node features as a manifold in a joint continuous space, whose evolution by a parametric diffusion PDE (driven by the downstream learning task) gives rise to feature learning, positional encoding, and possibly also graph rewiring. Our experimental results show very good performance on popular benchmarks with a small fraction of parameters used by other GNN models. Perhaps most importantly, our framework establishes important links between GNNs, differential geometry, and PDEs – fields with a rich history and profound results that in our opinion are still insufficiently explored in the graph ML community.

Future directions While we show that our framework generalizes many popular GNN architectures we seek to use the graph as a numerical representation of an underlying continuous space. This view of the graph as an approximation of a continuous latent structure is a common paradigm of manifold learning [86, 5, 20] and network geometry [8]. If adopted in graph ML, this mindset offers a rigorous mathematical framework for formalizing and generalizing some of the recent trends in the field, including the departure from the input graph as the basis for message passing [1], latent graph inference [42, 91, 29, 36] higher-order [7, 54] and directional [3] message passing (which can be expressed as anisotropic diffusion arising from additional structure of the underlying continuous space and different discretization of the PDEs e.g. based on finite elements), and exploiting efficient numerical PDE solvers [18]. We leave these exciting directions for future research.

Societal impact The main focus of our paper is on a theoretical perspective allowing a new interpretation of graph neural networks. Since graph neural networks are still experimental technology scarsely used in industrial systems, it is hard to evaluate the potential immediate adverse societal impacts of the discussed approaches. Certain hypothetical applications of graph neural networks (and our method as an instance thereof), e.g. on social networks and recommender systems might have societal implications that have to be studied more carefully. Our method in this sense is not different from other GNNs.

Limitations The very assumption that the graph can be modeled as a discretization of some continuous space makes our framework applicable only to cases where the edge and node features are continuous in nature. Applications e.g. to knowledge graphs with categorical attributes could only be addressed by first embedding such attributes in a continuous space.

References

[1] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

[2] Pedro H. C. Avelar, Anderson R. Tavares, Marco Gori, and Luis C. Lamb. Discrete and continuous deep residual learning over graphs. *arXiv:1911.09554*, 2019.

[3] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. *arXiv:2010.02863*, 2020.

[4] Filipe de Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *ICML*, 2020.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[6] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *PACMCGIT*, 2000.

[7] Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. *arXiv:2103.03212*, 2021.

[8] Marian Boguna, Ivan Bonamassa, Manlio De Domenico, Shlomo Havlin, Dmitri Krioukov, and M Ángeles Serrano. Network geometry. *Nature Reviews Physics*, pages 1–22, 2021.

[9] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv:2006.09252*, 2020.

[10] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478*, 2021.

[11] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *CVPR*, 2010.

[12] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *ICCV*, 2005.

[13] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

[14] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, 2007.

[15] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *arXiv:1705.10359*, 2017.

[16] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *NeurIPS*, 2019.

[17] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Trans. Image Processing*, 10(2):266–277, 2001.

[18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[19] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *NeurIPS*, 2020.

[20] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[21] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. 2020.

[22] Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv:1909.05862*, 2019.

[23] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

[24] John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[25] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In *NeurIPS*, 2019.

[26] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proc Computer Graphics and Interactive Techniques*, 2002.

[27] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv:2012.09699*, 2020.

[28] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: The world of Jacobian and kinetic regularization. In *ICML*, 2020.

[29] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *ICML*, 2019.

[30] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[31] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *NeurIPS*, 2020.

[32] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[33] DF Hawken, JJ Gottlieb, and JS Hansen. Review of some adaptive node-movement techniques in finite-element and finite-difference solutions of partial differential equations. *Journal of Computational Physics*, 95(2):254–302, 1991.

[34] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015.

[35] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv:2005.00687*, 2020.

[36] Anees Kazi, Luca Cosmo, Nassir Navab, and Michael Bronstein. Differentiable graph module (DGN) graph convolutional networks. *arXiv:2002.04999*, 2020.

[37] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. In *NeurIPS*, 2020.

[38] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. (1), 2020.

[39] Ron Kimmel. *Numerical geometry of images: Theory, algorithms, and applications*. Springer, 2012.

[40] Ron Kimmel, Ravi Malladi, and Nir Sochen. Images as embedded maps and minimal surfaces: movies, color, texture, and volumetric medical images. *IJCV*, 39(2):111–129, 2000.

[41] Ron Kimmel, Nir Sochen, and Ravi Malladi. From high energy physics to low level vision. pages 236–247. Springer, 1997.

[42] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

[43] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[44] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

[45] Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[46] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Processing*, 2017.

[47] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. 2020.

[48] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In *NeurIPS*, 2020.

[49] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convo-lutional networks. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[50] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. In *arXiv: 1807.05118.*, 2018.

[51] Roee Litman and Alexander M Bronstein. Learning spectral descriptors for deformable shape correspondence. *PAMI*, 36(1):171–180, 2013.

[52] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. *arXiv preprint arXiv:1910.12892*, 2019.

[53] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural SDE: Stabilizing neural ODE networks with stochastic noise. (2), 2019.

[54] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, pages 2153–2164, 2019.

[55] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural ODEs. 2020.

[56] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings Information Retrieval*, 2015.

[57] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[58] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017.

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

[60] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *Proceedings Mining and Learning with Graphs*, 2012.

[61] Maximillian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pages 3779–3788. PMLR, 2018.

[62] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. 2019.

[63] Lars Onsager. Reciprocal relations in irreversible processes. i. *Physical Review*, 37(4):405, 1931.

[64] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

[65] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, 1999.

[66] Giuseppe Patané. Laplacian spectral kernels and distances for geometry processing and shape analysis. *Computer Graphics Forum*, 35(2):599–624, 2016.

[67] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *PAMI*, 12(7):629–639, 1990.

[68] Bryan Perozzi, Rami Al-Rfou, and Steven 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.

[69] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. pages 6571–6583, 2019.

[70] Alexander M Polyakov. Quantum geometry of bosonic strings. *Physics Letters B*, 103(3):207–210, 1981.

[71] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[72] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[73] Alejandro F. Queiruga, N. Benjamin Erichson, Dane Taylor, and Michael W. Mahoney. Continuous-in-depth neural networks. 2020.

[74] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations. (Part II), 2017.

[75] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. SIGN: Scalable inception graph neural networks. *arXiv:2004.11198*, 2020.

[76] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[77] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ODE integrators. 2019.

[78] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 27(8):61–80, 2009.

[79] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

[80] Lawrence F Shampine. Some practical Runge-Kutta formulas. *Mathematics of Computation*, 46(173):135–150, 1986.

[81] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv:1811.05868*, 2018.

[82] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12:2539–2561, 2011.

[83] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[84] Nir Sochen, Ron Kimmel, and Ravi Malladi. A general framework for low level vision. *IEEE Trans. Image Processing*, 7(3):310–318, 1998.

[85] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

[86] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Dcience*, 290(5500):2319–2323, 2000.

[87] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *ICCV*, 1998.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Akob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[90] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[91] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graphics*, 38(5):1–12, 2019.

[92] Joachim Weickert. *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998.

[93] Joachim Weickert, Bart M. Ter Haar Romeny, and Max A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Trans. Image Processing*, 7(3):398–410, 1998.

[94] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*.

[95] Tailin Wu, Hongyu Ren, Pan Li, and Jure Leskovec. Graph information bottleneck. *NeurIPS*, 2020.

[96] Louis-pascal A C Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In *ICML*, 2020.

[97] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*.

[98] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[99] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *NeurIPS*, 2017.

[100] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S. Duncan. Ordinary differential equations on graph networks. Technical Report 1, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In limitations (S6) and an extensions (S3.4)
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Section 6
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? Theorem 1 is dealt with in the supplementary material [Yes] Supplementary Material
   (b) Did you include complete proofs of all theoretical results? [Yes] Supplementary Material
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] hyperparameters are included in Supplementary Material and splits are addressed in section 5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Results in Section 5 include error bars
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] AWS, details at the end of 5.1 and supplementary material
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] papers have been cited for the most relevant OS libraries
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]