GRAND: Graph Neural Diffusion

Benjamin P. Chamberlain [*1] **James Rowbottom** [*1] **Maria Gorinova** [1] **Stefan Webb** [1] **Emanuele Rossi** [1]
Michael M. Bronstein [1 2 3]

Abstract

We present Graph Neural Diffusion (GRAND) that approaches deep learning on graphs as a continuous diffusion process and treats Graph Neural Networks (GNNs) as discretisations of an underlying PDE. In our model, the layer structure and topology correspond to the discretisation choices of temporal and spatial operators. Our approach allows a principled development of a broad new class of GNNs that are able to address the common plights of graph learning models such as depth, oversmoothing, and bottlenecks. Key to the success of our models are stability with respect to perturbations in the data and this is addressed for both implicit and explicit discretisation schemes. We develop linear and nonlinear versions of GRAND, which achieve competitive results on many standard graph benchmarks.

1. Introduction

Machine learning on graphs and graph neural networks (GNNs) have been shown to be successful in a broad range of problems across different domains, extending way beyond machine learning. Important results have been achieved in the physical sciences (Li et al., 2020a;b), where *partial differential equations* (PDEs) have traditionally been the dominant modelling paradigm.

GNNs are in fact intimately connected to differential equations. The seminal work of Scarselli et al. (2009) was concerned with finding the fixed points of differential equations using the Almeida-Pineda algorithm (Almeida, 1987; Pineda, 1987). The currently predominant message passing paradigm (Gilmer et al., 2017) can be modelled as a differential equation. More recently, diffusion processes have been shown to be an effective preprocessing step for graph learning (Klicpera et al., 2019).

[*]Equal contribution [1]Twitter Inc., London, UK [2]Imperial College London, UK [3]IDSIA/USI, Switzerland. Correspondence to: Ben Chamberlain <bchamberlain@twitter.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

PDEs are among the most studied mathematical constructions, with a vast literature dating back at least to Leonhard Euler in the eighteenth century. This includes various discretisation schemes, numerical methods for approximate solutions, and theorems for their existence and stability. Historically, PDE-based methods have been used extensively in signal and image processing (Perona & Malik, 1990), computer graphics (Sun et al., 2009), and more recently, in machine learning (Chen et al., 2018).

Our goal is to show that the tools of PDEs can be used to understand existing GNN architectures and as a principled way to develop a broad class of new methods. We focus on GNN architectures that can be interpreted as information diffusion on graphs, modelled by the diffusion equation. In doing so, we show that many popular GNN architectures can be derived from a single mathematical framework by different choices of the form of diffusion equation and discretisation schemes. Standard GNNs are equivalent to the explicit single-step Euler scheme that is inefficient and requires small step sizes. We show that more advanced, adaptive multi-step schemes such as Runge-Kutta perform significantly better and using implicit schemes, which are unconditionally stable, amounts to larger multi-hop diffusion operators. Choosing different spatial discretisation amounts to *graph rewiring*, a technique recently used to improve the performance of GNNs (Klicpera et al., 2019; Alon & Yahav, 2021). We show that appropriate choices within our framework allow the design of deep GNN architectures with tens of layers. This is a feat hard to achieve otherwise due to feature oversmoothing (NT & Maehara, 2019; Oono & Suzuki, 2020) and bottlenecks (Alon & Yahav, 2021) – phenomena that are recognised as a common plight of most graph learning architectures.

Main contributions We describe a broad new class of GNNs based on the discretised diffusion PDE on graphs and study different numerical schemes for their solution. Second, we provide stability conditions for these schemes. Finally, based on our model, we develop linear and nonlinear Graph Neural Diffusion (GRAND) architectures that perform competitively on many popular benchmark datasets. We show detailed ablation studies shedding light on the choice of numerical schemes and parameters.

2. Background

Central to our work is the notion of diffusion processes. In this section, we provide a concise background on diffusion equations in the continuous setting, on which we build in Section 3 to develop similar notions on graphs. As we are concerned with continuous analogues of graph diffusion and graphs are associated with a broad array of underlying geometries, it is inadequate to formulate these processes in simple flat spaces and more general Riemannian manifolds are required.

Diffusion equation We are interested in studying diffusion processes on Ω. Informally, diffusion describes the movement of a substance from regions of higher to lower concentration. For example, when a hot object is placed on a cold surface, heat will diffuse from the object to the surface until both are of equal temperature.

Let $x(t)$ denote a family of scalar-valued functions on $\Omega \times [0, \infty)$ representing the distribution of some property (which we will assume to be temperature for simplicity) on Ω at some time, and let $x(u, t)$ be its value at point $u \in \Omega$ at time t. According to Fourier's law of heat conduction, the heat flux

$$h = -g\nabla x,$$

is proportional to the temperature *gradient* ∇x, where g is the *diffusivity* describing the thermal conductance properties of Ω. An idealized *homogeneous* setting assumes that g is a constant scalar throughout Ω. More generally, the diffusivity is a *inhomogeneous* (position-dependent) function that can be scalar-valued (in which case it simply scales the temperature gradient and is *isotropic*) or matrix-valued (in which case the diffusion is said to be *anisotropic*, or direction-dependent). The continuity condition $x_t = -\mathrm{div}(h)$ (roughly meaning that the only change in the temperature is due to the heat flux, as measured by the *divergence* operator, i.e., heat is not created or destroyed), leads to a PDE referred to as the *(heat) diffusion equation*,

$$\frac{\partial x(u,t)}{\partial t} = \mathrm{div}[g(u, x(u,t), t)\nabla x(u,t)],$$

with the initial condition $x(u, 0) = x_0(t)$; for simplicity, we assume no boundary conditions. The choice of the diffusivity function determines if the diffusion is homogeneous ($g = c$), inhomogeneous ($g(u, t)$), or anisotropic ($A(u, t)$). In the isotropic case, the diffusion equation can be expressed as $\frac{\partial x(u,t)}{\partial t} = \mathrm{div}(c\nabla x) = c\Delta x$, where $\Delta x = \mathrm{div}(\nabla x)$ is the *Laplacian* operator.

Diffusion on manifolds In our discussion so far we assumed some abstract domain Ω. The structure of the domain is manifested in the definition of the spatial differential operators in the diffusion PDE. In a general setting, we model Ω as a Riemannian manifold, and let $\mathcal{X}(\Omega)$ and $\mathcal{X}(T\Omega)$ denote the spaces of *scalar* and *(tangent) vector fields* on it, respectively. We denote by $\langle x, y\rangle$ and $\langle\!\langle \mathscr{X}, \mathscr{Y}\rangle\!\rangle$ the respective inner products on $\mathcal{X}(\Omega)$ and $\mathcal{X}(T\Omega)$. Furthermore, we denote by $\nabla : \mathcal{X}(\Omega) \to \mathcal{X}(T\Omega)$ and $\mathrm{div} = \nabla^* : \mathcal{X}(T\Omega) \to \mathcal{X}(\Omega)$ the *gradient* and *divergence* operators, which are adjoint w.r.t. the above inner products: $\langle\!\langle \nabla x, \mathscr{X}\rangle\!\rangle = \langle x, \mathrm{div}(\mathscr{X})\rangle$. Informally, the gradient ∇x of a scalar field x is a vector field providing at each point $u \in \Omega$ the direction $\nabla x(u)$ of the steepest change of x. The divergence $\mathrm{div}(\mathscr{X})$ of a vector field $\mathscr{X}$ is a scalar field providing, at each point, the flow of $\mathscr{X}$ through an infinitesimal volume. The Laplacian Δx can be interpreted as the local difference between the value of a scalar field x at a point and its infinitesimal neighbourhood.

Applications of diffusion equations In image processing, diffusion equations were used for nonlinear filtering of images. Given an image x defined on $\Omega = [0, 1]^2$, the non-homogeneous isotropic diffusion equation

$$\frac{\partial x(t)}{\partial t} = \mathrm{div}\left[g(\|\nabla x(u,t)\|)\nabla x(u,t)\right],$$

applied to the input image $x(u, 0) = x_0(u)$ as the initial condition, is often referred to as *Perona-Malik diffusion* or (erroneously) anisotropic diffusion (Perona & Malik, 1990). The scalar function $g \propto \|\nabla x(u,t)\|^{-1}$ is referred as an *edge indicator* and is designed to prevent diffusion across discontinuities (edges) in the image, thus preserving its sharpness while at the same time removing the noise. In computer graphics and geometry processing, non-Euclidean diffusion equations were studied as shape descriptors.

3. Diffusion equations on graphs

We now define *diffusion equations on graphs*, analogous to Section 2 and argue that formalizing GNNs under the diffusion equation framework provides a principled and rigorous way to develop new architectures for graph learning.

3.1. Graph diffusion equation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = e$ edges, and let $\mathbf{x}$ and $\mathscr{X}$ denote features defined on nodes and edges respectively.[1] The node and edge fields can be represented as n- and e-dimensional vectors assuming some arbitrary ordering of nodes. We adopt the same notation for the respective inner products:

$$\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i\in\mathcal{V}} x_i y_i \qquad \langle\!\langle \mathscr{X}, \mathscr{Y}\rangle\!\rangle = \sum_{i>j} w_{ij}\mathscr{X}_{ij}\mathscr{Y}_{ij}$$

[1] For simplicity, we assume these features to be scalar-valued and refer to them as node and edge fields, by analogy to scalar and vector fields on manifolds. In the rest of the paper, we assume vector-valued node features, a straightforward extension.

Here, w_{ij} denotes the *adjacency* of $\mathcal{G}$: $w_{ij} = w_{ji} = 1$ iff $(i,j) \in \mathcal{E}$. We tacitly assume edge fields to be *alternating*, so $\mathscr{X}_{ji} = -\mathscr{X}_{ij}$, and no self-edges, so $(i,i) \notin \mathcal{E}$. The gradient $(\nabla \mathbf{x})_{ij} = x_j - x_i$ assigns the edge $(i,j) \in \mathcal{E}$ the difference of its endpoint features and is alternating by definition. Similarly, the divergence $(\mathrm{div}(\mathscr{X}))_i$ assigns the node i the sum of the features of all edges it shares:

$$(\mathrm{div}(\mathscr{X}))_i = \sum_{j:(i,j)\in\mathcal{E}} \mathscr{X}_{ij} = \sum_{j=1}^{n} w_{ij}\mathscr{X}_{ij}$$

The two operators are adjoint, $\langle\langle \nabla \mathbf{x}, \mathscr{X} \rangle\rangle = \langle \mathbf{x}, \mathrm{div}(\mathscr{X}) \rangle$.

We consider the following diffusion equation on the graph

$$\frac{\partial \mathbf{x}(t)}{\partial t} = \mathrm{div}[\mathbf{G}(\mathbf{x}(t), t)\nabla \mathbf{x}(t)] \tag{1}$$

with an initial condition $\mathbf{x}(0)$. Here we denote by $\mathbf{G} = \mathrm{diag}(a(x_i(t), x_j(t), t))$ an $e \times e$ diagonal matrix and a is some function determining the similarity between nodes i and j. While in general $a(x_i, x_j, t)$ can be time-dependent, we will assume $a = a(x_i, x_j)$ for the sake of simplicity. Plugging in the expressions of ∇ and div, we get

$$\frac{\partial}{\partial t}\mathbf{x}(t) = (\mathbf{A}(\mathbf{x}(t)) - \mathbf{I})\mathbf{x}(t) = \bar{\mathbf{A}}(\mathbf{x}(t))\mathbf{x}(t) \tag{2}$$

where $\mathbf{A}(\mathbf{x}) = (a(x_i, x_j))$ is the $n \times n$ *attention matrix* with the same structure as the adjacency of the graph (we assume $a_{ij} = 0$ if $(i,j) \notin \mathcal{E}$). Note that in the setting when $\mathbf{A}(\mathbf{x}(t)) = \mathbf{A}$ we get a linear diffusion equation that can be solved analytically as $\mathbf{x}(t) = e^{\bar{\mathbf{A}}t}\mathbf{x}(0)$.

3.2. Properties of the graph diffusion equation

Differential equation stability is closely related to the concept of robustness in machine learning; changes in model outputs should be small under small changes in inputs. Formally, a solution $\mathbf{x}(t)$ of the PDE is said to be stable, if given any $\epsilon > 0$ there exists $\delta > 0$ such that for any solution $\hat{\mathbf{x}}(t)$, such that $|\mathbf{x}(0) - \hat{\mathbf{x}}(0)| \leq \delta$, it is also the case that $|\mathbf{x}(t) - \hat{\mathbf{x}}(t)| \leq \epsilon$ for all $t \geq 0$.

In the linear case, it is sufficient to show that the eigenvalues of $\bar{\mathbf{A}}$ are non-positive (see Appendix D for proof) For the general nonlinear case, we show

$$\max_i x_i(0) \geq x_i(t) \geq \min_i x_i(0) \quad \forall t \geq 0, \tag{3}$$

which follows from (i) the function $\bar{A}(\mathbf{x})\mathbf{x}$ being continuous in $\mathbf{x}$, (ii) the largest component of $\mathbf{x}(t)$ not increasing in time, and (iii) the smallest component is not decreasing in time.

Condition (i) holds as $\bar{\mathbf{A}}$ is a composition of Lipschitz-continuous functions (cf. equation (10)). Defining indices

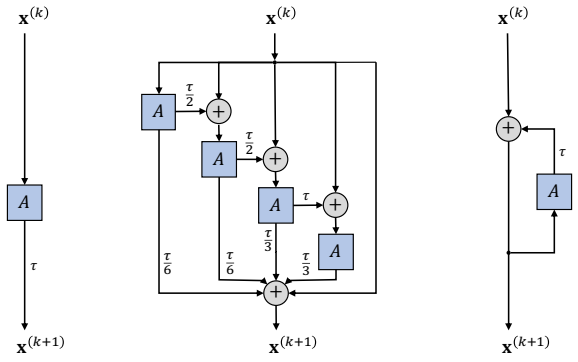

Figure 1. Block diagrams of (left to right) explicit Euler, 4th order Runge-Kutta, and implicit Euler schemes.

$k = \arg\max_i x_i$ and $l = \arg\min_i x_i$ we have

$$\frac{\partial x_k}{\partial t} = \sum_j \bar{a}_{kj}(x)x_j \leq x_k \sum_j \bar{a}_{kj} = 0 \tag{4}$$

$$\frac{\partial x_l}{\partial t} = \sum_j \bar{a}_{lj}(x)x_j \geq x_l \sum_j \bar{a}_{lj} = 0 \tag{5}$$

since $\mathbf{A}$ is right stochastic, which proves (ii) and (iii).

Furthermore, the derivative $\frac{\partial}{\partial x}\mathbf{A}(\mathbf{x})$ is Lipschitz-continuous (from the definition of the attention function we use), Taken together with continuity in time, the requirements of Picard-Lindelöf are satisfied and our PDE is also well posed.

3.3. Solving the graph diffusion equation

There are a wide range of numerical techniques for solving nonlinear diffusion equations. Our method most resembles the Method of Lines (MOL) where a finite difference method discretises the spatial derivatives, leaving a linear system of ODEs on the temporal axis that can be solved with numerical integrators. On a graph, the spatial operators are already discrete and follow the structure of the input graph; nevertheless, we show that different structures can be used, thus decoupling the input and computational graph.

For temporal discretisation, there exist two main schemes: *explicit* and *implicit*. Furthermore, we can distinguish between *single-step* and *multi-step* schemes; the latter use multiple function evaluations at different times to compute the next iterate (see Figure 1).

Explicit schemes. The simplest way to discretise Equation (1) is using the forward time difference:

$$\frac{x_i^{(k+1)} - x_i^{(k)}}{\tau} = \sum_{j:(i,j)\in\mathcal{E}} a\left(x_i^{(k)}, x_j^{(k)}\right)(x_j^{(k)} - x_i^{(k)}), \tag{6}$$

where k denotes the discrete time index (iteration), τ is the time step (discretisation parameter), and a is assumed to be

normalised, $\sum_j a(x_i^{(k)}, x_j^{(k)}) = 1$. Rewriting compactly in matrix-vector form, $\frac{\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}}{\tau} = \left(\mathbf{A}(\mathbf{x}^{(k)}) - I\right)\mathbf{x}^{(k)} = \bar{\mathbf{A}}(\mathbf{x}^{(k)})\mathbf{x}^{(k)}$, leads to the *explicit* or *forward Euler scheme* (Figure 1, left):

$$\mathbf{x}^{(k+1)} = \left(\mathbf{I} + \tau\bar{\mathbf{A}}(\mathbf{x}^{(k)})\right)\mathbf{x}^{(k)} = \mathbf{Q}^{(k)}\mathbf{x}^{(k)}, \quad (7)$$

where for $a_{ij}^{(k)} = a(x_i^{(k)}, x_j^{(k)})$, the matrix $\mathbf{Q}^{(k)}$ is given by $q_{ii}^{(k)} = 1 - \tau \sum_{\ell:(i,\ell)\in\mathcal{E}} a_{i\ell}^{(k)}$, $q_{ij}^{(k)} = \tau a_{ij}^{(k)}$ if $(i,j) \in \mathcal{E}$, and $q_{ij}^{(k)} = 0$ otherwise. This scheme is called explicit because the update $\mathbf{x}^{(k+1)}$ is deduced from $\mathbf{x}^{(k)}$ directly by the application of the diffusion operator $\mathbf{Q}^{(k)}$. The solution to the diffusion equation is computed by applying the scheme (7) multiple times in sequence, starting from some initial $\mathbf{x}^{(0)}$.

Implicit schemes use a backward time difference, $\frac{\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}}{\tau} = \bar{\mathbf{A}}\left(\mathbf{x}^{(k)}\right)\mathbf{x}^{(k+1)}$, which leads to the (semi-)*implicit* scheme (Figure 1, right):

$$\left(\mathbf{I} - \tau\bar{A}(\mathbf{x}^{(k)})\right)\mathbf{x}^{(k+1)} = \mathbf{B}(\mathbf{x}^{(k)})\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \quad (8)$$

This scheme is called (semi-)implicit because it requires solving a linear system in order to compute the update $\mathbf{x}^{(k+1)}$ from $\mathbf{x}^{(k)}$, amounting to the inversion of $\mathbf{B}$. The efficiency of this step is crucially dependent on the structure of $\mathbf{B}$ — for example, on grids, this matrix has a multi-diagonal structure, allowing $\mathcal{O}(n)$ inversion that was heavily exploited in PDE-based image processing applications (Weickert, 1997). In general, exact inversion is replaced with a few iterations of a linear solver.

Stability There exists a tradeoff between the number of iterations of the scheme K and the time step size τ. At the same time, the step size τ must be chosen in a way that guarantees that the scheme is stable. We summarise the stability results in the following theorems and provide additional details and proofs in Appendix D.

Theorem 1. *The explicit scheme (7) is stable for* $0 < \tau < 1$.

Theorem 2. *The implicit scheme (8) is unconditionally stable for any* $\tau > 0$.

Multi-step schemes use intermediate fractional time steps to obtain a higher-order numerical approximation, reusing the calculations for efficiency. Runge-Kutta (Figure 1, center) is among the most common multi-step schemes. General linear multi-step methods calculate the subsequent iterate using a linear combination of previous iterates of the form,

$$\sum_{j=0}^{s} \alpha_j \mathbf{x}^{(k+j)} = \tau \sum_{j=0}^{s} \beta_j \bar{\mathbf{A}}(\mathbf{x}^{(k+j)})\mathbf{x}^{(k+j)}. \quad (9)$$

and can be explicit or implicit depending s and $\{\alpha_j, \beta_j\}$.

The (explicit) Adams–Bashford and (implicit) Adams–Moulton methods are classes of linear multi-step methods that set $\alpha_{s-1} = -1$ and $\alpha_{s-2} = \ldots = \alpha_0 = 0$. For both, the $\{\beta_j\}$ coefficients are solved for by interpolating the dynamics function at the points of the previous solutions, $\mathbf{x}^{(k+j)}$ with a polynomial of order highest order possible using the Lagrange formula and substituting this into the integral form of the ODE. The methods differ in that Adams–Moulton interpolates through $\mathbf{x}^{(k+s)}$ and is consequently implicit whereas the Adams–Bashford methods do not. For Adams–Moulton methods, the implicit equations can be solved by Newton's method. Alternatively, one can use the predictor-corrector algorithm, which in this case takes an initial step with the explicit Adams–Bash method then multiple steps of Adams–Moulton, replacing the unknown $\mathbf{x}^{(k+s)}$ with the solution from the previous iteration, repeating until the difference between adjacent solutions is less than some threshold. In our experiments, we use fourth-order methods, $s = 4$. Additional details of multi-step schemes are provided in Appendix F.

Adaptive step size Adaptive step size solvers estimate the error in each iteration, which is then compared to an error tolerance; the step size is adapted to either increase or reduce the error. The error is estimated by comparing two methods, one with order p and one with order $p-1$. They are interwoven, i.e., they have common intermediate steps. As a result, estimating the error has little or negligible computational cost compared to a step with the higher-order method. Further details are given in Appendix F.

3.4. Connection to existing architectures

Many GNN architectures can be formalised as a discretisation scheme of (1). The discrete time index k corresponds to a (convolutional) layer of the graph neural network. Running the diffusion for multiple iterations thus amounts to applying a GNN layer multiple times. In the diffusion formalism, the time parameter t acts as a continuous analogy of the layers, in the spirit of Neural ODEs (Chen et al., 2018). This interpretation allows us to exploit more efficient numerical schemes and analyze the stability and convergence of the diffusion process.

The vast majority of GNN architectures are explicit single-step schemes of the form (7). For example, Equation (6) corresponds to the update formula of GAT (Veličković et al., 2018) with residual connection, assuming a is a learnable attention function and no non-linearity is used between the layers. Our choice of a time-independent attention function in the experiments in this paper amounts to all the layers *sharing the same parameters*. We will show that this is actually an advantage, as our models will be significantly

more lightweight and less prone to overfitting.

The diffusion equation is a PDE, with temporal and spatial components. In the graph setting, the former is continuous while the latter is discrete. Thus, the diffusion operator $\mathbf{Q}$ inherits the structure of the adjacency of the input graph. However, it is possible to consider the graph as a discretisation of a continuous object and thus regard the graph diffusion operator as a discrete derivative. In the same way that different discretisations of continuous derivatives with different support can be chosen, we can *rewire* the graph and make the structure of $\mathbf{Q}$ different from the input one and possibly *learnable*. Multiple GNN architecture *de facto* use a different computational graph from the input one, whether for reasons of scalability (e.g. sampling used in GraphSAGE (Hamilton et al., 2017)), denoising the input graph (Klicpera et al., 2019), or avoiding bottlenecks (Alon & Yahav, 2021). We argue that additional reasons are numerical convenience, to produce diffusion operators that are e.g. friendlier for matrix inversion.

In the following, we also show that the use of more efficient multi-step explicit schemes as well as unconditionally stable implicit schemes offers significant performance advantages. In particular, implicit schemes of the form (8) can be interpreted as multi-hop diffusion operators, since the inverse of $\mathbf{B}$ is typically dense (unlike $\mathbf{Q}$ in the explicit scheme (7) that has the same sparsity structure of the 1-hop adjacency matrix of the graph).

4. Graph Neural Diffusion

We now describe Graph Neural Diffusion (GRAND), a new class of GNN architectures derived from the graph diffusion formalism. We assume a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes and d-dimensional node-wise features represented as a matrix $\mathbf{X}_{\text{in}}$. GRAND architectures implement the learnable encoder/decoder functions ϕ, ψ and a learnable graph diffusion process, to produce node embeddings $\mathbf{Y} = \psi(\mathbf{X}(T))$,

$$\mathbf{X}(T) = \mathbf{X}(0) + \int_0^T \frac{\partial \mathbf{X}(t)}{\partial t} dt, \qquad \mathbf{X}(0) = \phi(\mathbf{X}_{\text{in}})$$

$\frac{\partial \mathbf{X}(t)}{\partial t}$ is given by the graph diffusion equation (1). Different GRAND architectures amount to the choice of the learnable diffusivity function $\mathbf{G}$ and spatial/temporal discretisations of equation (1).

The diffusivity is modelled with an attention function $a(.,.)$. Empirically, scaled dot product attention (Vaswani et al., 2017) outperforms the Bahdanau et al. attention used in GAT (Veličković et al., 2018). The scaled dot product attention is given by

$$a(\mathbf{X}_i, \mathbf{X}_j) = \text{softmax}\left(\frac{(\mathbf{W}_K \mathbf{X}_i)^\top \mathbf{W}_Q \mathbf{X}_j}{d_k}\right), \quad (10)$$

where $\mathbf{W}_K$ and $\mathbf{W}_Q$ are learned matrices, and d_k is a hyperparameter determining the dimension of W_k. We use multi-head attention which is useful to stabilise the learning (Veličković et al., 2018; Vaswani et al., 2017) by taking the expectation, $\mathbf{A}(\mathbf{X}) = \frac{1}{h}\sum_h \mathbf{A}^h(\mathbf{X})$. The attention weight matrix $\mathbf{A} = (a(\mathbf{X}_i, \mathbf{X}_j))$ is right-stochastic, allowing equation (12) to be written as

$$\frac{\partial}{\partial t}\mathbf{X} = (\mathbf{A}(\mathbf{X}) - \mathbf{I})\mathbf{X} = \bar{\mathbf{A}}(\mathbf{X})\mathbf{X} \quad (11)$$

As discussed in Section 3, a broad range of discretisations are possible. Temporal discretisations amount to the choice of numerical scheme, which can use either fixed or adaptive step sizes and be either explicit or implicit. Time forms a continuous analogy to the layer index, where each *layer* corresponds to an iteration of the solver. When using adaptive time step solvers, the number of layers is not specified a-priori. Explicit schemes use residual structures (e.g. Figure1, left and middle) that are usually more complex than those employed in resnets and which follow directly from rigorous numerical stability results (see Appendix F). Implicit numerical schemes offer a natural way of trading off *depth* and *width* (spatial support of the diffusion kernel). In Section 6.3 we explore several temporal discretisations using various numerical integrators.

Spatial discretisation amounts to modifying the given graph, or building one in settings where no graph is given and the data can be assumed to lie in some feature space or on a continuous manifold. When the input graph is given, which is the case in our experimental sections, we can rewire the given graph and use a different edge set in the diffusion equation.

While in general equation (11) is nonlinear due to the dependence of $\mathbf{A}$ on $\mathbf{X}$, it becomes linear if the attention weights are fixed inside the integral, $\bar{\mathbf{A}}(\mathbf{X}(t)) = \bar{\mathbf{A}}$ (note that $\mathbf{A}$ is still parametric and learnable, but does not change throughout the diffusion process). In this case, equation (11) can be solved analytically as $\mathbf{X}(t) = e^{\bar{\mathbf{A}}t}\mathbf{X}(0)$. As $\bar{\mathbf{A}}$ is a form of normalised Laplacian, all eigenvalues are non-positive and the steady state solution is given by the dominating eigenvector, which is the degree vector. However, as $\bar{\mathbf{A}}$ is learned, this limitation is not severe as the system can be (and in practice is) degenerate; the graph becomes (approximately) disconnected, with connected components permitted to have unique steady state solutions. We call this model **GRAND-l** for linear to distinguish it from the more general **GRAND-nl** for non-linear. The final variant is **GRAND-nl-rw** (nonlinear with rewiring), where rewiring is performed via a two step process: as a preprocessing step, the graph is densified

using diffusion weights as in (Klicpera et al., 2019), and then at runtime the subset of edges to use is learned based on attention weights. Equation (1) becomes:

$$\frac{\partial \mathbf{X}_i(t)}{\partial t} = \sum_{j:(i,j)\in\mathcal{E}'} a\left(\mathbf{X}_i(t), \mathbf{X}_j(t)\right)\left(\mathbf{X}_j(t) - \mathbf{X}_i(t)\right) \tag{12}$$

where $\mathcal{E}' = \{(i,j) : (i,j) \in \mathcal{E} \text{ and } a_{ij} > \rho\}$ with some threshold value ρ, is the 'rewired' edge set, which may now contain self-loops. While a changes throughout the diffusion process, rewiring is only performed at the start of the epoch based on features at $t = 0$.

GRAND shares parameters across layer/iteration and is thus more data-efficient than conventional GNNs. The full training objective is given in Appendix C. To update the parameters we either backpropagate through the computational graph of the numerical integrator or, when memory is constrained, use Pontryagin's maximum principle (Pontryagin, 2018).

5. Related work

Image processing and graphics. During the 1990s-2000s, a vast amount of image processing literature exploited the formalism of diffusion equations (Weickert, 1998), starting with the seminal work of Perona & Malik (1990). Sochen et al. (1998) developed a differential geometric framework ('Beltrami flow') considering the evolution of images represented as embedded manifolds. The related bilateral (Tomasi & Manduchi, 1998) and non-local means (Buades et al., 2005) filters, together with efficient numerical techniques (Weickert, 1997; Durand & Dorsey, 2002), have popularised these ideas in the image processing community. PDE-based methods were also used for low-level tasks such as image segmentation (Caselles et al., 1997; Chan & Vese, 2001) and inpainting (Bertalmio et al., 2000).

In computer graphics, solutions of non-Euclidean diffusion equations were studied as *heat kernel signature* (Sun et al., 2009; Bronstein & Kokkinos, 2010) local shape descriptors related to the Gaussian curvature. Non-Euclidean diffusion equations can be solved by using the Laplacian eigenvectors as the analogy of Fourier basis and the corresponding eigenvalues as frequencies. The solution can be represented as a spectral transfer function (Patané, 2016), which can also be learned (Litman & Bronstein, 2013). The non-Euclidean Fourier approach was exploited in the early work on deep learning on graphs (Henaff et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2017; Levie et al., 2017).

Graph diffusion processes techniques such as eigenmaps and diffusion maps (Coifman et al., 2005; Belkin & Niyogi, 2003) use linear diffusion PDEs with closed form solutions expressed through Laplacian eigenvectors. Diffusion-Convolutional Neural Networks (Atwood & Towsley, 2016) employ a diffusion operator for graph convolutions and LanczosNet (Liao et al., 2019) uses a polynomial filter on the Laplacian matrix, which corresponds to a multi-scale linear diffusion PDE. Adaptive LanczosNet (Liao et al., 2019) additionally allows learning the filters to reweight the graph using a kernel. The use of a polynomial filter approximates the solution of the PDE, and the diffusion is linear with a fixed operator.

Neural ODEs. Chen et al. (2018) introduced neural ODEs. Many follow-up works explored augmentation (Dupont et al., 2019) and regularization (Finlay et al., 2020) and provided extensions into new domains such as stochastic (Liu et al., 2019) differential equations. Neural ODEs have also been applied to GNNs: Avelar et al. (2019) model continuous residual layers with GCN. Poli et al. (2019) propose approaches for static and dynamic graphs using GCN to model static graphs and a hybrid approach where the latent state evolves continuously between RNN steps for dynamic graphs. Xhonneux et al. (2020) address continuous message passing. Their model is a solution to the constant linear diffusion PDE. Unlike most GNN, it scales with the size of the graph having $\mathcal{O}(n)$ parameters. Continuous GNNs were also explored by Gu et al. (2020) who, similarly to (Scarselli et al., 2009), addressed the solutions of fixed point equations. Ordinary Differential Equations on Graph Networks (GODE)(Zhuang et al., 2020) approach the problem using the technique of invertible ResNets. Finally, Sanchez-Gonzalez et al. (2019) used graph-based ODEs to generate physics simulations.

Neural PDEs. Using deep learning to solve PDEs was explored by Raissi et al. (2017). Neural networks appeared in (Li et al., 2020a) 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 (Li et al., 2020b). Belbute-Peres et al. (2020) 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 discretising PDEs as a principle to understand and design GNNs.

6. Results

We design experiments to answer the following: Are GNNs derived from the diffusion PDE competitive with existing popular methods? Can we address the problem of building deep graph neural networks? Under which conditions can implicit methods yield more efficient GNNs than explicit methods? Additional implementation details are provided in the Appendix.

GRAND is implemented in PyTorch (Paszke et al., 2019), using PyTorch geometric (Fey & Lenssen, 2019) and torchdiffeq (Chen et al., 2018). Code and instructions to reproduce the experiments are available at https://github.com/twitter-research/graph-neural-pde.

6.1. Node classification benchmarks

We measure the performance of GRAND on a range of common node classification benchmarks.

Methods We compare to four of the most popular GNN architectures: Graph Convolutional Network (GCN) (Kipf & Welling, 2017), Graph Attention Network (GAT) (Veličković et al., 2018), Mixture Model Networks (Monti et al., 2017) and GraphSage (Hamilton et al., 2017). Additionally we compare to recent ODE-based GNN models, Continuous Graph Neural Networks (CGNN) (Xhonneux et al., 2020), Graph Neural Ordinary Differential Equations (GDE) (Poli et al., 2019), and Ordinary Differential Equations on Graphs (GODE) (Zhuang et al., 2020) and two versions of LanczosNet (Liao et al., 2019) which approximate solutions to a linear diffusion PDE.

We study three variants of GRAND: linear, nonlinear and nonlinear with graph rewiring. In the GRAND-l, the attention weights are constant throughout the integration, producing a coupled system of linear ODEs. In GRAND-nl, the attention weights are updated at each step of the numerical integration. In both cases, the given graph is used as the spatial discretisation of the diffusion operator. In GRAND-nl-rw, the graph is rewired after each backward pass by thresholding the diffusivity attention mechanism. The rewiring is held constant throughout the integration.

Datasets We report results for the most widely used citation networks Cora (McCallum et al., 2000), Citeseer (Sen et al., 2008), Pubmed (Namata et al., 2012). These datasets contain fixed splits that are often used, which we include for direct comparison in Table 1. To address the limitations of this evaluation methodology (Shchur et al., 2018), we also report results for all datasets using 100 random splits with 20 random initializations. Additional datasets are the coauthor graph CoauthorCS (Shchur et al., 2018), the Amazon co-purchasing graphs Computer and Photo (McAuley et al., 2015), and the OGB arxiv dataset (Hu et al., 2020). In all cases, we use the largest connected component. Dataset statistics are included in Appendix A.

Experimental setup We follow the experimental methodology described in (Shchur et al., 2018) using 20 random weight initializations for datasets with fixed Planetoid splits and 100 random splits for the remaining datasets. Where available, results from (Shchur et al., 2018) were used. 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 (Liaw et al., 2018) with a thousand random trials using an asynchronous hyperband scheduler with a grace period of ten epochs and a 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 smaller datasets (Cora, Citeseer) we used the Anode augmentation scheme (Dupont et al., 2019) to stabilise training. The ogb-arxiv dataset used the Runge-Kutta method, for all others Dormand-Prince was used. For the larger datasets, we used kinetic energy and Jacobian regularization (Finlay et al., 2020; Kelly et al., 2020). 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 (Finlay et al., 2020).

Complexity For all datasets we use the adjoint method described in (Chen et al., 2018). The space complexity is dominated by evaluating Equation (10) over edges and is $\mathcal{O}(|\mathcal{E}'|d)$ where $\mathcal{E}'$ is the edge set following rewiring and d is dimension of features. The runtime complexity is $\mathcal{O}(|\mathcal{E}'|d)(E_b + E_f)$, split between the forward and backward pass and can be dominated by either depending on the number of function evaluations (E_b, E_f).

Number of parameters In traditional GNNs there is a linear relationship between the number of parameters and depth. Conversely, GRAND *shares parameters across layers* (due to our choice of a time-independent attention) and consequently, requires significantly less parameters than competing methods, while achieving on par or superior performance. The versions of GCN, SAGE and GAT used for the ogb-arxiv results required 143K, 219K and 1.63M parameters respectively, while our model only 70K.

Performance Tables 1–2 summarise the results of our experiments. GRAND variants consistently perform among the best methods, achieving first place on all but one dataset, where it is second. On ogb-arxiv, our results are slightly inferior to the best-performing GAT, which, however, requires 20 times as many parameters.

6.2. Depth

To demonstrate that our model solves the oversmoothing problem and performs well with many layers, we performed an experiment using the RK4 fixed step-size solver (with

Planetoid splits	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
GRAND-l (ours)	**84.7 ± 0.6**	**73.3 ± 0.4**	**80.4 ± 0.4**
GRAND-nl (ours)	**83.6 ± 0.5**	70.8 ± 1.1	79.7 ± 0.3
GRAND-nl-rw (ours)	82.9 ± 0.7	**73.6 ± 0.3**	**81.0 ± 0.4**

Table 1. Test accuracy and std for 20 random initializations using the original Planetoid train-val-test splits. *GODE and GDE comprises six and three separate models respectively. For each dataset we present the best performing variant of GODE and GDE. †Results obtained running the authors' code with the hyperparameters given in their paper using Pytorch Geometric data readers.

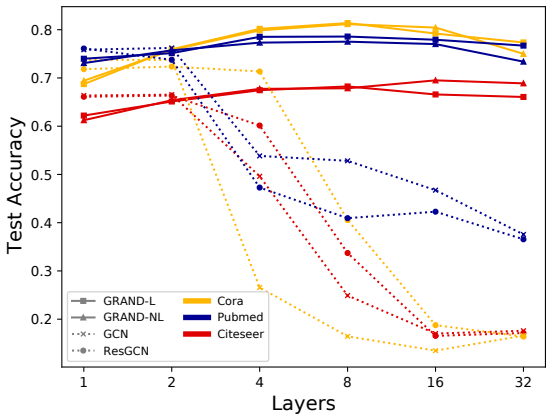

Figure 2. Performance of architectures of different depth.

step size $\tau = 1.0$), varying the integration time T while holding the other hyper-parameters fixed. This effectively produces architectures of varying depth. Figure 2 shows that compared to GCN and a GCN with residual connections, our model maintains performance as the layers increase whilst the baselines degrade by 50% after 4 layers.

6.3. Choice of discretisation scheme

We investigated the stability of explicit numerical schemes with a fixed step size and the tradeoff between step size and computational time for an equivalent implicit numerical scheme. We compared these to the Dormand–Prince adaptive step size scheme (DOPRI5).

Method choice We ran GRAND on Cora with the explicit Adams–Bashford method, an implicit Adams–Moulton method with a predictor-corrector algorithm, and the adaptive Runge-Kutta 4(5) method (see Figure 3, left), varying the step sizes for the two fixed-step size methods. We observe that the explicit Adams method is unstable for all but a small step size of $\tau = 0.005$, while the implicit Adams method is stable for all step sizes. Moreover, in this case the implicit method converges to the solution faster than a state-of-the-art adaptive step size solver for large enough step size. We note, however, that this may not always be the case. As the step size is increased, the implicit method can take fewer steps. However, as the step size is increased the implicit equations become more difficult to solve and require more iterations of the algorithm used to solve them.

Graph rewiring In this experiment, we rewired the Cora graph using the method of Klicpera et al. (2019), keeping the largest K coefficients for each node. We varied K to explore the tradeoff between sparsity, computation time, and accuracy (see Figure 3, right). As the graph is made sparser (K decreases), all methods become faster. The accuracy converges to similar values until the graph is so sparse that the flow of information is impeded ($K < 8$). We observe that for implicit solvers the benefit of sparsification is independent of the step size, and both can be combined somewhat to decrease the time per epoch without effecting accuracy. We hypothesize that, in general, a sparser graph is particularly desirable for implicit solvers since it may reduce the difficulty of solving the implicit equations (less iterations until convergence). A final observation is that we can draw a diagram similar to Figure 1 for the computational graph of the Adams–Moulton method with predictor–corrector steps; it is redolent of an RNN with adaptive computation time (Graves, 2016), where the stopping rule is deterministic rather than learnt (that is, continue unrolling the RNN until the difference between outputs is below a cutoff).

6.4. Diffusion on MNIST Image Data Experiments

We performed an experiment to illustrate the learned diffusion characteristics of GRAND. MNIST pixel data was used to construct a superpixel representation (Achanta et al., 2012) and adjacent patches were joined with edges, binary pixel labels were applied (number or background) with a 50% training mask. We evolved both GRAND-nl and a constant Laplacian diffusion model for $T = 4.8$ and $\tau = 0.8$, equating to a 6 layer GNN. We show the attention weights by the colour and thickness of the edges. Figure 4 shows Non-linear GRAND performs edge detection weighting diffusion within a class boundary in a way that preserves the image after diffusion. The Laplacian diffusion is unable to preserve the features of the original image.

7. Conclusion

We presented a new class of graph neural network called Graph Neural Diffusion (GRAND), based on the discretisation of diffusion PDEs on graphs. Our framework allows leveraging vast literature on PDEs relating to discrete

Random splits	CORA	CiteSeer	PubMed	Coathor 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	**72.17 ± 0.33**
GAT	81.8 ± 1.3	71.4 ± 1.9	**78.7 ± 2.3**	90.5 ± 0.6	78.0 ± 19.0	85.7 ± 20.3	**73.65 ± 0.11**†
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	N/A
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	N/A
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	71.39 ± 0.16
GS-maxpool	76.6 ± 1.9	67.5 ± 2.3	76.1 ± 2.3	85.0 ± 1.1	N/A	90.4 ± 1.3	N/A
CGNN	81.4 ± 1.6	66.9 ± 1.8	66.6 ± 4.4	**92.3 ± 0.2**	80.29 ±2.0	91.39 ± 1.5	58.70 ± 2.5
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
GRAND-l (ours)	**83.6 ± 1.0**	**73.4 ± 0.5**	**78.8 ± 1.7**	**92.9 ± 0.4**	**83.7 ± 1.2**	**92.3 ± 0.9**	71.87 ± 0.17
GRAND-nl (ours)	**82.3 ± 1.6**	70.9 ± 1.0	77.5 ± 1.8	**92.4 ± 0.3**	82.4 ± 2.1	**92.4 ± 0.8**	71.2 ± 0.2
GRAND-nl-rw (ours)	**83.3 ± 1.3**	**74.1 ± 1.7**	78.1 ± 2.1	91.3 ± 0.7	**85.8 ± 1.5**	**92.5 ± 1.0**	**72.23 ± 0.20**

Table 2. Test accuracy and std for 20 random initializations and 100 random train-val-test splits. *Using labels. †using 1.5M parameters.

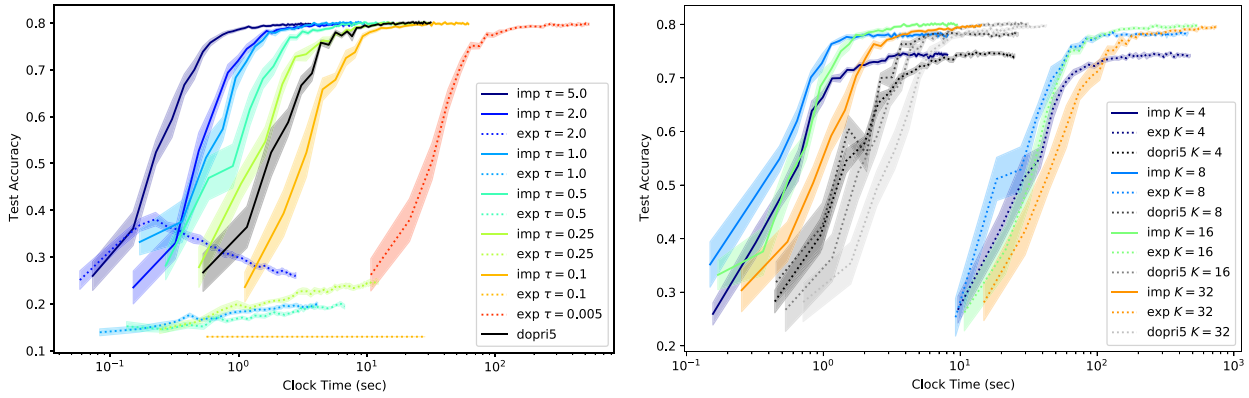

Figure 3. Performance of different solvers. Left: Test accuracy on plain Cora varying the step size, comparing explicit Adams–Bashford, implicit Adams–Moulton, and adaptive Runge-Kutta 4(5). We observe that the explicit fixed step size scheme is unstable for all but a small step size whereas the implicit scheme is stable for all step sizes tried. For a large step size, the implicit scheme is faster than a state-of-the-art explicit scheme with adaptive step size. Right: Diffusion-rewired Cora varying the sparsity of the graph by keeping the largest K coefficients for each node. Explicit Adams–Bashford has step size $\tau = 0.005$, and implicit Adams–Moulton $\tau = 1.0$. We observe a trade-off between sparsity, speed, and accuracy.

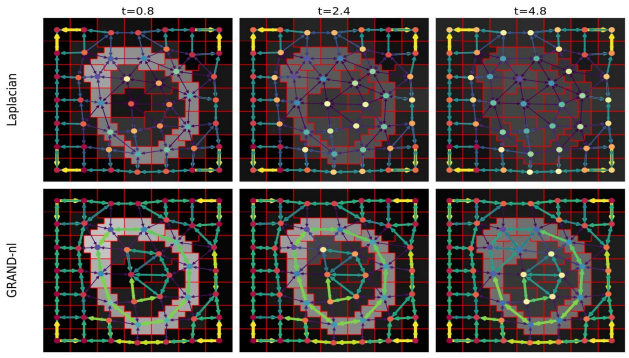

Figure 4. Illustration of the effect of attention weights on pixel diffusion

temporal and spatial operators and stability, and provides a blueprint for a principled design of new graph learning architectures. We show that appropriate choice of discretisation and numerical schemes in GRAND allows us to train very deep graph neural networks and results in superior performance on popular benchmarks.

Limitations We intentionally considered a form of the diffusion equation that is easier to treat mathematically. Our model is currently limited to learn only functions of the form $\frac{\partial \mathbf{x}}{\partial t} = f(\mathbf{x}(t), t, \theta)$, with an 'attentional' structure of f. This imposes two limitations that are not present in discrete neural networks: first, the size of the hidden state vector must be constant for all layers (a usual situation in GNNs), and second, the same set of parameters θ must be used for all layers. The later constraint comes as an advantage, allowing our model to use $10-20$ times less parameters than the top performing model on ogbn-arxiv. In future work, we intend to overcome these limitation by introducing a $\theta = \theta(t)$ as described in (Queiruga et al., 2020; Zhang et al., 2019). We will also consider more general nonlinear diffusion equations that result in message passing 'flavors' of GNN architectures.

Acknowledgements MB is supported in part by ERC Consolidator grant No. 724228 (LEMAN). We would like to thank Gabriele Corso, Nils Hammerla and our reviewers for many helpful suggestions that improved this manuscript.

A. Datasets

The statistics for the largest connected components of the experimental datasets are given in Table 3.

B. Diffusivity Formulations

GRAND can use any right stochastic attention matrix. We performed experiments with the multiheaded Bahdanau formulation (Bahdanau et al.) of attention, which has previously been applied to graphs in (Veličković et al., 2018)

$$a(x_i, x_j) = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[W\mathbf{x}_i \| W\mathbf{x}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[W\mathbf{x}_i \| W\mathbf{x}_k\right]\right)\right)}, \tag{13}$$

where W and $\mathbf{a}$ are learned and $\|$ is the concatenation operator. However, for all datasets, the scaled dot product attention performed better. This may be because GAT relies on dropout. Dropout performs poorly inside adaptive timestep numerical ODE solvers as the stochasticity in the forward pass drives $\tau \to 0$.

C. Full Training Objective

The full training program optimises cross entropy loss

$$\mathcal{L}(\mathbf{Y}, \mathbf{T}) = H(\mathbf{Y}, \mathbf{T}) = \sum_{i=1}^{n} \mathbf{t}_i^T \log \mathbf{y}_i \tag{14}$$

where $\mathbf{t}_i \in \mathbb{R}^{d_{\text{class}}}$ is the one-hot truth vector of the i^{th} node with prediction

$$\mathbf{y}_i = \psi(\mathbf{x}_i(T)) \tag{15}$$

where $\psi : \mathbb{R}^d \to \mathbb{R}^{d_{\text{class}}}$ is a linear layer decoder of the terminal value of the evolutionary PDE

$$\mathbf{y}_i = D\mathbf{x}_i(T) + \mathbf{b}_d \tag{16}$$

$$= D\left(\mathbf{X}(0) + \int_0^T \frac{\partial \mathbf{X}(t)}{\partial t} dt\right) + \mathbf{b}_d \tag{17}$$

$$= D\left(\phi(\mathbf{X}_{\text{in}}) + \int_0^T \frac{\partial \mathbf{X}(t)}{\partial t} dt\right) + \mathbf{b}_d \tag{18}$$

with the initial condition given by the linear layer encoder $\phi : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^d$ of the input data

$$\mathbf{y}_i = \left(D\left(E(\mathbf{X}_{\text{in}}) + \mathbf{b}_e + \int_0^T \frac{\partial \mathbf{X}(t)}{\partial t} dt\right) + \mathbf{b}_d\right)_i$$

and $f(\mathbf{X}(t), t, \theta) = \frac{\partial \mathbf{X}(t)}{\partial t}$ is the system dynamics that we wish to learn. For the nonlinear version of GRAND this is

$$f = \frac{\partial}{\partial t}\mathbf{X}(t) = (\mathbf{A}(\mathbf{X}(t)) - \mathbf{I})\mathbf{X}(t) = \bar{\mathbf{A}}(\mathbf{X}(t))\mathbf{X}(t)$$

D. Stability

D.1. Stability of linear ODE

In the main paper we reported the linear GRAND $\dot{\mathbf{x}} = \bar{A}\mathbf{x}$ has solution

$$\mathbf{x}(t) = \mathbf{x}(0)e^{\bar{A}t}. \tag{19}$$

As $\bar{A}$ is not diagonal this matrix exponential is not analytically recoverable. Performing eigenvalue decomposition the solution is

$$\mathbf{x}(t) = \bar{T}e^{\bar{D}t}\bar{T}^{-1}\mathbf{x}(0). \tag{20}$$

Assuming $\bar{T}^{-1}$ exists, $\bar{T}$ has full rank and both are bounded, the test equation becomes

$$\mathbf{y}(t) = e^{\bar{D}t}\mathbf{y}(0) \tag{21}$$

where $\mathbf{y}(t) = \bar{T}\mathbf{x}(t)$. If $\mathbf{x}(t)$ and $\hat{\mathbf{x}}(t)$ are two solutions of the ODE then their projections in eigenspace are $\mathbf{y}(t)$ and $\hat{\mathbf{y}}(t)$. For each node i:

$$|y_i(t) - \hat{y}_i(t)| = \left|(y_i(0) - \hat{y}_i(0))e^{\bar{\lambda}_i t}\right| \tag{22}$$

$$= |y_i(0) - \hat{y}_i(0)|e^{\mathcal{R}e(\bar{\lambda}_i)t} \tag{23}$$

for this to converge as $t \to \infty$ we require $\mathcal{R}e(\bar{\lambda}_i) \leq 0\ \forall i$. As A is right stochastic the eigenvalues of $\bar{A} = A - I$ satisfy this property.

E. Numerical Schemes

E.1. Proof of theorem 1: Stability of explicit Euler

For linear GRAND with an Euler numerical integrator

$$\mathbf{x}^{(t+1)} = \left(I + \tau\bar{A}(\mathbf{x}^{(t)})\right)\mathbf{x}^{(t)} \tag{24}$$

$$= Q^{(t)}\mathbf{x}^{(t)}. \tag{25}$$

We require that the amplification factor $||Q^{(t)}|| < 1$. It is sufficient to show that $Q^{(t)}$ is a right stochastic matrix, which has the property that its spectral radius $\lambda_{\max} \leq 1$. Q is right stochastic if

1. $\sum_{j=1}^{N} q_{ij} = 1$
2. $q_{ij} > 0 \quad \forall i, j$

as A is right stochastic $\sum_j I_{ij} + \tau(A_{ij} - I_{ij}) = 1$ proving 1). As $a_{ij} = q_{ij}$ for $i \neq j$, to prove 2) it remains to show that $1 + \tau(a_{ii} - 1) > 0 \iff \tau < 1$.

Dataset	Type	Classes	Features	Nodes	Edges	Label rate
Cora	citation	7	1433	2485	5069	0.056
Citeseer	citation	6	3703	2120	3679	0.057
PubMed	citation	3	500	19717	44324	0.003
Coauthor CS	co-author	15	6805	18333	81894	0.016
Computers	co-purchase	10	767	13381	245778	0.015
Photos	co-purchase	8	745	7487	119043	0.021
OGB-Arxiv	citation	40	128	169343	1166243	1

Table 3. Dataset Statistics

E.2. Proof of theorem 2: Implicit methods

For implicit Euler

$$\dot{x}_n = \frac{x_n - x_{n-1}}{\tau} = f(x_n, t_n) \tag{26}$$

$$x_n = \tau f(x_n, t_n) + x_{n-1}, \tag{27}$$

incrementing the indices gives

$$x_{n+1} = \tau f(x_{n+1}, t_{n+1}) + x_n \tag{28}$$

and now, unlike the explicit case, x_{n+1} now appears on both sides of the equation. If f is linear

$$x_{n+1} = \tau \bar{A} x_{n+1} + x_n \tag{29}$$

$$x_{n+1} = (I - \tau \bar{A})^{-1} x_n = B^{-1} x_n = Q x_n, \tag{30}$$

and the matrix B must be inverted. The inverse exists as B is diagonally dominant

$$I_{ii} - \tau(A_{ii} - I_{ii}) > \tau \sum_{j \neq i} A_{ij} = \tau(1 - A_{ii}) \tag{31}$$

By considering the action of B on $w = (1, ..., 1)^T$ it is clear that $Bw = w \implies Qw = w \implies \sum_j Q_{ij} = 1$. As B is diagonally dominant it is irreducible and satisfies $B_{ij} \leq 0$ $i \neq j$ and $B_{ii} > 0$ giving $Q_{ij} > 0$ $\forall i, j$ (**?**) and Q is a Markov matrix with spectral radius bounded by unity and the implicit scheme is stable for all choices of τ.

F. General Multistep Methods

A general multistep method (combining both implicit and explicit methods) can be written as

$$x_{n+1} + \sum_{i=1}^{s} \alpha_i x_{n+1-i} = \tau \sum_{i=0}^{s} \beta_i f_{n+1-i}, \tag{32}$$

where $f = \dot{x}$. If $\beta_0 = 0$ then x_{n+1} only depends on terms up to n and the method is explicit.

F.1. Order

The order of a method gives the approximation error in terms of a Taylor series expansion. If p is the order, then the error is a single step $\propto \tau^{p+1}$ and the error in the entire interval $\propto \tau^p$. In practice the order of a numerical method can be determined by measuring how the error changes with step size for a known integral.

Order	1	2	3	4	5	6	7	8
Evals	1	2	3	4	6	7	9	11

Table 4. Function evaluations grow super-linearly with order after 4.

F.2. Butcher Tableau

The set of coefficients for each multi step method are given by the Butcher Tableau. The simple case of forward Euler has $\alpha_1 = -1$, $\beta_1 = 1$ with all other terms zero.

There is a law of diminishing return that relates the minimum number of function evaluations and the order of a higher order Runge-Kutta solver. Table 4 shows why the Runge-Kutta 4 method (RK4) is often regarded as the optimal trade-off between speed and accuracy for multi step solvers.

F.3. Runge-Kutta 4

For all experiments we find that Runge-Kutta 4 (or it's adaptive step size variants) outperforms lower order methods. The Runge-Kutta 4 method follows the schema: if $f(\mathbf{x}, t) = \bar{A}(\mathbf{x}_t)\mathbf{x}_t$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \frac{1}{6}\tau\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{33}$$

$$\mathbf{k}_1 = f(\mathbf{x}_t, t) \tag{34}$$

$$\mathbf{k}_2 = f(\mathbf{x}_t + \tau\mathbf{k}_1/2, t + \tau/2) \tag{35}$$

$$\mathbf{k}_3 = f(\mathbf{x}_t + \tau\mathbf{k}_2/2, t + \tau/2) \tag{36}$$

$$\mathbf{k}_4 = f(\mathbf{x}_t + \tau\mathbf{k}_3, t + \tau) \tag{37}$$

F.4. Adaptive Step Size

Adaptive step size solvers estimate the error in x_{n+1}, which is compared to an error tolerance. The error is estimated by comparing two methods, one with order p and one with order $p - 1$. They are interwoven, i.e., they have common

intermediate steps. As a result, estimating the error has little or negligible computational cost compared to a step with the higher-order method.

$$x^*_{n+1} = x_n + \tau \sum_{i=1}^{s} b^*_i k_i \tag{38}$$

where k_i are the same as for the higher-order method. Then the error is

$$e_{n+1} = x_{n+1} - x^*_{n+1} = \tau \sum_{i=1}^{s} (b_i - b^*_i) k_i. \tag{39}$$

The time step is increased if the error is below tolerance and decreased otherwise.

G. Adaptive step size implementation details

Most results presented used the adaptive step size solver Dormand-Prince5. Key to getting this to work well is setting appropriate tolerances for the step size. Adaptive step size ODE solvers require two tolerance parameters; the relative tolerance rtol and the absolute atol. Both are used to assess the new step size

$$etol = atol + rtol * \max(|x_0|, |x_1|), \tag{40}$$

where x_0 and x_1 are successive estimations of the new state. Dupont et al. (2019) speculate that ResNets can learn a richer class of functions than ODEs because "the error arising from discrete steps allows trajectories to cross". We find that increasing the estimation error is also helpful when learning continuous diffusion functions and use value of $rtol$ and $etal$ that are $\times 10 - \times 1000$ larger than the defaults. This both improves prediction accuracy and reduces the runtime.

In hyperparameter search $atol$ and $rtol$ were paired together using a tolerance scale variable ts such that $atol = ts \times 10^{-12}$ and $atol = ts^{-6}$.

When using the adjoint method to backpropagate derivatives, two separate ODEs are being solved. This requires separate tolerance scales, which may differ: the forward pass tolerance, ts, controls for how close the approximated ODE solution is compared to the true solution, while the backward pass tolerance, ts_{adj}, controls the accuracy of the computed gradient. The hyperparameter search includes both ts and ts_{adj}.

References

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, 2012. doi: 10.1109/TPAMI.2012.120.

Almeida, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proc Neural Networks*, 1987.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *NeurIPS*, pp. 1993–2001, 2016.

Avelar, P. H. C., Tavares, A. R., Gori, M., and Lamb, L. C. Discrete and continuous deep residual learning over graphs. 2019.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *ICLR*. ISBN 0147-006X (Print). doi: 10.1146/annurev.neuro.26.041002.131047.

Belbute-Peres, F. d. A., Economon, T., and Kolter, Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *ICML*. PMLR, 2020.

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. Image inpainting. In *Proc Computer Graphics and Interactive Techniques*, 2000.

Bronstein, M. M. and Kokkinos, I. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *CVPR*, 2010.

Buades, A., Coll, B., and Morel, J.-M. A non-local algorithm for image denoising. In *ICCV*, 2005.

Caselles, V., Kimmel, R., and Sapiro, G. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

Chan, T. F. and Vese, L. A. Active contours without edges. *IEEE Trans. Image Processing*, 10(2):266–277, 2001.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *NeurIPS*, pp. 6571–6583, 2018.

Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the national academy of sciences*, 102(21):7426–7431, 2005.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural ODEs. In *NeurIPS*, 2019.

Durand, F. and Dorsey, J. Fast bilateral filtering for the display of high-dynamic-range images. In *Proc Computer Graphics and Interactive Techniques*, 2002.

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

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. How to train your neural ode: The world of Jacobian and kinetic regularization. In *ICML*, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, 2017.

Graves, A. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016.

Gu, F., Chang, H., Zhu, W., Sojoudi, S., and Ghaoui, L. E. Implicit graph neural networks. *arXiv:2009.06211*, 2020.

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

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv:2005.00687*, 2020.

Kelly, J., Bettencourt, J., Johnson, M. J., and Duvenaud, D. Learning differential equations that are easy to solve. In *NeurIPS*, 2020.

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

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *NeurIPS*, volume 32, 2019.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv:1705.07664*, 2017.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Multipole graph neural operator for parametric partial differential equations. In *NeurIPS*, 2020b.

Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. S. Lanczosnet: Multi-scale deep graph convolutional networks. In *ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A research platform for distributed model selection and training. 2018.

Litman, R. and Bronstein, A. M. Learning spectral descriptors for deformable shape correspondence. *PAMI*, 36(1): 171–180, 2013.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. Neural SDE: Stabilizing neural ODE networks with stochastic noise. (2), 2019.

McAuley, J., Targett, C., Shi, Q., and Van Den Hengel, A. Image-based recommendations on styles and substitutes. In *Proceedings Information Retrieval*, 2015.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017. ISBN 978-1-5386-0457-1.

Namata, G., London, B., Getoor, L., Huang, B., and EDU, U. Query-driven active surveying for collective classification. In *Proceedings Mining and Learning with Graphs*, 2012.

NT, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. 2019.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

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

Perona, P. and Malik, J. Scale-space and edge detection using anisotropic diffusion. *PAMI*, 12(7):629–639, 1990.

Pineda, F. J. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229, 1987.

Poli, M., Massaroli, S., Park, J., Yamashita, A., Asama, H., and Park, J. Graph neural ordinary differential equations. pp. 6571–6583, 2019.

Pontryagin, L. S. *Mathematical theory of optimal processes*. Routledge, 2018.

Queiruga, A. F., Erichson, N. B., Taylor, D., and Mahoney, M. W. Continuous-in-depth neural networks. (2), 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations. (Part II), 2017.

Sanchez-Gonzalez, A., Bapst, V., Cranmer, K., and Battaglia, P. Hamiltonian graph networks with ODE integrators. 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Trans. Neural Networks*, 27(8):61–80, 2009.

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

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv:1811.05868*, 2018.

Sochen, N., Kimmel, R., and Malladi, R. A general framework for low level vision. *IEEE Trans. Image Processing*, 7(3):310–318, 1998.

Sun, J., Ovsjanikov, M., and Guibas, L. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

Tomasi, C. and Manduchi, R. Bilateral filtering for gray and color images. In *ICCV*, 1998.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, A., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS*, pp. 5998–6008, 2017.

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

Weickert, J. A review of nonlinear diffusion filtering. In *Proceedings ScaleSpace*. Springer, 1997.

Weickert, J. *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998.

Xhonneux, L.-p. A. C., Qu, M., and Tang, J. Continuous graph neural networks. In *ICML*, 2020.

Zhang, T., Yao, Z., Gholami, A., Keutzer, K., Gonzalez, J., Biros, G., and Mahoney, M. W. ANODEV2: A coupled neural ODE framework. In *NeurIPS*, 2019.

Zhuang, J., Dvornek, N., Li, X., and Duncan, J. S. Ordinary differential equations on graph networks. Technical Report 1, 2020.