Graph Neural Networks and Physics

Danielius Stanevicius

August 6, 2025

Contents

1 Introduction

Many real-world systems can be naturally represented as graphs. This structure appears in numerous domains, including social networks, biological systems, and transportation infrastructures.

Graph Neural Networks have emerged as a leading approach to learning on graph-structured data. By using the connectivity and feature information encoded in a graph, GNNs can process local neighborhoods, capturing both topological patterns and node-level attributes. However, problems such as oversmoothing, where distinct node features become too similar deeper in the network remain active areas of research.

One promising direction is to incorporate ideas from Partial Differential Equations into GNNs. PDEs are widely used in physics and engineering to describe various dynamic systems, including heat diffusion, fluid flow, and wave propagation. By interpreting GNN message-passing steps as discrete approximations of PDEs, it becomes possible to guide the network with physical intuition.

In this paper, we explore how integrating physics-inspired loss terms and conservation principles into GNN training can improve the models. We focus on a simple diffusion setup on a line graph, examining how different loss-weighting strategies and model architectures perform on varying diffusion settings.

2 Background Theory

2.1 Graph

A graph is a type of data structure in which the data points are connected and have relations with each other. When considering graphs we refer to the data points as vertices, or nodes. The elements which connect the vertices are named edges, these values represent the relationships or interactions between the vertices. The edges can be of two types, directed or undirected, meaning that the relation between the vertices is of one-way or two-way, respectively. Based on stated information we can note a graph as $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Here the defined graph is a set of Vertices $\boldsymbol{V}$ and Edges $\boldsymbol{E}$, where $\boldsymbol{V}$ is a set of nodes, and $\boldsymbol{E}$ is a tuple set of all the edges connecting the nodes.

2.2 Graph Neural Networks

Graph Neural Networks are a type of neural networks designed to perform inference on data which can be structured as a graph described above. They effectively capture the complex relationships between nodes by making use of the structure and features of the graph. Unlike traditional neural networks that operate on fixed-size input vectors, GNNs are good at learning representations that encapsulate node features and graph topology.

The main idea behind GNNs is the iterative aggregation of information from a vertex neighbors to update its representation. This process is often referred to as *message passing* or *neighborhood aggregation*. Mathematically, the update of a node u at the k-th layer can be expressed as:

$$\mathbf{h}_u^{(k+1)} = \gamma^{(k)} \left(\mathbf{h}_u^{(k)}, a^k \left(\{ \mathbf{h}_v^{(k)} \mid \forall v \in \mathcal{N}(u) \} \right) \right) \tag{1}$$

Here the $\mathbf{h}_u^{(k)}$ is the feature vector of node u at layer k, $\mathcal{N}(u)$ the set of neighbours of node u, $a^k(\cdot)$ the aggregation function at layer k, and $\gamma^{(k)}(\cdot)$ the update function. To clarify, the aggregation function uses a method of choice which combines the features of the neighboring nodes, while the update function, encapsulates the aggregation of the current node with the neighbouring ones, application of weights and application of a non-linearity (e.g. Tanh or ReLU).

One of the challenges encountered in GNNs is called *oversmoothing*. Oversmoothing refers to the tendency of node representations to become similar as the

number of GNN layers increase. This homogenization of node features can decrese the performance of GNNs, particularly in tasks that rely on distinguishing between different nodes.

2.3 Partial Differential Equations

Partial Differential Equations are fundamental in describing various physical mechanics. They can consist of functions of several variables and their partial derivatives, allowing the shaping of systems where changes depend on multiple factors simultaneously.

2.3.1 The Laplacian

A key operator in the study of PDEs is the Laplacian, which measures how a function's value compares to its surroundings. In continuous domain, the Laplacian of scalar function f is defined as the divergence of its gradient. The Gradient (∇) turns a skalar function into a vector field, while the Divergence Operator ($\nabla\cdot$) measures the magnitude of flow of a vector field at a given point in space, showing how much the field diverge. An inportant note to make is as mentioned in [1], the Laplacian in the field of numerical PDEs is defined as $-L$, but the combinatorial Laplacian is defined as L. See below the continuous Laplacian.

$$\Delta f = \nabla \cdot \nabla f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} \tag{2}$$

Here, ∇ is the gradient operator, $\nabla\cdot$ represent the divergence, and n is the number of spatial dimensions. The Laplacian operator Δ measures the rate at which the average value of f around a point differs from the value at taht point.

When it comes to graphs the Laplacian is a matrix built from the Degree and Adjecency matrices, $\mathbf{D}$ and $\mathbf{A}$, respectively, encapsulating the structure of the Graph. Here the Degree Matrix $\mathbf{D}$ is a diagonal matrix in which each diagonal element D_{ii} represents the degree of node i, this is simply the number of edges connected to node i, the non-diagonal values are zero. Adjacency matrix $\mathbf{A}$ is a matrix where $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ if there is no connection between the nodes. Thus, making the symmetric Laplacian as follow:

$$L = \mathbf{D} - \mathbf{A} \tag{3}$$

This matrix L captures the difference between the connectivity of each node and its connections with neighboring nodes.

Nodes with many connections (high degrees) can dominate the properties of the Laplacian matrix. To reduce their excessive influence, we normalize the Laplacian by dividing its values by the degrees of these influential nodes. The symmetric normalized Laplacian is then expressed as:

$$\mathcal{L} = \mathbf{D}^{-\frac{1}{2}} L \mathbf{D}^{-\frac{1}{2}} = I - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} \tag{3.1}$$

2.3.2 Heat Equation

The Heat Equation is a partial differential equation which is used to model the distribution of heat in a given region over time. It describes how heat diffuses over a medium, where regions of high temperature transfer heat to the colder areas around. Compact notation of Heat Equation:

$$\frac{\partial u}{\partial t} = \alpha \Delta u \tag{4}$$

Here, $u = u(x, t)$ is the temperature at a point x in space and time t. The derivative of u with respect to time t respresent the rate of change of temperature with respect to time. α is a skalar value for diffusivity of the system, it simply controls the rate which heat diffuses at. Δ is the Laplacian operator.

2.3.3 Advection-Diffusion-Reaction

The Advection-Diffusion-Reaction[2] equation is a Partial Differential Equation that models the combined effects of advection, diffusion, and reaction on a scalar quantity. Adapting the ADR equation helps model dynamic processes such as information flow better, the following is a continuous PDE which acts as an ADR system:

$$\frac{\partial U}{\partial t} = \nabla \cdot (\mathbf{V} U) + K \Delta U + R(U) \tag{5}$$

Here the first term is called *Advection*, where V is the velocity matrix. It represents the total rate at which the u is transported. Note that the velocity matrix is a directed matrix which holds two directed edges each way between two nodes. The second term in the equation is the *Diffusion* term, which models the diffusion between the nodes. The third and last term is the *Reaction*. This term represent local transformations at each node, e.g. growth or decay. It is typically a function applied element-wise to the node feature vector u and include learnable parameters.

The graph representation of the ADR method considered in the paper [2] can be found in [3] and looks as follow:

$$\frac{\partial U}{\partial t} = div(\mathbf{V} \circ \mathbf{U}) + div(\mathbf{A}(\mathbf{U}) \cdot \nabla \mathbf{U}) + R(\mathbf{X}) \tag{6}$$

This is a more compact notation of the discretization. The full spactial discretisized graph equation is:

$$\begin{aligned} \frac{d\mathbf{U}(t)}{dt} &= \mathbf{DIV}(\mathbf{V}(\mathbf{U}(t), t; \mathbf{w}_a(t))\mathbf{U}(t)) \\ -\mathcal{L}\mathbf{U}(t)\mathbf{K}(t, \mathbf{w}_d(t)) &+ f(\mathbf{U}(t), \mathbf{X}, t; \mathbf{w}_r(t)) \end{aligned} \tag{7}$$

The important part for is to note the Velocity matrix $\mathbf{V}$ in the advection term, which is the first term of the equation. This matrix is built using incidence matrix which holds the structure and the directionality of the graph. For a graph with n nodes and m edges, the incidence matrix often notes as $\mathbf{B}$, and is an $n \times m$ matrix. Incidence matrix holds value 1 if node i is the head(destination) of edge j. If the node i is the tail(source) of edge j the value will be -1, this is illustrated in Figure 2. Other values in the matrix are 0 if there are no other connections. This matrix is then weighted using some learnable parameters, thus creating the $\mathbf{V}$ matrix.

2.4 Developed Methods

2.4.1 GRAND: Graph Neural Diffusion

GRAND [4] introduces a new way of learning node representation on graphs by using continuous diffusion processes and PDEs. This is done by interpreting

the *message-passing* in GNNs as a diffusion process, specifically the diffusion equation, aka. the Heat Equation. In this paper they also introduce a learnable attention matrix with the same structure as the adjacency matrix. It is represented as:

$$a(\mathbf{X}_i, \mathbf{X}_j) = softmax\left(\frac{(\mathbf{W}_K\mathbf{X}_i)^T\mathbf{W}_Q\mathbf{X}_j}{d_k}\right) \tag{8}$$

where $\mathbf{W}_K$ and $\mathbf{W}_Q$ hold learnable parameters, while d_k a dimension term. They rewrite the attention term as $\mathbf{A} = (a(\mathbf{X}_i, \mathbf{X}_j))$, and insert it into their graph diffusion equation resulting in:

$$\frac{\partial \mathbf{X}}{\partial t} = (\mathbf{A}(\mathbf{X}) - \mathbf{I})\mathbf{X} = \bar{\mathbf{A}}(\mathbf{X})\mathbf{X} \tag{9}$$

Here, the $\mathbf{X}$ is a matrix of node features, $\mathbf{A}(\mathbf{X})$ is the attention matrix which represents the weighted adjacency matrix, $\mathbf{I}$ is the idntity matrix, and the $\bar{\mathbf{A}}(\mathbf{X}) = \mathbf{A}(\mathbf{X}) - \mathbf{I}$ is attention matrix with self-attention.

2.4.2 PDE-GCN

The paper PDE-GCN[1] focuses on combining partial differential equations and graph convolutional networks to address the current limitations of GCNs. Since GCNs can be viewed as a generalization of Convolutional Neural Networks, and standard convolutions can be understood as differential operators on structured grids, the authors treat graph operations as discrete PDEs defined on graphs or manifolds. Through their approach they manage to build the proposed networks as deep as 64 layers without oversmoothing the features, effectivelly showing that this approach is viable. See below for the two main equations:

$PDE - GCN_D$, the diffusive equation:

$$\mathbf{f}^{(l+1)} = \mathbf{f}^{(l)} - h\mathbf{G}^T\mathbf{W}_l^T\sigma(\mathbf{W}_l\mathbf{G}\mathbf{f}^{(l)}) \tag{10}$$

This equation is viewed as a discrete function of a diffusion process on the graph. The term $\mathbf{G}\mathbf{f}^{(l)}$ computes the discrete gradient of node features,

identifying how features vary along edges. As mentioned in the section The Laplacian, we can express it as $-\mathbf{G}^T\mathbf{G}$, taking the divergence via $\mathbf{G}^T$, which feeds the information back into the node features. The parameter h acts as a time-step size in a PDE solver for stable evolution of the features.

$PDE - GCN_H$, the hyperbolic equation:

$$\mathbf{f}^{(l+1)} = 2\mathbf{f}^{(l)} - \mathbf{f}^{(l-1)} - h^2\mathbf{G}^T\mathbf{W}_l^T\sigma(\mathbf{W}_l\mathbf{G}\mathbf{f}^{(l)}) \tag{11}$$

This second-order formulation can be viewed to be similar to the discretized wave equation. The update depends on both the current and previous layer states. Because of this the features can propagate in a more dynamic manner. This prevents the trivial convergence to uniform features and reduce over-smoothing in deeper networks.

Here they also show that the weighted incidence matrix is discrete gradient operator on the graph, as follow:

$$(\mathbf{G}\mathbf{f})_{ij} = \mathbf{W}_{ij}(\mathbf{f}_i - \mathbf{f}_j) \tag{12}$$

For an edge connecting node i and j, $\mathbf{G}\mathbf{f}$ measures the difference in feature values along that edge, which is scaled by the edges weight $\mathbf{W}_{ij}$. This discrete differential operator makes the link between PDEs and GCNs.

2.5 PyTorch

PyTorch is a popular open-source library for building and training deep learning models. It provides an easy-to-use framework with support for dynamic computation graphs, making it flexible and ideal for research. PyTorch uses tensors, which are like multidimensional arrays, and supports GPU acceleration for faster computations. With tools for defining models, optimizing them, and handling data, PyTorch has become a key library in machine learning.

2.5.1 PyTorch Geometric

PyTorch Geometric [5] is an extension of PyTorch designed for working with graph data. It provides tools for building Graph Neural Networks.

PyG simplifies tasks like graph convolutions, pooling, and aggregation, and includes implementations of many GNN models. It is efficient, supports large graphs, and integrates with PyTorch, making it great for tasks like node classification, link prediction, and graph-level analysis.

3 Method

3.1 Synthetic Diffusion Data

In order to generate training data for our models, we simulate a diffusion process on a simple line graph of 22 nodes connected in a chain (Fig. 1). The graph G is constructed by placing the nodes in a line and adding edges between consecutive nodes, resulting in a single path-like structure. To introduce initial conditions, a number between 1 and 3 of randomly selected nodes are assigned feature values of 1.0, while all other nodes start with a negligible initial value (1^{-10}).

To simulate the diffusion dynamics, we employ a fourth-order Runge-Kutta (RK4) integration scheme. The underlying PDE-like update rule involves both a Laplacian operator (computed from the graph) and a divergence term derived from an incidence-based velocity matrix. Specifically, the incidence matrix B, together with a velocity vector V, determines how features flow along edges. In the case of an undirected graph, these edges naturally function bidirectionally, allowing features to propagate in both directions in accordance with the given velocities and the spatial distribution of node features. See the connection difference between undirected and directed graph in Figures 1 and 2.

When simulating diffusion with advection, a uniform velocity field ($V = 1$ for all edges) is chosen to maintain a controlled, consistent diffusion process. The set velocity on the graph will make the diffusion process propagate to the left. The RK4 method allows for stable and accurate numerical integration of the node feature vectors over time steps $h = 0.3$, where the process is stable for h values $0 < h <= 0.5$.

Over the course of multiple diffusion steps, we record the evolving node feature values at each time step. This time-series data is then used as input-output training pairs for our graph-based neural network models, enabling them to learn to predict future states from current ones.

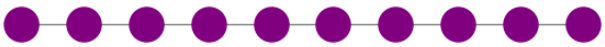


Figure 1: A simple undirected line graph of 10 nodes. Edges connect consecutive nodes.

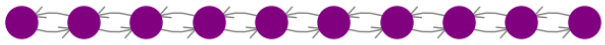


Figure 2: A simple bidirectional line graph of 10 nodes.

3.2 Diffusion Models

We test two classes of GNN models to predict the future states of the diffusion process: Graph Convolutional Network(GCNConv) and Graph Attention Network(GATConv). These models take as input the current node features and the graph structure and aim to forecast the feature distribution at the next time step. The GCN focus forcasting based on the features which are located on the nodes, while GAT has weighted edges, thus taking edge attributes into account. The models were initialized and trained with same parameters and sizes. The embedding sizes could be set to a chosen value. In our case all of the experiments were conducted by setting embedding size to 50, with following parameters:

Embedding Size	50
Non-Linearity	Tanh(.)
Loss Function	Custom
In Channels	1
Out Channels	1
Num. Layers	2
Learning Rate	$1 * 10^{-4}$

Table 1: Setup

Although the setup in table above is used in testing the models, some runs were done with slightly different values for the parameters, but these are used soly for visualization of stationary diffusion without conservation. In the result section these changes will be mentioned, and can be viewed on the Github Repository of the project.

3.2.1 Optimizer

For optimizing the GNN parameters, ADAM optimizer is used. ADAM combines the advantages of Adaptive Gradient (AdaGrad) and Root Mean Square Propagation (RMSProp), providing a robust and efficient training process. The learning rate for training was set to $1 * 10^{-4}$.

3.2.2 Physics-Informed Loss Function

The task is of regression type, thus we use Mean Squared Error loss function to measure the similarity between the network's predictions and the ground truth node feature values at the next time step. Minimizing MSE encourages the model to produce outputs that are numerically close to the true diffusion state.

We also employ a conservation of energy in the loss calculations, thus creating a custom loss function which is physics informed. The MSE loss function was extended by adding a term which calculates the difference between the sum of input features and sum of output features. The conservation term was then aggregated with MSE.

$$MSE = l(x, y) = (x_n - y_n)^2 \tag{13}$$

Here, the x is a prediction of a data point n, and y is the label for that same point, thus computing the difference between the prediction and the ground truth. The conservation loss is as follow:

$$CONS = \alpha * \left(\sum_{i}^{n} v_{input,i} - \sum_{i}^{n} v_{output,i} \right)^2 \tag{14}$$

Here, n is the number of nodes, the $v_{input,i}$ denotes the feature value of the node i in the input vector. Similarly, $v_{output,i}$ denotes a node i in the output vector. The constant value α scales the importance of the conservation loss term.

Further, combining the two terms mentioned, a complete Physics-Informed Loss Function is created:

$$Loss = MSE + CONS \tag{15}$$

This final loss function encourages the model to produce predictions that not only minimize numerical error but also stick to fundamental physical principles, resulting in more physically logical solutions.

4 Results and Discussion

The simulations which i will be refering to below can be viewed on Github Repository as mentioned earlier. The README files should be enough to guide you through it.

4.1 Diffusion without Conservation

Diffusion simulations without using the introduced Conservation term in the loss function has explosive or dying behaviour. Meaning, the sum of the features over the graph is not controlled, thus, this sum of features diverge to extreme values. Although explosive behavior is present, it can be observed in the beginning stages of diffusion that the models manage to somewhat mimic the shape of the diffusion before exploding or dying over time. This shows that the models are not stable at learning these processes. It can be also be seen that in some runs the models perform really well for many time steps before they degrade completely, even without the use of the physics-informed loss function. See these runs on no conservation simulations here.

4.2 Stationary Diffusion

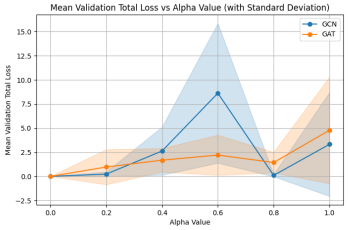
In this section we observe how the models behave when the physics-informed loss function is applied compared to the simulations without. In the two figures below we can view two different attempts at learning stationary diffusion by the two models. The runs were done with velocity set to 0, essentially simulating regular undirected inplace diffusion on the graph, see the simulations here. We observe the runs being similar for different α values in equation 14. The α values simulated are 0, 0.2, 0.4, 0.6, 0.8 and 1 for all runs. Comparing the case of $\alpha = 0$ and $\alpha > 0$, we note that $\alpha = 0$ leads to explosive behavior for both models. This confirms that the loss function does help in maintaining the total temperature of the graph.

In the figures below we compute the Mean Validation Loss and Variance for each α value using KFold Cross-Validation, where K=5. Meaning, 5 runs are done for each α value and the results are computed based on the conducted results. We also provide training losses for each α value, showing that the training process converges and illustrating how stable it is.

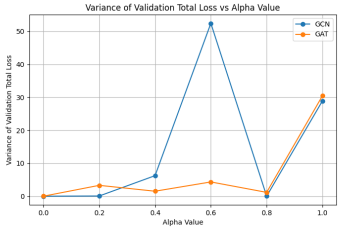
In both sets of runs, we see that variability increases as α increases. This is attributed to the conservation term becoming more influential within the loss function. Although this leads to higher variability, it successfully removes the explosive and dying behaviors observed without the conservation term.

4.2.1 Run 1

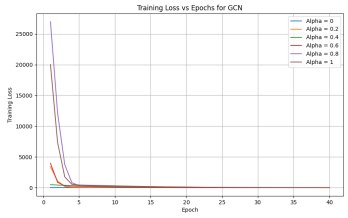
In this run, we observe a notable spike artifact around $\alpha = 0.6$. Such anomalies appear sporadically in both the GCN and GAT models, often occurring near $\alpha = 0.6$ or $\alpha = 0.8$. The exact cause of this phenomenon stays unclear. It could be related to the interplay between the conservation term and the model architectures, or perhaps some effect of the optimization dynamics. Further investigation, such as re-running the experiments with different random seeds or examining intermediate model states, would be helpful to explore the reason for this behavior. Scroll down in here and view the simulations.



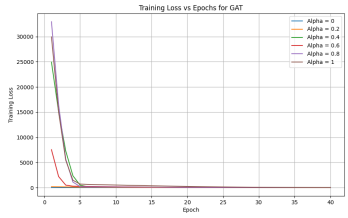
(a) Mean Validation Loss for each Alpha with Standard Deviation.



(b) Mean Variance over 5 Folds for each alpha value.



(c) GCN Training Loss for every α



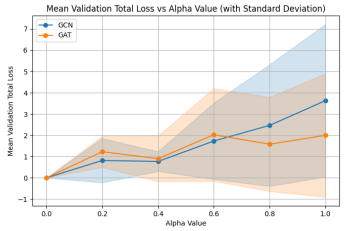
(d) GAT Training Loss for every α

Figure 3: Run 1 Results.

4.2.2 Run 2

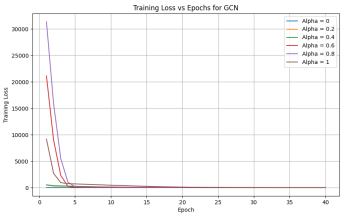
In the second run, the general trend observed previously is confirmed: as α grows, the variance also increases. This tells us that as the conservation term becomes more dominant in the loss function, the models show greater sensitivity to training conditions, leading to more noticable differences across folds. Despite this increased variability, using the conservation term still helps in mitigating explosive or degenerate behaviors, ultimately resulting

in more physically meaningful predictions. See the simulation here.

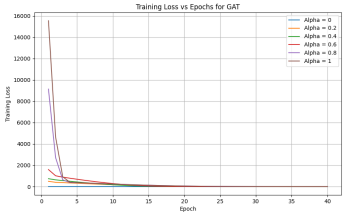


(a) Mean Validation Loss vs. each Alpha Value for conservation.

(b) Mean Variance over 5 Folds for each α value.



(c) GCN Training Loss for every α



(d) GAT Training Loss for every α

Figure 4: Run 2 Results.

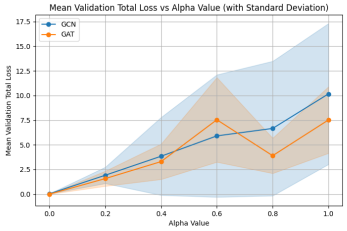
4.3 Diffusion with Advection

In this scenario, we introduce a uniform velocity field along the edges of the line graph, effectively adding an advection component to the diffusion process. Due to the attention mechanism, GAT is to be expected to hold better performance than GCN, as GATs inherently pay attention to and learn from edge-specific information. This could potentially allow them to capture directional flow patterns more effectively than GCNs, which treat neighborhood information more uniformly.

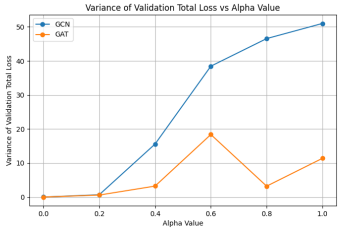
The velocity is set to 1 for all edges, imposing a consistent directional bias that pushes the 'heat' to the left along the line graph. The resulting flow is essentially a drift of features from right to left. For reference, both sets of advection simulations can be found here.

4.3.1 Run 1

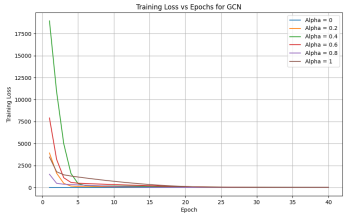
Interestingly, although the conservation term controls extreme behaviors, the GCN model still experiences 'explosive' tendencies after several time steps, failing to maintain stable node feature values. The GAT model, while not explosive, tends to exhibit 'dying' behaviors. Despite these behaviors, both models show stable training curves, indicating that the optimization process itself is not at fault. Instead, the difficulties arise from the capacity of each model to represent the advection-dominated dynamics. Even if the models managed to not die nor explode, the diffusion predictions would still be simply too poor. Note, the spike is again apparent on $\alpha = 0.6$ in Figure (a) below.



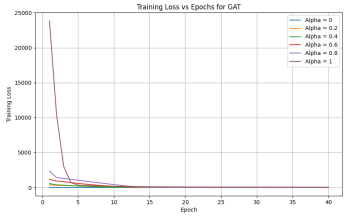
(a) Mean Validation Loss vs. each Alpha Value for conservation.



(b) Mean Variance over 5 Folds for each α value.



(c) GCN Training Loss for every α

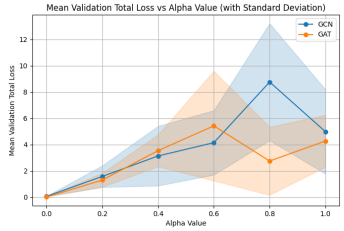


(d) GAT Training Loss for every α

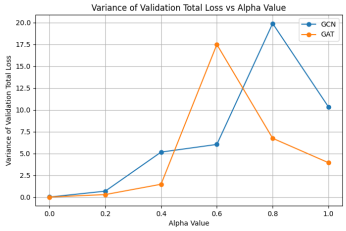
Figure 5: Run 1 Results.

4.3.2 Run 2

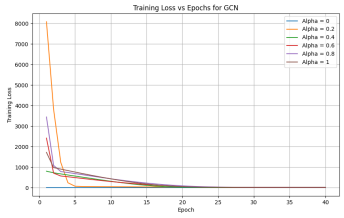
Theres not too much new to mention in this run other than the spikes appearing again for α values 0.6 and 0.8.



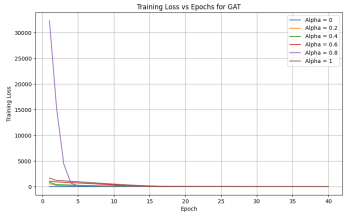
(a) Mean Validation Loss vs. each Alpha Value for conservation.



(b) Mean Variance over 5 Folds for each α value.



(c) GCN Training Loss for every α



(d) GAT Training Loss for every α

Figure 6: Run 2 Results.

4.4 Further Discussion

One explanation for the poor performance is that the implemented graph is a simple line graph without unique node identifiers, leaving the model with insufficient information about the global topology as mentioned in [6]. Although the suspicion is that the unique node identifiers could help, we would be diverging away from the true intended physical setting. As a result, it struggles to learn meaningful advection behavior and instead performs a form of averaging. This leads to a poorly learned diffusion-like process without advection rather than a true combination of diffusion AND advection.

An explanation for the observed variance spikes at α values 0.6 and 0.8 lies in the balance between the MSE term and the Conservation term in the loss function. The overall loss is defined as in equation 15

As α is varied, the relative importance of the conservation constraint changes. For very low α (near zero), the model primarily minimizes MSE and may heavily ignore conservation, potentially leading to explosive and not conserved

results. For very high α values, the model strongly enforces conservation, potentially at the cost of fitting the actual data, which can also introduce instability.

At $\alpha = 0.6$, the model sits in an intermediate regime where the influences of MSE and conservation are comparable. This case can create a 'tug-of-war' between fitting the data precisely using MSE and maintaining a global property (conservation). Such balancing might not be stable.

In other words, at certain mid-range α values, the optimization landscape may become more complex, featuring sharper minima or saddle points. The model could frequently jump between these regions during training, manifesting as spikes in variance. These spikes are not guaranteed to occur at exactly $\alpha = 0.6$ every time, but results suggest this value often places the model near an imbalance where neither MSE nor conservation fully dominates, making the training dynamics more sensitive and less predictable.

Further, due to hardware limitations, only 5-fold cross-validation was performed. Increasing the number of folds could create clearer statistical estimates of model performance, potentially clarifying the patterns behind variance. Expanding the range of α values tested may also help confirm whether the observed spikes at $\alpha = 0.6$ are general. Also, exploring other model architectures could improve overall performance and stability.

5 Conclusion

5.1 Summary of Findings

We investigated how incorporating physical principles into GNNs can improve the modeling of diffusion processes on a line graph. By introducing a physics-informed loss function that includes a conservation term, we aimed to prevent non-physical behaviors such as explosive or dying features on the graph. Our experiments show that, without conservation, both GCN and GAT models frequently diverge over time. Enabling the conservation term successfully removed these issues, maintaining a more stable total 'temperature' (sum of node features) throughout the stationary simulations.

5.2 Conservation Term and Parameter α

The parameter α scales the importance of the conservation term. Lower values of α allowed the model to fit data closely using MSE at the risk of loss in conservation. Higher values enforced conservation more but sometimes at the cost of fitting the data, resulting in increased variability. Interestingly, some values such as α=0.6 and α=0.8 produced spikes in variance, suggesting the existence of sensitive optimization areas where the interplay between MSE and conservation is particularly complex.

5.3 Struggle in Learning Advection

When adding to the diffusion a uniform velocity field (advection), both GCN and GAT models struggled to capture meaningful directional flows. Although GAT showed some advantage due to its attention mechanism, both models either tended to explosive or dying states. This outcome highlights the struggles of learning advection processes with the existing model architectures.

References

[1] Moshe Eliasof, Eldad Haber, and Eran Treister. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3836–3849. Curran Associates, Inc., 2021.

[2] Moshe Eliasof, Eldad Haber, and Eran Treister. Feature transportation improves graph neural networks, 2023.

[3] Andi Han, Dai Shi, Lequan Lin, and Junbin Gao. From continuous dynamics to graph neural networks: Neural diffusion and beyond, 2023.

[4] Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1407–1418. PMLR, 18–24 Jul 2021.

[5] Pytorch-geometric library documentation. `https://pytorch-geometric.readthedocs.io/en/latest/`.

[6] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

[7] Tobias S. Myrmel Antonsen. Propagating information like waves in gnns. `https://hdl.handle.net/10037/34272`, 2024-05-31.

[8] Benjamin Paul Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael M Bronstein. Beltrami flow and neural diffusion on graphs, 2021.

[9] Kai Zhao, Qiyu Kang, Yang Song, Rui She, Sijie Wang, and Wee Peng Tay. Graph neural convection-diffusion with heterophily, 2023.

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

[11] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020.