Flood Forecasting with Graph Neural Networks

Does River Network Topology Matter?

Nikolas Kirschstein

Mathematical Institute

University of Oxford

A thesis submitted for the degree of

Master of Science in Mathematics

Trinity 2023

This thesis is best printed in color.

Abstract

Due to climate change, riverine floods have become increasingly common. Forecasting them requires accurate discharge predictions. In this regard, deep learning methods recently started outperforming classical hydrological modeling techniques based on differential equations. The current state-the-art approaches treat forecasting at spatially distributed gauge stations as isolated problems. However, incorporating the known river network topology into the model has the potential to leverage the physical relationships between stations. Thus, we propose modeling river discharge for a network of gauging stations with a Graph Neural Network (GNN). To assess the benefit of relating stations to each other, we compare the forecasting performance achieved by different adjacency definitions: no adjacency at all, which is equivalent to existing approaches; binary adjacency of nearest up-/downstream stations; weighted adjacency according to physical relationships like stream length between stations; and learned adjacency via joint parameterization. Our results show that the model does not benefit from the river network topology information, regardless of the number of layers. The learned edge weights correlate with neither of the static definitions and exhibit no regular pattern. Furthermore, a worst-case analysis shows that the GNN struggles to predict sudden discharge spikes. In employing the Gradient Flow Framework (GRAFF), we find that parameter sharing across layers does not hurt model performance and that a mixture of attractive and repulsive forces act on vertex representations in the latent space of the GNN.

Contents

1 Introduction **1**

2 Background **4**
2.1 Spectral Graph Theory 4
2.2 Graph Neural Networks 9
2.2.1 Graph Convolutional Networks 9
2.2.2 The Gradient Flow Framework 12

3 Methodology **15**
3.1 Data Preprocessing 15
3.2 The Forecasting Task 17
3.2.1 Key Objects 17
3.2.2 Training 19
3.2.3 Inference 20

4 Results **22**
4.1 Experimental Setup 22
4.2 River Topology Impact 24
4.2.1 Performance Comparison 24
4.2.2 Learned Weights 25
4.2.3 Dirichlet Energy 26
4.3 Network Depth Impact 27
4.4 Worst Gauge Investigation 27
4.5 Using GRAFF 28

5 Conclusion **31**

Bibliography **32**

List of Figures

1.1 Share of occurrence of natural disasters by disaster type. Floods lead the ranking by a wide margin. (UNDRR & CRED, 2015). 1

1.2 A river network with gauges depicted in green (Kratzert et al., 2021). 2

3.1 Geographical overview of LamaH-CE. Circle color indicates gauge elevation and circle size indicates catchment size. (Klingler et al., 2021) 15

4.1 Boxplot of discharge autocorrelation for different lag values across gauges. Black circles denote outlier gauges which are consequently harder to forecast. 23

4.2 Mean and spread of learned weights during training. 25

4.3 Graph Dirichlet energy evolution for different adjacency types, averaged over the test set. In the isolated case, it is trivially zero, as no edges exist. 26

4.4 ResGCN performance with varying depth, averaged over gauges. Shaded areas indicate 95 % confidence intervals. 27

4.5 Rolling window forecasts for outlier gauge #85 with a 24-hour lead time. Negative time indicates past and positive time indicates future discharge. 29

4.6 Boxplots of the eigenvalues of the internal channel mixing matrix of all trained GRAFNNs. 30

List of Tables

4.1 Default hyperparameter choices for our experiments. 22

4.2 ResGCN performance on different river network topologies. Note how the isolated adjacency type leads to the exact same result across edge orientations as there are no edges, and we initialize with the same random seed. 24

4.3 Pearson correlation between learned weights and physical weights. . . 26

4.4 GRAFFNN performance on different river network topologies. Note how the isolated adjacency type leads to the exact same result across edge orientations as there are no edges, and we initialize with the same random seed. 28

Notation ∇

x	A scalar
$\boldsymbol{x}$	A vector
$\mathbf{X}$	A matrix
$\mathcal{X}$	A set
x_i	Element i of $\boldsymbol{x}$
$\\|\boldsymbol{x}\\|$	ℓ^2-norm of $\boldsymbol{x}$
$X_{i,j}$	Entry i, j of $\mathbf{X}$
$\mathbf{X}_i$	Row i of $\mathbf{X}$
$\mathbf{X}_{:,j}$	Column j of $\mathbf{X}$
$\mathbf{I}$	Identity matrix with dimensionality implied by context
$\mathbf{X}^\top$	Transpose of $\mathbf{X}$
S_n	Symmetric group on n elements
$\mathcal{O}_n(\mathbb{R})$	Orthogonal group in n dimensions

Chapter 1 Introduction

Floods are one of the most destructive natural disasters that occur on Earth, causing extensive damage to infrastructure, property, and human life. They account for almost half of all disaster events recorded by the United Nations (UNDRR & CRED, 2015, cp. Figure 1.1), making them the predominant type of disaster. In 2022 alone, 176 floods were recorded worldwide, which affected 57.1 million people, killed almost 8000, and caused 44.9 billion USD in damages (CRED, 2022). Due to ongoing climate change, floods have become increasingly frequent over the past decades and are expected to be even more prevalent in the future (UNDRR, 2022). Thus, early warning systems that can help authorities and individuals prepare for and respond to impending floods play a crucial role in mitigating fatalities and economic costs.

Operational flood forecasting systems like Google's Flood Forecasting Initiative (Nevo et al., 2022) typically focus on riverine floods as they are by far the most common and responsible for the vast majority of damages. A key component in these systems is the prediction of future river discharge[†] at a gauging station based on environmental indicators such as past discharge and precipitation.

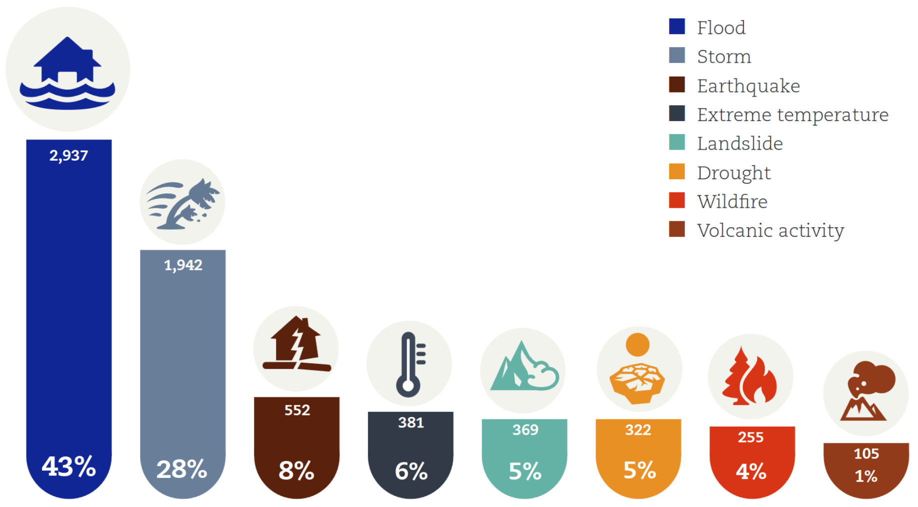


Figure 1.1: Share of occurrence of natural disasters by disaster type. Floods lead the ranking by a wide margin. (UNDRR & CRED, 2015).

[†]Discharge is the amount of water volume passing through a given river section per unit time.

Classical approaches toward river discharge prediction stem from finite-element solutions to partial differential equations such as the Saint-Venant shallow-water equations (Vreugdenhil, 1994; Wu, 2007). However, these models suffer from scalability issues since they become computationally prohibitive on larger scales, as required in the real world (Nevo et al., 2020). Furthermore, they impose a strong inductive bias by making numerous assumptions about the underlying physics.

On the other hand, data-driven methods and in particular deep learning provide excellent scaling properties and are less inductively biased. They are increasingly being explored for a plethora of hydrological applications, including discharge prediction (see surveys by Mosavi et al., 2018; Chang et al., 2019; Sit et al., 2020), where they tend to achieve higher accuracy than the classical models. The vast majority of studies employ Long Short-Term Memory models (LSTM; Hochreiter & Schmidhuber, 1997) due to their inherent suitability for sequential tasks and reliability in predicting extreme events (Frame et al., 2022). Whereas these studies usually consider forecasting for a single gauging station, Kratzert et al. (2019a,b) demonstrate the benefit of training a single spatially distributed LSTM model on multiple gauging sites jointly. While exploiting the shared underlying physics across gauges, their approach is still agnostic to the relationship between sites.

It appears reasonable that incorporating information from neighboring stations or even an entire river network like the one in Figure 1.2 into a spatially distributed model may be beneficial. Upstream gauges could "announce" the advent of significantly increased water masses to downstream gauges, which in turn could provide forewarning about flooding already ongoing further downstream. The input then becomes a graph whose vertices represent gauges and edges represent flow between gauges. The corresponding deep learning tool to capture these spatial dependencies is Graph Neural Networks (GNN). Kratzert et al. (2021) employ such a GNN as a post-processing step to route the per-gauge discharge predicted by a conventional LSTM along the river network, but it does not perform the actual prediction.

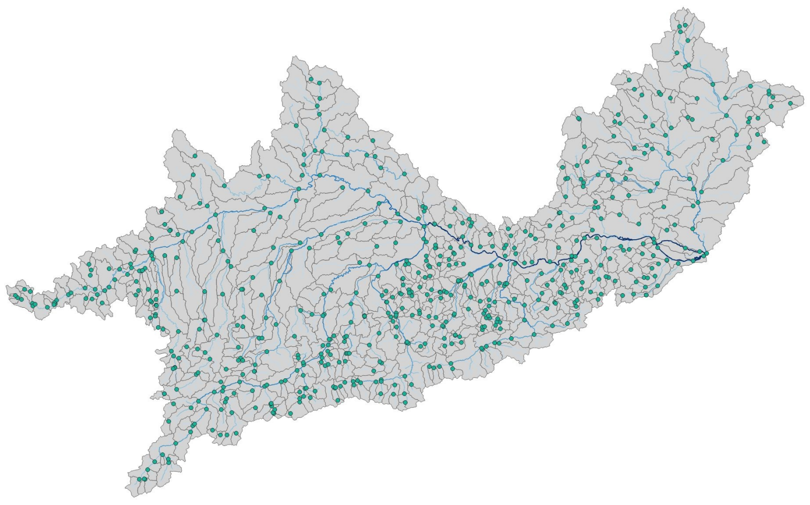


Figure 1.2: A river network with gauges depicted in green (Kratzert et al., 2021).

In this thesis, we propose to model discharge in a river network by a single end-to-end GNN to allow the network structure to be utilized during the prediction process itself. We train GNNs on the extensive Danube river network (Klingler et al., 2021) and, to assess the merit of incorporating the graph structure, compare the effect of different adjacency definitions:

(1) no adjacency, which is equivalent to existing approaches with shared parameters but isolated gauges,

(2) binary adjacency of neighboring gauges in the network,

(3) weighted adjacency according to physical relationships like stream length, elevation difference, and average slope between neighboring gauges, and

(4) learned adjacency by treating edge weights as a model parameter.

Furthermore, we explore the role of neural network depth on predictive capabilities. To gain insight into the behavior of trained models, we produce recurrent forecasts for the most challenging gauge and analyze the models using the recent Gradient Flow Framework (Di Giovanni et al., 2022). Our source code is publicly available[*].

The thesis is structured as follows: Chapter 2 provides necessary mathematical prerequisites on spectral graph theory and graph neural networks. Chapter 3 explains our approach in depth, covering data preprocessing and the machine learning pipeline. Chapter 4 reports and discusses our experimental results. Chapter 5 concludes the thesis and provides an outlook on future work.

[*]`https://anonymous.4open.science/r/FloodGNN`

Chapter 2: Background

In this chapter, we outline the mathematical prerequisites relevant to our study. We assume familiarity machine learning and deep learning fundamentals, for which we refer the reader to the textbooks by Bishop (2006) and Goodfellow et al. (2016), respectively. Section 2.1 highlights elements of spectral graph theory that we need, and Section 2.2 explains the mechanics of graph neural networks.

Throughout the entire chapter, we consider a *directed*, *weighted*, and *connected* finite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n := |\mathcal{V}|$ vertices and $m := |\mathcal{E}|$ edges. W.l.o.g., we write the vertex set as $\mathcal{V} = \{1, \dots, n\}$ and edge set as $\mathcal{E} = w^{-1}(\mathbb{R}_{>0})$ based on a non-negative weight assignment $w : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ where $w(i, j) = 0$ indicates the absence of an edge between i and j. Furthermore, we have a d-dimensional vertex signal $\mathbf{X} \in \mathbb{R}^{n \times d}$ given as a stack of row vectors $\boldsymbol{x}_i^\top \in \mathbb{R}^d$. We make the following technical assumptions:

1. The graph is *oriented*, i.e., it does not contain bidirectional edges:
$$(i, j) \in \mathcal{E} \implies (j, i) \notin \mathcal{E}.$$
2. The graph is *simple*, i.e., it does not contain self-loops:
$$(i, j) \in \mathcal{E} \implies i \neq j.$$

Note that conceptually, any undirected graph can be turned into an oriented one by assigning an arbitrary orientation to each edge. In this vein, most results carry over immediately to simple connected undirected graphs. We will remark on this in the appropriate places.

2.1 Spectral Graph Theory

The area of spectral graph theory studies properties of graphs by representing them as matrices. This is an essential prerequisite for performing calculations on a graph. Before we start defining these matrices, we introduce some convenient notation.

Definition 1. Let $i, j \in \mathcal{V}$ be vertices. We write $i \to j$ iff there is an edge from i to j, i.e., $(i, j) \in \mathcal{E}$.

Note that the relation $\to$ is never reflexive as $\mathcal{G}$ contains no self-loops, and never symmetric as $\mathcal{G}$ is oriented.

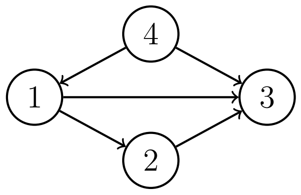
The most straightforward way to represent $\mathcal{G}$ is as a lookup table of w.

Definition 2. The *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$ represents the weight assignment w:

$$A_{i,j} := w(i,j).$$

Remark 3. The transpose $\mathbf{A}^\top$ represents the weight assignment with flipped edges and $\mathbf{A} + \mathbf{A}^\top$ the weight assignment with bidirectional edges. Furthermore, note that as $\mathcal{G}$ contains no self-loops, these matrices have an all-zero diagonal.

Example 4. For illustration purposes and to highlight a special case, the unweighted graph on the left serves as a running example throughout this section. Its adjacency matrix is shown on the right.



$$\rightsquigarrow \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

The adjacency matrix encodes all information contained in the edges. Hence, many properties of the graph can be directly obtained from it. When we sum up a row or column of $\mathbf{A}$, we get the total weight of incoming or outgoing edges, respectively.

Definition 5. Let $i \in \mathcal{V}$ be a vertex. We define

(a) the *in-degree* of i as the column sum $d_{i,\text{in}} := \sum_{j:j \to i} w(j,i)$, and

(b) the *out-degree* of i as the row sum $d_{i,\text{out}} := \sum_{j:i \to j} w(i,j)$.

As a shorthand, we can trivially pack the degree information into separate matrices.

Definition 6. The *in-/out-degree matrix* of $\mathcal{G}$ is the diagonal matrix of in-/out-degrees:

$$\mathbf{D}_{\text{in}} := \begin{bmatrix} d_{1,\text{in}} & & \\ & \ddots & \\ & & d_{n,\text{in}} \end{bmatrix}, \qquad \mathbf{D}_{\text{out}} := \begin{bmatrix} d_{1,\text{out}} & & \\ & \ddots & \\ & & d_{n,\text{out}}. \end{bmatrix}$$

Example 7. In the setting of Example 4, we have

$$\mathbf{D}_{\text{in}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{D}_{\text{out}} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

An alternative and more subtle way of encoding the weight assignment is to construct a non-square matrix that contains one row per edge, indicating the endpoints of the associated edge. For row indexing, we implicitly use an arbitrary enumeration $\mathcal{E} \to \{1, \ldots, m\}$ of the edges.

Definition 8. The *incidence matrix* $\boldsymbol{\nabla} \in \mathbb{R}^{m \times n}$ of $\mathcal{G}$ is given by

$$\nabla_{(i,j),k} := \begin{cases} -\sqrt{w(i,j)} & \text{if } k = i, \\ \sqrt{w(i,j)} & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases}$$

Example 9. For the graph in Example 4, the incidence matrix is

$$\boldsymbol{\nabla} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Remark 10. The concept of an incidence matrix does not trivially carry over to undirected graphs since it inherently requires a sense of direction on edges. Hence, *an* incidence matrix of an undirected graph always depends on the chosen edge orientation and is thus not unique.

It is crucial to realize that the rows of $\boldsymbol{\nabla}$ are indexed by the *edges* of $\mathcal{G}$, not the vertices. This means that, by virtue of matrix multiplication from the left, the incidence matrix transforms a vertex signal into an edge signal.

Lemma 11. *Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a vertex signal. The product $\boldsymbol{\nabla}\mathbf{X} \in \mathbb{R}^{m \times d}$ is an edge signal where each row is the weighted difference of the features of its endpoints:*

$$(\boldsymbol{\nabla}\mathbf{X})_{(i,j)} = \sqrt{w(i,j)}(\boldsymbol{x}_j^\top - \boldsymbol{x}_i^\top).$$

Proof. Consider the row-matrix product $(\boldsymbol{\nabla}\mathbf{X})_{(i,j)} = \sum_{k \in \mathcal{V}} \nabla_{(i,j),k} \cdot \boldsymbol{x}_k^\top$. □

The differences of neighboring features appearing in Lemma 11 can be interpreted as a discrete derivative of the vertex signal $\mathbf{X}$ (Zhou & Schölkopf, 2005).

Definition 12. We call $\boldsymbol{\nabla}\mathbf{X}$ the *graph gradient* of $\mathbf{X}$.

The graph gradient behaves analogously to the continuous gradient, which turns a scalar field $f : \mathbb{R}^n \to \mathbb{R}$ into a vector field $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$. Another key operator in vector analysis is divergence, the adjoint operator of the gradient, which turns a vector field $g : \mathbb{R}^n \to \mathbb{R}^n$ back into a scalar field $\operatorname{div} g : \mathbb{R}^n \to \mathbb{R}$ by summing the rate of change in each coordinate. We can define an analogous construct for graphs. Formally, the graph divergence should be the adjoint operator of the graph gradient. Since the incidence matrix represents a linear operator, this is just its transpose.

Lemma 13. *Let $\mathbf{Y} \in \mathbb{R}^{m \times d}$ be an edge signal. The product $\boldsymbol{\nabla}^\top\mathbf{Y} \in \mathbb{R}^{n \times d}$ is a vertex signal where each row measures the overall "flow" through the vertex:*

$$(\boldsymbol{\nabla}^\top\mathbf{Y})_k = \sum_{i : i \to k} \underbrace{\sqrt{w(i,k)}\mathbf{Y}_{(i,k)}}_{\text{inflow from } i} - \sum_{j : k \to j} \underbrace{\sqrt{w(k,j)}\mathbf{Y}_{(k,j)}}_{\text{outflow to } j}.$$

Proof. Consider the row-matrix product $(\boldsymbol{\nabla}^\top \mathbf{Y})_k = \sum_{(i,j)\in\mathcal{E}} \nabla_{(i,j),k} \mathbf{Y}_{(i,j)}$. □

Definition 14. We call $\boldsymbol{\nabla}^\top \mathbf{Y}$ the *graph divergence* of $\mathbf{Y}$.

In vector analysis, the combination of gradient and divergence yields the well-known Laplace operator $\Delta = \operatorname{div} \nabla$. We immediately get the analogous construct on graphs using the same definition.

Definition 15. The *graph Laplacian matrix* is given by

$$\boldsymbol{\Delta} \coloneqq \boldsymbol{\nabla}^\top \boldsymbol{\nabla}.$$

There is a more practicable expression that only involves matrix addition.

Lemma 16. *The Laplacian can be explicitly written as*

$$\boldsymbol{\Delta} = (\mathbf{D}_{\text{in}} + \mathbf{D}_{\text{out}}) - (\mathbf{A} + \mathbf{A}^\top),$$

which means, since $\mathcal{G}$ contains no self-loops,

$$\Delta_{i,j} = \begin{cases} d_{i,\text{in}} + d_{i,\text{out}} & \text{if } i = j \\ -w(i,j) & \text{if } i \to j \\ -w(j,i) & \text{if } j \to i \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Proof. Each entry of the product is an inner product between columns of $\boldsymbol{\nabla}$:

$$\Delta_{i,j} = (\boldsymbol{\nabla}^\top \boldsymbol{\nabla})_{i,j} = \langle \boldsymbol{\nabla}_{:,i}, \boldsymbol{\nabla}_{:,j} \rangle = \begin{cases} d_{i,\text{in}} + d_{i,\text{out}} & \text{if } i = j, \\ -w(i,j) & \text{if } i \to j \text{ and not } j \to i, \\ -w(j,i) & \text{if } j \to i \text{ and not } i \to j, \\ -w(i,j) - w(j,i) & \text{if both } i \to j \text{ and } j \to i, \\ 0 & \text{otherwise.} \end{cases}$$

Since the fourth case never occurs as $\mathcal{G}$ is oriented, this shows Equation (2.1). □

Example 17. The Laplacian matrix of the graph in Example 4 is

$$\boldsymbol{\Delta} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & 3 \\ -1 & 0 & -1 & 2 \end{bmatrix}.$$

Remark 18. Note that while the incidence matrix depends on the orientation chosen for an undirected graph, the Laplacian does not due to its symmetric definition with respect to edge orientation. Therefore, the Laplacian of an undirected graph is unique.

The graph Laplacian constitutes the main object of interest in spectral graph theory, owing to the following observation.

Lemma 19. $\mathbf{\Delta}$ *is symmetric positive semi-definite.*

Proof. Symmetry follows from Definition 15 as

$$\mathbf{\Delta}^\top = (\mathbf{\nabla}^\top \mathbf{\nabla})^\top = \mathbf{\nabla}^\top (\mathbf{\nabla}^\top)^\top = \mathbf{\Delta}.$$

To see positive semi-definiteness, consider for any $\boldsymbol{v} \in \mathbb{R}^n$ the quadratic form

$$\boldsymbol{v}^\top \mathbf{\Delta} \boldsymbol{v} = (\mathbf{\nabla} \boldsymbol{v})^\top \mathbf{\nabla} \boldsymbol{v} = \|\mathbf{\nabla} \boldsymbol{v}\|^2 \geq 0. \qquad \square$$

Importantly for us, Lemma 19 implies via the spectral theorem that $\mathbf{\Delta}$ is orthogonally diagonalizable and has only real non-negative eigenvalues, enabling the full toolbox of spectral analysis. First of all, $\mathbf{\Delta}$ always has a zero eigenvalue since its rows sum up to zero and thus any constant vector is in the null space. Furthermore, the multiplicity of that eigenvalue gives the number of connected components in the graph (Chung, 1997). Generally speaking, the spectrum of $\mathbf{\Delta}$ yields a “fingerprint” of the graph structure.

Of particular interest is the quadratic form defined by $\mathbf{\Delta}$, which for a normalized vector $\boldsymbol{v} \in \mathbb{R}^n$ equals the Rayleigh quotient known from numerical linear algebra. Hence, if $\boldsymbol{v}$ is in addition an eigenvector of $\mathbf{\Delta}$ with eigenvalue λ, then

$$\lambda = \boldsymbol{v}^\top \lambda \boldsymbol{v} = \boldsymbol{v}^\top \mathbf{\Delta} \boldsymbol{v} = \|\mathbf{\nabla} \boldsymbol{v}\|^2 = \sum_{(i,j)\in\mathcal{E}} (\mathbf{\nabla} \boldsymbol{v})^2_{(i,j)} = \sum_{(i,j)\in\mathcal{E}} w(i,j)(v_i - v_j)^2.$$

Therefore, λ indicates the smoothness of its eigenvectors along edges and can be seen as a *frequency* inherent to the graph structure. Larger eigenvalues correspond to higher frequencies, as the corresponding eigenvectors “wiggle” more along edges. For arbitrary vectors, we can decompose them as a linear combination of eigenvectors and say that they contain different frequency components. For a multi-dimensional vertex signal $\mathbf{X}$, the diagonal of the product $\mathbf{X}^\top \mathbf{\Delta} \mathbf{X}$ contains the channel-wise quadratic forms. We thus obtain a general smoothness measure via its trace, with a normalization factor added for differentiation convenience.

Definition 20. The *graph Dirichlet energy* of $\mathbf{X}$ is given by

$$\mathsf{E}_{\text{DIR}}(\mathbf{X}) \coloneqq \tfrac{1}{2}\operatorname{tr}(\mathbf{X}^\top \mathbf{\Delta} \mathbf{X}) = \tfrac{1}{2} \sum_{(i,j)\in\mathcal{E}} \|(\mathbf{\nabla} \mathbf{X})_{(i,j)}\|^2 = \tfrac{1}{2} \sum_{(i,j)\in\mathcal{E}} w(i,j)\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2.$$

Clearly, E_{DIR} is always non-negative. The relationship with the continuous Dirichlet energy arises by viewing the summation as integration on the discrete Hilbert space of edge signals.

2.2 Graph Neural Networks

In this work, we employ graph neural networks (GNNs), which generalize conventional neural nets for data on a grid to data with a generic graph topology. In particular, GNNs exploit the relational inductive bias given by the graph structure. This development of porting neural networks from the Euclidian plane to other geometries is part of a broader effort called Geometric Deep Learning (Bronstein et al., 2017).

Recall that the vertex labeling $\mathcal{V} = \{1, \ldots, n\}$ and thus the order of the rows in the feature matrix and the rows/columns in the adjacency matrix is completely arbitrary. Any permutation of the labels yields an isomorphic graph, which is a property that needs to be incorporated into the GNN architecture. Each layer in a GNN is a function $\mathrm{GNNLayer}_\theta : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times d'}$ parameterized by θ that takes an input signal and the graph structure represented by the adjacency matrix to produce an output signal. It should be operating in a way so that permutations of the vertices do not alter the output except for permuted order.

Definition 21. We call $\mathrm{GNNLayer}_\theta$ *permutation-equivariant* if for all permutations $\pi \in S_n$ it holds that

$$\mathrm{GNNLayer}_\theta(\mathbf{P}_\pi \mathbf{X},\ \mathbf{P}_\pi \mathbf{A} {\mathbf{P}_\pi}^\top) = \mathbf{P}_\pi\, \mathrm{GNNLayer}_\theta(\mathbf{X}, \mathbf{A}),$$

where the permutation matrix $\mathbf{P}_\pi$ is defined using the Kronecker δ as

$$\mathbf{P}_\pi := \begin{bmatrix} & \vdots & \\ \cdots & \delta_{\pi(i),j} & \cdots \\ & \vdots & \end{bmatrix} \in \mathcal{O}_n(\mathbb{R}).$$

2.2.1 Graph Convolutional Networks

The choice

$$\mathrm{GNNLayer}_\theta(\mathbf{X}, \mathbf{A}) = \sigma(\mathbf{X}\mathbf{W})$$

for an element-wise activation function $\sigma : \mathbb{R} \to \mathbb{R}$ and a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times d'}$ corresponds to a standard multi-layer perceptron (MLP) without bias term and does not take the graph topology into account. The weight matrix acts on each row independently and can thus be seen as performing *channel mixing*. The most straightforward way to incorporate topology is by multiplying a variant of the adjacency matrix from the left, so that it operates across rows. Commonly, its transpose is used[†]:

$$\mathrm{GNNLayer}_\theta(\mathbf{X}, \mathbf{A}) = \sigma(\mathbf{A}^\top \mathbf{X}\mathbf{W})$$

Denoting by $\boldsymbol{x}_i$ the i-th row of $\mathbf{X}$ as a column vector, we can read this row-wise as a vertex feature update

$$\boldsymbol{x}_i \leftarrow \sigma\Big(\sum_{j:j\to i} w(j,i)\mathbf{W}^\top \boldsymbol{x}_j\Big) \tag{2.2}$$

[†]The literature is often sloppy with this detail and omits the transposition as it usually considers undirected graphs where the adjacency matrix is symmetric anyway.

where the channel-mixed features, called *messages*, of neighbors on incoming edges are "pulled in" and combined in a weighted sum. This is very reminiscent of the aggregation behavior of the well-known convolutional neural networks on grids. The general framework of aggregating messages from a certain neighborhood is known in the literature as *message passing* (Wu et al., 2022). While the graph structure is now accounted for, there remain two technical issues with Equation (2.2):

- *self-loops*: As $\mathcal{G}$ has no self-loops, the aggregation over all vertices with incoming edges to i does not include the feature of vertex i itself. It is, however, desirable for the updated feature of vertex i to also depend on its own previous value. We can achieve this by enforcing self-loops, i.e., using an augmented adjacency matrix $\hat{\mathbf{A}} := \mathbf{A} + c\mathbf{I}$ and degree matrix $\hat{\mathbf{D}}_{\text{in}} := \mathbf{D}_{\text{in}} + c\mathbf{I}$ for $c > 0$.
- *normalization*: The number of summands depends on the in-degree and therefore differs across vertices. When stacking multiple layers, this can lead to vastly differing scales and thus cause numerical issues. We normalize by the in-degrees of the vertices participating in an edge to fix the scale issue.

Definition 22. We call $\bar{\mathbf{A}} := \hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}$ the *normalized augmented adjacency matrix.*

With this modified adjacency definition, we obtain a directed version of the vanilla GCN layer definition introduced by Kipf & Welling (2017) for undirected graphs.

Definition 23. A *GCN layer* with parameters $\theta = \{\mathbf{W}\}$ and element-wise activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is given by

$$\text{GCNLayer}_\theta(\mathbf{X}, \mathbf{A}) := \sigma(\bar{\mathbf{A}}^\top \mathbf{X}\mathbf{W}) = \sigma(\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\hat{\mathbf{A}}^\top\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}).$$

The weight $\mathbf{W} \in \mathbb{R}^{d\times d'}$ is called the *channel-mixing* matrix. The fully expanded row-wise vertex feature update equation reads

$$\boldsymbol{x}_i \leftarrow \sigma\Big(\tfrac{c}{d_{i,\text{in}}+c}\mathbf{W}^\top\boldsymbol{x}_i + \sum_{j:j\to i}\tfrac{w(j,i)}{\sqrt{(d_{i,\text{in}}+c)(d_{j,\text{in}}+c)}}\mathbf{W}^\top\boldsymbol{x}_j\Big). \tag{2.3}$$

Recall that we assumed $\mathcal{G}$ to be connected, guaranteeing positive degrees and thus ensuring the normalization is well-defined. Finally, we must verify that this layer respects the graph symmetries.

Lemma 24. GCNLayer *is permutation-equivariant.*

Proof. The key is $\mathbf{P}_\pi$ being orthogonal. Let $\mathbf{A}_\pi := \mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi{}^\top$ and $\mathbf{D}_\pi := \mathbf{P}_\pi\mathbf{D}_{\text{in}}\mathbf{P}_\pi{}^\top$ denote the permuted adjacency and in-degree matrix, respectively. We have

$$\begin{aligned}
\hat{\mathbf{A}}_\pi &= \mathbf{A}_\pi + \mathbf{I} = \mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi{}^\top + \mathbf{P}_\pi\mathbf{P}_\pi{}^\top = \mathbf{P}_\pi(\mathbf{A}+\mathbf{I})\mathbf{P}_\pi{}^\top = \mathbf{P}_\pi\hat{\mathbf{A}}\mathbf{P}_\pi{}^\top \\
\hat{\mathbf{D}}_\pi &= \mathbf{D}_\pi + \mathbf{I} = \mathbf{P}_\pi\mathbf{D}_{\text{in}}\mathbf{P}_\pi{}^\top + \mathbf{P}_\pi\mathbf{P}_\pi{}^\top = \mathbf{P}_\pi(\mathbf{D}_{\text{in}}+\mathbf{I})\mathbf{P}_\pi{}^\top = \mathbf{P}_\pi\hat{\mathbf{D}}_{\text{in}}\mathbf{P}_\pi{}^\top
\end{aligned}$$

due to orthogonality, and $\hat{\mathbf{D}}_\pi^{-\frac{1}{2}} = \mathbf{P}_\pi\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\mathbf{P}_\pi{}^\top$ since

$$\hat{\mathbf{D}}_\pi(\mathbf{P}_\pi\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\mathbf{P}_\pi{}^\top)^2 = \mathbf{P}_\pi\hat{\mathbf{D}}_{\text{in}}\underbrace{\mathbf{P}_\pi{}^\top\mathbf{P}_\pi}_{\mathbf{I}}\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\underbrace{\mathbf{P}_\pi{}^\top\mathbf{P}_\pi}_{\mathbf{I}}\hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}}\mathbf{P}_\pi{}^\top = \mathbf{P}_\pi\hat{\mathbf{D}}_{\text{in}}\hat{\mathbf{D}}_{\text{in}}^{-1}\mathbf{P}_\pi{}^\top = \mathbf{I},$$

which leads to

$$\bar{\mathbf{A}}_\pi = \hat{\mathbf{D}}_\pi^{-\frac{1}{2}} \hat{\mathbf{A}}_\pi \hat{\mathbf{D}}_\pi^{-\frac{1}{2}} = \mathbf{P}_\pi \hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}} \cancel{\mathbf{P}_\pi^\top \mathbf{P}_\pi} \hat{\mathbf{A}} \cancel{\mathbf{P}_\pi^\top \mathbf{P}_\pi} \hat{\mathbf{D}}_{\text{in}}^{-\frac{1}{2}} \mathbf{P}_\pi^\top = \mathbf{P}_\pi \bar{\mathbf{A}} \mathbf{P}_\pi^\top.$$

Putting everything together and using that the non-linearity is point-wise, we get

$$\begin{aligned}
\operatorname{GCNLayer}_\theta(\mathbf{P}_\pi \mathbf{X}, \mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^\top) &= \sigma(\bar{\mathbf{A}}_\pi^\top \mathbf{P}_\pi \mathbf{X} \mathbf{W}) \\
&= \sigma(\mathbf{P}_\pi \bar{\mathbf{A}}^\top \cancel{\mathbf{P}_\pi^\top \mathbf{P}_\pi} \mathbf{X} \mathbf{W}) \\
&= \mathbf{P}_\pi \sigma(\bar{\mathbf{A}}^\top \mathbf{X} \mathbf{W}) \\
&= \mathbf{P}_\pi \operatorname{GCNLayer}_\theta(\mathbf{X}, \mathbf{A}).
\end{aligned}$$ □

Remark 25. When assuming $\mathcal{G}$ to be undirected, there is an alternative and more principled way to arrive at the GCN layer equation, which Defferrard et al. (2016) and Kipf & Welling (2017) used to derive it originally. It draws from the frequency interpretation of the eigenvalues of $\mathbf{\Delta}$ to define graph analogs of the Fourier transform and the convolution operation known from continuous signal processing. A particular choice of filter to be convolved with the vertex signal then yields Definition 23. However, this derivation explicitly relies on the adjacency matrix to be symmetric. It does not work in the more general directed case, which is the one we consider in this work. Therefore, we had to derive it ad-hoc without the graph signal processing motivation.

A subtle issue remains: by setting $\widetilde{w}(i,i) := c$ and $\widetilde{w}(i,j) := w(i,j)$ for $i \neq j$, as well as $\widetilde{d}_{i,\text{in}} := d_{i,\text{in}} + c$, we can rewrite the vectorized layer equation (2.3) to

$$\boldsymbol{x}_i \leftarrow \sigma(\mathbf{W}^\top \sum_{j: j \to i \text{ or } j = i} \frac{\widetilde{w}(j,i)}{\sqrt{\widetilde{d}_{i,\text{in}} \widetilde{d}_{j,\text{in}}}} \boldsymbol{x}_j).$$

Hence, the layer's output for a vertex is essentially a weighted neighborhood feature average which gets transformed by $\mathbf{W}$ and σ. This means that applying many of these updates in succession leads to the features of adjacent vertices converging, a phenomenon known as *oversmoothing* (Oono & Suzuki, 2020). The graph Dirichlet energy allows to make this statement precise.

Lemma 26. *For a stack of GCN layers given by* $\mathbf{H}^{(k)} := \operatorname{GCNLayer}_{\theta_k}(\mathbf{H}^{(k-1)}, \mathbf{A})$, *we have* $\mathsf{E}_{\text{DIR}}(\mathbf{H}^{(k)}) \xrightarrow{k \to \infty} 0$.

Proof. See Cai & Wang (2020). □

A common remedy is introducing a *residual connection* into each layer by adding the input to the output (Chen et al., 2020). In this vein, the neural network successively performs additive changes to the input instead of transforming it completely at every layer. Note that this now requires the input and output dimensionality to match up.

Definition 27. A *residual GCN layer* with parameters $\theta = \{\mathbf{W}\}$ where $\mathbf{W} \in \mathbb{R}^{d \times d}$ is square and element-wise activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is given by

$$\operatorname{ResGCNLayer}_\theta(\mathbf{X}, \mathbf{A}) := \mathbf{X} + \operatorname{GCNLayer}_\theta(\mathbf{X}, \mathbf{A}).$$

2.2.2 The Gradient Flow Framework

For understanding the behavior of GNNs, Di Giovanni et al. (2022) propose a principled framework based on the concept of gradient flows. It relies on the key fact that residual GNNs[†] can be viewed as simulating the evolution of an ordinary differential equation (Chen et al., 2018; Xhonneux et al., 2020). To see this, let $\mathrm{GNNLayer}_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ be a GNN layer with parameters θ and consider an autonomous first-order ODE of the form

$$\dot{\mathbf{X}}(t) = \mathrm{GNNLayer}_\theta(\mathbf{X}(t)) \tag{2.4}$$

evolving the initial state $\mathbf{X}(0) = \mathbf{X}$ for $t \geq 0$. To discretize this, we choose a *step size* $\tau > 0$ and construct the Taylor expansion around $\mathbf{X}(t)$:

$$\mathbf{X}(t+\tau) = \mathbf{X}(t) + \tau\,\mathrm{GNNLayer}_\theta(\mathbf{X}(t)) + \mathcal{O}(\tau^2).$$

By dropping all higher-order terms, we obtain the forward Euler method for approximately solving Equation (2.4). It starts at $\mathbf{X}^{(0)} := \mathbf{X}$ and iteratively updates

$$\mathbf{X}^{(k+1)} := \mathbf{X}^{(k)} + \tau\,\mathrm{GNNLayer}_\theta(\mathbf{X}^{(k)}), \tag{2.5}$$

for $k = 0, \dots, N-1$. This is reminiscent of an N-layer *residual* neural network with the parameters θ *shared* across layers. If the step size τ is sufficiently small, we have $\mathbf{X}^{(k)} \approx \mathbf{X}(k\tau)$. The ODE dynamics can be recovered for $\tau \to 0$ as the continuous-time limit of the GNN.

Definition 28. The ODE (2.4) is called a *gradient flow* if there exists an energy functional $\mathsf{E}_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}$ such that the right-hand side is its negative gradient:

$$\mathrm{GNNLayer}_\theta(\mathbf{X}(t)) = -\nabla\mathsf{E}_\theta(\mathbf{X}(t)). \tag{2.6}$$

Lemma 29. *The energy E_θ defining a gradient flow decreases monotonically along a solution $\mathbf{X}$ of the gradient flow.*

Proof. $\frac{\mathrm{d}}{\mathrm{d}t}\mathsf{E}_\theta(\mathbf{X}(t)) = \langle -\nabla\mathsf{E}_\theta(\mathbf{X}(t)), \dot{\mathbf{X}}(t)\rangle = -\|\nabla\mathsf{E}_\theta(\mathbf{X}(t))\|^2 \leq 0.$ □

The corresponding forward Euler step simulation algorithm obtained by plugging (2.6) into (2.5) is gradient descent:

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} - \tau\nabla\mathsf{E}_\theta(\mathbf{X}^{(k)}). \tag{2.7}$$

Due to the energy minimization perspective, gradient flow ODEs are ubiquitous in physics and provide high interpretability. Specifically, we can consider the vertex features as particles in $\mathbb{R}^d$ with energy E_θ. The only remaining design choice is the definition of E_θ. In an attempt to generalize the continuous Dirichlet energy stemming from the heat diffusion equation, Di Giovanni et al. propose the following quadratic energy parameterized by two symmetric matrices:

[†]In fact, any residual neural network

Definition 30. The *GRAFF energy* of $\mathbf{X}$ parameterized by $\theta = \{\mathbf{W}, \boldsymbol{\Omega}\} \subseteq \mathbb{R}^{d\times d}$ where $\mathbf{W}, \boldsymbol{\Omega}$ are symmetric is given by

$$\mathsf{E}_\theta(\mathbf{X}) \coloneqq \underbrace{\tfrac{1}{2}\sum_{i\in\mathcal{V}} \|\boldsymbol{x}_i\|^2_{\boldsymbol{\Omega}}}_{\text{external energy}} \underbrace{-\tfrac{1}{2}\sum_{(i,j)\in\mathcal{E}} \bar{A}_{ij}\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle_{\mathbf{W}}}_{\text{internal energy}} . \tag{2.8}$$

The first term is independent of $\bar{\mathbf{A}}$ and describes an external energy component, while the second term accounts for pair-wise interactions. The symmetry requirement is a technical necessity arising when we differentiate to obtain the gradient flow equation.

Lemma 31. $\dot{\mathbf{X}}(t) = -\nabla \mathsf{E}_\theta(\mathbf{X}(t)) = \bar{\mathbf{A}}\mathbf{X}(t)\mathbf{W} - \mathbf{X}(t)\boldsymbol{\Omega}.$ (2.9)

Proof. Differentiation, see Di Giovanni et al. (2022, Appendix B). □

The resulting neural network definition by plugging Equation (2.9) into (2.5) reads

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \tau(\bar{\mathbf{A}}\mathbf{X}^{(k)}\mathbf{W} - \mathbf{X}^{(k)}\boldsymbol{\Omega}). \tag{2.10}$$

For $\tau = 1$ and $\boldsymbol{\Omega} = \mathbf{0}$, we obtain a residual GCN with *shared* symmetric weights that lacks non-linear activations. However, Di Giovanni et al. prove that E_θ also decreases along solutions of an "upgraded" version of Equation (2.9) which includes a non-linearity.

Lemma 32. *The GRAFF energy E_θ also monotonically decreases along solutions $\mathbf{X}$ of $\dot{\mathbf{X}}(t) = \sigma(-\nabla \mathsf{E}_\theta(\mathbf{X}(t)))$ where $\sigma : \mathbb{R} \to \mathbb{R}$ satisfies $x\sigma(x) \geq 0$.*

Proof. See Di Giovanni et al. (2022, Appendix D). □

Combining Equation (2.10) and Lemma 32, we arrive at a new type of GNN layer.

Definition 33. A *GRAFF layer* with parameters $\theta = \{\mathbf{W}, \boldsymbol{\Omega}\}$ and activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is given by

$$\mathrm{GRAFFLayer}_\theta(\mathbf{X}, \mathbf{A}) \coloneqq \mathbf{X} + \tau\sigma(\bar{\mathbf{A}}\mathbf{X}\mathbf{W} - \mathbf{X}\boldsymbol{\Omega}). \tag{2.11}$$

The symmetric square weights $\mathbf{W}, \boldsymbol{\Omega} \in \mathbb{R}^{d\times d}$ are called the *internal* and *external channel-mixing* matrix, respectively.

Importantly, weight symmetry does not diminish the expressive power of the GNN (Hu et al., 2019). Let us briefly verify that this new layer is well-behaved.

Lemma 34. GRAFFLayer *is permutation-equivariant.*

Proof. Re-using $\bar{\mathbf{A}}_\pi$ from the proof of Lemma 24, we have:

$$\begin{aligned}
\mathrm{GRAFFLayer}_\theta(\mathbf{P}_\pi\mathbf{X}, \mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi{}^\top) &= \mathbf{P}_\pi\mathbf{X} + \tau\sigma(\bar{\mathbf{A}}_\pi\mathbf{P}_\pi\mathbf{X}\mathbf{W} - \mathbf{P}_\pi\mathbf{X}\boldsymbol{\Omega}) \\
&= \mathbf{P}_\pi\mathbf{X} + \tau\sigma(\mathbf{P}_\pi\bar{\mathbf{A}}\cancel{\mathbf{P}_\pi{}^\top}\mathbf{P}_\pi\mathbf{X}\mathbf{W} - \mathbf{P}_\pi\mathbf{X}\boldsymbol{\Omega}) \\
&= \mathbf{P}_\pi\mathbf{X} + \tau\sigma(\mathbf{P}_\pi(\bar{\mathbf{A}}\mathbf{P}_\pi\mathbf{X}\mathbf{W} - \mathbf{X}\boldsymbol{\Omega})) \\
&= \mathbf{P}_\pi\,\mathrm{GRAFFLayer}_\theta(\mathbf{X}, \mathbf{A}).
\end{aligned}$$

□

Stacking multiple GRAFF layers yields a GNN that reduces the GRAFF energy functional with increasing depth, either directly by simulating its gradient flow or indirectly thanks to Lemma 32. We can use this connection to gain more insight into the dynamics of such a GNN.

In particular, we consider the eigenvalue decomposition of the (symmetric) internal channel mixing matrix $\mathbf{W} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$. Split $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}_+ - \boldsymbol{\Lambda}_-$ into the positive part $\boldsymbol{\Lambda}_+ \coloneqq \max\{\mathbf{0}, \boldsymbol{\Lambda}\}$ and the negative part $\boldsymbol{\Lambda}_+ \coloneqq \max\{\mathbf{0}, -\boldsymbol{\Lambda}\}$ of the spectrum to obtain

$$\mathbf{W} = \underbrace{\mathbf{U}\boldsymbol{\Lambda}_+\mathbf{U}^\top}_{=:\mathbf{W}_+} - \underbrace{\mathbf{U}\boldsymbol{\Lambda}_-\mathbf{U}^\top}_{=:\mathbf{W}_-} = \boldsymbol{\Theta}_+^\top\boldsymbol{\Theta}_+ - \boldsymbol{\Theta}_-^\top\boldsymbol{\Theta}_-.$$

By design, $\mathbf{W}_+$ and $\mathbf{W}_-$ are symmetric positive definite. Hence, their Cholesky decompositions $\mathbf{W}_+ = \boldsymbol{\Theta}_+^\top\boldsymbol{\Theta}_+$ and $\mathbf{W}_- = \boldsymbol{\Theta}_-^\top\boldsymbol{\Theta}_-$ are well-defined.

Lemma 35. *The GRAFF energy decomposes as follows:*

$$\mathsf{E}_\theta(\mathbf{X}) = \frac{1}{2}\sum_{i\in\mathcal{V}}\|\boldsymbol{x}_i\|^2_{\boldsymbol{\Omega}-\mathbf{W}} + \underbrace{\frac{1}{4}\sum_{(i,j)\in\mathcal{E}}\|\boldsymbol{\Theta}_+(\nabla\mathbf{X})_{(i,j)}\|^2}_{\text{attraction}} - \underbrace{\frac{1}{4}\sum_{(i,j)\in\mathcal{E}}\|\boldsymbol{\Theta}_-(\nabla\mathbf{X})_{(i,j)}\|^2}_{\text{repulsion}} \tag{2.12}$$

Proof. See Di Giovanni et al. (2022). □

Recall that by Lemma 32, the energy E_θ decreases with deeper layers of the GNN. Hence, in (2.12) the terms $\|\boldsymbol{\Theta}_+(\nabla\mathbf{X})_{(i,j)}\|^2$ decrease over time, so that adjacent vertex representations become aligned in the non-null singular subspaces of $\boldsymbol{\Theta}_+$. This behavior can be interpreted as *attractive* forces that lead to a smoothing effect. Analogously, in the orthogonal complement, adjacent representations are pushed apart by *repulsive* forces leading to a sharpening effect due to the growth of $\|\boldsymbol{\Theta}_-(\nabla\mathbf{X})_{(i,j)}\|^2$.

Chapter 3 Methodology

In this chapter, we explain our approach in depth. Section 3.1 outlines our data preprocessing steps, and Section 3.2 fully specifies the central machine learning task we consider.

3.1 Data Preprocessing

Our study is based on the LamaH-CE[†] dataset (Klingler et al., 2021). It contains historical discharge measurements on an hourly resolution for 859 gauging locations in the broader Danube river network shown in Figure 3.1. Covering an area of 170 000 km^2 with diverse environmental conditions, Klingler et al. expect that results from large-scale investigations on this dataset carry over to other river networks.

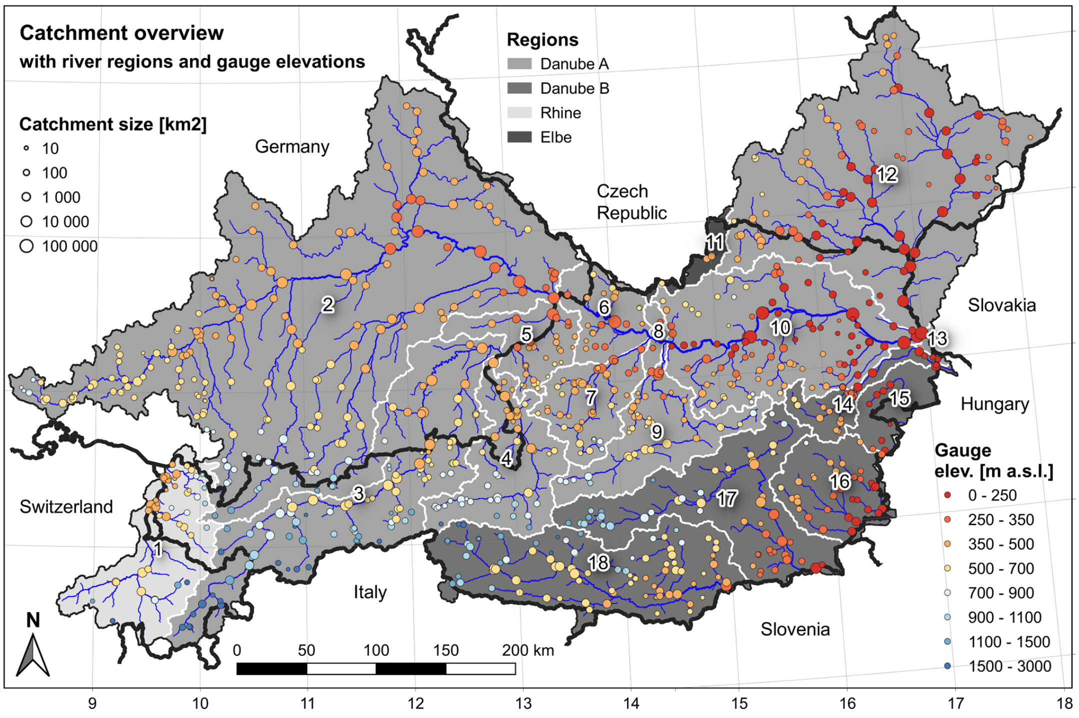


Figure 3.1: Geographical overview of LamaH-CE. Circle color indicates gauge elevation and circle size indicates catchment size. (Klingler et al., 2021)

[†]**LA**rge-Sa**M**ple D**A**ta for **H**ydrology for **C**entral **E**urope

Next to the discharge measurements, LamaH-CE also offers meteorological time series data with indicators like temperature, precipitation, and wind force. Due to computational limitations, we cannot take into account the meteorological data in this study. However, this is not an issue since we are only interested in a conceptual relative comparison, rather than absolute performance. Nevertheless, future work should aim to incorporate the meteorological data as well, for it is likely to improve the forecasting capabilities of a statistical predictor.

The river network defined by LamaH-CE naturally forms a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The vertices $\mathcal{V}$ represent gauges, and the edges $\mathcal{E}$ represent flow between a gauge and the next downstream gauges. Hence, $\mathcal{G}$ is *anti-transitive*, i.e., no skip connections exist:

$$\forall u, v, w \in \mathcal{V} : (u, v) \in \mathcal{E} \land (v, w) \in \mathcal{E} \implies (u, w) \notin \mathcal{E}.$$

Region Selection. Figure 3.1 shows that $\mathcal{G}$ contains four different connected components, of which we choose “Danube A” for our study as it is by far the largest one. Its most downstream gauge close to the Austrian-Hungarian border has complete discharge data for the years 2000 through 2017. Starting at this gauge, we determine all connected gauges of the Danube A region by performing an inverse depth-first search given by Algorithm 1. Overall, 608 out of the original 859 gauges belong to this connected component.

Algorithm 1: Inverse depth-first search

```
Input: DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, start vertex $v_0 \in \mathcal{V}$
Output: All direct and indirect predecessors of $v_0$ in $\mathcal{G}$

inverseDFS($\mathcal{G}, v_0$)
1   $\mathcal{V}_{\text{in}} \leftarrow \{v \in \mathcal{V} \mid (v, v_0) \in \mathcal{E}\}$
2   if $\mathcal{V}_{\text{in}} = \emptyset$ then
3       return $\{v_0\}$
4   else
5       return $\{v_0\} \cup \bigcup_{v \in \mathcal{V}_{\text{in}}}$ invDFS($v$)
```

Gauge Filtering. A critical issue with the historical measurements is gaps in the data. Klingler et al. have filled any consecutive gaps of at most six hours by linear interpolation and marked the remaining longer gaps with the value -999. We only want to consider gauges that (a) do not have these longer periods of missing values and (b) provide discharge for at least the same timeframe (2000 to 2017) as the most downstream gauge. To this end, we remove all gauges that violate these requirements from the graph using Algorithm 2. Predecessors and successors of a deleted vertex get newly connected, so that network connectivity is maintained. Note that thanks to antitransitivity, a duplicate check is unnecessary when inserting the new edges. After this pre-processing step, we are left with 375 out of the previously 608 gauges, which is quite a drastic reduction.

Algorithm 2: Rewire-removal of a vertex

Input: antitransitive DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, moribund vertex $v_{\text{RIP}} \in \mathcal{V}$
Output: $\mathcal{G}$ without v_{RIP} where its predecessors and successors are rewired

```
rewireRemove($\mathcal{G}, v_{\text{RIP}}$)
1    $\mathcal{V}_{\text{in}} \leftarrow \{v \in \mathcal{V} \mid (v, v_{\text{RIP}}) \in \mathcal{E}\}$
2    $\mathcal{V}_{\text{out}} \leftarrow \{v \in \mathcal{V} \mid (v_{\text{RIP}}, v) \in \mathcal{E}\}$
3    $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v_{\text{RIP}}\}$
4    $\mathcal{E} \leftarrow \mathcal{E} \setminus (\mathcal{V}_{\text{in}} \times \{v_{\text{RIP}}\}) \setminus (\{v_{\text{RIP}}\} \times \mathcal{V}_{\text{out}}) \cup (\mathcal{V}_{\text{in}} \times \mathcal{V}_{\text{out}})$
```

Overall, the reduced graph $\mathcal{G}$ now consists of $n := |\mathcal{V}| = 375$ gauges with T hours of discharge measurements for the years 2000 to 2017, which we can conceptually represent as a vertex signal $\mathbf{Q} = \left[\boldsymbol{q}^{(1)} \mid \boldsymbol{q}^{(2)} \mid \dots \mid \boldsymbol{q}^{(T)}\right] \in \mathbb{R}^{n \times T}$.

Normalization. As is common practice in deep learning, we normalize the data to surrender all gauges to the same scale and accelerate the training process (LeCun et al., 2002). In particular, we normalize the discharge time series per gauge using the standard score, i.e., we calculate the per-gauge empirical means and variances

$$\boldsymbol{\mu} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{q}^{(t)}, \qquad \boldsymbol{\sigma}^2 = \frac{1}{T-1}\sum_{i=1}^{T} (\boldsymbol{q}^{(t)} - \boldsymbol{\mu})^2,$$

and replace each time step as $\boldsymbol{q}^{(t)} \leftarrow \frac{\boldsymbol{q}^{(t)} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$. All operations are applied element-wise.

Train-test split. To assess the performance of a trained model on unseen data, we reserve the last two years, i.e., one ninth ($\sim$11 %) of all observations, as a test set. Training thus only uses data from the years 2000 to 2015.

3.2 The Forecasting Task

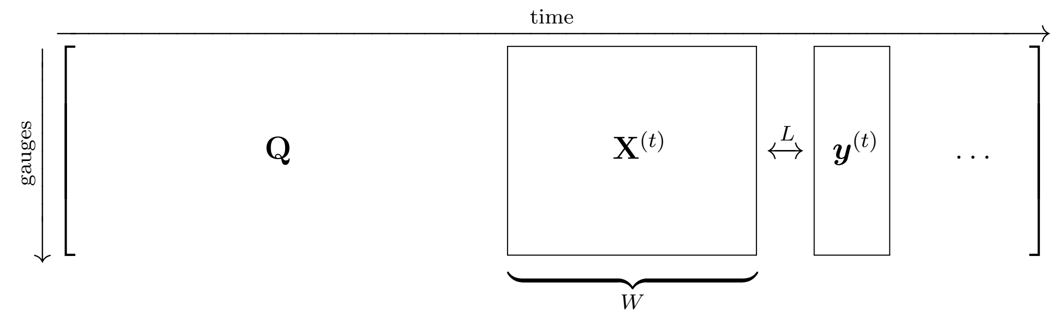
With the data being sorted, we now define the machine learning task for the GNN, which is an instance of supervised vertex autoregression. Assume we are given a certain amount of $W \in \mathbb{N}$ most recent hours of discharge measurements for all gauges. Our goal is to predict the discharge $L \in \mathbb{N}$ hours in the future. We call W the *window size* and L the *lead time* of the predictor.

3.2.1 Key Objects

Features/Targets. To conduct supervised learning, we extract input-output pairs from the time series represented by $\mathbf{Q}$. For $t = W, \dots, T - L$, we define the feature matrix at time step t and the corresponding target vector as

$$\mathbf{X}^{(t)} := \left[\boldsymbol{q}^{(t-W+1)} \,\middle|\, \dots \,\middle|\, \boldsymbol{q}^{(t-1)} \,\middle|\, \boldsymbol{q}^{(t)} \right] \in \mathbb{R}^{n\times W}, \qquad \boldsymbol{y}^{(t)} := \boldsymbol{q}^{(t+L)} \in \mathbb{R}^{n}.$$

We collect all samples into the set $\mathcal{D} = \{(\mathbf{X}^{(t)}, \boldsymbol{y}^{(t)})\}_{t=W}^{T-L}$ and partition it according to the train-test split into $\mathcal{D} = \mathcal{D}_{\text{train}} \uplus \mathcal{D}_{\text{test}}$. The extraction process can be illustrated as follows:



Adjacency. Besides the discharge history, we feed the GNN the river network topology. For the definition of adjacency matrix entries corresponding to edges $(i, j) \in \mathcal{E}$ (the rest being zero), we consider the following options:

- *isolated:* $\mathbf{A}_{i,j} := 0$ equates to removing all edges and results in the augmented normalized adjacency matrix to be a multiple of the identity so that the GCNLayer degenerates to a vertex-wise linear layer.
- *binary:* $\mathbf{A}_{i,j} := 1$ corresponds to the unaltered adjacency matrix as it comes with the LamaH-CE dataset.
- *weighted:* $\mathbf{A}_{i,j} := w_{(i,j)}$ quantifies a physical relationship, for which LamaH-CE provides three alternatives:
  - the *stream length* along the river between i and j,
  - the *elevation difference* along the river between i and j, and
  - the *average slope* of the river between i and j.
- *learned:* $\mathbf{A}_{i,j} := \omega_{(i,j)}$ where $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{E}|}$ is a learnable model parameter.

The first two options allow us to compare the effect of introducing the river network topology into the model at all. The last two options enable insights into what kind of relative importance of edges is beneficial for the predictor. To fully specify the augmented normalized adjacency matrix $\bar{\mathbf{A}}$, we generally set the weights for the artificial self-loops as the mean of all incoming edge weights to assign equal importance to self-loops. The only exception to this is option one, where the mean would be zero and thus result in no information flow whatsoever, so that we set the self-loop weights to one instead.

Model. Our desideratum is a GNN $f_\theta : \mathbb{R}^{n\times W} \to \mathbb{R}^n$ parameterized by θ which closely approximates the mapping of windows $\mathbf{X}$ to targets $\boldsymbol{y}$, i.e., $\hat{\boldsymbol{y}} \coloneqq f_\theta(\mathbf{X}) \approx \boldsymbol{y}$. All our models have a sandwich architecture: a linear layer $\text{Encoder}_{\theta_\text{E}} : \mathbb{R}^{n\times W} \to \mathbb{R}^{n\times d}$ embeds the W-dimensional input per gauge into a d-dimensional latent space. On this space, a sequence of N GNN layers $\text{GNNLayer}_{\theta_i} : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times d}$ are applied. Finally, another linear layer $\text{Decoder}_{\theta_\text{D}} : \mathbb{R}^{n\times d} \to \mathbb{R}^n$ projects from the latent space to scalars per gauge. In symbols, this reads:

$$\begin{aligned}
\mathbf{H}^{(0)} &\coloneqq \text{Encoder}_{\theta_\text{E}}(\mathbf{X}) \\
\mathbf{H}^{(i)} &\coloneqq \text{GNNLayer}_{\theta_i}(\mathbf{H}^{(i-1)}, \mathbf{A}) \qquad \text{for } i = 1, \dots, N \\
\hat{\boldsymbol{y}} &\coloneqq \text{Decoder}_{\theta_\text{D}}(\mathbf{H}^{(N)}).
\end{aligned}$$

The rationale behind the sandwich construction is to allow all hidden GNN layers to operate on the same latent space $\mathbb{R}^d$, which is a requirement for residual layers like ResGCNLayer (Definition 27) and GRAFFLayer (Definition 33). We may illustrate it as follows:

$$\begin{tikzcd}
\mathbf{X} \in \mathbb{R}^{n\times W} \arrow[r, mapsto, "\text{Encoder}"] & \mathbf{H}^{(i)} \in \mathbb{R}^{n\times d} \arrow[loop below, "N \times \text{GNNLayer}"] \arrow[r, mapsto, "\text{Decoder}"] & \hat{\boldsymbol{y}} \in \mathbb{R}^n
\end{tikzcd}$$

3.2.2 Training

Optimization Objective. To measure the error between a model prediction $\hat{\boldsymbol{y}}$ and the ground truth $\boldsymbol{y}$, we use the multi-dimensional square loss

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) \coloneqq \tfrac{1}{n}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2.$$

Training is then defined as optimizing the expected loss over the empirical distribution of training samples in $\mathcal{D}_\text{train}$. The optimal model parameters are given by

$$\arg\min_\theta \mathbb{E}_{(\mathbf{X},\boldsymbol{y})\sim\mathcal{D}_\text{train}}[L(f_\theta(\mathbf{X}, \bar{\mathbf{A}}), \boldsymbol{y})].$$

Graph Batching. For computational efficiency, we do not feed the model a single sample at a time during training but a *batch* of B different samples $(\mathbf{X}^{(t_i)}, \boldsymbol{y}^{(t_i)}) \in \mathcal{D}_\text{train}$. To this end, we stack the samples on top of each other and duplicate the normalized adjacency matrix along the diagonal of a B times larger matrix:

$$\mathbf{X}_\text{batch} \coloneqq \begin{bmatrix} \mathbf{X}^{(t_1)} \\ \mathbf{X}^{(t_2)} \\ \vdots \\ \mathbf{X}^{(t_B)} \end{bmatrix}, \qquad \boldsymbol{y}_\text{batch} \coloneqq \begin{bmatrix} \boldsymbol{y}^{(t_1)} \\ \boldsymbol{y}^{(t_2)} \\ \vdots \\ \boldsymbol{y}^{(t_B)} \end{bmatrix}, \qquad \bar{\mathbf{A}}_\text{batch} \coloneqq \begin{bmatrix} \bar{\mathbf{A}} & & & \\ & \bar{\mathbf{A}} & & \\ & & \ddots & \\ & & & \bar{\mathbf{A}} \end{bmatrix}.$$

Note that graph convolutional layers such as GCNLayer and GRAFFLayer only operate on neighbors and thus vertices within the same connected component. Since $\bar{\mathbf{A}}_{\text{batch}}$ defines B different connected components isolated from each other, the result will be the same as if the computation had been performed individually. However, batching allows us to exploit parallelism on modern scientific computing hardware like TPUs and IPUs.

3.2.3 Inference

Metrics. Recall that we perform training on the normalized samples. This means that the square loss objective during training treats all gauges equally as they have the same scale, which is a desirable property for training. However, when evaluating the performance of the model, we need to calculate metrics on the unnormalized version of the predictions and targets:

$$\begin{aligned}\hat{\boldsymbol{y}}_{\text{orig}} &\coloneqq \boldsymbol{\sigma} \odot \hat{\boldsymbol{y}} + \boldsymbol{\mu},\\ \boldsymbol{y}_{\text{orig}} &\coloneqq \boldsymbol{\sigma} \odot \boldsymbol{y} + \boldsymbol{\mu}.\end{aligned}$$

The most intuitive regression metric is the *Mean Squared Error* (MSE). In our multi-dimensional regression problem, it is defined as the error vector

$$\mathbf{MSE} \coloneqq \tfrac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i=1}^{|\mathcal{D}_{\text{test}}|} (\hat{\boldsymbol{y}}_{\text{orig}}^{(t_i)} - \boldsymbol{y}_{\text{orig}}^{(t_i)})^2 = \boldsymbol{\sigma}^2 \odot \tfrac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i=1}^{|\mathcal{D}_{\text{test}}|} (\hat{\boldsymbol{y}}^{(t_i)} - \boldsymbol{y}^{(t_i)})^2$$

where all operations are applied element-wise. We clearly see how the factor $\boldsymbol{\sigma}^2$ re-introduces the scale which normalization removed.

Next to the MSE, the most widely used metric in hydrology is the *Nash-Sutcliffe Efficiency* (NSE; Nash & Sutcliffe, 1970). It compares the sum of squared errors of the model to the sum of squared errors of the constant mean-predictor and subtracts this value from one to obtain a percentage score in $[0, 1]$.

$$\mathbf{NSE} \coloneqq 1 - \frac{\sum_{i=1}^{|\mathcal{D}_{\text{test}}|} (\hat{\boldsymbol{y}}_{\text{orig}}^{(t_i)} - \boldsymbol{y}_{\text{orig}}^{(t_i)})^2}{\sum_{i=1}^{|\mathcal{D}_{\text{test}}|} (\boldsymbol{\mu} - \boldsymbol{y}_{\text{orig}}^{(t_i)})^2} = 1 - \frac{\mathbf{NSE}}{\boldsymbol{\sigma}^2}$$

An NSE of zero means that the model's predictive capability is no better than that of the empirical mean, while an NSE of one means that all model predictions are perfect. We straightforwardly obtain summary metrics by averaging across gauges:

$$\overline{\text{MSE}} \coloneqq \frac{1}{n} \sum_{g=1}^{n} \text{MSE}_g, \qquad \overline{\text{NSE}} \coloneqq \frac{1}{n} \sum_{g=1}^{n} \text{NSE}_g\,.$$

We leave the consideration of further evaluation metrics like the ones proposed by Yilmaz et al. (2008) to future work.

Recurrent Forecasting. Recall that the model output is a single discharge prediction for lead time L. However, in practice, it is desirable to produce a contiguous series of hourly future discharge predictions. One solution is to simply train multiple models for lead times $L = 1, 2, \ldots$ which is computationally demanding. Alternatively, the predictions of a single model trained with $L = 1$ can be fed back in a recurrent fashion to simulate higher lead times. Algorithm 3 specifies this shift-and-repeat approach.

Algorithm 3: Rolling Window Forecasting

```
Input: Trained model $f_\theta : \mathbb{R}^{n\times W} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times W}$
       Discharge observations $\mathbf{X} \in \mathbb{R}^{n\times W}$
       Adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$
       Number of recurrent forecasting steps $K \in \mathbb{N}$
Output: Recurrent forecast $L$ steps into the future

rollingForecast($f_\theta, \mathbf{X}, L$)
  $\hat{\mathbf{Y}} \leftarrow [\ ]$
  for $K$ iterations do
      $\hat{\boldsymbol{y}} \leftarrow f_\theta(\mathbf{X}, \mathbf{A})$
      $\mathbf{X} \leftarrow \begin{bmatrix} | & | & & | & | \\ \mathbf{X}_{:,2} & \mathbf{X}_{:,3} & \cdots & \mathbf{X}_{:,W} & \hat{\boldsymbol{y}} \\ | & | & & | & | \end{bmatrix}$
      $\hat{\mathbf{Y}} \leftarrow [\hat{\mathbf{Y}} \quad \hat{\boldsymbol{y}}]$
  return $\hat{\mathbf{Y}}$
```

Chapter

Results

4

In this chapter, we present and discuss our empirical results. Section 4.1 details the experimental setup, Section 4.2 explores the impact of river topology, Section 4.3 explores the impact of model depth, Section 4.4 investigates the worst-case performance, and Section 4.5 adds the Gradient Flow Framework to the picture.

4.1 Experimental Setup

We conduct all experiments on Graphcore IPUs using Paperspace. Our Python implementation is based on the libraries PyTorch (Paszke et al., 2019), PyTorch Geometric (Fey & Lenssen, 2019), and PopTorch (Graphcore, 2020). The code to reproduce our experiments is publicly available as an anonymized Git repository[†]. Table 4.1 lists the relevant hyperparameters we use throughout all experiments unless stated otherwise, categorized into data, model, and training parameters.

	Hyperparameter	Value
data	window size (W)	24 h
	lead time (L)	1 h
	normalization?	yes
model	architecture	ResGCN
	network depth (N)	20
	latent space dim (d)	64
	shared parameters?	no
	edge direction	downstream
	adjacency type	binary
training	initialization	glorot
	optimizer	adam
	# epochs	20
	batch size (B)	16
	learning rate (α)	10^{-4}

Table 4.1: Default hyperparameter choices for our experiments.

On the data side, we choose a window size of $W = 24$ as a compromise between sufficiently many past observations and computational efficiency. We further justify this choice at the end of this section. We set the lead time to $L = 1$ since we can achieve higher lead times by recurrent forecasting (Algorithm 3).

On the model side, we employ a residual GCN with ReLU non-linearities. We choose a depth of $N = 20$ layers to allow it to propagate information along the entire river graph as we determined the longest path in the graph to consist of 19 edges. The latent space dimensionality of $d = 64$ is rather arbitrary and mostly a memory concern. We usually give each layer in the neural network its own set of parameters, but consider the case of shared parameters in Section 4.5. The information flow direction and adjacency type hyperparameters will be explored in detail in Section 4.2.

[†]`https://anonymous.4open.science/r/FloodGNN`

On the optimization side, all neural network parameters are randomly initialized using the theoretically well-founded Glorot initialization scheme (Glorot & Bengio, 2010). We then perform 20 epochs (passes through the dataset) of stochastic mini-batch gradient descent, which is enough for the process to converge. The descent algorithm used is the sophisticated Adaptive Moments (Adam) optimizer (Kingma & Ba, 2015) with a base learning rate of 10^{-4}, which results in stable training. To prevent overfitting, we randomly hold out 25% of the training set, which corresponds to three years of observations, and select the parameters from the epoch in which the loss calculated over this holdout set was the lowest.

To provide further justification for the choice of window size W, consider the autocorrelation of discharge, i.e., the Pearson correlation of the entire discharge time series with a time-lagged version of itself. Intuitively, only time lags for which this autocorrelation is high are informative for predicting a time step. We calculate this autocorrelation across time steps for each gauge individually and visualize the resulting distributions over gauges in Figure 4.1 with boxplots. We see that discharge, on average, correlates very well ($\rho \geq 0.8$) with the past 15 hours. However, one particular outlier, corresponding to the lowest sequence of black circles, has plummeting autocorrelation for time lags of more than two hours. We will pay special attention to this gauge (ID 85 in the dataset) in Section 4.4 as it can be expected to be particularly hard to forecast.

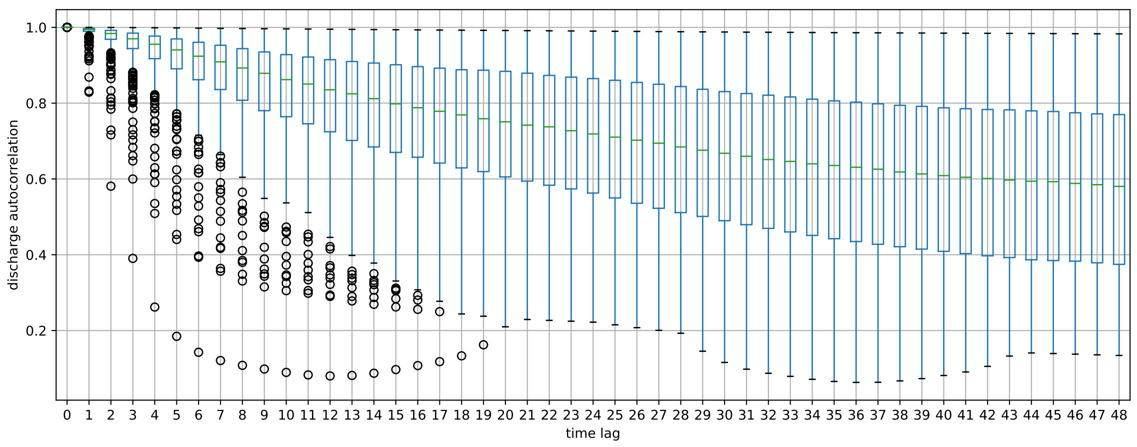


Figure 4.1: Boxplot of discharge autocorrelation for different lag values across gauges. Black circles denote outlier gauges which are consequently harder to forecast.

4.2 River Topology Impact

Our main experiment compares the impact of the different gauge adjacency definitions detailed in Section 3.2. The case of isolated gauges, where the GNN acts like a conventional MLP, serves as a baseline. In addition, we also consider alternative edge orientations, which determine the direction of information flow in the GNN, as it is not a priori clear which one benefits the model most. The *downstream* orientation is given by the dataset, the *upstream* orientation results from reversing all edges, and the *bidirectional* orientation from adding all reverse edges to the forward ones.

4.2.1 Performance Comparison

For all 18 topology combinations, we calculate the average MSE and NSE metrics defined in Section 3.2.3 on the test set and report the results in Table 4.2. Note that they come with a caveat: we only trained one model for each combination due to computational limitations. Future endeavors should account for stochasticity in the network initialization and holdout set selection by training multiple models per combination with different random seeds. Nevertheless, many of the insights that our results provide are so unambiguous that they likely hold across initial conditions.

	downstream		upstream		bidirectional	
adjacency type	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$
isolated	17.0	98.66 %	17.0	98.66 %	17.0	98.66 %
binary	15.3	98.60 %	14.3	98.13 %	23.4	98.72 %
stream length	13.6	98.62 %	14.4	98.38 %	21.5	98.74 %
elevation difference	17.6	98.60 %	18.4	98.40 %	35.2	98.63 %
average slope	13.7	98.62 %	14.1	98.36 %	23.8	98.73 %
learned	11.0	98.66 %	13.7	98.65 %	25.3	98.71 %

Table 4.2: ResGCN performance on different river network topologies. Note how the isolated adjacency type leads to the exact same result across edge orientations as there are no edges, and we initialize with the same random seed.

Surprisingly, model performance shows almost no sensitivity to the choice of graph topology. Most importantly, isolating the gauges does not harm the performance at all. This indicates that the forecasting task for a gauge mainly benefits from the past discharge at that gauge but not from the discharge at neighboring gauges. The river graph topology appears to make no difference. Even when the model is allowed to learn an optimal edge weight assignment, it does not manage to outperform the baseline.

In absolute terms, the performance is very good throughout all combinations, as indicated by the high average NSE scores. Given that MSE is a squared measure, the models make average errors on the order of magnitude of $10\,\text{m}^3/\text{s}$. Potentially, the forecasting task on the LamaH-CE dataset is “too easy”, making a simple MLP hit the performance ceiling already. Future work is encouraged to check whether this behavior persists when the model receives meteorological inputs as well.

4.2.2 Learned Weights

The case of learned edge weights is of particular interest. They were initialized by drawing from the uniform distribution in $[0.9, 1.1]$ to arrange them neutrally around one while still introducing sufficient noise to break symmetry. Whenever learned weights get negative during training, we clip them by replacing them with zero. Figure 4.2 reveals the development of their mean and spread over time during training. For a scale-independent measure of spread, we divide the standard deviation by the mean.

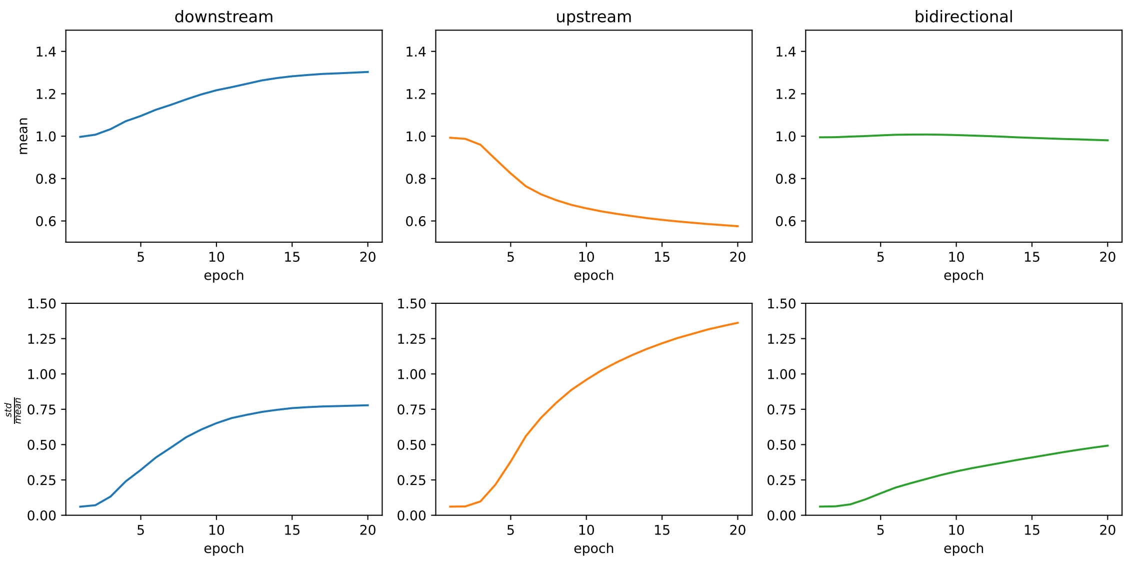


Figure 4.2: Mean and spread of learned weights during training.

Owing to their initialization, the weights start with a mean of one and a small standard deviation across edges. While with a downstream orientation, the weights begin to grow on average, they shrink with an upstream orientation and stay approximately constant in the bidirectional case. However, in all cases, the mean appears to converge to a value on the same order of magnitude. Furthermore, the spread of the weights increases significantly in all three cases. This suggests that a constant weight assignment on edges, like the binary one, is not an optimum for the GNN.

To see if the learned weights exhibit any similarities with the physical weights, we calculate Pearson correlation coefficients for all combinations. Table 4.3 shows that none of the physical weightings correlate well with the learned weights. Hence, the physical weights are not in the optimal regime either.

	LEARNED WEIGHTS		
PHYSICAL WEIGHTS	DOWNSTREAM	UPSTREAM	BIDIRECTIONAL
STREAM LENGTH	0.048	0.140	0.034
ELEVATION DIFFERENCE	0.153	-0.211	-0.142
AVERAGE SLOPE	0.093	-0.246	-0.146

Table 4.3: Pearson correlation between learned weights and physical weights.

4.2.3 Dirichlet Energy

To gain insight into the dynamics of vertex representations in the GNN's latent space, we track the graph Dirichlet energy (Definition 20) of the hidden layer activations during the forward pass. Figure 4.3 shows that for all topology combinations except those with isolated gauges, the Dirichlet energy increases monotonically with positive curvature. This confirms that thanks to the residual connection, over-smoothing is avoided. Instead, a strong sharpening process occurs where features of neighboring gauges become more dissimilar in latent space throughout the layers. Given that the graph Dirichlet energy is the sum of unnormalized Rayleigh quotients of the graph Laplacian, we infer that high-frequency dynamics dominate.

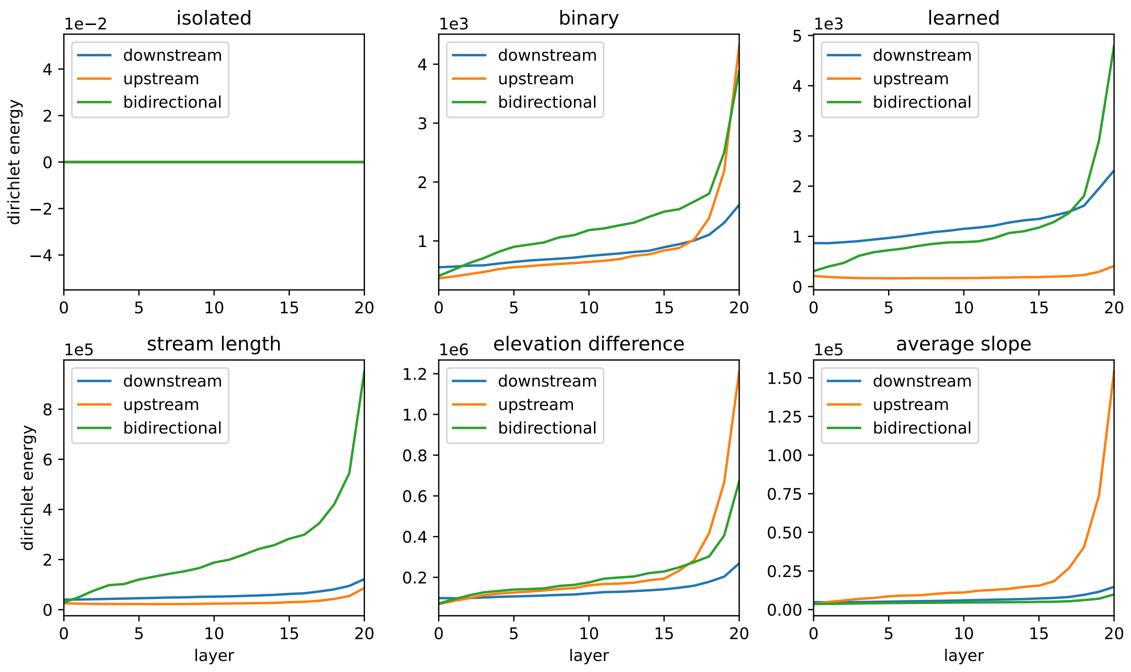


Figure 4.3: Graph Dirichlet energy evolution for different adjacency types, averaged over the test set. In the isolated case, it is trivially zero, as no edges exist.

4.3 Network Depth Impact

The rationale for setting the number of layers to $N = 20$ was to allow information to propagate across the entire river network. However, since removing all edges from the graph does not deteriorate the performance (cp. Table 4.2), we can also consider shallower neural networks. In particular, we want to exclude that the considerable depth is causing the GCN to not outperform the baseline MLP due to more general issues with training very deep networks. In this case, a GCN with fewer layers could profit more from the graph structure, despite not achieving global information propagation. Hence, we train a residual GCN with the default hyperparameters from Table 4.1 where we only vary the number of layers in steps of two from 2 to 20. The resulting MSE and NSE scores across gauges are shown in Figure 4.4.

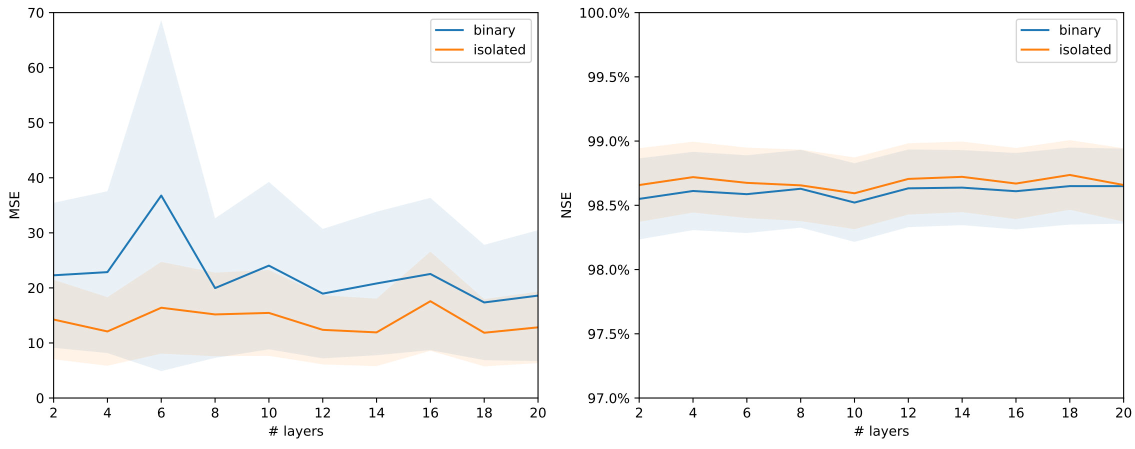


Figure 4.4: ResGCN performance with varying depth, averaged over gauges. Shaded areas indicate 95 % confidence intervals.

The experiment provides two insights. First, the inability of the binary model to outperform the isolated one is consistent across network depths so that we can rule out training issues. Second, the performance is independent of model depth, which means that the larger receptive field achieved by more layers does not help. Both corroborate the previous observations that ResGCN fails to take advantage of the graph structure.

4.4 Worst Gauge Investigation

Recall from Section 4.1 that gauge #85 in the dataset exhibits particularly low autocorrelation of discharge with time lags of more than two hours. Consequently, the performance on this gauge of all trained models is considerably below the mean. For instance, the binary-downstream model achieves its worst NSE of only 70.34 % on this outlier gauge. To better understand the model's predictive behavior, we produce recurrent forecasts according to Algorithm 3 for 24 hours into the future and compare them with the ground truth observations. Figure 4.5 shows this comparison for a random subset of 18 samples from the test set.

First, we observe that the outlier gauge is characterized by an abundance of sudden spikes, which are inherently hard to forecast for any predictor. The gauge is likely located behind a floodgate. As a result, the forecasting performance is mediocre, with the recurrent forecast often missing spikes and drops. In addition, the rolling window forecasts more generally suffer from exponential accumulation as predictions are increasingly used as input. In fact, the very last prediction for the 24-hour lead time merely depends on a single true observation. However, the alternative approach of training 24 models for each choice of lead time is computationally prohibitive and thus left for future work.

4.5 Using GRAFF

To see if the theoretical benefits of the Gradient Flow Framework (cp. Section 2.2.2) carry over to practice, we repeat the topology comparison from Section 4.2 with a GRAFFNN model architecture. We set the step size to $\tau = 1$ and share the channel mixing matrices across layers to match the ODE discretization interpretation. The results in Table 4.4 are essentially equivalent to those of the ResGCN and exhibit the same tendencies. This is true as well for the evolution of dirichlet energy and the recurrent forecasting performance, which is why we do not include the very similar figures here. The fact that GRAFFNN achieves equal performance is remarkable, as it has 20 times fewer parameters due to sharing. This indicates that the dynamical systems viewpoint is adequate for the forecasting task.

	Downstream		Upstream		Bidirectional	
Adjacency type	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$	$\overline{\text{MSE}} \downarrow$	$\overline{\text{NSE}} \uparrow$
isolated	17.9	98.58 %	17.9	98.58 %	17.9	98.58 %
binary	38.6	98.65 %	22.9	98.57 %	31.8	98.64 %
stream length	33.9	98.65 %	17.1	98.64 %	29.7	98.66 %
elevation difference	42.6	98.56 %	18.3	98.59 %	35.3	98.66 %
average slope	45.0	98.64 %	15.8	98.64 %	27.9	98.68 %
learned	14.3	98.70 %	15.3	98.64 %	25.1	98.68 %

Table 4.4: GRAFFNN performance on different river network topologies. Note how the isolated adjacency type leads to the exact same result across edge orientations as there are no edges, and we initialize with the same random seed.

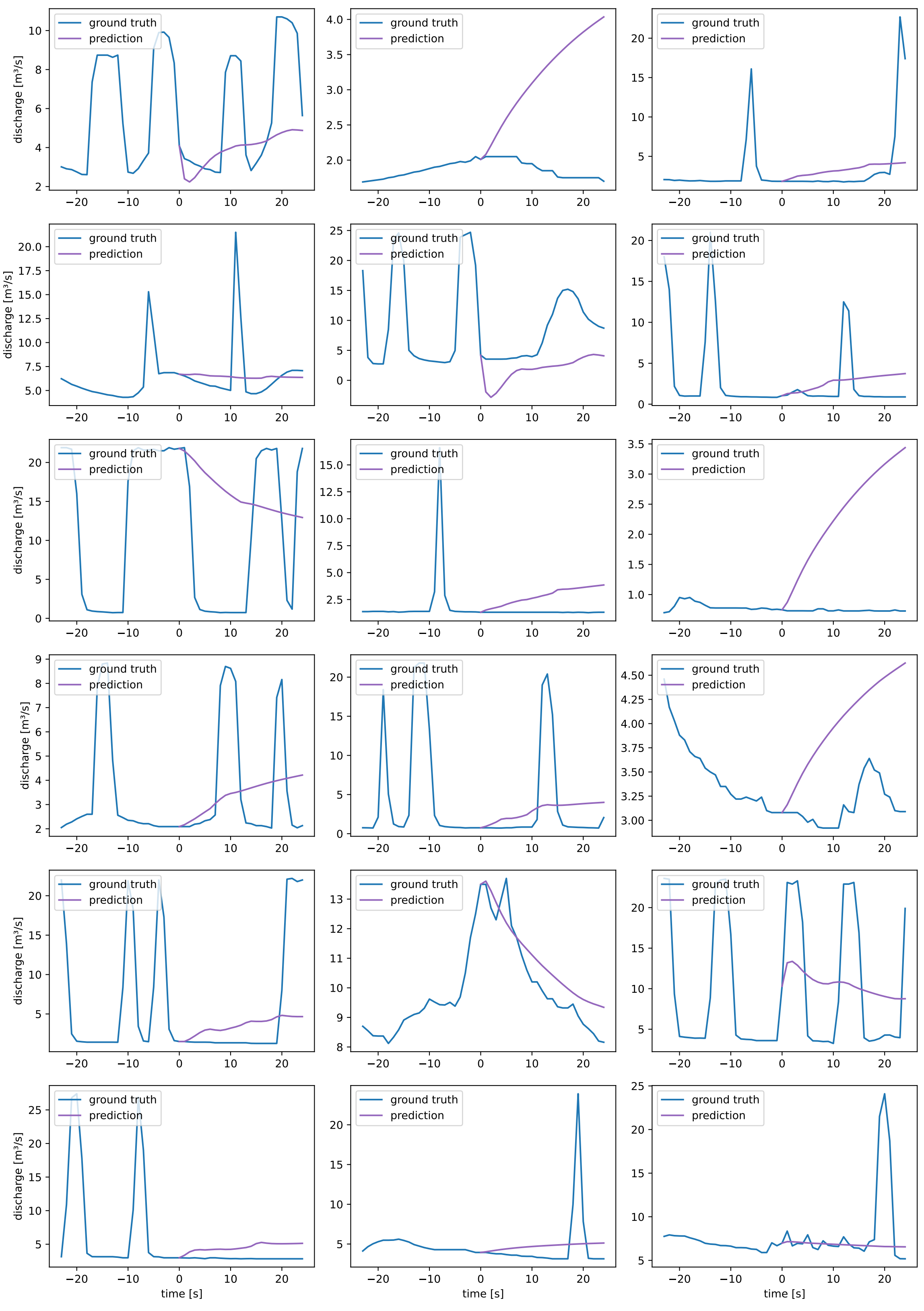


Figure 4.5: Rolling window forecasts for outlier gauge #85 with a 24-hour lead time. Negative time indicates past and positive time indicates future discharge.

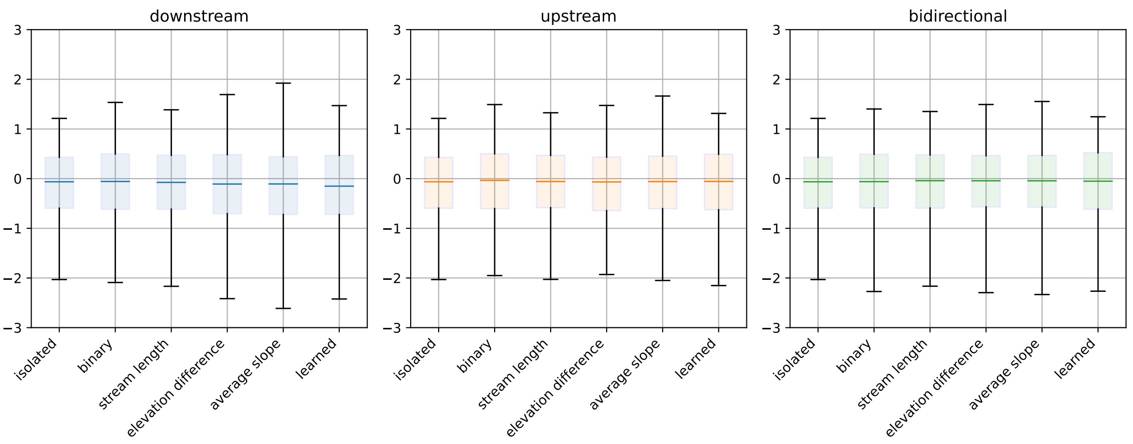


Figure 4.6: Boxplots of the eigenvalues of the internal channel mixing matrix of all trained GRAFNNs.

Since GRAFF provides us with an attraction-repulsion paradigm, we take a closer look at the eigenvalues of the internal channel mixing matrix. Figure 4.6 visualizes their distribution for each topology combination. There is a bias toward negative eigenvalues, and we find that in most combinations, the spectrum consists of 30 positive and 34 negative eigenvalues. Hence, both attractive and repulsive forces act on vertex representations in the latent space of the GRAFNN, with the repulsive ones slightly dominating.

Chapter 5 Conclusion

In this work, we explored the applicability of GNNs to holistic flood forecasting in a river network graph. Based on the LamaH-CE dataset, we framed a supervised vertex autoregression machine learning task for predicting future discharge at all gauging stations in the graph given past observations. By modifying the adjacency matrix, we compared the impact of different adjacency definitions on the prediction performance.

Our results reveal that the impact of river topology is negligible. The GNN performs equally well even when all edges are removed from the graph, which makes it act like an MLP. It does not benefit from weighted edges that resemble physical relationships between gauges. When the model is allowed to jointly learn the weights along with the other parameters, they correlate with neither constant weights nor any of the physical weightings given by the dataset. A depth study shows that the results are not caused by issues with training deep models but prove consistent throughout any number of layers. Investigations on a challenging outlier gauge show that the GNN struggles to predict sudden discharge spikes. By employing GRAFF, we find that parameter sharing across layers does not deteriorate performance and that a mixture of attractive and repulsive forces act on vertex representations in the latent space of the GNN, while the graph Dirichlet energy evolution indicates high-frequency dynamics.

There is a myriad of avenues left to investigate in future work. On the modeling front, recent approaches catering to the DAG structure of a river network such as DGCN (Tong et al., 2020), MagNet (Zhang et al., 2021), and DAGNN (Thost & Chen, 2021) are worth investigating. Alternatively, given that a river network graph is very sparse, it might help to replace the edge set with its transitive closure in a vanilla GCN. Furthermore, one might consider shifting the task definition to a graph-level discharge prediction at the most downstream gauge. In any case, multiple training repetitions with different seeds should be conducted to account for stochasticity if the computational resources are available.

On the data front, future work is advised to exploit the meteorological time series contained in LamaH-CE as an additional input to the model. Beyond LamaH-CE, there is a broader issue to be addressed: we used a river network dataset from central Europe as discharge measurements are readily available there for long time periods. However, the regions most affected by floods are typically in low-income countries where data is scarce. Future work should make an effort to install gauges and collect large-scale datasets in these regions to enable more relevant studies and alleviate the Western bias prevalent in research.

Bibliography ∇

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

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

Cai, C. and Wang, Y. A Note on Over-Smoothing for Graph Neural Networks. *arXiv preprint arXiv:2006.13318*, 2020.

Centre for Research on the Epidemiology of Disasters (CRED). Disasters in Numbers 2022. Technical report, 2022.

Chang, F.-J., Hsu, K., and Chang, L.-C. *Flood Forecasting Using Machine Learning Methods*. MDPI, February 2019.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and Deep Graph Convolutional Networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, July 2020.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural Ordinary Differential Equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Chung, F. R. *Spectral graph theory*, volume 92. American Mathematical Society, 1997.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Di Giovanni, F., Rowbottom, J., Chamberlain, B. P., Markovich, T., and Bronstein, M. M. Graph Neural Networks as Gradient Flows: understanding graph convolutions via energy. 2022. Publisher: arXiv Version Number: 3.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Frame, J. M., Kratzert, F., Klotz, D., Gauch, M., Shalev, G., Gilon, O., Qualls, L. M., Gupta, H. V., and Nearing, G. S. Deep learning rainfall–runoff predictions of extreme events. *Hydrology and Earth System Sciences*, 26(13):3377–3392, July 2022.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR.

Goodfellow, I. J., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

Graphcore. PopTorch and PopTorch Geometric, 2020. GitHub repository.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

Hu, S. X., Zagoruyko, S., and Komodakis, N. Exploring weight symmetry in deep neural networks. *Computer Vision and Image Understanding*, 187:102786, October 2019.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

Klingler, C., Schulz, K., and Herrnegger, M. LamaH-CE: LArge-SaMple DAta for Hydrology and Environmental Sciences for Central Europe. *Earth System Science Data*, 13(9):4529–4565, September 2021.

Kratzert, F., Klotz, D., Herrnegger, M., Sampson, A. K., Hochreiter, S., and Nearing, G. S. Toward Improved Predictions in Ungauged Basins: Exploiting the Power of Machine Learning. *Water Resources Research*, 55(12):11344–11354, December 2019a.

Kratzert, F., Klotz, D., Shalev, G., Klambauer, G., Hochreiter, S., and Nearing, G. Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets. *Hydrology and Earth System Sciences*, 23 (12):5089–5110, December 2019b.

Kratzert, F., Klotz, D., Gauch, M., Klingler, C., Nearing, G., and Hochreiter, S. Large-scale river network modeling using Graph Neural Networks. Technical report, March 2021.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–50. Springer, 2002.

Mosavi, A., Ozturk, P., and Chau, K.-w. Flood Prediction Using Machine Learning Models: Literature Review. *Water*, 10(11):1536, October 2018.

Nash, J. and Sutcliffe, J. River flow forecasting through conceptual models part I — A discussion of principles. *Journal of Hydrology*, 10(3):282–290, April 1970.

Nevo, S., Elidan, G., Hassidim, A., Shalev, G., Gilon, O., Nearing, G., and Matias, Y. ML-based Flood Forecasting: Advances in Scale, Accuracy and Reach, 2020. _eprint: 2012.00671.

Nevo, S., Morin, E., Gerzi Rosenthal, A., Metzger, A., Barshai, C., Weitzner, D., Voloshin, D., Kratzert, F., Elidan, G., Dror, G., Begelman, G., Nearing, G., Shalev, G., Noga, H., Shavitt, I., Yuklea, L., Royz, M., Giladi, N., Peled Levi, N., Reich, O., Gilon, O., Maor, R., Timnat, S., Shechter, T., Anisimov, V., Gigi, Y., Levin, Y., Moshe, Z., Ben-Haim, Z., Hassidim, A., and Matias, Y. Flood forecasting with machine learning models in an operational framework. *Hydrology and Earth System Sciences*, 26(15):4013–4032, August 2022.

Oono, K. and Suzuki, T. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*, 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 Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Sit, M., Demiray, B. Z., Xiang, Z., Ewing, G. J., Sermet, Y., and Demir, I. A comprehensive review of deep learning applications in hydrology and water resources. *Water Science and Technology*, 82(12):2635–2670, December 2020.

Thost, V. and Chen, J. Directed Acyclic Graph Neural Networks. In *International Conference on Learning Representations*, 2021.

Tong, Z., Liang, Y., Sun, C., Rosenblum, D. S., and Lim, A. Directed Graph Convolutional Network. *arXiv preprint arXiv:2004.13970*, 2020.

United Nations Office for Disaster Risk Reduction (UNDRR). Global Assessment Report on Disaster Risk Reduction – Our World at Risk: Transforming Governance for a Resilient Future. Technical report, 2022.

United Nations Office for Disaster Risk Reduction (UNDRR) and Centre for Research on the Epidemiology of Disasters (CRED). The Human Cost of Weather Related Disasters. Technical report, 2015.

Vreugdenhil, C. B. *Numerical methods for shallow-water flow*, volume 13. Springer Science & Business Media, 1994.

Wu, L., Cui, P., Pei, J., and Zhao, L. (eds.). *Graph neural networks: foundations, frontiers, and applications.* Springer, Singapore, 2022.

Wu, W. *Computational River Dynamics.* CRC Press, 0 edition, November 2007.

Xhonneux, L.-P., Qu, M., and Tang, J. Continuous Graph Neural Networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10432–10441. PMLR, July 2020.

Yilmaz, K. K., Gupta, H. V., and Wagener, T. A process-based diagnostic approach to model evaluation: Application to the NWS distributed hydrologic model. *Water Resources Research*, 44(9), September 2008.

Zhang, X., He, Y., Brugnone, N., Perlmutter, M., and Hirn, M. MagNet: A Neural Network for Directed Graphs. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P. S., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 27003–27015. Curran Associates, Inc., 2021.

Zhou, D. and Schölkopf, B. Regularization on Discrete Spaces. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Kropatsch, W. G., Sablatnig, R., and Hanbury, A. (eds.), *Pattern Recognition*, volume 3663, pp. 361–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. Series Title: Lecture Notes in Computer Science.