# Incorporating Sequential and Geometric Structure into Deep Neural Networks


Álvaro Arroyo Núñez

University College

University of Oxford

A thesis submitted for the degree of

*Doctor of Philosophy*

Trinity 2025

*A mis padres,*
*Gustavo Arroyo y María Jesús Núñez*

*En memoria de mis abuelos,*
*Santos Núñez y María Jesús García Crespo*

# Acknowledgements

Four years ago, I could not have imagined all that has happened, nor the remarkable people I would have the good fortune to meet and work with. Although a thesis is typically regarded as a solo achievement, I firmly believe that the researcher I have become has been shaped not only by my own ambitions and a measure of good fortune, but also by the many individuals who have supported me — both throughout this thesis and long before.

First and foremost, I am immensely grateful to my supervisors Xiaowen Dong and Álvaro Cartea, for their support and confidence in me. During a challenging period, Xiaowen welcomed me into his lab and provided the tools and freedom I needed to grow as a researcher. On the other hand, Álvaro opened the doors of the Oxford–Man Institute at the very beginning of my journey and took an active role in the development of our joint papers, as well as giving great support along the way. Beyond the academic realm, I will always cherish the great times we shared. I am likewise indebted to Michael Bronstein and Pierre Vandergheynst for their involvement in the final stages of this thesis. I also appreciated stimulating conversations with senior colleagues such as Mihai Cucuringu, Anthony Ledford, Leandro-Sánchez Betancourt, Pietro Liò, and Samuel Cohen. My thanks extend to Jen Desmond for her help over these four years.

Throughout this time, I enjoyed many memorable moments with friends and colleagues. The first part of my thesis was marked by good times with Fayçal Drissi, Patrick Chang, Marcello Monga, Harrison Waldon, Anthony Coache, and Fernando Moreno-Pino. The second part was particularly enjoyable thanks to Sergio Calvo Ordóñez, Jonathan Plenk, Gabriel García Arenas, Federico Barbero, Gerardo Durán-Martín, Scott Le Roux, Ben Finkelshtein, and Jacob Bamberger, with whom I shared many discussions. I am also grateful to the other students at the Oxford–Man Institute and Eagle House, who made my days there pleasant.

I owe a special debt to my co-authors — Fernando Moreno-Pino, Alessio Gravina, Federico Barbero, Benjamin Gutteridge, Haitz Sáez de Ocáriz Borde — whose collaboration was instrumental in delivering the papers that comprise this thesis. I also thank Ali

Hariri, Moshe Eliasof, Christos Perivolaropoulos, Petar Veličković, and Razvan Pascanu for insightful discussions and collaboration on projects not included in this thesis.

I am thankful to those who have generously shared their knowledge and provided mentorship along the way: Angelos Filos, Philip Ball, Samuel Kessler, Bryan Lim, Daniel Poh, Vincent Tan, and Guillermo Ortiz-Jiménez. A special acknowledgment goes to Bruno Scalzo, who introduced me to research five years ago when there was nothing in it for him; that act of kindness was one of the most decisive factors in shaping my career. I hope I get the chance to work with you again. Even before that, Sharon Conlisk believed in me during my teenage years; I am very grateful for her faith.

I also wish to thank those closest to me personally, who have played a profound role in my development over these past years. To my extended family, for the many wonderful moments we have shared; to my friends David, Ismael, Iñigo G., and Carlos, for the unforgettable times in Oxford; and to Ángel, Pepe, Iñigo J., Alberto, Marcos, Samuel, and Gem, who have been constant companions and friends. I am especially grateful to Mariana Muniz for her support during challenging times. Above all, I dedicate this thesis to my parents and to the memory of my maternal grandparents, whose encouragement, guidance, and values have helped me every step of the way.

Finally, I gratefully acknowledge financial support received from (a) Rafael del Pino Foundation (b) Oxford-Man Institute (c) G-Research (d) University College.

*Alvaro Arroyo Núñez*
*Eagle House, Walton Well Road, Oxford, May 2025*

# Abstract

Modern deep learning models have achieved great success, but they struggle with *structured* inputs unless they ingest massive amounts of data, or are guided by inductive biases that reflect the data's inherent sequential or geometric structure. This thesis focuses on improving model performance in settings like time series and graph-structured data, which are common in real-world domains such as finance or healthcare. In particular, it addresses key challenges in these domains, such as the presence of long-range dependencies, irregular sampling, over-smoothing, and over-squashing, aiming to design architectures that are both expressive, performant, and computationally efficient.

We divide this work into two sections. The first section is concerned with learning from data with *sequential* structure, and tackles several well-known problems associated with using deep learning in this data modality. In the second section, we focus on *geometric* deep learning, where we build a bridge between the recurrent and graph deep learning worlds in §5, and also study an independent problem in the area of manifold learning.

Firstly, in §3, we develop a deep survival analysis method for limit order books that leverages a convolutional-Transformer to generate latent representations and a monotonic neural network to abide by the constraints of the function being modelled. By integrating local convolutional filters with a self-attention mechanism and a right-censored log-likelihood loss, our approach effectively captures the complex temporal dynamics inherent in high-frequency financial data, yielding superior estimates of order fill probabilities compared to standard benchmarks.

Next, in §4, we introduce *Rough Transformers*—a new architecture that integrates signature transforms into the Transformer model. By extracting both local and global signature features, the Rough Transformer efficiently models irregularly sampled time-series data while dramatically reducing computational cost.

In the realm of graph neural networks, we revisit the over-smoothing and over-squashing phenomena by analyzing their connection with vanishing gradients in §5. We show that the contractive nature of normalized adjacency matrices leads to extreme gradient vanishing and representational collapse. By reinterpreting GNNs as state-space models, we introduce GNN-SSMs that provide explicit control over the layer-wise Jacobian spectrum. This state-space formulation mitigates over-smoothing and enhances long-range information propagation, effectively bridging ideas from recurrent and graph learning.

Finally, in §6, we propose *Neural Latent Geometry Search (NLGS)*, a framework for automatically identifying optimal latent space geometries. Modeling latent spaces as products of constant-curvature manifolds, we introduce a principled measure based on the Gromov–Hausdorff distance to compare candidate latent geometries. By constructing a geometry-informed graph search space and applying Bayesian optimization, our approach efficiently discovers the product manifold signature that best aligns with the underlying data structure, as demonstrated on tasks such as image reconstruction and latent graph inference.

Collectively, these contributions seek to advance deep learning by integrating temporal and geometric inductive biases into network architectures, enhancing model expressiveness, scalability, and performance across diverse real-world applications.

# Declaration of Originality

I, Álvaro Arroyo Nuñez, hereby declare that, except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Álvaro Arroyo Núñez,
Trinity 2025

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Modern deep learning models have achieved remarkable success across a wide range of applications. However, when it comes to handling *structured data*, these models often encounter significant challenges. Traditional architectures can struggle in these settings and frequently require the integration of *inductive biases* — predefined assumptions or constraints that guide the learning process — to achieve sample-efficient learning and improved performance. In this thesis, we will focus on incorporating these inductive biases in settings when data exhibit *sequential* or *geometric* structure. These settings are prolific in real-world applications, such as finance, where remembering information from the distant past and understanding spillover effects in asset networks can improve predictions.

Within *sequence* or *temporal* modeling, real-world time series often have long-range dependencies and irregular sampling intervals, which can be problematic for standard recurrent or Transformer architectures [260]. Furthermore, attention-based models like Transformers struggle with efficiency on long sequences due to their quadratic complexity, while recurrent networks suffer from vanishing gradients [142] that hinder learning long-term information propagation. Similarly, in *geometric* settings, graph-structured data and other non-Euclidean domains pose difficulties for deep neural networks. In particular, Graph Neural Networks (GNNs) that rely on the message-passing paradigm [162, 261] are known to suffer from *over-smoothing* (node representations becoming indistinguishable in deep layers) and *over-squashing* (inability to transmit information from distant parts of the graph) [82]. These phenomena limit the depth and expressiveness of GNNs. Addressing such sequential and geometric issues is an open problem. As such, we ask the question:

*How can we design deep networks that capture long-term sequential structure and complex geometric relationships without succumbing to computational or optimization issues?*

## 1.2 Contributions and Outline

This dissertation aims to address these challenges in four chapters, each representing a research paper. While the primary motivation for this work arises from problems in quantitative finance, our goal extends beyond this discipline. Instead, we develop a set of methodological innovations that not only address challenges in the field but also have broad applicability across various disciplines. Key contributions as first or co-first author include:

- **Deep sequence modeling for limit order books [8]:** A convolutional-Transformer architecture for survival analysis on high-frequency trading data, which predicts the fill probabilities of limit orders with greater accuracy than traditional static methodologies [173]. This chapter aims to make a dent in two quantitative finance problems (optimal trade execution and market making) by capturing complex time-varying patterns in limit order books and using them to improve predictions of the fill probability.
- **Efficient continuous-time transformer models [196]:** We introduce the *Rough Transformer*, a lightweight Transformer variant that operates on continuous-time representations of irregularly sampled sequences. By using multi-view signature attention to capture multi-scale dependencies, it achieves improved long-range forecasting performance at a fraction (two to three orders of magnitude in some experiments) of the computational cost of standard Transformers and other continuous-time deep learning models [227, 161]. This advances sequence modeling efficiency for applications like irregular financial time-series data, like that observed in Limit Order Books. We also make links to the graph machine learning literature to analyze the representational collapse of representations and the interactions between different channels in the time series.
- **Bridging sequence models and graph networks [9]:** A unified theoretical analysis that connects recurrent sequence models with GNNs through the lens of the vanishing gradient problem. We theoretically characterize the link between vanishing gradients and the over-smoothing and over-squashing problems in deep graph networks, and propose a state-space GNN model that mitigates these issues. By adapting techniques from sequence models to graph learning, this work enables deeper, more stable GNNs and provides insight into training dynamics.

- **Neural latent geometry search [77]:** A novel methodology to automatically infer the optimal latent space geometry for deep models. Rather than assuming a Euclidean latent space, we perform Bayesian optimization over combinations of curved manifolds (hyperbolic, spherical, etc.), using the Gromov–Hausdorff-based distance to guide the search. This approach shows that aligning latent geometry with underlying data structure can significantly boost model performance.

## 1.3 Publications

The thesis is mostly based on a series of articles, as either first author or co-first author ($\star$ indicates co-first authorship):

**Chapter 3.** *Deep Attentive Survival Analysis in Limit Order Books: Estimating Fill Probabilities with Convolutional-Transformers.* **A. Arroyo**$^\star$, A. Cartea, F. Moreno-Pino$^\star$, S. Zohren. In: Quantitative Finance, 1-23. 2024.

**Chapter 4.** *Rough Transformers: Lightweight and Continuous Time Series Modelling through Signature Patching.* F. Moreno-Pino$^\star$, **A. Arroyo**$^\star$, H. Waldon$^\star$, X. Dong, A. Cartea In: Advances in Neural Information Processing Systems (NeurIPS 2024). 2024.

**Chapter 5.** *On Vanishing Gradients in GNNs: Bridging Recurrent and Graph Learning.* **A. Arroyo**$^\star$, A. Gravina$^\star$, B. Gutteridge, F. Barbero, C. Gallicchio, X. Dong, M. Bronstein, P. Vandergheynst. In: Advances in Neural Information Processing Systems (NeurIPS 2025). 2025.

**Chapter 6.** *Neural Latent Geometry Search: Product Manifold Inference via Gromov-Hausdorff-Informed Bayesian Optimization.* H. Borde$^\star$, **A. Arroyo**$^\star$, I. Morales, I. Posner, X. Dong. In: Advances in Neural Information Processing Systems (NeurIPS 2023). 2023.

Other work not included in this thesis can also be found below:

- *Why do LLMs Attend to the First Token?* F. Barbero$^\star$, **A. Arroyo**$^\star$, X. Gu, C. Perivolaropoulos, M. Bronstein, P. Veličković, Razvan Pascanu. In: Second Conference on Language Modeling (COLM 2025). 2025.

- *Return of ChebNet: Understanding and Improving an Overlooked GNN on Long Range Tasks* A. Hariri*, **A. Arroyo***, A. Gravina*, C.B. Schönlieb, D. Bacciu, K. Azizzadenesheli, X. Dong, P. Vandergheynst. Advances in Neural Information Processing Systems (NeurIPS 2025, Spotlight). 2025.

# Chapter 2

# Background & Related Work

## 2.1 Introduction

In this chapter, we introduce the fundamental concepts and methodologies for modelling sequential and graph-structured data. We first provide background on sequence modelling techniques—including recurrent neural networks, attention mechanisms, and transformers—while discussing common challenges such as the vanishing and exploding gradient problem. We then review graph modelling approaches based on message passing neural networks. Furthermore, we will place particular emphasis on outlining the similarities between recurrent and message passing models and interpreting them through a common framework. This overview lays the groundwork for understanding the methods and challenges addressed in subsequent chapters.

## 2.2 Background on Sequence Modelling

Sequence modelling is a foundational concept in machine learning that deals with causally ordered data, such as text [1], speech [114], or time series [177]. While such sequential tasks have often relied on handcrafted priors or simple linear models with closed form solutions, the increased availability of compute power and data has led to the advent of deep learning approaches. Here, we will cover some of the most common frameworks and architectures in this domain.

### 2.2.1 Recurrent Neural Networks

A Recurrent Neural Network is designed to handle sequential data by maintaining a hidden state $\mathbf{h}^{(k)} \in \mathbb{R}^{d_h}$ that captures information from previous time steps. Historically, RNN-based architectures have been extensively used in sequence modeling [66], achieving strong

results on a variety of natural language processing tasks. One of the simplest RNN variants, the Elman RNN [97], which updates its internal state according to

$$\mathbf{h}^{(t)} = \sigma\left(\mathbf{W}_h\,\mathbf{h}^{(t-1)} + \mathbf{W}_x\,\mathbf{x}^{(t)} + \mathbf{b}_h\right), \tag{2.1}$$

$$\mathbf{y}^{(t)} = \sigma\left(\mathbf{W}_y\,\mathbf{h}^{(t)} + \mathbf{b}_y\right), \tag{2.2}$$

where $\mathbf{h}^{(t)}$ represents the hidden state at time $t$, $\mathbf{y}^{(t)}$ is the output at that time, and $\mathbf{W}_h$, $\mathbf{W}_y$ and $\mathbf{W}_x$ are weight matrices, and $\mathbf{b}_h$ and $\mathbf{b}_y$ are the bias vectors. The activation function $\sigma(\cdot)$ introduces nonlinearity to the state update. Although the basic RNN model captures sequential dependencies implicitly through its recursive update, more advanced models such as Long Short-Term Memory (LSTM) networks [143] and Gated Recurrent Units (GRUs) [64] have been introduced to address issues like vanishing gradients, which we will introduce more formally in the following subsection.

### 2.2.2 The Vanishing and Exploding Gradient Problem

While RNNs have been successfully trained and used in short sequences, long-range sequence modeling is a challenging problem as a consequence of the *exploding and vanishing gradients problem* [22, 143, 210]. This occurs because, during backpropagation through time (BPTT), the gradients of the hidden states are computed by repeatedly applying the chain rule across many time steps. As a result, the process involves multiplying a long sequence of Jacobians that are correlated to each other due to being the Jacobian of the same update rule. When this sequence is long, the accumulated product can either shrink rapidly toward zero or blow up uncontrollably, which leads to uncontrolled gradient propagation. In particular, following [210], we let $\mathcal{L}^{(t)}$ denote the loss at time step $t$. The total loss over a sequence of length $T$ is then given by

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}^{(t)}. \tag{2.3}$$

Accordingly, the gradient of the total loss with respect to the parameters $\boldsymbol{\theta}$ can be expressed as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \left( \frac{\partial \mathcal{L}^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \boldsymbol{\theta}} \right). \tag{2.4}$$

As identified by [210], a major issue arises from the product Jacobian,

$$\mathbf{J} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}} = \prod_{i=k+1}^{t} \mathbf{J}_i. \tag{2.5}$$

If $\|\mathbf{J}_k\|_2 \approx \lambda$ for all layers, then

$$\|\mathbf{J}\|_2 \leqslant \lambda^{t-k}. \tag{2.6}$$

Thus, it is necessary that $\lambda \approx 1$ for gradients to neither vanish nor explode—a condition often referred to as the *edge of chaos*. Specifically, if $\lambda < 1$ for certain terms in the gradient sum, these terms will disappear and will be absent from the gradient calculation. Conversely, if $\lambda > 1$, these gradients will explode and the gradient norm will increase greatly. RNN architectures such as LSTMs [143] or GRUs [64] mitigate vanishing gradients via gated additive recurrence, but remain susceptible to exploding gradients and also struggle with long sequences. As such, a very relevant line of research has been dedicated to fixing this issue, examples of which include norm-preserving weight constraints [5, 140, 56, 175], initialization schemes [254], and physics-inspired inductive biases [100, 156].

A fundamental limitation of RNNs is how information must propagate: at each time step, the network must compute its hidden state based on the output of the previous time step, resulting in a sequential bottleneck. This inherent temporal dependency means that computations cannot be parallelized across the sequence—each step must wait for the one before it to complete. As a result, training becomes slow, especially for long sequences. By contrast, Transformers enabled parallelization during training, and therefore were able to take advantage of modern parallel hardware (e.g., GPUs or TPUs). Recently, a strand of work has adapted RNNs by replacing the non-linear recurrence with a linear counterpart followed by an MLP block, enabling efficient parallel computations during training, while maintaining the fast – compared to Transformers – inference speed typical to RNNs. The earliest instances of this framework are Structured State Space Models (SSMs), which rely on a parametrization derived from discretizing a continuous time linear dynamical system and rely on a deterministic initialization of the recurrent weights (known as the HiPPO matrix [123]). The effectiveness of these models has been linked to the eigenspectrum of the Jacobian matrix, as demonstrated in [208] which replicates the earlier finding in the setting of linear RNNs while benefiting from fast training and inference through the use of parallel scan algorithms [188]. Interestingly, the design of Linear Recurrent Units (LRUs) in [208] shares conceptual similarities with techniques from the reservoir computing literature [148, 251], particularly in how recurrent dynamics are leveraged to guarantee stability (also known as the "echo-state property").[1]

[1]It should be noted that the propagation of gradients through the network has also been amply studied in the feedforward and deep linear network regime. This includes the seminal work of [234, 237, 212], which characterized the dynamics of learning based on initialization and choice on nonlinearities using ideas from Random Matrix Theory (RMT).

### 2.2.3 Attention

The attention mechanism was introduced in the context of machine translation [64, 14], enabling sequence modeling architectures to focus on specific parts of the input when making predictions. In a self-attention framework, each token's representation is dynamically updated by attending over all positions in the input sequence. In the original attention formulation [13] each position $t$ computed a new context-aware representation $\mathbf{c}_t$ as a weighted sum of all hidden states:

$$\mathbf{c}_t = \sum_{t'=1}^{T} \alpha_{t,t'} \, \mathbf{h}_{t'}, \tag{2.7}$$

where the vector of attention weights

$$\boldsymbol{\alpha}_t = [\alpha(t,1), \ldots, \alpha(t,T)] \tag{2.8}$$

assigns a relevance score to each hidden state $\mathbf{h}_{t'}$. The weights $\alpha_{t,t'}$ are computed in two steps. First, an alignment score is calculated as:

$$e_{t,t'} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}_a \, \mathbf{h}_t + \mathbf{U}_a \, \mathbf{h}_{t'}\right), \tag{2.9}$$

where $\mathbf{h}_t$ acts as the *query* vector for position $t$, $\mathbf{h}_{t'}$ serves as the *key* vector for position $t'$, $\mathbf{W}_a$ and $\mathbf{U}_a$ are weight matrices, $\mathbf{v}_a$ is a linear projection. Next, the alignment scores are normalized via a softmax function to obtain the attention weights:

$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{k=1}^{T} \exp(e_{t,k})}. \tag{2.10}$$

The resulting context vector $\mathbf{c}_t$ then encapsulates information from the entire input sequence, weighted according to the relevance of each position with respect to position $t$. This self-attention mechanism allows the model to integrate global contextual information without relying on a separate encoder.

### 2.2.4 Transformers

The appearance of the attention mechanism led to the introduction of the Transformer architecture [260]. This model relies on the self-attention mechanism, which is executed in parallel by multiple heads, forming a multi-head Transformer. Each self-attention head applies scaled dot-product attention to calculate similarity scores between different inputs in the sequence. For the $i^{\text{th}}$ head, the scaled dot-product attention is defined as

$$\mathbf{h}_i = \text{Attention}\left(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i\right) = \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}}\right)\mathbf{V}_i, \tag{2.11}$$

where $d_k$ is a scaling factor. Each Transformer's head learns an attention function that captures complex dependencies within the input data. With multiple heads, the model jointly attends to different subspaces of the input sequence. The outputs from each head are combined to obtain a joint representation:

$$\text{MultiHead}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{Concat}(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_H). \tag{2.12}$$

Merging all heads' outputs through a linear function produces a latent representation which encodes the most relevant information from the input features. In autoregressive tasks, such as next-token prediction, a causal mask is essential to prevent a token from accessing future information. The standard attention computation is modified by adding a mask $\mathbf{M}$ to the scaled dot-products, as shown in

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}, \tag{2.13}$$

where the mask is defined by

$$M_{ij} = \begin{cases} 0, & j \leqslant i, \\ -\infty, & j > i, \end{cases} \tag{2.14}$$

thereby ensuring that positions corresponding to future tokens obtain zero attention after the softmax. This mechanism is critical for maintaining the autoregressive property by ensuring that each token is computed solely on its preceding context, which preserves the sequential nature of the data. Multi-head attention layers are typically not stacked one after the other, but interleaved with normalization and residual layers to form a *Transformer block*, which is depicted in Figure 2.1. The reason why Attention layers cannot be stacked one after another is due to the issue of *rank-collapse*, introduced in [86], that bears great resemblance to the oversmoothing problem in Graph Neural Networks, which will be introduced in the next subsection. Note that while the effect of residual connections was explored in [85], the effect of normalization techniques and causal masking on rank collapse has been explored more recently in [271].

As mentioned, normalization methods are then applied as a way to stabilize training and prevent rank collapse. Layer Normalization is applied across the features of each individual sample and is defined as

$$\text{LN}(\mathbf{x}) = \gamma \frac{\mathbf{x} - \mu_{\text{layer}}}{\sqrt{\sigma^2_{\text{layer}} + \varepsilon}} + \beta, \tag{2.15}$$

where $\mu_{\text{layer}} = \frac{1}{d}\sum_{i=1}^{d} x_i$ is the mean computed across the features of a single sample, $\sigma^2_{\text{layer}} = \frac{1}{d}\sum_{i=1}^{d} (x_i - \mu_{\text{layer}})^2$ is the variance of the features, $\gamma$ and $\beta$ are learnable scaling

and shifting parameters, and $\varepsilon$ ensures numerical stability. Unlike Batch Normalization, which computes the normalization statistics over an entire mini-batch—thereby introducing dependencies between examples and becoming sensitive to batch size—Layer Normalization computes these statistics independently for each sample, making it invariant to batch size. This independence is especially advantageous for sequential models and transformer architectures and is one of the key reasons why Layer Normalization is preferred in large language models (LLMs), as it ensures stable and consistent training performance even with small or variable batch sizes and heterogeneous input characteristics. In the original Transformer architecture [260], the LN block was placed after the residual connection (also known as post-LN). However, in this original post-LN setup, large gradients at initialization force the use of a warm-up phase to prevent instability. As such, [273] proposed to reposition the LN inside the residual branch, (pre-LN), which led to well-behaved gradients that neither explode nor vanish even with a large initial learning rate, thus removing the necessity for a warm-up stage. An alternative and similarly performant normalization scheme is RMSNorm, which normalizes based on the root-mean-square value

$$\mathrm{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}}\, g, \tag{2.16}$$

which has a single gain term $g$, halving parameter count and reducing per-layer computation.

In addition to the self-attention mechanism and normalization techniques, Transformer models require positional encoding to incorporate sequence order information. Unlike recurrent or convolutional architectures, Transformers lack any inherent token-order mechanism; thus, positional encodings are added to the input embeddings to provide a notion of the token positions. A widely-adopted approach, introduced in [260], employs sinusoidal functions to generate these encodings:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \tag{2.17}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \tag{2.18}$$

which are added to the projected representations. This formulation enables the model to learn relative positions effectively, thereby allowing the Transformer to capture sequential relationships despite its non-recurrent nature. An elegant alternative to traditional additive sinusoidal encodings is Rotary Positional Embeddings (RoPE), which integrates positional information directly into the attention mechanism via a rotational transformation. Instead

of merely adding a fixed positional vector to the input embeddings, RoPE rotates the query and key vectors in their embedding space. For each pair of adjacent dimensions of a query vector, say $(q_{2i}, q_{2i+1})$, at token position $p$, RoPE applies a 2D rotation parameterized by an angle $\theta_{p,i}$, defined typically as $\theta_{p,i} = \frac{p}{10000^{2i/d}}$. This rotation is given by

$$\begin{pmatrix} \widetilde{q}_{2i} \\ \widetilde{q}_{2i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta_{p,i} & -\sin\theta_{p,i} \\ \sin\theta_{p,i} & \cos\theta_{p,i} \end{pmatrix} \begin{pmatrix} q_{2i} \\ q_{2i+1} \end{pmatrix}, \tag{2.19}$$

and an analogous transformation is applied to the key vectors. This multiplicative, rotation-based encoding ensures that the inner product between rotated query and key vectors inherently captures the relative positional differences between tokens.

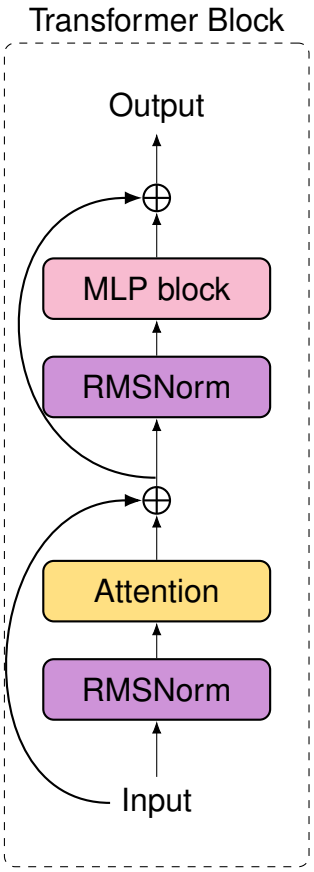


Figure 2.1: Transformer block structure.

Despite its strengths, self-attention runs in $\mathcal{O}(n^2d + nd^2)$ time and requires $\mathcal{O}(n^2 + nd)$ memory—forming the score matrix $\mathbf{QK}^T$ costs $\mathcal{O}(n^2d)$ and the input/output projections cost $\mathcal{O}(nd^2)$. Attention remains dominant because these operations map to huge, parallel batched matrix multiplies that GPUs handle with high arithmetic intensity, predictable memory access, and static tensor shapes that allow aggressive kernel fusion. This marriage

between algorithm and hardware (also termed the "hardware lottery" [144]) has led to the effective parameter scaling of attention-based models in practice [1].

## 2.3 Background on Graph Neural Networks

Many real-world systems—from molecular structures and protein–protein interaction networks in biology to citation graphs and social networks in information science—naturally manifest as graphs, where nodes represent entities and edges encode relationships. Learning predictive models on such graph-structured data is crucial for tasks like node classification, link prediction, and graph-level regression, yet traditional machine learning methods that assume independent and identically distributed (i.i.d.) samples cannot exploit the rich relational inductive biases inherent in these domains.

Graph Neural Networks (GNNs) [249, 117, 235, 36, 79] have emerged as a powerful framework for learning on graphs by combining spectral and spatial insights. In the spectral view, graph convolutions are defined via eigendecompositions of the Laplacian, enabling filters in the frequency domain [36, 79], while spatial methods directly aggregate information from each node's local neighborhood. Most contemporary architectures adopt a message-passing paradigm, wherein nodes iteratively exchange and transform "messages" with their neighbors—culminating in the Message-Passing Neural Network (MPNN) framework [111]. In this section, we will provide the necessary background on MPNNs and discuss related challenges.

### 2.3.1 Message Passing Neural Networks

Let a graph $\mathbf{G}$ be a tuple $(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the set of nodes and $\mathbf{E}$ is the set of edges. An edge from node $\mathbf{u} \in \mathbf{V}$ to node $\mathbf{v} \in \mathbf{V}$ is denoted by $(\mathbf{u}, \mathbf{v}) \in \mathbf{E}$. The graph's connectivity is captured by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $n$ is the number of nodes. We assume that $\mathbf{G}$ is undirected and that each node $\mathbf{v}$ is associated with a feature vector $\mathbf{h}_v \in \mathbb{R}^d$. Graph Neural Networks (GNNs) are functions

$$\mathbf{f}_{\boldsymbol{\theta}} : (\mathbf{G}, \{\mathbf{h}_v\}) \mapsto \mathbf{y}, \tag{2.20}$$

with parameters $\boldsymbol{\theta}$ learned via gradient descent, and $\mathbf{y}$ representing a prediction at the node or graph level. These models typically adopt the Message Passing Neural Network

(MPNN) framework, which computes latent representations by composing $K$ layers of the following node-wise operation:

$$\mathbf{h}_u^{(k)} = \phi^{(k)}\Big(\mathbf{h}_u^{(k-1)}, \psi^{(k)}\big(\{\mathbf{h}_v^{(k-1)} : (\mathbf{u}, \mathbf{v}) \in \mathbf{E}\}\big)\Big), \tag{2.21}$$

for $k = 1, \ldots, K$. Here, $\psi^{(k)}$ is a permutation invariant aggregation function, and $\phi^{(k)}$ combines the incoming messages from a node's neighbors with its previous embedding to produce an updated representation. A common aggregation function is given by:

$$\psi^{(k)}\Big(\{\mathbf{h}_v^{(k-1)} : (\mathbf{u}, \mathbf{v}) \in \mathbf{E}\}\Big) = \sum_{\mathbf{v}} \widetilde{\mathbf{A}}_{\mathbf{uv}}\, \mathbf{h}_v^{(k-1)}, \tag{2.22}$$

where $\widetilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\,\mathbf{D}^{-\frac{1}{2}}$ and $\mathbf{D} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. Alternatively, one can express the node features in a matrix form $\mathbf{H}^{(k)} \in \mathbb{R}^{n\times d_k}$. A widely used instance of GNNs is the Graph Convolutional Network (GCN) [162], whose update equation is

$$\mathbf{H}^{(k)} = \sigma(\widehat{\mathbf{A}}\,\mathbf{H}^{(k-1)}\,\mathbf{W}^{(k-1)}), \tag{2.23}$$

with

$$\widehat{\mathbf{A}} = (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}, \tag{2.24}$$

and $\sigma(\cdot)$ denoting a nonlinearity. Another popular instance of an MPNN is Graph Attention Networks (GATs) [261], where a learned adjacency matrix replaces the fixed normalized adjacency to dynamically modulate connectivity while preserving key spectral properties. In particular, the aggregation is computed as

$$\mathbf{h}_i^{(l+1)} \;=\; \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)}\,\mathbf{W}\,\mathbf{h}_j^{(l)}\Big), \tag{2.25}$$

where the attention coefficients are given by

$$\alpha_{ij}^{(l)} \;=\; \frac{\exp\big(\mathrm{LeakyReLU}\big(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i^{(l)} \,\|\, \mathbf{W}\mathbf{h}_j^{(l)}]\big)\big)}{\sum_{k\in\mathcal{N}(i)} \exp\big(\mathrm{LeakyReLU}\big(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i^{(l)} \,\|\, \mathbf{W}\mathbf{h}_k^{(l)}]\big)\big)}. \tag{2.26}$$

Multi-head attention is then introduced by computing $K$ independent sets of coefficients $\{\alpha_{ij}^{(l),k}\}$ (with corresponding weights $\mathbf{W}^{(k)}$, $\mathbf{a}^{(k)}$) and either concatenating or averaging their outputs:

$$\mathbf{h}_i^{(l+1)} = \Big\|_{k=1}^{K} \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l),k}\,\mathbf{W}^{(k)}\mathbf{h}_j^{(l)}\Big) \quad \text{or} \quad \frac{1}{K}\sum_{k=1}^{K} \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l),k}\,\mathbf{W}^{(k)}\mathbf{h}_j^{(l)}\Big). \tag{2.27}$$

Note that some fundamental limitations of the GAT model were identified and fixed in GATv2 [34]. Other notable MPNNs include GraphSAGE [129], Residual Gated Graph Convolutional Networks (Gated GCNs) [33], and Graph Isomorphism Networks (GIN) [274].

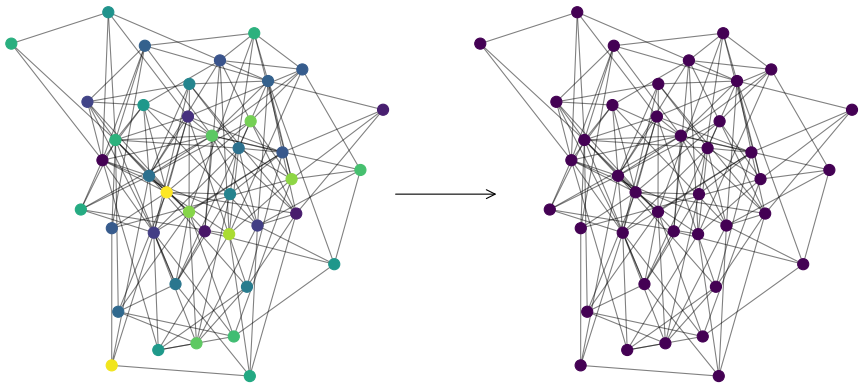

Figure 2.2: Illustration of the effect of oversmoothing on the node features of a graph.

### 2.3.2 Oversmoothing in Graph Neural Networks

Oversmoothing refers to the phenomenon whereby the repeated application of message-passing operations in Graph Neural Networks (GNNs) causes the node features to become increasingly similar. Some early works in this area include [206, 37]. A common way to quantify this effect is via the *unnormalized Dirichlet energy* of the node representations, which has been used in a number of works in the area [229, 228]. Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $n$ nodes and node features $\mathbf{H} \in \mathbb{R}^{n \times d}$, the unnormalized Dirichlet energy is defined as

$$\mathcal{E}(\mathbf{H}) = \sum_{(u,v) \in E} \|\mathbf{h}_u - \mathbf{h}_v\|^2, \tag{2.28}$$

where $\mathbf{h}_u$ and $\mathbf{h}_v$ denote the feature vectors for nodes $u$ and $v$, respectively. This energy measures the variability between features of adjacent nodes; a lower energy indicates smoother (i.e., more similar) node representations. In many GNN architectures, as more layers are stacked, the iterative aggregation tends to minimize $\mathcal{E}(\mathbf{H})$ to the extent that all node features converge toward a common value or rank-1 representation. While some degree of smoothing is beneficial, excessive oversmoothing leads to a loss of discriminative power, thereby hindering performance in downstream tasks. Several works have aimed to prevent oversmoothing in GNNs, from simpler approaches [223, 284] to physics-inspired approaches [28, 82].

### 2.3.3 Oversquashing in Graph Neural Networks

Oversquashing was originally introduced in [3], and it refers to the compression of information from distant nodes into a fixed-size representation during the message-passing process. This results in a vanishing sensitivity of the final node embeddings to changes in

remote inputs. This phenomenon was formally quantified in [257, 81] through the Jacobian. In particular, this was done by considering an MPNN whose node representations are updated layer-by-layer. Let $\mathbf{h}_v^{(K)}$ denote the representation of node $v$ after $K$ layers, and $\mathbf{h}_u^{(0)}$ the initial representation of a distant node $u$. Di Giovanni et al. [81] derived the following bound on the sensitivity:

**Theorem 2.3.1** (Sensitivity bounds, [81])**.** *Consider a standard MPNN with $k$ layers, where $c_\sigma$ is the Lipschitz constant of the activation $\sigma$, $w$ is the maximal entry-value over all weight matrices, and $d$ is the embedding dimension. For $u, v \in V$ we have*

$$\left\| \frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{h}_u^{(0)}} \right\| \leqslant \underbrace{(c_\sigma w d)^k}_{model} \underbrace{(\mathbf{O}^k)_{vu}}_{topology}, \tag{2.29}$$

*where $\mathbf{O} = c_r \mathbf{I} + c_a \mathbf{A} \in \mathbb{R}^{n \times n}$ is the message passing matrix adopted by the MPNN, and where $c_r$ and $c_a$ are the contributions of the self-connection and aggregation term.*

where $c_\sigma$ is the Lipschitz constant of the nonlinearity, $w$ the maximum entry value of the network weights, $d$ is the hidden dimensionality of the node features, and $\mathbf{O}$ is a matrix (often derived from the normalized adjacency matrix) that encapsulates the graph propagation dynamics, which reveals that both the model and the topology are responsible for oversquashing.

# Part I

# Learning from Sequential Data

# Chapter 3

# Deep Survival Analysis in Limit Order Books

## 3.1 Introduction

Most electronic financial exchanges use limit order books (LOBs) to organise and clear the demand and supply of liquidity in various asset classes. LOBs offer several types of orders, where *limit orders* and *market orders* are the most common types. Market orders cross the bid-ask spread and obtain immediate execution, while limit orders can be placed at various levels of the order book, and, if executed, obtain a better price than that of a market order. The price improvement of the limit order over the market order comes at a tradeoff. Market orders are immediately executed at the best prices available in the book, while limit orders rest in the LOB until they are filled by an incoming market order or they are withdrawn from the LOB; thus, there is no guarantee that a limit order will be executed. The length of time a limit order takes to get filled is known as *time-to-fill*, which can be estimated using different methods. In this work, we use *survival analysis* to calculate the fill probabilities of limit orders placed at different depths of the LOB.

We propose a deep learning method to estimate survival functions from longitudinal data. Our approach uses an encoder-decoder neural network architecture based on the Transformer model [260] and partially monotonic neural networks. The model uses the self-attention mechanism in the encoder to summarise the most recent events over a look-back window before a limit order is placed in the order book, and employs this latent representation of the LOB to estimate the probability of the order being filled after its submission. This self-attention mechanism employs a locally-aware representation of the time series as input, which is obtained through a convolutional network. The attention

mechanism and convolutional filters provide a more informative summary of the time series, which improves the estimate of the survival function conditioned on the most recent trades. To evaluate the performance of the estimated survival function we make use of *proper scoring rules* [113, 11, 222], which ensure we fit the true survival function and not to an incorrect proxy. This is in contrast with the common approach in the survival analysis literature where performance is typically evaluated using improper scoring rules such as time-dependant concordance, which can lead to an incorrect evaluation of model fit.

In this chapter, we focus on financial applications and study LOB data from Nasdaq exchange. We use our model to predict the survival functions of orders placed at different levels of the LOB, where matching of orders is determined by price-time priority. Our results show that the proposed architecture significantly outperforms baseline off-the-shelf architectures and standard benchmarks in the survival analysis literature. These results are consistent throughout a universe of assets with different characteristics, which speaks to the versatility of our model-free approach. We also carry out a significant financial analysis of the fill probability of limit orders for assets with different queue dynamics and order arrival rates (of both limit and market orders). Furthermore, we provide the first statistical evaluation of the fill probability of limit orders that are placed within the bid-ask spread. Finally, we use Shapley values [182] to perform an interpretability analysis and show that the model relies heavily on high-frequency information to perform the estimation, and gives less importance to slow-moving features which provide information about seasonal intraday patterns in the fill probability.

## 3.2 Literature Review

Time-to-event analysis, also known as survival analysis, is widely used in various fields, including the estimation of the time-to-recovery of a patient [170], clinical trials [244], churn prediction [169], and others [287, 250, 84]. A fundamental issue in most applications is how to relate the distribution of event times and the features (or *covariates*) describing a system. Simple parametric models, such as the *Cox proportional hazards model* [72] and the *accelerated failure time model* [268], are commonly used to make these connections. However, in recent years, several approaches integrate more complex deep learning models into survival analysis, particularly in the medical field. One early example of this is [101], who use a feed-forward neural network to extend the Cox proportional hazards model. Similar approaches are in [153] and [168], who incorporate techniques such as dropout, which are now common in deep learning. Additionally, some popular models output a discretised

survival function without imposing any assumptions on its shape or form [174, 172]. Subsequent work aims to improve upon these models [266, 286, 145], while [222] highlights the importance of using proper scoring rules in survival analysis and illustrates the shortcomings of previous approaches. Other notable works use Gaussian processes [104, 2], random forests [147], or adversarial approaches [58].

In quantitative finance, [63] assume that the shape of the survival function of filltimes follows a Weibull distribution. The authors track limit orders throughout the trading day, considering cancellations as right-censoring, to determine the fill probability. If a limit order is matched against multiple liquidity taking orders, the total time-to-fill is the weighted average of the individual time-to-fills, where the size of each execution is the weight. The authors treat partial fills as fills of orders that were initially sent with a reduced volume. [179] use the generalised gamma distribution to model the filltimes and use the accelerated failure time model to incorporate market features. The authors establish separate models for time-to-completion, time-to-first fill, and time-to-cancellation. They discuss hypothetical limit orders, first proposed by [131], to study limit-order execution times as the first-passage time to the limit price boundary. [49] and [126] derive optimal trading strategies with exponential fill rates that do not depend on time and depend on the distance between the price level in the limit order and the midprice in the book. [185] use recurrent neural networks to predict the fill probability of limit orders, which is a widespread approach in the survival analysis literature. To train their model, they use hypothetical limit orders that are placed in the book and kept at a fixed price throughout the trading day, even if the price moves unfavourably over time. Their work benchmarks its results with the AUC-ROC score, which is an improper scoring rule in survival analysis; hence the assessment of model fit may not be correct. Within optimal execution and market making, we highlight various works which tackle these problems in both traditional and decentralized exchanges [23, 88, 46, 42, 87, 44, 43, 47, 45].

Our work provides a different perspective. We introduce a Transformer-based architecture that outperforms previous benchmarks in terms of proper scoring rules. Furthermore, we evaluate and present several approaches to generate training data, including repositioning hypothetical limit orders and predicting the fill probability directly from the limit orders observed in the LOB. The remainder of the chapter is organised as follows. Section 3.3 introduces the survival analysis and discusses proper scoring rules. Next, Section 3.4 presents a statistical analysis of the fill probabilities. Section 3.5 presents our neural network model, and Section 3.6 presents our results and an interpretability analysis.

## 3.3 Survival Analysis

### 3.3.1 Preliminaries

The *event time* $T_l \in \mathbb{R}_{>0}$ is a positive valued random variable that describes the time-to-fill of a limit order placed at level $l$ of the order book. Our objective is to predict event times by conditioning on a set of market features $\mathbf{x} \in \mathbb{R}^p$. All events are subject to right-censoring, because the final event may not be observed. When a limit order is cancelled or reaches the end of the trading day without being filled, it is considered a censored event. We consider a set of $N$ observations of the triplet $(\mathbf{x}_i, z_i, d_i)$, where $d_i = \mathbb{1}\{z_i = t_i\}$ is an indicator function that is equal to zero if the event is censored, and $z_i$ and $\mathbf{x}_i$ are the observed event time and the observed market features up to the instant of order submission, respectively. We use the triplet to estimate the *survival function* $S_{T_l}(t \mid \mathbf{x}) = \mathbb{P}\{T_l > t \mid \mathbf{x}\}$.[1]

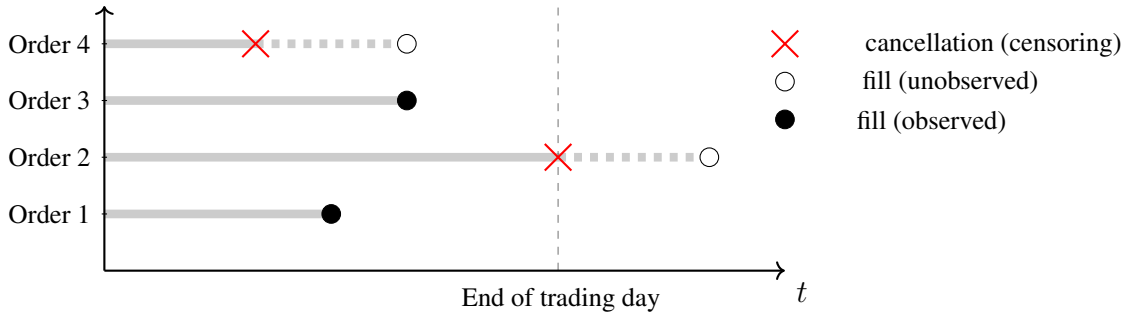


Figure 3.1: Events that can occur after the submission of a limit order.

The survival function returns the probability that a limit order posted at level $l$ will not be filled before time $t$. The link between the survival function $S_{T_l}(t|\mathbf{x})$ (which shows the probability of the order not being filled before a particular time) and the cumulative density function $F_{T_l}(t|\mathbf{x})$ of the event time is given by $S_{T_l}(t|\mathbf{x}) = 1 - F_{T_l}(t|\mathbf{x})$ (hence being the opposite of the survival function). Thus, the density function is $f_{T_l}(t|\mathbf{x}) = -\frac{d}{dt}S_{T_l}(t|\mathbf{x})$, which describes the probability of an order being filled in a particular time after submission. Further, the *hazard rate*, which indicates the propensity that an order will be filled after time $t$, is given by

$$h_{T_l}(t|\mathbf{x}) = \frac{f_{T_l}(t|\mathbf{x})}{1 - F_{T_l}(t|\mathbf{x})} \,. \tag{3.1}$$

[1] This is an instance of *survival regression*, which is equivalent to survival analysis but conditioning on a set of features $\mathbf{x}$.

It suffices to obtain one of the previous functions to derive the remaining three, because

$$S_{T_l}(t|\mathbf{x}) = \mathbb{P}\{T_l > t|\mathbf{x}\} = 1 - F_{T_l}(t|\mathbf{x}) = \exp\Big( - \int_0^t h_{T_l}(s|\mathbf{x})\, ds \Big) . \tag{3.2}$$

The shape of survival functions is described by a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}$. One may assume that the survival function is given by a tractable distribution (*e.g.*, a Weibull distribution) and estimate the relevant parameters with standard methods. However, there is a trade-off between mathematical tractability and goodness of fit to the data. An alternative is to use neural networks to estimate the survival function, which increases the number of parameters substantially but improves fit to the data. Here, we use the latter approach to model the survival function of limit orders because its shape is expected to have a non-linear relationship with market features. We perform *maximum likelihood estimation* to obtain the vector of parameters that best fit to the data. Specifically, we train our deep learning model to maximize the *right censored log-likelihood function*

$$\mathcal{L}(\boldsymbol{\theta}) = \log(L_N(\boldsymbol{\theta})) = \sum_{k=1}^{N} d_k \log(\widehat{f}(z_k|\mathbf{x}_k, \boldsymbol{\theta})) + (1 - d_k) \log(\widehat{S}(z_k|\mathbf{x}_k, \boldsymbol{\theta})) , \tag{3.3}$$

where $\widehat{S}$ and $\widehat{f}$ are the neural network estimates of the survival function and the density function, respectively. See 3.8.2 for a derivation of (3.3). Training on right-censored log-likelihood requires a model to output $\widehat{S}(z_k|\mathbf{x}_k, \boldsymbol{\theta})$ at the exact time-instant $z_k$. This is challenging in deep learning models that use the softmax activation function to discretise the survival function, because these models require an interpolation scheme to tractably train on (3.3). Our model avoids this problem by inputing the observed time $z_k$ to only the monotonically restricted decoder, together with the generated latent representation from the LOB time series.

To evaluate the quality of model fit to the survival function, we use the concept of *scoring rule*, see [222]. A scoring rule $\mathcal{S}$ takes as input a distribution $S$ over a set $\mathcal{Y}$, with an observed sample $y \in \mathcal{Y}$, and returns a score $\mathcal{S}(S, y)$. With positive scoring rules, higher scores indicate an improvement in model fit. In survival regression, a scoring rule $\mathcal{S}$ is *proper* if

$$\mathbb{E}_{t,c,\mathbf{x}}[\mathcal{S}(S(t|\mathbf{x}), (z, d))] > \mathbb{E}_{t,c,\mathbf{x}}[\mathcal{S}(\widehat{S}(t|\mathbf{x}), (z, d))] \tag{3.4}$$

for all survival function estimates $\widehat{S}(t|x)$. The most commonly used scoring rules in the literature are improper, as outlined in the next subsection. On the other hand, [222] show that right-censored log-likelihood (RCLL) is proper scoring rule. Throughout our analysis,

we use RCLL as a proper scoring rule to evaluate the precision with which we fit the true survival function, and therefore evaluate the performance of the proposed model. This guarantees that we evaluate model estimates to guarantee fit to the true underlying survival function and not an incorrect proxy.

### 3.3.2 Benchmark survival functions estimates

We now present a series of methods that approximate survival functions, which will be used as comparative benchmarks in Section 3.6. An initial approach is to use a Kaplan–Meier estimate [151] to estimate the survival function i.e.,

$$\widehat{S}(t) = \prod_{i:t_i<t} \left(1 - \frac{k_i}{n_i}\right), \tag{3.5}$$

where $t_i$ is the time when at least one event occurred, $k_i$ is the number of events that occurred at time $t_i$, and $n_i$ denotes the limit orders known to have survived up to time $t_i$. A follow-up model to the Kaplan–Meier estimate which conditions on a feature vector is the Cox proportional hazards model [72], where the hazard rate follows the definition

$$h(t|\mathbf{x}) = h_0(t)\text{exp}(\boldsymbol{\beta}^T\mathbf{x}), \tag{3.6}$$

where $\boldsymbol{\beta}$ are coefficients for the feature vector $\mathbf{x}$, and $h_0(t)$ is a baseline hazard directly estimated from the data. Given $N$ observations, the regression coefficients which maximize

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{\delta_i=1} \frac{\text{exp}(\boldsymbol{\beta}^T\mathbf{x})}{\sum_{j:t_j>t_i}\text{exp}(\boldsymbol{\beta}^T\mathbf{x})}. \tag{3.7}$$

Another popular model used in survival analysis is the accelerated failure time model [268], in which the hazard rates are determined by

$$h(t|\mathbf{x}) = \phi(\boldsymbol{x})h_0(\phi(\boldsymbol{x})t), \tag{3.8}$$

where $\phi(\boldsymbol{x})$ models the effect of the covariates, usually through the relationship $\phi(\boldsymbol{x}) = \text{exp}(\boldsymbol{\beta}^T\mathbf{x})$. In practice, the assumption of linear interaction between features and the proportional hazards assumption are often violated. This motivated the extension of the Cox model with deep learning to capture non-linearities between features of interest [153, 168]

$$h(t|\mathbf{x}) = h_0(t)\text{exp}(f_{\boldsymbol{\theta}}(\mathbf{x}, t)). \tag{3.9}$$

More recent work [174, 172, 222] focuses on directly learning the survival function conditioning on input features

$$S(t|\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x}, t). \tag{3.10}$$

### 3.3.3 Scoring rules

To evaluate the quality of model fit to the survival function, we use the concept of *scoring rule*, see [222]. A scoring rule $\mathcal{S}$ takes as input a distribution $S$ over a set $\mathcal{Y}$, with an observed sample $y \in \mathcal{Y}$, and returns a score $\mathcal{S}(S, y)$. With positive scoring rules, higher scores indicate an improvement in model fit. In survival regression, a scoring rule $\mathcal{S}$ is *proper* if

$$\mathbb{E}_{t,c,\mathbf{x}}[\mathcal{S}(S(t|\mathbf{x}), (z, \delta))] > \mathbb{E}_{t,c,\mathbf{x}}[\mathcal{S}(\widehat{S}(t|\mathbf{x}), (z, \delta))] \tag{3.11}$$

for all survival function estimates $\widehat{S}(t|x)$. This means that in expectation, a proper scoring rule will give higher scores to the true survival function over any other estimate. Commonly used scores for survival functions include time-dependant concordance [4], given by

$$C_{td} = \mathbb{P}[\widehat{S}(z_i|\mathbf{x}_i) < \widehat{S}(z_i|\mathbf{x}_j)|z_i < z_j, \delta_i = 1] \tag{3.12}$$

$$\approx \frac{\sum_{i=1}^{N}\sum_{j=1;i\neq j}^{N} \mathbb{1}[\widehat{S}(z_i|\mathbf{x}_i) < \widehat{S}(z_j|\mathbf{x}_j)]\pi_{ij}}{\sum_{i=1}^{N}\sum_{j=1;i\neq j}^{N}\pi_{ij}}, \tag{3.13}$$

where $\pi_{ij}$ is an indicator of the pair $(i, j)$ being amenable for comparison, i.e., if the pair is "concordant". The intuition behind time-varying concordance [132] is based on the idea that the predicted survival probability for an order $i$ evaluated at time $z_i$ and conditioned on market features $\mathbf{x}_i$ should be lower than that of order $j$ evaluated at the same time and conditioned on market features $\mathbf{x}_j$ if order $i$ was filled faster than order $j$. In addition to time-varying concordance, another frequently used scoring is the Brier score for right-censored data [118], defined as

$$\beta = \frac{\widehat{S}(t|x)^2\mathbb{1}\{z < t, d = 1\}}{\widehat{G}(z)} + \frac{(1 - \widehat{S}(t|x))^2\mathbb{1}\{z > t\}}{\widehat{G}(z)}, \tag{3.14}$$

where $\widehat{G}$ is the Kaplan–Meier estimate of the censoring distribution.

Time-varying concordance and Brier score are the two most common scores to evaluate models in the survival analysis literature [174, 172, 286]. However, recent work [222] shows that both scoring rules (as well as a number of others) are improper, meaning that they can potentially give higher scores to wrongly fitted distributions.[2] Right-censored log-likelihood is a proper scoring rule, which is the key result in [222].

[2]Brier score is a proper scoring rule under the assumption of independence between censoring and covariates, as well as a perfect estimate of the censoring distribution. These assumptions do not hold in the context of limit order executions, given that orders of larger size are prone to cancellations (which are interpreted as censoring in this work) and there is not a tractable way of obtaining a perfect estimate of the distribution of cancelled orders.

## 3.4 Empirical and Statistical Evidence of Fill Rate Executions

Filltime data for limit orders must be calculated from message data in the LOB. Therefore, in this section we present two different ways to compute this data. The first involves tracking the outcome of limit orders places in the LOB, and the second is through hypothetical orders to allow model limit order repegging. We compare the resulting survival functions associated to different levels of the order book. Furthermore, we analyze the changes in fill probability when orders are placed inside the spread for assets with different queue behaviour and order arrival speed.

### 3.4.1 Generation of training data

One way to obtain the dataset of triplets $\{(\mathbf{x}_i, z_i, d_i)\}_{i=1}^{N}$ detailed in the previous section is to track all the messages associated to a particular order after its submission. If the final message corresponds to an execution, we treat the order as filled and censored otherwise. The time-to-fill is the time between order submission and the time the final message is observed.

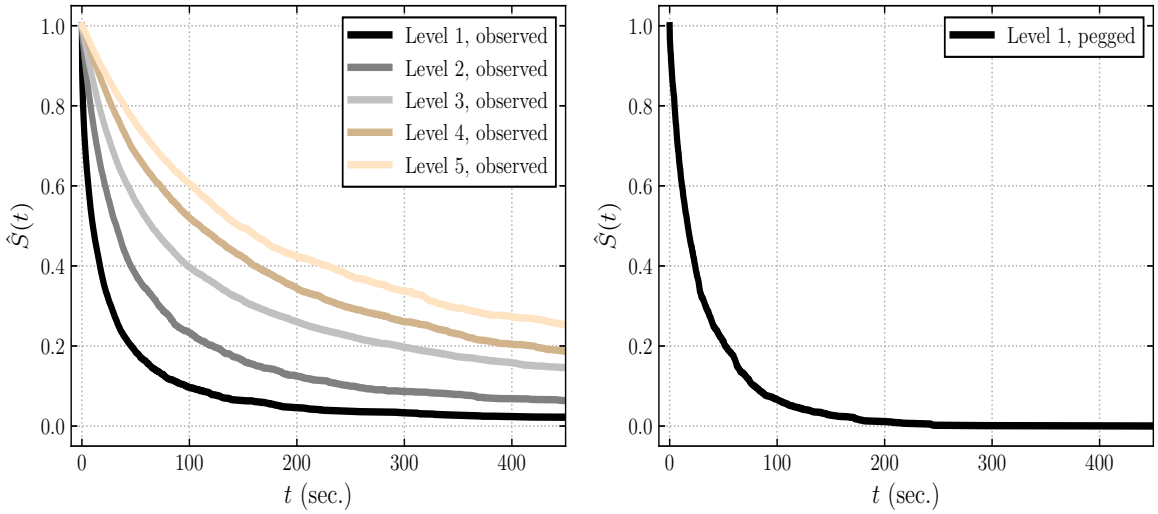


Figure 3.2: **Left:** Kaplan–Meier estimates of orders placed at different levels of the AAPL LOB. **Right:** Kaplan–Meier estimate of hypothetical order pegged to the first level in the LOB of AAPL.

Another way is to use "hypothetical" limit orders, which are orders of one share in volume placed at the top of the queue of a level in the order book. In particular, we consider the changes to the survival function when the order follows the price of a particular level. We refer to this as *pegging* the limit order to that price level. Such an approach probes a

set of "fill conditions", which are detailed in 3.8.1, every time the order book is updated to determine if an order has been filled. We assume that hypothetical limit orders do to not have market impact, which is consistent with the work of [130] and [185]. With hypothetical limit orders we model different behavior to what is observed in the order book data *ex post*.

A significant number of orders resting outside the best level in the LOB result in censored data because very few aggressive orders walk the LOB beyond the best quotes. Therefore, we focus on estimating the fill probability of orders placed or pegged to the best bid and best ask of the LOB. In both cases, we select the orders to track randomly throughout the trading day. The survival functions are visualized through Kaplan-Meier estimates (see 3.8.2 for more details on their estimation) in Figure 3.2. This visualization provides an averaged view of the survival function and cannot capture the effect of microstructural features on the the fill probability of the orders; this motivates our deep learning approach, which we present in the next section.

### 3.4.2 Fill statisics of limit orders placed inside the bid-ask spread

Here, we compute the fill probabilities of orders placed within the spread of the order book. We consider nine stocks, some of which are of small or large tick and that also vary in their average trading activity. Table 3.1 shows the average spread, average volume on the best bid and best ask, and average trades per minute over the month of October 2022.[3]

Intuitively, agents trading large tick stocks are incentivised to place orders inside the bid-ask spread (whenever this is possible) not to place their LOs at the end of the queue. The cost of executing this trade is almost as much as crossing the spread, and there is no guarantee of immediate execution. However, it is approximately 3 to 6 times more likely that an order is filled when it improves the best quotes and reduces the time to fill by a similar proportion, see Figure 3.3 and Table 3.2. This is expected because traders who observe orders that improve the best quotes will aim to quickly match them with a liquidity taking order before they are cancelled. Figure 3.3 also shows that the fill probability of orders placed in the spread of large tick stocks, when this is possible, rises sharply for small time

[3]Large tick stocks are such that the bid-ask spread is on average close to one tick in size, while small tick stocks the spread for small tick ticks is several times larger than that. There is a significant difference in the dynamics between the two types, see [95] for more details. This is a consequence of the ratio between the price and the tick size of the exchange. When this is low, traders tend to be more reluctant to cross the bid-ask spread which results in the formation of large queues at the best bid and best ask. In our case, we consider a large tick stock to have a average spread of less that 1.3 ticks, as detailed in [25].

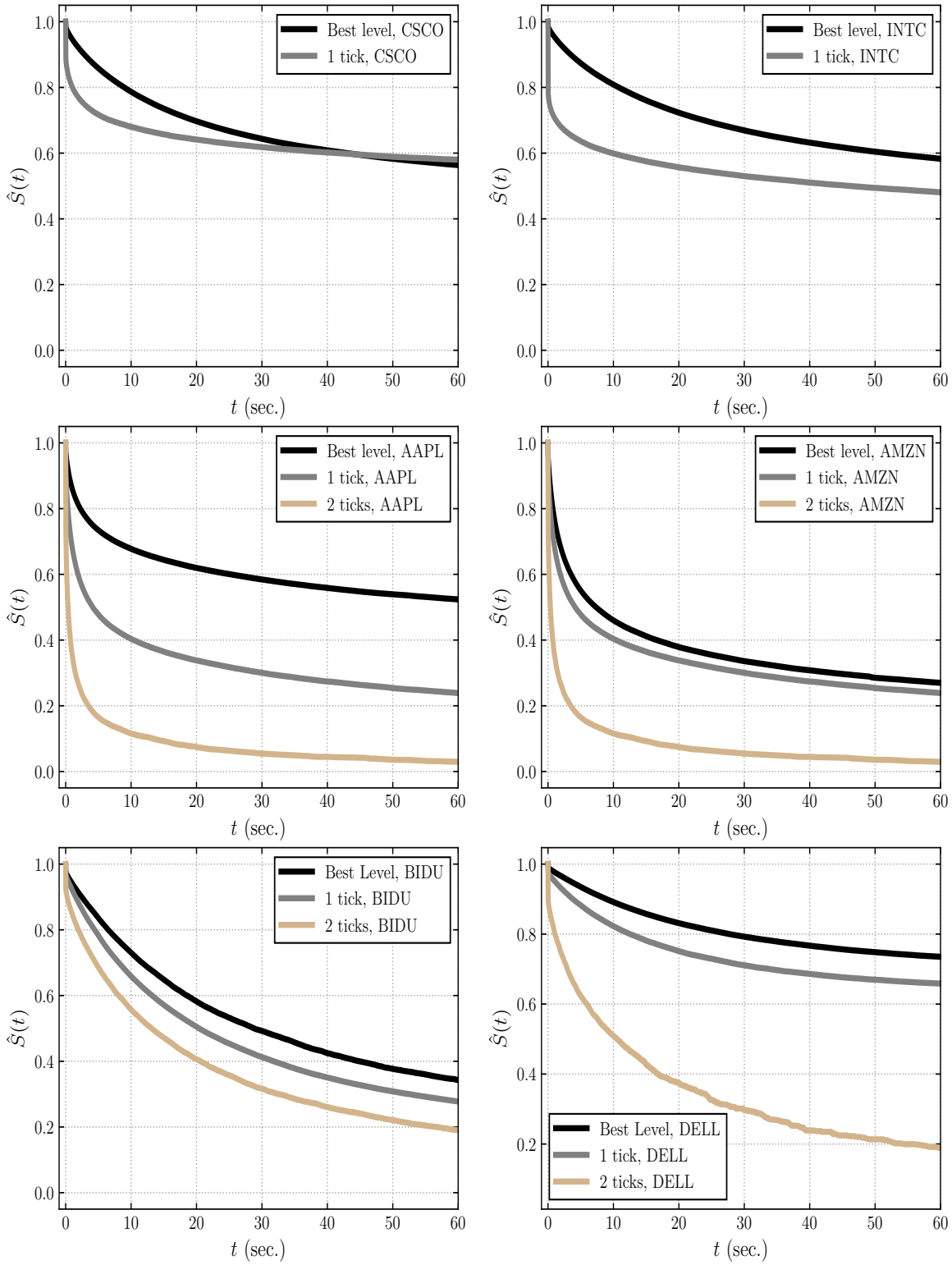


Figure 3.3: Survival functions when placing orders at different depths of the bid-ask spread. **Top left:** CSCO, **Top left:** INTC, **Middle left:** AAPL, **Middle right:** AMZN, **Bottom left:** BIDU, **Bottom right:** DELL.

Table 3.1: Statistics of small and large tick stocks on October 2022.

| | Avg. spread (ticks) | Avg. volume best ask | Avg. volume best bid | Avg. midprice | Avg. executed trades/min. |
|---|---|---|---|---|---|
| **AAPL** | 1.44 | 487.19 | 442.03 | 144.34 | 405.02 |
| **AMZN** | 1.91 | 298.41 | 307.72 | 114.80 | 319.44 |
| **BIDU** | 8.75 | 138.98 | 97.24 | 101.89 | 20.86 |
| **COST** | 30.89 | 67.80 | 60.89 | 478.21 | 44.27 |
| **DELL** | 1.77 | 287.02 | 283.92 | 35.97 | 14.48 |
| **GOOG** | 1.66 | 299.26 | 232.08 | 99.59 | 120.84 |
| **MSFT** | 2.88 | 159.17 | 159.14 | 236.89 | 204.86 |
| **CSCO** | 1.15 | 2212.01 | 2193.81 | 41.90 | 74.87 |
| **INTC** | 1.13 | 5995.54 | 5533.79 | 26.55 | 110.88 |

Table 3.2: Fill statistics at the best level and at different depths in the spread for the entire universe of assets considered between 1 October 2022 and 27 December 2022.

| | **Fill probability** | | | | | | **Avg. Filltime (s.)** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Depth | **Best** | **1** | **2** | **3** | **4** | **5** | **Best** | **1** | **2** | **3** | **4** | **5** |
| **AAPL** | 0.0539 | 0.1317 | 0.3601 | 0.4165 | 0.4382 | 0.4431 | 1.32 | 0.64 | 0.19 | 0.11 | 0.14 | 0.08 |
| **AMZN** | 0.0923 | 0.1312 | 0.3360 | 0.4139 | 0.4343 | 0.4485 | 1.07 | 0.70 | 0.38 | 0.25 | 0.11 | 0.21 |
| **BIDU** | 0.0948 | 0.0528 | 0.1533 | 0.1692 | 0.1868 | 0.2140 | 4.40 | 3.82 | 3.93 | 3.59 | 3.26 | 3.07 |
| **COST** | 0.0458 | 0.0122 | 0.0826 | 0.0898 | 0.1061 | 0.1135 | 5.18 | 4.85 | 6.03 | 5.33 | 4.69 | 4.53 |
| **CSCO** | 0.0573 | 0.1753 | 0.3549 | 0.3579 | 0.2484 | 0.4558 | 6.26 | 1.71 | 0.47 | 0.51 | 0.25 | 0.36 |
| **DELL** | 0.0392 | 0.0620 | 0.2263 | 0.2762 | 0.2902 | 0.2995 | 6.87 | 4.46 | 3.39 | 2.07 | 2.34 | 1.5 |
| **GOOG** | 0.0618 | 0.1088 | 0.3199 | 0.3969 | 0.4450 | 0.5012 | 1.82 | 0.85 | 0.84 | 0.13 | 0.15 | 0.07 |
| **INTC** | 0.0570 | 0.2962 | 0.3215 | 0.2463 | 0.2474 | 0.1165 | 7.33 | 1.95 | 0.27 | 0.29 | 0.8 | 0.71 |
| **MSFT** | 0.0679 | 0.0734 | 0.2632 | 0.3690 | 0.4292 | 0.4466 | 0.99 | 0.70 | 0.45 | 0.24 | 0.17 | 0.12 |

horizons before eventually decreasing to levels comparable to those of orders resting at the best level in the asset's limit order book. This suggests that if an order is executed within a few seconds of being placed into the book, the price moved in the favorable direction and there is a fill because the order is at the beginning of the queue. If this is not the case, it is likely that the price moved adversely, making the order rest deeper in its side of the book. Further, the survival function of stocks with larger trading activity exhibit a quicker decay than those of less active stocks.

The activity inside the spread of small tick stocks is several orders of magnitude bigger than that of large tick stocks, see Table 3.3. This also shows how improbable it is for spreads to widen in large tick stocks. Obtaining improvements in the fill probabilities for small tick assets requires placing limit orders deeper within the spread, where trading activity is comparable to that of LOs placed one level deep in large tick stocks.

Table 3.3: Trading activity inside the spread for the entire universe of assets considered between 1 October 2022 and 27 December 2022.

| | 1 Tick | 2 Tick | 3 Tick | 4 Tick | 5 Tick |
|---|---|---|---|---|---|
| **AAPL** | 51,096 | 17,862 | 11,692 | 4,059 | 2,004 |
| **AMZN** | 6,215,869 | 169,024 | 24,456 | 6,469 | 2,470 |
| **BIDU** | 1,250,831 | 163,567 | 100,719 | 63,407 | 41,017 |
| **COST** | 2,594,091 | 113,709 | 87,820 | 69,689 | 61,457 |
| **CSCO** | 466,645 | 2,817 | 841 | 318 | 136 |
| **DELL** | 753,603 | 15,584 | 2,548 | 937 | 454 |
| **GOOG** | 1,548,080 | 52,565 | 8,344 | 2,274 | 782 |
| **INTC** | 317,834 | 992 | 276 | 97 | 103 |
| **MSFT** | 9,654,578 | 492,218 | 107,990 | 33,885 | 13,710 |

## 3.5 Monotonic Encoder-Decoder Convolutional-Transformer

### 3.5.1 General Architecture

In this section, we present our encoder-decoder architecture which learns the mapping between states of the LOB and distribution of limit order survival times. Figure 3.4 illustrates the two components of our framework. The encoder, parameterised by $\mathbf{\Phi} \in \mathbb{R}^{m_\Phi}$, processes the LOB data and obtains a latent representation from it, which is used by the decoder, parameterised by $\mathbf{\Psi} \in \mathbb{R}^{m_\Psi +}$, to predict the survival function of the limit orders. The decoder comprises a monotonic neural network that guarantees a monotonically decreasing survival function. Further, we use a convolutional-Transformer encoder to model the complex dependencies and interactions within the LOB data and to compress useful information into a lower-dimensional representation, subsequently used by the monotonic-decoder.

### 3.5.2 Convolutional-Transformer Encoder

We propose a convolutional-Transformer encoder to identify patterns in the LOB and to obtain accurate estimates of the fill probabilities of limit orders. The architecture of the encoder, see Figure 3.5, processes the LOB time series data and captures its non-Markovian dynamics through a latent representation of the time series. This representation encapsulates the most relevant information which is used by the decoder to predict the fill probabilities.

The encoder consists of two components: a locally-aware convolutional network and a Transformer model. The locally-aware convolutional network consists of three different

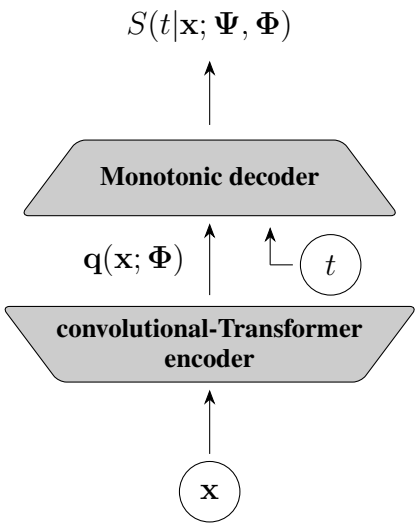


Figure 3.4: Encoder-decoder architecture to estimate the survival function. The first block, an encoder with parameter $\mathbf{\Phi} \in \mathbb{R}^{m_\Phi}$, uses an attention-based mechanism to project the LOB observations to a latent representation that captures relevant information. The second block, a monotonic decoder takes as input both this latent representation of the time series and the time variable $t$. The weights of the decoder are positive to enforce a monotonically decreasing survival function.

Dilated Causal Convolutional (DCC) neural networks [207] that process the LOB data and generate the corresponding queries, keys, and values which serve as inputs to the Transformer model. These DCCs, which are based on Convolutional Neural Networks (CNNs) [171, 282, 281], use inner product operations based on entries that are a fixed number of steps apart from each other, contrary to CNNs and Causal-CNNs, which operate with consecutive entries, as shown in (3.15). Further, causal convolutions ensure that the current position does not use future information. Previously, DCCs have been successfully applied in time series forecasting [30, 198].

The queries, keys, and values created by the DCCs are three different representations resulting from the convolution operation on the LOB input features and are collectively used by the Transformer model to perform self-attention and capture dependencies between different parts of the original time series. Using convolutional networks to generate the input features to the Transformer allows our encoder to be more aware of local context, granting the Transformer the ability to discern if observed values are anomalies, part of patterns, etc. This constitutes a clear advantage over using a multi-layer perceptron (MLP) to obtain the queries, keys, and values, which is the most common approach in the literature. Therefore, the operation of the DCCs can be understood as a set of data-driven local filters. The pro-

jection they perform from the original LOB time series to a hidden representation enhances the depiction of the LOB dynamics. This operation enables the Transformer's self-attention mechanism to capture complex local dependencies between datapoints, because it operates on a locally aware hidden representation, rather than point-wise values that lack local context. Additionally, the convolutional-Transformer optimises the parameters of each of the three DCCs to extract different relevant features from the LOB data. For example, some filters may be optimised to detect trends, while others may identify anomalies or change-points. Each of the three convolutional neural networks used to obtain the corresponding query, key, and values, that serve as input to the Transformer model [260], consists of only one layer performing a causal convolutional operation between the input sequence, $\mathbf{x} \in \mathbb{R}$, and the corresponding convolutional kernel $\mathbf{k}$ of size $s \in \mathbb{Z}$:

$$\begin{cases} Q(t) & = \left(\mathbf{x} *_p \mathbf{k}^Q\right)(t) = \sum_{\tau=0}^{s-1} k_\tau^Q \cdot x_{t-p\cdot\tau}\,, \\ K(t) & = \left(\mathbf{x} *_p \mathbf{k}^K\right)(t) = \sum_{\tau=0}^{s-1} k_\tau^K \cdot x_{t-p\cdot\tau}\,, \\ V(t) & = \left(\mathbf{x} *_p \mathbf{k}^V\right)(t) = \sum_{\tau=0}^{s-1} k_\tau^V \cdot x_{t-p\cdot\tau}\,, \end{cases} \tag{3.15}$$

where $p$ is the dilation factor.[4] We use the convolutional network solely as a feature extractor that incorporates local context into the Transformer's self-attention mechanism because, in our model, the Transformer model is only responsible for extracting the patterns within the data. Therefore, we restrict the encoder's convolutional network to a single layer, but its complexity can be easily extended to $L$ convolutional layers, see 3.8.3.

After the CNNs extract the relevant features from the LOB data and produce the queries, keys, and values, these are fed to the Transformer model. Transformer models were initially introduced for Natural Language Processing (NLP), but they have been widely applied in time series-related problems [197]. These models propose a new architecture that leverages the attention mechanism [13] to process sequences of data. They have significant advantages over more classical approaches because of their ability to maintain lookback windows with large horizons, which makes them able to detect long-term dependencies in the data. On the other hand, canonical Transformers' point-wise dot-product attention makes them prone to anomalies and optimisation issues because of their space complexity, which grows quadratically with the input length. Furthermore, canonical Transformers are locally-agnostic, because dot-product attention does not allow the model to be aware of the

[4] Note that $p = 1$ results in a Causal-CNN.

local context while operating with time series data. The convolutional-Transformer alleviates these problems: the integration of convolutional networks makes the model locally-aware and a sparse self-attention mechanism mitigates its space complexity, reducing the cost of computing the attention scores from $\mathcal{O}(L^2)$ to $\mathcal{O}(L(\log(L))^2)$, where $L$ is the input length.

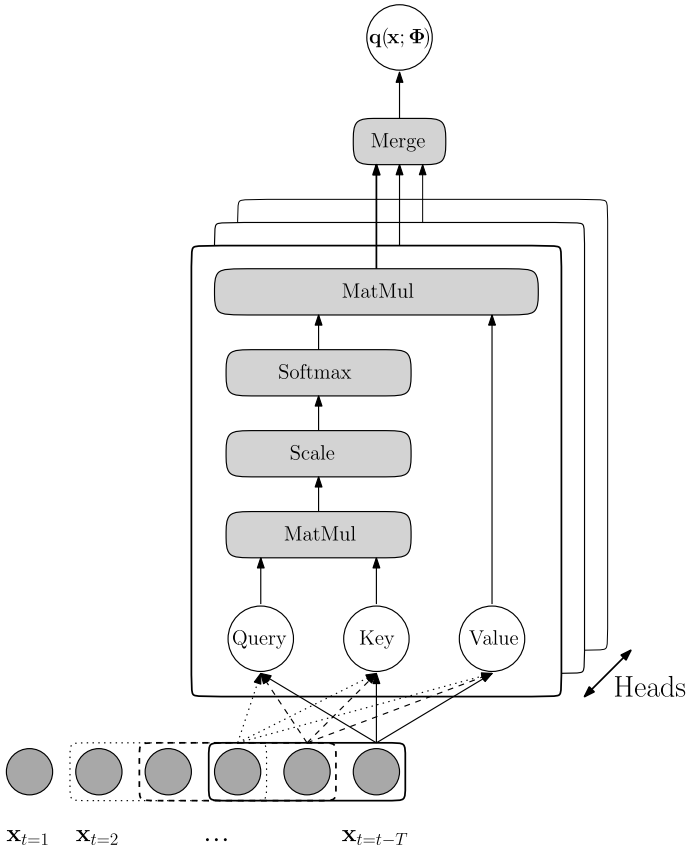


Figure 3.5: Structure of multi-head convolutional Transformer.

### 3.5.3 Monotonic Decoder

In the context of survival analysis, the survival function needs to be decreasing with respect to time. We encode this inductive bias into our architecture to avoid the well-known *crossing problem* [252]. To this end, we use monotonically restricted neural networks [62, 222] in our monotonic decoder, see Figure 3.6. This type of neural network allows us to estimate a cumulative density function (CDF) denoted by $F(t|\mathbf{x})$ with response variable $t$ and conditioned on input features $\mathbf{x}$, which in our case is the latent representation obtained from the LOB time series through the encoder. The output of the decoder's network, $f_{\Psi}(\,\cdot\,)$, is

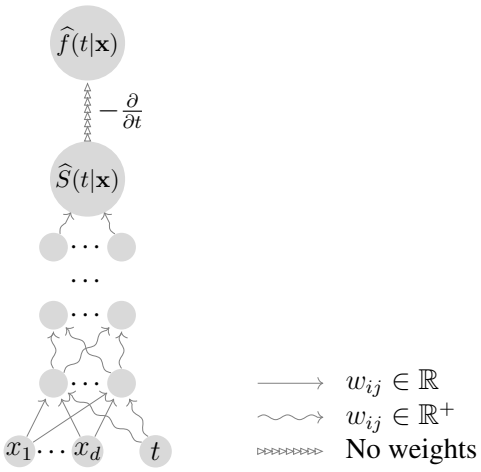


Figure 3.6: Monotonic decoder's structure. The last node represents the operation of differentiating the conditional survival distribution with respect to time, which results in the conditional density function of the survival time. This model guarantees a decreasing survival curve. (Adapted from Figure 1 of [62].)

consistent with the properties of a CDF because it satisfies: (i) $\lim_{t\to-\infty} f_{\mathbf{\Psi}}(t, \mathbf{x}) = 0$ , (ii) $\lim_{t\to\infty} f_{\mathbf{\Psi}}(t, \mathbf{x}) = 1$ , (iii) $\frac{\partial f_{\mathbf{\Psi}}(t,\mathbf{x})}{\partial t} > 0$ ,

where the third condition is the most difficult to guarantee because neural networks are a composition of nonlinear functions, which makes them difficult to interpret or control. See 3.8.4 for more details.

We remark that imposing this monotonicity on the decoder does not hinder other beneficial properties of deep neural networks such as universal function approximation [76, 160] or convexity [178] in the over-parameterised case. The reason behind this is that the restriction is only enforced on the decoder, which can be made arbitrarily small (in terms of parameters) compared to the time series encoder, where the network parameters are not restricted.

## 3.6 Experiments

### 3.6.1 Predictive Features

To estimate the survival function we distinguish between *slow-moving* and *fast-moving* features. Slow moving features provide information about intraday patterns of the trading day, which have a predictable influence on the probability that a limit order will be filled. One such pattern is the intraday behaviour of the volatility, which is generally high at the beginning of the trading day due to uncertainty and an adjustment to overnight information. This

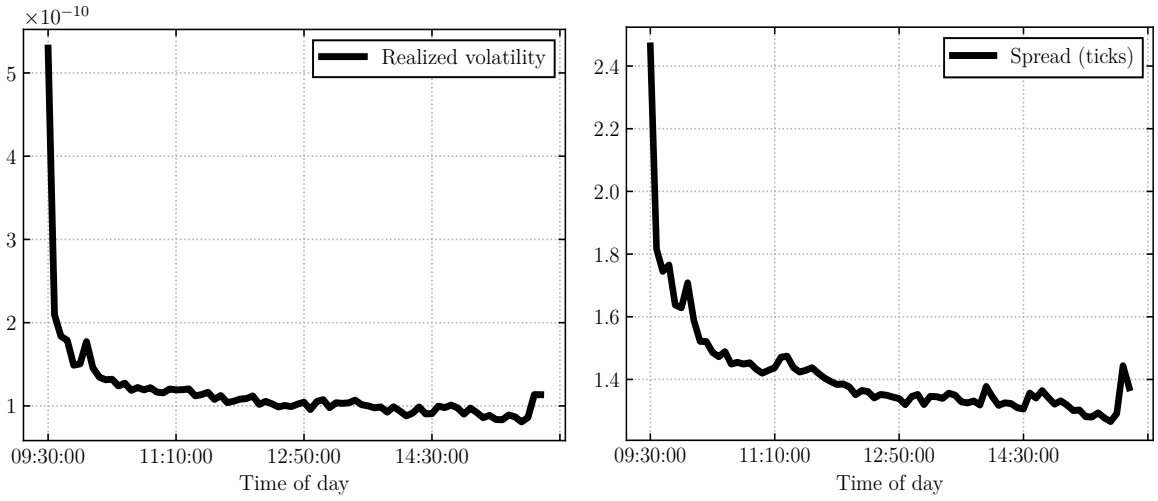


Figure 3.7: Realised volatility and spread of AAPL stock over October 2022. **Left:** Realized volatility. **Right:** Spread.

shown in Figure 3.7, along with the intraday behaviour of the AAPL stock over the month of October 2022.[5]

A similar effect is visible with the daily traded volume. Overnight information generally cause larger trading volume, which tends to stabilise during the trading day at a reduced value, and peaks again at the end of the day when traders have more urgency to close their positions. This effect is shown in Figure 3.8, along with the variation of the fill probability during the trading day. Given the persistence of these intraday seasonalities, a simple linear or non-linear model should be able to make a good prediction given these variables.[6]

Aside from seasonal patterns, we want to predict the changes in the fill probability over more granular time-scales by using *fast-moving features*. In particular, we hypothesize that some of the variables that are most important in the estimation of the survival function of limit orders include future evolution of the bid-ask spread (smaller spreads would incentivise traders to place a liquidity taking order on the other side of the book), volatility (due to its correlation with the spread), or order arrival speed (as traders who observe large queues forming will be more incentivised to cross the bid-ask spread). Volatility, bid-ask spread, and book size (which is a proxy execution immediacy) have been shown to exhibit significant persistance and cross-correlations, see [25], which further motivates

[5]The volatility is estimated using a rolling mean of 1000 trades over the squared returns of the midprice time-series.

[6]Another possible way of homogenizing these effects is to consider the evolution of the day in *volume time*.

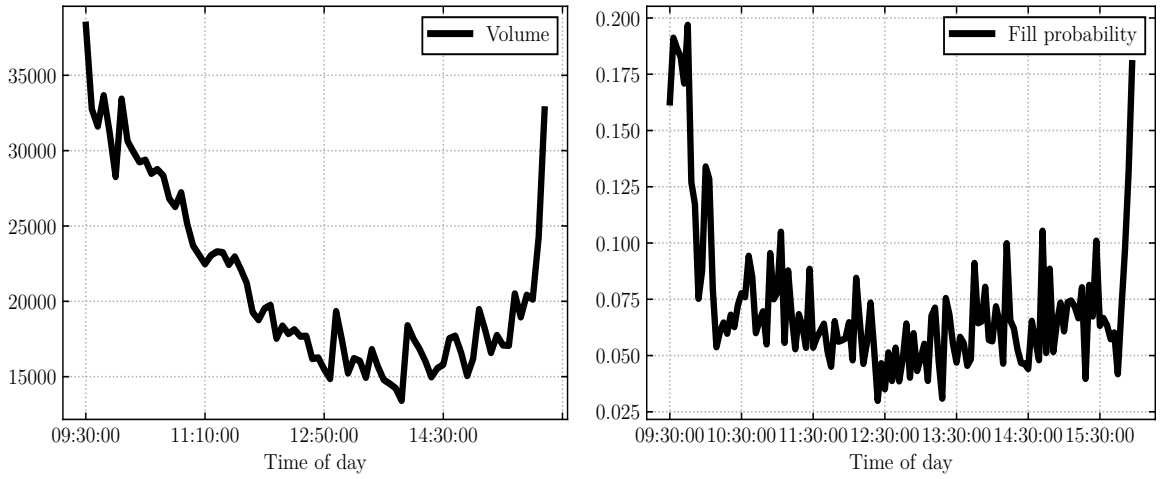


Figure 3.8: 5-minute bucket average statistics for AAPL stock over October 2022. **Left:** Daily traded volume. **Right:** Fill probability of limit orders posted at the best level in the book.

the use of an attention-based encoder to capture long-ranging dependencies between the different time-series.

We build two handcrafted signals with fast moving features. These are *volume imbalance*, which is defined as

$$\Upsilon_t = \frac{v_b^1(t) - v_a^1(t)}{v_b^1(t) + v_a^1(t)} \in [-1, 1]\,, \tag{3.16}$$

and the *microprice*, which is given by

$$M_t = \frac{v_b^1(t)}{v_b^1(t) + v_a^1(t)}\, p_a^1(t) + \frac{v_a^1(t)}{v_b^1(t) + v_a^1(t)}\, p_b^1(t)\,. \tag{3.17}$$

Volume imbalance captures the difference between the buy and sell pressures in the order book at time $t$, and it is a predictor of the arrival of aggressive orders, limit orders, and short-term price moves, as detailed in [41] and [51]. When $\Upsilon_t$ is close to 1, there is buy pressure, and when it is close to $-1$, there is sell pressure. Similarly, the microprice reflects the tendency of the price to move toward the bid or the ask side of the book. An example evolution of these indicators over a horizon of 1000 trades is shown in Figure 3.9.

These two signals are added as inputs to the model given their widespread use and well-known predictive power over short horizons. We also include the raw volumes and prices of the top five levels of the order book to allow our model to find more complex inter-dependencies directly from the data.

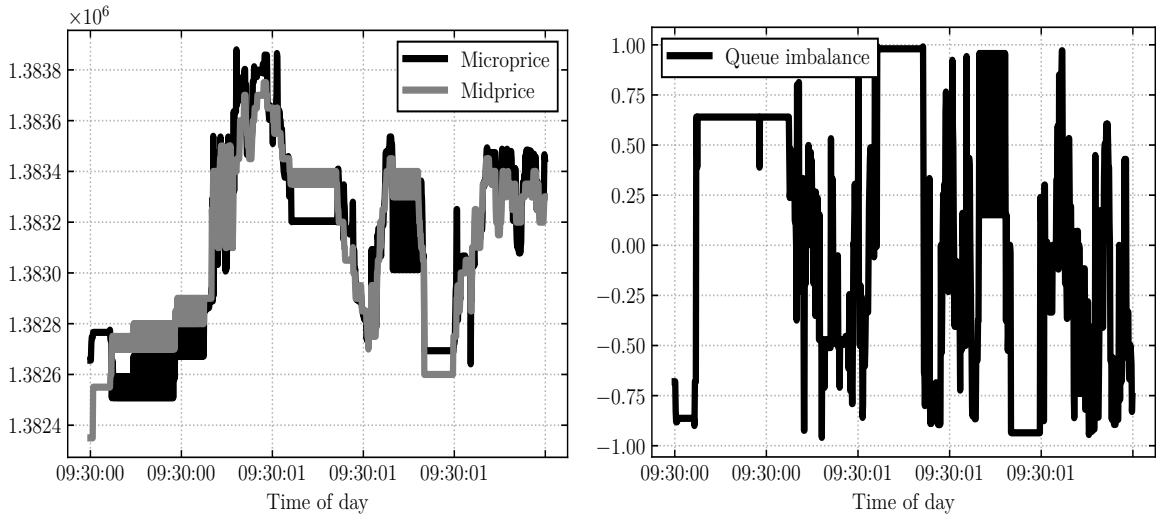


Figure 3.9: Evolution of indicators over the first 1000 trades of 1 October 2022 for the AAPL stock. **Left:** Midprice and microprice. **Right:** Queue imbalance.

### 3.6.2 Model Fit

In this subsection, we report the results of our model. We compare the performance of our model with classic deep learning benchmarks from the surival analysis literature. In particular, we consider the DeepSurv [154] and DeepHit [173] models. Furthermore, we test the effectiveness of our encoder by replacing it with an Multi-Layer Perceptron (MLP) (which results in the architecture introduced in [222]), a CNN, and a long short-term memory (LSTM) network [142]. The latter two models are the ones expected to be in closest competition with the convolutional transformer, as they also model the temporal dynamics observed in the data.

We train our model with data from 1 September 2022 to 26 December 2022. We use the tickers shown in Table 3.1, and the features described in the previous subsection (time of day, volatility, volume imbalance, microprice and prices and volumes of the best five levels). For each trading day, we either track the fill statistics of 100 random orders that were placed by market participants into the order book or place and track the same number of hypothetical limit orders pegged to the best level of the book. We store the features over different lookback horizons of 50, 500, and 1000 trades to explore whether information in the distant past is informative to predict the fill probability.

The performance of the model in terms of negative RCLL is provided in Tables 3.4 and 3.5 for the tracked and hypothetical limit orders respectively. In turn, Tables 3.7 and 3.6 show the performance improvement of each model over using the one using an MLP as

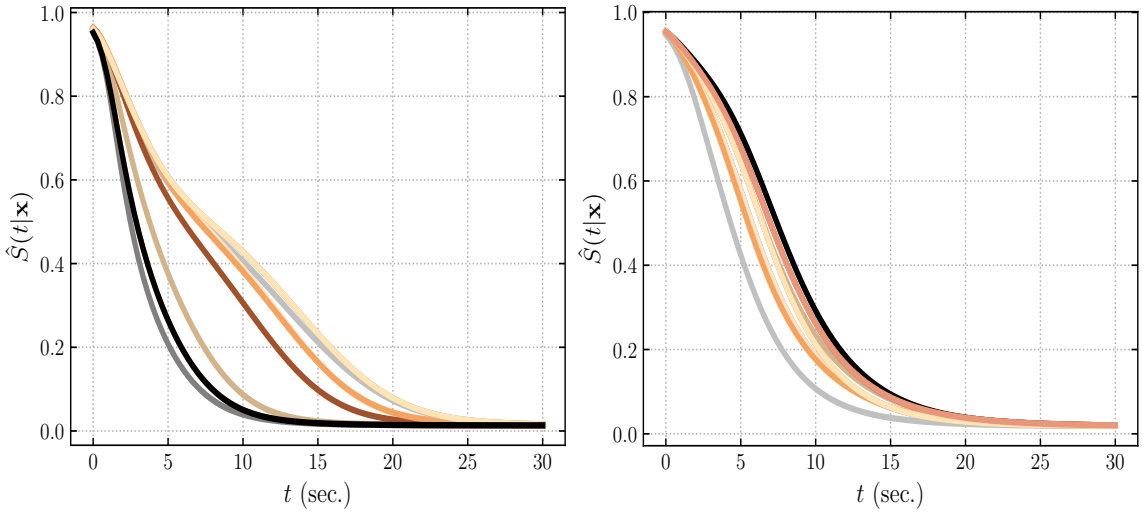


Figure 3.10: Survival functions predicted by the encoder-decoder monotonic convolutional-Transformer model for a batch of different limit orders. **Left:** AAPL, **Right:** AMZN.

an encoder for both order types. The best result for each ticker appears in bold, with our proposed model outperforming all benchmarks. All models which account for the time-varying dynamics of the limit order book achieve significant performance gains, which suggests that high-frequency microstructural information plays a mayor role in the prediction. Furthermore, models with an LSTM or CNN encoder do not exhibit as significant gains (or even incur in some performance degradation) when considering longer lookback windows, due to the inability to summarize the information of the entire horizon. In contrast, the convolutional-Transformer encoder is a able to weigh the information over the entire horizon by its relevance to the final prediction, allowing it to achive the best performance for longer time windows. As mentioned our model achieves this performance at a lower parameter count in comparison to other models such as the one using the CNN as an encoder.

For completeness, we use an order flow representation of the order book to carry out the same analysis and summarise the results in 3.8.7. The benefits of such an approach are summarised in [163] and [181], where authors suggest that considering order flow or volume representations of the order book increase predictive performance for step-ahead forecasts, while making the use of more complex models unnecessary. In our case, however, we find that the convolutional transformer still outperforms all other models in this setting. This suggests that some of the representations used to perform directional price forecasts are not as relevant when estimating the survival functions of limit orders. A pos-

Table 3.4: Model performance for observed orders dataset, in terms of RCLL.

| Mean ± STD Negative RCLL | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** |
| *No recurrence* | | | | | | | | | |
| **DeepSurv** | 1.572 ± 0.032 | 1.155 ± 0.369 | 1.913 ± 0.192 | 1.243 ± 0.326 | 1.214 ± 0.286 | 0.998 ± 0.035 | 1.432 ± 0.261 | 1.288 ± 0.214 | 1.282 ± 0.049 |
| **DeepHit** | 1.230 ± 0.054 | 1.418 ± 0.098 | 1.926 ± 0.093 | 1.304 ± 0.243 | 1.122 ± 0.089 | 0.922 ± 0.096 | 1.314 ± 0.116 | 1.188 ± .0399 | 1.362 ± 0.075 |
| **MN-MLP** | 1.465 ± 0.364 | 1.754 ± 0.295 | 1.943 ± 0.972 | 1.987 ± 0.261 | 1.273 ± 0.410 | 1.312 ± 0.349 | 1.553 ± 1.405 | 1.511 ± 0.352 | 1.912 ± 1.058 |
| *T=50* | | | | | | | | | |
| **MN-CNN** | 0.571 ± 0.095 | 0.594 ± 0.071 | 0.954 ± 0.042 | 0.647 ± 0.037 | 1.200 ± 0.165 | 0.835 ± 0.057 | 0.657 ± 0.006 | 0.757 ± 0.174 | 0.676 ± 0.109 |
| **MN-LSTM** | 0.820 ± 0.611 | 0.390 ± 0.198 | 1.265 ± 0.808 | 0.883 ± 0.409 | 0.884 ± 0.377 | 0.559 ± 0.468 | 1.234 ± 0.731 | 1.066 ± 0.650 | 0.963 ± 0.268 |
| **MN-Conv-Trans** | **0.142 ± 0.185** | 0.132 ± 0.195 | 0.406 ± 0.364 | 0.178 ± 0.299 | 0.242 ± 0.303 | 0.168 ± 0.255 | 0.233 ± 0.210 | 0.341 ± 0.454 | 0.180 ± 0.201 |
| *T=500* | | | | | | | | | |
| **MN-CNN** | 1.043 ± 0.177 | 0.629 ± 0.059 | 0.714 ± 0.104 | 0.362 ± - 0.0 | 0.805 ± 0.121 | 0.754 ± 0.108 | 0.854 ± 0.117 | 1.188 ± 0.166 | 1.456 ± 0.122 |
| **MN-LSTM** | 0.977 ± 0.500 | 0.414 ± 0.097 | 0.966 ± 0.374 | 0.078 ± 0.129 | 1.145 ± 0.517 | 0.449 ± 0.311 | 0.620 ± 0.208 | 0.959 ± 0.423 | 1.322 ± 0.546 |
| **MN-Conv-Trans** | 0.180 ± 0.153 | 0.177 ± 0.166 | **0.296 ± 0.300** | **0.027 ± 0.040** | 0.365 ± 0.323 | **0.167 ± 0.250** | 0.308 ± 0.236 | **0.281 ± 0.293** | 0.188 ± 0.119 |
| *T=1000* | | | | | | | | | |
| **MN-CNN** | 0.757 ± 0.28 | 0.581 ± 0.092 | 0.919 ± 0.075 | 1.345 ± 0.068 | 1.185 ± 0.277 | 0.904 ± 0.138 | 0.657 ± 0.074 | 1.303 ± 0.247 | 0.677 ± 0.092 |
| **MN-LSTM** | 1.013 ± 0.495 | 0.402 ± 0.173 | 1.633 ± 0.475 | 1.053 ± 0.663 | 0.889 ± 0.406 | 0.871 ± 0.386 | 1.240 ± 0.647 | 1.405 ± 0.501 | 0.858 ± 0.374 |
| **MN-Conv-Trans** | 0.192 ± 0.168 | **0.129 ± 0.171** | 0.405 ± 0.320 | 0.241 ± 0.236 | **0.236 ± 0.291** | 0.179 ± 0.206 | **0.208 ± 0.197** | 0.313 ± 0.345 | **0.179 ± 0.252** |

Table 3.5: Model performance for pegged orders dataset, in terms of RCLL.

| | **Mean ± STD Negative RCLL** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** |
| | | | | | *No recurrence* | | | | |
| **DeepSurv** | 8.109 ± 0.065 | 8.557 ± 0.034 | 7.915 ± 0.103 | 7.982 ± 0.066 | 9.978 ± 0.067 | 8.885 ± 0.064 | 9.131 ± 0.024 | 8.462 ± 0.096 | 8.564 ± 0.008 |
| **DeepHit** | 10.098 ± 0.065 | 10.119 ± 0.028 | 9.716 ± 0.146 | 9.731 ± 0.165 | 9.978 ± 0.021 | 9.816 ± 0.081 | 10.046 ± 0.065 | 9.874 ± 0.022 | 10.074 ± 0.055 |
| **MN-MLP** | 10.554 ± 0.364 | 10.709 ± 0.351 | 13.300 ± 0.171 | 13.603 ± 0.470 | 12.364 ± 0.422 | 13.151 ± 0.445 | 11.330 ± 0.398 | 12.110 ± 0.415 | 10.784 ± 0.354 |
| | | | | | $T = 50$ | | | | |
| **MN-CNN** | 5.075 ± 0.366 | 4.795 ± 0.288 | 13.743 ± 0.158 | 9.474 ± 0.139 | 6.569 ± 0.126 | 8.778 ± 0.553 | 5.535 ± 0.246 | 6.588 ± 0.206 | 5.351 ± 0.045 |
| **MN-LSTM** | 3.896 ± 0.497 | 3.302 ± 0.067 | 6.452 ± 0.181 | 9.539 ± 1.958 | 6.065 ± 0.221 | 7.056 ± 0.114 | 4.385 ± 0.215 | 6.954 ± 0.169 | 4.077 ± 0.255 |
| **MN-Conv-Trans** | 3.201 ± 0.827 | 2.997 ± 0.675 | 5.930 ± 0.863 | 5.957 ± 0.808 | 5.019 ± 1.149 | **5.522 ± 0.860** | 3.730 ± 0.901 | 5.002 ± 1.167 | 3.533 ± 1.090 |
| | | | | | $T = 500$ | | | | |
| **MN-CNN** | 5.056 ± 0.165 | 5.401 ± 0.077 | 10.801 ± 0.221 | 8.910 ± 0.253 | 6.699 ± 0.130 | 7.422 ± 0.135 | 5.536 ± 0.157 | 6.768 ± 0.307 | 5.284 ± 0.165 |
| **MN-LSTM** | 3.701 ± 0.202 | 4.135 ± 0.270 | 7.658 ± 0.303 | 7.681 ± 0.272 | 5.636 ± 0.224 | 7.479 ± 0.385 | 4.338 ± 0.169 | 6.545 ± 0.232 | 4.369 ± 0.189 |
| **MN-Conv-Trans** | **3.171 ± 0.157** | 3.111 ± 0.249 | 6.428 ± 1.231 | **5.822 ± 0.290** | 4.997 ± 0.255 | 5.814 ± 0.424 | 3.729 ± 0.925 | 5.077 ± 0.352 | **3.326 ± 0.309** |
| | | | | | $T = 1000$ | | | | |
| **MN-CNN** | 5.925 ± 0.010 | 4.796 ± 0.142 | 7.485 ± 0.271 | 8.052 ± 0.022 | 6.581 ± 0.290 | 7.171 ± 0.174 | 5.536 ± 0.157 | 7.676 ± 0.540 | 5.347 ± 0.093 |
| **MN-LSTM** | 5.404 ± 0.023 | 3.932 ± 0.154 | 6.945 ± 0.06 | 7.199 ± 0.030 | 6.043 ± 0.219 | 7.202 ± 0.583 | 4.338 ± 0.169 | 5.52 ± .241 | 3.904 ± 0.141 |
| **MN-Conv-Trans** | 3.724 ± 1.226 | **2.980 ± 0.713** | **5.887 ± 0.918** | 6.089 ± 1.073 | **4.974 ± 0.834** | 5.625 ± 0.919 | **3.453 ± 0.498** | **4.951 ± 1.053** | 3.560 ± 1.122 |

Table 3.6: Percentage improvement over the MN-MLP model for observed limit orders dataset.

| | Improvement over MN-MLP (%) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** | **Average** |
| | | | | | *No recurrence* | | | | | |
| **DeepSurv** | -7.30 | 34.15 | 1.54 | 37.44 | 4.63 | 23.93 | 7.79 | 14.76 | 32.95 | 16.66 |
| **DeepHit** | 16.04 | 19.16 | 0.87 | 34.37 | 11.86 | 29.73 | 15.39 | 21.38 | 28.77 | 19.73 |
| | | | | | *T=50* | | | | | |
| **MN-CNN** | 61.02 | 66.13 | 50.90 | 67.44 | 5.73 | 36.36 | 57.69 | 49.90 | 64.64 | 51.09 |
| **MN-LSTM** | 44.03 | 77.77 | 34.89 | 55.56 | 30.56 | 57.39 | 20.54 | 29.45 | 49.63 | 44.42 |
| **MN-Conv-Trans** | **90.31** | 92.47 | 79.10 | 91.04 | 80.99 | 87.20 | 85.00 | 77.43 | **90.59** | **86.01** |
| | | | | | *T=500* | | | | | |
| **MN-CNN** | 28.81 | 64.14 | 63.25 | 81.78 | 36.76 | 42.53 | 45.01 | 21.38 | 23.85 | 45.28 |
| **MN-LSTM** | 33.31 | 76.40 | 50.28 | 96.07 | 10.05 | 65.78 | 60.08 | 36.53 | 30.86 | 51.04 |
| **MN-Conv-Trans** | 87.71 | 89.91 | **84.77** | **98.64** | 71.33 | **87.27** | 80.17 | **81.40** | 90.17 | 85.71 |
| | | | | | *T=1000* | | | | | |
| **MN-CNN** | 60.34 | 66.88 | 13.77 | 32.31 | 32.31 | 31.10 | 57.69 | 13.77 | 13.77 | 35.77 |
| **MN-LSTM** | 30.85 | 77.08 | 7.02 | 47.01 | 47.01 | 33.61 | 20.15 | 7.02 | 7.02 | 30.75 |
| **MN-Conv-Trans** | 86.89 | **93.51** | 79.29 | 87.87 | **87.87** | 86.36 | **86.61** | 79.29 | 79.29 | 85.22 |

Table 3.7: Percentage improvement over the MN-MLP model for pegged limit orders dataset.

| | Improvement over MN-MLP (%) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** | **Average** |
| | | | | | *No Recurrence* | | | | | |
| **DeepSurv** | 23.17 | 20.10 | 40.49 | 41.32 | 19.30 | 32.44 | 19.41 | 30.12 | 20.59 | 27.44 |
| **DeepHit** | 4.32 | 5.51 | 26.95 | 28.46 | 19.30 | 25.36 | 11.33 | 18.46 | 6.58 | 16.25 |
| | | | | | $T = 50$ | | | | | |
| **MN-CNN** | 51.91 | 55.22 | -3.33 | 30.35 | 46.87 | 33.25 | 51.15 | 45.60 | 50.38 | 40.16 |
| **MN-LSTM** | 63.09 | 69.17 | 51.49 | 29.88 | 50.95 | 46.35 | 61.30 | 42.58 | 62.19 | 53.00 |
| **MN-Conv-Trans** | 69.67 | 72.01 | 55.41 | 56.21 | 59.41 | **58.01** | 67.08 | 58.70 | 67.24 | 62.64 |
| | | | | | $T = 500$ | | | | | |
| **MN-CNN** | 52.09 | 49.57 | 18.79 | 34.50 | 45.82 | 43.56 | 51.14 | 44.11 | 51.00 | 43.40 |
| **MN-LSTM** | 64.93 | 61.39 | 42.42 | 43.53 | 54.42 | 43.13 | 61.71 | 45.95 | 59.49 | 53.00 |
| **MN-Conv-Trans** | **69.95** | 70.95 | 51.67 | **57.20** | 59.58 | 55.79 | **67.09** | 58.08 | **69.16** | 62.16 |
| | | | | | $T = 1000$ | | | | | |
| **MN-CNN** | 54.56 | 55.22 | 43.72 | 40.81 | 46.77 | 45.47 | 51.15 | 36.61 | 50.42 | 47.19 |
| **MN-LSTM** | 48.80 | 63.28 | 47.78 | 47.08 | 51.12 | 45.24 | 61.30 | 54.42 | 63.80 | 53.65 |
| **MN-Conv-Trans** | 64.71 | **78.09** | **55.74** | 55.24 | **59.77** | 57.23 | 67.08 | **59.12** | 66.99 | **62.66** |

sible reason for this is that price prediction is a harder task given that it is a directional forecast. Moreover, the prediction of the fill probability is closely linked to the behaviour of the spread, whose prediction is non-directional, and is closely linked to the volatility (which exhibits higher persistence than a regular price time series). As such, the model might be focusing on properties of the features that aid in the prediction of future volatility, making representations that aid in directional price forecasts redundant.

### 3.6.3 Model Interpretability

In this section, we focus on the interpretability of our model. To do so, we analyse both the time and feature domains. In particular, we use attention heatmaps to visualise which parts of the past values of the signals are more important to the model to predict the survival function. Secondly, we use Shapley values [133] to quantify the relative importance of each input feature to the output of the model.

#### 3.6.3.1 Attention Heatmaps

The convolutional-Transformer employs (3.15) to obtain, through the convolutional network, the self-attention input features described in Section 3.5.2: the query, key, and value matrices. The model then performs the dot-product computation between the attention's queries and keys, see (2.11). This operation results in a matrix of dimensions $\mathbb{R}^{T\times T}$, where $T \in \mathbb{Z}$ is the lookback window's length. Finally, the softmax function is applied to this matrix, see (2.11), the output is used to multiply the previously obtained self-attention's values. Therefore, with the matrix resulting from the dot-product operation, one visualises which regions of the lookback window (more precisely, of its non-linear projection), are given the highest weighting by the model when estimating the survival function. These are commonly known as *attention heatmaps*.

Figure 3.11 shows the four attention heatmaps of our model, one per head, for a single estimate. Further, 3.8.5 shows the corresponding evolution in time of the features. The attention heatmaps display the self-attention weights and provide information on the weight given of each time-step by the model. As shown in the plots, head-0 focuses on samples of 400 trades ago when there was a significant reduction in volatility and the size of the spread. The remaining of the heads have quite a sparse attention pattern, showing that the models accounts for both short and long-term information to make the forecast.

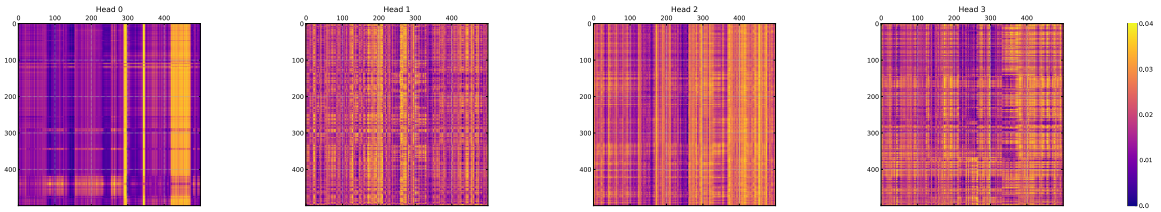


Figure 3.11: Attention heatmaps obtained and example order.

#### 3.6.3.2 Shapley Values

Shapley values provide a method to attribute the predictions of the model to the individual features of the input. In this case, this helps to understand which of the market features are most important and how they contribute to the model's overall prediction. To calculate the Shapley values, we follow the DeepSHAP approach in [182]. To measure the importance per feature, we define $\mathbf{S} \subseteq \mathbf{F}$ as all possible feature subsets, where $\mathbf{F}$ is the set of all features. First, integrate over many background samples to obtain the expected model output $\mathbb{E}[\widehat{f}_{\mathbf{S}}(t|\mathbf{x})]$, where $\widehat{f}_{\mathbf{S}}(t|\mathbf{x})$ denotes the model's prediction while using all the available features. Next, we approximate Shapley values such that they sum up to the difference between the expected output of the model and the predicted values, *i.e.*, $\widehat{f}_{\mathbf{S}}(t|\mathbf{x}) - \mathbb{E}[\widehat{f}(t|\mathbf{x})]$. The contribution of feature $i^{\text{th}}$ is

$$C_i = \widehat{f}_{\mathbf{S}}(t|\mathbf{x}) - \widehat{f}_{\mathbf{S}\setminus\{i\}}(t|\mathbf{x}), \tag{3.18}$$

where $\widehat{f}_{\mathbf{S}\setminus\{i\}}(x)$ is the model's prediction without using the $i^{\text{th}}$ feature, whose Shapley value is given by

$$\phi_i = \frac{1}{n!} \sum_{p=0}^{n} p!\,(n-p)!\,C_i\,, \tag{3.19}$$

where $n$ is the total number of features and $p$ the number of features present in the input sample.

Figure 3.12 shows a Beeswarm plot, which reveals the relative importance of each feature and its relationship with the predicted outcome. Each of the datapoints is represented with a single dot, where colours ranging from blue to red indicate lower to higher values per feature. The vertical axis is ordered in accordance to the importance of each feature for the convolutional-Transformer on average, while the horizontal axis displays the associated Shapley value. This analysis is associated to the more basic MN-MLP model, as the these quantities cannot be computed easily for models which explicitly model the time evolution of the features. This means that the information we extract form this cannot be directly

translated to analyze our proposed model, but can give a good indication of what features are most influencial.

The figure shows that the model gives most importance to fast-moving features. For example, the micro-price is the most important feature because it provides a relatively good estimate of the perceived fundamental values of the asset, which in practice has an effect on where liquidity taking orders are placed. The volatilty of the asset also plays a major role in the prediction, which could be attributed to the fact that it serves a good proxy of the amount volume being traded. If more volume is being traded in the market, there is a higher chance that a liquidity taking order will cross the spread and match an outstanding order. Other features such as time of day shows little to no importance, which is likely due to the fact that time of day shows a very pesistent pattern that is robust to small perturbations in its value.

## 3.7 Conclusion

This chapter presents a novel approach to estimating the fill probabilities of limit orders posted in the LOB. The proposed data-driven approach integrates a novel convolutional-Transformer model to raise local-awareness from LOB data, and a monotonic neural network which provides guarantees on the theoretical correctness of the proposed method. To train and evaluate the proposed model, we use right-censored log-likelihood, which is a proper scoring rule, unlike other scoring rules which are commonly used in the literature. To demonstrate the effectiveness of the proposed method, we conduct a set of experiments on real LOB data. These experiments show that the monotonic encoder-decoder convolutional-Transformer significantly outperforms state-of-the-art benchmarks, and provides a new general framework with which to perform survival analysis from time-series observations. Finally, we provide an interpretability analysis based on Shapley values and attention heatmaps, which provides insight on which of the features considered by the model are the most influencial.

## 3.8 Additional Materials

### 3.8.1 Conditions for an order fill

Following a similar approach to that of [185], we use the following fill conditions for hypothetical limit orders:

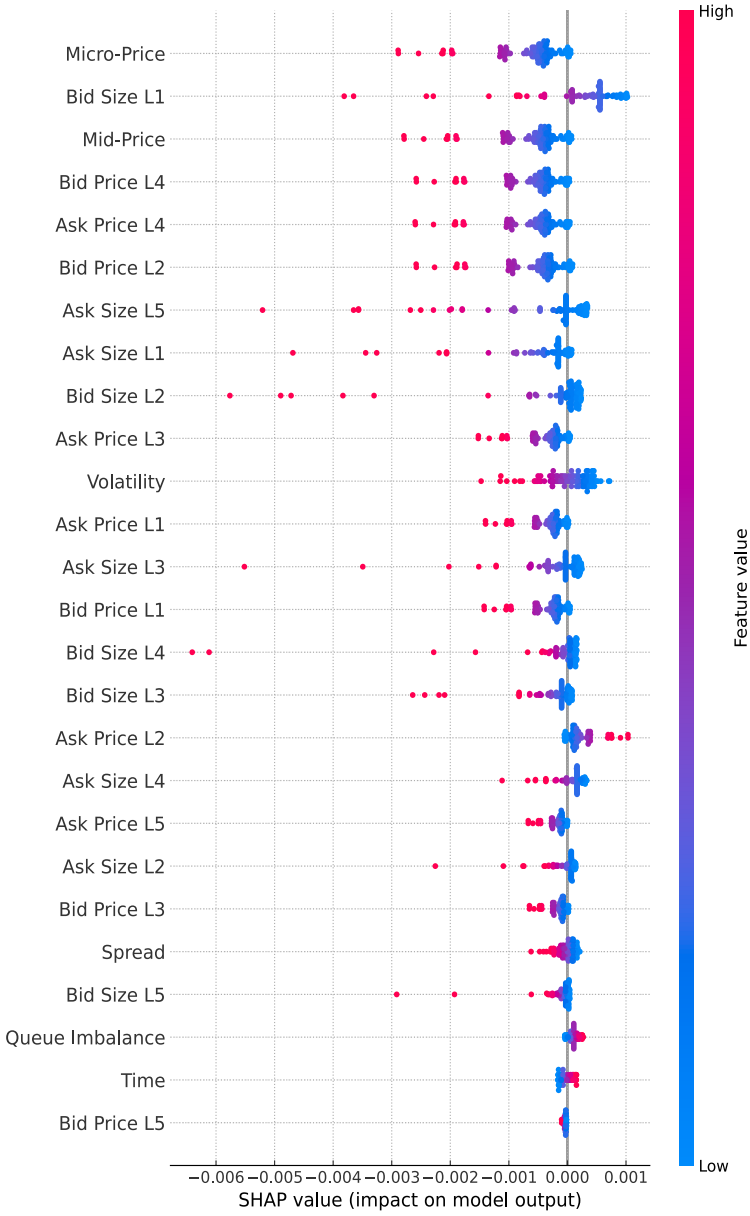


Figure 3.12: Shapley values for 100 predictions the MN-MLP model.

1. A new hypothetical limit order is filled if:

    - A new buy/sell order arrives at a higher/lower price than the synthetic sell/buy limit order, then the hypothetical order is considered filled.
    - If an order in front of the hypothetical order in the execution queue is filled/partially filled, we consider the hypothetical order to be filled as well.

2. A new market order comes in at the same price than our synthetic order:

    - If the market order crossed against any of the orders being tracked, then our hypothetical order is considered filled.

From an implementation perspective, we do not require maintaining a full log of all the orders in the queue using a linked list. It suffices to track the messages associated to orders in front of the hypothetical limit order in the queue and market orders arriving above/below the hypothetical order, depending on the side it was placed.

### 3.8.2 Derivation of right-censored log-likelihood

Assume we have a dataset of observations $\mathcal{D} = \{(\mathbf{x}_k, z_k, \delta_k)\}_{k=1}^N$, where $z_k = \min\{T_l, C_l\}$, where the random variables $T_l$ and $C_l$ denote the random fill and cancellation (censoring) times, respectively. For clarity, in the remaining derivations of the section, we drop the subscript $l$. The likelihood function is

$$\begin{aligned} L &= f(z_1, \delta_1, \ldots, z_N, \delta_N) \\ &= \prod_{k=1}^{N} f(z_k, \delta_k), \end{aligned} \tag{3.20}$$

where we assume that pairs are independent. To re-write this equation depending on the values of the indicator variable $\delta_k$, we first consider the case in which the order is filled, and the event is therefore observed ($\delta_k = 1$)

$$\begin{aligned} f(z_k, \delta_k) &= \mathbb{P}\{Z = z_k, \delta_k = 1\} \\ &= \mathbb{P}\{T = z_k, T < C\} \\ &= \mathbb{P}\{T = z_k, z_k < C\} \\ &= \mathbb{P}\{T = z_k\}\mathbb{P}\{z_k < C\} \quad \text{(assuming } T \text{ is independent of } C\text{)} \\ &= f_T(z_k) S_C(z_k). \end{aligned} \tag{3.21}$$

Moreover, if the observation is censored, we have that $\delta_k = 0$ and

$$\begin{aligned} f(z_k, \delta_k) &= \mathbb{P}\{Z = z_k, \delta_k = 0\} \\ &= \mathbb{P}\{C = z_k, T > C\} \\ &= \mathbb{P}\{C = z_k, z_k < T\} \\ &= \mathbb{P}\{C = z_k\}\mathbb{P}\{z_k < T\} \quad \text{(assuming } T \text{ is independent of } C\text{)} \\ &= f_C(z_k)S_T(z_k), \end{aligned} \tag{3.22}$$

and re-write (3.20) as

$$\begin{aligned} L &= \prod_{k=1}^{N} \left[f_T(z_k)S_C(z_k)\right]^{\delta_k} \left[f_C(z_k)S_T(z_k)\right]^{(1-\delta_k)} \\ &= \prod_{k=1}^{N} \left[f_T(z_k)^{\delta_k} S_T(z_k)^{(1-\delta_k)}\right] \left[f_C(z_k)^{(1-\delta_k)} S_C(z_k)^{\delta_k}\right]. \end{aligned} \tag{3.23}$$

We are concerned with the estimation of $S_T(t)$ and not $S_C(t)$, because $S_C(t)$ contains information related to censoring mechanism. Only $S_T(t)$ contains information about the fill-times, which is the variable of interest. Thus, the terms that do not involve $T$ are considered constants, and the log-likelihood is

$$\mathcal{L} = \log(L) = \sum_{k=1}^{N} \delta_k \log(\widehat{f}(z_k)) + (1 - \delta_k) \log(\widehat{S}(z_k)). \tag{3.24}$$

### 3.8.3 Extending the Encoder's Dilated Causal Convolutional Neural Network to $L$ layers

To extend the original encoder's DCC in charge of obtaining the corresponding queries, keys, and keys in the proposed convolutional-Transformer, the causal convolutional network should be modified as follows. The first layer would perform the same operation, convolving the input sequences $x$ and the kernel $k$:

$$F^{(l=1)}(t) = \left(x *_p k^{(l=1)}\right)(t) = \sum_{\tau=0}^{s-1} k_\tau^{(l=1)} \cdot x_{t-p\cdot\tau}, \tag{3.25}$$

where $p$ is the dilation factor and $k$ the convolutional filter with size $s \in \mathbb{Z}$. For each of the rest $l$-th layers, we could define the convolution operation as:

$$F^{(l)}(t) = \left(F^{(l-1)} *_d k^{(l)}\right)(t) = \sum_{\tau=0}^{s-1} k_\tau^{(l)} \cdot F_{t-p\cdot\tau}^{l-1}(t). \tag{3.26}$$

Each of the layers in this hierarchical structure defines the kernel operation as an affine function acting between layers:

$$k^{(l)} : \mathbb{R}^{N_l} \longrightarrow \mathbb{R}^{N_{l+1}}, 1 \leqslant l \leqslant L. \tag{3.27}$$

The previous equation shows how, through the use of residual connections, firstly proposed in [138], the encoder's convolutional network could connect $l^{\text{th}}$ layer's output to $(l+1)^{\text{th}}$ layer's input, enabling the usage of deeper models with larger receptive fields to generate the hidden representation that is used by the Transformer self-attention's inputs.

### 3.8.4 Monotonicity of the Decoder

This condition is satisfied, because we consider a set-up where all the intermediate layers $h$ of the network, with $1 < h < H$ (not to be confused with $h_i$ in Section 3.5.2 to accredit the Transformer model's head $h_i \in H$), have the following input-output relationship for each node $j$ and input $\mathbf{x}^h \in \mathbb{R}^{M^h}$ (ignoring the bias terms)

$$\mathbf{y}_j^h = \tanh\left( \sum_{i=1}^{M^h} w_{ij}^h \, x_i^h \right) , \tag{3.28}$$

where the final output is given by

$$f_{\mathbf{\Psi}}(t, \mathbf{x}) = P(T < t | \mathbf{X} = \mathbf{x}) = \sigma\left( \sum_{i=1}^{M^H} w_{i1}^H \, x_i^H \right) . \tag{3.29}$$

Here, $w_{ij}^h$ and $b_{ij}^h$ are the individual weight and bias terms associated to node $j$ and layer $h$, respectively, and $\tanh(\cdot)$ and $\sigma(\cdot)$ are the hyperbolic tangent and sigmoid functions, respectively. To enforce monotonicity of the output with respect to the response variable $t$, we impose $w_{ij}^h > 0 \quad \forall h \in \{1, \ldots, H\}$, because the derivative of the output of each node in the first layer is

$$\frac{\partial \mathbf{y}_j^1}{\partial t} = \tanh'\left( w_{1M^1}^1 t + \sum_{i=2}^{M^1} w_{ij}^1 \, x_i^1 \right) w_{1M^1}^1 , \tag{3.30}$$

which requires positivity of the $w_{1M^1}^1$ to guarantee the monotonicity condition. From the chain rule, a similar argument holds for all nodes in subsequent layers of the network. Finally, the remaining two conditions are satisfied empirically given the likelihood-based training method.

### 3.8.5 Order Features

The evolution over 500 trades of the features considered to produce the attention heatmaps are shown in the Figure 3.13.

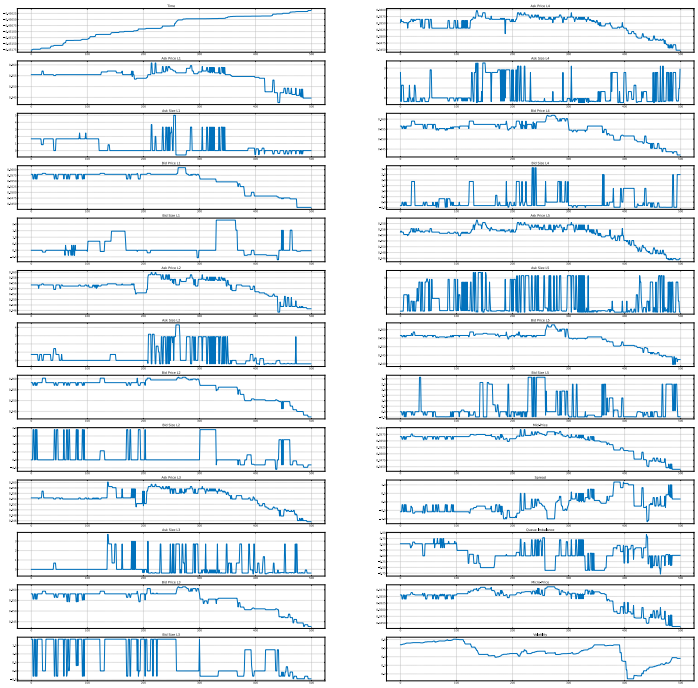

Figure 3.13: Evolution of order features over 500 trades.

### 3.8.6 Kernel Sizes

Table 3.8 explores different kernel sizes, $s \in \{1, 2, 3, 5, 10, 25, 50\}$, for the convolutional operation of the MN-Conv-Trans model for the estimation of AAPL's survival function with a lookback window of $T = 500$ trades. Recall that a convolutional network with kernel size $s = 1$ results on canonical self-attention, in which case, there is an evident decline in the negative RCLL in comparison to larger kernel sizes. No substantial variations in the performance are observed among the other kernel sizes. Therefore, a value of $s = 3$ seems reasonable to avoid an unnecessary increase in parametric complexity.

Table 3.8: Performance variation while when different kernel's sizes for the MN-Conv-Trans DCC network. AAPL stock with a lookback window of 500 trades.

| **Kernel Size** | **Mean $\pm$ STD Negative RCLL** |
|---|---|
| $s = 1$ | 3.245 $\pm$ 0.207 |
| $s = 2$ | 3.184 $\pm$ 0.343 |
| $s = 3$ | 3.171 $\pm$ 0.157 |
| $s = 5$ | 3.189 $\pm$ 0.433 |
| $s = 10$ | 3.179 $\pm$ 0.367 |
| $s = 25$ | 3.196 $\pm$ 0.383 |
| $s = 50$ | 3.170 $\pm$ 0.370 |

### 3.8.7 Performance of Order Flow Representations

Table 3.9: Models evaluated using the negative right-censored log-likelihood for a lookback window of $T = 500$ on the order flow data, and percentage improvement, over the MN-MLP, for each of the evaluated models.

| **Mean ± STD Negative RCLL** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** |
| **DeepSurv** | 8.053 ± 0.019 | 8.346 ± 0.011 | 7.236 ± 0.051 | 9.712 ± 0.047 | 8.441 ± 0.022 | 8.441 ± 0.022 | 8.242 ± 0.004 | 6.211 ± 0.057 | 9.117 ± 0.043 |
| **DeepHit** | 10.015 ± 0.091 | 10.102 ± 0.024 | 9.627 ± 0.064 | 10.036 ± 0.043 | 9.843 ± 0.083 | 9.843 ± 0.083 | 10.142 ± 0.042 | 9.795 ± 0.157 | 10.112 ± 0.062 |
| **MN-MLP** | 10.535 ± 0.339 | 10.801 ± 0.347 | 13.518 ± 0.497 | 12.502 ± 0.431 | 13.158 ± 0.495 | 12.160 ± 0.451 | 10.800 ± 0.311 | 12.938 ± 0.422 | 10.963 ± 0.326 |
| **MN-CNN** | 5.176 ± 0.343 | 5.427 ± 0.173 | 7.921 ± 0.162 | 6.741 ± 0.224 | 7.434 ± 0.212 | 6.754 ± 0.291 | 5.469 ± 0.621 | 7.331 ± - 0.0 | 5.646 ± 0.163 |
| **MN-LSTM** | 3.746 ± 0.136 | 4.838 ± 0.248 | 7.32 ± .613 | 5.941 ± 0.195 | 7.193 ± 0.129 | 6.487 ± 0.198 | 4.890 ± 0.538 | 8.583 ± 0.387 | 4.352 ± 0.231 |
| **MN-Conv-Trans** | **3.158 ± 1.138** | **3.546 ± 1.002** | **6.107 ± 1.008** | **5.220 ± 1.013** | **6.211 ± 1.103** | **5.245 ± 1.162** | **3.595 ± 1.001** | **5.997 ± 1.173** | **3.772 ± 0.918** |
| **Improvement over MN-MLP (%)** | | | | | | | | | |
| | **AAPL** | **AMZN** | **BIDU** | **COST** | **CSCO** | **DELL** | **GOOG** | **INTC** | **MSFT** |
| **DeepSurv** | 23.56 | 22.73 | 46.47 | 22.32 | 35.85 | 30.58 | 23.69 | 51.99 | 16.84 |
| **DeepHit** | 4.94 | 6.47 | 28.78 | 19.72 | 25.19 | 19.05 | 6.09 | 24.29 | 7.76 |
| **MN-MLP** | - | - | - | - | - | - | - | - | - |
| **MN-CNN** | 50.87 | 49.75 | 41.40 | 46.08 | 43.50 | 44.46 | 49.36 | 43.34 | 48.50 |
| **MN-LSTM** | 64.44 | 55.21 | 45.85 | 52.48 | 45.33 | 46.65 | 54.72 | 33.66 | 60.30 |
| **MN-Conv-Trans** | **70.02** | **67.17** | **54.82** | **58.25** | **52.80** | **56.87** | **66.71** | **53.65** | **65.59** |

# Chapter 4

# Rough Transformers

## 4.1 Introduction

Real-world sequential data in areas such as healthcare [213], finance [137], and biology [106] often are irregularly sampled, of variable length, and exhibit long-range dependencies. Furthermore, these data, which may be drawn from financial limit order books [50] (as seen in the previous chapter) or EEG readings [259], are often sampled at high frequency, yielding long sequences of data. Hence, many popular machine learning models struggle to model real-world sequential data, due to input dimension inflexibility, memory constraints, and computational bottlenecks. Rather than treating these data as *discrete* sequences, effective theoretical models often assume data are generated from some underlying *continuous-time* process [195, 219]. Hence, there is an increased interest in developing machine learning methods that use *continuous-time* representations to analyze sequential data.

One recent approach to modelling continuous-time data involves the development of continuous-time analogues of standard deep learning models, such as Neural ODEs [60] and Neural CDEs [161], which extend ResNets [138] and RNNs [108], respectively, to continuous-time settings. Instead of processing discrete data directly, these models operate on a latent continuous-time representation of input sequences. This approach is successful in continuous-time modelling tasks where standard deep recurrent models fail. In particular, extensions of vanilla Neural ODEs to the time-series setting [227, 161] succeed in various domains such as adaptive uncertainty quantification [201], counterfactual inference [239], or generative modelling [38].

In the meantime, in many practical settings, such as financial market volatility [71, 198] or heart rate fluctuations [136], continuous-time data also exhibit long-range dependencies. That is, data from the distant past may impact the system's current behavior. Deep recurrent models struggle in this setting due to vanishing gradients. Several recent works [191, 200] successfully extract long-range dependencies from sequential data with Transformers [260], which learn temporal dependencies of a tokenized representation of input sequences. The parallelizable nature of the Transformer allows for rapid training and evaluation on sequences of moderate length, and has contributed to its success in fields such as natural language processing (NLP).

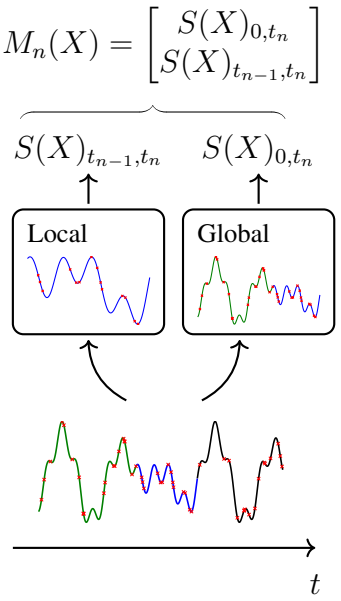


Figure 4.1: A representation of the multi-view signature. The continuous-time path is irregularly sampled at points marked with a red $\times$. The local and global signatures of a linear interpolation of these points are computed and concatenated to form the multi-view signature. The multi-view signature transform consists of $\overline{L}$ multi-view signatures.

While the above approaches succeed in certain settings, several limitations hinder their wider applications. On the one hand, Neural ODEs and their analogues [161, 227] bear substantial computational costs when modelling long sequences of high dimension; see [199]. On the other hand, Transformers operate on discrete-time representations of input sequences, whose relative ordering is represented by the positional embedding. This representation may inhibit their expressivity in continuous-time data modelling tasks [279]. Moreover, Transformer-based models suffer from a number of difficulties, including (i) input sequences must be sampled at the same times, (ii) the sequence length must be fixed, and (iii) the computational cost scales quadratically in the length of the input sequence.

These difficulties severely limit the application of Transformers to continuous-time data modelling.

**Contributions 1)** We introduce *Rough Transformers*, a variant of the Transformer architecture amenable to the processing of continuous-time signals, which can be easily integrated into existing code-bases. The Rough Transformer is built upon the path signature from Rough Path Theory [183]. We define a novel, multi-scale transformation which projects discrete input data to a continuous-time path and compresses the input data with minimal information loss. Moreover, this transformation is an efficient feature representation of continuous-time paths, because linear functionals of path signatures approximate continuous functions of paths arbitrarily well (see Theorem 4.7.2 in Section 4.7.1).

**2)** We introduce the *multi-view attention mechanism* to extract both local and global dependencies of very long time-series efficiently. This mechanism operates directly on continuous-time representations of data without the need for expensive numerical solvers or constraints on the smoothness of the data stream. Moreover, the multi-view attention mechanism is provably robust to irregularly sampled data.

**3)** We carry out extensive experimentation on long and irregularly sampled time-series data. In particular, we show that Rough Transformers (i) improve the learning dynamics of the Transformer, making it more sample-efficient and allowing it to achieve better out-of-sample results, (ii) reduce the training cost by a factor of up to $25\times$ when compared with vanilla Transformers and more when compared with Neural ODE based architectures, (iii) maintain similar performance when data are irregularly sampled, where traditional recurrent-based models suffer a substantial decrease in performance [227], and (iv) yield improved spatial processing, accounting for relationships between different temporal channels without having to pre-define a specific inter-channel relation structure.

## 4.2 Background and Methodology

**Problem Formulation.** In many real-world scenarios, sequential data are time-series sampled from some underlying continuous-time process, so datasets consist of long, irregularly sampled sequences of varied lengths. In these settings, the problem of sequence modelling is described as follows. Let $\widehat{\mathbf{X}} \in C(\mathbb{R}^+;\mathbb{R}^d)$ be a continuous-time *path*. A time-series of length $L$ with sampling times $\mathcal{T}_{\mathbf{X}} = (t_1, ..., t_n)$ is defined as $\mathbf{X} = ((t_0, \mathbf{X}_0), ..., (t_L, \mathbf{X}_L))$, where $\mathbf{X}_i = \widehat{\mathbf{X}}(t_i) \in \mathbb{R}^d$. Now, define a functional $f : C(\mathbb{R}^+;\mathbb{R}^d) \to \mathbb{R}^k$, where $C(\mathbb{R}^+;\mathbb{R}^d) = \{g : \mathbb{R}^+ \to \mathbb{R}^d \,|\, g \text{ continuous}\}$. Next define a dataset $\mathcal{D} = \left\{(\mathbf{X}^i, f(\widehat{\mathbf{X}}^i))_{i=1}^N\right\}$.

We seek to approximate the function $f$ from the set $\mathcal{D}$ for some downstream task. Importantly, we do not assume that $\mathcal{T}_{\mathbf{X}} = \mathcal{T}_{\mathbf{Y}}$ for all $\mathbf{X}, \mathbf{Y} \in \mathcal{D}$, so that $\mathcal{D}$ may be irregularly sampled.

**Sequence Modelling with Transformers.** Transformers are used extensively as a baseline architecture to approximate functions of discrete-time sequential data and are successfully applied to settings when input sequences are of similarly short length, and sampled at regular intervals. The input length $L$ of all time series Transformer is typically assumed to be the same or similar. To evaluate the Transformer on a time-series $\mathbf{X}$ with $|\mathcal{T}_{\mathbf{X}}| \neq n$, one must perform some transformation (interpolation, padding, etc.) in order to train on a batch of time series, which may degrade the performance of the model. Furthermore, the memory and time complexity of the Transformer scales with $O(L^2)$, which presents a substantial difficulty in modelling long sequences.

**Rough Path Signatures.** Broadly, the difficulties faced by the Transformer in modelling time-series stem from time-series being sampled from underlying *continuous-time* objects, while the attention mechanism underpinning the Transformer is designed to model discrete sequences. To address these difficulties, Rough Transformers augment standard Transformers by lifting the input time-series to the space of continuous-time functions and performing the self-attention calculation in this infinite-dimensional space. To achieve this, we use the path signature from Rough Path Theory.

For a continuous-time path $\widehat{\mathbf{X}} \in C_b^1(\mathbb{R}^+; \mathbb{R}^d)$ and times $s, t \in \mathbb{R}^+$, the path signature of $\widehat{\mathbf{X}}$ from $s$ to $t$, denoted $S(\widehat{\mathbf{X}})_{s,t}$, is defined as follows. First, let

$$\mathcal{I}_d = \{(i_1, ..., i_p) : i_j \in \{1, ..., n\} \, \forall \, j \text{ and } p \in \mathbb{N}\} \tag{4.1}$$

denote the set of all $d$-multi-indices and $\mathcal{I}_d^n = \{I \in \mathcal{I}_d : |I| = n\}$. Next, set $S(\widehat{\mathbf{X}})_{s,t}^0 := 1$ and for any $I \in \mathcal{I}_d$, define

$$S(\widehat{\mathbf{X}})_{s,t}^I = \int_{s<u_1<\ldots<u_p<t} \dot{\widehat{\mathbf{X}}}^{i_1}(u_1) \cdots \dot{\widehat{\mathbf{X}}}^{i_p}(u_p) \, du_1 \ldots du_p \,, \tag{4.2}$$

where $\dot{\widehat{\mathbf{X}}}^j = d\widehat{\mathbf{X}}^j/dt$. Abusing notation, define level $n$ of the signature as

$$S(\widehat{\mathbf{X}})_{s,t}^n = \left\{ S(\widehat{\mathbf{X}})_{s,t}^I : I \in \mathcal{I}_d^n \right\} . \tag{4.3}$$

and define the signature as the infinite sequence

$$S(\widehat{\mathbf{X}})_{s,t}^n = (S(\widehat{\mathbf{X}})_{s,t}^0, S(\widehat{\mathbf{X}})_{s,t}^1, ..., S(\widehat{\mathbf{X}})_{s,t}^n, ...) \, . \tag{4.4}$$

Finally, define the truncation of the signature $S(\widehat{\mathbf{X}})_{s,t}^{\leqslant n} = (S(\widehat{\mathbf{X}})_{s,t}^{0}, ..., S(\widehat{\mathbf{X}})_{s,t}^{n})$, where $S(\widehat{\mathbf{X}})_{s,t}^{n}$ can be interpreted as an element of the *extended tensor algebra* of $\mathbb{R}^d$:

$$T((\mathbb{R}^d)) = \left\{(a_0, ..., a_n, ...) : a_n \in \mathbb{R}^{d \otimes n}\right\} . \tag{4.5}$$

Analogously, we say that $S(\widehat{\mathbf{X}})_{s,t}^{\leqslant n} \in T((\mathbb{R}^d))_{\leqslant n}$. A central property of the signature is that is invariant with respect to time-reparameterization [183]. That is, let $\gamma : [0, T] \to [0, T]$ be surjective, continuous, and non-decreasing. Then we have

$$S(\widehat{\mathbf{X}})_{0,T} = S(\widehat{\mathbf{X}} \circ \gamma)_{0,T} , \tag{4.6}$$

which will be crucial to demonstrate the Rough Transformer's robustness to irregularly sampled data. For a more rigorous presentation of signatures and a description of additional properties, see Section 4.7.1 and [183].

## 4.3 Rough Transformers

Now, we construct the Rough Transformer, a Transformer-based architecture that operates on continuous-time sequential data by means of the path signature.

Let $\mathcal{D}$ be a dataset of irregularly sampled time-series. To project a discretized time-series $\mathbf{X} \in \mathcal{D}$ to a continuous-time object, let $\widetilde{\mathbf{X}}$ denote the piecewise-linear interpolation of $\mathbf{X}$.[1] Next, for $t_k \in \mathcal{T}$, define the *multi-view signature*

$$M(\mathbf{X})_k := \left(S(\widetilde{\mathbf{X}})_{0,t_k}, S(\widetilde{\mathbf{X}})_{t_{k-1},t_k}\right) . \tag{4.7}$$

In what follows, we refer to the components of $(S(\widetilde{\mathbf{X}})_{0,t_k}, S(\widetilde{\mathbf{X}})_{t_{k-1},t_k})$ as *global* and *local*, respectively; see Figure 4.1. Intuitively, one can interpret the global component as an efficient representation of long-term information (see Theorem 4.7.2 in Section 4.7.1), and the local component as a type of convolutional filter that is invariant to the sampling rate of the signal. Now, define the *multi-view signature transform* $M(\mathbf{X}) = (M(\mathbf{X})_1, \ldots, M(\mathbf{X})_{\overline{L}})$ , and denote by $M(\mathbf{X})^{\leqslant n}$ the truncated signature for a truncation level $n$. Next, define the *multi-view attention mechanism*, which uses the multi-view signature transform to extend the standard attention mechanism to the space of continuous functions [183]. First, fix a

[1] Any continuous-time interpolation of $\mathbf{X}$ can be used, e.g., splines. However, the signature computation of piecewise-linear paths is particularly fast; see Section 4.7.1.

truncation level $n \in \mathbb{N}$, and let $\overline{d} \in \mathbb{N}$ be such that $M(\mathbf{X})_k^{\leqslant n} \in \mathbb{R}^{\overline{d}}$. For $h = 1, ..., H$ let $\mathbf{W}_h^{\widetilde{\mathbf{Q}},\widetilde{\mathbf{K}},\widetilde{\mathbf{V}}} \in \mathbb{R}^{\overline{d} \times \overline{d}'}$ for some $\overline{d}' \in \mathbb{N}$, and let

$$\widetilde{\mathbf{Q}}_h = M(\mathbf{X})^{\leqslant n} \mathbf{W}_h^{\widetilde{\mathbf{Q}}}, \tag{4.8}$$

$$\widetilde{\mathbf{K}}_h = M(\mathbf{X})^{\leqslant n} \mathbf{W}_h^{\widetilde{\mathbf{K}}}, \tag{4.9}$$

$$\widetilde{\mathbf{V}}_h = M(\mathbf{X})^{\leqslant n} \mathbf{W}_h^{\widetilde{\mathbf{V}}}. \tag{4.10}$$

Then, the attention calculation is given by

$$O_h = \text{softmax}\left(\frac{\widetilde{\mathbf{Q}}_h \widetilde{\mathbf{K}}_h^\intercal}{\sqrt{\overline{d}}}\right) \widetilde{\mathbf{V}}_h \,. \tag{4.11}$$

Notice that the attention calculation is similar to standard attention, however, we stress that the multi-view attention is built on *continuous-time* objects, the signatures, while the standard attention mechanism acts on discrete objects. The multi-view signature provides a compressed representation of the time series, minimizing the computational costs associated to quadratic scaling without excessive loss of representational capacity, see Section 4.7.4.

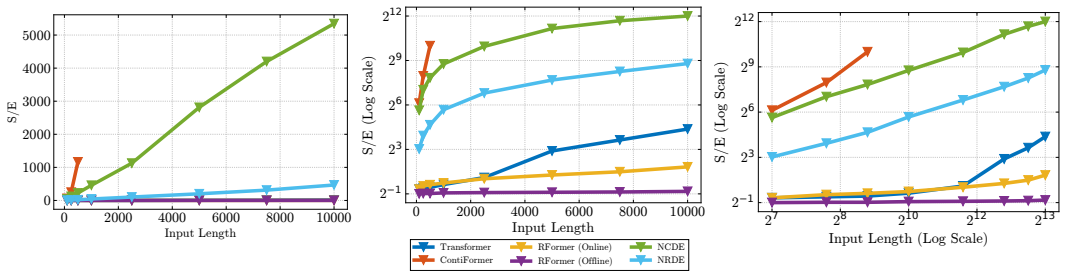


Figure 4.2: Seconds per epoch for growing input length and for different model types on the sinusoidal dataset. **Left:** Log Scale. **Middle:** Regular Scale. **Right:** Log-log scale. When a line stops, it indicates an OOM error.

### 4.3.1 Advantages of Rough Transformers

**Computational Efficiency.** As will be demonstrated in Section 4.4, the multi-view attention mechanism can substantially reduce the computational cost of vanilla Transformers. In particular, the attention calculation decreases from $O(L^2\, d)$ in the vanilla case to $O(\overline{L}^2\, d)$, where $\overline{L} << L$ with Rough Transformers. This enables both faster wall-clock training time and the ability to process long input sequences which would otherwise yield out-of-memory errors for the vanilla Transformer, see Figure 4.2. Moreover, the multi-view attention mechanism does not require backpropagation through the signature calculation and can be computed *offline*. This is significantly more computationally efficient compared

with the complexity of computing signatures batch-wise in every training step. Finally, the signature of piecewise-linear paths can be computed explicitly, see Section 4.7.1, and there are a number of Python packages devoted to optimized signature calculation [159, 221].

**Variable Length and Irregular Sampling.** The multi-view signature transform underpinning Rough Transformers is evaluated by constructing a continuous-time interpolation of input data and computing a series of iterated integrals of this interpolation. The bounds of these integrals are a fixed set of time points, meaning that the sequence length of the multi-view attention mechanism is fixed and independent of the sequence length of input samples. Furthermore, the following proposition shows that the output of the Rough Transformer for two (possibly irregular) samplings of the same path is similar.

**Proposition 4.3.1.** *Let $\mathbb{T}$ be a Rough Transformer. Suppose $\widehat{\mathbf{X}} : [0, T] \to \mathbb{R}^d$ is a continuous-time process, and let $\gamma : [0, T] \to [0, T]$ denote a time-reparameterization. Suppose $\mathbf{X}$ and $\mathbf{X}'$ are samplings of $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{X}} \circ \gamma$, respectively. Then $\mathbb{T}(\mathbf{X}) \approx \mathbb{T}(\mathbf{X}')$.*

*Proof.* By (4.6), $S(\widehat{\mathbf{X}})_{s,t} = S(\widehat{\mathbf{X}} \circ \gamma)_{s,t}$ for all $s, t \in [0, T]$. Hence, one has $M(X^1) \approx M(X^2)$. Finally, $\mathbb{T}(X^1) \approx \mathbb{T}(X^2)$ because the attention mechanism and final MLP are both continuous. □

Hence, the Rough Transformer is robust to irregular sampling. In many tasks, the sampling times convey important information about the time-series. In these settings, one may augment the input time-series with its sampling times, that is, write $X = ((t_0, \mathbf{X}_0), ..., (t_L, \mathbf{X}_L))$.

**Spatial Processing.** While an interpolation of input data could be sampled to make vanilla Transformers independent of the length of the input sequence, important locality information could be lost, see Section 4.7.5. Instead, Rough Transformers summarize spatial interactions between channels by means of the multi-view signature transform. One may notice that in (4.4), the dimension of the signature grows exponentially in the level of the signature $n$. In particular, when $\mathbf{X}_i \in \mathbb{R}^d$, $|S(\widetilde{\mathbf{X}})^{\leqslant n}_{0,t}| = \frac{d(d^n-1)}{d-1} = O(d^n)$, so the multi-view attention calculation is of order $O(\overline{L}^2 d^n)$. In many practical time-series modelling problems, however, the value of $d$ is not very large. The signature terms also decay factorially in the signature level $n$ (see Proposition 4.7.3 in Section 4.7.1), so in practice, one may take the value of $n$ to be small without sacrificing performance. The majority of computational savings result from the reduction of the sequence length to $\overline{L}$, and in practice, we take $\overline{L} << L$.

When the dimension $d$ is large, there are three possible remedies to maintain computational efficiency. First, instead of computing the signature in $M(X)_k = (S(\mathbf{X})_{0,t_k}, S(\widetilde{\mathbf{X}}_{t_{k-1},t_k})$,

one may compute the *log-signature*, which is a compressed version of the signature [220]. When the dimension is large enough such that the log-signature is computationally infeasible, one may instead compute the *univariate* signatures of features coupled with the time channel. That is, consider $\widehat{\mathbf{X}} \in C([0,T];\mathbb{R}^d)$, with $\widehat{\mathbf{X}}(t) = (\widehat{\mathbf{X}}_1(t), ..., \widehat{\mathbf{X}}_d(t))$. Denote the time-added function $\overline{\mathbf{X}}_i(t) := (t, \widehat{\mathbf{X}}_i(t))$. Then we define the *univariate multi-view signature*

$$\widehat{M}(\widehat{\mathbf{X}})_k = \left(M(\overline{\mathbf{X}}_1)_k, ..., M(\overline{\mathbf{X}}_d)_k\right) . \tag{4.12}$$

The attention mechanism in this case is constructed as before. Fixing the maximum signature depth to be some value $n^*$, one sees that the number of features in the univariate multi-view signature is approximately $2^{n^*}d$. In practice we find that $n^* \leqslant 5$ provides sufficient performance, so the order of the attention calculation is $O(C\,\overline{L}^2\,d)$ for $C \leqslant 2^{n^*}$. Finally, one may use randomized signatures to reduce dimension by using a Johnson-Lindenstrauss-type projection to a low-dimensional latent space and computing the signature in this space, as in [73, 70].

## 4.4 Experiments

In this section, we present empirical results for the effectiveness of the Rough Transformer, hereafter denoted `RFormer`, on a variety of time-series-related tasks. We consider long multivariate time-series as our main experimental setting because we expect signatures to perform best in this scenario. Additional experimentation on long-range reasoning tasks on image-based datasets is left for future work, as these would likely require additional inductive biases.

To benchmark `RFormer`, we select both discrete-time and continuous-time models. In particular, we include as main baselines traditional RNN models (`GRU` [64]), ODE-based methods designed for sequential data (`Neural-CDE` [161]), as well as ODE-based methods explicitly designed for long time-series (`Neural-RDE` [199]).[2] Furthermore, we compare against a vanilla `Transformer` [260] with single-headed attention which is the `RFormer` backbone. Finally, we present comparisons with a recent continuous-time Transformer model, `ContiFormer` [61], to highlight the computational efficiency gap between `RFormer` and similar continuous-time models. We note that the first two tasks focus on evaluating the performance improvement of `RFormer` over the `Transformer`

[2]We only benchmark `Neural-CDE` models in settings where time series are of relatively short length, due to the computational demands of this model for longer sequences.

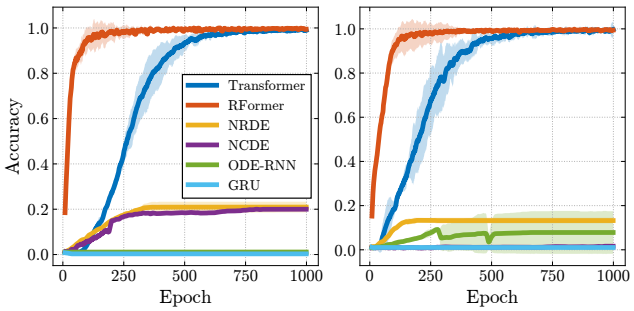


Figure 4.3: Test accuracy per epoch for the frequency classification task across three random seeds. **Left:** Sinusoidal dataset. **Right:** Long Sinusoidal dataset.

baseline. For other long-range tasks, we include comparisons to recent state-space models [121, 208, 247]. In the irregular sampling regime, we benchmark against state-of-the-art models tailored to that setting [205, 236].

### 4.4.1 Time Series Processing

**Frequency Classification.**

Our first experiment is based on a set of synthetically generated time series from continuous paths of the form

$$\widehat{\mathbf{X}}(t) = g(t)\sin(\omega\, t + \nu) + \eta(t)\,, \tag{4.13}$$

where $g(t)$ is a non-linear trend component, $\nu$ and $\eta$ are two noise terms, and $\omega$ is the frequency. Here, the task of the model is to classify the time-series according to its frequency $\omega$. We consider $1000$ samples in $100$ classes with $\omega$ evenly distributed from $10$ to $500$. Each time-series is regularly sampled with $2000$ times-steps on the interval $[0, 1]$. This synthetic experiment is similar to others in recent work on time-series modelling [176, 278, 197]. We include an additional experiment in which we alter the signal in (4.13) so its frequency is $\omega_0$ for $t < t_0$ and $\omega_1$ afterward, where the task is to classify the sinusoid based on the first frequency. We call this dataset the "long sinusoidal" dataset. This extension of the original experiment aims to test the ability of the model to perform long-range reasoning effectively. Note that for this task, we also add `ODE-RNN` [227] to the previously mentioned baselines.

Figure 4.3 shows that the inclusion of both local and global information with the multi-view signature enhances the sample efficiency of the `RFormer` over the vanilla `Transformer` model, even though the attention mechanism is now operating on a much shorter sequence. When compared with other models, we see that `GRU` and `ODE-RNN` fail to capture the information in the signal, and are not able to obtain any meaningful performance improvement throughout the training period. This highlights the shortcomings of most RNN-based models when processing sequences of moderate length, which are very common in real-world

applications. Both `Neural-CDE` and `Neural-RDE` capture some useful dependencies in the time series but fall short compared with both vanilla `Transformer` and `RFormer`.

**HR dataset.** Next, we consider the Heart Rate dataset from the TSR archive [255], originally sourced from Beth Israel Deaconess Medical Center (BIDMC). This dataset consists of time-series sampled from patient ECG readings, and each model is tasked to perform a regression by forecasting the patient's heart rate (HR) at the sample's conclusion. The data, sampled at 125Hz, consists of three-channel time-series (including time), each spanning $4000$ time steps. We used the L2 loss metric to assess the performance. Table 4.1 shows the results, where $\diamond$ denotes the results from [199] and † our reproduction. The sequences in the HR dataset are sufficiently short to remain within memory when running the `Transformer` model. The baseline `Transformer` model improves over `GRU`, and `ODE-RNN`, however, it is less competitive when compared with `Neural-RDE`, suggesting that the Transformer is not particularly well-suited for this type of task. However, the `RFormer` model improves over the baseline `Transformer` by $67\%$. Across all tasks, we see significant improvements in efficiency as a consequence of the signature computation. We elaborate on this in more detail in the following subsection.

Table 4.1: Test RMSE (mean $\pm$ std) computed across five seeds on the Heart Rate (HR) dataset.

| Model | **HR** |
|---|---|
| | RMSE $\downarrow$ |
| ODE-RNN$^\diamond$ | $13.06 \pm 0.00$ |
| Neural-CDE$^\diamond$ | $9.82 \pm 0.34$ |
| Neural-RDE$^\diamond$ | $2.97 \pm 0.45$ |
| GRU$^\dagger$ | $13.06 \pm 0.00$ |
| ODE-RNN$^\dagger$ | $13.06 \pm 0.00$ |
| Neural-RDE$^\dagger$ | $\underline{4.04} \pm 0.11$ |
| Transformer | $8.24 \pm 2.24$ |
| ContiFormer | OOM |
| **RFormer** | $\mathbf{2.66 \pm 0.21}$ |

**Long Time Series Classification.** We now evaluate the performance of `RFormer` on five long time series classification tasks from the UEA time series classification archive [12]. A summary of these datasets is provided in Table 4.6 in Section 4.7.3. As previously done in [199], the original train and test datasets are merged and then randomly divided into new train, validation, and test sets, following a 70/15/15 split. The resulting performance

metrics are summarized in Table 4.2.[3]

In this setting, we see that `RFormer` generally matches or slightly outperforms the continuous-time and SSM baselines. Due to the scaling problems of `ContiFormer` with respect to sequence length, we were unable to run this baseline within GPU memory constraints in most cases, and thus no results are reported (see Section 4.7.6.2 for efficiency comparisons between models). In contrast, `RFormer` can cheaply train on the same device (see Section 4.4.2 for details) due to its ability to take advantage of the parallel nature of GPU processing and compress the original time series. This is especially noticeable when compared to continuous-time models (`Neural-CDE`, `Neural-RDE`, `LogCDE`), which are sometimes orders of magnitude slower than our model and consistently report lower or similar results. Additional experimental details can be found in Section 4.7.6, as well as some experiments on hyperparameter sensitivity.

Table 4.2: Classification performance on various long context temporal datasets from UCR TS archive.

| Dataset | LRU | S5 | S6 | Mamba | NCDE | NRDE | LogNCDE | RFormer |
|---|---|---|---|---|---|---|---|---|
| SCP1 | 82.6 ± 3.4 | **89.9 ± 4.6** | 82.8 ± 2.7 | 80.7 ± 1.4 | 79.8 ± 5.6 | 80.9 ± 2.5 | 83.1 ± 2.8 | 82.6 ± 2.7 |
| SCP2 | 51.2 ± 3.6 | 50.5 ± 2.6 | 49.9 ± 9.5 | 48.2 ± 3.9 | 53.0 ± 2.8 | **53.7 ± 6.9** | **53.7 ± 4.1** | 53.3 ± 7.2 |
| MI | 48.4 ± 5.0 | 47.7 ± 5.5 | 51.3 ± 4.7 | 47.7 ± 4.5 | 49.5 ± 2.8 | 47.0 ± 5.7 | 53.7 ± 5.3 | **59.2 ± 4.4** |
| EW | **87.8 ± 2.8** | 81.1 ± 3.7 | 85.0 ± 16.1 | 70.9 ± 15.8 | 75.0 ± 3.9 | 83.9 ± 7.3 | 85.6 ± 5.1 | 86.8 ± 0.7 |
| ETC | 21.5 ± 2.1 | 24.1 ± 4.3 | 26.4 ± 6.4 | 27.9 ± 4.5 | 29.9 ± 6.5 | 25.3 ± 1.8 | 34.4 ± 6.4 | 31.6 ± 3.4 |
| HB | **78.4 ± 6.7** | 77.7 ± 5.5 | 76.5 ± 8.3 | 76.2 ± 3.8 | 73.9 ± 2.6 | 72.9 ± 4.8 | 75.2 ± 4.6 | 72.7 ± 5.1 |
| Av. | 61.7 | 61.8 | 62.0 | 58.6 | 60.2 | 60.6 | 64.3 | **64.4** |

## 4.4.2 Training Efficiency

Here, we focus on the computational gains of the model when compared with vanilla Transformers and methods that require numerical ODE solvers.

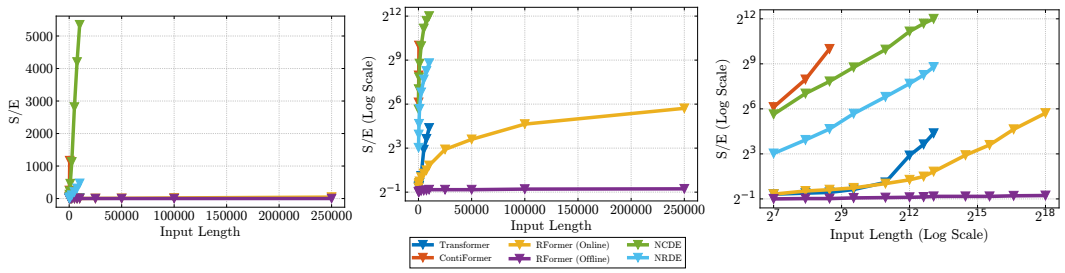


Figure 4.4: Seconds per epoch for growing input length and for different model types on the sinusoidal dataset for extremely long lengths (up to 250k) **Left:** Log Scale. **Middle:** Regular Scale. **Right:** Log-log scale. When a line stops, it indicates an OOM error.

[3]We note that baseline results for this task were taken from [262].

Attention-based architectures are highly parallelizable on modern GPUs, as opposed to traditional RNN models which require sequential updating. However, vanilla attention experiences a bottleneck in memory and time complexity as the sequence length $L$ grows. As covered above in Section 4.3, variations of the signature transform allow the model to operate on a reduced sequence length $\overline{L}$ without increasing the dimensionality in a way that would become problematic for the model. This allows us to bypass the quadratic complexity of the model without resorting to sparsity techniques commonly used in the literature [103, 176].

Table 4.3: Seconds per epoch for all models considered.

| Model | Sec. / Epoch | | |
|---|---|---|---|
| | **Sine** | **EW** | **HR** |
| GRU | **0.12** | <u>0.25</u> | <u>1.07</u> |
| ODE-RNN | 5.39 | 48.59 | 50.71 |
| Neural-CDE | 9.83 | - | - |
| Neural-RDE | 0.85 | 5.23 | 9.52 |
| Transformer | 0.77 | OOM | 11.71 |
| **RFormer** | <u>0.55</u> | **0.11** | **0.45** |
| **Speedup** | $1.4\times$ | - | $26.11\times$ |

Tables 4.1-4.3 show that `RFormer` is competitive when modelling datasets with extremely long sequences without an explosion in the memory requirements. `RFormer` exploits the parallelism of the attention mechanism to significantly accelerate training time, as the length of the input sequence is decreased substantially. In particular, we observe speedups of $\mathbf{1.4}\times$ to $\mathbf{26.11}\times$ with respect to standard attention, and higher when compared with all methods requiring numerical solutions to ODEs. The computational efficiency gains of `RFormer` are attained due to the signature transform reducing the length of the time-series with minimal information loss. The effectiveness of this transformation can be seen from the ablation study carried out in Section 4.7.4. This contrasts with NRDEs [199], which augment NCDEs with local signatures of input data, and find that smaller windows often perform better. Furthermore, NRDEs do not experience the same computational gains as `RFormer` because they must perform many costly ODE integration steps.

Table 4.4: Dataset processing times for training, validation, and testing phases.

| Dataset | **Train** | **Val** | **Test** |
|---|---|---|---|
| Eigenworms | 1.11 s. | 0.19 s. | 0.19 s. |
| HR | 4.23 s. | 0.84 s. | 0.85 s. |
| Sine (1k) | 0.39 s. | 0.39 s. | 0.39 s. |
| Sine (5k) | 0.51 s. | 0.51 s. | 0.51 s. |
| Sine (20k) | 1.64 s. | 1.64 s. | 1.64 s. |
| Sine (100k) | 5.74 s. | 5.74 s. | 5.74 s. |

In Figure 4.2, we showcase the improvements in computational efficiency of `RFormer` compared to vanilla Transformers [260], continuous-time Transformers [61], and other continuous-time RNNs [161, 199] when processing sequences from $L = 100$ samples up to $L = 10$k. As seen, `RFormer` is significantly more efficient than its continuous-time and vanilla counterparts, even when performing the signature computation online, which involves computing the signatures for each batch during training, resulting in significant redundant computation. When signatures are precomputed just once before training, the computational time of each epoch remains *constant* across input all sequence lengths including $L = 10$k (see the exact signature computation times for different datasets in Table 4.4). We also stress the fact that `RFormer` also scales gracefully for extremely long sequences (up to $L = 250$k) with both online and offline computation of the signatures, as shown in Figure 4.4. Finally, we highlight that `ContiFormer` has a sample complexity of $\mathcal{O}(L^2d^2S)$, where $S$ represents the normalized number of function evaluations of the numerical ODE solver, which makes `ContiFormer` orders of magnitude more computationally intensive when compared to `RFormer` and prevents the model from running on sequences longer than 500 points due to out-of-memory errors.

### 4.4.3 Irregular Time Series Classification

So far, we mainly focused on the efficiency and inductive bias afforded to the model through the use of signatures. However, a key element of `RFormer` is that it can naturally deal with irregularly sampled sequences without expensive numerical ODE solvers. This property follows from the fact that signatures are *invariant to time reparameterization*, see Proposition 4.3.1. In this subsection, we empirically test this property by training the model on the same datasets but randomly dropping 50% of the points at every epoch. This test intends to find if the model is able to learn continuous-time representations of the original input

time-series.

The results can be found in Table 4.5. We find that `RFormer` consistently results in the best performance, with a small performance drop when compared to the full dataset. Importantly, this property is achieved in conjunction with the efficiency gains afforded to the model and without the use of expensive numerical ODE solvers.[4] Finally, we perform an additional set of experiments on the 15 univariate classification datasets from the UEA time series classification archive and compare our model with recent state-of-the-art models for irregular time series [205, 236]. Across the board, we find that our model is both faster and more accurate than the continuous-time benchmark *despite having a discrete-time Transformer backbone*. For more details and more exhaustive experimentation on random data drops, see Section 4.7.6.

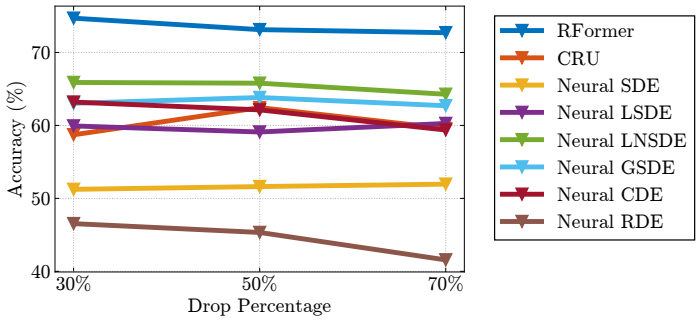


Figure 4.5: Average performance of all models on the 15 univariate datasets from the UEA Time Series archive under different degrees of data drop.

Table 4.5: Performance of all models under a random 50% drop in datapoints per epoch.

| Model | **50% Drop Performance** | | | |
|---|---|---|---|---|
| | EW (%) ↑ | HR ↓ | Sine (%) ↑ | Sine Long (%) ↑ |
| GRU | 35.90 | 13.06 | 0.96 | 1.16 |
| ODE-RNN | 37.61 | 13.06 | 1.06 | 1.23 |
| Neural-RDE | <u>60.68</u> | <u>4.67</u> | 0.94 | 0.87 |
| Transformer | OOM | 12.73 | <u>7.37</u> | <u>20.23</u> |
| **RFormer** | **87.69** | **2.96** | **59.57** | **93.17** |

[4]We use our own reproduction to test the performance of all models in irregularly sampled datasets. Random dropping requires window sizes larger than 2 because signatures cannot be computed over a single point.

## 4.5 Reasons for improved model performance

In this final section, we provide explanations for the superior inductive bias of the `RFormer` model compared to its vanilla Transformer counterpart, despite its lower computational cost.

### 4.5.1 Spatial Processing

First, we highlight that a key reason the model achieves significant compression benefits in the tasks considered is its ability to *jointly* account for temporal and spatial interactions through the self-attention mechanism and signature terms, respectively. In particular, we believe that for certain datasets, the relationships between different channels of the time series may hold more importance than the temporal information itself, which can often be redundant. This is exemplified in the `Eigenworms` dataset, which experiences a 20% performance drop when employing univariate signatures, but is able to achieve state-of-the-art performance with a 600$\times$ compression rate in the temporal dimension when signatures are applied across all channels, as shown in Figure 4.7. To this end, we draw parallels between the use of signatures and the field of temporal graph processing, where the use of the signature over all channels can be seen as a fully connected graph, capturing information from all channels, and the univariate signature would correspond to a graph with only self-connections between the nodes, as depicted in Figure 4.6. In our view, this hints towards the idea of using sparse graph learning techniques [67, 78] to reduce the explosion in signature terms while retaining the ability to perform effective spatial processing.

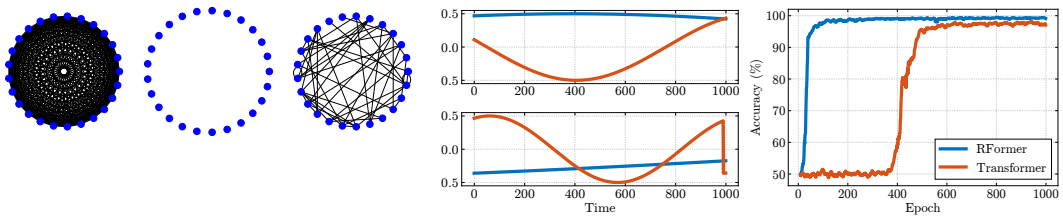


Figure 4.6: **Left:** Graph connectivity structures for multivariate, univariate and sparse signature. **Middle:** Example samples for synthetic task. **Right:** Performance on spatial synthetic experiment.

To empirically test these claims, we design a synthetic experiment using a 2-channel time series. Each channel contains a signal of the form $\sin(\omega_i t + \nu_i), i = 1, 2$, where $\omega_i$ and $\nu_i$ are randomly sampled from the interval $[0, 2\pi]$. For half of the dataset, the last 1% of temporal samples in the second channel are set to match the frequency of the first channel. The task is to classify whether the samples in this final interval are of the same

frequency. As shown in Figure 4.6, `RFormer` demonstrates greater sample efficiency and achieves higher test accuracy compared to its vanilla Transformer counterpart, highlighting the effectiveness of signatures in spatial processing.

### 4.5.2 Sequence Coarsening as an Inductive Bias for Transformers

In addition to the benefits of higher-order signature terms, we empirically observe that even using level-one signature terms resulted in performance improvements when compared to processing sequences without any transformation. We believe that the reduction in input signal length, achieved without significant information loss through the signature transform is another important factor in the improved inductive bias of `RFormer`. This finding aligns with the concurrent work of [16], which highlights some of the drawbacks of decoder-only Transformers for long sequences in terms of both *oversquashing* and *representational collapse*.

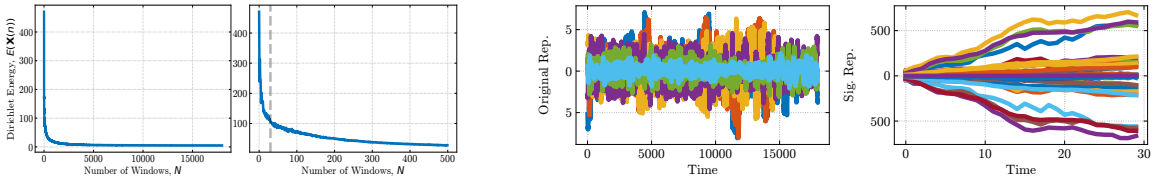


Figure 4.7: **Left:** Dirichlet energy as a function of window size for the Eigenworms dataset. **Right:** Original and hidden representation after signature layer for two examples in the EW dataset.

To measure the degree of coarsening in the sequence, we find that interpreting the temporal sequence as a path graph and using ideas from the oversmoothing literature [228] serves as a good way to measure how similar the representations being fed to the Transformer are. In particular, we compute the unnormalized Dirichlet Energy, defined in this case as $E(\mathbf{X}) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{X}_i - \mathbf{X}_{i-1}||_2$ of the temporal sequence resulting from taking increasing window sizes of the global signature. An example of this is shown in Figure 4.7 for the `Eigenworms` dataset, where we compared different numbers of windows (from 2 to 18k). Interestingly, we found that the "elbow" of the Dirichlet energy corresponded to 30 windows in this dataset, which we found empirically to be one of the the most performant settings. This hints at the idea of the Dirichlet energy being used for signature hyperparameter tuning as well.

## 4.6 Conclusion

We introduced the Rough Transformer, a variant of the original Transformer that allows the processing of discrete-time series as continuous-time signals through the use of multi-view signature attention. Empirical comparisons showed that Rough Transformers outperform vanilla Transformers and continuous-time models on a variety of time-series tasks and are robust to the sampling rate of the signal. Finally, we showed that RFormer provides significant speedups in training time compared to regular attention and ODE-based methods, without the need for major architectural modifications or sparsity constraints.

## 4.7 Additional Materials

### 4.7.1 Properties of Path Signatures

First, we recall that the path is uniquely determined by its signature, which motivates its use as a feature map.

**Proposition 4.7.1.** *Given a path $\widehat{\mathbf{X}} : [0, T] \to \mathbb{R}^d$, then the map $P : [0, T] \to \mathbb{R}^{1+d}$ where $P(t) = (t, \widehat{\mathbf{X}}(t))$ is uniquely determined by it's signature $S(P)_{0,T}$.*

The proof can be found in [128].

For Rough Transformers, several features of path signatures are important. First, linear functionals on path signatures possess universal approximation properties for continuous functionals.

**Theorem 4.7.2.** *Fix $T > 0$, and let $K \subset C_b^1([0, T]; \mathbb{R}^d)$. Let $f : K \to \mathbb{R}$ be continuous with respect to the sup-norm topology on $C_b^1([0, T]; \mathbb{R}^d)$. Then for any $\varepsilon > 0$, there exists a linear functional $\ell$ such that*

$$|f(\overline{\mathbf{X}}) - \langle \ell, S(\overline{\mathbf{X}})_{0,T} \rangle| \leqslant \varepsilon \,, \tag{4.14}$$

*for any $\widehat{\mathbf{X}} \in K$, where $\overline{\mathbf{X}}$ denotes the time-added augmentation of $\widehat{\mathbf{X}}$.*

For a proof of 4.7.2, see [7]. Even though Theorem 4.7.2 guarantees that *linear* functionals are sufficient for universal approximation, linear models are not always sufficient in practice. This motivates the development of nonlinear models built upon the path signature which efficiently extract path behavior.

The second feature is that the terms of the path signature decay factorially, as described by the following proposition.

**Proposition 4.7.3.** *Given $\widehat{\mathbf{X}} \in C_b^1([0,T];\mathbb{R}^d)$, for any $s,t \in [0,T]$, we have that for any $I \in \mathcal{I}_d^n$*

$$|S(\widehat{\mathbf{X}})_{0,T}^I| = O\left(1/n!\right) . \tag{4.15}$$

For a proof of Proposition 4.7.3, see [183]. Hence, the number of terms in the signature grows exponentially in the level of the signature, but the tail of the signature is well-behaved, so only a few levels in a truncated signature are necessary to adequately approximate continuous functionals.

### 4.7.2 Signatures of Piecewise Linear Paths.

In the Rough Transformer, we use linear interpolation of input time-series to get a continuous-time representation of the data. As mentioned in Section 4.3, the signature computation in this case is particularly simple.

Suppose $\widehat{\mathbf{X}}_k : [t_k, t_{k+1}] \to \mathbb{R}^d$ is a linear interpolation between two points $\mathbf{X}_k, \mathbf{X}_{k+1} \in \mathbb{R}^d$. That is,

$$\widehat{\mathbf{X}}_k(t) = \mathbf{X}_k + \frac{t - t_k}{t_{k+1} - t_k} \left(\mathbf{X}_{k+1} - \mathbf{X}_k\right) . \tag{4.16}$$

Then the signature of $\widehat{\mathbf{X}}_k$ is given explicitly by

$$S(\widehat{\mathbf{X}}_k)_{t_k,t_{k+1}} = \left(1, \mathbf{X}_{k+1} - \mathbf{X}_k, \frac{1}{2}(\mathbf{X}_{k+1} - \mathbf{X}_k)^{\otimes 2}, \frac{1}{3!}(\mathbf{X}_{k+1} - \mathbf{X}_k)^{\otimes 3}, ..., \frac{1}{n!}(\mathbf{X}_{k+1} - \mathbf{X}_k)^{\otimes n}, ...\right) , \tag{4.17}$$

where $\otimes$ denotes the tensor product. Let $\widehat{\mathbf{X}}_k * \widehat{\mathbf{X}}_{k+1}$ denote the *concatenation* of $\widehat{\mathbf{X}}_k$ and $\widehat{\mathbf{X}}_{k+1}$. That is, $\widehat{\mathbf{X}}_k * \widehat{\mathbf{X}}_{k+1} : [t_k, t_{k+2}] \to \mathbb{R}^d$ is given by

$$\widehat{\mathbf{X}}_k * \widehat{\mathbf{X}}_{k+1}(t) = \begin{cases} \widehat{\mathbf{X}}_k(t) & t \in [t_k, t_{k+1}] \\ \widehat{\mathbf{X}}_{k+1}(t) & t \in (t_2, t_{k+2}] . \end{cases} \tag{4.18}$$

The signature of the concatenation $\widehat{\mathbf{X}}_k * \widehat{\mathbf{X}}_{k+1}$ is given by *Chen's relation*, whose proof is in [183].

**Proposition 4.7.4** (Chen's Relation)**.** *The following identity holds:*

$$S(\widehat{\mathbf{X}}_k * \widehat{\mathbf{X}}_{k+1})_{t_k,t_{k+2}} = S(\widehat{\mathbf{X}}_k)_{t_k,t_{k+1}} \otimes S(\widehat{\mathbf{X}}_{k+1})_{t_{k+1},t_{k+2}} , \tag{4.19}$$

*where for elements $A, B \in T((\mathbb{R}^d))$ with $A = (A_0, A_1, A_2, ...)$ and $B = (B_0, B_1, B_2, ...)$ the tensor product $\otimes$ is defined*

$$A \otimes B = \left(\sum_{j=0}^{k} A_j \otimes B_{k-j}\right)_{k \geqslant 0} . \tag{4.20}$$

Let $\mathbf{X} = (\mathbf{X}_0, ..., \mathbf{X}_L)$ be a time-series. Then the linear interpolation $\widetilde{\mathbf{X}} : [0, T] \to \mathbb{R}^d$ can be represented as the concatenation of a finite number of linear paths:

$$\widetilde{\mathbf{X}} = \widehat{\mathbf{X}}_0 * \cdots * \widehat{\mathbf{X}}_{L-1} \,. \tag{4.21}$$

Hence, the signature is

$$S(\widetilde{\mathbf{X}})_{0,T} = S(\widehat{\mathbf{X}}_0)_{0,t_1} \otimes \cdots \otimes S(\widehat{\mathbf{X}}_{L-1})_{t_{L-1},T} \,. \tag{4.22}$$

### 4.7.3 Long Temporal Datasets Details

Table 4.6 summarises the long temporal modeling datasets from the UEA time series classification archive [12] used in Section 4.4.

Table 4.6: Summary of datasets used in the long time-series classification task.

| **Dataset** | **#Sequences** | **Length** | **#Classes** | **#Dimensions** |
|---|---|---|---|---|
| SelfRegulationSCP1 (SCP1) | 561 | 896 | 2 | 6 |
| SelfRegulationSCP2 (SCP2) | 380 | 1152 | 2 | 7 |
| MotorImagery (MI) | 378 | 3000 | 2 | 64 |
| EigenWorms (EW) | 259 | 17984 | 5 | 6 |
| EthanolConcentration (ETC) | 524 | 1751 | 4 | 3 |

To prevent excessive growth in signature terms, we sometimes use datasets. As an alternative, one could employ randomized signatures [70] or low-rank approximations [48, 57, 91].

### 4.7.4 Ablation Studies

#### 4.7.4.1 Global and Local Signature Components

In this section, we ablate the use of the multi-view signature transform over both global and local transformations of the input signal. The results for the sinusoidal datasets are shown in Figure 4.8. In most cases, the use of both local and global components improves the performance of `RFormer`. This choice, however, can be seen as a hyperparameter and will be dataset-dependent.

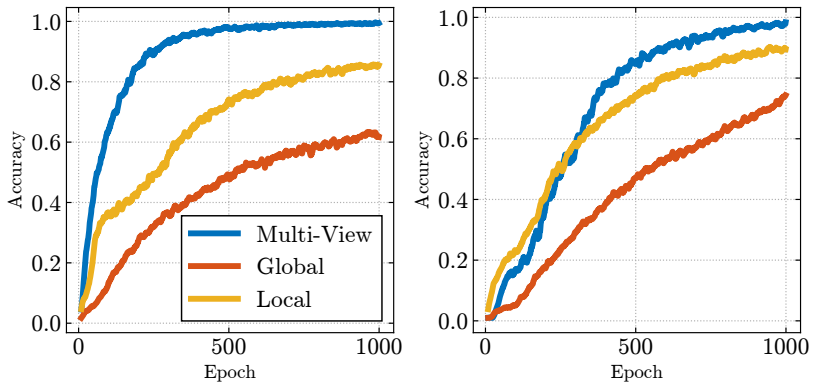


Figure 4.8: Ablation of local and local components of the multi-view signature for the sinusoidal datasets.

### 4.7.5 Signature Level and Naive Downsampling

One of the main points of the chapter is that the shorter representation of the time-series endowed by the signatures helps to significantly reduce the computational cost of the self-attention operation with minimal information loss (and with improved performance in many of the experiments). By equation (4.17), one sees that the first level of the signature of a linear function is the difference between its endpoints. Hence, using multi-view attention with signature level one operates on the increments of piecewise-linear interpolated data, which corresponds to naive downsampling. To test that higher levels of the signature provide improvements in performance, we compare the result of using the signature on the datasets tested in Table 4.7 below.

Table 4.7: Comparative performance of different methods on datasets.

| Dataset | Linear-Interpolation + Vanilla | Rough Transformer with sig level (n) | Improvement |
|---|---|---|---|
| EigenWorms | 64.10% | 90.24% (2) | 40.77% |
| HR | 10.56 | 2.66 (4) | 74.81% |

There is a significant performance gain in considering higher levels of the signature because one can capture the higher-order interactions between the different time-series.

### 4.7.6 Additional Experiments and Comparisons

#### 4.7.6.1 Random Drop Experiments

Furthermore, we conduct a new set of experiments in which we dropped 30% and 70% of the dataset for RFormer. Note that even with a 70% drop rate in the EigenWorms dataset,

the vanilla Transformer fails to run due to memory limitations. Therefore, to provide results for the Transformer model on the EigenWorms dataset, we conduct experiments with an 85% drop rate. This comparison highlights the performance gap between the vanilla Transformer and our proposed model under these conditions, with the RFormer model yielding superior results. All results are computed across five seeds and are summarized in the tables and figure below.

Table 4.8: Performance of models under various data drop scenarios for EW dataset.

| Model | Full | 30% Drop | 50% Drop | 70% Drop | 85% Drop |
|---|---|---|---|---|---|
| Transformer | OOM | OOM | OOM | OOM | 72.45% $\pm$ 3.36 |
| RFormer | 90.24% $\pm$ 2.15 | 87.86% $\pm$ 3.28 | 87.69% $\pm$ 4.97 | 83.35% $\pm$ 2.86 | 82.74% $\pm$ 2.13 |

Table 4.9: Performance consistency of RFormer under data drop scenarios for HR dataset.

| Model | Full | 30% Drop | 50% Drop | 70% Drop |
|---|---|---|---|---|
| RFormer | 2.66 $\pm$ 0.21 | 2.72 $\pm$ 0.19 | 2.82 $\pm$ 0.05 | 2.98 $\pm$ 0.08 |

Table 4.10: Epoch-wise performance under different data drop scenarios for the sinusoidal dataset.

| | Epoch 100 | Epoch 250 | Epoch 500 | Epoch 1000 |
|---|---|---|---|---|
| 30% Drop | 48.6% | 82.5% | 91.4% | 99.3% |
| 70% Drop | 35.7% | 56.8% | 64.9% | 67.8% |

Table 4.11: Epoch-wise performance under different data drop scenarios for the long sinusoidal dataset.

| | Epoch 100 | Epoch 250 | Epoch 500 | Epoch 1000 |
|---|---|---|---|---|
| 30% Drop | 39.1% | 72.6% | 96.2% | 98.2% |
| 70% Drop | 27.5% | 66.7% | 78.5% | 85.3% |

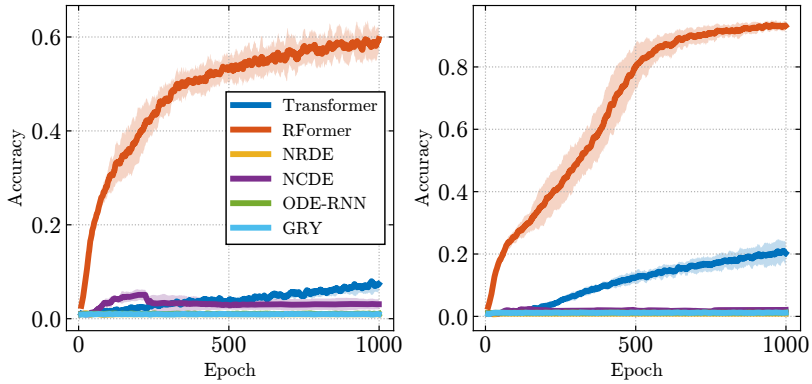


Figure 4.9: Test accuracy per epoch for the frequency classification task across three random seeds for sinusoidal datasets with 50% random drop per epoch. **Left:** Sinusoidal dataset. **Right:** Long Sinusoidal dataset.

Finally, Table 4.12 compares CRU and RFormer in an irregularly sampled synthetic data setting, featuring shorter sinusoids and fewer classes than the experiments in Section 4.4.1. Additionally, Table 4.13 presents the hyperparameter validation for CRU (see Table 4.14 for training time analysis). These experiments demonstrate that recurrent models perform well with short sequences. Note that despite RFormer's superior performance, our model is significantly faster than other continuous-time models, as shown in Section 4.7.6.2, particularly in Table 4.14.

Table 4.12: Comparison of RFormer and CRU (two best and simplest performing instances [Num.basis/Bandwidth$= 20/3$]) at different random drop percentages.

| L | Random Drop | RFormer | CRU (LSD=10) | CRU (LSD=20) |
|---|---|---|---|---|
| 100 | 0% | 100.00% | 100% | 100.00% |
| | 30% | 98.60% | 65.90% | 99.60% |
| | 50% | 97.80% | 34.40% | 94.70% |
| | 70% | 96.10% | 43.00% | 78.60% |
| | 85% | 85.50% | 32.30% | 57.30% |
| 250 | 0% | 100.00% | 100.00% | 100% |
| | 30% | 99.90% | 42.95% | 94.90% |
| | 50% | 99.40% | 43.65% | 77.30% |
| | 70% | 98.30% | 45.40% | 94.40% |
| | 85% | 86.20% | 38.80% | 83.60% |
| 500 | 0% | 100.00% | 100.00% | OOM |
| | 30% | 99.90% | 47.15% | OOM |
| | 50% | 99.70% | 48.80% | OOM |
| | 70% | 99.30% | 55.15% | OOM |
| | 85% | 87.70% | 46.50% | OOM |

Table 4.13: CRU's hyperparameters ($L = 100$) (latent state dimension (LSD), number of basis matrices (Num.basis), and their bandwidth).

| LSD | Num. basis | Bandwidth | Acc (30 Epochs) |
|---|---|---|---|
| 10 | 15 | 3 | 78% |
| | 15 | 10 | - |
| | 20 | 3 | 100% |
| | 20 | 10 | - |
| 20 | 15 | 3 | 81.30% |
| | 15 | 10 | 91.70% |
| | 20 | 3 | 100% |
| | 20 | 10 | 99.90% |
| 40 | 15 | 3 | 99.90% |
| | 15 | 10 | 97.50% |
| | 20 | 3 | 100% |
| | 20 | 10 | 100% |

#### 4.7.6.2 Additional Efficiency Experiments and Discussion

We conduct additional experiments to compare the runtime of Rough Transformers with other models. In this experiment, we use the synthetic sinusoidal dataset considered in our work and compute the runtime per epoch for varying sequence lengths. We demonstrate results for two variants of RFormer: "online", which corresponds to computing the signatures of each batch during training (resulting in significant redundant computation), and "offline", which corresponds to computing the signatures in one go at the beginning of training. We include a recent RNN-based model as a basis for comparison with high-performing RNN baselines. In addition to the models discussed in Section 4.4, we introduce Continuous Recurrent Units (CRU) [236] as a new baseline. See Table 4.14 for a summary of the results.

Table 4.14: Seconds per epoch for growing input length and for different model types on the sinusoidal dataset.

| Model | S/E for Varying Context Length ↓ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **L=100** | **L=250** | **L=500** | **L=1000** | **L=2500** | **L=5000** | **L=7.5k** | **L=10k** |
| NRDE | 5.87 | 11.67 | 20.27 | 44.01 | 103.11 | 201.21 | 312.31 | 467.47 |
| NCDE | 42.59 | 121.82 | 225.14 | 458.09 | 1126.77 | 2813.42 | 4199.50 | 5345.39 |
| GRU | 1.56 | 1.55 | 1.65 | 1.63 | 1.78 | 2.37 | 3.65 | 4.79 |
| CRU | 59.22 | 199.15 | 789.28 | OOM | OOM | OOM | OOM | OOM |
| ContiFormer | 61.36 | 248.31 | 1165.02 | OOM | OOM | OOM | OOM | OOM |
| Transformer | 0.75 | 0.79 | 0.82 | 0.95 | 1.36 | 5.31 | 9.32 | 16.32 |
| RFormer (Online) | 0.75 | 0.88 | 0.94 | 1.03 | 1.28 | 1.55 | 1.83 | 2.35 |
| RFormer (Offline) | 0.67 | 0.64 | 0.63 | 0.65 | 0.60 | 0.59 | 0.62 | 0.60 |

We remark that previous running times are obtained with a batch size of 10. Further, the `ContiFormer` model could be run for $L = 1000$ if decreasing the batch size to 2 (which significantly affects the parallelization process), avoiding OOM issues and resulting in 4025 seconds/epoch, which is several orders of magnitude larger than `RFormer`. As an additional experiment, we tested the epoch time (S/E) of `RFormer` for extremely oversampled sinusoidal time series. We show our results in the table below.

Table 4.15: Seconds per epoch for very large input length.

| Model | S/E for Varying Context Length ↓ | | | |
|---|---|---|---|---|
| | **L=25k** | **L=50k** | **L=100k** | **L=250k** |
| RFormer (Online) | 5.39 | 9.06 | 19.95 | 45.20 |
| RFormer (Offline) | 0.60 | 0.61 | 0.60 | 0.63 |

Thus, the time needed to compute the signature is inconsequential when compared with the time required to train standard models on the full or even downsampled datasets, since this step has to be carried out only once. To put this into context with an example, we note that it takes 4s to compute the signature representations for the HR dataset (which is about half the time it takes for the Vanilla Transformer to go through one epoch) and results in a $26\times$ increase in computational speed for RFormer when compared to the vanilla Transformer.

To showcase that this is the case for not only sequences of moderate length but also extremely long sequences, we also carry out the following experiment where we compute the

signature representation for the sine dataset, with a progressively increasing number of datapoints. As seen in the following table, this does not cause an explosion in computational time:

Table 4.16: Processing times for different sizes on the sinusoidal dataset.

| **Size** | **100** | **250** | **500** | **1k** | **2.5k** | **5k** | **7.5k** | **10k** | **25k** | **50k** | **75k** | **100k** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Time** | 0.15 s | 0.21 s | 0.24 s | 0.39 s | 0.42 s | 0.51 s | 0.70 s | 1.09 s | 1.64 s | 2.94 s | 4.49 s | 5.74 s |

#### 4.7.6.3 Additional ContiFormer Comparisons

Also, to provide some context of the performance of `ContiFormer` compared with our method (and not only results on complexity and training times), we run the model on the sinusoidal classification task for signals of length $L = 100$ and $L = 250$. Due to the slow running time of the `ContiFormer` model, we did not consider sequence lengths of $L > 250$. We evaluate the `ContiFormer` model using one head. However, given the subpar results we obtain, we also test it with four heads, using the hyperparameters originally used in the paper for their irregularly sampled time series classification experiments. By contrast, all variations of `RFormer` tested in this work employ only one head, but reported significantly better results.

Table 4.17: Model performance for $L = 100$.

| Model | Epoch 100 | Epoch 250 | Epoch 500 |
|---|---|---|---|
| ContiFormer (1 Head) | 2.3% | 2.8% | 3.1% |
| ContiFormer (4 Heads) | 8.5% | 17.3% | 20.0% |
| Transformer (1 Head) | 13.7% | 40.1% | 82.8% |
| RFormer (1 Head) | 38.7% | 81.1% | 92.3% |

# Part II

# Learning from Geometric Data

# Chapter 5

# Vanishing Gradients in GNNs: Bridging Recurrent and Graph Learning

Graph Neural Networks (GNNs) are models that leverage the graph structure to transmit information between nodes, typically through the message-passing operation. While widely successful, this approach is well-known to suffer from the over-smoothing and over-squashing phenomena, which result in representational collapse as the number of layers increases and insensitivity to the information contained at distant and poorly connected nodes, respectively. In this paper, we present a unified view of these problems through the lens of *vanishing gradients*, using ideas from linear control theory for our analysis. We propose an interpretation of GNNs as recurrent models and empirically demonstrate that a simple state-space formulation of an GNN effectively alleviates over-smoothing and over-squashing *at no extra trainable parameter cost*. Further, we show theoretically and empirically that (i) GNNs are by design prone to extreme gradient vanishing even after few layers; (ii) Over-smoothing is directly related to the mechanism causing vanishing gradients; (iii) Over-squashing is most easily alleviated by a combination of graph rewiring and vanishing gradient mitigation. We believe our work will help bridge the gap between the recurrent and graph neural network literature and will unlock the design of new deep and performant GNNs.

## 5.1 Introduction

Graph Neural Networks (GNNs) [249, 117, 235, 36, 79] have become a widely used architecture for processing information on graph domains. Most GNNs operate via *mes-*

*sage passing*, where information is exchanged between neighboring nodes, giving rise to Message-Passing Neural Networks (MPNNs). Some of the most popular instances of this type of architecture include GCN [162], GAT [261], GIN [274], and GraphSAGE [129].

Despite its widespread use, this paradigm also suffers from some fundamental limitations. Most importantly, we highlight the issue of *over-smoothing* [203, 37, 228], where feature representations become exponentially similar as the number of layers increases, and *over-squashing* [3, 257, 81], which describes the difficulty of propagating information across faraway nodes, as the exponential growth in a node's receptive field results in many messages being compressed into fixed-size vectors. Although these two issues have been studied extensively, and there exists evidence that they are trade-offs of each other [112], there is no unified theoretical framework that explains *why architectures which solve these problems work* and whether there exists a common *underlying cause* that governs these problems.

In this work, we analyze over-smoothing and over-squashing from the lens of **vanishing gradients**. In particular, we ask several questions about the appearance and consequences of this phenomenon in GNNs: (i) How prone are GNNs to gradient vanishing? (ii) What is the effect of gradient vanishing on over-smoothing? (iii) Can preventing vanishing gradients effectively mitigate over-squashing? (iv) Can methods used in the non-linear [143, 210, 20] and more recently linear [122, 209] recurrent neural network (RNN) literature be effective at dealing with over-smoothing and over-squashing? With the latter questions, we aim to fill the gaps and open questions left in the over-squashing analysis of [81]. Further, we hope the point on over-smoothing will help provide a theoretical explanation for why certain architectural choices have led to the design of deep and performant GNNs.

**Contributions and outline.** In summary, the contributions of this chapter are the following:

- In Section 5.2, we explore the connection between GNNs and sequence models and demonstrate how graph convolutional and attentional models are susceptible to a phenomenon we term *extreme gradient vanishing*. We propose GNN-SSM, a GNN model that is written as a state-space model, allowing for a better control of the spectrum of the Jacobian.

- In Section 5.3, we show how vanishing gradients contribute to node feature evolution, providing a more precise explanation for why GNNs struggle with depth and how node feature collapse emerges via spectral analysis of the layer-wise Jacobians. We show that GNN-SSMs are able to *exactly* control the rate of collapse of deep representations.

- In Section 5.4, we show how vanishing gradients are related to over-squashing. We argue that over-squashing should therefore be tackled by approaches that both perform graph rewiring *and* mitigate vanishing gradients.

Overall, we believe that our work provides a new and interesting perspective on well-known problems that occur in GNNs, from the point of view of sequence models. We believe this to be an important observation connecting two very wide – yet surprisingly disjoint – bodies of literature.

### 5.1.1 Over-smoothing, Over-squashing, and Vanishing Gradients in GNNs

**Over-smoothing.** Over-smoothing [37, 206, 228] describes the tendency of GNNs to produce *smoother* representations as more and more layers are added. In particular, this has been related to the convergence to the 1-dimensional kernel of the graph Laplacian and equivalently as a minimization process of the Dirichlet energy [82]. In Section 5.3, we study this issue from the lens of vanishing gradients and show that **over-smoothing has a much more simple explanation**: it occurs due to the norm-contracting nature of GNN updates.

**Over-squashing.** Over-squashing [3, 257, 81, 16] was originally introduced as a *bottleneck* resulting from 'squashing' into node representations amounts of information that are growing potentially exponentially quickly due to the topology of the graph. It is often characterized by the quantity $\left\|\partial \mathbf{h}_u^{(K)} / \partial \mathbf{h}_v^{(0)}\right\|$ being low, implying that the final representation of node $u$ is not very sensitive to the initial representation at some other node $v$. While the relationship between over-squashing and vanishing gradients was hinted at by [81], in Section 5.4 we explore this relationship in detail by showing that **techniques aimed to mitigate vanishing gradients in sequence models help to mitigate over-squashing in GNNs**.

**Vanishing gradients.** Vanishing gradients have been extensively studied in RNNs [22, 143, 210], while this problem has been surprisingly mostly overlooked in the GNN community. For a detailed discussion on the relevant literature, we point the reader to the Section 5.10. We simply highlight that there are works that have seen success in taking ideas from sequence modelling [229, 119, 230, 264, 21, 158] or signal propagation [99, 238] and bridging them to GNNs, but they rarely have a detailed discussion on vanishing gradients. In Section 5.4, we show that **vanishing gradient mitigation techniques from RNNs seem to be very effective towards the mitigation of over-smoothing and over-squashing in GNNs** and argue that the two communities have at a fundamental level very aligned problems and goals.

## 5.2 Connecting Sequence and Graph Learning through State-Space Models

In this section, we study GNNs from a sequence model perspective. We show that the most common classes of GNNs are more prone to vanishing gradients than feedforward or recurrent networks due to the spectral contractive nature of the normalized adjacency matrix. We then propose GNN-SSMs, a state-space-model-inspired construction of a GNN that allows more direct control of the spectrum.

### 5.2.1 Similarities and differences between learning on sequences and graphs

The GNN architectures that first popularized deep learning on graphs [36, 79] were initially presented as a generalization of Convolutional Neural Networks (CNNs) to irregular domains. GCNs [162] subsequently restricted the architecture in [79] to a one-hop neighborhood. While this is still termed "convolutional" (due to weight sharing across nodes), the iterative process of aggregating information from each node's neighborhood can also be viewed as *recurrent-like* state updates.

If we consider an RNN unrolled over time, it forms a directed path graph feeding into a state node with a self-connection— making it a special case of a GNN. Conversely, node representations in GNNs can be stacked using matrix vectorization, allowing us to interpret GNN layer operations as iterative state updates. This connection suggests that the main difficulty faced by RNNs — namely the vanishing and exploding gradients problem [210] — may likewise hinder the learning ability of GNNs. We note, however, that one

key difference between RNNs and GNNs is that RNN memory *only* depends on how much information is dissipated by the model during the hidden state update, whereas GNNs normalize messages by the inverse node degree, which introduces an additional information dissipation step that we will explore in more detail in Section 5.4.

### 5.2.2 Graph convolutional and attentional models are prone to extreme gradient vanishing

Based on the previously introduced notion of stacking node representations using the matrix vectorization operation, we now analyze the gradient dynamics of GNN. In particular, we focus on the gradient propagation capabilities of graph convolutional and attentional models at initialization, given their widespread use in the literature. Specifically, we demonstrate that the singular values of the layer-wise Jacobian in these models form a highly contractive mapping, which prevents effective information propagation beyond a few layers. We formalize this claim in Lemma 5.2.1 and Theorem 5.2.2, and we refer the reader to Section 5.6.1 for the corresponding proofs.

**Lemma 5.2.1** (Spectrum of the Jacobian's singular values)**.** *Let* $\mathbf{H}^{(k)} = \widetilde{\mathbf{A}}\,\mathbf{H}^{(k-1)}\,\mathbf{W}$ *be a linear GCN layer, where* $\widetilde{\mathbf{A}}$ *has eigenvalues* $\{\lambda_1, \ldots, \lambda_n\}$ *and* $\mathbf{W}\,\mathbf{W}^T$ *has eigenvalues* $\{\mu_1, \ldots, \mu_{d_k}\}$*. Consider the layer-wise Jacobian* $\mathbf{J} = \partial \operatorname{vec}\big(\mathbf{H}^{(k)}\big) / \partial \operatorname{vec}\big(\mathbf{H}^{(k-1)}\big)$*, Then the squared singular values of* $\mathbf{J}$ *are given by the set*

$$\big\{\, \lambda_i^2\, \mu_j \;\;\big|\;\; i = 1, \ldots, n,\;\; j = 1, \ldots, d_k \big\}.$$

**Theorem 5.2.2** (Jacobian singular-value distribution)**.** *Assume the setting of Lemma 5.2.1, and let* $\mathbf{W} \in \mathbb{R}^{d_{k-1} \times d_k}$ *be initialized with i.i.d.* $\mathcal{N}(0, \sigma^2)$ *entries. Denote the squared singular values of the Jacobian by* $\gamma_{i,j}$*. Then, for sufficiently large* $d_k$ *the empirical eigenvalue distribution of* $\mathbf{W}\mathbf{W}^T$ *converges to the Marchenko-Pastur distribution. Then, the mean and variance of each* $\gamma_{i,j}$ *are*

$$\mathbb{E}\big[\gamma_{i,j}\big] = \lambda_i^2\, \sigma^2, \tag{5.1}$$

$$\operatorname{Var}\big[\gamma_{i,j}\big] = \lambda_i^4\, \sigma^4\, \frac{d_k}{d_{k-1}}. \tag{5.2}$$

Theorem 5.2.2 shows that the singular-value spectrum of the Jacobian is modulated by the squared spectrum of the normalized adjacency. Since $|\lambda_i| < 1$ for all eigenvalues of the normalized adjacency, the ability of GCNs to propagate gradients is in expectation worse than that of RNNs or MLPs. In particular, iterating these operations causes the majority

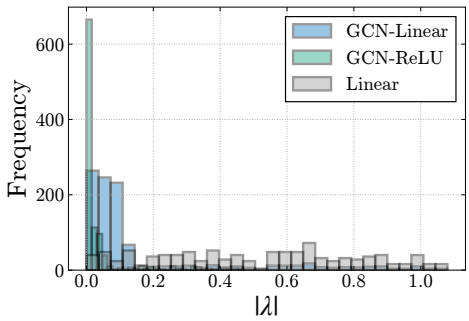


Figure 5.1: Histogram of eigenvalue modulus of the Jacobian for linear, linear convolutional, and nonlinear convolutional layers.

of the spectrum to shrink to zero more quickly than in classical deep linear [234] or nonlinear [212] networks. Moreover, using sigmoidal activations and orthogonal weights will not push the singular-value spectrum to the edge of chaos as in [212], due to the additional contraction from the adjacency. The effect of each operation on the layer-wise Jacobian is empirically demonstrated in Figure 5.1, which also showcases the contraction effect of the normalized adjacency. The figure reveals that even a single layer's Jacobian exhibits a long tail of squared singular values near zero. This spectral structure leads to ill-conditioned gradient propagation and non-isometric (not norm-preserving) signal dynamics. The same results hold for GATs, as the adjacency still exhibits a contractive spectral structure despite being learned during training Finally, we also plot the Jacobian's singular value spectrum for several GNN architectures in Section 5.9.1, which shows how this phenomenon is also present in other GNNs, even though it is less pronounced.

Note that to overcome this contraction without altering the architecture, one would have to both set $\sigma^2$ in a way that precisely compensates for the normalized adjacency (which can be computationally expensive to estimate)and choose the nonlinearity carefully. In the next subsection, we present a general, simple, and computationally efficient method to place the Jacobian at the edge of chaos at initialization by writing feature updates in a state-space representation.

### 5.2.3 GNN-SSM: Improving the training dynamics of GNNs through state-space models

To allow direct control of the signal propagation dynamics of any GNN, we can rewrite its layer-to-layer update as a state-space model. Concretely, we express the update as

$$\begin{aligned}\mathbf{H}^{(k+1)} &= \mathbf{\Lambda}\mathbf{H}^{(k)} + \mathbf{B}\mathbf{X}^{(k)} \\ &= \mathbf{\Lambda}\mathbf{H}^{(k)} + \mathbf{B}\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{H}^{(k)}, k),\end{aligned} \tag{5.3}$$

where we refer to $\mathbf{\Lambda}$ as the *state transition matrix* and $\mathbf{B}$ as the *input matrix*,[1] and $\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{H}^{(k)}, k)$ as a time-varying *coupling function* which connects each node to some neighborhood. We refer to the model defined in Equation (5.3) as `GNN-SSM`. From an RNN perspective, $\mathbf{\Lambda}$ plays the role of the "memory", in charge of recalling all the representations at each layer at the readout layer, while the neighborhood aggregation plays the role of an input injected into the state via $\mathbf{B}$. In traditional GNNs, this recurrent memory mechanism is absent, so these models act in a *memoryless* way: Features at one layer do not explicitly store or retrieve past information in the way a stateful model would.

In the state-space view, the eigenvalues of $\mathbf{\Lambda}$ determine the *memory dynamics*: large eigenvalues can preserve signals (or, if above unity, cause exploding modes), whereas small eigenvalues quickly attenuate them. Meanwhile, $\mathbf{B}$ controls which aspects of the node features get injected into the hidden state at each step. Because this framework is agnostic to the exact coupling function, any MPNN layer can serve as $\mathbf{F}_{\boldsymbol{\theta}}$. We showcase the effect of these matrices on the layer-wise Jacobian in Proposition 5.2.3.

**Proposition 5.2.3** (Effect of state-space matrices)**.** *Consider the setting in* (5.3) *and* $\Gamma = \partial \operatorname{vec}(\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{H}^{(k)}))/\partial \operatorname{vec}(\mathbf{H}^{(k)})$*. Let* $\otimes$ *denote the Kronecker product. Then, the norm of the vectorized Jacobian* $\mathbf{J}$ *is bounded as:*

$$\begin{aligned}\|\mathbf{J}\|_2 &\leqslant \|I_{d_k} \otimes \mathbf{\Lambda}\|_2 + \|I_{d_k} \otimes \mathbf{B}\|_2 \|\Gamma\|_2 \\ &= \|\mathbf{\Lambda}\|_2 + \|\mathbf{B}\|_2 \|\Gamma\|_2,\end{aligned} \tag{5.4}$$

The result above shows that the spectrum of the Jacobian is controlled through the eigenvalues of the memory matrix $\mathbf{\Lambda}$. Since $\operatorname{eig}(\Gamma) \approx 0$ (see previous subsection), it

[1] Here, we deviate from the traditional state-space formalism, which uses $\mathbf{A}$ as the state transition matrix, since we use this notation for the adjacency.

suffices to have $\text{eig}(\Lambda) \approx 1$ to bring the vectorized Jacobian to the edge of chaos. We empirically validate this in Figure 5.2.

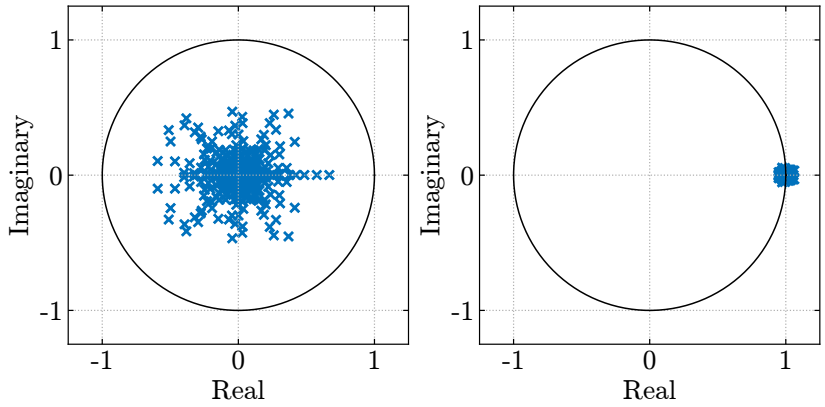


Figure 5.2: Vectorized Jacobian for different models. **Left:** GCN. **Right:** GCN-SMM with $\text{eig}(\mathbf{\Lambda}) \approx 1$ and $\text{eig}(\mathbf{B}) \approx 0.1$.

For simplicity and clarity of conclusions, we consider $\mathbf{\Lambda}$ and $\mathbf{B}$ to be *shared across layers* and *fixed* (i.e., not trained by gradient descent). Only the coupling function $\mathbf{F}_{\boldsymbol{\theta}}$ is optimized. Empirically, we observe that this simpler scheme actually improves downstream performance in some settings. We highlight, however, that this is the most simple instance of a more general framework that aims to incorporate ideas from recurrent processing into GNNs without losing permutation-equivariance. One could easily extend this state-space idea to include more complex gating [143, 230] or other constraints on the state transition matrix.

## 5.3 How does Extreme Gradient Vanishing affect node feature evolution?

In this section, we study the practical implications of the mechanism causing vanishing gradients in GNNs in relation to node feature evolution. We show empirically and theoretically how GNN layers acting as *contractions* make node features collapse to a fixed point. We experimentally validate our points by analyzing Dirichlet energy, node feature norms, and node classification performance for increasing numbers of layers. Overall, we believe this section provides a more *practical and general* understanding of the consequences of extreme vanishing in GNNs by analyzing them from the point of view of their layer-wise Jacobians.

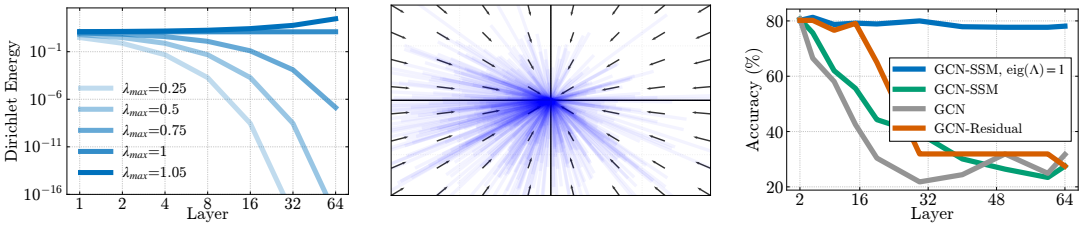


Figure 5.3: Experimental evaluation on Cora for an increasing number of layers. **Left:** Dirichlet Energy evolution for different $||\Lambda||_2$. **Middle:** 2-Dimensional feature projection evolution with a fixed point at zero. **Right:** Node classification performance.

### 5.3.1 Over-smoothing secretly occurs due to contractions in the Jacobian

We consider in our analysis GNN layers as in Equation 2.23. We view a GNN layer as a map $f_k : \mathbb{R}^{nd} \to \mathbb{R}^{nd}$ and construct a deep GNN $f$ via composition of $K$ layers, i.e. $f = f_K \circ \cdots \circ f_1$. Let $\mathbf{J}_f \in \mathbb{R}^{nd \times nd}$ denote the layer-wise Jacobian of a GNN $f$. [2] The supremum of the Jacobian (if well-defined) of $f$ over a convex set $U$ corresponds to the Lipschitz constant $\|f\|_{\text{Lip}}$ [134], i.e. $\|f\|_{\text{Lip}} = \sup_{\mathbf{H} \in U} \|\mathbf{J}_f(\mathbf{H})\|$, where by submultiplicativity of Lipschitz constants we have that $\|f\|_{\text{Lip}} \leqslant \prod_{k=1}^{K} \|f_k\|_{\text{Lip}}$. We point to the Section 5.6.2 for a more detailed explanation of the objects in question. A Lipschitz function $f$ is *contractive* if $\|f\|_{\text{Lip}} < 1$. We now assume that $\|f_k\|_{\text{Lip}} < 1$ for all $k$, meaning that each layer is a *contraction mapping*.[3]

**Lemma 5.3.1** (Banach Fixed Point Theorem [15])**.** *Let $f$ be an operator with Lipschitz constant $\|f\|_{Lip} < 1$. Then for any starting point $\mathbf{x}_0$, the fixed-point iteration $\mathbf{x}_{n+1} = f(\mathbf{x}_n)$ converges to the unique fixed point of $f$ at a linear rate $O\big(1/\|f\|_{Lip}^{n}\big)$*

From Lemma 5.3.1 above and the extreme gradient vanishing results presented in Section 5.2.2, we can distinguish two cases for contractive GNN layers: (1) With shared layers (as in [82]), repeated applications of the GNN will result in convergence to a unique fixed point; (2) with repeated application of non-shared but highly contracting layers, the overall GNN function will converge to, or very close to, a unique fixed point in a single forward pass due to the low Lipschitz constant of the overall GNN. Both cases result in convergence towards a fixed point as all nodes evolve using common transformations. In practice, we observe that node representations tend to collapse to a zero norm node state, see Figure 5.3.

[2] In our analysis, it is important that the input to the GNN is a vector in $\mathbb{R}^{nd}$ rather than a matrix in $\mathbb{R}^{n \times d}$, as the Jacobians and norms are different for the two cases. For this reason, it is important to take care in the definitions of these objects.

[3] Note that the analysis holds for any submultiplicative matrix norm.

To study this further, we consider a GNN update, under the following assumption, which is consistent with the setup in (2.23):

**Lemma 5.3.2.** *Consider a GNN layer $f_K$ as in Equation 2.23, with non-linearity $\sigma$ such that $\sigma(0) = 0$ (e.g. ReLU or $\tanh$). Then, $f(\mathbf{0}) = \mathbf{0}$, i.e. $\mathbf{0}$ is a fixed point of $f$.*

Then, we have that node representations will evolve as in Proposition 5.3.3 presented next.

**Proposition 5.3.3** (Convergence to a unique fixed point.)**.** *Let $\|f_k\|_{Lip} < 1 - \varepsilon$ for some $\varepsilon > 0$ for all $k = 1 \dots L$. Then, for $\mathbf{H} \in U \subseteq \mathbb{R}^{nd}$, we have that:*

$$\|f(\mathbf{H})\| < (1-\varepsilon)^K \|\mathbf{H}\| < \|\mathbf{H}\| . \tag{5.5}$$

*In particular, as $K \to \infty$, $f(\mathbf{H}) \to \mathbf{0}$.*

In other words, in the setting we have considered, if layers $f_k$ are contractive, their repeated application will monotonically converge to the *unique* fixed point $\mathbf{0}$, by Lemmas 5.3.2 and 5.3.1, which serves to explain the behavior observed empirically. We note that this is similar to the analysis in [224], but we highlight that our analysis is much more broad, as it applied to a number of models beyond GCNs, and is it only required knowledge of behavior of each layer through the Lipschitz constant. Furthermore, we emphasize the important connection between the Lipschitz constant and the vanishing gradients problem, which are linked through the Jacobian.

The collapse of node features to a single point has been typically described as *over-smoothing*. In particular, over-smoothing describes the tendency of node features to become too similar to each other as more layers are added in GNNs [37]. A common way of measuring over-smoothing in GNNs, see [229, 228], is via the *unnormalized Dirichlet energy* $\mathcal{E}(\mathbf{H})$. Given a feature matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$ on an unweighted graph $G$, $\mathcal{E}(\mathbf{H})$ takes the form:

$$\mathcal{E}(\mathbf{H}) = \operatorname{tr}\left(\mathbf{H}^\top \mathbf{\Delta} \mathbf{H}\right) = \sum_{(u,v)\in E} \|\mathbf{h}_u - \mathbf{h}_v\|^2 , \tag{5.6}$$

where $\mathbf{\Delta}$ is the unnormalized graph Laplacian [65]. The Dirichlet energy measures the *smoothness* of a signal over a graph and will be minimized when the signal is constant over each node – at least when using the unnormalized Laplacian. In Proposition 5.3.4, we show how the layer-wise Jacobians relate to the unnormalized Dirichlet energy.

**Proposition 5.3.4** (Contractions decrease Dirichlet energy.)**.** *Let $f$ be a GNN, $|E|$ be the number of edges in $G$, and $\mathbf{H} \in \mathbb{R}^{nd}$. We have the following bound:*

$$\mathcal{E}(f(\mathbf{H})) \leqslant 2|E| \prod_{k=1}^{K} \|f_k\|_{Lip}^2 \|\mathbf{H}\|^2 . \tag{5.7}$$

*In particular, if $\|f_k\|_{Lip} < 1 - \varepsilon$ for some $\varepsilon > 0$ for all $k = 1 \dots K$, then as $K \to \infty$ we have that $\mathcal{E}(f(\mathbf{H})) \to 0$.*

This result shows that the unnormalized energy is directly controlled by the norm of the input signal $\mathbf{H}$ and by the contracting effect of the layers $f_k$. The repeated application of contractive layers results in the unnormalized Dirichlet energy being artificially lowered as signals are gradually reduced in norm.

**Important consequences of our theoretical results.** **1)** The most important takeaway of the analysis above is that *vanishing gradients are directly connected to feature collapse through the Lipschitz constant*. In particular, the same mechanism that causes gradient vanishing issues, is responsible for the collapse of all features to a unique fixed point where the Dirichlet energy is minimized. The quick collapse of traditional graph convolutional and attentional models can also be understood from the extreme gradient vanishing result introduced in Section 5.2.

**2)** This result also provides a connection between the study of GNNs and the study of signal propagation (or dynamical isometry) in feedforward networks [234, 216, 212] and recurrent neural networks [143, 6, 209]. In the dynamical isometry literature, the primary interest is to improve the learning times of deep feedforward networks, whereas the recurrent neural network literature is interested in memory and long-range information retrieval. We highlight that *connecting ideas from these fields of study will enable the design of new models that benefit from both worlds*, even though these techniques were originally developed with other objectives in mind. This also serves as an explanation of why simple modifications such as residual connections or normalization worked in practice to mitigate over-smoothing, given their links to dynamical isometry [275, 192].

**3)** Finally, we highlight that this result provides an objective *evaluation metric* to gauge whether a GNN will over-smooth or not. We hope that the eigenanalysis of the Jacobian will become a widespread empirical test used for this purpose.

### 5.3.2 Experimental validation of theoretical results

To validate the theory above, we perform a series of empirical tests. In particular, we check the evolution of the Dirichlet energy, latent vector norms, and node classification accuracy

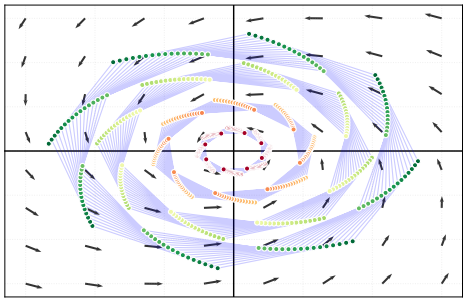

Figure 5.4: Latent evolution of 2-dimensional node features when passing through layers of a GNN-SSM with $\mathrm{eig}(\Lambda) \approx 1$. Colors indicate node feature norm, and the vector field indicates direction.

on the Cora dataset as the number of layers of different models is increased. The results are presented in Figure 5.3. Further, we present additional experiments for different graph structures and models in Section 5.9.2.

From Figure 5.3, we see that one can exactly control the evolution of the Dirichlet energy of the system through the spectrum of the Jacobian, which can, in turn, be modified through the spectrum of $\mathbf{\Lambda}$. Furthermore, this shrinks faster the lower the norm of the Jacobian is, which validates Proposition 5.3.4. From the phase plot describing the two-dimensional evolution of the features, it is also clear that these converge to a unique fixed point at zero, in line with Proposition 5.3.3. Beyond a Dirichlet energy analysis of the system, notice that node classification performance does not deteriorate when $\mathrm{eig}(\mathbf{\Lambda}) \approx 1$, and improves over simply applying an SSM layer with no modulation of the hyperparameters or a residual connection. The dynamics of the GNN in this setting are shown in Figure 5.4.

## 5.4 The Impact of Vanishing Gradients on Over-squashing

In this section, we study the connection between vanishing gradients and over-squashing in GNNs.

### 5.4.1 Mitigating over-squashing by combining increased connectivity and non-dissipativity

Over-squashing is typically measured via the sensitivity of a node embedding after $k$ layers with respect to the input of another node using the node-wise Jacobian. Theorem 2.3.1 shows that the sensitivity of the node embedding arises from a combination of (i) a term

based on the graph topology and (ii) a term dependent on the model dynamics, with over-squashing occurring when the right-hand side of Equation (2.29) becomes too small. We highlight that this differs from the standard product Jacobian which arises in RNNs. This is because in MPNNs, messages are scaled by the inverse node degree, incurring an extra information dissipation step. Consequently, while recurrent architectures only need to adjust their dynamics to ensure long memory, MPNNs must *simultaneously* enhance graph connectivity and modify their dynamics to mitigate vanishing gradients.

Even though the sensitivity bound in Theorem 2.3.1 is controlled by two components, the majority of the literature has typically focused on addressing only the topological term via *graph rewiring* [109, 257, 152, 19, 105], with some methods also targeting the model dynamics [119, 120, 139]. In fact, [81] explicitly discourages increasing the model term in Theorem 2.3.1 and claims that doing so could lead to over-fitting and poorer generalization. However, we argue that increasing the model term — directly linked to vanishing gradients as discussed in Section 5.3 — is essential to mitigate over-squashing. Rather than harming performance, boosting this term helps prevent over-smoothing, since even in a well-connected graph where information can be reached in fewer hops, unaddressed vanishing gradients due to the model term will cause the target node's features to decay to zero during message passing.

Frameworks combining these strategies include [127], which integrates graph rewiring with a delay term, and [83], which merges multi-hop aggregation with ideas from SSMs.[4] These approaches have generally led to state-of-the-art results, significantly improving performance over standalone rewiring techniques.

### 5.4.2 Empirical validation of claims

We focus our empirical validation on answering the following questions: (i) What is the result of combining an effective rewiring scheme with vanishing gradient mitigation? (ii) Will this result in similar state-of-the-art results? To investigate this, we construct a minimal model that combines high connectivity with non-dissipativity. In particular, we make of the GNN-SSM model and employ a k-hop aggregation scheme for the coupling function $\mathbf{F}_{\boldsymbol{\theta}}$, which we term `kGNN-SSM` (more details are provided in Section 5.7).

[4] Further links between the delay term and vanishing gradients are discussed in Section 5.9.3. Further, we show that models tend to converge to the edge of chaos during training in Section 5.9.4.

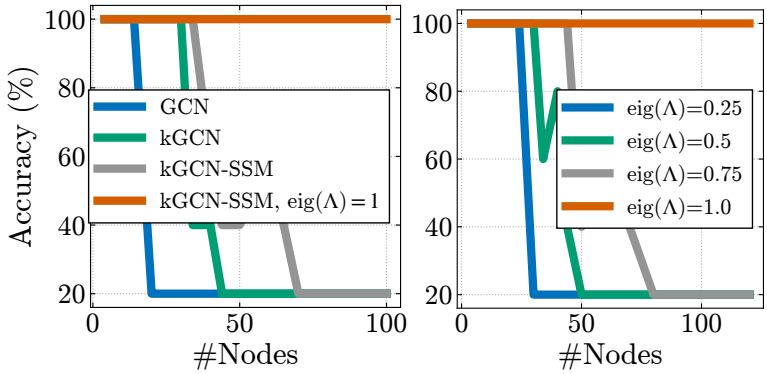


Figure 5.5: **Left:** Performance on the RingTransfer task for different models. **Right:** Effect of dissipativity on performance.

Table 5.1: Ablation on LRGB datasets. Here, $d \uparrow$ means and increase in the latent dimention, while $-$ indicates the removal of a component and $+$ indicates an addition of a component.

| Model | `Pept-func` AP ↑ | `Pept-struct` MAE ↓ |
|---|---|---|
| GCN | $60.93_{\pm 0.138}$ | $33.41_{\pm 0.041}$ |
| kGCN-SSM | $\underline{69.02}_{\pm 0.218}$ | $28.98_{\pm 0.324}$ |
| + $d \uparrow$ | $\mathbf{72.12}_{\pm 0.268}$ | $27.01_{\pm 0.071}$ |
| − eig(Λ) ≈ 1 | $61.41_{\pm 0.724}$ | $\mathbf{25.81}_{\pm 0.032}$ |
| − SSM | $57.76_{\pm 1.971}$ | $\underline{26.02}_{\pm 0.213}$ |
| − khop | $60.93_{\pm 0.138}$ | $33.41_{\pm 0.041}$ |
| DRew-GCN | $68.04_{\pm 1.442}$ | $27.66_{\pm 0.187}$ |
| + $d \uparrow$ | $68.05_{\pm 0.626}$ | $27.64_{\pm 0.067}$ |
| − Delay | $49.02_{\pm 2.512}$ | $27.08_{\pm 0.041}$ |

We start by testing the performance on the RingTransfer task introduced in [81], as it is a task where we certifiably know that long-range dependencies exist. We modify the eig(Λ) in the `kGNN-SSM` to move the Jacobian from the edge of stability to a progressively more dissipative state. The results are shown in Figure 5.5. From the figure, we see that (i) `kGNN-SSM` achieves state-of-the art performance only when coupling strong connectivity and an edge of chaos Jacobian (ii) making the model more dissipative directly results in worse long-range modeling capabilities. We believe the latter point demonstrates the importance of the model term in Theorem 2.3.1.

Next, we ablate each component of the model on three graph property prediction tasks introduced in [119] alongside the real-world long-range graph benchmark (LRGB) from [94], focusing on the `peptides-func` and `peptides-struct` tasks. Additional de-

tails regarding the datasets and the experimental setting are reported in Section 5.8. Here, we focus on ablating the effect of rewiring, adding an SSM layer, and placing the model at the edge of chaos through $\mathbf{\Lambda}$. In the LRBG tasks, we additionally ablate the effect of increasing the hidden memory size, as we consider forty layers in the `peptides-func` dataset, which requires more long-range capabilities. Here, we also ablate `DRew` [127] under the same settings. We also provide a more detailed comparison with other models in Section 5.9.4, and provide additional comments around the LRGB tasks in Section 5.9.5.

Table 5.2: Mean and std. for test $log_{10}(\text{MSE})$ averaged over 4 random weight initializations on the GPP tasks. Lower is better.

| Model | `Diam.` | `SSSP` | `Ecc.` |
|---|---|---|---|
| GCN | $0.742_{\pm 0.047}$ | $0.950_{\pm 9.18 \cdot 10^{-5}}$ | $0.847_{\pm 0.003}$ |
| + SSM | $-2.431_{\pm 0.033}$ | $-2.821_{\pm 0.565}$ | $-2.245_{\pm 0.003}$ |
| + eig($\Lambda$) $\approx 1$ | $\underline{-2.444}_{\pm 0.098}$ | $\underline{-3.593}_{\pm 0.103}$ | $\underline{-2.258}_{\pm 0.009}$ |
| + k-hop | $\mathbf{-3.075}_{\pm 0.055}$ | $\mathbf{-3.604}_{\pm 0.029}$ | $\mathbf{-4.265}_{\pm 0.178}$ |
| DRew-GCN | $-2.369_{\pm 0.105}$ | $-1.591_{\pm 0.003}$ | $-2.100_{\pm 0.026}$ |
| + delay | $-2.402_{\pm 0.110}$ | $-1.602_{\pm 0.008}$ | $-2.029_{\pm 0.024}$ |

The results are shown in Tables 5.1 and 5.2. Across the board, we observe that `kGNN-SSM` not only matches `DRew-Delay`, but also outperforms it by a large amount in all cases, showcasing the strength of our state-space approach. In particular, we generally observe significant decreases in performance when removing both the high connectivity and non-dissipativity components of the model, highlighting their individual importance. Finally, we see that increasing memory size plays a big role in the `peptides-struct` task, which is in line with what has been observed in sequence modeling [124].

## 5.5 Conclusion

In this chapter, we revisit the well-known problems of representational collapse and over-squashing in GNNs from the lens of *vanishing gradients*, by studying GNNs from the perspective of recurrent and state-space models. In particular, we show that GNNs are prone to a phenomenon we term *extreme gradient vanishing*, which results in ill-conditioned signal propagation with few layers. As such, we argue that it is important to control the layerwise Jacobian and propose a state-space-inspired GNN model, termed `GNN-SSM`, to do so. We then uncover that vanishing gradients result in a *specific* form of over-smoothing in which all signals converge exactly to a unique fixed point, and support this claim empirically. Finally, we theoretically argue and empirically show that the mitigation of over-squashing

is best achieved through a combination of strong graph connectivity and non-dissipative dynamics.

## 5.6 Theoretical Results

### 5.6.1 Proofs of Jacobian Theorems

**Definition 5.6.1** (Vectorization and Kronecker product)**.** *Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a real matrix. The* vectorization *of* $\mathbf{X}$*, denoted* $\mathrm{vec}(\mathbf{X})$*, is the $(mn)$-dimensional column vector obtained by stacking the columns of* $\mathbf{X}$*:*

$$\mathrm{vec}(\mathbf{X}) \;=\; \begin{bmatrix} \mathbf{X}_{:,1} \\ \mathbf{X}_{:,2} \\ \vdots \\ \mathbf{X}_{:,n} \end{bmatrix} \;\in\; \mathbb{R}^{mn}.$$

*One key property of the vectorization operator is its relationship to the Kronecker product. In particular, for compatible matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, we have*

$$\mathrm{vec}\big(\mathbf{A}\,\mathbf{B}\,\mathbf{C}\big) \;=\; (\mathbf{C}^T \otimes \mathbf{A})\,\mathrm{vec}\big(\mathbf{B}\big).$$

*Here, $\otimes$ denotes the Kronecker product.*

**Definition 5.6.2** (Wishart matrix)**.** *Let $\mathbf{X} \in \mathbb{R}^{n\times p}$ be a matrix with i.i.d. entries $X_{ij} \sim \mathcal{N}(0, \sigma^2)$. The random matrix $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{p\times p}$ is called a* Wishart matrix *(up to a scaling factor). In particular, such a matrix follows the Wishart distribution $\mathcal{W}_p(n, \sigma^2)$ in certain parametrizations.*

**Definition 5.6.3** (Marchenko–Pastur distribution. [186])**.** *In the high-dimensional limit ($n, p \to \infty$ at a fixed ratio $p/n \to c$), the empirical eigenvalue distribution of the (properly normalized) Wishart matrix $\mathbf{X}^T\mathbf{X}$ converges to the* Marchenko–Pastur distribution*. Concretely, if $\mathbf{X} \in \mathbb{R}^{n\times p}$ has entries $\mathcal{N}(0, 1)$, then the eigenvalues of $\mathbf{X}^T\mathbf{X}$ lie within $[(1-\sqrt{c})^2,\ (1+\sqrt{c})^2]$ for large $n, p$, and their density converges to*

$$f_{\mathrm{MP}}(x) \;=\; \frac{1}{2\pi c\, x}\,\sqrt{(x-a_{\min})(a_{\max}-x)}, \quad x \in [a_{\min}, a_{\max}],$$

*with $a_{\min} = (1-\sqrt{c})^2$ and $a_{\max} = (1+\sqrt{c})^2$. If the entries of $\mathbf{X}$ have variance $\sigma^2 \neq 1$, then the support is rescaled by $\sigma^2$.*

**Lemma 5.6.4** (Spectrum of the Jacobian's singular values)**.** *Let* $\mathbf{H}^{(k)} = \widetilde{\mathbf{A}}\,\mathbf{H}^{(k-1)}\,\mathbf{W}$ *be a linear GCN layer, where* $\widetilde{\mathbf{A}}$ *has eigenvalues* $\{\lambda_1, \dots, \lambda_n\}$ *and* $\mathbf{W}\,\mathbf{W}^T$ *has eigenvalues* $\{\mu_1, \dots, \mu_{d_k}\}$*. Consider the layer-wise Jacobian* $\mathbf{J} = \partial \operatorname{vec}\big(\mathbf{H}^{(k)}\big) / \partial \operatorname{vec}\big(\mathbf{H}^{(k-1)}\big)$*, Then the squared singular values of* $\mathbf{J}$ *are given by the set*

$$\big\{\, \lambda_i^2\, \mu_j \;\big|\; i = 1, \dots, n,\;\; j = 1, \dots, d_k \big\}.$$

*Proof.* By the property of vectorization (Definition 5.6.1), we have

$$\operatorname{vec}\big(\widetilde{\mathbf{A}}\,\mathbf{H}^{(k-1)}\,\mathbf{W}\big) \;=\; (\mathbf{W}^T \otimes \widetilde{\mathbf{A}})\operatorname{vec}\big(\mathbf{H}^{(k-1)}\big).$$

Hence

$$\mathbf{J} \;=\; \mathbf{W}^T \otimes \widetilde{\mathbf{A}}.$$

By properties of the Kronecker product, the eigenvalues of $\mathbf{J}\,\mathbf{J}^T$ are the products of the eigenvalues of $\mathbf{W}^T\mathbf{W}$ and $\widetilde{\mathbf{A}}^2$. Equivalently,

$$\operatorname{spec}\big(\mathbf{J}\,\mathbf{J}^T\big) \;=\; \operatorname{spec}\big(\mathbf{W}^T\mathbf{W}\big) \;\otimes\; \operatorname{spec}\big(\widetilde{\mathbf{A}}^2\big),$$

where spec is the vectorized version of the set of eigenvalues of a matrix. If $\mathbf{W}^T\mathbf{W}$ has eigenvalues $\{\mu_j\}_{j=1}^{d_k}$ and $\widetilde{\mathbf{A}}^2$ has eigenvalues $\{\lambda_i^2\}_{i=1}^{n}$, then the squared singular values of $\mathbf{J}$ are precisely $\lambda_i^2\,\mu_j$ for $i \in \{1, \dots, n\}$, $j \in \{1, \dots, d_k\}$. □

**Theorem 5.6.5** (Jacobian singular-value distribution)**.** *Assume the setting of Lemma 5.2.1, and let* $\mathbf{W} \in \mathbb{R}^{d_{k-1} \times d_k}$ *be initialized with i.i.d.* $\mathcal{N}(0, \sigma^2)$ *entries. Denote the squared singular values of the Jacobian by* $\gamma_{i,j}$*. Then, for sufficiently large* $d_k$ *the empirical eigenvalue distribution* $\mathbf{W}\mathbf{W}^T$ *converges to the Marchenko-Pastur distribution. Then, the mean and variance of each* $\gamma_{i,j}$ *are*

$$\mathbb{E}\big[\gamma_{i,j}\big] = \lambda_i^2\,\sigma^2, \tag{5.8}$$

$$\operatorname{Var}\big[\gamma_{i,j}\big] = \lambda_i^4\,\sigma^4\,\frac{d_k}{d_{k-1}}. \tag{5.9}$$

*Proof.* In this setting, $\mathbf{W}\mathbf{W}^T$ is Wishart if $\mathbf{W}$ has i.i.d. Gaussian entries. Its eigenvalues $\mu_j$ thus converge to the Marchenko–Pastur distribution for large $d_k$. From standard results on the moments of Wishart eigenvalues,

$$\mathbb{E}(\mu_j) \;=\; \sigma^2, \quad \operatorname{Var}(\mu_j) \;=\; \sigma^4\,\frac{d_k}{d_{k-1}}.$$

Since $\gamma_{i,j} = \lambda_i^2\,\mu_j$, we obtain

$$\mathbb{E}[\gamma_{i,j}] \;=\; \lambda_i^2\,\mathbb{E}[\mu_j] \;=\; \lambda_i^2\,\sigma^2,$$

$$\operatorname{Var}[\gamma_{i,j}] \;=\; \lambda_i^4\operatorname{Var}(\mu_j) \;=\; \lambda_i^4\,\sigma^4\,\frac{d_k}{d_{k-1}}.$$

This completes the proof. □

**Proposition 5.6.6** (Effect of state-space matrices)**.** *Consider the setting in* (5.3) *and* $\Gamma = \partial \operatorname{vec}(\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{H}^{(k)}))/\partial \operatorname{vec}(\mathbf{H}^{(k)})$*. Let* $\otimes$ *denote the Kronecker product. Then, the norm of the vectorized Jacobian* $\mathbf{J}$ *is bounded as:*

$$\begin{aligned} \|\mathbf{J}\|_2 &\leqslant \|I_{d_k} \otimes \mathbf{\Lambda}\|_2 + \|I_{d_k} \otimes \mathbf{B}\|_2 \|\Gamma\|_2 \\ &= \|\mathbf{\Lambda}\|_2 + \|\mathbf{B}\|_2 \|\Gamma\|_2, \end{aligned} \tag{5.10}$$

*Proof.* We start by writing

$$\mathbf{J} \;=\; (I_{d_k} \otimes \mathbf{\Lambda}) \;+\; (I_{d_k} \otimes \mathbf{B})\,\Gamma.$$

Using the triangle inequality for the spectral norm,

$$\|\mathbf{J}\|_2 \;=\; \left\|(I_{d_k} \otimes \mathbf{\Lambda}) \;+\; (I_{d_k} \otimes \mathbf{B})\,\Gamma\right\|_2 \;<\; \|I_{d_k} \otimes \mathbf{\Lambda}\|_2 \;+\; \|(I_{d_k} \otimes \mathbf{B})\,\Gamma\|_2.$$

By the submultiplicative property of the spectral norm,

$$\|(I_{d_k} \otimes \mathbf{B})\,\Gamma\|_2 \;<\; \|I_{d_k} \otimes \mathbf{B}\|_2\,\|\Gamma\|_2.$$

Since $\|I_{d_k} \otimes \mathbf{M}\|_2 = \|\mathbf{M}\|_2$ for any matrix $\mathbf{M}$, we obtain

$$\|I_{d_k} \otimes \mathbf{\Lambda}\|_2 \;=\; \|\mathbf{\Lambda}\|_2 \quad \text{and} \quad \|I_{d_k} \otimes \mathbf{B}\|_2 \;=\; \|\mathbf{B}\|_2.$$

Hence,

$$\|\mathbf{J}\|_2 \;<\; \|\mathbf{\Lambda}\|_2 \;+\; \|\mathbf{B}\|_2\,\|\Gamma\|_2.$$

□

### 5.6.2 Proofs to Smoothing Theorems

**Definition 5.6.7** (Lipschitz continuity)**.** *A function* $f : \mathbb{R}^n \to \mathbb{R}^m$ *is Lipschitz continuous if there exists an* $L \geqslant 0$ *such that for all* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*, we have that:*

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leqslant L\,\|\mathbf{x} - \mathbf{y}\|\,,$$

*where we equip* $\mathbb{R}^n$ *and* $\mathbb{R}^m$ *with their respective norms. The minimal such* $L$ *is called the Lipschitz constant of* $f$*.*

The notion of Lipschitz continuity is effectively a bound on the rate of change of a function. It is therefore not surprising that one can relate the Lipschitz constant to the Jacobian of $f$. In particular, we state a useful and well-known result [134] that relates the (continuous) Jacobian map $\mathbf{J}_f$ of a continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ to its Lipschitz constant $L \geqslant 0$. In particular, the Lipschitz constant is is the supremum of the (induced) norm of the Jacobian taken over its domain.

**Lemma 5.6.8** ([134]). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be continuous, with continuous Jacobian $\mathbf{J}_f$. Consider a convex set $U \subseteq \mathcal{R}^n$ If there exists $L \geqslant 0$ such that $\|\mathbf{J}_f(\mathbf{x})\| \leqslant L$ for all $\mathbf{x} \in U$, then $\|f(\mathbf{x}) - f(\mathbf{y})\| \leqslant L \|\mathbf{x} - \mathbf{y}\|$. In particular, we have that the Lipschitz constant of $f$ $L$ is:*

$$L = \sup_{\mathbf{x} \in U} \|\mathbf{J}_f(\mathbf{x})\| .$$

The condition of $U$ being convex is a technicality that is easily achieved in practice with the assumption that input features are bounded and that therefore they live in a convex hull $U$. In particular, at each layer $k$ one can also find a convex hull $U_k$ such that the image of the layer $k-1$ is contained within $U_k$. We highlight that for non-linearities such as ReLU, there are technical difficulties when taking this supremum as there is a non-differentiable point at $0$. This can be circumvented by considering instead a supremum of the (Clarke) generalized Jacobian [149]. We ignore this small detail in this chapter for simplicity as for ReLU this is equivalent to considering the supremum over $U/\mathbf{0}$, i.e. simply ignoring the problematic point $\mathbf{0}$.

**Lemma 5.6.9.** *Consider a GNN layer $f_\ell$ as in Equation 2.23, with non-linearity $\sigma$ such that $\sigma(0) = 0$ (e.g. ReLU or $\tanh$). Then, $f(\mathbf{0}) = \mathbf{0}$, i.e. $\mathbf{0}$ is a fixed point of $f$.*

*Proof.* $f_\ell(\mathbf{0}) = \sigma\left(\widehat{\mathbf{A}}\mathbf{0}\mathbf{W}\right) = \sigma(\mathbf{0}) = \mathbf{0}$. □

**Proposition 5.6.10** (Convergence to unique fixed point.). *Let $\|f_\ell\|_{Lip} \leqslant 1 - \varepsilon$ for some $\varepsilon > 0$ for all $\ell = 1 \dots L$. Then, for $\mathbf{H} \in U \subseteq \mathbb{R}^{nd}$, we have that:*

$$\|f(\mathbf{H})\| \leqslant (1-\varepsilon)^L \|\mathbf{H}\| < \|\mathbf{H}\| . \tag{5.11}$$

*In particular, as $L \to \infty$, $f(\mathbf{H}) \to \mathbf{0}$.*

*Proof.* By Lipschitz regularity of $f$ over $U$, we have that $\|f(\mathbf{x}) - f(\mathbf{y})\| \leqslant \|f\|_{\text{Lip}} \|\mathbf{x} - \mathbf{y}\|$. Recall that by Lemma 5.3.2, we have that $f(\mathbf{0}) = \mathbf{0}$. This implies:

$$\begin{aligned}
\|f(\mathbf{H}) - f(\mathbf{0})\| &= \|f(\mathbf{H})\| \\
&\leqslant \|f\|_{\text{Lip}} \|\mathbf{H}\| \\
&\leqslant \prod_{\ell=1}^{L} \|f_\ell\|_{\text{Lip}} \|\mathbf{H}\| \\
&< \|\mathbf{H}\| ,
\end{aligned}$$

where in the last step we use the fact that Lipschitz constants are submultiplicative and that for all $\ell$ we have that $\|f_\ell\|_{\text{Lip}} < 1$ by assumption. The final statement is immediate by

the Banach fixed point theorem and by noting that $f_\ell$ all share the same fixed point $\mathbf{0}$ by Lemma 5.3.2. □

**Proposition 5.6.11** (Contractions decrease Dirichlet energy.)**.** *Let $f$ be a GNN, $|E|$ be the number of edges in $G$, and $\mathbf{H} \in \mathbb{R}^{nd}$. We have the following bound:*

$$\mathcal{E}(f(\mathbf{H})) \leqslant 2|E| \prod_{\ell=1}^{L} \|f_\ell\|_{Lip}^2 \|\mathbf{H}\|^2 . \tag{5.12}$$

*In particular, if $\|f_\ell\|_{Lip} \leqslant 1-\varepsilon$ for some $\varepsilon > 0$ for all $\ell = 1 \dots L$, then as $L \to \infty$, $\mathcal{E}(f(\mathbf{H})) \to 0$.*

*Proof.* We denote by $f(\mathbf{H})|_i \in \mathbb{R}^d$, the $d$-dimensional evaluation of f(H) at node $i$. We make use of the inequality $\|f(\mathbf{H})|_i\| \leqslant \|\mathbf{H}\|$.

$$\begin{aligned}
\mathcal{E}(f(\mathbf{H})) &= \sum_{i\sim j} \|f(\mathbf{H})|_i - f(\mathbf{H})|_j\|^2 \\
&\leqslant \sum_{i\sim j} \|f(\mathbf{H})|_i\|^2 + \|f(\mathbf{H})|_j\|^2 \\
&\leqslant 2\sum_{i\sim j} \|f(\mathbf{H})\|^2 \\
&\leqslant 2\,\|f\|_{\text{Lip}}^2 \sum_{i\sim j} \|\mathbf{H}\|^2 \\
&= 2\,\|f\|_{\text{Lip}}^2\,|E|\,\|\mathbf{H}\|^2 \\
&\leqslant 2\prod_{\ell=1}^{L} \|f_\ell\|_{\text{Lip}}^2\,|E|\,\|\mathbf{H}\|^2 .
\end{aligned}$$

It is then clear that, if $\|f_\ell\|_{\text{Lip}} \leqslant 1-\varepsilon$ for some $\varepsilon > 0$ for all $\ell = 1 \dots L$, $\prod_{\ell=1}^{L} \|f_\ell\|_{\text{Lip}}^2 \leqslant (1-\varepsilon)^{2L} \to 0$ as $L \to \infty$. □

## 5.7 kGNN-SSM: A simple method to combine high connectivity and non-dissipativity.

To test our assumption on more complex downstream tasks, we construct a minimal model that combines high connectivity with non-dissipativity. To guarantee high connectivity, we employ a k-hop aggregation scheme. In particular, each node $i$ at layer $k$ will aggregate information as

$$a_{i,k}^{(k)} = \psi^k\Big(\{h_j^{(k)} : j \in \mathcal{N}_k(i)\}\Big), \tag{5.13}$$

where

$$\mathcal{N}_k(i) := \{j \in V : d_G(i,j) = k\}$$

and $d_G : V \times V \to \mathbb{R}_{\geqslant 0}$ is the length of the minimal walk connecting nodes $i$ and $j$. This approach avoids a large amount of information being squashed into a single vector, and is more in line with the recurrent paradigm. We note that this scheme is similar to [83], but in this case we do not consider different block or parameter sharing, and our recurrent mechanism is based on an untrained SSM layer.

We denote a GNN endowed with this rewiring scheme and wrapped with our SSM layer as `kGNN-SSM`.

## 5.8 Experimental Details

In this section, we provide additional experimental details, including dataset and experimental setting description and employed hyperparameters.

**Over-smoothing task.** In this task, we aim to analyze the dynamics of the Dirichlet energy across three different graph topologies: Cora [276], Texas [211], and a grid graph. The Cora dataset is a citation network consisting of 2,708 nodes (papers) and 10,556 edges (citations). The Texas dataset represents a webpage graph with 183 nodes (web pages) and 499 edges (hyperlinks). Lastly, the grid graph is a two-dimensional $10 \times 10$ regular grid with 4-neighbor connectivity. For all three graphs, node features are randomly initialized from a normal distribution with a mean of 0 and variance of 1. These node features are then propagated over 80 layers (or iterations) using untrained GNNs to observe the energy dynamics.

**Graph Property Prediction.** This experiment consists of predicting two node-level (i.e., eccentricity and single source shortest path) and one graph-level (i.e., graph diameter) properties on synthetic graphs sampled from different distribution, i.e., Erdos–Rényi, Barabasi-Albert, grid, caveman, tree, ladder, line, star, caterpillar, and lobster. Each graph contains between 25 and 35 nodes, with nodes assigned with input features sampled from a uniform distribution in the interval $[0, 1)$. The target values correspond to the predicted graph property. The dataset consists of 5,120 graphs for training, 640 for validation and 1,280 for testing.

We employ the same experimental setting and data outlined in [119]. Each model is designed as three components: the encoder, the graph convolution, and the readout. We perform hyperparameter tuning via grid search, optimizing the Mean Square Error (MSE).

Table 5.3: The grid of hyperparameters employed during model selection for the graph property prediction tasks (*GraphProp*), and `peptides-func` and `peptides-struct`.

| **Hyperparameters** | **Values** | |
|---|---|---|
| | ***GraphProp*** | `peptides-(func,struct)` |
| Optimizer | Adam | AdamW |
| Learning rate | 0.003 | 0.001 |
| Weight decay | $10^{-6}$ | - |
| N. Layers | 10 | 17, 40 |
| embedding dim | 20, 30 | 105 |
| $\sigma$ | tanh | ReLU |
| eig($\Lambda$) | 0.5, 0.75, 1.0 | 1.0 |

The models are trained using the Adam optimizer for a maximum of 1500 epochs, with early stopping based on the validation error, applying a 100 epochs patience. For each model configuration, we perform 4 training runs with different weight initializations and report the average results. We report in Table 5.3 the employed grid of hyperparameters.

**Long-Range Graph Benchmark.** We consider the `peptides-func` and `peptides-struct` datasets from [94]. Both datasets consist of 15,535 graphs, where each graph corresponds to 1D amino acid chain (i.e., peptide), where nodes are the heavy atoms of the peptide and edges are the bonds between them. `peptides-func` is a multi-label graph classification dataset whose objective is to predict the peptide function, such as antibacterial and antiviral function. `peptides-struct` is a multi-label graph regression dataset focused on predicting the 3D structural properties of peptides, such as the inertia of the molecule and maximum atom-pair distance.

We use the same experimental setting and splits from [94]. We perform hyperparameter tuning via grid search, optimizing the Average Precision (AP) in the Peptide-func and Mean Absolute Error (MAE) in the Peptide-struct. The models are trained using the AdamW optimizer for a maximum of 300 epochs. For each model configuration, we perform four training runs with different weight initializations and report the average results. We report in Table 5.3 the employed grid of hyperparameters.

**Tested Hyperparameters.** In Table 5.3 we report the grid of hyperparameters employed in our experiments by our method.

## 5.9 Additional empirical results

In this section, we propose additional empirical results on over-smoothing and over-squashing, as well as the eigendistribution of the layerwise Jacobians of various standard GNNs.

### 5.9.1 Additional MPNN Jacobians

Here, we present in Figure 5.6 the eigendistribution of the layerwise Jacobians of GCN, GIN [274] and Gated-GCN [33]. Across the board, we observe similar contraction effects in the Jacobian as those presented in the main paper, with a long number of eigenvalues accumulating at zero, with no significant changes in the distribution during training. However, the maximum eigenvalues for both GIN and Gated-GCN are much larger than those of GCN. We also compare a nonlinear feedforward network and a nonlinear GCN in Figure 5.7.

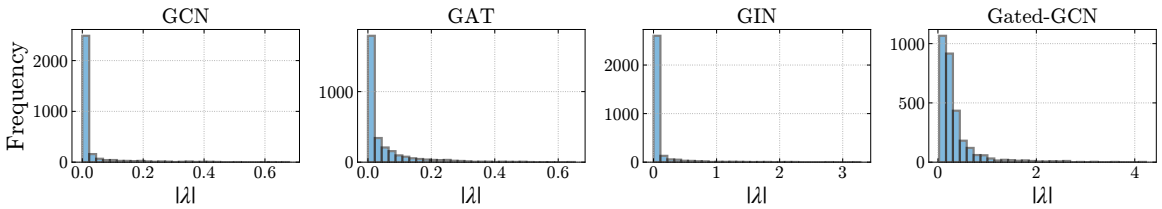


Figure 5.6: Eigenvalues of layer-to-layer Jacobian of different GNN models.

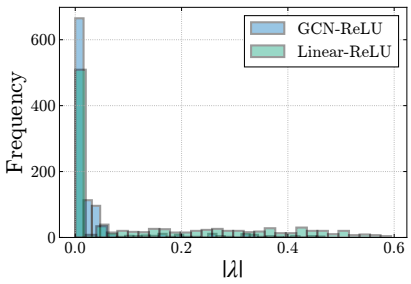


Figure 5.7: Histogram of eigenvalue modulus of the layerwise Jacobian for a nonlinear convolutional and a nonlinear feedforward layer.

### 5.9.2 Over-smoothing

Here, we include additional results related to over-smoothing experiments. Figure 5.8 shows the effect of $||\Lambda||_2$ in GCN-SSM on different graph structures, showing that lower Jacobian norms leads to a rapid decay of the Dirichlet energy, whereas values closer to one result in a more stable energy evolution. This result is also confirmed by Figure 5.10 and Figure 5.11. The former presents the vectorized Jacobian for ADGN [119], SWAN [120],

and PHDGN [139] on Cora, while the latter the Dirichlet energy evolution of different models on different topologies. Notably, in Figure 5.11, ADGN, SWAN, and PHDGN exhibit stable Dirichlet energy across layers, and Figure 5.10 reveals that these Jacobian norms are close to one. These results confirm that stable dynamics also ensure a non-decaying Dirichlet energy, effectively preventing over-smoothing.

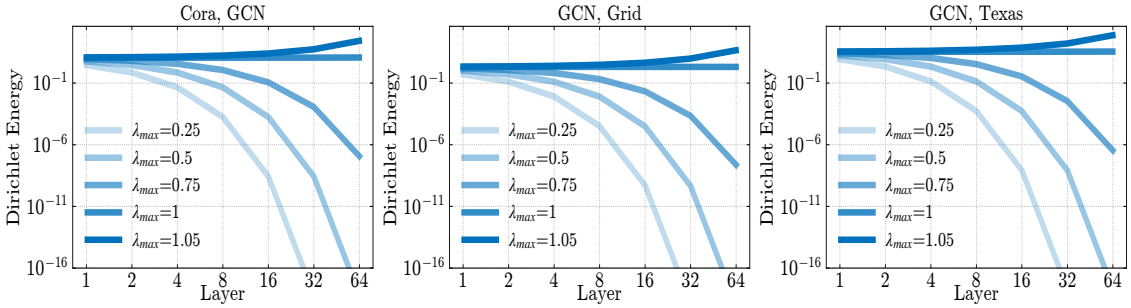


Figure 5.8: Dirichlet Energy evolution of GCN-SSM for different $||\Lambda||_2$ on different graph topologies. **Left:** Cora. **Middle:** Grid graph. **Right:** Texas.

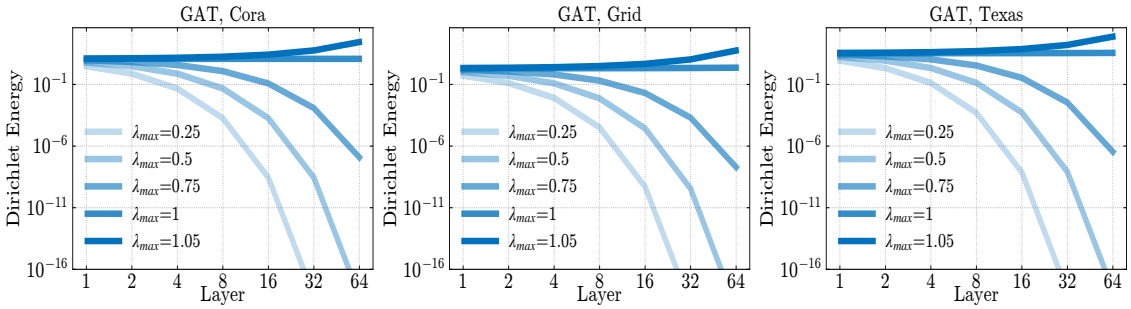


Figure 5.9: Dirichlet Energy evolution of GAT-SSM for different $||\Lambda||_2$ on different graph topologies. **Left:** Cora. **Middle:** Grid graph. **Right:** Texas.

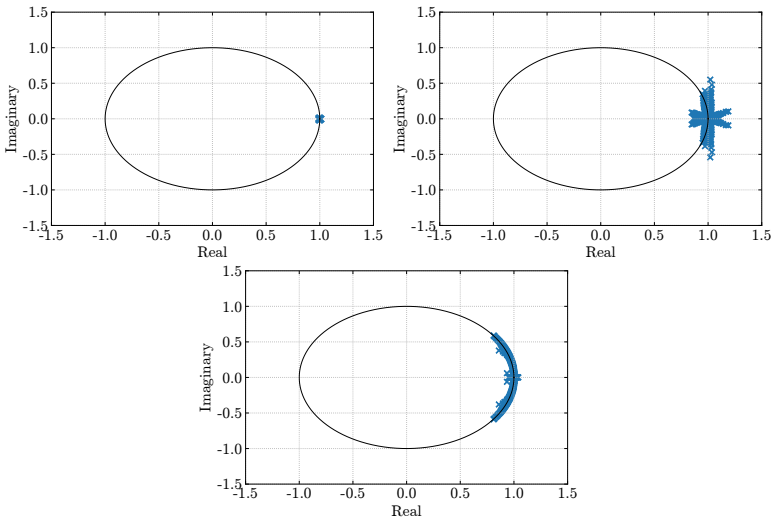


Figure 5.10: Vectorized Jacobian for ADGN [119], SWAN [120], and PHDGN [139] on Cora. **Left:** ADGN. **Middle:** SWAN. **Right:** PHDGN.

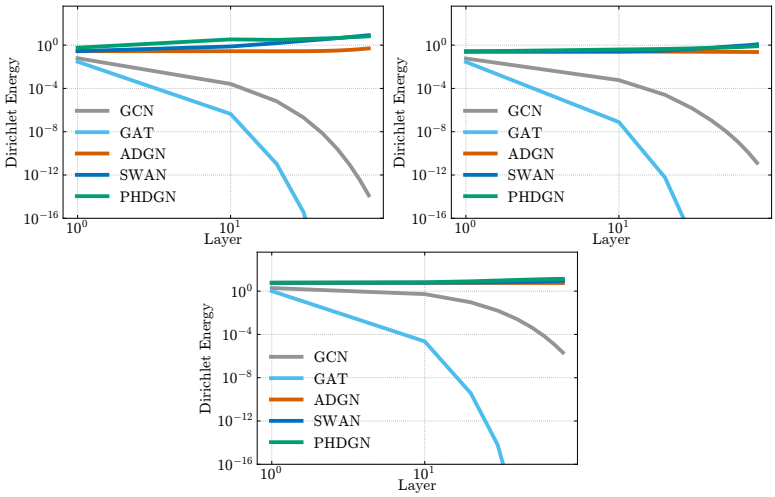


Figure 5.11: Dirichlet Energy evolution of different models on different topologies. **Left:** Cora. **Middle:** Grid graph. **Right:** Texas.

### 5.9.3 Link between delay and vanishing gradients

Here, we show how the delay term in [127] is directly related to preventing vanishing gradients. We do so by showing that adding the delay term to a GCN is effective at preventing over-smoothing, see Figure 5.12, as well as by checking the histogram of eigenvalues of the Jacobian, see Figure 5.14.

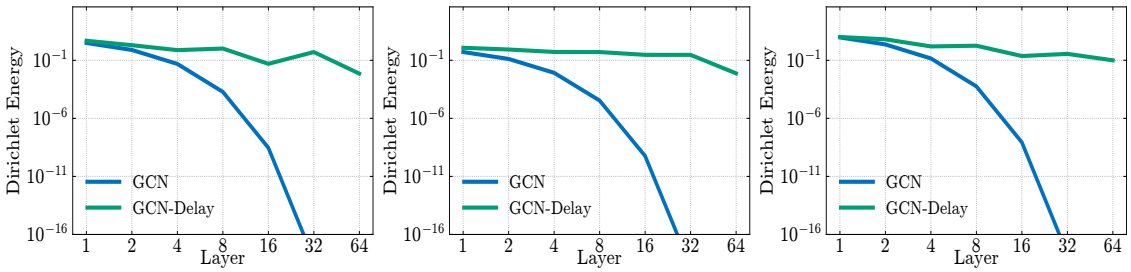


Figure 5.12: Dirichlet Energy evolution of GCN (+delay mechanism) on different topologies. **Left:** Cora. **Middle:** Grid graph. **Right:** Texas.

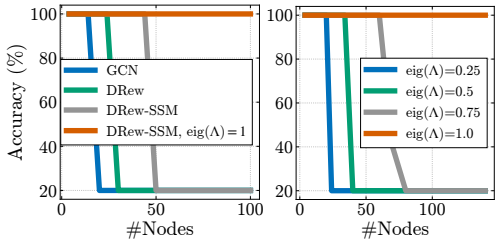


Figure 5.13: **Left:** Performance on the RingTransfer task for DRew [127]. **Right:** Effect of dissipativity on performance.

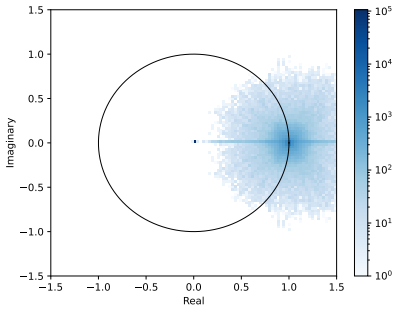


Figure 5.14: Eigenvalue distribution of DRew-GCN+delay on the ring transfer task.

### 5.9.4 Graph Property Prediction

**Edge-of-chaos behavior and long-range propagation.** To further support our claim that mitigating gradient vanishing is key to strong long-range performance, Figure 5.15 shows each method's average Jacobian eigenvalue distance to the edge-of-chaos (EoC) region. The figure demonstrates that methods such as ADGN [119] and SWAN [120], which remain closer to EoC, effectively propagate information over large graph radii, resulting in superior performance across all three tasks. Figure 5.16 presents an ablation study on multiple ADGN variants, controlled by the hyperparameter $\gamma$, which governs the positioning of

the Jacobian eigenvalues ($\gamma < 0$ places them outside the stability region, $\gamma > 0$ inside, and $\gamma = 0$ on the unit circle). Notably, regardless of the initial value of $\gamma$, ADGN consistently converges towards the EoC region as performance improves.

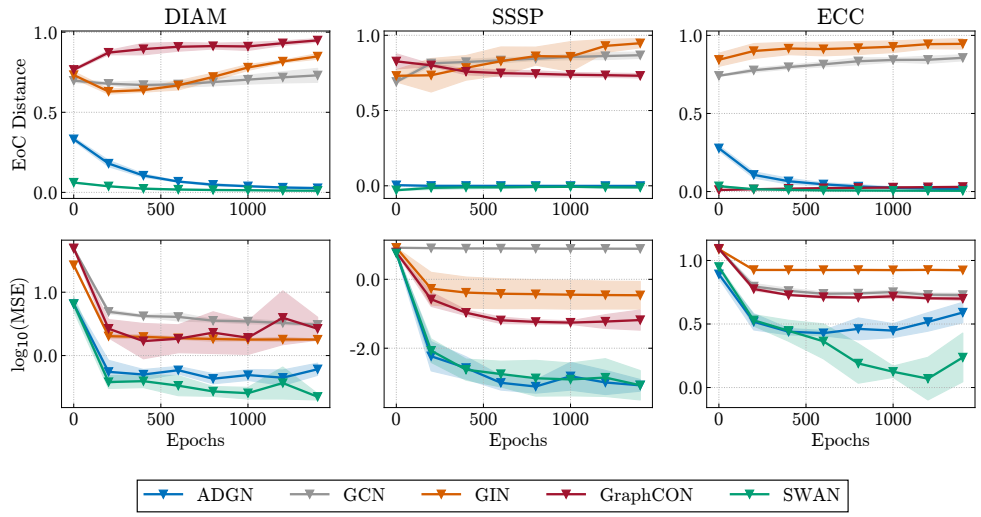


Figure 5.15: Performance on Graph Property Prediction tasks and average Jacobian eigenvalue distance to the edge of chaos (EoC) region for different GNN models.

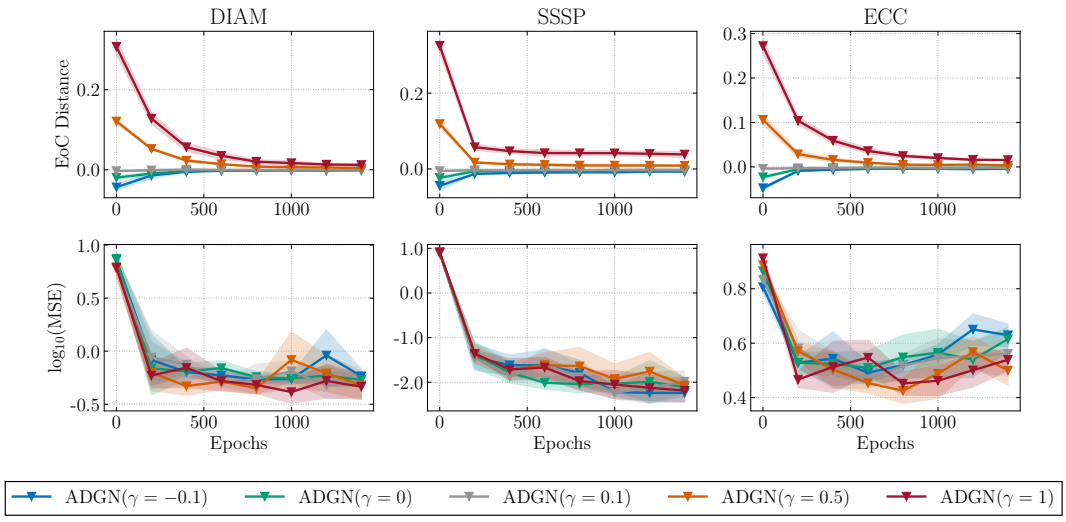


Figure 5.16: Performance on Graph Property Prediction tasks and average Jacobian eigenvalue distance to the edge of chaos (EoC) region for different ADGN dynamics, i.e., $\gamma \in [-0.1, 1]$. Negative values of $\gamma$ places the eigenvalues of the ADGN Jacobian outside the stability region, otherwise for positive values.

**Complete results.** Table 5.4 compares our method on graph property prediction tasks against a range of state-of-the-art approaches, including GCN [162], GAT [261], GraphSAGE [129], GIN [274], GCNII [59], DGC [265], GRAND [54], GraphCON [229], ADGN [119],

SWAN [120], PH-DGN [139], and DRew [127]. Our method achieves exceptional results across all three tasks, consistently surpassing MPNN baselines, differential equation-inspired GNNs, and multi-hop GNNs. These findings underscore how combining powerful model dynamics with improved connectivity provides substantial benefits in tasks that require long-range information propagation.

Table 5.4: Mean test set $log_{10}(\text{MSE})(\downarrow)$ and std averaged on 4 random weight initializations on Graph Property Prediction tasks. The lower, the better. Baseline results are reported from [119, 120, 139].

| Model | Diameter | SSSP | Eccentricity |
|---|---|---|---|
| **MPNNs** | | | |
| GCN | $0.7424_{\pm 0.0466}$ | $0.9499_{\pm 0.0001}$ | $0.8468_{\pm 0.0028}$ |
| GAT | $0.8221_{\pm 0.0752}$ | $0.6951_{\pm 0.1499}$ | $0.7909_{\pm 0.0222}$ |
| GraphSAGE | $0.8645_{\pm 0.0401}$ | $0.2863_{\pm 0.1843}$ | $0.7863_{\pm 0.0207}$ |
| GIN | $0.6131_{\pm 0.0990}$ | $-0.5408_{\pm 0.4193}$ | $0.9504_{\pm 0.0007}$ |
| GCNII | $0.5287_{\pm 0.0570}$ | $-1.1329_{\pm 0.0135}$ | $0.7640_{\pm 0.0355}$ |
| **Differential Equation inspired GNNs** | | | |
| DGC | $0.6028_{\pm 0.0050}$ | $-0.1483_{\pm 0.0231}$ | $0.8261_{\pm 0.0032}$ |
| GRAND | $0.6715_{\pm 0.0490}$ | $-0.0942_{\pm 0.3897}$ | $0.6602_{\pm 0.1393}$ |
| GraphCON | $0.0964_{\pm 0.0620}$ | $-1.3836_{\pm 0.0092}$ | $0.6833_{\pm 0.0074}$ |
| ADGN | $-0.5188_{\pm 0.1812}$ | $-3.2417_{\pm 0.0751}$ | $0.4296_{\pm 0.1003}$ |
| SWAN | $-0.5981_{\pm 0.1145}$ | $-3.5425_{\pm 0.0830}$ | $-0.0739_{\pm 0.2190}$ |
| PH-DGN | $-0.5473_{\pm 0.1074}$ | $\mathbf{-4.2993}_{\pm 0.0721}$ | $-0.9348_{\pm 0.2097}$ |
| **Graph Transformers** | | | |
| GPS | $-0.5121_{\pm 0.0426}$ | $-3.5990_{\pm 0.1949}$ | $0.6077_{\pm 0.0282}$ |
| **Multi-hop GNNs** | | | |
| DRew-GCN | $-2.3692_{\pm 0.1054}$ | $-1.5905_{\pm 0.0034}$ | $-2.1004_{\pm 0.0256}$ |
| + delay | $-2.4018_{\pm 0.1097}$ | $-1.6023_{\pm 0.0078}$ | $-2.0291_{\pm 0.0240}$ |
| **Our** | | | |
| GCN-SSM | $-2.4312_{\pm 0.0329}$ | $-2.8206_{\pm 0.5654}$ | $-2.2446_{\pm 0.0027}$ |
| + eig$(\Lambda) \approx 1$ | $\underline{-2.4442}_{\pm 0.0984}$ | $-3.5928_{\pm 0.1026}$ | $\underline{-2.2583}_{\pm 0.0085}$ |
| + k-hop | $\mathbf{-3.0748}_{\pm 0.0545}$ | $\underline{-3.6044}_{\pm 0.0291}$ | $\mathbf{-4.2652}_{\pm 0.1776}$ |

### 5.9.5 Additional comments on LRGB tasks

In our experiments with the LRGB tasks, we observe that the `peptides-func` task exhibits significantly longer-range dependencies than the `peptides-struct` task. Notably, the `peptides-struct` task performs best when the model is not initialized at the edge of chaos and requires fewer layers. Conversely, on `peptides-struct` the model performs best when it is set to be at the edge of chaos, and shows a monotonic performance increase with additional layers, with optimal results achieved when using forty layers.

Furthermore, we highlight that while our experiments with a small hidden dimension

adhere to the parameter budget established in [94], increasing the hidden dimension ($d \uparrow$) to 256 causes us to exceed the 500k parameter budget limit, even though our model maintains the same number of parameters as a regular GCN. While this budget is a useful tool to benchmark different models, we highlight that this restriction results in models running with fewer layers and small hidden dimensions. However, a large number of layers is crucial for effective long-range learning in graphs that are not highly connected, while increasing the hidden dimension also directly affects the bound in Theorem 2.3.1. As such, we believe that this parameter budget indirectly benefits models with higher connectivity graphs, inadvertently hindering models that do not perform edge addition.

## 5.10 Supplementary Related Work

**Long-range propagation on GNNs.** Learning long-range dependencies on graphs involves effectively propagating and preserving information across distant nodes. Despite recent advancements, ensuring effective long-range communication between nodes remains an open problem [241]. Several techniques have been proposed to address this issue, including graph rewiring methods, such as [109, 257, 152, 19, 127, 26], which modify the graph topology to enhance connectivity and facilitate information flow. Similarly, Graph Transformers enhance the connectivity to capture both local and global interactions, as demonstrated by [277, 93, 242, 166, 217, 270]. Other approaches incorporate non-local dynamics by using a fractional power of the graph shift operator [189], leverage quantum diffusion kernels [187], regularize the model's weight space [119, 120], or exploit port-hamiltonian dynamics [139].

Despite the effectiveness of these methods in learning long-range dependencies on graphs, they primarily introduce solutions to mitigate the problem rather than establishing a unified theoretical framework that defines its underlying cause.

**Vanishing gradients in sequence modelling and deep learning.** One of the primary challenges in training recurrent neural networks lies in the vanishing (and sometimes exploding) gradient problem, which can hinder the model's ability to learn and retain information over long sequences. In response, researchers have proposed numerous architectures aimed at preserving or enhancing gradients through time. Examples include Unitary RNNs [6], Orthogonal RNNs [**?** ], coRNNs [231], Linear Recurrent Units [209], and Structured State Space Models [124, 122]. By leveraging properties such as orthogonality, carefully designed parameterizations, or alternative update mechanisms, these models seek to alleviate gradient decay and capture longer-range temporal relationships more effectively.

**Dynamical-systems-inspired neural networks.** Since the introduction of Neural ODEs in [60], there have been various methods that employ ideas of dynamical systems within neural networks, including continuous-time methods [227, 202, 38, 39, 24, 196, 40] or state-space approaches [57, 90, 92]. Within graph neural networks, we highlight PDE-GCN [96], GRAND [54], BLEND [52] and Neural Sheaf Diffusion [28].

# Chapter 6

# Neural Latent Geometry Search

Recent research indicates that the performance of machine learning models can be improved by aligning the geometry of the latent space with the underlying data structure. Rather than relying solely on Euclidean space, researchers have proposed using hyperbolic and spherical spaces with constant curvature, or combinations thereof, to better model the latent space and enhance model performance. However, little attention has been given to the problem of automatically identifying the optimal latent geometry for the downstream task. We mathematically define this novel formulation and coin it as neural latent geometry search (NLGS). More specifically, we introduce an initial attempt to search for a latent geometry composed of a product of constant curvature model spaces with a small number of query evaluations, under some simplifying assumptions. To accomplish this, we propose a novel notion of distance between candidate latent geometries based on the Gromov-Hausdorff distance from metric geometry. In order to compute the Gromov-Hausdorff distance, we introduce a mapping function that enables the comparison of different manifolds by embedding them in a common high-dimensional ambient space. We then design a graph search space based on the notion of *smoothness* between latent geometries, and employ the calculated distances as an additional inductive bias. Finally, we use Bayesian optimization to search for the optimal latent geometry in a query-efficient manner. This is a general method which can be applied to search for the optimal latent geometry for a variety of models and downstream tasks. We perform experiments on synthetic and real-world datasets to identify the optimal latent geometry for multiple machine learning problems.

## 6.1 Introduction

There has been a recent surge of research employing ideas from differential geometry and topology to improve the performance of learning algorithms [31, 53, 146, 17, 18]. Tradi-

tionally, Euclidean spaces have been the preferred choice to model the geometry of latent spaces in the ML community [267, 35]. However, recent work has found that representing the latent space with a geometry that better matches the structure of the data can provide significant performance enhancements in both reconstruction and other downstream tasks [243]. In particular, most works have employed *constant curvature model spaces* such as the Poincaré ball model [190], the hyperboloid [55], or the hypersphere [283], to encode latent representations of data in a relatively simple and computationally tractable way.

While individual model spaces have sometimes shown superior performance when compared to their Euclidean counterparts, more recent works [125, 245, 233, 232, 280, 107, 214] have leveraged the notion of *product spaces* (also known as *product manifolds*) to model the latent space. This idea allows to generate more complex representations of the latent space for improved performance by taking Cartesian products of model spaces, while retaining the computational tractability of mathematical objects such as *exponential maps* or *geodesic distances*, see [246]. Despite the success of this methodology, there exists no principled way of obtaining the *product manifold signature* (*i.e.*, the choice and number of manifold components used to generate the product manifold and their respective dimensionalities) to optimally represent the data for downstream task performance. This procedure is typically performed heuristically, often involving a random search over the discrete combinatorial space of all possible combinations of product manifold signatures, which is an exceedingly large search space that hampers computational efficiency and practical applicability. Some other work related to latent space geometry modelling can be found in [180, 135, 10].

**Contributions. 1)** In this chapter, we consider a novel problem setting where we aim to search for an optimal latent geometry that best suits the model and downstream task. As a particular instance of this setting, we consider searching for the optimal product manifold signature. Due to the conceptual similarity with neural architecture search (NAS) strategies [98, 288, 215], we coin this problem as *neural latent geometry search (NLGS)*, which we hope will encourage additional work in the direction of optimal latent geometry inference. We test our framework on a variety of use cases, such as autoencoder reconstruction [190] and latent graph inference [233], for which we create a set of custom datasets.

**2)** To search for the product manifold signature, we must be able to compare product manifolds. This is traditionally done by computing the Hausdorff distance between manifolds [253], which however requires manifolds to reside in the same metric space. To

address this limitation, in this work we develop to our knowledge the first computational method to mathematically compare product manifolds. Our approach generalizes classical algorithms and allows the comparison of manifolds existing in different spaces. This is achieved by defining an isometric embedding that maps the manifolds to a common high-dimensional ambient space, which enables the computation of their Gromov-Hausdorff distances.

**3)** Leveraging the Gromov-Hausdorff distances between candidate latent space manifolds, we design a principled and query-efficient framework to search for an optimal latent geometry, in the sense that it yields the best performance with respect to a given machine learning model and downstream task. Our approach consists of constructing a geometry-informed graph search space where each node in the graph represents a unique candidate product manifold, associated with the model performance using this manifold as its embedding space. The strength of edges in the graph are based on the inverse of the Gromov-Hausdorff distance, thereby encoding a notion of "closeness" between manifolds in the search space. We then perform efficient search over this space using Bayesian optimization (BO). We compare our proposed method with other search algorithms that lack the topological prior inherent in our model. Empirical results demonstrate that our method outperforms the baselines by a significant margin in finding the optimal product manifold.

**Outline.** In Section 6.2 we discuss manifold learning, and product manifolds of constant curvature model spaces such as the Euclidean plane, the hyperboloid, and the hypersphere. We also review relevant mathematical concepts, particularly the Hausdorff and Gromov-Hausdorff distances from metric geometry [116]. Section 6.3 presents the problem formulation, the proposed methodology to compare product manifolds, as well as how the search space over which to perform geometry-informed Bayesian optimization is constructed. Finally, Section 6.4 explains how our custom synthetic and real-world datasets were obtained, and the empirical results. Lastly, in Section 6.5 we conclude and discuss avenues for future work.[1]

## 6.2 Background

**Differential and Riemannian Geometry** A *smooth manifold* is a space that locally resembles $\mathbb{R}^n$ but may have non-Euclidean global structure. In differential geometry, one uses

[1]For a high level description of the proposed framework, we recommend skipping to Section 6.3

calculus to study curves, surfaces, and their higher-dimensional analogues. A *Riemannian manifold* $(\mathcal{M}, g)$ is a smooth manifold $\mathcal{M}$ equipped with a smoothly varying inner product

$$g_p : T_p\mathcal{M} \times T_p\mathcal{M} \longrightarrow \mathbb{R}, \tag{6.1}$$

on each tangent space $T_p\mathcal{M}$. Concretely, $g_p$ takes two tangent vectors $X, Y \in T_p\mathcal{M}$ and returns a real number $g_p(X, Y)$ that depends smoothly on the base point $p$. This metric lets us define the norm of a tangent vector $X \in T_p\mathcal{M}$ by

$$\|X\|_p = \sqrt{g_p(X, X)}\,, \tag{6.2}$$

If $c : [0, 1] \to \mathcal{M}$ is a smooth curve with velocity $c'(t) \in T_{c(t)}\mathcal{M}$, its length is

$$L(c) \;=\; \int_0^1 \big\| \, c'(t) \big\|_{c(t)} \, dt \;=\; \int_0^1 \sqrt{g_{\,c(t)}\big(c'(t),\, c'(t)\big)} \; dt. \tag{6.3}$$

Minimizing this length among all curves connecting two points in $\mathcal{M}$ yields *geodesics*, which generalize "straight lines" to curved spaces and locally minimize distance. The infimum of lengths over all such curves defines the *geodesic distance* $d(p, q)$ between any two points $p, q \in \mathcal{M}$.

Given a tangent vector $v \in T_p\mathcal{M}$, one can solve the geodesic equation (a second-order ordinary differential equation determined by $g$) to obtain a unique geodesic $\gamma_v(t)$ with initial conditions $\gamma_v(0) = p$ and $\gamma_v'(0) = v$. Solving this ODE with yields a unique curve $\gamma_v(t)$ whose initial "velocity" is exactly the tangent vector $v \in T_p\mathcal{M}$.

The *exponential map* at $p$ is then defined by

$$\exp_p : T_p\mathcal{M} \longrightarrow \mathcal{M}, \qquad \exp_p(v) \;=\; \gamma_v(1), \tag{6.4}$$

meaning that $\exp_p(v)$ is the point reached by traveling along the geodesic in direction $v$ for unit time. The *logarithm map* at $p$ is (at least locally) the inverse of $\exp_p$. In applications, the exponential and logarithm maps allow us to move back and forth between the curved manifold $\mathcal{M}$ and the linear tangent space $T_p\mathcal{M}$, enabling tools from linear algebra and calculus to analyze inherently nonlinear spaces in a locally faithful way.

On the other hand, a product manifold results from taking the Cartesian product of two or more manifolds, resulting in a manifold with a natural product structure. This enables us to examine the local and global geometry of the product manifold by studying the geometry

of its individual factors. A *product manifold* $\mathcal{P} = M_1 \times \cdots \times M_k$ carries the direct-sum metric

$$g_{\mathcal{P}} = g_1 \oplus \cdots \oplus g_k, \tag{6.5}$$

so that for $x = (x^1, \ldots, x^k)$ and $y = (y^1, \ldots, y^k)$,

$$d_{\mathcal{P}}(x, y) = \sqrt{\sum_{i=1}^{k} d_i(x^i, y^i)^2}. \tag{6.6}$$

**Manifold Learning** Manifold learning is a sub-field of machine learning that uses tools from differential geometry to model high-dimensional datasets by mapping them to a low-dimensional latent space. This allows researchers to analyze the underlying structure of data and improve machine learning models by capturing the geometry of the data more accurately. Manifold learning is based on the assumption that most observed data can be encoded within a low-dimensional manifold (see Figure 6.1) embedded in a high-dimensional space [102]. This has seen applications in dimensionality reduction [225, 256], generative models [115, 89, 31], and graph structure learning for graph neural networks (GNNs) [258]. In all these application, the key is to find a topological representation as an abstract encoding that describes the data optimally for the downstream task.

**Constant Curvature Model Spaces** Euclidean space,

$$\mathbb{E}^{d_{\mathbb{E}}}_{K_{\mathbb{E}}} = \mathbb{R}^{d_{\mathbb{E}}} \tag{6.7}$$

is a flat space with curvature $K_{\mathbb{E}} = 0$. We note that in this context, the notation $d_{\mathbb{E}}$ is used to represent the dimensionality. In contrast, hyperbolic and spherical spaces possess negative and positive curvature, respectively. We define hyperboloids $\mathbb{H}^{d_{\mathbb{H}}}_{K_{\mathbb{H}}}$ as

$$\mathbb{H}^{d_{\mathbb{H}}}_{K_{\mathbb{H}}} = \{\mathbf{x}_p \in \mathbb{R}^{d_{\mathbb{H}}+1} : \langle \mathbf{x}_p, \mathbf{x}_p \rangle_{\mathcal{L}} = 1/K_{\mathbb{H}}\}, \tag{6.8}$$

where $K_{\mathbb{H}} < 0$ and $\langle \cdot, \cdot \rangle_{\mathcal{L}}$ is the Lorentz inner product

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} = -x_1 y_1 + \sum_{j=2}^{d_{\mathbb{H}}+1} x_j y_j, \quad \forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^{d_{\mathbb{H}}+1}, \tag{6.9}$$

and hyperspheres $\mathbb{S}^{d_{\mathbb{S}}}_{K_{\mathbb{S}}}$ as

$$\mathbb{S}^{d_{\mathbb{S}}}_{K_{\mathbb{S}}} = \{\mathbf{x}_p \in \mathbb{R}^{d_{\mathbb{S}}+1} : \langle \mathbf{x}_p, \mathbf{x}_p \rangle_2 = 1/K_{\mathbb{S}}\}, \tag{6.10}$$

where $K_{\mathbb{S}} > 0$ and $\langle \cdot, \cdot \rangle_2$ is the standard Euclidean inner product

$$\langle \mathbf{x}, \mathbf{y} \rangle_2 = \sum_{j=1}^{d_{\mathbb{S}}+1} x_j y_j, \ \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^{d_{\mathbb{S}}+1}, \tag{6.11}$$

Table 6.1 summarizes the key operators in Euclidean, hyperbolic, and spherical spaces with arbitrary curvatures. It is worth noting that the three model spaces discussed above can cover any curvature value within the range of $(-\infty, \infty)$. However, there is a potential issue with the hyperboloid and hypersphere becoming divergent as their respective curvatures approach zero. This results in the manifolds becoming flat, and their distance and metric tensors do not become Euclidean at zero-curvature points, which could hinder curvature learning. Therefore, stereographic projections are considered to be suitable alternatives to these spaces as they maintain a non-Euclidean structure and inherit many of the properties of the hyperboloid and hypersphere.

Table 6.1: Relevant operators (exponential maps and distances between two points) in Euclidean, hyperbolic, and spherical spaces with arbitrary constant curvatures.

| Space Model | $exp_{\mathbf{x}_p}(\mathbf{x})$ | $\mathfrak{d}(\mathbf{x}, \mathbf{y})$ |
|---|---|---|
| $\mathbb{E}$, Euclidean | $\mathbf{x}_p + \mathbf{x}$ | $\|\|\mathbf{x} - \mathbf{y}\|\|_2$ |
| $\mathbb{H}$, hyperboloid | $\cosh\left(\sqrt{-K_{\mathbb{H}}}\|\|\mathbf{x}\|\|\right)\mathbf{x}_p + \sinh\left(\sqrt{-K_{\mathbb{H}}}\|\|\mathbf{x}\|\|\right)\frac{\mathbf{x}}{\sqrt{-K_{\mathbb{H}}}\|\|\mathbf{x}\|\|}$ | $\frac{1}{\sqrt{-K_{\mathbb{H}}}} \operatorname{arccosh}\left(K_{\mathbb{H}}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\right)$ |
| $\mathbb{S}$, hypersphere | $\cos\left(\sqrt{K_{\mathbb{S}}}\|\|\mathbf{x}\|\|\right)\mathbf{x}_p + \sin\left(\sqrt{K_{\mathbb{S}}}\|\|\mathbf{x}\|\|\right)\frac{\mathbf{x}}{\sqrt{K_{\mathbb{S}}}\|\|\mathbf{x}\|\|}$ | $\frac{1}{\sqrt{K_{\mathbb{S}}}} \arccos\left(K_{\mathbb{S}}\langle \mathbf{x}, \mathbf{y} \rangle_2\right)$ |

**Constructing Product Manifolds** By combining Euclidean, hyperbolic, and spherical components, one obtains

$$\mathcal{P} = \mathbb{E}^{d_E} \times \left(\bigtimes_{j=1}^{n_H} \mathbb{H}^{d_j}_{K_j}\right) \times \left(\bigtimes_{k=1}^{n_S} \mathbb{S}^{d_k}_{K_k}\right), \tag{6.12}$$

a flexible latent space with closed-form geometry. Exponential maps and distances are obtained by applying each factor's formulas and aggregating via the Euclidean norm. Most manifolds lack closed-form geodesics or distances, necessitating expensive numerical solvers. Product manifolds of constant-curvature spaces strike a balance between expressive power and computational tractability, making them ideal for geometric deep learning and latent-space modeling.

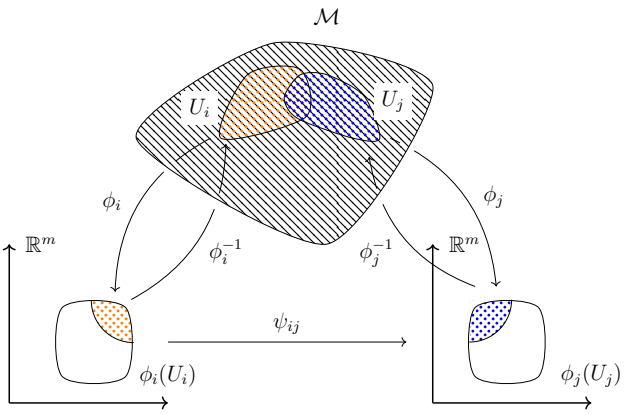


Figure 6.1: Schematic of a manifold $\mathcal{M}$ and open subsets $U_i$ and $U_j$. An open chart is a homeomorphism of an open subset of the manifold onto an open subset of the Euclidean hyperplane. Here, $\psi_{ij}$ is a transition function.

**Hausdorff and Gromov-Hausdorff Distances for Comparing Manifolds.** The Hausdorff distance between two subsets of a metric space refers to the greatest distance between any point on the first set and its closest point on the second set [150]. Given a metric space $X$ with metric $d_X$, and two subsets $A$ and $B$, we can define the *Hausdorff distance* between $A$ and $B$ in $X$ by

$$\mathrm{d}_{\mathrm{H}}^{X}(A,B) = \max\left(\sup_{a\in A}\inf_{b\in B} d_X(a,b),\, \sup_{b\in B}\inf_{a\in A} d_X(b,a)\right) \tag{6.13}$$

$$= \max\left(\sup_{a\in A} d_X(a,B),\, \sup_{b\in B} d_X(b,A)\right). \tag{6.14}$$

A priori this quantity may be infinite. Hence we will restrict to compact subsets $A$ and $B$. In this case, we can equivalently define $\mathrm{d}_{\mathrm{H}}(A,B)$ as the smallest real number $c \geqslant 0$ such that for every $a \in A$ and every $b \in B$ there exist $a' \in A$ and $b' \in B$ such that both $d_X(a,b')$ and $d_X(a',b)$ are at most $c$.

We note that the previous definition does not require any differentiable structures on $X$, $A$ and $B$. They can be merely metric spaces. This generality allows us to distinguish the metric properties of Euclidean, hyperbolic and spherical geometries beyond analytic notions such as curvature. However, the definition in Equation 6.13 only allows to compare spaces $A$ and $B$ that are embedded in a certain metric space $X$. The notion of distance that we shall consider is the Gromov-Hausdorff distance, which we define below in Equation 6.16.

Given a metric space $X$ and two isometric embeddings $f : A \to X$ and $g : B \to X$, we define

$$\mathrm{d}_{\mathrm{H}}^{X,f,g}(A, B) = \mathrm{d}_{\mathrm{H}}^{X}(f(A), g(B)). \tag{6.15}$$

Now, given two metric spaces $A$ and $B$, we denote by $\mathrm{ES}(A, B)$ (standing for "embedding spaces of $A$ and $B$") as the triple $(X, f, g)$ where $X$ is a metric space and $f : A \to X$ and $g : B \to X$ are isometric embeddings. We define the *Gromov-Hausdorff distance between $A$ and $B$* as:

$$\mathrm{d}_{\mathrm{GH}}(A, B) = \inf_{(X,f,g)\in \mathrm{ES}(A,B)} \mathrm{d}_{\mathrm{H}}^{X,f,g}(f(A), g(B)). \tag{6.16}$$

## 6.3 Neural Latent Geometry Search: Latent Product Manifold Inference

In this section, we leverage ideas discussed in Section 6.2 to introduce a principled way to find the optimal latent product manifold. First, we introduce the problem formulation of NLGS. Next, we outline the strategy used to compute the Gromov-Hausdorff distance between product manifolds, and we discuss how this notion of similarity can be used in practice to construct a graph search space of latent geometries. Lastly, we explain how the Gromov-Hausdorff-informed graph search space can be used to perform NLGS via BO.

### 6.3.1 Problem Formulation

The problem of NLGS can be formulated as follows. Given a search space $\mathfrak{G}$ denoting the set of all possible latent geometries, and the objective function $L_{T,A}(g)$ which evaluates the performance of a given geometry $g$ on a downstream task $T$ for a machine learning model architecture $A$, the objective is to find an optimal latent geometry $g^*$:

$$g^* = \arg\min_{g\in\mathfrak{G}} L_{T,A}(g). \tag{6.17}$$

In our case we model the latent space geometry using product manifolds. Hence we effectively restrict Equation 6.17 to finding the optimal product manifold signature:

$$n_{\mathcal{P}}^*, \{d_i\}_{i\in n_{\mathcal{P}}^*}^*, \{K_i\}_{i\in n_{\mathcal{P}}^*}^* = \arg\min_{n_{\mathcal{P}}\in\mathbb{Z}, d_i\in\mathbb{Z}, K_i\in\mathbb{R}} L_{T,A}(n_{\mathcal{P}}, \{d_i\}_{i\in n_{\mathcal{P}}}, \{K_i\}_{i\in n_{\mathcal{P}}}), \tag{6.18}$$

where $n_{\mathcal{P}}$ is the number of model spaces composing the product manifold $\mathcal{P}$, $\{d_i\}_{i\in n_{\mathcal{P}}}$ are the dimensions of each of the model spaces of constant curvature, and $\{K_i\}_{i\in n_{\mathcal{P}}}$ their respective curvatures. We further simplify the problem by setting $d_i = 2, \forall i$, and by restricting ourselves to $K_i \in \{-1, 0, 1\}$, in order to limit the size of the hypothesis space.

### 6.3.2 Quantifying the Difference Between Product Manifolds

**Motivation.** From a computational perspective, we can think of the Hausdorff distance as a measure of dissimilarity between two point sets, each representing a discretized version of the two underlying continuous manifolds we wish to compare. [253] proposed an efficient algorithm to compute the exact Hausdorff distance between two point sets with nearly-linear complexity leveraging early breaking and random sampling in place of scanning. However, the original algorithm assumes that both point sets live in the same space and have the same dimensionality, which is a limiting requirement. Gromov-Hausdorff distances, as opposed to usual Hausdorff distances, allow us to measure the distance between two metric spaces that a priori are not embedded in a common bigger ambient space. However, this has the caveat that they are not computable. For instance, for our application we must calculate the distance between each pair of the following three spaces: $\mathbb{E}^n$, $\mathbb{S}^n$ and $\mathbb{H}^n$. However, $\mathbb{S}^n$ does not isometrically embed in $\mathbb{E}^n$ and hence in order to compare $\mathbb{E}^n$ and $\mathbb{S}^n$, we must work in a higher dimensional ambient space such as $\mathbb{E}^{n+1}$. In the case of $\mathbb{H}^n$, finding an embedding to a Euclidean space is more complicated and is described in Section 6.6.2 ($\mathbb{H}^n$ will be embedded isometrically into $\mathbb{E}^{6n-6}$). However, in this process there will be choices made about which embeddings to consider (in particular, we cannot exactly compute the infimum that appears in the definition of Gromov-Hausdorff distance in Equation 6.16). Likewise, using the original algorithm by [253] it is not possible to compute the Hausdorff distance between product manifolds based on an unequal number of model spaces, for instance, there is no way of computing the distance between $\mathbb{E}^n$ and $\mathbb{E}^n \bigtimes \mathbb{H}^n$. In this section we give an upper bound for $\mathrm{d}_{\mathrm{GH}}(\mathbb{E}^n, \mathbb{S}^n)$, and then describe an algorithm to give an upper bound for $\mathrm{d}_{\mathrm{GH}}(\mathbb{E}^n, \mathbb{H}^n)$ and $\mathrm{d}_{\mathrm{GH}}(\mathbb{S}^n, \mathbb{H}^n)$.

**Strategy.** The spaces $\mathbb{E}^n$ and $\mathbb{S}^n$ isometrically embed into $\mathbb{E}^{6n-6}$ in many ways. This may seem redundant because both spaces already embed in $\mathbb{E}^{n+1}$. However, the interest of considering this higher dimensional Euclidean space is that $\mathbb{H}^n$ will also isometrically embed into it. Crucially, this will provide a common underlying space in which to compute Hausdorff distances between our geometries $\mathbb{E}^n$, $\mathbb{S}^n$ and $\mathbb{H}^n$, which will lead to an estimation of their mutual Gromov-Hausdorff distance. The embedding of $\mathbb{H}^n$ into $\mathbb{E}^{6n-6}$ that we shall describe appears in [141] and is made explicit in [27]. We also refer the reader to the exposition [32, Chapter 5], which puts this results in a broader context while also summarising related advances on the topic of isometrically embedding homogeneous spaces into higher dimensional ones. For $n = 2$, we name this embedding $F : \mathbb{H}^2 \to \mathbb{E}^6$ (Section 6.6.2). For simplicity, in our experiments in Section 6.4, we will work with product manifolds generated based on constant curvature model spaces of dimension $n = 2$.

Now we can summarise our strategy to estimate $\mathrm{d_{GH}}(B_{\mathbb{E}^2}, B_{\mathbb{H}^2})$ as follows (for $\mathrm{d_{GH}}(B_{\mathbb{H}^2}, B_{\mathbb{S}^2})$ it will be entirely analogous). The first step consists of approximating our infinite smooth spaces by finite discrete ones. For this, we consider several collections of points $\{P_i\}_{i\in I}$ in $\mathbb{E}^2$ that are sufficiently well distributed. The exponential map can be applied to the collection of points $\exp : T_0\mathbb{H}^2 \cong \mathbb{R}^2 \to B_{\mathbb{H}^2}$ to get several collections of points $Q$ in $B_{\mathbb{H}^2}$ (again, well distributed by construction). In addition, we will consider several isometric embeddings $f_k : B_{\mathbb{E}^2} \to \mathbb{R}^6$. Hence, we take

$$\mathrm{d_{GH}}(B_{\mathbb{E}^2}, B_{\mathbb{H}^2}) \approx \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6, f_k, F}(P_i, Q_j) = \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6}(f_k(P_i), F(Q_j)). \tag{6.19}$$

In Section 6.6, we gradually unravel the previous formula and give explicit examples of the involved elements. In particular, in Section 6.6.1 we explain how to generate points in the balls of radius one, and in Section 6.6.2 how to describe the isometric embedding $F : B_{\mathbb{H}^2} \to \mathbb{E}^6$. The results obtained for the Gromov-Hausdorff distances between product manifolds are used to generate the graph search space introduced in the next section.

### 6.3.3 The Gromov-Hausdorff-Informed Graph Search Space

Table 6.2: Estimated Gromov-Hausdorff distances (up to two decimal places) between model spaces and corresponding edge weights in the graph search space.

| Comparison Pair | $d_{\mathrm{GH}}(\cdot)$ | $w_{(\cdot)}$ |
|---|---|---|
| $(\mathbb{E}^2, \mathbb{S}^2)$ | 0.23 | 4.35 |
| $(\mathbb{E}^2, \mathbb{H}^2)$ | 0.77 | 1.30 |
| $(\mathbb{S}^2, \mathbb{H}^2)$ | 0.84 | 1.20 |

**Gromov-Hausdorff Edge Weights.** In Section 6.2, we used Cartesian products of constant curvature model spaces to generate candidate latent geometries. We now turn our attention to constructing a search space to find the optimal latent geometry. To do so, we first consider all possible combinations of product manifolds based on a given number of model spaces, represented by $n_s$. Furthermore, we denote the total number of products (model spaces) used to form the product manifold $\mathcal{P}$ with $n_p$. Conceptually, $n_s$ is the number of model space *types* used, while $n_p$ refers to the overall number, or *quantity*, of model spaces that form the resulting product space. For instance, if only the Euclidean plane, the hyperboloid, and the hypersphere are taken into account, then $n_s = 3$. If all product manifold combinations are considered, the number of elements in the search space increases

to $\sum_{i=1}^{n_p} n_s^i$. However, we assume *commutativity* for latent product manifolds, implying that the output of a trained neural network with a latent geometry $\mathcal{M}_i \times \mathcal{M}_j$ should be the same as that with $\mathcal{M}_j \times \mathcal{M}_i$. We refer to this concept as the *symmetry of parameterization*, as neural networks can rearrange the weight matrices of their neurons to achieve optimal performance for two equivalent latent manifolds. This assumption reduces the search space from growing exponentially to $\mathcal{O}(n_p^2)$, assuming three constant curvature model spaces are considered (see Section 6.6.5 for a more complete explanation).

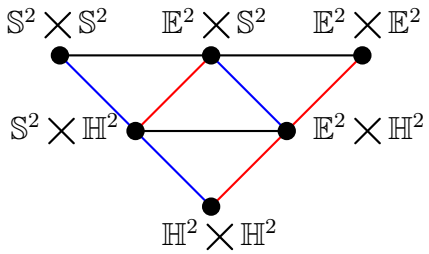


Figure 6.2: Slice of the graph search space for latent geometries of dimension 4: product manifolds obtained using 2 models spaces of dimension 2. The graph edges are shown in different colours to depict a different degree of connectivity (black: $w_{\mathbb{E}^2,\mathbb{S}^2}$, red: $w_{\mathbb{E}^2,\mathbb{H}^2}$, blue: $w_{\mathbb{S}^2,\mathbb{H}^2}$), this is determined by the inverse of the Gromov-Hausdorff distance between the different product manifolds.

We model the search space as a graph over which we perform BO. In our case, we focus on a *weighted* and *undirected* graph. We consider a setting in which the function $\mathfrak{f}(\cdot)$ to minimize is defined over the nodes of the graph, and the objective of using BO is to find the node associated to the minimum value $\mathbf{v}^* = \arg\min_{\mathbf{v} \in V} \mathfrak{f}(\mathbf{v})$.

In our setting, each node in the graph represents a different latent space product manifold, and the value at that node is the validation set performance of a neural network architecture using this latent geometry. We use the inverse of the Gromov-Hausdorff distance between product manifolds to obtain edge weights. We denote by $w_{\mathcal{M}_1,\mathcal{M}_2}$ (edge weights) the inverse Gromov-Hausdorff distance between model spaces $\mathcal{M}_1$ and $\mathcal{M}_2$. The approximate values of these coefficients are presented in Table 6.2. In particular, the Gromov-Hausdorff distance $\mathrm{d_{GH}}(\mathbb{E}^2, \mathbb{S}^2)$ used to compute $w_{\mathbb{E}^2,\mathbb{S}^2}$ is derived analytically in Section 6.6.3, while the remaining coefficients are obtained through computational approximations of Equation 6.19, which are depicted in Section 6.6.4. Further, the Gromov-Hausdorff distance between manifolds of different dimensions is one (see Section 6.2).

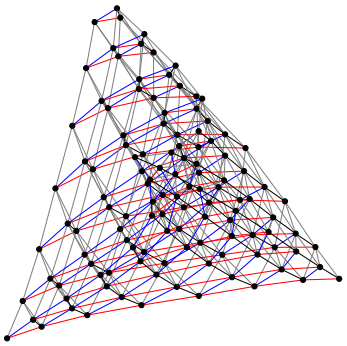

Figure 6.3: Example graph search space for product manifolds composed of up to seven model spaces. Manifolds of different dimensionality are connected with edges coloured grey. Node labels have been omitted for visual clarity.

In order to impose additional structure on the search space, we only allow connections between nodes corresponding to product manifolds which differ by a single model space. For example, within the same dimension, $\mathbb{S}^2 \times \mathbb{H}^2$ and $\mathbb{E}^2 \times \mathbb{H}^2$ would be connected with edge weighting $w_{\mathbb{E}^2,\mathbb{S}^2}$ while $\mathbb{S}^2 \times \mathbb{H}^2$ and $\mathbb{E}^2 \times \mathbb{E}^2$ would have no connection in the graph. Furthermore, product manifolds in different dimensions follow the same rule. For instance, we would have a connection of strength one between $\mathbb{E}^2 \times \mathbb{H}^2$ and $\mathbb{E}^2 \times \mathbb{H}^2 \times \mathbb{H}^2$, but no connection between, for instance, $\mathbb{E}^2$ and $\mathbb{E}^2 \times \mathbb{H}^2 \times \mathbb{H}^2$ or $\mathbb{S}^2$ and $\mathbb{E}^2 \times \mathbb{H}^2$. This construction induces a sense of directionality into the graph and generates clusters of product manifolds of the same dimension. Finally, it should be mentioned that in practice there are only four edge weights. For example, the connectivity strength between $\mathbb{E}^2 \times \mathbb{H}^2$ and $\mathbb{H}^2 \times \mathbb{H}^2$ is $w_{\mathbb{E}^2,\mathbb{H}^2}$ since $\mathrm{d_{GH}}(\mathbb{E}^2 \times \mathbb{H}^2, \mathbb{H}^2 \times \mathbb{H}^2) = \mathrm{d_{GH}}(\mathbb{E}^2, \mathbb{H}^2)$ given that $\mathrm{d_{GH}}(\mathbb{H}^2, \mathbb{H}^2) = 0$. Visual representations of the graph search space can be found Figures 6.2 and 6.3. Note that both figures use the same colour scheme, and in Figure 6.3 there is an increase in the dimensionality of the product manifolds from left to right (e.g. on the top left corner, one can see a triangle corresponding to the three model spaces). We refer the reader to Section 6.6.6 for additional visualizations.

**Bayesian Optimization over the Graph Search Space.** We aim to find the minimum point within the graph search space through the use of BO (see Section 6.7). While performing BO on graphs allows us to search for the minimum over a categorical search space, some of the key notions used in BO over Eucledian spaces do not translate directly when

operating over a graph. For example, notions of similarity or "closeness" which are trivially found in Eucledian space through the $\ell_1$ or $\ell_2$ norms require more careful consideration when using graphs.

In our setting, we employ a *diffusion kernel* [248, 164] to compute the similarity between the nodes in the graph. The diffusion kernel is based on the eigendecomposition of the graph Laplacian $\mathbf{L} \in \mathbb{R}^{N \times N}$, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is the degree matrix of the graph. In particular, the eigendecomposition of the Laplacian is given by $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$, where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ is a matrix containing eigenvectors as columns and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix containing increasingly ordered eigenvalues. The *covariance matrix* used to define the GP over the graph is given by

$$\mathcal{K}(\mathcal{V}, \mathcal{V}) = \mathbf{U} e^{-\beta \mathbf{\Lambda}} \mathbf{U}^{\mathrm{T}}, \tag{6.20}$$

where $\beta$ is the *lengthscale* parameter. Our approach is therefore conceptually similar to that of [204], which employ a diffusion kernel to carry out a NAS procedure on a graph Cartesian product. We highlight, however, that the main contribution of this work is the construction of the graph search space, and we employ this search procedure to showcase the suitability of our method in finding the optimal latent product manifold. For reference, we note that other works employing Bayesian optimization in graph-related settings include [74, 69, 184, 226, 75, 263].

## 6.4 Experimental Setup and Results

In this section, a detailed outline of the experimental results is provided. It comprises of experiments performed on synthetic datasets, as well as experimental validation on custom-designed real-world datasets obtained by morphing the latent space of mixed-curvature autoencoders and latent graph inference. The results demonstrate that the Gromov-Hausdorff-informed graph search space can be leveraged to perform NLGS across a variety of tasks and datasets.

### 6.4.1 Synthetic Experiments on Product Manifold Inference

In order to evaluate the effectiveness of the proposed graph search space, we conduct a series of tests using synthetic data. We wish to present a setting for which the latent optimal product manifold is known by construction. To do so, we start by generating a random vector $\mathbf{x}$ and mapping it to a "ground truth" product manifold $\mathcal{P}_T$, which we choose arbitrarily, using the corresponding exponential map. The resulting projected vector is then decoded using a neural network with frozen weights, $f_\theta$, to obtain a reference signal $y^{\mathcal{P}_T}$,

which we wish to recover using NLGS. To generate the rest of the dataset, we then consider the same random vector but map it to a number of other product manifolds $\{\mathcal{P}_i\}_{i\in n_{\mathcal{P}}}$ with the same number of model spaces as $\mathcal{P}_T$ but different signatures. Decoding the projected vector through the same neural network yields a set of signal, $\{y^{\mathcal{P}_i}\}_{i\in n_{\mathcal{P}}}$. We use the aforementioned signals, to set the value associated with each node of the graph search space to be $MSE(y^{\mathcal{P}_T}, y^{\mathcal{P}_i})$. In this way, our search algorithm should aim to find the node in the graph that minimizes the error and hence find the latent geometry that recovers the original reference signal.

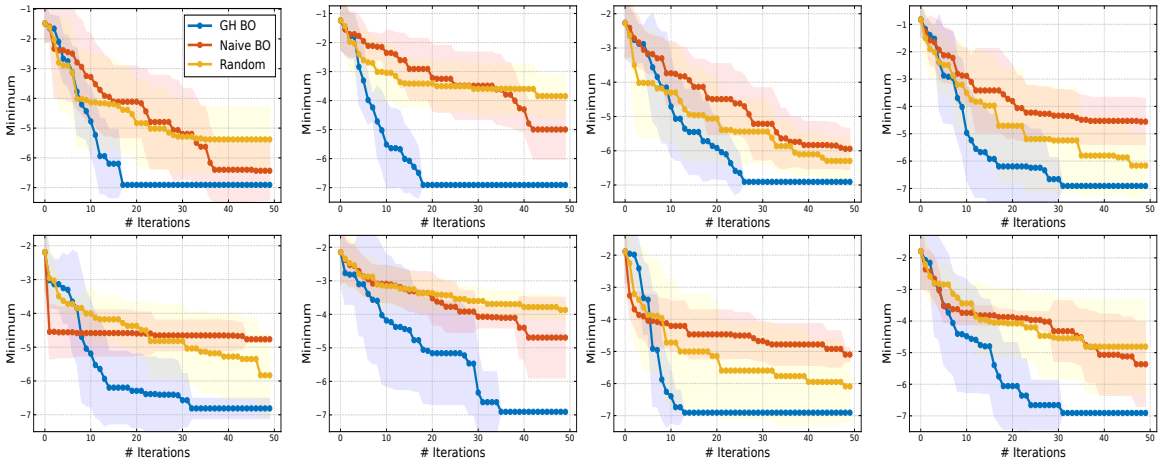


Figure 6.4: Results (mean and standard deviation over 10 runs) for candidate latent geometries involving product manifolds composed of 13 model spaces. For each plot a different ground truth product manifold $\mathcal{P}_T$ is used to generate the reference signal.

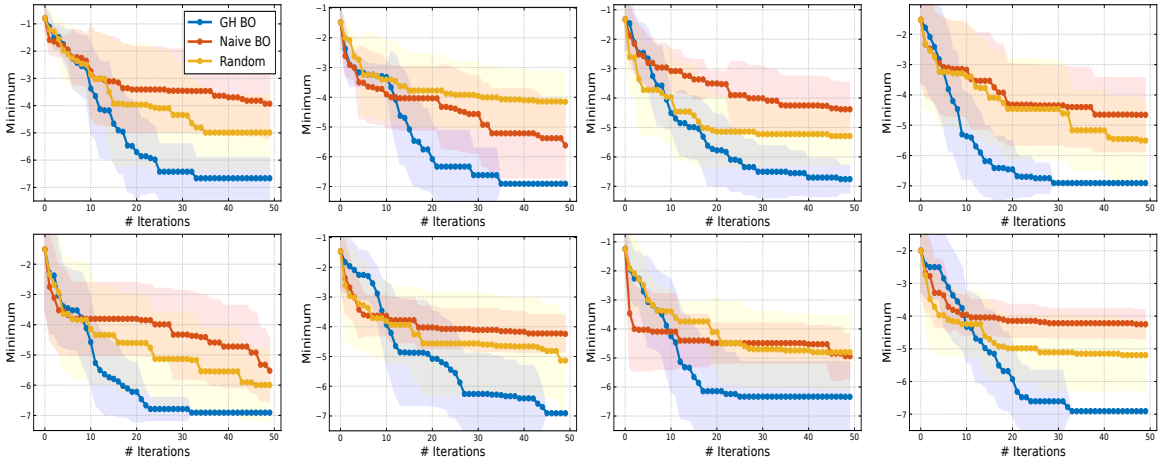


Figure 6.5: Results (mean and standard deviation over 10 runs) for candidate latent geometries involving product manifolds composed of 20 model spaces. Same setup as above.

Our method consists in using BO over our Gromov-Hausdorff-informed search space.

We compare it against random search, and what we call "Naive BO", which performs BO over a fully-connected graph which disregards the Gromov-Hausdorff distance between candidate latent geometries. The figures presented, namely Figures 6.4 and 6.5, display the results. Each plot illustrates the performance of the algorithms and baselines as they select different optimal latent product manifolds denoted as $\mathcal{P}_T$. Notably, the figures demonstrate that the algorithm utilizing the Gromov-Hausdorff-informed search space surpasses all other baselines under consideration and consistently achieves the true function minimum. Figure 6.4 generally requires fewer iterations compared to Figure 6.5 to attain the global minimum, primarily due to the smaller size of the search space. It is important to note that our benchmark solely involves the same algorithm but with a change in the search space to a topologically uninformative one. This modification allows us to evaluate the impact of incorporating this information into the optimization process. We have not considered other benchmarks such as evolutionary algorithms since they always rely on a notion of distance between sampled points. Furthermore, as there are no existing algorithms to compute the distance between arbitrary product manifolds, we have not included these methods in our benchmark as they would simply converge to random search. All results are shown on a log scale, and we apply a small offset of $\varepsilon = 10^{-3}$ to the plots to avoid computing $\log(0)$ when the reference signal is found.

### 6.4.2 Experiments on Real-World Datasets

To further validate our method, we conduct additional tests using image and graph datasets. Specifically, we focus on image reconstruction and node classification tasks.

#### Image Reconstruction with Autoencoder

We consider four well-known image datasets: MNIST [80], CIFAR-10 [167], Fashion MNIST [272] and eMNIST [68]. Some of these datasets have been shown to benefit from additional topological priors, see [157] and [194]. We use an autoencoder, detailed more thoroughly in Section 6.8, to encode the image in a low dimensional latent space and then reconstruct it based on its latent representation. To test the performance of different latent geometries, we project the latent vector onto the product manifold being evaluated and use the reconstruction loss at the end of training as the reference signal to use in the graph search space. We consider a search space size of $n_p = 7$ for MNIST and $n_p = 8$ for CIFAR-10, Fashion MNIST and eMNIST. The results are displayed in Figure 6.6. In line with previous experiments, the Gromov-Hausdorff-informed search graph enables us to find better solutions in a smaller amount of evaluations.

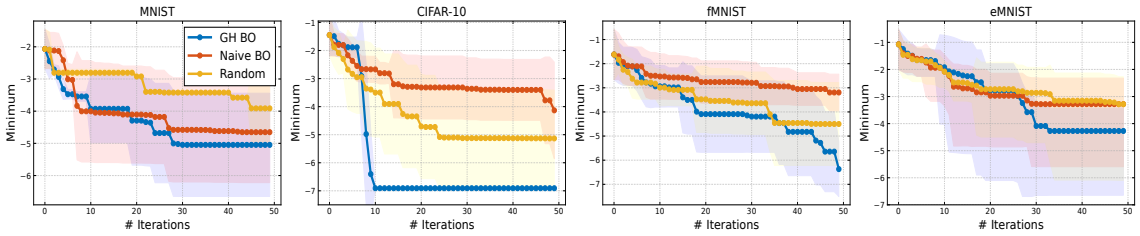


Figure 6.6: Results (mean and standard deviation over 10 runs) for image reconstruction tasks.

### Latent Graph Inference

Finally, we consider searching for the optimal product manifold for a graph neural network node classification task using latent graph inference. In particular, we build upon previous work by [233] which used product manifolds to produce richer embeddings spaces for latent graph inference. We consider the Cora [240] and Citeseer [110] citation network datasets, and a search space consisting of product manifolds of up to seven model spaces $n_p = 7$. The aim is to find the latent space which gives the best results for the node classification problem using minimal query evaluations. Performing BO alongside the Gromov-Hausdorff informed search space gives a clear competitive advantage over Naive BO and random search in the case of Cora. For Citeseer, the experiments do not give such a clear-cut improvement, which can be attributed to the lack of smoothness of the signal on the graph search space and its incompatibility with the intrinsic limitations of the diffusion kernel.

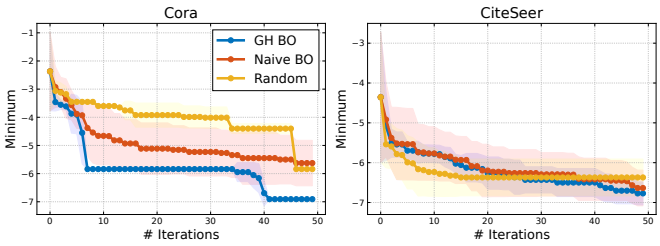


Figure 6.7: Results (mean and standard deviation over 10 runs) for latent graph inference datasets.

## 6.5 Conclusion

In this work, we have introduced *neural latent geometry search (NLGS)*, a novel problem formulation that consists in finding the optimal latent space geometry of machine learning algorithms using minimal query evaluations. In particular, we have modeled the latent space using product manifolds based on Cartesian products of constant curvature model

spaces. To find the optimal product manifold, we propose using Bayesian Optimization over a graph search space. The graph is constructed based on a principled measure of similarity, utilizing the Gromov-Hausdorff distance from metric geometry. The effectiveness of the proposed method is demonstrated through our experiments conducted on a variety of tasks, based on custom-designed synthetic and real-world datasets.

**Limitations and Future Work.** While the NLGS framework is general, we have restricted ourselves to using product manifolds of constant curvature model spaces to model the geometry of the latent space. Furthermore, we have only considered curvatures $\{-1, 0, 1\}$, and model spaces of dimension two in order for the optimization problem to be tractable. In future research, there is potential to explore alternative approaches for modeling the latent space manifold. Additionally, the field of Bayesian Optimization over graphs is still in its early stages. Incorporating recent advancements in the area of kernels on graphs [29, 285] could lead to improved performance. In our current research, our emphasis is on introducing NLGS and providing an initial solution under a set of simplifying assumptions related to the potential latent manifolds available and the optimization algorithm. The investigation of the impact of using different similarity measures to compare latent structures also remains a subject for future research.

## 6.6 Further Details on the Gromov-Hausdorff Algorithm for Neural Latent Geometry Search

As discussed in Section 6.3, there are limitations to the Hausdorff distance as a metric for comparing point sets that represent discretized versions of continuous manifolds. The original algorithm by [253] for computing the Hausdorff distance assumes that the point sets reside in the same space with the same dimensionality, which is a limiting requirement. However, the Gromov-Hausdorff distance proposed in this work allows us to compare metric spaces that are not embedded in a common ambient space, but it is not exactly computable. We will focus on providing an upper bound for $\mathrm{d_{GH}}(\mathbb{E}^n, \mathbb{S}^n)$ and describing an algorithm for obtaining upper bounds for $\mathrm{d_{GH}}(\mathbb{E}^n, \mathbb{H}^n)$ and $\mathrm{d_{GH}}(\mathbb{S}^n, \mathbb{H}^n)$ too. As previously mentioned, $\mathbb{E}^n$ and $\mathbb{S}^n$ can be isometrically embedded into $\mathbb{E}^{6n-6}$ in many ways, which provides a common underlying space for computing Hausdorff distances between $\mathbb{E}^n$, $\mathbb{S}^n$, and $\mathbb{H}^n$. The embeddings of different spaces require making choices: in particular, there is no way to exactly compute the infimum in the definition of Gromov-Hausdorff distance given in Section 6.2:

$$\mathrm{d_{GH}}(A, B) = \inf_{(X,f,g)\in \mathrm{ES}(A,B)} \mathrm{d_H}^{X,f,g}(f(A), g(B)). \tag{6.21}$$

so we resort to numerical approximations which are later described in this section.

To estimate the mutual Gromov-Hausdorff distance between the infinite smooth spaces $\mathbb{E}^2$ and $\mathbb{H}^2$, the first step is to approximate them using finite discrete versions. This is done by considering well-distributed collections of points in $\mathbb{E}^2$ and $\mathbb{H}^2$, obtained by applying the exponential map to a collection of points in $\mathbb{R}^2$. Multiple isometric embeddings of $\mathbb{E}^2$ into $\mathbb{R}^6$ are also considered. The estimation is obtained by computing the minimum of the Hausdorff distance between the point sets obtained from the collections in $\mathbb{R}^6$ and the isometric embedding of $\mathbb{H}^2$ into $\mathbb{R}^6$. The same applies for spherical space.

### 6.6.1 Generating Points in the Corresponding Balls of Radius One

All our computations will be done in dimension two, so we will simply precise how to generate points in $B_{\mathbb{E}^2}$, $B_{\mathbb{S}^2}$ and $B_{\mathbb{H}^2}$. For $B_{\mathbb{E}^2}$ and $B_{\mathbb{S}^2}$, we will use very elementary trigonometry. For $B_{\mathbb{H}^2}$, we will need an explicit description of the exponential map of $\mathbb{H}^2$. Using the descriptions

$$B_{\mathbb{E}^2} = \{(r\cos(t), r\sin(t)) : r \in [0,1], t \in [0, 2\pi)\}, \tag{6.22}$$

and

$$B_{\mathbb{S}^2} = \{(\sin(\beta)\cos(\alpha), \sin(\beta)\sin(\alpha), \cos(\beta)) : \alpha \in [0, 2\pi), \beta \in [0,1]\}, \tag{6.23}$$

it will be easy to generate collections of points in $B_{\mathbb{E}^2}$ and $B_{\mathbb{S}^2}$. As we anticipated above in the outline of our strategy to estimate $\mathrm{d_{GH}}(B_{\mathbb{E}^2}, B_{\mathbb{H}^2})$, in order to give explicit well-distributed collection of points in $B_{\mathbb{H}^2}$, it is enough to give a well-distributed collection of points in $B_{\mathbb{E}^2}$ and consider its image under the exponential map $\exp_0 : B_{\mathbb{E}^2} \to B_{\mathbb{H}^2}$, where we have identified $B_{\mathbb{E}^2}$ with the ball of radius one of $\mathbb{R}^2 \cong T_0\mathbb{H}^2$, i.e. the tangent space of $\mathbb{H}^2$ at the point $0$. However, we should remark that our description of $\mathbb{H}^n$ will not be any of the three most standard ones: namely the *hyperboloid model*, the *the Poincaré ball model*, or the *upper half-plane model*. We describe $\mathbb{H}^n$ as the differentiable manifold $\mathbb{R}^n$ (of Cartesian coordinates $x, y_1, \ldots, y_{n-1}$) equipped with the Riemannian metric $g_{-1} = \mathrm{d}x^2 + e^{2x}(\mathrm{d}y_1^2 + \cdots \mathrm{d}y_{n-1}^2)$. This metric is complete and has constant sectional curvature of -1, which implies that $(\mathbb{R}^n, g_{-1})$ is isometric to $\mathbb{H}^n$. In the hyperboloid model of $\mathbb{H}^2$, we view this space as a submanifold of $\mathbb{R}^3$ (although with a distorted metric, not the one induced from the ambient $\mathbb{R}^3$). In this model of $\mathbb{H}^2$, one can explicitly describe by the following assignment $\exp_0 : \mathbb{R}^2 \to \mathbb{H}^2$ by

$$(x,y) \mapsto \left( \frac{x \sinh(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}, \frac{y \sinh(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}, \cosh(\sqrt{x^2+y^2}) \right). \tag{6.24}$$

We use this explicit formula of the exponential map with coordinates in the hyperboloid model of $\mathbb{H}^2$, to give a explicit formula for the exponential map with coordinates in the model of $\mathbb{H}^2$ that we introduced above, denoted by $(\mathbb{R}^2, g_{-1})$. In order to change coordinates from the hyperboloid model to the Poincaré ball model, we use the following isometry (which is well-known and can be thought of as a hyperbolic version of the stereographic projection from $\mathbb{S}^2$ to $\mathbb{R}^2$):

$$(x,y,z) \mapsto \left( \frac{x}{1+z}, \frac{y}{1+z} \right). \tag{6.25}$$

To change coordinates from the Poincaré ball model to the upper half-plane model, one can use the following isometry:

$$(x,y) \mapsto \left( \frac{-2y}{(x-1)^2+y^2}, \frac{1-x^2-y^2}{(x-1)^2+y^2} \right). \tag{6.26}$$

The previous assignment comes from the standard Möbius transformation $z \mapsto \frac{z+i}{iz+1}$ that identifies the Euclidean ball of radius one of $\mathbb{C}$ and the upper half plane of $\mathbb{C}$ as Riemann surfaces (which we give explicitly in this case, although it is known to exist and to be unique by the classical Riemann mapping theorem). Finally, to go from coordinates in the upper half-plane model of $\mathbb{H}^2$ to our model $(\mathbb{R}^2, g_{-1})$, we use the isometry

$$(x,y) \to (-\log y, x). \tag{6.27}$$

### 6.6.2 Embedding of $\mathbb{H}^n$ into $\mathbb{E}^{6n-6}$

We want to define an isometric embedding of $B^1_{\mathbb{H}^n}$ into $\mathbb{E}^{6n-6}$ ($F$ from Section 6.3.2). This higher dimensional space is our candidate to fit in the three geometries $\mathbb{E}^n$, $\mathbb{H}^n$ and $\mathbb{S}^n$ to compute their Hausdorff dimensions as subspaces of $\mathbb{E}^{6n-6}$ and hence estimate their Gromov-Hausdorff distances. Before describing such embedding, we introduce several preliminary auxiliary functions.

Let $\chi(t) = \sin(\pi t) \cdot e^{-\sin^{-2}(\pi t)}$ for non-integer values of $t$. A priori, the inverse of $\sin(0) = 0$ does not make sense but since $\lim_{t\to 0^+} \chi(t) = \lim_{t\to 0^-} \chi(t) = 0$, we can set $\chi(0) = 0$ so it is still continuous. In fact, it is smooth and, in particular, integrable. We can say the same at all points when $t$ is an integer, so we set $\chi(t) = 0$ for all integers $t$ and we obtain an smooth function $\chi$ defined on $\mathbb{R}$.

$$A = \int_0^1 \chi(t)dt. \tag{6.28}$$

We also define

$$\psi_1(x) = \sqrt{\frac{1}{A} \cdot \int_0^{1+x} \chi(t)\,\mathrm{d}t}, \tag{6.29}$$

and

$$\psi_2(x) = \sqrt{\frac{1}{A} \cdot \int_0^{x} \chi(t)\,\mathrm{d}t}. \tag{6.30}$$

We set $c$ to be the constant

$$c = 2\max\left\{G_1, G_2\right\}, \tag{6.31}$$

defined in terms of the following

$$G_1 = \left\| \frac{\mathrm{d}}{\mathrm{d}x}\left(\sinh(x)\cdot\psi_1(x)\right)\right\|_{L^\infty[-2,2]}, \tag{6.32}$$

$$G_2 = \left\| \frac{\mathrm{d}}{\mathrm{d}x}\left(\sinh(x)\cdot\psi_2(x)\right)\right\|_{L^\infty[-2,2]}, \tag{6.33}$$

$$h(x,y) = \frac{\sinh(x)}{c}\Big(\psi_1(x)\cos(c\cdot y), \psi_1(x)\sin(c\cdot y), \psi_2(x)\cos(c\cdot y), \psi_2(x)\sin(c\cdot y)\Big), \tag{6.34}$$

$$\psi(x,y) = \left(\sinh^{-1}(ye^x), \log(\sqrt{e^{-2x}+y^2})\right). \tag{6.35}$$

We also define $f_0(x,y) = \left(\int_0^{\sinh^{-1}(ye^x)} \sqrt{1-\varepsilon(t)^2}\mathrm{d}t, \log(\sqrt{e^{-2x}+y^2}), h(\psi(x,y))\right)$, with $\varepsilon$ being

$$\varepsilon = \frac{G_1^2+G_2^2}{c^2}. \tag{6.36}$$

This way, we can set

$$f(x,y_1,\ldots,y_{n-1}) = \frac{1}{\sqrt{n-1}}\big(f_0(x,\sqrt{n-1}y_1),\ldots,f_0(x,\sqrt{n-1}y_{n-1}\big).$$

Recall that in our case, we use the function $F(\cdot) = f(n=2,\cdot)$ to map the set of points $Q$ in $\mathbb{H}^2$ to $\mathbb{R}^6$, and for computing

$$\mathrm{d}_{\mathrm{GH}}(B_{\mathbb{E}^2}, B_{\mathbb{H}^2}) \approx \min_{i,j,k} \mathrm{d}_{\mathrm{H}}^{\mathbb{R}^6, f_k, F}(P_i, Q_j) = \min_{i,j,k} \mathrm{d}_{\mathrm{H}}^{\mathbb{R}^6}(f_k(P_i), F(Q_j)). \tag{6.37}$$

### 6.6.3 Estimation of the Gromov-Hausdorff distance between the Euclidean and Spherical Spaces

In this section, we give the analytical derivation of the Gromov-Hausdorff distance between the Euclidean space $\mathbb{E}^n$ and the Spherical space $\mathbb{S}^n$. We first explain the case $n = 1$, where we can visually understand the situation in a better way because all distances will be measured in $\mathbb{E}^2$. Afterwards, we replicate the same argument for arbitrary $n$. Recall that the Spherical model $\mathbb{S}^n$ can be described as the metric subspace of $\mathbb{E}^{n+1}$ corresponding to the Euclidean sphere of radius one. Hence, as a subspace of $\mathbb{R}^{n+1}$, it corresponds to $\{(x_0, \ldots, x_n) : \sum_{i=0}^{n} x_i^2 = 1\}$.

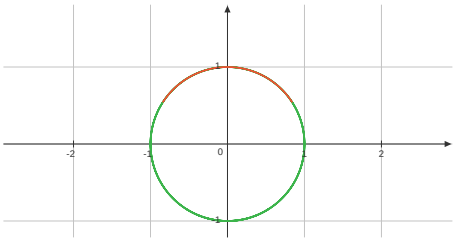


Figure 6.8: The one-dimensional model of spherical geometry $\mathbb{S}^1$ isometrically embedded in $\mathbb{R}^2$.

The ball of radius one of $\mathbb{S}^1$, denoted by $B_{\mathbb{S}^1}$, is highlighted in red. Our estimation of the Gromov-Hausdorff distance between $\mathbb{E}^1$ and $\mathbb{S}^1$ is motivated by the following observation. In the $y$-axis, the red arc ranges from $\frac{1+\cos(1)}{2}$ to 1. If we consider the following blue

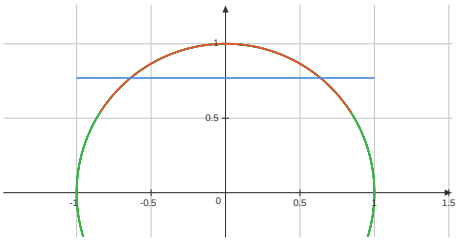


Figure 6.9: Comparison between $B_{\mathbb{E}^1}$ (in red) and $B_{\mathbb{S}^1}$ (in blue) inside $\mathbb{E}^2$.

segment, representing the ball of radius one of $\mathbb{E}^1$ (denoted by $B_{\mathbb{E}^1}$), we get

$$\sup_{x \in B_{\mathbb{S}^1}} d_{\mathbb{E}^2}(x, B_{\mathbb{E}^1}) = \frac{1 - \cos(1)}{2} \approx 0.23. \tag{6.38}$$

However, it can be seen that $\sup_{x \in B_{\mathbb{E}^1}} d_{\mathbb{E}^2}(x, B_{\mathbb{S}^1}) = 0.279$. More generally, if the blue line is chosen to be at the height $1 - x$, for some $x$ lying in the closed interval $[0, 1 - \cos(1)]$, it

is not hard to see that, for such embedding $f : B_{\mathbb{E}^1} \to \mathbb{E}^2$,

$$dh^{\mathbb{E}^2}(f(B_{\mathbb{E}^1}), B_{\mathbb{S}^1}) = \max\left\{x, 1-x, \sqrt{(1-\sin(1))^2 + (1-\cos(1)-x)^2}\right\}, \quad (6.39)$$

whose minimum is attained exactly when $x = \sqrt{(1-\sin(1))^2 + (1-x-\cos(1))^2}$, i.e. when

$$x = \frac{(1-\sin(1))^2 + (1-\cos(1))^2}{2-2\cos(1)} \approx 0.257. \quad (6.40)$$

This can be seen in the following picture. The two pink lines represent the biggest lengths

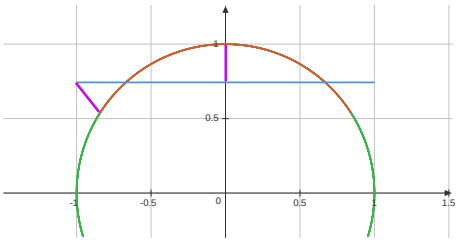


Figure 6.10: Comparison between $B_{\mathbb{E}^1}$ (in red) and $B_{\mathbb{S}^1}$ (in blue) inside $\mathbb{E}^2$ and a schematic of the biggest lengths (in pink) between the two point sets.

between points in $B_{\mathbb{E}^1}$ and $B_{\mathbb{S}^1}$, whose length is approximately equal to 0.257. Since we cannot compute exactly the Gromov-Hausdorff distance $\mathrm{d_{GH}}(\mathbb{S}^1, \mathbb{E}^1)$, choices have to be made. We have discussed why we would expect it to be somewhere in between $0.23$ and $0.257$. Since we are considering a very simple embedding for these estimations, it is reasonable to expect $\mathrm{d_{GH}}(\mathbb{S}^1, \mathbb{E}^1)$ to get closer to the lowest number when considering embeddings of these spaces in more complicated higher dimensional spaces, as the definition of the Gromov-Hausdorff distance allows.

For an arbitrary $n$, we reduce the estimation of $\mathrm{d_{GH}}(B_{\mathbb{E}^n}, B_{\mathbb{S}^n})$ to the one-dimensional case as follows. Analogously as we did for $n = 1$, given any value of $x$, we can consider $B_{\mathbb{E}^n}$ isometrically embedded in $\mathbb{E}^{n+1}$ by the map $f_x : B_{\mathbb{E}^n} \to \mathbb{R}^{n+1}$ defined by $f((x_0, x_1, \ldots, x_n)) = (x_0, x_1, \ldots, x_n, x)$. We view $B_{\mathbb{S}^n}$ isometrically embedded into $\mathbb{R}^{n+1}$ as the ball of radius one of $\mathbb{S}^n \subset \mathbb{R}^{n+1}$ with centre in the north pole of the sphere, i.e. the point with coordinates $(0, 0, \ldots, 0, 1)$. Given any unit vector $u$ in $\mathbb{R}^{n+1}$ orthogonal to $\vec{n} = (0, 0, \ldots, 0, 1)$, we define $\pi_u$ to be the two-dimensional plane linearly spanned by $u$ and $\vec{n}$. It is clear that, as $u$ ranges over the orthogonal space of $(0, 0, \ldots, 0, 1)$, the intersections $B_{\mathbb{S}^n} \cap \pi_u$ cover the whole $B_{\mathbb{S}^n}$ and the intersections $f_x(B_{\mathbb{E}^n}) \cap \pi_u$ cover the whole $f_x(B_{\mathbb{E}^n})$. Hence, in order to compute $\mathrm{d_H^{\mathbb{E}^{n+1}}}(f_x(B_{\mathbb{E}^n}), B_{\mathbb{S}^n})$, it suffices to compute $\mathrm{d_H^{\mathbb{E}^{n+1}}}(f_x(B_{\mathbb{E}^n}) \cap \pi_u, B_{\mathbb{S}^n} \cap \pi_u)$ for all unit vectors $u$ orthogonal to $\vec{n}$. Moreover, for two such unit vectors $u$ and $v$, there is a rigid motion (a rotation) of the whole ambient space

$\mathbb{R}^{n+1}$, that fixes $\vec{n}$ and that maps isometrically $f_x(B_{\mathbb{E}^n}) \cap \pi_u$ to $f_x(B_{\mathbb{E}^n}) \cap \pi_v$ and $B_{\mathbb{S}^n} \cap \pi_u$ to $B_{\mathbb{S}^n} \cap \pi_v$. In particular,

$$\mathrm{d}_{\mathrm{H}}^{\mathbb{E}^{n+1}}(f_x(B_{\mathbb{E}^n}) \cap \pi_u, B_{\mathbb{S}^n} \cap \pi_u) = \mathrm{d}_{\mathrm{H}}^{\mathbb{E}^{n+1}}(f_x(B_{\mathbb{E}^n}) \cap \pi_v, B_{\mathbb{S}^n} \cap \pi_v).$$

So we can do the previous computation for the specific value of $u = (0, 0, \ldots, 1, 0)$ (for clarity, where we mean the unit vector where the $n$-th coordinate is equal to 1 and the rest are zero). Crucially, the projection of $\mathbb{R}^{n+1}$ unto $\mathbb{R}^2$ (by projecting onto the last two coordinates) restrict to isometries on $B_{\mathbb{S}^n} \cap \pi_u$ and $B_{\mathbb{E}^n} \cap \pi_u$, from where it is deduced, analogously as in 6.39, that, as long as $x \in [0, 1 - \cos(1)]$, we have the following:

$$\mathrm{d}_{\mathrm{H}}^{\mathbb{E}^{n+1}}(f_x(B_{\mathbb{E}^n}) \cap \pi_u, B_{\mathbb{S}^n} \cap \pi_u) = \max\left\{x, 1 - x, \sqrt{(1 - \sin(1))^2 + (1 - \cos(1) - x)^2}\right\}.$$

### 6.6.4 Computational Implementation of the Gromov-Hausdorff Distance between the Remaining Model Spaces of Constant Curvature

In this subsection, we provide further details regarding how the Gromov-Hausdorff distances between the remaining manifolds (between $B_{\mathbb{E}^2}$ and $B_{\mathbb{H}^2}$, and $B_{\mathbb{S}^2}$ and $B_{\mathbb{H}^2}$) were approximated computationally. Note that as discussed in Section 6.3.3, these distances will be leveraged to compare not only model spaces but product manifolds as well.

#### 6.6.4.1 Discretizing the Original Continuous Manifolds

This section discuss how the the points in the corresponding balls of radius one described in Section 6.6.1 are generated from a practical perspective. The Euclidean, hyperbolic, and spherical spaces are continuous manifolds. To proximate the Gromov-Hausdorff distance between them we must discretize the spaces. A more fine-grained discretization results in a better approximation. As previously mentioned in Section 6.2, to compare $\mathbb{E}^n$, $\mathbb{S}^n$, and $\mathbb{H}^n$, we can take closed balls of radius one in each space. Since these spaces are homogeneous Riemannian manifolds, every ball of radius one is isometric. We can estimate or provide an upper bound for their Gromov-Hausdorff distance by using this method.

In the case of the Euclidean plane, we sample points from the ball, $B_{\mathbb{E}^2} = \{(r\cos(t), r\sin(t)) : r \in [0, 1], t \in [0, 2\pi)\}$, with a discretization of $10,000$ points in both $r$ and $t$. To obtain points in $B_{\mathbb{H}^2}$ we will use the points in $B_{\mathbb{E}^2}$ as a reference, and apply the exponential map, and the change of coordinates described in Section 6.6.1 to convert points in $B_{\mathbb{E}^2}$ to points in $B_{\mathbb{H}^2}$. However, due to numerical instabilities instead of sampling from $B_{\mathbb{E}^2}$, we will restrict ourselves to $B'_{\mathbb{E}^2} = \{(r\cos(t), r\sin(t)) : r \in [0.00000001, 0.97], t \in [0, 2\pi)\}$ and use a discretization of $10,000$ as before. Lastly to sample points for the spherical space,

$B_{\mathbb{S}^2} = \{(\sin(\beta)\cos(\alpha), \sin(\beta)\sin(\alpha), \cos(\beta)) : \alpha \in [0, 2\pi), \beta \in [0, 1]\}$, we use a discretization of 100 for both $\alpha$ and $\beta$. The granualirity of the discretization was chosen as a trade-off between resolution and computational time. We observed that the Gromov-Hausdorff distance stabilized for our discretization. However, given the nature of the Gromov-Hausdorff distance, it is difficult to conclude whether some unexpected behaviour could be observed with greater discretization.

#### 6.6.4.2 Calculating Constants for the Embedding Function

The next step is to obtain a computational approximation of the mapping function defined in Section 6.6.2. The embedding of $\mathbb{H}^n$ into $\mathbb{E}^{6n-6}$ requires computing several constants. Using a standard integral solvers $A \approx 0.141328$, (Equation 6.28) can be approximated. To calculate the constant $c \approx 10.255014502464228$, we discretize the input $x \in [-2, 2]$ using a step size of $10^{-8}$ to compute $G_1$ and $G_2$ and use a for loop to calculate the max in Equation 6.31. Note that this requires to constantly reevaluate the integrals for $\psi_1(x)$ and $\psi_2(x)$. Likewise, $G_1$ and $G_2$ are also used to obtain $\varepsilon$ in Equation 6.36, which is in turn used to calculate the mapping function that maps points in $\mathbb{H}^2$ to the higher dimensional embedding Euclidean space $\mathbb{E}^6$. We use $F$ to map $B_{\mathbb{H}^2}$ to $\mathbb{E}^6$, and we keep those points fixed in space.

#### 6.6.4.3 Optimizing the Embedding Functions for the Euclidean and Spherical Spaces

Next to approximate the Gromov-Hausdorff distances:

$$\mathrm{d_{GH}}(B_{\mathbb{E}^2}, B_{\mathbb{H}^2}) \approx \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6, f_k, F}(P_i^H, Q_j) = \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6}(f_k(P_i^H), F(Q_j)), \tag{6.41}$$

and,

$$\mathrm{d_{GH}}(B_{\mathbb{S}^2}, B_{\mathbb{H}^2}) \approx \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6, g_k, F}(P_i^S, Q_j) = \min_{i,j,k} \mathrm{d_H}^{\mathbb{R}^6}(g_k(P_i^S), F(Q_j)), \tag{6.42}$$

we must optimize $f_k$ and $g_k$. These are the functions used to embed $B_{\mathbb{E}^2}$ and $B_{\mathbb{S}^2}$ in $\mathbb{E}^{6n-6}$. Note that in practice, $B_{\mathbb{E}^2}$ and $B_{\mathbb{S}^2}$ are discretized into $P_i^H$ and $P_i^S$, respectively.

To optimize for $f_k$ we consider all possible permutations of the basis vectors of $\mathbb{E}^6$: $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ for each $f_k$ we consider two elements $\{e_i, e_j\}$ and use those dimensions to embed $\mathbb{E}^2$ in $\mathbb{E}^6$. In principle, we should also consider a small offset given by $F(\mathbf{0}) = \mathbf{0}$, but it is zero regardless. Additionally, during the optimization we also add a small vector (with all entries but a single dimension between zero) to the mapping function

to translate the plane in different directions by an offset of between $-0.5$ and $0.5$, with a total of a 100 steps between these two quantities.

To optimize for $g_k$ we follow a similar procedure in which we consider all permutations of the basis vectors. Note however, that in this case we would have three basis vectors instead of two, given how $B_{\mathbb{S}^2} = \{(\sin(\beta)\cos(\alpha), \sin(\beta)\sin(\alpha), \cos(\beta)) : \alpha \in [0, 2\pi), \beta \in [0, 1]\}$ is sampled. For each permutation family, $P_i^S$, we also consider its negative counterpart, $-P_i^S$, to compute the Gromov-Hausdorff distance, as well as experimenting with offsetting the mapping function.

### 6.6.5 Derivation of Number of Product Manifold Combinations in the Search Space

Here, we derive the exact number of nodes in the graph search space in our setting. In particular, we consider the case with three model spaces (the Eucledian plane, the hyperboloid and the hypershere). The growth of the search space can be modelled with growth of a tree, as depicted in Figure 6.11.

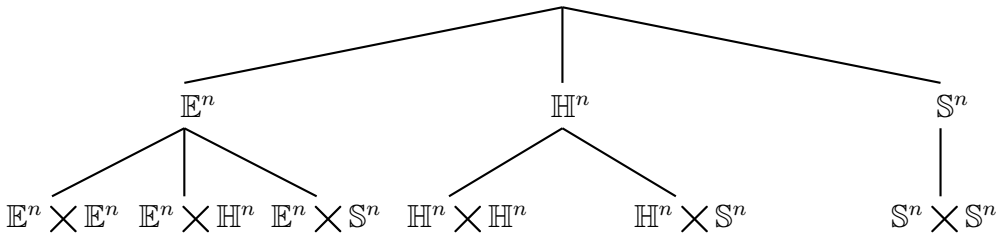


Figure 6.11: The growth of the graph search space as a function of the number of model spaces used to generate product manifolds can be represented as a tree. Note that as discussed in Section 6.3.3 we assume commutativity: $\mathcal{M}_i \times \mathcal{M}_j = \mathcal{M}_j \times \mathcal{M}_i$.

To calculate the number of elements in the search space, we define the total number of products at level $h$ of the tree as $N$. We have that $N(h) = N_E(h) + N_H(h) + N_S(h)$, where $N_E(h)$ is the number of Euclidean spaces added to the product manifolds at depth $h$ of the tree, and $N_H(h)$ and $N_S(h)$ represent the same for the hyperboloid and the hypersphere respectively. By recursion, we can write

$$N_E(h) = N_E(h-1) = 1 \tag{6.43}$$

$$N_H(h) = N_E(h-1) + N_H(h-1) \tag{6.44}$$

$$= 1 + h \tag{6.45}$$

$$N_S = N_E(h-1) + N_H(h-1) + N_S(h-1) \tag{6.46}$$
$$= \frac{h(h+1)}{2} N_E(1) + hN_H(1) + N_S(1) \tag{6.47}$$
$$= \frac{h(h+1)}{2} + h + 1 \tag{6.48}$$

hence we have that

$$N(h) = 1 + (1+h) + (1 + h + \frac{h(h+1)}{2}) \tag{6.49}$$
$$= 3 + \frac{5}{2}h + \frac{1}{2}h^2 \tag{6.50}$$

To be consistent with the previous notation, we write the above in terms of the number of products $n_p$

$$N(n_p) = 3 + \frac{5}{2}n_p + \frac{1}{2}n_p^2 \tag{6.51}$$

The total number of nodes in the graph search space for a number of product $n_p$ is then

$$N_T = \sum_{i=1}^{n_p} N(i) \tag{6.52}$$

### 6.6.6 Visualizing the Graph Search Space

In this subsection, we give a visual depiction of the graph search space we construct for neural latent geometry search. Figure 6.12 and Figure 6.13 provide plots of the graph search space as the size of the product manifolds considered increases. For product manifolds composed of a high number of model spaces, visualizing the search space becomes difficult. We omit the strengths of the connections for visual clarity in this plots.

If we focus on Figure 6.12, we use `e`, `h`, and `s` to refer to the model spaces of constant curvature $\mathbb{E}^2$, $\mathbb{H}^2$, and $\mathbb{S}^2$. In this work, we have considered model spaces of dimensionality two, but the graph search space would be analogous if we were to change the dimensionality of the model spaces. Nodes that include more than one letter, such as `hh`, `ss`, `ee`, etc, refer to product manifolds. For example, $hh$ would correspond to the product manifold $\mathbb{H}^2 \times \mathbb{H}^2$. As discussed in Section 6.3.3, we can see that connections are only allowed between product manifolds that differ by a single model space. To try to clarify this further, we can see that `e`, `h` and `s` are all interconnected since they only differ by a single model space (one deletion and one addition). Likewise, `e` is connected to `ee`, `eh`, and `ee` since they only differ by a single model space (one addition). However, `e` is not connected to `hs` (one deletion and two additions) nor is `ee` connected to `ss` (two deletions and two additions).

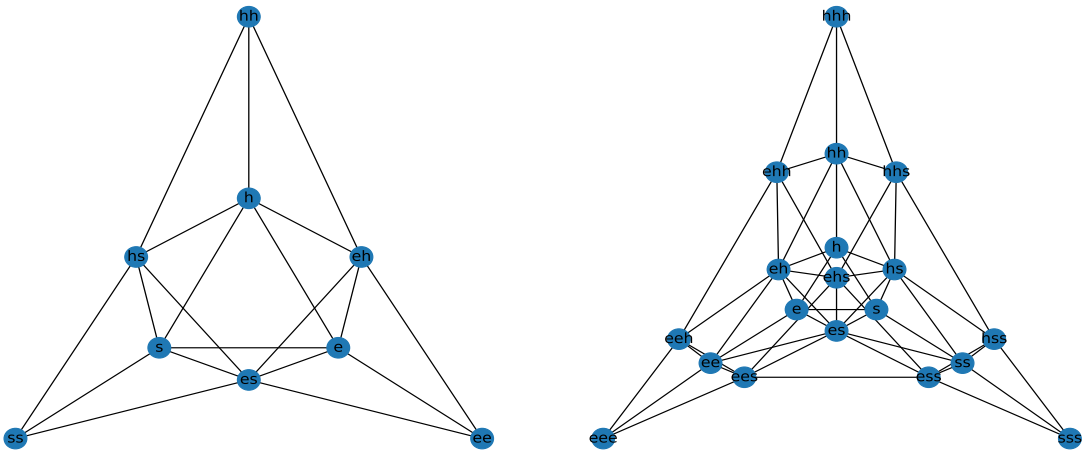


Figure 6.12: Graph search spaces for $n_p = 2$ (left) and $n_p = 3$ (right). Strength of connectivity is not depicted in the graph.

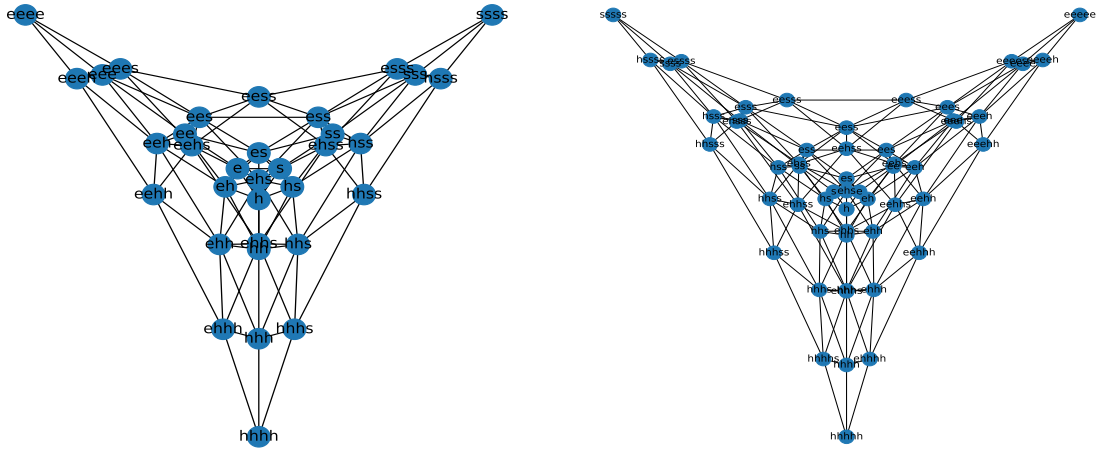


Figure 6.13: Graph search spaces for $n_p = 4$ (left) and $n_p = 5$ (right). Strength of connectivity is not depicted in the graph.

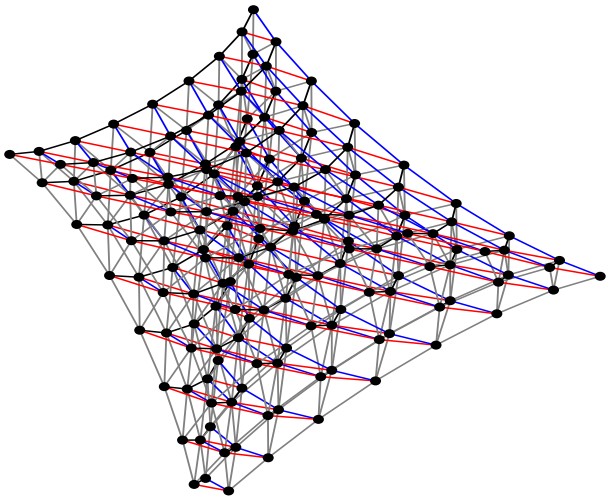

Figure 6.14: Graph search spaces for $n_p = 8$.

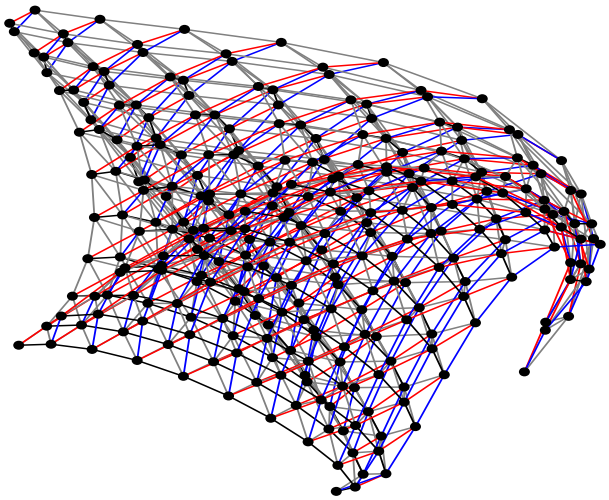

Figure 6.15: Graph search spaces for $n_p = 10$.

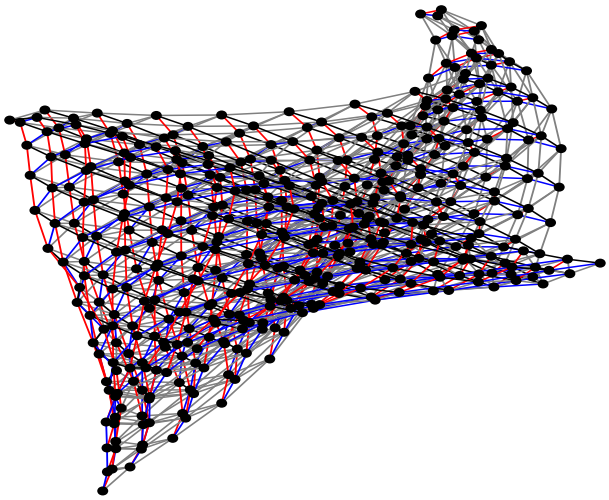

Figure 6.16: Graph search spaces for $n_p = 12$.

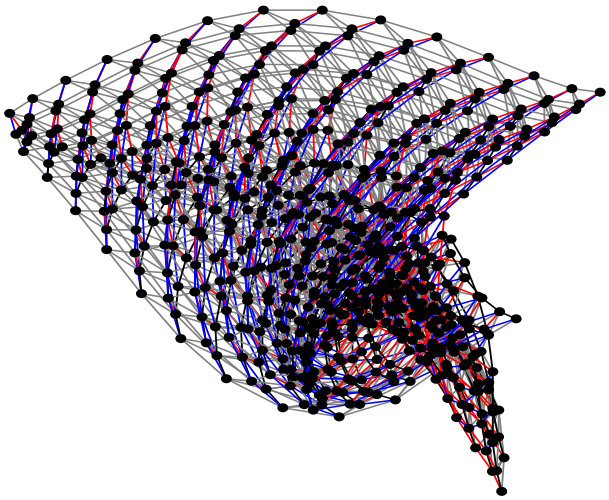

Figure 6.17: Graph search spaces for $n_p = 14$.

### 6.6.7 Motivating Gromov-Hausdorff Distances for Comparing Latent Geometries

The use of Gromov-Hausdorff distances can be motivated from a theoretical and practical perspective. From a theoretical perspective, the Gromov-Hausdorff distance offers a way to gauge the similarity between metric spaces. This is achieved by casting their representation within a shared space, all the while maintaining the original pairwise distances. In the context of machine learning models, particularly those involving generative models (such as VAEs or GANs), the latent space assumes a pivotal role in shaping the characteristics of the generated data. This also holds true in the case of reconstruction tasks, such as those carried out by autoencoders. Through the application of a metric that factors in their inherent geometries, the aim is to capture the concept of resemblance in data generation, reconstruction, and other downstream tasks which will be heavily affected by the choice of latent geometry.

While traditional metrics like Euclidean or cosine distances might suffice for some cases, they do not always capture the complex and nonlinear relationships between data points in high-dimensional spaces. The Gromov-Hausdorff distance considers the overall structure and shape of these spaces, rather than just individual distances, which can lead to better generalization and robustness. This is especially relevant when dealing with complex data distributions or high-dimensional latent spaces. Moreover, another appealing aspect of Gromov-Hausdorff distance is its invariance to isometric transformations, such as rotations, translations, and reflections. This is a desirable property when comparing latent spaces, as the metric used should reflect the similarity in shape and structure, rather than being influenced by trivial transformations.

While the direct relationship between Gromov-Hausdorff distance and model performance might not be immediately obvious, we argue that if two models have similar latent space geometries, this suggests that they might capture similar underlying data structures, which could lead to similar performance on downstream tasks. This is known as an assumption of *smoothness* in the NAS literature, see [204]. However, it is important to note that this relationship is not guaranteed, and the effectiveness of Gromov-Hausdorff distance as a proxy measure depends on the specific application and the nature of the models being compared.

### 6.6.8 Method Scalability

The proposed approach sidesteps scalability concerns by preventing the need to recalibrate GH coefficients, as long as one adheres to latent spaces derived from products of model

spaces of dimension two. If a higher-dimensional latent space was needed, this could be solved by adding additional "graph slices" with augmented dimensions (as depicted in Figure 6.2). These slices can be integrated with the lower-dimensional counterpart of the graph search space following the procedure described in the chapter. The number of coefficients remains constant at 3, rendering the Gromov-Hausdorff distances free from extra computation overhead when increasing the dimension.

Moreover, in alignment with the manifold hypothesis, data is expected to exist within a low-dimensional manifold. This discourages the exploration of higher dimensions for latent is work has already delved into relatively large search spaces, which include the exploration of high-dimensional latent spaces.

## 6.7 Background on Bayesian Optimization

Bayesian optimization (BO) is a query-based optimization framework for black-box functions that are expensive to evaluate. It builds a probabilistic model of the objective function using past queries to automate the selection of meta-parameters and minimize computational costs. BO seeks to find the global optimum of a black-box function $\mathfrak{f} : \mathcal{X} \to \mathbb{R}$ by querying a point $\mathbf{x}_n \in \mathcal{X}$ at each iteration $n$ and obtaining the value $y_n = \mathfrak{f}(\mathbf{x}_n) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is the noise in the observation.

To do so, BO employs a surrogate to model the function $\mathfrak{f}(\cdot)$ being minimized given the input and output pairs $\mathcal{D}_t = \{(\mathbf{x}_i, y_i = \mathfrak{f}(x_i)\}_{i=1}^N$, and selects the next point to query by maximizing an *acquisition function*. In our work, we use the Gaussian Process (GP)[269] as the surrogate model and expected improvement (EI)[193] as the acquisition function.

**Preliminaries.** Bayesian optimization (BO) is particularly beneficial in problems for which evaluation is costly, behave as a black box, and for which it is impossible to compute gradients with respect to the loss function. This is the case when tuning the hyperparameters of machine learning models. In our case, we will use it to find the optimal product manifold signature. Effectively, Bayesian optimization allows us to automate the selection of critical meta-parameters while trying to minimize computational cost. This is done building a probabilistic proxy model for the objective using outcomes recorded in past experiments as training data. More formally, BO tries to find the global optimum of a black-box function $\mathfrak{f} : \mathcal{X} \to \mathbb{R}$. To do this, a point $\mathbf{x}_n \in \mathcal{X}$ is queried at every iteration

$n$ and yields the value $y_n = \mathfrak{f}(\mathbf{x}_n) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, is a noisy observation of the function evaluation at that point. BO can be phrased using decision theory

$$L(\mathfrak{f}, \{\mathbf{x}_n\}_{n=1}^N) = N \times c + \min_{1 \leqslant n \leqslant N} \mathfrak{f}(\mathbf{x}_n), \tag{6.53}$$

where $N$ is the maximum number of evaluations, $c$ is the cost incurred by each evaluation, and the loss $L$ is minimized by finding a lower $\mathfrak{f}(\mathbf{x})$. In general, Equation 6.53 is intractable and the closed-form optimum cannot be found, so heuristic approaches must be used. Instead of directly minimizing the loss function, BO employs a surrogate model to model the function $\mathfrak{f}(\cdot)$ being minimized given the input and output pairs $\mathcal{D}_t = \{(\mathbf{x}_i, y_i = \mathfrak{f}(x_i)\}_{i=1}^N$. Furthermore, in order to select the next point to query, an *acquisition function* is maximized. In this work, the surrogate model is chosen to be a Gaussian Process (GP) [269] and expected improvement (EI) [193] is used as the acquisition function.

**Gaussian Processes.** A GP is a collection of random variables such that every finite collection of those random variables has a multivariate normal distribution. A GP is fully specified by a *mean function*, $\mu(\mathbf{x})$, and *covariance function* (or *kernel*), denoted as $k(\mathbf{x}, \mathbf{x}')$. The prior over the mean function is typically set to zero, and most of the complexity of the model hence stems from the kernel function. Given $t$ iterations of the optimization algorithm, with an input $\mathbf{x}_t = [x_1, \ldots, x_t]^T$ and output $\mathbf{y}_t = [y_1, \ldots, y_t]^T$, the posterior mean and variance are given by

$$\mu(x_{t+1}|\mathcal{D}_t) = \mathbf{k}(x_{t+1}, \mathbf{x}_t)[\mathbf{K}_{1:t} + \sigma_n^2 \mathbf{I}_t]^{-1} \mathbf{y}_t, \tag{6.54}$$

and

$$\sigma(x_{t+1}|\mathcal{D}_t) = \mathbf{k}(x_{t+1}, x_{t+1}) - \mathbf{k}(x_{t+1}, \mathbf{x}_t)[\mathbf{K}_{1:t} + \sigma_n^2 \mathbf{I}_t]^{-1} \mathbf{k}(\mathbf{x}_t, x_{t+1}), \tag{6.55}$$

where we define $[\mathbf{K}_{1:t}]_{i,j} = k(x_i, x_j)$ is the $(i, j)$-th element of the Gram matrix.

**Expected Improvement.** EI is a widely used acquisition function for Bayesian optimization. EI improves the objective function through a greedy heuristic which chooses the point which provides the greatest expected improvement over the current best sample point. EI is calculated as

$$\alpha_{\text{EI}}(\mathbf{x}) = \sigma(\mathbf{x})[\Gamma(\mathbf{x})\Phi(\Gamma(\mathbf{x})) + \mathcal{N}(\Gamma(\mathbf{x})|0, 1)], \tag{6.56}$$

where

$$\Gamma(\mathbf{x}) = \frac{\mathfrak{f}(\mathbf{x}_{best}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})}, \tag{6.57}$$

and $\Phi(\cdot)$ denotes the CDF of a standard normal distribution.

## 6.8 Experimental Setup

In this section we provide additional details for the implementation of the experimental setup, including how the datasets are generated, relevant theoretical background, and the hyperparameters used for BO over our discrete graph search space.

### 6.8.1 Synthetic Experiments

To assess the effectiveness of the proposed graph search space, we conduct a series of tests using synthetic data. Our objective is to establish a scenario in which we have control over the latent space and manually construct the optimal latent space product manifold. To achieve this, we initiated the process by generating a random vector, denoted as $\mathbf{x}$, and mapping it to a predetermined "ground truth" product manifold $\mathcal{P}_T$. This mapping is performed using the exponential map for the product manifold $\mathcal{P}_T$, which can be derived based on the model spaces of constant curvature that constitute it. Subsequently, the resulting projected vector is decoded via a neural network with fixed weights, denoted as $f_\theta$, to yield a reference signal $y^{\mathcal{P}_T}$. The rationale behind employing a frozen network is to introduce a non-linear mapping from the latent space to the signal space. This approach aims to mimic the behavior of a trained network in a downstream task, while disregarding the specific task or model weights involved. By utilizing a frozen network, we can capture the essence of a non-linear mapping without relying on the exact details of the task or specific model weights. The primary goal is to recover this reference signal using the neural latent geometry search (NLGS) approach. To generate the remaining dataset, we employ the same random vector and mapped it to several other product manifolds, denoted as $\{\mathcal{P}_i\}_{i\in n_\mathcal{P}}$, comprising an equivalent number of model spaces as $\mathcal{P}_T$ but with distinct signatures. Decoding the projected vector using the same neural network produces a set of signals, denoted as $\{y^{\mathcal{P}_i}\}_{i\in n_\mathcal{P}}$. To populate the nodes of the graph search space, we assign the corresponding value of mean squared error (MSE) between $y^{\mathcal{P}_T}$ and $y^{\mathcal{P}_i}$ for each pair. In this manner, our search algorithm aims to identify the node within the graph that minimizes the error, effectively determining the latent geometry capable of recovering the original reference signal. A schematic of the proposed method and specifics on the construction of the network used to decode the signal are shown in Figure 6.18 and Table 6.3.

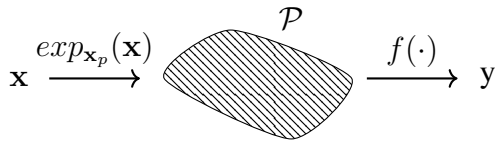


Figure 6.18: Schematic of procedure used to generate synthetic datasets. We model $f$ as an MLP.

Table 6.3: Summary of network to generate synthetic datasets.

| **Model** |
|---|
| Linear (data dim, 100) - ELU |
| Linear (100, 100) - ELU |
| Linear (100, 100) - ELU |
| Linear (100, 5) - ELU |

### 6.8.2 Autoencoders

Autoencoders are a type of neural network architecture that can be used for unsupervised learning tasks. They are composed of two main parts: an encoder and a decoder. The encoder takes an input and compresses it into a low-dimensional representation, while the decoder takes that representation and generates an output that tries to match the original input. The goal of an autoencoder is to learn a compressed representation of the input data that captures the most important features, and can be used for tasks such as image denoising, dimensionality reduction, and anomaly detection. They have been used in a wide range of applications, including natural language processing, computer vision, and audio analysis.

Table 6.4: Autoencoder architecture summary.

| Component | Layers |
|---|---|
| **Encoder** | Conv2d (1, 20, 3) - BatchNorm2d - SiLU<br>Conv2d (20, 20, 3) - BatchNorm2d - SiLU<br>⋮<br>(9 repetitions)<br>⋮<br>Conv2d (20, 2, 3) - BatchNorm2d - SiLU<br>Flatten - Linear - SiLU |
| **Decoder** | Linear - SiLU - Unflatten<br>ConvTranspose2d (2, 20, 3) - BatchNorm2d - SiLU<br>ConvTranspose2d (20, 20, 3) - BatchNorm2d - SiLU<br>⋮<br>(9 repetitions)<br>⋮<br>ConvTranspose2d (20, 1, 3) - BatchNorm2d - Sigmoid |

The autoencoder's objective is to learn a compressed representation (latent space) of the input data and use it to reconstruct the original input as accurately as possible. In our experiments, the encoder takes an input image and applies a series of convolutional layers with batch normalization and SiLU activation functions, reducing the image's dimensions while increasing the number of filters or feature maps. The final output of the encoder is a tensor of shape $(\texttt{batchsize}, \texttt{latentdim}, 6, 6)$ or $(\texttt{batchsize}, \texttt{latentdim}, 10, 10)$, where `latentdim` is the desired size of the latent space. After flattening this tensor, a fully connected layer is used to map it to the desired latent space size. The encoder's output is a tensor of shape $(\texttt{batchsize}, \texttt{latentdim})$, which represents the compressed representation of the input image. The decoder takes the latent space representation as input and applies a series of transpose convolutional layers with batch normalization and SiLU activation functions, gradually increasing the image's dimensions while decreasing the number of feature maps. The final output of the decoder is an image tensor of the same shape as the input image. The loss functions used for training are the `MSELoss` (mean squared error) and `BCELoss` (binary cross-entropy) from the `PyTorch` library. A summary of the autoencoder is provided in Table 6.4.

### 6.8.3 Latent Graph Inference

Graph Neural Networks (GNNs) leverage the connectivity structure of graph data to achieve state-of-the-art performance in various applications. Most current GNN architectures assume a fixed topology of the input graph during training. Research has focused on improving diffusion using different types of GNN layers, but discovering an optimal graph topology that can help diffusion has only recently gained attention. In many real-world applications, data may only be accessible as a point cloud of data, making it challenging to access the underlying but unknown graph structure. The majority of Geometric Deep Learning research has relied on human annotators or simplistic pre-processing algorithms to generate the graph structure, and the correct graph may often be suboptimal for the task at hand, which may benefit from rewiring. Latent graph inference refers to the process of inferring the underlying graph structure of data when it is not explicitly available. In many real-world applications, data may only be represented as a point cloud, without any knowledge of the graph structure. However, this does not mean that the data is not intrinsically related, and its connectivity can be utilized to make more accurate predictions.

For these experiments we reproduce the architectures described in [233], see Table 6.5 and Table 6.6 for the original architectures. In our case, we use the GCN-dDGM model leveraging the original input graph inductive bias.

Table 6.5: Summary of model architectures for experiments for Cora and CiteSeer.

| | | Model | | |
|---|---|---|---|---|
| | | **MLP** | **GCN** | **GCN-dDGM** |
| No. Layer parameters | Activation | Layer type | | |
| | | N/A | N/A | dDGM |
| (No. features, 32) | ELU | Linear | Graph Conv | Graph Conv |
| | | N/A | N/A | dDGM |
| (32, 16) | ELU | Linear | Graph Conv | Graph Conv |
| | | N/A | N/A | dDGM |
| (16, 8) | ELU | Linear | Graph Conv | Graph Conv |
| (8, 8) | ELU | Linear | Linear | Linear |
| (8, 8) | ELU | Linear | Linear | Linear |
| (8, No. classes) | - | Linear | Linear | Linear |

Table 6.6: dDGM$^*$ and dDGM architectures for Cora and CiteSeer.

| | | **dDGM$^*$** | **dDGM** |
|---|---|---|---|
| No. Layer parameters | Activation | Layer type | |
| (No. features, 32) | ELU | Linear | Linear |
| (32, 16 per model space) | ELU | Linear | Graph Conv |
| (16 per model space, 4 per model space) | Sigmoid | Linear | Graph Conv |

#### 6.8.3.1 Differentiable Graph Module

In their work, [155] presented a general method for learning the latent graph by leveraging the output features of each layer. They also introduced a technique to optimize the parameters responsible for generating the latent graph. The key concept is to use a similarity metric between the latent node features to generate optimal latent graphs for each layer $l$. In this context, $\mathbf{X}^{(0)}$ and $\mathbf{A}^{(0)}$ represent the original input node feature matrix and adjacency matrix, respectively. So that

$$\mathbf{X}^{(0)} = \begin{bmatrix} -\mathbf{x}_1^{(0)} - \\ -\mathbf{x}_2^{(0)} - \\ \vdots \\ -\mathbf{x}_n^{(0)} - \end{bmatrix}, \tag{6.58}$$

and $\mathbf{A}^{(0)} = \mathbf{A}$ if the adjacency matrix from the dataset, or $\mathbf{A}^{(0)} = \mathbf{I}$ if $\mathcal{G} = (\mathcal{V}, \emptyset)$. The proposed architecture in [155] consists of two primary components: the Differentiable Graph Module (DGM) and the Diffusion Module. The Diffusion Module, $\mathbf{X}^{(l+1)} = g_{\phi}(\mathbf{X}^{(l)}, \mathbf{A}^{(l)})$, may be one (or multiple) standard GNN layers. The first DGM module takes the original node features and connectivity information and produces an updated adjacency matrix

$$\mathbf{X}'^{(l=1)}, \mathbf{A}^{(l=1)} = \text{DGM}(\mathbf{X}^{(0)}, \mathbf{A}^{(0)}). \tag{6.59}$$

In principle, the DGM module utilizes information from previous layers to generate adjacency matrices at each layer

$$\mathbf{X}'^{(l+1)}, \mathbf{A}^{(l+1)} = \text{DGM}(concat(\mathbf{X}^{(l)}, \mathbf{X}'^{(l)}), \mathbf{A}^{(l)}). \tag{6.60}$$

To do so, a measure of similarity is used

$$\varphi(\mathbf{x}_i'^{(l+1)}, \mathbf{x}_j'^{(l+1)}) = \varphi(f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_i^{(l)}), f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_j^{(l)})). \tag{6.61}$$

In summary, the proposed approach utilizes a parameterized function $f_{\mathbf{\Theta}^{(l)}}$ with learnable parameters to transform node features and a similarity measure $\varphi$ to compare them. The function can be an MLP or composed of GNN layers if connectivity information is available. The output of the similarity measure is used to create a fully-connected weighted adjacency matrix for the continuous differentiable graph module (cDGM) approach or a sparse and discrete adjacency matrix for the discrete Differentiable Graph Module (dDGM) approach, with the latter being more computationally efficient and recommended by the authors of the DGM paper [155]. To improve the similarity measure $\varphi$ and construct better

latent graphs, the approach employs product spaces and Riemannian geometry. Additionally, an extra loss term is used to update the learnable parameters of the dDGM module.

Lastly, we will examine the dDGM module, which utilizes the Gumbel Top-k [165] technique to generate a sparse $k$-degree graph by stochastically sampling edges from the probability matrix $\mathbf{P}^{(l)}(\mathbf{X}^{(l)}; \mathbf{\Theta}^{(l)}, T)$, which is a stochastic relaxation of the kNN rule, where each entry corresponds to

$$p_{ij}^{(l)} = \exp(-\varphi(T)(\mathbf{x'}_i^{(l+1)}, \mathbf{x'}_j^{(l+1)})). \tag{6.62}$$

$T$ being a learnable parameter. The primary similarity measure utilized in [155] involved computing the distance between the features of two nodes in the graph embedding space

$$p_{ij}^{(l)} = \exp(-T\Delta(\mathbf{x'}_i^{(l+1)}, \mathbf{x'}_j^{(l+1)})) = \exp(-T\Delta(f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_i^{(l)}), f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_j^{(l)})), \tag{6.63}$$

where $\Delta(\cdot, \cdot)$ denotes a generic measure of distance between two points. They assumed that the latent features laid in an Euclidean plane of constant curvature $K_{\mathbb{E}} = 0$, so that

$$p_{ij}^{(l)} = \exp(-T\mathfrak{d}_{\mathbb{E}}(f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_i^{(l)}), f_{\mathbf{\Theta}}^{(l)}(\mathbf{x}_j^{(l)}))), \tag{6.64}$$

where $\mathfrak{d}_{\mathbb{E}}$ is the distance in Euclidean space. Based on

$$\text{argsort}(\log(\mathbf{p}_i^{(l)}) - \log(-\log(\mathbf{q}))) \tag{6.65}$$

where $\mathbf{q} \in \mathbb{R}^N$ is uniform i.i.d in the interval $[0, 1]$, we can sample the edges

$$\mathcal{E}^{(l)}(\mathbf{X}^{(l)}; \mathbf{\Theta}^{(l)}, T, k) = \{(i, j_{i,1}), (i, j_{i,2}), ..., (i, j_{i,k}) : i = 1, ..., N\}, \tag{6.66}$$

$k$ being the number of sampled connections using the Gumbel Top-k trick. The Gumbel Top-k approach utilizes the categorical distribution $\frac{p_{ij}^{(l)}}{\Sigma_r p_{ir}^{(l)}}$ for sampling, and the resulting unweighted adjacency matrix $\mathbf{A}^{(l)}(\mathbf{X}^{(l)}; \mathbf{\Theta}^{(l)}, T, k)$ is used to represent $\mathcal{E}(\mathbf{X}^{(l)}; \mathbf{\Theta}^{(l)}, T, k)$. It is worth noting that including noise in the edge sampling process can generate random edges in the latent graphs, which can serve as a form of regularization.

Finally, we can summarize a multi-layer GNN using the dDGM module as

$$\mathbf{X'}^{(l+1)} = f_{\mathbf{\Theta}}^{(l)}(concat(\mathbf{X}^{(l)}, \mathbf{X'}^{(l)}), \mathbf{A}^{(l)}), \tag{6.67}$$

$$\mathbf{A}^{(l+1)} \sim \mathbf{P}^{(l)}(\mathbf{X'}^{(l+1)}), \tag{6.68}$$

$$\mathbf{X}^{(l+1)} = g_{\boldsymbol{\phi}}(\mathbf{X}^{(l)}, \mathbf{A}^{(l+1)}). \tag{6.69}$$

Equations 6.67 and 6.68 belong to the dDGM module, while Equation 6.69 is associated with the Diffusion Module. In our study, we extend Equation 6.62 to measure distances without relying on the assumption utilized in [155], which restricts the analysis to fixed-curvature spaces, specifically to Euclidean space where $K_{\mathbb{E}} = 0$.

### 6.8.4 Naive Bayesian Optimization

Naive BO in the main text considers performing BO over a fully-connected unweighted graph. Such a graph is latent geometry agnostic, that is, it does not include any metric geometry inductive bias to provide the optimization algorithm with a sense of closeness between the candidate latent geometries.

## 6.9 Ablation of the Gromov-Hausdorff Coefficients

We evaluate the impact of Gromov-Hausdorff coefficients on the graph search space by conducting an additional set of experiments. We compare the use of an unweighted, pruned graph search space with the introduction of Gromov-Hausdorff coefficients in the datasets considered in this chapter. The results are show in Figures 6.19 to 6.22 below.

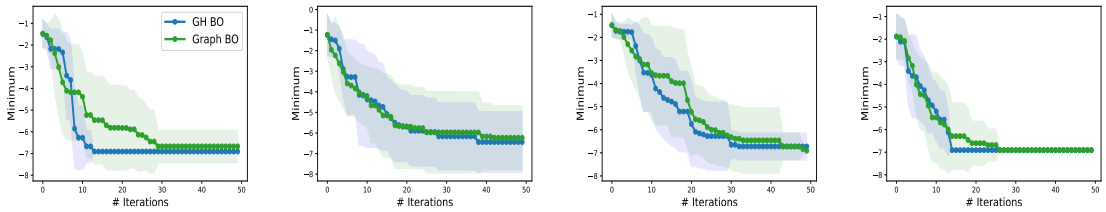


Figure 6.19: Results (mean and standard deviation over 10 runs) for candidate latent geometries involving product manifolds composed of 13 model spaces. For each plot a different ground truth product manifold $\mathcal{P}_T$ is used to generate the reference signal.

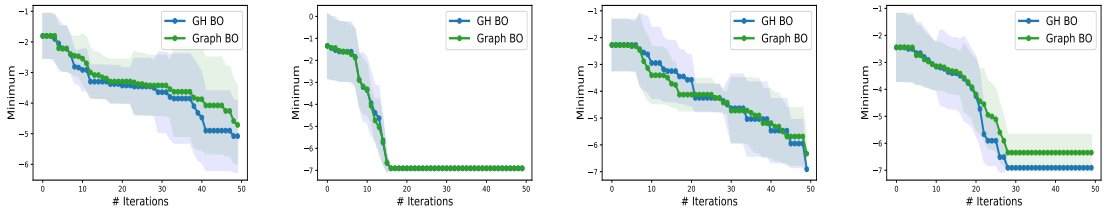


Figure 6.20: Results (mean and standard deviation over 10 runs) for candidate latent geometries involving product manifolds composed of 15 model spaces. For each plot a different ground truth product manifold $\mathcal{P}_T$ is used to generate the reference signal.

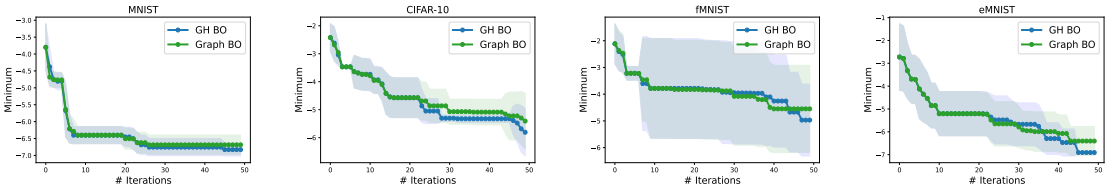


Figure 6.21: Results (mean and standard deviation over 10 runs) on image reconstruction tasks, for $n_p = 7$.

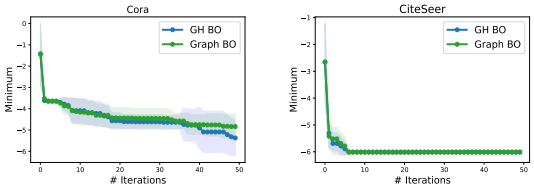


Figure 6.22: Results (mean and standard deviation over 10 runs) on latent graph inference tasks.

The plots above show that in the majority of cases, the GH weights appear to either match or enhance the performance of the search algorithm. This observation is particularly pronounced in synthetic tasks, which can be intuitively justified due to the significant impact of changes in the latent geometry on the network's output in such a setup. However, we highlight that a large amount of the performance gains in the algorithm seem to come from the inductive bias endowed to the search algorithm through the graph search space. This could be due to the fact that in image reconstruction tasks, searching for the optimal dimension is more important for reconstruction than finding the optimal latent geometry within a particular dimension. Since the GH coefficients are unity between dimensions, this makes the performance of the two search spaces similar in this context.

It should be noted that setting the inverse of the Gromov-Hausdorff distance as the edges of a search graph is only one potential way to add a geometric inductive bias into the search space. Future work could use similar mechanisms as those developed in this chapter but in a different context in order to improve optimization performance. Finally, we would like to highlight that a more thorough analysis of the links between the performance of neural networks and the similarity of their latent geometries is a relevant question that merits further study, which is left for future work.

# Chapter 7

# Afterword

Over the previous chapters, we have shown that incorporating temporal and geometric structure is a useful inductive bias for a number of deep learning architectures, and have sought a deeper understanding of some of these architectures and methods. We have focused on two types of structure that permeate data in the real world: *temporal* or *causal* ordering and *geometry*, and have argued that incorporating these inductive biases into deep learning models can significantly enhance them. To this end, we have pursued four lines of work that embody this goal. These include: *(i)* An extension of survival analysis models to the temporal domain, with an architecture capable of extracting both local and global dependencies, focusing on limit order book data (§3) *(ii)* The introduction of the *Rough Transformer*, an architecture to compress irregular streams into path-signatures before applying attention, retaining the expressiveness of Transformers while reducing memory and training time (§4) *(iii)* A state-space re-interpretation of message-passing networks that exposes, and then repairs, the link between vanishing gradients, over-smoothing and over-squashing (§5) *(iv)* A Bayesian-optimization search over products of constant-curvature manifolds, allowing the geometry itself to become a learnable hyper-parameter (§6). While we have provided evidence supporting the importance of incorporating structure into deep learning architectures, we highlight two open challenges and promising avenues for future work.

**Understanding transformers through the lens of graphs:** At their core, Transformers have a deep connection with MPNNs, since both architectures mix tokens or nodes through a graph, which in the case of transformers is learned through the self-attention operation. From this perspective, the behaviors observed in GCNs serve as a microcosm of those emerging in Transformers: for example, the rank collapse phenomenon [86] in attention

mirrors the oversmoothing effect in GCNs. More recently, [16] also uncovered links between oversquashing in GNNs and analogous bottleneck effects in Transformers. Together, these insights suggest that many of the conceptual tools and analytical frameworks developed in graph representation learning can be directly translated to advance our theoretical understanding of sequence models, in particular attention-based ones.

**An improved sequence modeling perspective for learning on graphs:** The GCN-SSM framework introduced in §5 served primarily as a theoretical lens for emphasizing the role of vanishing gradients in graph neural networks. Its concrete instantiation, however, remained quite rudimentary—relying on fixed state and input matrices without learnable adaptations. To drive this line of work forward, one can incorporate advances from modern sequence modeling—such as structured state space models [124, 122] and modern Hopfield networks [218] — to endow the architecture with richer dynamics. Moreover, drawing on insights from the dynamical isometry literature could offer a principled understanding of gradient flow and stability within these enhanced graph-sequence hybrids, potentially leading to both stronger theoretical guarantees and improved empirical performance.

# Bibliography


[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Ahmed M Alaa and Mihaela van der Schaar. Deep multi-task gaussian processes for survival analysis with competing risks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 2326–2334, 2017.

[3] Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. 2021.

[4] Laura Antolini, Patrizia Boracchi, and Elia Biganzoli. A time-dependent discrimination index for survival data. *Statistics in medicine*, 24(24):3927–3944, 2005.

[5] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[6] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[7] Imanol Perez Arribas. Derivatives pricing using signature payoffs. *arXiv preprint arXiv:1809.09466*, 2018.

[8] Alvaro Arroyo, Alvaro Cartea, Fernando Moreno-Pino, and Stefan Zohren. Deep attentive survival analysis in limit order books: Estimating fill probabilities with convolutional-transformers. *Quantitative Finance*, pages 1–23, 2024.

[9] Álvaro Arroyo, Alessio Gravina, Benjamin Gutteridge, Federico Barbero, Claudio Gallicchio, Xiaowen Dong, Michael Bronstein, and Pierre Vandergheynst. On vanishing gradients, over-smoothing, and over-squashing in gnns: Bridging recurrent and graph learning. *arXiv preprint arXiv:2502.10818*, 2025.

[10] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*, 2017.

[11] Anand Avati, Tony Duan, Sharon Zhou, Kenneth Jung, Nigam H Shah, and Andrew Y Ng. Countdown regression: sharp and calibrated survival predictions. In *Uncertainty in Artificial Intelligence*, pages 145–155. PMLR, 2020.

[12] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

[13] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

[16] Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João GM Araújo, Alex Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers need glasses! information over-squashing in language tasks. *arXiv preprint arXiv:2406.04267*, 2024.

[17] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. Sheaf neural networks with connection laplacians. In *Topological, Algebraic and Geometric Learning Workshops 2022*, pages 28–36. PMLR, 2022.

[18] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, and Pietro Liò. Sheaf attention networks. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022.

[19] Federico Barbero, Ameya Velingker, Amin Saberi, Michael Bronstein, and Francesco Di Giovanni. Locality-aware graph-rewiring in gnns. *arXiv preprint arXiv:2310.01668*, 2023.

[20] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[21] Ali Behrouz and Farnoosh Hashemi. Graph Mamba: Towards Learning on Graphs with State Space Models, 2024.

[22] Yoshua Bengio. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[23] Philippe Bergault, Fayçal Drissi, and Olivier Guéant. Multi-asset optimal execution and statistical arbitrage strategies under Ornstein–Uhlenbeck dynamics. *SIAM Journal on Financial Mathematics*, 13(1):353–390, 2022.

[24] Richard Bergna, Sergio Calvo-Ordonez, Felix L Opolka, Pietro Liò, and Jose Miguel Hernandez-Lobato. Uncertainty modeling in graph neural networks via stochastic differential equations. *arXiv preprint arXiv:2408.16115*, 2024.

[25] Mikołaj Bińkowski and Charles-Albert Lehalle. Endogenous dynamics of intraday liquidity. *The Journal of Portfolio Management*, 48(6):145–169, 2022.

[26] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[27] Danilo Blanuša. Über die Einbettung hyperbolischer Räume in euklidische Räume. *Monatsh. Math.*, 59:217–229, 1955.

[28] Cristian Bodnar, Francesco Di Giovanni, Benjamin P. Chamberlain, Pietro Liò, and Michael M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs, 2022.

[29] Viacheslav Borovitskiy, Iskander Azangulov, Alexander Terenin, Peter Mostowsky, Marc Deisenroth, and Nicolas Durrande. Matérn gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2593–2601. PMLR, 2021.

[30] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[31] Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and A. Doucet. Riemannian score-based generative modeling. *ArXiv*, abs/2202.02763, 2022.

[32] David Brander. Isometric embeddings between space forms. 2003. Master thesis.

[33] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[34] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

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

[36] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

[37] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

[38] Sergio Calvo-Ordonez, Jiahao Huang, Lipei Zhang, Guang Yang, Carola-Bibiane Schonlieb, and Angelica I Aviles-Rivero. Beyond u: Making diffusion models faster & lighter. *arXiv preprint arXiv:2310.20092*, 2023.

[39] Sergio Calvo-Ordonez, Matthieu Meunier, Francesco Piatti, and Yuantao Shi. Partially stochastic infinitely deep bayesian neural networks. *arXiv preprint arXiv:2402.03495*, 2024.

[40] Sergio Calvo-Ordoñez, Jonathan Plenk, Richard Bergna, Alvaro Cartea, Jose Miguel Hernandez-Lobato, Konstantina Palla, and Kamil Ciosek. Observation noise and initialization in wide neural networks. *arXiv preprint arXiv:2502.01556*, 2025.

[41] Alvaro Cartea, Ryan Donnelly, and Sebastian Jaimungal. Enhancing trading strategies with order book signals. *Applied Mathematical Finance*, 25(1):1–35, 2018.

[42] Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Execution and statistical arbitrage with signals in multiple automated market makers. *Available at SSRN*, 2023.

[43] Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Predictable losses of liquidity provision in constant function markets and concentrated liquidity markets. *Applied Mathematical Finance*, 30(2):69–93, 2023.

[44] Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Decentralized finance and automated market making: Predictable loss and optimal liquidity provision. *SIAM Journal on Financial Mathematics*, 15(3):931–959, 2024.

[45] Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Decentralised finance and automated market making: Execution and speculation. *Journal of Economic Dynamics and Control*, page 105134, 2025.

[46] Álvaro Cartea, Fayçal Drissi, and Pierre Osselin. Bandits for algorithmic trading with signals. *Available at SSRN 4484004*, 2023.

[47] Álvaro Cartea, Fayçal Drissi, Leandro Sánchez-Betancourt, David Siska, and Lukasz Szpruch. Automated market makers designs beyond constant functions. *Available at SSRN 4459177*, 2023.

[48] Álvaro Cartea, Gerardo Duran-Martin, and Leandro Sánchez-Betancourt. Detecting toxic flow. *arXiv preprint arXiv:2312.05827*, 2023.

[49] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[50] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[51] Álvaro Cartea, Sebastian Jaimungal, and Yixuan Wang. Spoofing and price manipulation in order-driven markets. *Applied Mathematical Finance*, 27(1-2):67–98, 2020.

[52] Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34:1594–1609, 2021.

[53] Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34:1594–1609, 2021.

[54] Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. GRAND: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, pages 1407–1418. PMLR, 2021.

[55] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks, 2019.

[56] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[57] Peter G. Chang, Gerardo Duran-Martin, Alex Shestopaloff, Matt Jones, and Kevin Patrick Murphy. Low-rank extended kalman filtering for online learning of neural networks from streaming data. 232:1025–1071, 22–25 Aug 2023.

[58] Paidamoyo Chapfuwa, Chenyang Tao, Chunyuan Li, Courtney Page, Benjamin Goldstein, Lawrence Carin Duke, and Ricardo Henao. Adversarial time-to-event modeling. In *International Conference on Machine Learning*, pages 735–744. PMLR, 2018.

[59] 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.

[60] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[61] Yuqi Chen, Kan Ren, Yansen Wang, Yuchen Fang, Weiwei Sun, and Dongsheng Li. Contiformer: Continuous-time transformer for irregular time series modeling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[62] Pawel Chilinski and Ricardo Silva. Neural likelihoods via cumulative distribution functions. In *Conference on Uncertainty in Artificial Intelligence*, pages 420–429. PMLR, 2020.

[63] Jin-Wan Cho and Edward Nelling. The probability of limit-order execution. *Financial Analysts Journal*, 56(5):28–33, 2000.

[64] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[65] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[66] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[67] Andrea Cini, Ivan Marisca, Daniele Zambon, and Cesare Alippi. Taming local effects in graph-based spatiotemporal forecasting. *Advances in Neural Information Processing Systems*, 36, 2024.

[68] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[69] Giacomo Como, Raffaele Damiano, and Fabio Fagnani. Discrete bayesian optimization algorithms and applications. 2020.

[70] Enea Monzio Compagnoni, Anna Scampicchio, Luca Biggio, Antonio Orvieto, Thomas Hofmann, and Josef Teichmann. On the effectiveness of randomized signatures as reservoir for learning rough dynamics. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2023.

[71] Fulvio Corsi. A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, 7(2):174–196, 2009.

[72] David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

[73] Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Juan-Pablo Ortega, and Josef Teichmann. Discrete-time signatures and randomness in reservoir computing. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6321–6330, 2021.

[74] Jiaxu Cui and Bo Yang. Graph bayesian optimization: Algorithms, evaluations and applications. *ArXiv*, abs/1805.01157, 2018.

[75] Jiaxu Cui, Bo Yang, Bingyi Sun, Xia Ben Hu, and Jiming Liu. Scalable and parallel deep bayesian optimization on attributed graphs. *IEEE Transactions on Neural Networks and Learning Systems*, 33:103–116, 2020.

[76] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[77] Haitz Sáez de Ocáriz Borde, Alvaro Arroyo, Ismael Morales, Ingmar Posner, and Xiaowen Dong. Neural latent geometry search: Product manifold inference via gromov-hausdorff-informed bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, 2023.

[78] Haitz Sáez de Ocáriz Borde, Alvaro Arroyo, and Ingmar Posner. Projections of model spaces for latent graph inference. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[79] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

[80] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[81] Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pages 7865–7885. PMLR, 2023.

[82] Francesco Di Giovanni, James Rowbottom, Benjamin P Chamberlain, Thomas Markovich, and Michael M Bronstein. Graph neural networks as gradient flows: understanding graph convolutions via energy. *arXiv preprint arXiv:2206.10991*, 2022.

[83] Yuhui Ding, Antonio Orvieto, Bobby He, and Thomas Hofmann. Recurrent distance filtering for graph representation learning. In *Forty-first International Conference on Machine Learning*, 2024.

[84] Lore Dirick, Gerda Claeskens, and Bart Baesens. Time to default in credit scoring using survival analysis: a benchmark study. *Journal of the Operational Research Society*, 68(6):652–665, 2017.

[85] Xiaowen Dong, Dorina Thanou, Michael G. Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36:44–63, 2019.

[86] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International conference on machine learning*, pages 2793–2803. PMLR, 2021.

[87] Fayçal Drissi. Solvability of differential riccati equations and applications to algorithmic trading with signals. *Available at SSRN 4308008*, 2022.

[88] Fayçal Drissi. Models of market liquidity: Applications to traditional markets and automated market makers. *Available at SSRN 4424010*, 2023.

[89] Yilun Du, Katherine Collins, Joshua B. Tenenbaum, and Vincent Sitzmann. Learning signal-agnostic manifolds of neural fields. In *Neural Information Processing Systems*, 2021.

[90] Gerardo Duran-Martin, Matias Altamirano, Alex Shestopaloff, Leandro Sánchez-Betancourt, Jeremias Knoblauch, Matt Jones, Francois-Xavier Briol, and Kevin Patrick Murphy. Outlier-robust kalman filtering through generalised bayes. pages 12138–12171, 2024.

[91] Gerardo Duran-Martin, Aleyna Kara, and Kevin Murphy. Efficient online bayesian inference for neural bandits. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 6002–6021. PMLR, 28–30 Mar 2022.

[92] Gerardo Duran-Martin, Leandro Sánchez-Betancourt, Alex Shestopaloff, and Kevin Patrick Murphy. A unifying framework for generalised bayesian online learning in non-stationary environments. *Transactions on Machine Learning Research*, 2025.

[93] Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

[94] Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In S. Koyejo,

S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22326–22340. Curran Associates, Inc., 2022.

[95] Zoltan Eisler, Jean-Philippe Bouchaud, and Julien Kockelkoren. The price impact of order book events: market orders, limit orders and cancellations. *Quantitative Finance*, 12(9):1395–1419, 2012.

[96] Moshe Eliasof, Eldad Haber, and Eran Treister. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in neural information processing systems*, 34:3836–3849, 2021.

[97] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[98] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2018.

[99] Bastian Epping, Alexandre René, Moritz Helias, and Michael T Schaub. Graph neural networks do not always oversmooth. *arXiv preprint arXiv:2406.02269*, 2024.

[100] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2020.

[101] David Faraggi and Richard Simon. A neural network model for survival data. *Statistics in medicine*, 14(1):73–82, 1995.

[102] Charles Fefferman, Sanjoy K. Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *arXiv: Statistics Theory*, 2013.

[103] Aosong Feng, Irene Li, Yuang Jiang, and Rex Ying. Diffuser: efficient transformers with multi-hop attention diffusion for long sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12772–12780, 2023.

[104] Tamara Fernández, Nicolás Rivera, and Yee Whye Teh. Gaussian processes for survival analysis. *Advances in Neural Information Processing Systems*, 29, 2016.

[105] Ben Finkelshtein, Xingyue Huang, Michael M Bronstein, and Ismail Ilkan Ceylan. Cooperative graph neural networks. In *Forty-first International Conference on Machine Learning*, 2024.

[106] CH Fleming, D Sheldon, WF Fagan, P Leimgruber, T Mueller, D Nandintsetseg, MJ Noonan, KA Olson, Edy Setyawan, A Sianipar, et al. Correcting for missing and irregular data in home-range estimation. *Ecological Applications*, 28(4):1003–1010, 2018.

[107] Marco Fumero, Luca Di Cosmo, Simone Melzi, and Emanuele Rodolà. Learning disentangled representations via product manifold projection. *ArXiv*, abs/2103.01638, 2021.

[108] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[109] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

[110] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

[111] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[112] Jhony H Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D Malliaros. On the trade-off between over-smoothing and over-squashing in deep graph neural networks. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pages 566–576, 2023.

[113] Tilmann Gneiting and Roopesh Ranjan. Comparing density forecasts using threshold-and quantile-weighted scoring rules. *Journal of Business & Economic Statistics*, 29(3):411–422, 2011.

[114] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. In *International conference on machine learning*, pages 7616–7633. PMLR, 2022.

[115] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[116] Dhananjay Gopal, Aniruddha V. Deshmukh, Abhay S. Ranadive, and Shubham Yadav. An introduction to metric spaces. 2020.

[117] M Gori, G Monfardini, and F Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[118] Erika Graf, Claudia Schmoor, Willi Sauerbrei, and Martin Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in medicine*, 18(17-18):2529–2545, 1999.

[119] Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-Symmetric DGN: a stable architecture for Deep Graph Networks. In *The Eleventh International Conference on Learning Representations*, 2023.

[120] Alessio Gravina, Moshe Eliasof, Claudio Gallicchio, Davide Bacciu, and Carola-Bibiane Schönlieb. On oversquashing in graph neural networks through the lens of dynamical systems. In *The 39th Annual AAAI Conference on Artificial Intelligence*, 2025.

[121] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[122] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[123] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[124] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

[125] Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations*, 2018.

[126] Olivier Guéant. *The Financial Mathematics of Market Liquidity: From optimal execution to market making*, volume 33. CRC Press, 2016.

[127] Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. DRew: dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023.

[128] Ben Hambly and Terry Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, pages 109–167, 2010.

[129] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[130] Puneet Handa, Robert Schwartz, and Ashish Tiwari. Quote setting and price formation in an order driven market. *Journal of financial markets*, 6(4):461–489, 2003.

[131] Puneet Handa and Robert A Schwartz. Limit order trading. *The Journal of Finance*, 51(5):1835–1861, 1996.

[132] Frank E Harrell, Robert M Califf, David B Pryor, Kerry L Lee, and Robert A Rosati. Evaluating the yield of medical tests. *Jama*, 247(18):2543–2546, 1982.

[133] Sergiu Hart. Shapley value. In *Game theory*, pages 210–216. Springer, 1989.

[134] K Khalil Hassan et al. Nonlinear systems. *Departement of Electrical and computer Engineering, Michigan State University*, 2002.

[135] Søren Hauberg, Oren Freifeld, and Michael Black. A geometric take on metric learning. *Advances in Neural Information Processing Systems*, 25, 2012.

[136] Jeffrey M Hausdorff and C-K Peng. Multiscaled randomness: A possible source of 1/f noise in biology. *Physical review E*, 54(2):2154, 1996.

[137] Nikolaus Hautsch. *Modelling irregularly spaced financial data: theory and practice of dynamic duration models*. Springer Science & Business Media, 2004.

[138] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[139] Simon Heilig, Alessio Gravina, Alessandro Trenta, Claudio Gallicchio, and Davide Bacciu. Port-hamiltonian architectural bias for long-range propagation in deep graph networks. 2025.

[140] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, pages 2034–2042. PMLR, 2016.

[141] Wolfgang Henke and Wolfgang Nettekoven. The hyperbolic $n$-space as a graph in Euclidean $(6n-6)$-space. *Manuscripta Math.*, 59(1):13–20, 1987.

[142] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[143] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[144] Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021.

[145] Shi Hu, Egill Fridgeirsson, Guido van Wingen, and Max Welling. Transformer-based deep survival analysis. In *Survival Prediction-Algorithms, Challenges and Applications*, pages 132–148. PMLR, 2021.

[146] Chin-Wei Huang, Milad Aghajohari, A. Bose, P. Panangaden, and Aaron C. Courville. Riemannian diffusion models. *ArXiv*, abs/2208.07949, 2022.

[147] Hemant Ishwaran, Udaya B Kogalur, Eugene H Blackstone, and Michael S Lauer. Random survival forests. *The annals of applied statistics*, 2(3):841–860, 2008.

[148] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[149] Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of relu networks. *Advances in Neural Information Processing Systems*, 33:7344–7353, 2020.

[150] Paul Jungeblut, Linda Kleist, and Tillmann Miltzow. The complexity of the hausdorff distance. In *International Symposium on Computational Geometry*, 2021.

[151] Edward L Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[152] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montúfar. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. *arXiv preprint arXiv:2210.11790*, 2022.

[153] Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018.

[154] Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018.

[155] Anees Kazi, Luca Cosmo, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael Bronstein. Differentiable graph module (DGM) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022.

[156] T Anderson Keller, Lyle Muller, Terrence Sejnowski, and Max Welling. Traveling waves encode the recent past and enhance sequence learning. In *The Twelfth International Conference on Learning Representations*, 2023.

[157] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6428, 2020.

[158] Bobak T Kiani, Lukas Fesser, and Melanie Weber. Unitary convolutions for learning on graphs and groups. *arXiv preprint arXiv:2410.05499*, 2024.

[159] Patrick Kidger and Terry Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both cpu and gpu. *arXiv preprint arXiv:2001.00706*, 2020.

[160] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

[161] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[162] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[163] Petter N Kolm, Jeremy Turiel, and Nicholas Westray. Deep order flow imbalance: Extracting alpha at multiple horizons from the limit order book. *Available at SSRN 3900141*, 2021.

[164] Risi Kondor and Jean-Philippe Vert. Diffusion kernels. *kernel methods in computational biology*, pages 171–192, 2004.

[165] Wouter Kool, Herke van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement, 2019.

[166] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[167] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014.

[168] Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. Time-to-event prediction with neural networks and cox regression. *arXiv preprint arXiv:1907.00825*, 2019.

[169] Bart Larivière and Dirk Van den Poel. Investigating the role of product features in preventing customer churn, by using survival analysis and choice modeling: The case of financial services. *Expert Systems with Applications*, 27(2):277–285, 2004.

[170] John A Laurie, Charles G Moertel, Thomas R Fleming, Harry S Wieand, John E Leigh, Jebal Rubin, Greg W McCormack, James B Gerstner, James E Krook, and James Malliard. Surgical adjuvant therapy of large-bowel carcinoma: an evaluation of levamisole and the combination of levamisole and fluorouracil. the north central cancer treatment group and the mayo clinic. *Journal of Clinical Oncology*, 7(10):1447–1456, 1989.

[171] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[172] Changhee Lee, Jinsung Yoon, and Mihaela Van Der Schaar. Dynamic-deephit: A deep learning approach for dynamic survival analysis with competing risks based on

longitudinal data. *IEEE Transactions on Biomedical Engineering*, 67(1):122–133, 2019.

[173] Changhee Lee, William Zame, Jinsung Yoon, and Mihaela Van Der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[174] Changhee Lee, William Zame, Jinsung Yoon, and Mihaela Van Der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[175] Mario Lezcano-Casado and David Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803. PMLR, 2019.

[176] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[177] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[178] Etai Littwin and Lior Wolf. On the convex behavior of deep neural networks in relation to the layers' width. *arXiv preprint arXiv:2001.04878*, 2020.

[179] Andrew W Lo, A Craig MacKinlay, and June Zhang. Econometric models of limit-order executions. *Journal of Financial Economics*, 65(1):31–71, 2002.

[180] Shane Lubold, Arun G Chandrasekhar, and Tyler H McCormick. Identifying the latent space geometry of network models through analysis of curvature. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 85(2):240–292, 2023.

[181] Lorenzo Lucchese, Mikko Pakkanen, and Almut Veraart. The short-term predictability of returns in order book markets: a deep learning perspective. *arXiv preprint arXiv:2211.13777*, 2022.

[182] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[183] Terry J Lyons, Michael Caruana, and Thierry Lévy. *Differential equations driven by rough paths*. Springer, 2007.

[184] Lizheng Ma, Jiaxu Cui, and Bo Yang. Deep neural architecture search with deep graph bayesian optimization. *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 500–507, 2019.

[185] Costis Maglaras, Ciamac Moallemi, and Muye Wang. A Deep Learning Approach to Estimating Fill Probabilities in a Limit Order Book, 2021.

[186] Vladimir Alexandrovich Marchenko and Leonid Andreevich Pastur. Distribution of eigenvalues for some sets of random matrices. *Matematicheskii Sbornik*, 114(4):507–536, 1967.

[187] Thomas Markovich. Qdc: Quantum diffusion convolution kernels on graphs, 2023.

[188] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. *arXiv preprint arXiv:1709.04057*, 2017.

[189] Sohir Maskey, Raffaele Paolino, Aras Bacho, and Gitta Kutyniok. A fractional graph laplacian approach to oversmoothing. *Advances in Neural Information Processing Systems*, 36, 2024.

[190] Emile Mathieu, Charline Le Lan, Chris J Maddison, Ryota Tomioka, and Yee Whye Teh. Continuous hierarchical representations with poincaré variational auto-encoders. *Advances in neural information processing systems*, 32, 2019.

[191] Valentyn Melnychuk, Dennis Frauen, and Stefan Feuerriegel. Causal transformer for estimating counterfactual outcomes. In *International Conference on Machine Learning*, pages 15293–15329. PMLR, 2022.

[192] Alexandru Meterez, Amir Joudaki, Francesco Orabona, Alexander Immer, Gunnar Ratsch, and Hadi Daneshmand. Towards training without depth limits: Batch normalization without gradient explosion. In *The Twelfth International Conference on Learning Representations*, 2024.

[193] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[194] Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders. In *International conference on machine learning*, pages 7045–7054. PMLR, 2020.

[195] Maxime Morariu-Patrichi and Mikko S Pakkanen. State-dependent hawkes processes and their application to limit order book modelling. *Quantitative Finance*, 22(3):563–583, 2022.

[196] Fernando Moreno-Pino, Álvaro Arroyo, Harrison Waldon, Xiaowen Dong, and Álvaro Cartea. Rough transformers for continuous and efficient time-series modelling. *arXiv preprint arXiv:2403.10288*, 2024.

[197] Fernando Moreno-Pino, Pablo M Olmos, and Antonio Artés-Rodríguez. Deep autoregressive models with spectral attention. *Pattern Recognition*, 133:109014, 2023.

[198] Fernando Moreno-Pino and Stefan Zohren. Deepvol: Volatility forecasting from high-frequency data with dilated causal convolutions. *arXiv preprint arXiv:2210.04797*, 2022.

[199] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[200] Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In *International Conference on Machine Learning*, pages 16569–16594. PMLR, 2022.

[201] Alexander Norcliffe, Cristian Bodnar, Ben Day, Jacob Moss, and Pietro Liò. Neural ode processes. In *International Conference on Learning Representations*, 2020.

[202] Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes. *Advances in neural information processing systems*, 33:5911–5921, 2020.

[203] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[204] Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. Combinatorial bayesian optimization using the graph cartesian product. *Advances in Neural Information Processing Systems*, 32, 2019.

[205] YongKyung Oh, Dongyoung Lim, and Sungil Kim. Stable neural stochastic differential equations in analyzing irregular time series data. In *The Twelfth International Conference on Learning Representations*.

[206] Kenta Oono and Taiji Suzuki. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*, 2020.

[207] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[208] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR, 2023.

[209] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR, 2023.

[210] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28, pages III–1310, 2013.

[211] Hongbin Pei, Bingzhen Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *ArXiv*, 2020.

[212] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *Advances in neural information processing systems*, 30, 2017.

[213] Sajida Perveen, Muhammad Shahbaz, Tanzila Saba, Karim Keshavjee, Amjad Rehman, and Aziz Guergachi. Handling irregularly sampled longitudinal data and prognostic modeling of diabetes using machine learning technique. *IEEE Access*, 8:21875–21885, 2020.

[214] David Pfau, Irina Higgins, Aleksandar Botev, and Sébastien Racanière. Disentangling by subspace diffusion. *ArXiv*, abs/2006.12982, 2020.

[215] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ArXiv*, abs/1802.03268, 2018.

[216] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29, 2016.

[217] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems*, 35, 2022.

[218] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

[219] Roger Ratcliff, Philip L Smith, Scott D Brown, and Gail McKoon. Diffusion decision model: Current issues and history. *Trends in cognitive sciences*, 20(4):260–281, 2016.

[220] Jeremy Reizenstein. Calculation of iterated-integral signatures and log signatures. *arXiv preprint arXiv:1712.02757*, 2017.

[221] Jeremy Reizenstein and Benjamin Graham. The iisignature library: efficient calculation of iterated-integral signatures and log signatures. *arXiv preprint arXiv:1802.08252*, 2018.

[222] D Rindt, R Hu, D Steinsaltz, and D Sejdinovic. Survival regression with proper scoring rules and monotonic neural networks. *25th International Conference on Artificial Intelligence and Statistics (AISTATS 2022)*, 2022.

[223] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[224] Andreas Roth and Thomas Liebig. Rank collapse causes over-smoothing and over-correlation in graph neural networks. In *Learning on Graphs Conference*, pages 35–1. PMLR, 2024.

[225] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000.

[226] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. *arXiv preprint arXiv:2006.07556*, 2020.

[227] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

[228] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[229] T Konstantin Rusch, Ben Chamberlain, James Rowbottom, Siddhartha Mishra, and Michael Bronstein. Graph-coupled oscillator networks. In *International Conference on Machine Learning*, pages 18888–18909. PMLR, 2022.

[230] T Konstantin Rusch, Benjamin Paul Chamberlain, Michael W Mahoney, Michael M Bronstein, and Siddhartha Mishra. Gradient gating for deep multi-rate learning on graphs. In *The Eleventh International Conference on Learning Representations*, 2023.

[231] T Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2020.

[232] Haitz Sáez de Ocáriz Borde, Alvaro Arroyo, and Ingmar Posner. Projections of model spaces for latent graph inference. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[233] Haitz Sáez de Ocáriz Borde, Anees Kazi, Federico Barbero, and Pietro Lio. Latent graph inference using product manifolds. *The Eleventh International Conference on Learning Representations*, 2023.

[234] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[235] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[236] Mona Schirmer, Mazin Eltayeb, Stefan Lessmann, and Maja Rudolph. Modeling irregular time series with continuous recurrent units. In *International conference on machine learning*, pages 19388–19405. PMLR, 2022.

[237] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

[238] Michael Scholkemper, Xinyi Wu, Ali Jadbabaie, and Michael T Schaub. Residual connections and normalization can provably prevent oversmoothing in gnns. *arXiv preprint arXiv:2406.02997*, 2024.

[239] Nabeel Seedat, Fergus Imrie, Alexis Bellot, Zhaozhi Qian, and Mihaela van der Schaar. Continuous-time modeling of counterfactual outcomes using neural controlled differential equations. In *International Conference on Machine Learning*, pages 19497–19521. PMLR, 2022.

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

[241] Dai Shi, Andi Han, Lequan Lin, Yi Guo, and Junbin Gao. Exposition on over-squashing problem on gnns: Current methods, benchmarks and challenges, 2023.

[242] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[243] Ankita Shukla, Shagun Uppal, Sarthak Bhagat, Saket Anand, and Pavan K. Turaga. Geometry of deep generative models for disentangled representations. *Proceedings of the 11th Indian Conference on Computer Vision, Graphics and Image Processing*, 2018.

[244] Ritesh Singh and Keshab Mukhopadhyay. Survival analysis in clinical trials: Basics and must know areas. *Perspectives in clinical research*, 2(4):145, 2011.

[245] Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. *arXiv preprint arXiv:1911.08411*, 2019.

[246] Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. *ArXiv*, 2020.

[247] Jimmy TH Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*.

[248] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 144–158. Springer, 2003.

[249] Alessandro Sperduti. Encoding labeled graphs by labeling raam. *Advances in Neural Information Processing Systems*, 6, 1993.

[250] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE transactions on industrial informatics*, 11(3):812–820, 2014.

[251] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[252] Natasa Tagasovska and David Lopez-Paz. Single-model uncertainties for deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[253] Abdel Aziz Taha and Allan Hanbury. An efficient algorithm for calculating the exact hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:2153–2163, 2015.

[254] Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? *arXiv preprint arXiv:1804.11188*, 2018.

[255] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb. Monash university, uea, ucr time series extrinsic regression archive. *arXiv preprint arXiv:2006.10996*, 2020.

[256] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 2000.

[257] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.

[258] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *ArXiv*, abs/2111.14522, 2021.

[259] Amirali Vahid, Moritz Mückschel, Sebastian Stober, Ann-Kathrin Stock, and Christian Beste. Applying deep learning to single-trial eeg data provides evidence for complementary theories on action control. *Communications biology*, 3(1):112, 2020.

[260] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

[262] Benjamin Walker, Andrew D McLeod, Tiexin Qin, Yichuan Cheng, Haoliang Li, and Terry Lyons. Log neural controlled differential equations: The lie brackets make a difference. *arXiv preprint arXiv:2402.18512*, 2024.

[263] X. Wan, H. Kenlay, B. Ru, A. Blaas, M. A. Osborne, and X. Dong. Adversarial attacks on graph classifiers via bayesian optimisation. *Conference on Neural Information Processing Systems*, 2021.

[264] Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. *arXiv preprint arXiv:2402.00789*, 2024.

[265] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the Diffusion Process in Linear Graph Convolutional Networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 5758–5769. Curran Associates, Inc., 2021.

[266] Zifeng Wang and Jimeng Sun. Survtrace: transformers for survival analysis with competing events. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–9, 2022.

[267] Melanie Weber. Curvature and representation learning: Identifying embedding spaces for relational data. 2019.

[268] Lee-Jen Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

[269] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[270] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion. In *The Eleventh International Conference on Learning Representations*, 2023.

[271] Xinyi Wu, Amir Ajorlou, Yifei Wang, Stefanie Jegelka, and Ali Jadbabaie. On the role of attention masks and layernorm in transformers. *arXiv preprint arXiv:2405.18781*, 2024.

[272] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[273] Zheng Xiong, Luisa Zintgraf, Jacob Beck, Risto Vuorio, and Shimon Whiteson. On the practical consistency of meta-reinforcement learning algorithms, 2021.

[274] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[275] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. A mean field theory of batch normalization. *arXiv preprint arXiv:1902.08129*, 2019.

[276] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 40–48, New York, New York, USA, 20–22 Jun 2016. PMLR.

[277] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[278] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

[279] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023.

[280] Sharon Zhang, Amit Moscovich, and Amit Singer. Product manifold learning. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[281] Zihao Zhang and Stefan Zohren. Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units. *arXiv preprint arXiv:2105.10430*, 2021.

[282] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.

[283] Deli Zhao, Jiapeng Zhu, and Bo Zhang. Latent variables on spheres for autoencoders in high dimensions. *arXiv: Learning*, 2019.

[284] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[285] Yin-Cong Zhi, Yin Cheng Ng, and Xiaowen Dong. Gaussian processes on graphs via spectral kernel learning. *IEEE Transactions on Signal and Information Processing over Networks*, 2023.

[286] Qixian Zhong, Jonas W Mueller, and Jane-Ling Wang. Deep extended hazard models for survival analysis. *Advances in Neural Information Processing Systems*, 34:15111–15124, 2021.

[287] Matthias Ziehm and Janet M Thornton. Unlocking the potential of survival data for model organisms through a new database and online analysis platform: S urv c urv. *Aging cell*, 12(5):910–916, 2013.

[288] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016.