Diss. ETH No. 29544

Physics-inspired Machine Learning

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

THORBEN KONSTANTIN RUSCH

M.Sc. Computational Applied Mathematics,
University of Edinburgh
born on 25.08.1995
citizen of Germany

accepted on the recommendation of

Prof. Dr. Siddhartha Mishra, ETH Zurich, examiner
Prof. Dr. Michael W. Mahoney, UC Berkeley, co-examiner
Prof. Dr. Max Welling, University of Amsterdam, co-examiner

2023

Abstract

Physics-inspired machine learning can be seen as incorporating structure from physical systems (e.g., given by ordinary or partial differential equations) into machine learning methods to obtain models with better inductive biases. In this thesis, we provide several of the earliest examples of such methods in the fields of sequence modelling and graph representation learning. We subsequently show that physics-inspired inductive biases can be leveraged to mitigate important and central issues in each particular field. More concretely, we demonstrate that systems of coupled nonlinear oscillators and Hamiltonian systems lead to recurrent sequence models that are able to process sequential interactions over long time scales by mitigating the exploding and vanishing gradients problem. Additionally, we rigorously prove that neural systems of oscillators are universal approximators for continuous and causal operators. Moreover, we show that sequence models derived from multiscale dynamical systems not only mitigate the exploding and vanishing gradients problem (and are thus able to learn long-term dependencies), but equally importantly yield expressive models for learning on (real-world) multiscale data. We further show the impact of physics-inspired approaches on graph representation learning. In particular, systems of graph-coupled nonlinear oscillators denote a powerful framework for learning on graphs that allows for stacking many graph neural network (GNN) layers on top of each other. Thereby, we prove that these systems mitigate the oversmoothing issue in GNNs, where node features exponentially converge to the same constant node vector for increasing number of GNN layers. Finally, we propose to incorporate multiple rates that are inferred from the underlying graph data into the message-passing framework of GNNs. Moreover, we leverage the graph gradient modulated through gating functions to obtain multiple rates that automatically mitigate the oversmoothing issue. We extensively test all proposed methods on a variety of versatile synthetic and real-world datasets, ranging from image recognition, speech recognition, natural language processing (NLP), medical applications, and scientific computing for sequence models, to citation networks, computational chemistry applications, and article and website networks for graph learning models.

Zusammenfassung

Die Einbettung von Strukturen physikalischer Systeme (beispielsweise gegeben durch gewöhnliche oder partielle Differentialgleichungen) in Methoden des maschinellen Lernens, um Modelle mit einem besseren induktiven Bias zu erhalten, kann als physikinspiriertes maschinelles Lernen bezeichnet werden. Wir präsentieren in dieser Arbeit einige der frühesten Beispiele solcher Methoden in den Gebieten der Modellierung von Sequenzen und des Graph-basierten Lernens. Wir zeigen dabei, dass der Physik-basierte induktive Bias dazu genutzt werden kann, um wichtige und zentrale Probleme in beiden Bereichen zu lösen. Wir zeigen beispielsweise, dass Systeme nichtlinearer gekoppelter Oszillatoren und Hamiltonsche Systeme zu rekurrenten Methoden der Modellierung von Sequenzen führen, welche lange sequenzielle Interaktionen verarbeiten können, indem diese das Problem der verschwindenden und explodierenden Gradienten lösen. Zusätzlich zeigen wir, dass neuronale Systeme von Oszillatoren universell in der Klasse kontinuierlicher und kausaler Operatoren zwischen Zeitreihen sind. Wir zeigen weiter, dass Methoden der Modellierung von Sequenzen basierend auf multiskalen dynamischen Systemen nicht nur das Problem der verschwindenden und explodierenden Gradienten lösen, sondern auch expressive Modelle zum Lernen von realen multiskalen Datensätzen darstellen. Des weiteren zeigen wir, wie physikinspiriertes maschinelles Lernen für Graph-strukturierte Anwendungen genutzt werden kann. Dazu ermöglichen insbesondere Graph-gekoppelte nichtlineare Oszillatoren die Konstruktion von tiefen Graph Neuronalen Netzwerken (GNN). Dies ist möglich, da Graph-gekoppelte nichtlineare Oszillatoren das sogenannte oversmoothing Problem lösen, in welchem alle GNN Feature Vektoren zu demselben konstanten Feature Vektor konvergieren. Abschließend schlagen wir den Einsatz von multiplen Raten im message-passing Vorgang in GNN vor, welche direkt von den gegebenen Daten gelernt werden. Zusätzlich dazu modulieren wir die gelernten multiplen Raten mithilfe von Graph-Gradienten und Gating Funktionen, um das oversmoothing Problem für jedes GNN Modell zu lösen. Wir testen alle vorgeschlagenen Methoden umfassend an einer Vielzahl künstlicher und realer Datensätze, die von der Bilderkennung, Spracherkennung, natürlicher Sprachverarbeitung, medizinischen Anwendungen und wissenschaftlichen Rechnen bis hin zu Zitationsnetzwerken, Anwendungen in der computergestützten Chemie, und Artikel und Website Netzwerken reichen.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Siddhartha Mishra, for his continuous support. I deeply appreciate his guidance in research, academia and beyond. It speaks for itself that some of my fondest memories from my time as a PhD student are our hour-long chats and our particularly close collaborations during conference rebuttal phases. I further thank my second supervisor and co-examiner Michael Mahoney for the great collaborations over the last few years. I am particularly grateful that he invited me to spend time with his group at ICSI and UC Berkeley as a visiting researcher, which has been one of the happiest and most exciting times of my life. Moreover, I am very thankful to my co-examiner, Max Welling, for reading my dissertation and the very interesting discussion during my PhD examination. I also thank my close collaborators over the last few years, in particular Michael Bronstein, Benjamin Chamberlain, and Samuel Lanthaler. Furthermore, I would like to acknowledge Tim De Ryck for proofreading my dissertation. Outside of research, I would like to thank my office mates Oliver, Marcello, and Joost, as well as all other members of SAM for making my time as a PhD student such a pleasurable experience. Finally, I would like to thank my family for their support outside of academia, especially my parents, to whom I dedicate this dissertation.

Contents

Introduction

Many problems in science and engineering require understanding and modelling of potentially large amounts of measurement data. Prototypical examples include the design of crafts under physical constraints (e.g., optimizing the lift and the drag across the wing of an aircraft for different operating conditions), climate simulation (e.g., computing the mean global sea surface temperatures that result from different levels of CO2 emissions), and de novo drug design (e.g., leveraging data-based insights from known active binders to form a ligand pharmacophore model for inferring novel structures), to name just a few. Thus, there has been a long record of versatile methods for pattern recognition and data-driven modelling. Early examples include support vector machines [Steinwart and Christmann, 2008], general kernel methods [Shawe-Taylor et al., 2004], clustering algorithms [Xu and Wunsch, 2005], and Bayesian methods for probabilistic inference [Box and Tiao, 2011]. While artificial neural networks have been around for a long time, arguably for centuries in its very simplistic form of linear regression (also known as least squares method), their current widespread use is mainly due to the efficient computation of their gradients with respect to trainable parameters (i.e., backpropagation algorithm [Rosenblatt, 1961, Rumelhart et al., 1985]) and the exponentially increasing amount of available computational resources. In particular the latter enabled training significantly deeper models on large datasets, i.e., artificial neural networks with several layers stacked on top of each other, a field termed *deep learning* [Goodfellow et al., 2016].

Deep learning is now among the most popular approaches in machine learning highly impacting whole sub-fields of computer science and engineering, such as computer vision, autonomous systems and robotics, natural language processing (NLP), and speech recognition. At the core of deep learning lies the deep fully-connected feed-forward neural network (also termed multilayer perceptron [Rosenblatt, 1961, Werbos, 1982, Rumelhart et al., 1985]) which can be described as multiple (hence dubbed "deep") concatenations of affine transformations in alternation with element-wise nonlinear activation functions. More concretely, consider an input $\mathbf{u} \in \mathbb{R}^d$. A deep fully-connected feed-forward neural network (Fig. 1) with $N \in \mathbb{N}$ layers can then be compactly written as,

$$\begin{aligned} \mathbf{h}_n &= \sigma(\mathbf{A}_n \mathbf{h}_{n-1} + \mathbf{b}_n), \quad n = 1, \dots, N-1, \\ \mathbf{h}_N &= \mathbf{A}_N \mathbf{h}_{N-1} + \mathbf{b}_N \end{aligned} \tag{1}$$

with hidden states $\mathbf{h}_n \in \mathbb{R}^{m_n}$, weights $\mathbf{A}_n \in \mathbb{R}^{m_n \times m_{n-1}}$, and biases $\mathbf{b}_n \in \mathbb{R}^{m_n}$ for all layers $n = 1, \dots, N$, initial input given by $\mathbf{h}_0 = \mathbf{u}$ (i.e., $m_0 = d$ is the input dimension), and σ denoting a nonlinear activation function, such as $\tanh(x)$ or $\mathrm{relu}(x) = \max\{0, x\}$ for $x \in \mathbb{R}$, which is applied element-wise. Deep neural networks (1) can be trained by defining an appropriate loss function measuring the difference between the output $\mathbf{h}_N$ of the neural network and the ground-truth data, followed by applying a gradient descent method (such as stochastic gradient descent [Robbins and Monro, 1951]) based on the gradient of the loss function with respect to the parameters $\{\mathbf{A}_n, \mathbf{b}_n\}_{n=1}^N$ of the neural network (1). Thereby, the gradients are efficiently computed using the backpropagation algorithm.

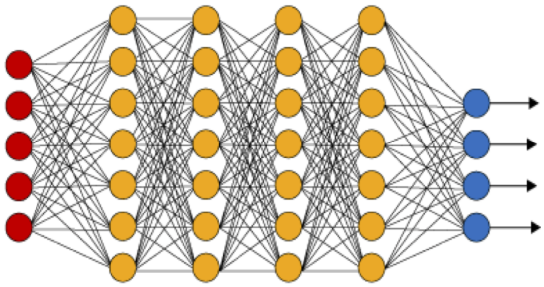

Figure 1: An illustration of a (fully connected) deep neural network. The red neurons represent the inputs to the network and the blue neurons denote the output layer. They are connected by hidden layers with yellow neurons. Each hidden unit (neuron) is connected by affine linear maps between units in different layers and then with nonlinear (scalar) activation functions within units.

While deep fully-connected neural networks can in principle be applied to any collection of data, the tremendous success of deep learning in many areas of application (such as computer vision or NLP) required substantial structural changes of the neural networks. These structural changes are often associated with "inductive biases", emphasizing the incorporated structure reasoned from the underlying data. For example, a class of widely used neural networks in computer vision are convolutional neural networks (CNNs) [LeCun et al., 1989], which apply learnable convolutions to input pixel data, e.g. images. CNNs are known to be shift equivariant [Fukushima, 1980, Bronstein et al., 2021, McGreivy and Hakim, 2022], due to weight sharing. This inductive bias, biologically-inspired by local receptive fields [LeCun and Bengio, 1995], enables CNNs to significantly outperform fully-connected neural networks on image data.

While many neural network architectures are biologically-inspired there is an increasing interest in drawing inspirations from physics to construct novel machine learning models which can successfully be applied to problems in the physical sciences. Although many approaches in this context do not possess physics-based inductive biases, but rather apply well-established models from computer vision, NLP or graph learning to problems in physics, e.g., [Sanchez-Gonzalez et al., 2020, Vlachas et al., 2020, Stachenfeld et al., 2021], machine learning models with physics-inspired inductive biases are increasingly used for applications in the physical sciences. For instance, equivariant neural networks and Lie algebra based approaches are used in the context of molecular sciences, e.g. molecular simulations [Batzner et al., 2022], and molecular property predictions [Satorras et al., 2021]. While equivariance is an important concept in the physical sciences, there are more symmetries and structures one can potentially leverage to build neural network architectures with better inductive biases. For instance conservation of energy, such as in Hamiltonian neural networks [Greydanus et al., 2019, Chen et al., 2020c], where the neural network is parameterized according to a Hamiltonian. This enables the network to learn Hamiltonian systems from data, which automatically preserve energy (i.e., the Hamiltonian) over time.

Physics-inspired inductive biases for neural networks are not limited to applications in the physical sciences. On the contrary, recent physics-inspired machine learning methods are used in many classical fields of deep learning. For instance in generative modelling, where diffusion models [Yang et al., 2022] are state-of-the-art. These models leverage a physical process, namely diffusion, to incrementally corrupt input data with random noise. Once the training is finished, new data can be generated by inverting the diffusion process. Since this approach is grounded in physical processes, it is no surprise that many models in this context leverage insights from physical systems to construct diffusion models with better inductive biases [Dockhorn et al., 2021, Lai et al., 2022, Salimans and Ho, 2021]. Another example includes graph representation learning, where physics-inspired graph neural networks (GNNs) are increasingly

used. For instance, grounding fundamental GNN operations (so-called message-passing [Bronstein et al., 2021]) in anisotropic diffusion [Chamberlain et al., 2021b], gradient flows [Di Giovanni et al., 2022], reaction-diffusion equations [Choi et al., 2022], or hyperbolic partial differential equations (PDEs) [Eliasof et al., 2021], to name just a few.

In this thesis, we present several of the earliest examples of physics-inspired machine learning models. More concretely, we present recurrent sequence models inspired by systems of nonlinear coupled oscillators, (time-dependent) Hamiltonian systems, and multiscale dynamical systems. We subsequently show that these models, due to their physics-based inductive biases, exhibit very favorable properties, such as the ability to handle long-term interactions in sequential data, or improve the expressive power on real-world datasets. Moreover, we present physics-inspired models for graph representation learning, which are inspired by graph-dynamical systems, namely graph-coupled oscillators as well as multi-rate graph-dynamical systems. We further prove various desirable properties for these models, such as the mitigation of a central issue in graph representation learning using GNNs, i.e., oversmoothing [Nt and Maehara, 2019, Oono and Suzuki, 2020], as well as handling heterophilic (large-scale) graph datasets.

Outline

The first part of this thesis focuses on sequence modeling from a physics-inspired perspective. We begin Part I by introducing sequence models, namely Transformers and Recurrent Neural Networks (RNNs), and propose three novel physics-inspired RNN architectures subsequently. The first architecture we propose is called Coupled Oscillatory RNN (coRNN). We introduce coRNN in Chapter 2 and provide a rigorous theoretical analysis of the proposed model in section 2.2. Moreover, we provide an extensive empirical evaluation of coRNN in section 2.3. This chapter is based on the publication [Rusch and Mishra, 2021a]. We further introduce the Undamped Independent Controlled Oscillatory RNN (UnICORNN) in Chapter 3. Likewise, we provide a rigorous analysis in section 3.2 as well as an extensive empirical evaluation of the proposed UnICORNN architecture in section 3.3. This chapter is based on the publication [Rusch and Mishra, 2021b]. In Chapter 4, we generalize both coRNN and UnICORNN by introducing neural oscillators. Subsequently, we rigorously prove that neural oscillators are universal. This chapter is based on the preprint [Lanthaler et al., 2023]. In the last chapter of part one, i.e. Chapter 5, we propose the Long Expressive Memory (LEM), an expressive recurrent sequence model for learning long-term dependencies. In section 5.2, we rigorously prove gradient stability of the proposed LEM and show that LEM is a universal approximator for general (Lipschitz continuous) multiscale dynamical systems. Moreover, we show that LEM emulates the heterogeneous multiscale method for ordinary differential equations (ODEs). Finally, in section 5.3, we provide a variety of versatile numerical experiments for the proposed LEM architecture. This chapter is based on the publication [Rusch et al., 2022b].

The second part of this thesis deals with physics-inspired graph representation learning. We start Part II by introducing modern Graph Neural Networks and the underlying message-passing framework. Moreover, we outline important limitations of current frameworks, in particular the oversmoothing issue in deep GNNs (the provided Definition of oversmoothing is based on [Rusch et al., 2023]). Subsequently, we propose two physics-inspired message-passing frameworks. The first approach we propose is called Graph-Coupled Oscillator Network (GraphCON) in Chapter 7. We further provide a rigorous analysis of GraphCON in Section 7.2 as well as extensive empirical evaluations in Section 7.3. This chapter is based on the publication [Rusch et al., 2022a]. The final chapter of this part, i.e., Chapter 8, introduces the Gradient Gating framework ($\mathbf{G}^2$). We provide theoretical insights into the proposed $\mathbf{G}^2$ framework in Section 8.2 and numerically test its performance on a variety of (large-scale) graph datasets in Section 8.3. This chapter is based on the publication [Rusch et al., 2022a]. Last but not least, we conclude this thesis in Chapter 9.

List of publications

The content of this thesis is based on the results obtained in the following papers.

[1] Samuel Lanthaler, **T. Konstantin Rusch**, Siddhartha Mishra: "Neural Oscillators are Universal". *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*. 2023.

[2] **T. Konstantin Rusch**, Michael M. Bronstein, Siddhartha Mishra: "A Survey on Oversmoothing in Graph Neural Networks". *arXiv preprint*. 2023.

[3] **T. Konstantin Rusch**, Benjamin P. Chamberlain, Michael W. Mahoney, Michael M. Bronstein, Siddhartha Mishra: "Gradient gating for deep multi-rate learning on graphs". *Proceedings of the 11th International Conference on Learning Representations (ICLR)*. 2023.

[4] **T. Konstantin Rusch**, Benjamin P. Chamberlain, James Rowbottom, Siddhartha Mishra, Michael M. Bronstein: "Graph-Coupled Oscillator Networks". *Proceedings of the 39th International Conference on Machine Learning (ICML)*. 2022.

[5] **T. Konstantin Rusch**, Siddhartha Mishra, N. Benjamin Erichson, Michael W. Mahoney: "Long expressive memory for sequence modeling". *Proceedings of the 10th International Conference on Learning Representations (ICLR)*. 2022.

[6] **T. Konstantin Rusch**, Siddhartha Mishra: "UnICORNN: A recurrent model for learning very long time dependencies". *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021; PMLR 139:9168-9178.

[7] **T. Konstantin Rusch**, Siddhartha Mishra: "Coupled Oscillatory Recurrent Neural Network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies". *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. 2021.

Other contributions:

[8] Francesco Di Giovanni, **T. Konstantin Rusch**, Michael M. Bronstein, Andreea Deac, Marc Lackenby, Siddhartha Mishra, Petar Veličković: "How does over-squashing affect the power of GNNs?". *arXiv preprint*. 2023.

[9] Léonard Equer, **T. Konstantin Rusch**, Siddhartha Mishra: "Multi-Scale Message Passing Neural PDE Solvers". *ICLR Workshop on Physics for Machine Learning*. 2023.

[10] Marcello Longo, Siddhartha Mishra, **T. Konstantin Rusch**, Christoph Schwab: "Higher-order Quasi-Monte Carlo Training of Deep Neural Networks". *SIAM Journal on Scientific Computing*. 2021; 43(6), A3938–A3966.

[11] Siddhartha Mishra, **T. Konstantin Rusch**: "Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences".
SIAM Journal on Numerical Analysis. 2021; 59(3), 1811–1834.

Part I

Physics-inspired Sequence Modeling

Chapter 1

Introduction to Sequence Modeling

Learning tasks with sequential data as inputs (and possibly outputs) arise in a wide variety of contexts, including computer vision, text and speech recognition, natural language processing, and time series analysis in the sciences and engineering, among others. Sequence models are the prevalent choice of methods in this context and come in a variety of different flavours.

In this thesis we focus on recurrent neural networks (RNNs). RNNs incorporate temporal structure into neural networks by recurrently processing input sequences, i.e., following the recurrent update rule,

$$\mathbf{y}_n = \mathbf{F}_\theta(\mathbf{y}_{n-1}, \mathbf{u}_n), \quad \forall n = 1, \dots, N, \tag{1.1}$$

with hidden states $\mathbf{y}_n \in \mathbb{R}^m$, input $\mathbf{u}_n \in \mathbb{R}^d$, nonlinear function $\mathbf{F}_\theta$, trainable parameters θ (i.e., weights and biases of the RNN), and the initial value $\mathbf{y}_0 \in \mathbb{R}^m$, which is set to the zero vector in most cases. Specific choices of the nonlinear function $\mathbf{F}_\theta$ yield popular RNN architectures such as Long Short-Term Memories (LSTMs) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Units (GRUs) [Cho et al., 2014].

While RNNs have been successfully used in processing sequential data sets, it is well-known that training these models to process (very) long sequential inputs is extremely challenging on account of the so-called *exploding and vanishing gradients problem* [Pascanu et al., 2013]. This arises as calculating hidden state gradients in the backpropagation through time algorithm (BPTT) entails the computation of an iterative product of gradients over a large number of steps. Consequently, this (long) product can easily grow or decay exponentially in the number of recurrent interactions. More concretely, we train an RNN (1.1) to minimize a loss function for instance given as the mean-squared error (summed over time),

$$\mathcal{E} := \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}_n, \quad \mathcal{E}_n = \frac{1}{2} \|\mathbf{y}_n - \bar{\mathbf{y}}_n\|_2^2, \tag{1.2}$$

with $\bar{\mathbf{y}}$ being the underlying ground truth (training data). During training, we compute gradients of the loss function (1.2) with respect to the weights and biases θ of the underlying RNN (1.1) at every step of the gradient descent procedure, i.e.,

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{E}_n}{\partial \theta}. \tag{1.3}$$

Following Pascanu et al. [2013], one uses the chain rule to show

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{1 \le k \le n} \frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}, \quad \frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \frac{\partial \mathcal{E}_n}{\partial \mathbf{y}_n} \frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_k} \frac{\partial^+ \mathbf{y}_k}{\partial \theta}. \tag{1.4}$$

Here, the notation $\frac{\partial^+ \mathbf{y}_k}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{y}_k$ with respect to the parameter θ, while keeping the other arguments constant. In general, for recurrent models, the partial gradient $\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}$, which measures the contribution to the hidden state gradient at step n arising from step k of the model, can behave as $\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} \sim \gamma^{n-k}$, for some $\gamma > 0$ [Pascanu et al., 2013] due to the long product,

$$\frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_k} = \prod_{k<i\leq n} \frac{\partial \mathbf{y}_i}{\partial \mathbf{y}_{i-1}},$$

in $\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}$ (1.4). If $\gamma > 1$, then the partial gradient grows exponentially in sequence length, for long-term dependencies $k << n$, leading to the *exploding gradient problem*. On the other hand, if $\gamma < 1$, then the partial gradient decays exponentially for $k << n$, leading to the *vanishing gradient problem*. Thus, mitigation of the exploding and vanishing gradients problem entails deriving bounds on the gradients.

Existing RNN architectures that address the exploding and vanishing gradients problem include gated networks such as LSTMs [Hochreiter and Schmidhuber, 1997] and GRUs [Cho et al., 2014], where an additive structure of the gradients is leveraged to stabilize training. However, it is known that LSTMs and GRUs still suffer from the exploding gradients problem. Moreover, these networks still struggle to learn very long-term dependencies Li et al. [2018]. Other attempts enforce constrains on the structure of the hidden-to-hidden weight matrices of the RNN, i.e., to be orthogonal or unitary Henaff et al. [2016], Arjovsky et al. [2016], Wisdom et al. [2016], Kerg et al. [2019]. However, these structural constraints significantly impair the expressivity of the network. The challenge of designing novel RNN architectures is thus to ensure their ability to learn long-term dependencies (i.e., mitigating the exploding and vanishing gradients problem) while at the same time improve (and in particular not sacrifice) their expressive power.

Other recent and very popular models in the context of sequence modelling include so-called Transformers [Vaswani et al., 2017], such as BERT [Devlin et al., 2018], RoBERTa [Liu et al., 2019a], and GPT [Radford et al., 2018], that rely on a self-attention mechanism. This mechanism, introduced in Vaswani et al. [2017], allows these models to efficiently process input sequences in a non-recurrent manner. Another class of efficient sequence models are temporal convolutional networks (TCNs) [Bai et al., 2018, Romero et al., 2021b,a]. TCNs process input sequences based on dilated causal 1D convolutional layers. Finally, state-space models have recently experienced a renaissance in modern sequence modeling by incorporating a specific structure into the hidden-to-hidden weight matrix (i.e., HiPPO matrix [Gu et al., 2020]) that leads to a very efficient and fast training. These structural state-space models for sequence modeling (S4) [Gu et al., 2021] and their variants [Hasani et al., 2022, Smith et al., 2022] have successfully been applied to challenging sequential long-range tasks, as well as tasks in natural language processing and audio processing.

In this part we propose several novel RNN architectures with physics-inspired inductive biases (namely that of nonlinear coupled oscillators, Hamiltonian systems, and multiscale dynamical systems), which are able to process sequential inputs over very long time-scales. In particular, we show that these models exhibit several favorable properties, such as memory efficiency as well as fast training and inference speed. Moreover, we rigorously prove that these methods successfully mitigate the exploding and vanishing gradients problem and exhibit high expressive power on various challenging (real-world) sequential datasets.

Chapter 2

Coupled Oscillatory Recurrent Neural Network

Coupled networks of controlled nonlinear forced and damped oscillators arise in many physical, engineering and biological systems, ensuring expressive representations while constraining the dynamics of state variables and their gradients. This motivates us to propose a novel architecture for RNNs, based on time-discretizations of second-order systems of nonlinear ordinary differential equations (ODEs) (2.1) that model coupled oscillators. Under verifiable hypotheses, we are able to *rigorously prove precise bounds on the hidden states of these RNNs and their gradients, enabling a possible solution of the exploding and vanishing gradients problem*, while demonstrating through benchmark numerical experiments, that the resulting system still retains sufficient expressivity, i.e., ability to process complex inputs, with a competitive performance, with respect to the state-of-the-art, on a variety of sequential learning tasks.

2.1 The proposed RNN

Our proposed RNN is based on the following second-order system of ODEs,

$$\mathbf{y}'' = \sigma\left(\mathbf{W}\mathbf{y} + \mathcal{W}\mathbf{y}' + \mathbf{V}\mathbf{u} + \mathbf{b}\right) - \gamma\mathbf{y} - \epsilon\mathbf{y}'. \tag{2.1}$$

Here, $t \in [0,1]$ is the (continuous) time variable, $\mathbf{u} = \mathbf{u}(t) \in \mathbb{R}^d$ is the time-dependent *input signal*, $\mathbf{y} = \mathbf{y}(t) \in \mathbb{R}^m$ is the *hidden state* of the RNN with $\mathbf{W}, \mathcal{W} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{m\times d}$ are weight matrices, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector and $0 < \gamma, \epsilon$ are parameters, representing oscillation frequency and the amount of damping (friction) in the system, respectively. $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is the *activation function*, set to $\sigma(u) = \tanh(u)$ here. By introducing the so-called *velocity* variable $\mathbf{z} = \mathbf{y}'(t) \in \mathbb{R}^m$, we rewrite (2.1) as the first-order system:

$$\mathbf{y}' = \mathbf{z}, \quad \mathbf{z}' = \sigma\left(\mathbf{W}\mathbf{y} + \mathcal{W}\mathbf{z} + \mathbf{V}\mathbf{u} + \mathbf{b}\right) - \gamma\mathbf{y} - \epsilon\mathbf{z}. \tag{2.2}$$

We fix a timestep $0 < \Delta t < 1$ and define our proposed RNN hidden states at time $t_n = n\Delta t \in [0,1]$ (while omitting the affine output state) as the following IMEX (implicit-explicit) discretization of the first order system (2.2):

$$\begin{aligned} \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t\mathbf{z}_n, \\ \mathbf{z}_n &= \mathbf{z}_{n-1} + \Delta t\sigma\left(\mathbf{W}\mathbf{y}_{n-1} + \mathcal{W}\mathbf{z}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}\right) - \Delta t\gamma\mathbf{y}_{n-1} - \Delta t\epsilon\mathbf{z}_{\bar{n}}, \end{aligned} \tag{2.3}$$

with either $\bar{n} = n$ or $\bar{n} = n - 1$. Note that the only difference in the two versions of the RNN (2.3) lies in the implicit ($\bar{n} = n$) or explicit ($\bar{n} = n - 1$) treatment of the damping term $-\epsilon\mathbf{z}$ in (2.2), whereas both versions retain the implicit treatment of the first equation in (2.2).

Motivation and background. To see that the underlying ODE (2.2) models a *coupled network of controlled forced and damped nonlinear oscillators*, we start with the single neuron (scalar) case by setting $d = m = 1$ in (2.1) and assume an identity activation function $\sigma(x) = x$. Setting $\mathbf{W} = \mathcal{W} = \mathbf{V} = \mathbf{b} = \epsilon = 0$ leads to the simple ODE, $\mathbf{y}'' + \gamma\mathbf{y} = 0$, which exactly models *simple harmonic motion* with *frequency* γ, for instance that of a mass attached to a spring [Guckenheimer and Holmes, 1990]. Letting $\epsilon > 0$ in (2.1) adds *damping* or friction to the system [Guckenheimer and Holmes, 1990]. Then, by introducing non-zero $\mathbf{V}$ in (2.1), we drive the system with a driving force proportional to the input signal $\mathbf{u}(t)$. The parameters $\mathbf{V}, \mathbf{b}$ modulate the effect of the driving force, $\mathbf{W}$ controls the frequency of oscillations and $\mathcal{W}$ the amount of damping in the system. Finally, the tanh activation mediates a nonlinear response in the oscillator. In the coupled network (2.2) with $m > 1$, each neuron updates its hidden state based on the input signal as well as information from other neurons. The diagonal entries of $\mathbf{W}$ (and the scalar hyperparameter γ) control the frequency whereas the diagonal entries of $\mathcal{W}$ (and the hyperparameter ϵ) determine the amount of damping for each neuron, respectively, whereas the non-diagonal entries of these matrices modulate interactions between neurons. Hence, given this behavior of the underlying ODE (2.2), we term the RNN (2.3) as a *coupled oscillatory Recurrent Neural Network* (coRNN).

Oscillator networks are ubiquitous in nature and in engineering systems [Guckenheimer and Holmes, 1990, Strogatz, 2001] with canonical examples being pendulums (classical mechanics), business cycles (economics), heartbeat (biology) for single oscillators and electrical circuits for networks of oscillators. Our motivating examples arise in neurobiology, where individual biological neurons can be viewed as oscillators with periodic spiking and firing of the action potential. Moreover, functional circuits of the brain, such as cortical columns and prefrontal-striatal-hippocampal circuits, are being increasingly interpreted by networks of oscillatory neurons, see Stiefel and Ermentrout [2016] for an overview. Following well-established paths in machine learning, such as for convolutional neural networks [LeCun et al., 2015], our focus here is to abstract the essence of functional brain circuits being networks of oscillators and design an RNN based on much simpler mechanistic systems, such as those modeled by (2.2), while ignoring the complicated biological details of neural function.

Related work. There is an increasing trend of basing RNN architectures on ODEs and dynamical systems. These approaches can roughly be classified into two branches, namely RNNs based on discretized ODEs and continuous-time RNNs. Examples of continuous-time approaches include neural ODEs [Chen et al., 2018] with ODE-RNNs [Rubanova et al., 2019] as its recurrent extension as well as E [2017] and references therein, to name just a few. We focus, however, in this chapter on an ODE-inspired discrete-time RNN, as the proposed coRNN is derived from a discretization of the ODE (2.1). A good example for a discrete-time ODE-based RNNs is the so-called *anti-symmetric* RNN of Chang et al. [2019], where the RNN architecture is based on a stable ODE resulting from a skew-symmetric hidden weight matrix, thus constraining the stable (gradient) dynamics of the network. This approach has much in common with previously mentioned unitary/orthogonal/non-normal RNNs in constraining the structure of the hidden-to-hidden layer weight matrices. However, adding such strong constraints might reduce expressivity of the resulting RNN and might lead to inadequate performance on complex tasks. In contrast to these approaches, our proposed coRNN does not explicitly constrain the weight matrices but relies on the dynamics of the underlying ODE (and the IMEX discretization (2.3)), to provide gradient stability. Moreover, no gating mechanisms as in LSTMs/GRUs are used in the current version of coRNN. There is also an increasing interest in designing *hybrid* methods, which use a discretization of an ODE (in

particular a Hamiltonian system) in order to learn the continuous representation of the data, see for instance Greydanus et al. [2019], Chen et al. [2020c]. Overall, our approach here differs from these papers in our use of networks of oscillators to build the RNN.

2.2 Rigorous analysis of coRNN

2.2.1 Continuous-time system

In the following, we present bounds that show how the continuous time dynamics of the ODE system (2.2), modeling nonlinear damped and forced networks of oscillators, is constrained. We start with the following estimate on the *energy* of the solutions of the system (2.2).

Proposition 2.2.1. *Let $\mathbf{y}(t), \mathbf{z}(t)$ be the solutions of the ODE system (2.2) at any time $t \in [0, T]$ and assume that the damping parameter $\epsilon \geq \frac{1}{2}$ and the initial data for (2.2) is given by,*

$$\mathbf{y}(0) = \mathbf{z}(0) \equiv 0.$$

Then, the solutions are bounded as,

$$\mathbf{y}(t)^\top \mathbf{y}(t) \leq \frac{mt}{\gamma}, \quad \mathbf{z}(t)^\top \mathbf{z}(t) \leq mt, \quad \forall t \in (0, T]. \tag{2.4}$$

Proof. To prove this proposition, we multiply the first equation in (2.2) with $y(t)^\top$ and the second equation in (2.2) with $\frac{1}{\gamma} z(t)^\top$ to obtain,

$$\frac{d}{dt}\left(\frac{\mathbf{y}(t)^\top \mathbf{y}(t)}{2} + \frac{\mathbf{z}(t)^\top \mathbf{z}(t)}{2\gamma}\right) = \frac{\mathbf{z}(t)^\top \sigma(\mathbf{A}(t))}{\gamma} - \frac{\epsilon}{\gamma}\mathbf{z}(t)^\top \mathbf{z}(t), \tag{2.5}$$

with

$$\mathbf{A}(t) = \mathbf{W}\mathbf{y}(t) + \mathcal{W}\mathbf{z}(t) + \mathbf{V}\mathbf{u}(t) + \mathbf{b}.$$

Using the elementary Cauchy's inequality repeatedly in (2.5) results in,

$$\begin{aligned}\frac{d}{dt}\left(\frac{\mathbf{y}(t)^\top \mathbf{y}(t)}{2} + \frac{\mathbf{z}(t)^\top \mathbf{z}(t)}{2\gamma}\right) &\leq \frac{\sigma(A)^\top \sigma(A)}{2\gamma} + \frac{1}{\gamma}\left(\frac{1}{2} - \epsilon\right)\mathbf{z}^\top \mathbf{z} \\ &\leq \frac{m}{2\gamma} \quad (\text{as } |\sigma| \leq 1 \text{ and } \epsilon \geq \frac{1}{2}).\end{aligned}$$

Integrating the above inequality over the time interval $[0, t]$ and using the fact that the initial data are $\mathbf{y}(0) = \mathbf{z}(0) \equiv 0$, we obtain the bounds (2.4). □

The above proposition and estimate (2.4) clearly demonstrate that the dynamics of the network of coupled nonlinear oscillators (2.1) is bounded. The fact that the nonlinear activation function $\sigma = \tanh$ is uniformly bounded in its arguments played a crucial role in deriving the energy bound (2.4). A straightforward adaptation of this argument leads to the following proposition about the sensitivity of the system to inputs,

Proposition 2.2.2. *Let $\mathbf{y}(t), \mathbf{z}(t)$ be the solutions of the ODE system (2.2) with respect to the input signal $\mathbf{u}(t)$. Let $\bar{\mathbf{y}}(t), \bar{\mathbf{z}}(t)$ be the solutions of the ODE system (2.2), but with respect to the input signal $\bar{\mathbf{u}}(t)$. Assume that the damping parameter $\epsilon \geq \frac{1}{2}$ and the initial data are given by,*

$$\mathbf{y}(0) = \mathbf{z}(0) = \bar{\mathbf{y}}(0) = \bar{\mathbf{z}}(0) \equiv 0.$$

Then we have the following bound,

$$(\mathbf{y}(t)-\bar{\mathbf{y}}(t))^\top(\mathbf{y}(t)-\bar{\mathbf{y}}(t)) \leq \frac{4mt}{\gamma}, \quad (\mathbf{z}(t)-\bar{\mathbf{z}}(t))^\top(\mathbf{z}(t)-\bar{\mathbf{z}}(t)) \leq 4mt, \quad \forall t \in (0,T]. \tag{2.6}$$

Thus from the bound (2.6), there can be *atmost* linear separation (in time) with respect to the trajectories of the ODE (2.2) for different input signals. Hence, chaotic behavior, which is characterized by the (super-)exponential separation of trajectories is ruled out by the structure of the ODE system (2.2). Note that this property of the ODE system was primarily a result of the uniform boundedness of the activation function σ. Using a different activation function such as ReLU might enable to obtain an exponential separation of trajectories that is a prerequisite for a chaotic dynamical system.

We further bound the gradient dynamics corresponding to the ODE system (2.2). Therefore, let θ denote the i,j-th entry of the weight matrices $\mathbf{W}, \mathcal{W}, \mathbf{V}$ or the i-th entry of the bias vector $\mathbf{b}$. We are interested in finding out how the gradients of the hidden state $\mathbf{y}$ (and the auxiliary hidden state $\mathbf{z}$) with respect to parameter θ, vary with time. Note that these gradients are precisely the objects of interest in the training of an RNN, based on a discretization of the ODE system (2.2). To this end, we differentiate (2.2) with respect to the parameter θ and denote

$$\mathbf{y}_\theta(t) = \frac{\partial \mathbf{y}}{\partial \theta}(t), \ \mathbf{z}_\theta(t) = \frac{\partial \mathbf{z}}{\partial \theta}(t),$$

to obtain,

$$\begin{aligned} \mathbf{y}'_\theta &= \mathbf{z}_\theta, \\ \mathbf{z}'_\theta &= \operatorname{diag}(\sigma'(\mathbf{A}))\left[\mathbf{W}\mathbf{y}_\theta + \mathcal{W}\mathbf{z}_\theta\right] + \mathbf{Z}^{i,j}_{m,\bar{m}}(\mathbf{A})\rho - \gamma\mathbf{y}_\theta - \epsilon\mathbf{z}_\theta. \end{aligned} \tag{2.7}$$

Here, $\mathbf{Z}^{i,j}_{m,\bar{m}}(\mathbf{A}) \in \mathbb{R}^{m\times\bar{m}}$ is a matrix with all elements are zero except for the (i,j)-th entry which is set to $\sigma'(\mathbf{A}(t))_i$, i.e. the i-th entry of $\sigma'(\mathbf{A})$, and we have,

$$\begin{aligned} &\rho = \mathbf{y}, \quad \bar{m} = m, \quad \text{if } \theta = \mathbf{W}_{i,j}, \\ &\rho = \mathbf{z}, \quad \bar{m} = m, \quad \text{if } \theta = \mathcal{W}_{i,j}, \\ &\rho = \mathbf{u}, \quad \bar{m} = d, \quad \text{if } \theta = \mathbf{V}_{i,j}, \\ &\rho = \mathbf{1}, \quad \bar{m} = 1, \quad \text{if } \theta = \mathbf{b}_i. \end{aligned}$$

We see from (2.7) that the ODEs governing the gradients with respect to the parameter θ also represent a system of oscillators but with additional coupling and forcing terms, proportional to the hidden states $\mathbf{y}, \mathbf{z}$ or input signal $\mathbf{u}$. As we have already proved with estimate (2.4) that the hidden states are always bounded and the input signal is assumed to be bounded, it is natural to expect that the gradients of the states with respect to θ are also bounded. We make this statement explicit in the following proposition, which for simplicity of exposition, we consider the case of $\theta = \mathbf{W}_{i,j}$, as the other values of θ are very similar in their behavior.

Proposition 2.2.3. *Let $\theta = \mathbf{W}_{i,j}$ and $\mathbf{y}, \mathbf{z}$ be the solutions of the ODE system* (2.2)*. Assume that the weights and the damping parameter satisfy,*

$$\|\mathbf{W}\|_\infty + \|\mathcal{W}\|_\infty \leq \epsilon,$$

then we have the following bounds on the gradients,

$$\begin{aligned} \mathbf{y}_\theta(t)^\top\mathbf{y}_\theta(t) + \frac{1}{\gamma}\left(\mathbf{z}_\theta(t)^\top\mathbf{z}_\theta(t)\right) &\leq \left[\mathbf{y}_\theta(0)^\top\mathbf{y}_\theta(0) + \frac{1}{\gamma}\left(\mathbf{z}_\theta(0)^\top\mathbf{z}_\theta(0)\right)\right]e^{Ct} + \frac{mt^2}{2\gamma^2}, \quad t \in (0,T], \\ C &= \max\left\{\frac{\|\mathbf{W}\|_1}{\gamma}, 1 + \|\mathcal{W}\|_1\right\}. \end{aligned} \tag{2.8}$$

The proof of this proposition follows exactly along the same lines as the proof of proposition 2.2.1 and we skip the details, while noting the crucial role played by the energy bound (2.4). We remark that the bound (2.8) indicates that as long as the initial gradients with respect to θ are bounded and the weights are controlled by the damping parameter, the hidden state gradients remain bounded in time.

2.2.2 Discretized system with implicit damping

Having established the attractive features of the ODE system (2.2) (i.e., bounded hidden states as well as bounded gradients), one can expect that a suitable discretization of the ODE (2.2) that preserves these bounds will not have exploding gradients. We claim that one such *structure preserving discretization* is given by the IMEX discretization that results in the RNN (2.3) and proceed to derive bounds on this RNN below.

Following standard practice we set $\mathbf{y}(0) = \mathbf{z}(0) = 0$ and purely for the simplicity of exposition, we set the control parameters, $\epsilon = \gamma = 1$ and $\bar{n} = n$ in (2.3) leading to,

$$\begin{aligned} \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t \mathbf{z}_n, \\ \mathbf{z}_n &= \tfrac{\mathbf{z}_{n-1}}{1+\Delta t} + \tfrac{\Delta t}{1+\Delta t}\sigma(\mathbf{A}_{n-1}) - \tfrac{\Delta t}{1+\Delta t}\mathbf{y}_{n-1}, \ \mathbf{A}_{n-1} := \mathbf{W}\mathbf{y}_{n-1} + \mathcal{W}\mathbf{z}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}. \end{aligned} \tag{2.9}$$

Analogous results and proofs for the case where $\bar{n} = n-1$ and for general values of ϵ, γ follow subsequently.

As with the underlying ODE (2.2), the hidden states of the RNN (2.3) are bounded, i.e.

Proposition 2.2.4. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states of the RNN (2.9) for $1 \le n \le N$, then the hidden states satisfy the following (energy) bounds:*

$$\mathbf{y}_n^\top \mathbf{y}_n + \mathbf{z}_n^\top \mathbf{z}_n \le nm\Delta t = mt_n \le m. \tag{2.10}$$

Proof. We prove this proposition by multiplying $(\mathbf{y}_{n-1}^\top, \mathbf{z}_n^\top)$ to (2.3) and using the elementary identities,

$$\mathbf{a}^\top(\mathbf{a}-\mathbf{b}) = \frac{\mathbf{a}^\top\mathbf{a}}{2} - \frac{\mathbf{b}^\top\mathbf{b}}{2} + \frac{1}{2}(\mathbf{a}-\mathbf{b})^\top(\mathbf{a}-\mathbf{b}), \quad \mathbf{b}^\top(\mathbf{a}-\mathbf{b}) = \frac{\mathbf{a}^\top\mathbf{a}}{2} - \frac{\mathbf{b}^\top\mathbf{b}}{2} - \frac{1}{2}(\mathbf{a}-\mathbf{b})^\top(\mathbf{a}-\mathbf{b}),$$

to obtain the following,

$$\begin{aligned} \frac{\mathbf{y}_n^\top\mathbf{y}_n + \mathbf{z}_n^\top\mathbf{z}_n}{2} &= \frac{\mathbf{y}_{n-1}^\top\mathbf{y}_{n-1} + \mathbf{z}_{n-1}^\top\mathbf{z}_{n-1}}{2} + \frac{(\mathbf{y}_n - \mathbf{y}_{n-1})^\top(\mathbf{y}_n - \mathbf{y}_{n-1})}{2} \\ &\quad - \frac{(\mathbf{z}_n - \mathbf{z}_{n-1})^\top(\mathbf{z}_n - \mathbf{z}_{n-1})}{2} + \Delta t \mathbf{z}_n^\top\sigma(\mathbf{A}_{n-1}) - \Delta t \mathbf{z}_n^\top\mathbf{z}_n \\ &\le \frac{\mathbf{y}_{n-1}^\top\mathbf{y}_{n-1} + \mathbf{z}_{n-1}^\top\mathbf{z}_{n-1}}{2} + \Delta t\,(1/2 + \Delta t/2 - 1)\,\mathbf{z}_n^\top\mathbf{z}_n + \frac{\Delta t}{2}\sigma^\top(\mathbf{A}_{n-1})\sigma(\mathbf{A}_{n-1}) \\ &\le \frac{\mathbf{y}_{n-1}^\top\mathbf{y}_{n-1} + \mathbf{z}_{n-1}^\top\mathbf{z}_{n-1}}{2} + \frac{m\Delta t}{2} \quad \text{as } \sigma^2 \le 1 \text{ and } \epsilon > \Delta t << 1. \end{aligned}$$

Iterating the above inequality n times leads to the energy bound,

$$\mathbf{y}_n^\top\mathbf{y}_n + \mathbf{z}_n^\top\mathbf{z}_n \le \mathbf{y}_0^\top\mathbf{y}_0 + \mathbf{z}_0^\top\mathbf{z}_0 + nm\Delta t = mt_n, \tag{2.11}$$

as $\mathbf{y}_0 = \mathbf{z}_0 = 0$. □

Next, we examine how changes in the input signal $\mathbf{u}$ affect the dynamics. We have the following proposition:

Proposition 2.2.5. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states of the trained RNN (2.9) with respect to the input $\mathbf{u} = \{\mathbf{u}_n\}_{n=1}^N$ and let $\overline{\mathbf{y}}_n, \overline{\mathbf{z}}_n$ be the hidden states of the same RNN (2.9), but with respect to the input $\overline{\mathbf{u}} = \{\overline{\mathbf{u}}_n\}_{n=1}^N$, then the differences in the hidden states are bounded by,*

$$(\mathbf{y}_n - \overline{\mathbf{y}}_n)^\top (\mathbf{y}_n - \overline{\mathbf{y}}_n) + (\mathbf{z}_n - \overline{\mathbf{z}}_n)^\top (\mathbf{z}_n - \overline{\mathbf{z}}_n) \leq 4mt_n. \tag{2.12}$$

The proof of this proposition is completely analogous to the proof of proposition 2.2.4, we subtract

$$\begin{array}{ll} \overline{\mathbf{y}}_n & = \overline{\mathbf{y}}_{n-1} + \Delta t \overline{\mathbf{z}}_n, \\ \overline{\mathbf{z}}_n & = \frac{\overline{\mathbf{z}}_{n-1}}{1+\Delta t} + \frac{\Delta t}{1+\Delta t}\sigma(\overline{\mathbf{A}}_{n-1}) - \frac{\Delta t}{1+\Delta t}\overline{\mathbf{y}}_{n-1}, \ \overline{\mathbf{A}}_{n-1} := \mathbf{W}\overline{\mathbf{y}}_{n-1} + \mathcal{W}\overline{\mathbf{z}}_{n-1} + \mathbf{V}\overline{\mathbf{u}}_n + \mathbf{b}. \end{array} \tag{2.13}$$

from (2.9) and multiply $\left((\mathbf{y}_n - \overline{\mathbf{y}}_n)^\top, (\mathbf{z}_n - \overline{\mathbf{z}}_n)^\top\right)$ to the difference. The estimate (2.12) follows identically to the proof of (2.10) (presented above) by realizing that $\sigma(\mathbf{A}_{n-1}) - \sigma(\overline{\mathbf{A}}_{n-1}) \leq 2$.

Note that the bound (2.12) ensures that the hidden states can only separate linearly in time for changes in the input. Thus, chaotic behavior, such as for Duffing type oscillators, characterized by at least exponential separation of trajectories, is ruled out for this proposed RNN, showing that it is stable with respect to changes in the input. This is largely on account of the fact that the activation function σ in (2.3) is globally bounded.

On the exploding gradient problem. We continue to provide rigorous bounds on the gradients of the discretized system (2.3). To this end, we train the RNN (2.3) to minimize the loss function,

$$\mathcal{E} := \frac{1}{N}\sum_{n=1}^N \mathcal{E}_n, \quad \mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n - \bar{\mathbf{y}}_n\|_2^2, \tag{2.14}$$

with $\bar{\mathbf{y}}$ being the underlying ground truth (training data). During training, we compute gradients of the loss function (2.14) with respect to the weights and biases $\mathbf{\Theta} = [\mathbf{W}, \mathcal{W}, \mathbf{V}, \mathbf{b}]$, i.e.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{1}{N}\sum_{n=1}^N \frac{\partial \mathcal{E}_n}{\partial \theta}, \quad \forall\ \theta \in \mathbf{\Theta}. \tag{2.15}$$

Proposition 2.2.6. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states generated by the RNN (2.9). We assume that the time step $\Delta t << 1$ can be chosen such that,*

$$\max\left\{\frac{\Delta t(1+\|\mathbf{W}\|_\infty)}{1+\Delta t}, \frac{\Delta t\|\mathcal{W}\|_\infty}{1+\Delta t}\right\} = \eta \leq \Delta t^r, \quad \frac{1}{2} \leq r \leq 1. \tag{2.16}$$

Denoting $\delta = \frac{1}{1+\Delta t}$, the gradient of the loss function $\mathcal{E}$ (2.14) with respect to any parameter $\theta \in \mathbf{\Theta}$ is bounded as,

$$\left|\frac{\partial \mathcal{E}}{\partial \theta}\right| \leq \frac{3}{2}\left(m + \bar{Y}\sqrt{m}\right), \tag{2.17}$$

with $\bar{Y} = \max\limits_{1\leq n\leq N} \|\bar{\mathbf{y}}_n\|_\infty$ be a bound on the underlying training data.

Proof. Denoting $\mathbf{X}_n = [\mathbf{y}_n, \mathbf{z}_n]$, we can apply the chain rule repeatedly (for instance as in Pascanu et al. [2013]) to obtain,

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{1\leq k\leq n} \underbrace{\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n}\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k}\frac{\partial^+ \mathbf{X}_k}{\partial \theta}}_{\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}}. \tag{2.18}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k$ with respect to the parameter θ, while keeping the other arguments constant. This quantity can be readily calculated from the structure of the RNN (2.9),

$$\frac{\partial^+ \mathbf{X}_k}{\partial \theta} = \begin{cases} \left[\left(\frac{\Delta t^2}{1+\Delta t} \mathbf{Z}^{i,j}_{m,m}(\mathbf{A}_{k-1}) \mathbf{y}_{k-1} \right)^\top, \left(\frac{\Delta t}{1+\Delta t} \mathbf{Z}^{i,j}_{m,m}(\mathbf{A}_{k-1}) \mathbf{y}_{k-1} \right)^\top \right]^\top & \text{if} \quad \theta = (i,j)-\text{th entry of } \mathbf{W}, \\ \left[\left(\frac{\Delta t^2}{1+\Delta t} \mathbf{Z}^{i,j}_{m,m}(\mathbf{A}_{k-1}) \mathbf{z}_{k-1} \right)^\top, \left(\frac{\Delta t}{1+\Delta t} \mathbf{Z}^{i,j}_{m,m}(\mathbf{A}_{k-1}) \mathbf{z}_{k-1} \right)^\top \right]^\top & \text{if} \quad \theta = (i,j)-\text{th entry of } \mathcal{W}, \\ \left[\left(\frac{\Delta t^2}{1+\Delta t} \mathbf{Z}^{i,j}_{m,d}(\mathbf{A}_{k-1}) \mathbf{u}_k \right)^\top, \left(\frac{\Delta t}{1+\Delta t} \mathbf{Z}^{i,j}_{m,d}(\mathbf{A}_{k-1}) \mathbf{u}_k \right)^\top \right]^\top & \text{if} \quad \theta = (i,j)-\text{th entry of } \mathbf{V}, \\ \left[\left(\frac{\Delta t^2}{1+\Delta t} \mathbf{Z}^{i,1}_{m,1}(\mathbf{A}_{k-1}) \right)^\top, \left(\frac{\Delta t}{1+\Delta t} \mathbf{Z}^{i,1}_{m,1}(\mathbf{A}_{k-1}) \right)^\top \right]^\top & \text{if} \quad \theta = i-\text{th entry of } \mathbf{b}, \end{cases} \tag{2.19}$$

where $\mathbf{Z}^{i,j}_{m,\bar{m}}(\mathbf{A}_{k-1}) \in \mathbb{R}^{m \times \bar{m}}$ is a matrix with all elements are zero except for the (i,j)-th entry which is set to $\sigma'(\mathbf{A}_{k-1})_i$, i.e. the i-th entry of $\sigma'(\mathbf{A}_{k-1})$. We easily see that $\|\mathbf{Z}^{i,j}_{m,\bar{m}}(\mathbf{A}_{k-1})\|_\infty \leq 1$ for all $i, j, m, \bar{m}$ and all choices of $\mathbf{A}_{k-1}$. Moreover, from (2.14), we readily calculate that,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n} = [\mathbf{y}_n - \bar{\mathbf{y}}_n, 0]\,. \tag{2.20}$$

Further repeated application of the chain rule and a direct calculation with (2.9) yields,

$$\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} = \prod_{k<i\leq n} \frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}}, \quad \frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}} = \begin{bmatrix} \mathrm{I} + \Delta t \mathbf{B}_{i-1} & \Delta t \mathbf{C}_{i-1} \\ \mathbf{B}_{i-1} & \mathbf{C}_{i-1} \end{bmatrix}, \tag{2.21}$$

where I is the identity matrix and

$$\mathbf{B}_{i-1} = \delta \Delta t \left(\mathrm{diag}(\sigma'(\mathbf{A}_{i-1})) \mathbf{W} - \mathrm{I} \right), \quad \mathbf{C}_{i-1} = \delta \left(\mathrm{I} + \Delta t\, \mathrm{diag}(\sigma'(\mathbf{A}_{i-1})) \mathcal{W} \right). \tag{2.22}$$

It is straightforward to calculate using the assumption (2.16) that $\|\mathbf{B}_{i-1}\|_\infty < \eta$ and $\|\mathbf{C}_{i-1}\|_\infty \leq \eta + \delta$. Using the definitions of matrix norms and (2.16), we obtain:

$$\begin{aligned} \left\| \frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}} \right\|_\infty &\leq \max\left(1 + \Delta t(\|\mathbf{B}_{i-1}\|_\infty + \|\mathbf{C}_{i-1}\|_\infty), \|\mathbf{B}_{i-1}\|_\infty + \|\mathbf{C}_{i-1}\|_\infty\right) \\ &\leq \max\left(1 + \Delta t(\delta + 2\eta), \delta + 2\eta\right) \leq 1 + 3\Delta t^r. \end{aligned} \tag{2.23}$$

Therefore, using (2.21), we have

$$\left\| \frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} \right\|_\infty \leq \prod_{k<i\leq n} \left\| \frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}} \right\|_\infty \leq (1 + 3\Delta t^r)^{n-k} \approx 1 + 3(n-k)\Delta t^r. \tag{2.24}$$

Note that we have used an expansion around 1 and neglected terms of $\mathcal{O}(\Delta t^{2r})$ as $\Delta t << 1$. We remark that the bound (2.23) is the *crux of our argument* about gradient control as we see from the structure of the RNN that the recurrent matrices have close to unit norm.

Now, using definitions of matrix and vector norms and applying (2.24) in (2.18), together with (2.20) and (2.19), we obtain the following estimate on the norm:

$$\left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| \leq \begin{cases} (\|\mathbf{y}_n\|_\infty + \|\bar{\mathbf{y}}_n\|_\infty)(1+3(n-k)\Delta t^r)\delta\Delta t\|\mathbf{y}_{k-1}\|_\infty, & \text{if} \quad \theta \text{ is entry of } \mathbf{W}, \\ (\|\mathbf{y}_n\|_\infty + \|\bar{\mathbf{y}}_n\|_\infty)(1+3(n-k)\Delta t^r)\delta\Delta t\|\mathbf{z}_{k-1}\|_\infty, & \text{if} \quad \theta \text{ is entry of } \mathcal{W}, \\ (\|\mathbf{y}_n\|_\infty + \|\bar{\mathbf{y}}_n\|_\infty)(1+3(n-k)\Delta t^r)\delta\Delta t\|\mathbf{u}_k\|_\infty, & \text{if} \quad \theta \text{ is entry of } \mathbf{V}, \\ (\|\mathbf{y}_n\|_\infty + \|\bar{\mathbf{y}}_n\|_\infty)(1+3(n-k)\Delta t^r)\delta\Delta t, & \text{if} \quad \theta \text{ is entry of } \mathbf{b}. \end{cases} \tag{2.25}$$

We will estimate the above term, just for the case of θ is an entry of $\mathbf{W}$, the rest of the terms are very similar to estimate.

For simplicity of notation, we let $k-1 \approx k$ and aim to estimate the term,

$$\begin{aligned} \left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| &\leq \|\mathbf{y}_n\|_\infty\|\mathbf{y}_k\|_\infty(1+3(n-k)\Delta t^r)\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\|\mathbf{y}_k\|_\infty(1+3(n-k)\Delta t^r)\delta\Delta t \\ &\leq m\sqrt{nk}\Delta t(1+3(n-k)\Delta t^r)\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{mk}\sqrt{\Delta t}(1+3(n-k)\Delta t^r)\delta\Delta t \quad \text{(by (2.10))} \\ &\leq m\sqrt{nk}\delta\Delta t^2 + 3m\sqrt{nk}(n-k)\delta\Delta t^{r+2} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{mk}\sqrt{\Delta t}(1+3(n-k)\Delta t^r)\delta\Delta t. \end{aligned} \tag{2.26}$$

To further analyze the above estimate, we recall that $n\Delta t = t_n \leq 1$ and consider two different regimes. Let us start by considering *short-term dependencies* by letting $k \approx n$, i.e $n-k=c$ with constant $c \sim \mathcal{O}(1)$, independent of n, k. In this case, a straightforward application of the above assumptions in the bound (2.26) yields,

$$\begin{aligned} \left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| &\leq m\sqrt{nk}\delta\Delta t^2 + 3m\sqrt{nk}(n-k)\delta\Delta t^{r+2} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\sqrt{t_n}\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\sqrt{t_n}c\delta\Delta t^{r+1} \\ &\leq mt_n\delta\Delta t + mct_n\delta\Delta t^{r+1} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\sqrt{t_n}\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\sqrt{t_n}c\delta\Delta t^{r+1} \\ &\leq t_n m\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\sqrt{t_n}\delta\Delta t \quad \text{(for } \Delta t << 1 \text{ as } r \geq 1/2) \\ &\leq m\delta\Delta t + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t. \end{aligned} \tag{2.27}$$

Next, we consider *long-term dependencies* by setting $k << n$ and estimating,

$$\begin{aligned} \left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| &\leq m\sqrt{nk}\delta\Delta t^2 + 3m\sqrt{nk}(n-k)\delta\Delta t^{r+2} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t^{\frac{3}{2}} + 3\|\bar{\mathbf{y}}_n\|_\infty\sqrt{mn}\delta\Delta t^{r+\frac{3}{2}} \\ &\leq m\sqrt{t_n}\delta\Delta t^{\frac{3}{2}} + 3mt_n^{\frac{3}{2}}\delta\Delta t^{r+\frac{1}{2}} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t^{\frac{3}{2}} + 3\|\bar{\mathbf{y}}_n\|_\infty\sqrt{mt_n}\delta\Delta t^{r+\frac{1}{2}} \\ &\leq m\delta\Delta t^{\frac{3}{2}} + 3m\delta\Delta t^{r+\frac{1}{2}} + \|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t^{\frac{3}{2}} + 3\|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t^{r+\frac{1}{2}} \quad (\text{as } t_n < 1) \\ &\leq 3m\delta\Delta t^{r+\frac{1}{2}} + 3\|\bar{\mathbf{y}}_n\|_\infty\sqrt{m}\delta\Delta t^{r+\frac{1}{2}} \quad (\text{as } r \leq 1 \text{ and } \Delta t << 1). \end{aligned} \tag{2.28}$$

Thus, in all cases, we have that,

$$\left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| \leq 3\delta\Delta t\left(m + \sqrt{m}\|\bar{\mathbf{y}}_n\|_\infty\right) \quad (\text{as } r \geq 1/2). \tag{2.29}$$

Applying the above estimate in (2.18) allows us to bound the gradient by,

$$\left|\frac{\partial \mathcal{E}_n}{\partial \theta}\right| \leq \sum_{1\leq k\leq n}\left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right| \leq 3\delta t_n\left(m + \sqrt{m}\|\bar{\mathbf{y}}_n\|_\infty\right). \tag{2.30}$$

Therefore, the gradient of the loss function (2.14) can be bounded as,

$$\begin{aligned}
\left|\frac{\partial \mathcal{E}}{\partial \theta}\right| &\leq \frac{1}{N}\sum_{n=1}^{N}\left|\frac{\partial \mathcal{E}_n}{\partial \theta}\right| \\
&\leq 3\delta\left[\frac{m\Delta t}{N}\sum_{n=1}^{N} n + \frac{\sqrt{m}\Delta t}{N}\sum_{n=1}^{N}\|\bar{\mathbf{y}}_n\|_\infty n\right] \\
&\leq 3\delta\left[\frac{m\Delta t}{N}\sum_{n=1}^{N} n + \frac{\sqrt{m}\bar{Y}\Delta t}{N}\sum_{n=1}^{N} n\right] \\
&\leq \frac{3}{2}\delta(N+1)\Delta t\left(m+\bar{Y}\sqrt{m}\right) \\
&\leq \frac{3}{2}\delta(t_N+\Delta t)\left(m+\bar{Y}\sqrt{m}\right) \\
&\leq \frac{3}{2}\delta(1+\Delta t)\left(m+\bar{Y}\sqrt{m}\right) \qquad (\text{as } t_N = 1) \\
&\leq \frac{3}{2}\left(m+\bar{Y}\sqrt{m}\right),
\end{aligned} \tag{2.31}$$

which is the desired estimate (2.17). □

As the entire gradient of the loss function (2.14), with respect to the weights and biases of the network, is bounded above in (2.17), the *exploding gradient problem* is mitigated for the proposed coRNN architecture.

On the assumption (2.16) **and training.** Note that all the estimates were based on the fact that we were able to choose a time step Δt in (2.3) that enforces the condition (2.16). For any fixed weights $\mathbf{W}, \mathcal{W}$, we can indeed choose such a value of ϵ to satisfy (2.16). However, we *train* the RNN to find the weights that minimize the loss function (2.14). Can we find a hyperparameter Δt such that (2.16) is satisfied at every step of the stochastic gradient descent method for training?

To investigate this issue, we consider a simple gradient descent method of the form:

$$\theta_{\ell+1} = \theta_\ell - \zeta\frac{\partial \mathcal{E}}{\partial \theta}(\theta_\ell). \tag{2.32}$$

Note that ζ is the constant (non-adapted) learning rate. We assume for simplicity that $\theta_0 = 0$ (other choices lead to the addition of a constant). Then, a straightforward estimate on the weight is given by,

$$\begin{aligned}
|\theta_{\ell+1}| &\leq |\theta_\ell| + \zeta\left|\frac{\partial \mathcal{E}}{\partial \theta}(\theta_\ell)\right| \\
&\leq |\theta_\ell| + \zeta\frac{3}{2}\left(m+\bar{Y}\sqrt{m}\right) \quad (\text{by (2.31)}) \\
&\leq |\theta_0| + \ell\zeta\frac{3}{2}\left(m+\bar{Y}\sqrt{m}\right) = \ell\zeta\frac{3}{2}\left(m+\bar{Y}\sqrt{m}\right).
\end{aligned} \tag{2.33}$$

In order to calculate the minimum number of steps L in the gradient descent method (2.32) such that the condition (2.16) is satisfied, we set $\ell = L$ in (2.33) and applying it to the condition (2.16) leads to the straightforward estimate,

$$L \geq \frac{1}{\zeta\frac{3}{2}\left(m+\bar{Y}\sqrt{m}\right)m\Delta t^{1-r}\delta}. \tag{2.34}$$

Note that the parameter is set to $\delta < 1$, while in general, the learning rate is set to $\zeta << 1$. Thus, as long as $r \leq 1$, we see that the assumption (2.16) holds for a large number of steps of the gradient descent method. We remark that the above estimate (2.34) is a large underestimate on L. In the experiments we present in Section 2.3, we are able to take a very large number of training steps, while the gradients remain within a range (see Fig. 2.4).

On the vanishing gradient problem. The vanishing gradient problem [Pascanu et al., 2013] arises if $\left|\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}\right|$, defined in (2.18), $\to 0$ exponentially fast in k, for $k << n$ (long-term dependencies). In that case, the RNN does not have long-term memory, as the contribution of the k-th hidden state to error at time step t_n is infinitesimally small. We already see from (2.24) that $\left\|\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k}\right\|_\infty \approx 1$ (independently of k). Thus, we should not expect the products in (2.18) to decay fast. In fact, we will provide a much more precise characterization of this gradient. To this end, we introduce the following *order*-notation,

$$\begin{aligned} &\beta = \mathcal{O}(\alpha), \text{for } \alpha, \beta \in \mathbb{R}_+ \quad \text{if there exist constants } \overline{C}, \underline{C} \text{ such that } \underline{C}\alpha \leq \beta \leq \overline{C}\alpha. \\ &\mathbf{M} = \mathcal{O}(\alpha), \text{for } \mathbf{M} \in \mathbb{R}^{d_1 \times d_2}, \alpha \in \mathbb{R}_+ \quad \text{if there exists a constant } \overline{C} \text{ such that } \|\mathbf{M}\| \leq \overline{C}\alpha. \end{aligned} \tag{2.35}$$

For simplicity of notation, we will also set $\bar{\mathbf{y}}_n = \mathbf{u}_n \equiv 0$, for all n, $\mathbf{b} = 0$ and $r = 1$ in (2.16) and we will only consider $\theta = \mathbf{W}_{i,j}$ for some $1 \leq i, j \leq m$ in the following proposition.

Proposition 2.2.7. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN* (2.9)*. Under the assumption that $\mathbf{y}_n^i = \mathcal{O}(\sqrt{t_n})$, for all $1 \leq i \leq m$ and* (2.16)*, the gradient for long-term dependencies satisfies,*

$$\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \mathcal{O}\left(\hat{c}\delta\Delta t^{\frac{3}{2}}\right) + \mathcal{O}\left(\hat{c}\delta(1+\delta)\Delta t^{\frac{5}{2}}\right) + \mathcal{O}(\Delta t^3),\ \hat{c} = sech^2\left(\sqrt{k\Delta t}(1+\Delta t)\right),\ k << n. \tag{2.36}$$

Proof. We start with the following decomposition of the recurrent matrices:

$$\begin{aligned} &\frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}} = M_{i-1} + \Delta t \tilde{M}_{i-1}, \\ &M_{i-1} := \begin{bmatrix} \mathrm{I} & \Delta t \mathbf{C}_{i-1} \\ \mathbf{B}_{i-1} & \mathbf{C}_{i-1} \end{bmatrix}, \quad \tilde{M}_{i-1} := \begin{bmatrix} \mathbf{B}_{i-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \end{aligned}$$

with $\mathbf{B}, \mathbf{C}$ defined in (2.22). By the assumption (2.16), one can readily check that $\|\tilde{M}_{i-1}\|_\infty \leq \Delta t$, for all $k \leq i \leq n-1$.

We will use an induction argument to show the following representation formula for the product of Jacobians,

$$\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} = \prod_{k<i\leq n} \frac{\partial \mathbf{X}_i}{\partial \mathbf{X}_{i-1}} = \begin{bmatrix} \mathrm{I} & \Delta t \sum_{j=k}^{n-1} \prod_{i=j}^{k} \mathbf{C}_i \\ \mathbf{B}_{n-1} + \sum_{j=n-2}^{k} \left(\prod_{i=n-1}^{j+1} \mathbf{C}_i\right) \mathbf{B}_j & \prod_{i=n-1}^{k} \mathbf{C}_i \end{bmatrix} + \mathcal{O}(\Delta t). \tag{2.37}$$

We start by the outermost product and calculate,

$$\begin{aligned} \frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_{n-1}} \frac{\partial \mathbf{X}_{n-1}}{\partial \mathbf{X}_{n-2}} &= \left(M_{n-1} + \Delta t \tilde{M}_{n-1}\right)\left(M_{n-2} + \Delta t \tilde{M}_{n-2}\right) \\ &= M_{n-1}M_{n-2} + \Delta t(\tilde{M}_{n-1}M_{n-2} + M_{n-1}\tilde{M}_{n-2}) + \mathcal{O}(\Delta t^2). \end{aligned}$$

By direct multiplication, we obtain,

$$M_{n-1}M_{n-2} = \begin{bmatrix} \mathrm{I} & \Delta t\left(\mathbf{C}_{n-2} + \mathbf{C}_{n-1}\mathbf{C}_{n-2}\right) \\ \mathbf{B}_{n-1} + \mathbf{C}_{n-1}\mathbf{B}_{n-2} & \mathbf{C}_{n-1}\mathbf{C}_{n-2} \end{bmatrix} + \Delta t \begin{bmatrix} \mathbf{C}_{n-1}\mathbf{B}_{n-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{n-1}\mathbf{C}_{n-2} \end{bmatrix}.$$

Using the definitions in (2.22) and (2.16), we can easily see that

$$\begin{bmatrix} \mathbf{C}_{n-1}\mathbf{B}_{n-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{n-1}\mathbf{C}_{n-2} \end{bmatrix} = \mathcal{O}(\Delta t).$$

Similarly, it is easy to show that

$$\tilde{M}_{n-1}M_{n-2}, M_{n-1}\tilde{M}_{n-2} \sim \mathcal{O}(\Delta t).$$

Plugging all the above estimates yields,

$$\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_{n-1}}\frac{\partial \mathbf{X}_{n-1}}{\partial \mathbf{X}_{n-2}} = \begin{bmatrix} \mathrm{I} & \Delta t\left(\mathbf{C}_{n-2} + \mathbf{C}_{n-1}\mathbf{C}_{n-2}\right) \\ \mathbf{B}_{n-1} + \mathbf{C}_{n-1}\mathbf{B}_{n-2} & \mathbf{C}_{n-1}\mathbf{C}_{n-2} \end{bmatrix} + \mathcal{O}(\Delta t^2),$$

which is exactly the form of the leading term (2.37).

Iterating the above calculations $(n-k)$ times and realizing that $(n-k)\Delta t^2 \approx n\Delta t^2 = t_n \Delta t$ yields the formula (2.37).

Recall that we have set $\theta = \mathbf{W}_{i,j}$, for some $1 \leq i, j \leq m$ in proposition 2.2.7. Directly calculating with (2.20), (2.19) and the representation formula (2.37) yields the formula,

$$\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \mathbf{y}_n^\top \Delta t^2 \delta \mathbf{Z}_{m,m}^{i,j}(\mathbf{A}_{k-1})\mathbf{y}_{k-1} + \mathbf{y}_n^\top \Delta t^2 \delta \mathbf{C}^* \mathbf{Z}_{m,m}^{i,j}(\mathbf{A}_{k-1})\mathbf{y}_{k-1} + \mathcal{O}(\Delta t^3), \tag{2.38}$$

with matrix $\mathbf{C}^*$ defined as,

$$\mathbf{C}^* := \sum_{j=k}^{n-1}\prod_{i=j}^{k}\mathbf{C}_i,$$

and $\mathbf{Z}_{m,m}^{i,j}(\mathbf{A}_{k-1}) \in \mathbb{R}^{m\times m}$ is a matrix with all elements are zero except for the (i,j)-th entry which is set to $\sigma'(a_{k-1}^i)$, i.e. the i-th entry of $\sigma'(\mathbf{A}_{k-1})$.

Note that the formula (2.38) can be explicitly written as,

$$\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \delta \Delta t^2 \sigma'(a_{k-1}^i)\mathbf{y}_n^i \mathbf{y}_{k-1}^j + \delta \Delta t^2 \sigma'(a_{k-1}^i)\sum_{\ell=1}^{m}\mathbf{C}_{\ell i}^* \mathbf{y}_n^\ell \mathbf{y}_{k-1}^j + \mathcal{O}(\Delta t^3), \tag{2.39}$$

with $\mathbf{y}_n^j$ denoting the j-th element of vector $\mathbf{y}_n$, and

$$a_{k-1}^i := \sum_{\ell=1}^{m}\mathbf{W}_{i\ell}\mathbf{y}_{k-1}^\ell + \sum_{\ell=1}^{m}\mathcal{W}_{i\ell}\mathbf{z}_{k-1}^\ell. \tag{2.40}$$

By the assumption (2.16), we can readily see that

$$\|\mathbf{W}\|_\infty, \|\mathcal{W}\|_\infty \leq 1 + \Delta t.$$

Therefore by the fact that $\sigma' = sech^2$, the assumption $\mathbf{y}_k^i = \mathcal{O}(\sqrt{t_k})$ and (2.40), we obtain,

$$\hat{c} = sech^2(\sqrt{k\Delta t}(1+\Delta t) \leq \sigma'(a_i^{k-1}) \leq 1. \tag{2.41}$$

Using (2.41) in (2.39), we obtain,

$$\delta \Delta t^2 \sigma'(a_{k-1}^i)\mathbf{y}_n^i \mathbf{y}_{k-1}^j = \mathcal{O}\left(\hat{c}\delta\Delta t^{\frac{5}{2}}\right). \tag{2.42}$$

Using the definition of $\mathbf{C}_i$, we can expand the product in $\mathbf{C}^*$ and neglect terms of order $\mathcal{O}(\Delta t^4)$, to obtain

$$\prod_{i=j}^{k} \mathbf{C}_i = (\mathcal{O}(1) + \mathcal{O}((j-k+1)\delta\Delta t^2))\mathrm{I}.$$

Summing over j and using the fact that $k << n$, we obtain that

$$\mathbf{C}^* = (\mathcal{O}(n) + \mathcal{O}(\delta\Delta t^0))\mathrm{I}. \tag{2.43}$$

Plugging (2.43) and (2.41) into (2.39) leads to,

$$\delta \Delta t^2 \sigma'(a_{k-1}^i) \sum_{\ell=1}^{m} \mathbf{C}_{\ell i}^* \mathbf{y}_n^\ell \mathbf{y}_{k-1}^j = \mathcal{O}\left(\hat{c}\delta\Delta t^{\frac{3}{2}}\right) + \mathcal{O}\left(\hat{c}\delta^2\Delta t^{\frac{5}{2}}\right). \tag{2.44}$$

Combining (2.42) and (2.44) yields the desired estimate (2.36). □

Remark 2.2.8. *A careful examination of the above proof reveals that the constants hidden in the prefactors of the leading term* $\mathcal{O}\left(\hat{c}\delta\Delta t^{\frac{3}{2}}\right)$ *of* (2.36) *stem from the formula* (2.44)*. Here, we have used the assumption that* $\mathbf{y}_k^i = \mathcal{O}(\sqrt{t_k})$*. Note that this assumption implicitly assumes that the energy bound* (2.10) *is* equidistributed *among all the elements of the vector* $\mathbf{y}_k$ *and results in the obfuscation of the constants in the leading term of* (2.36)*. Given that the energy bound* (2.10) *is too coarse to allow for precise upper and lower bounds on each individual element of the hidden state vector* $\mathbf{y}_k$*, we do not see any other way of, in general, determining the distribution of energy among individual entries of the hidden state vector. Thus, assuming equidistribution seems reasonable. On the other hand, in practice, one has access to all the terms in formula* (2.44) *for each numerical experiment and if one is interested, then one can directly evaluate the precise bound on the leading term of the formula* (2.36)*.*

This precise bound (2.36) on the gradient shows that although the gradient can be small, i.e $\mathcal{O}(\Delta t^{\frac{3}{2}})$, it is in fact *independent of* k, ensuring that long-term dependencies contribute to gradients at much later steps and mitigating the vanishing gradient problem.

Summarizing, we see that the RNN (2.3) indeed satisfied similar bounds to the underlying ODE (2.2) that resulted in upper bounds on the hidden states and its gradients. However, the lower bound on the gradient (2.36) is due to the specific choice of this discretization and does not appear to have a continuous analogue, making the specific choice of discretization of (2.2) crucial for mitigating the vanishing gradient problem.

2.2.3 Discretized system with explicit damping

In this section, we will provide rigorous estimates, similar to that of propositions 2.2.4, 2.2.5 and 2.2.6 for the version of coRNN (2.3) that results by setting $\bar{n} = n-1$ in (2.3) leading to,

$$\begin{aligned} \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t \mathbf{z}_n, \\ \mathbf{z}_n &= \mathbf{z}_{n-1} + \Delta t \sigma\left(\mathbf{W}\mathbf{y}_{n-1} + \mathcal{W}\mathbf{z}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}\right) - \Delta t \gamma \mathbf{y}_{n-1} - \Delta t \epsilon \mathbf{z}_{n-1}. \end{aligned} \tag{2.45}$$

Note that (2.45) can be equivalently written as,

$$\begin{aligned}\mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t \mathbf{z}_n,\\ \mathbf{z}_n &= (1-\epsilon\Delta t)\,\mathbf{z}_{n-1} + \Delta t\sigma\left(\mathbf{W}\mathbf{y}_{n-1} + \mathcal{W}\mathbf{z}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}\right) - \Delta t\gamma\mathbf{y}_{n-1}.\end{aligned} \tag{2.46}$$

We will also consider the case of non-unit values of the control parameters γ and ϵ below.

We start with the following bound on the hidden states of (2.45),

Proposition 2.2.9. *Let the damping parameter $\epsilon > \frac{1}{2}$ and the time step Δt in the RNN (2.45) satisfy the following condition,*

$$\Delta t < \frac{2\epsilon - 1}{\gamma + \epsilon^2}. \tag{2.47}$$

Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states of the RNN (2.45) for $1 \le n \le N$, then the hidden states satisfy the following (energy) bounds:

$$\mathbf{y}_n^\top \mathbf{y}_n + \frac{1}{\gamma}\mathbf{z}_n^\top \mathbf{z}_n \le \frac{mt_n}{\gamma}. \tag{2.48}$$

Proof. We set $\mathbf{A}_{n-1} = \mathbf{W}\mathbf{y}_{n-1} + \mathcal{W}\mathbf{z}_{n-1} + \mathbf{V}\mathbf{u}_{n-1} + \mathbf{b}$ and as in the proof of proposition 2.2.4, we multiply $(\mathbf{y}_{n-1}^\top, \frac{1}{\gamma}\mathbf{z}_n^\top)$ to (2.45) and use elementary identities and rearrange terms to obtain,

$$\begin{aligned}\frac{\mathbf{y}_n^\top \mathbf{y}_n}{2} + \frac{\mathbf{z}_n^\top \mathbf{z}_n}{2\gamma} = \frac{\mathbf{y}_{n-1}^\top \mathbf{y}_{n-1}}{2} + \frac{\mathbf{z}_{n-1}^\top \mathbf{z}_{n-1}}{2\gamma} + \frac{(\mathbf{y}_n - \mathbf{y}_{n-1})^\top(\mathbf{y}_n - \mathbf{y}_{n-1})}{2}\\ - \frac{(\mathbf{z}_n - \mathbf{z}_{n-1})^\top(\mathbf{z}_n - \mathbf{z}_{n-1})}{2\gamma}\\ + \frac{\Delta t}{\gamma}\mathbf{z}_n^\top \sigma(\mathbf{A}_{n-1}) - \frac{\epsilon\Delta t}{\gamma}\mathbf{z}_n^\top \mathbf{z}_n + \frac{\epsilon\Delta t}{\gamma}\mathbf{z}_n^\top(\mathbf{z}_n - \mathbf{z}_{n-1}).\end{aligned}$$

We use a *rescaled version* of the well-known Cauchy's inequality

$$ab \le \frac{ca^2}{2} + \frac{b^2}{2c},$$

for a constant $c > 0$ to be determined, to rewrite the above identity as,

$$\begin{aligned}\frac{\mathbf{y}_n^\top \mathbf{y}_n}{2} + \frac{\mathbf{z}_n^\top \mathbf{z}_n}{2\gamma} \le \frac{\mathbf{y}_{n-1}^\top \mathbf{y}_{n-1}}{2} + \frac{\mathbf{z}_{n-1}^\top \mathbf{z}_{n-1}}{2\gamma} + \frac{(\mathbf{y}_n - \mathbf{y}_{n-1})^\top(\mathbf{y}_n - \mathbf{y}_{n-1})}{2}\\ + \left(\frac{\epsilon\Delta t}{2c\gamma} - \frac{1}{2\gamma}\right)(\mathbf{z}_n - \mathbf{z}_{n-1})^\top(\mathbf{z}_n - \mathbf{z}_{n-1}) + \frac{\Delta t}{2\gamma}\sigma(\mathbf{A}_{n-1})^\top\sigma(\mathbf{A}_{n-1})\\ + \left(\frac{\Delta t}{2\gamma} + \frac{c\epsilon\Delta t}{2\gamma} - \frac{\epsilon\Delta t}{\gamma}\right)\mathbf{z}_n^\top \mathbf{z}_n.\end{aligned}$$

Using the first equation in (2.45), the above inequality reduces to,

$$\begin{aligned}\frac{\mathbf{y}_n^\top \mathbf{y}_n}{2} + \frac{\mathbf{z}_n^\top \mathbf{z}_n}{2\gamma} \le \frac{\mathbf{y}_{n-1}^\top \mathbf{y}_{n-1}}{2} + \frac{\mathbf{z}_{n-1}^\top \mathbf{z}_{n-1}}{2\gamma}\\ + \left(\frac{\epsilon\Delta t}{2c\gamma} - \frac{1}{2\gamma}\right)(\mathbf{z}_n - \mathbf{z}_{n-1})^\top(\mathbf{z}_n - \mathbf{z}_{n-1}) + \frac{\Delta t}{2\gamma}\sigma(\mathbf{A}_{n-1})^\top\sigma(\mathbf{A}_{n-1})\\ + \left(\frac{\Delta t^2}{2} + \frac{\Delta t}{2\gamma} + \frac{c\epsilon\Delta t}{2\gamma} - \frac{\epsilon\Delta t}{\gamma}\right)\mathbf{z}_n^\top \mathbf{z}_n.\end{aligned}$$

As long as,

$$\Delta t \leq \min\left(\frac{c}{\epsilon}, \frac{(2-c)\epsilon - 1}{\gamma}\right), \tag{2.49}$$

we can easily check that,

$$\begin{aligned}\frac{\mathbf{y}_n^\top \mathbf{y}_n}{2} + \frac{\mathbf{z}_n^\top \mathbf{z}_n}{2\gamma} &\leq \frac{\mathbf{y}_{n-1}^\top \mathbf{y}_{n-1}}{2} + \frac{\mathbf{z}_{n-1}^\top \mathbf{z}_{n-1}}{2\gamma} + \frac{\Delta t}{2\gamma}\sigma(\mathbf{A}_{n-1})^\top \sigma(\mathbf{A}_{n-1}) \\ &\leq \frac{\mathbf{y}_{n-1}^\top \mathbf{y}_{n-1}}{2} + \frac{\mathbf{z}_{n-1}^\top \mathbf{z}_{n-1}}{2\gamma} + \frac{m\Delta t}{2\gamma} \quad (\sigma \leq 1).\end{aligned}$$

Iterating the above bound till $n = 0$ and using the zero initial data yields the desired (2.48) as long as we find a c such that the condition (2.49) is satisfied. To do so, we equalize the two terms on the right hand side of (2.49) to obtain,

$$c = \frac{\epsilon(2\epsilon - 1)}{\gamma + \epsilon^2}.$$

From the assumption (2.47) and the fact that $\epsilon > \frac{1}{2}$, we see that such a $c > 0$ always exists for any value of $\gamma > 0$ and (2.49) is satisfied, which completes the proof. □

We remark that the same bound on the hidden states is obtained for both versions of coRNN, i.e. (2.3) with $\bar{n} = n$ and (2.45). However, the difference lies in the constraint on the time step Δt. In contrast to (2.47), a careful examination of the proof of proposition 2.2.4 reveals that the condition on the time step for the stability of (2.3) with $\bar{n} = n$ is given by,

$$\Delta t < \frac{2\epsilon - 1}{\gamma}, \tag{2.50}$$

and is clearly less stringent than the condition (2.49) for the stability of (2.45). For instance, in the prototypical case of $\gamma = \epsilon = 1$, the stability of (2.3) with $\bar{n} = n$ is ensured for any $\Delta t < 1$. On the other hand, the stability of (2.45) is ensured as long as $\Delta t < \frac{1}{2}$. However, it is essential to recall that these conditions are only sufficient to ensure stability and are by no means necessary. Thus in practice, the coRNN version (2.45) is found to be stable in the same range of time steps as the version (2.3) with $\bar{n} = n$.

On the exploding and vanishing gradients problem for coRNN with explicit damping (2.45)**.** Next, we have the following upper bound on the hidden state gradients for the version (2.45) of coRNN,

Proposition 2.2.10. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states generated by the RNN* (2.45)*. We assume that the damping parameter $\epsilon > \frac{1}{2}$ and the time step Δt can be chosen such that in addition to* (2.49) *it also satisfies,*

$$\max\left\{\Delta t(\gamma + \|\mathbf{W}\|_\infty), \Delta t\|\mathcal{W}\|_\infty\right\} = \eta \leq \tilde{C}\Delta t^r, \quad \frac{1}{2} \leq r \leq 1, \tag{2.51}$$

and with the constant $\tilde{C}$ independent of the other parameters of the RNN (2.45)*. Then the gradient of the loss function $\mathcal{E}$* (2.14) *with respect to any parameter $\theta \in \mathbf{\Theta}$ is bounded as,*

$$\left|\frac{\partial \mathcal{E}}{\partial \theta}\right| \leq \frac{3(\tilde{C})\left(m + \bar{Y}\sqrt{m}\right)}{2\gamma}, \tag{2.52}$$

with the constant $\tilde{C}$, defined in (2.51) *and $\bar{Y} = \max\limits_{1\leq n\leq N} \|\bar{\mathbf{y}}_n\|_\infty$ be a bound on the underlying training data*

The proof of this proposition is completely analogous to the proof of proposition 2.2.6 and we omit the details here.

Note that the bound (2.52) enforces that hidden state gradients cannot explode for version (2.45) of coRNN. A similar statement for the vanishing gradient problem is inferred from the proposition below.

Proposition 2.2.11. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN* (2.45)*. Under the assumption that $\mathbf{y}_n^i = \mathcal{O}(\sqrt{\frac{t_n}{\gamma}})$, for all $1 \leq i \leq m$ and* (2.51)*, the gradient for long-term dependencies satisfies,*

$$\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \mathcal{O}\left(\frac{\hat{c}}{\gamma}\Delta t^{\frac{3}{2}}\right) + \mathcal{O}\left(\frac{\hat{c}}{\gamma}\delta(1+\delta)\Delta t^{\frac{5}{2}}\right) + \mathcal{O}(\Delta t^3), \quad \hat{c} = sech^2\left(\sqrt{k\Delta t}(1+\Delta t)\right) \quad k << n. \tag{2.53}$$

The proof is a repetition of the steps of the proof of proposition 2.2.7, with suitable modifications for the structure of the RNN and non-unit ϵ, γ and we omit the tedious calculations here. Note that (2.53) rules out the vanishing gradient problem for the coRNN version with explicit damping (2.45).

2.3 Empirical results

We present results on a variety of learning tasks with coRNN (2.3) with $\bar{n} = n - 1$, as this version resulted in marginally better performance than the version with $\bar{n} = n$. Details of the training procedure for each experiment can be found at the end of this section. We wish to clarify here that we use a straightforward hyperparameter tuning protocol based on a validation set and do not use additional performance enhancing tools, such as dropout [Srivastava et al., 2014], gradient clipping [Pascanu et al., 2013] or batch normalization [Ioffe and Szegedy, 2015], which might further improve the performance of coRNNs.

Adding problem. We start with the well-known adding problem [Hochreiter and Schmidhuber, 1997], proposed to test the ability of an RNN to learn (very) long-term dependencies. The input is a two-dimensional sequence of length T, with the first dimension consisting of random numbers drawn from $\mathcal{U}([0, 1])$ and with two non-zero entries (both set to 1) in the second dimension, chosen at random locations, but one each in both halves of the sequence. The output is the sum of two numbers of the first dimension at positions, corresponding to the two 1 entries in the second dimension. We compare the proposed coRNN to three recently proposed RNNs, which were explicitly designed to learn long-term dependencies, namely the FastRNN [Kusupati et al., 2018], the antisymmetric (anti.sym.) RNN [Chang et al., 2019] and the expRNN [Lezcano-Casado and Martınez-Rubio, 2019], and to a plain vanilla tanh RNN, with the goal of beating the baseline mean square error (MSE) of 0.167 (which stems from the variance of the baseline output 1). All methods have 128 hidden units (dimensionality of the hidden state $\mathbf{y}$) and the same training protocol is used in all cases. Fig. 2.1 shows the results for different lengths T of the input sequences. We can see that while the tanh RNN is not able to beat the baseline for any sequence length, the other methods successfully learn the adding task for $T = 500$. However, in this case, coRNN converges significantly faster and reaches a lower test MSE than other tested methods. When setting the length to the much more challenging case of $T = 2000$, we see that only coRNN and the expRNN beat the baseline. However, the expRNN fails to reach a desired test MSE of 0.01 within training time. In order to further demonstrate the superiority of coRNN over recently proposed RNN architectures for learning long-term dependencies, we consider the adding problem for $T = 5000$ and observe that coRNN converges very quickly even in this case, while expRNN fails to consistently beat the baseline. We thus conclude that the coRNN mitigates the vanishing/exploding gradient problem even for very long sequences.

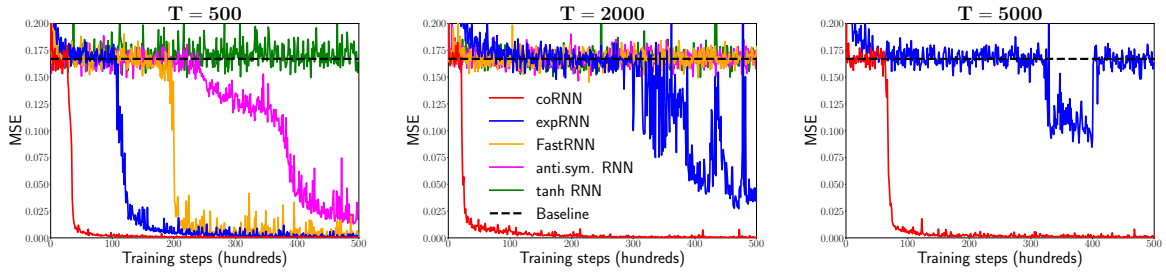

Figure 2.1: Results of the adding problem for coRNN, expRNN, FastRNN, anti.sym. RNN and tanh RNN based on three different sequence lengths T, i.e. $T = 500$, $T = 2000$ and $T = 5000$.

Sequential (permuted) MNIST. Sequential MNIST (sMNIST) [Le et al., 2015] is a benchmark for RNNs, in which the model is required to classify an MNIST [LeCun et al., 1998] digit one pixel at a time leading to a classification task with a sequence length of $T = 784$. In permuted sequential MNIST (psMNIST), a fixed random permutation is applied in order to increase the time-delay between interdependent pixels and to make the problem harder. In Table 2.1, we compare the test accuracy for coRNN on sMNIST and psMNIST with recently published best case results for other recurrent models, which were explicitly designed to solve long-term dependencies together with baselines corresponding to gated and unitary RNNs. To the best of our knowledge the proposed coRNN outperforms all single-layer recurrent architectures, published in the literature, for both the sMNIST and psMNIST. Moreover in Fig. 2.3, we present the performance (with respect to number of epochs) of different RNN architectures for psMNIST with the same fixed random permutation and the same number of hidden units, i.e. 128. As seen from this figure, coRNN clearly outperforms the other architectures, some of which were explicitly designed to learn long-term dependencies, handily for this permutation.

Table 2.1: Results on sMNIST and psMNIST for coRNN as well as uRNN [Arjovsky et al., 2016], LSTM (result taken from Helfrich et al. [2018]), GRU (result taken from Chang et al. [2017]), antisymmetric RNN, DTRIV∞ [Casado, 2019], and FastGRNN. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	sMNIST	psMNIST	# units	# parameters
uRNN	95.1%	91.4%	512	9k
LSTM	98.9%	92.9%	256	270k
GRU	**99.1%**	94.1%	256	200k
antisymmetric RNN	98.0%	95.8%	128	10k
DTRIV∞	99.0%	**96.8%**	512	137k
FastGRNN	98.7%	94.8%	128	18k
coRNN (small)	**99.3%**	**96.6%**	128	34k
coRNN (big)	**99.4%**	**97.3%**	256	134k

Noise padded CIFAR-10. Another challenging test problem for learning long-term dependencies is the recently proposed noise padded CIFAR-10 experiment by Chang et al. [2019], in which CIFAR-10 data points [Krizhevsky et al., 2009] are fed to the RNN row-wise and flattened along the channels resulting in sequences of length 32. To test the long term memory, entries of uniform random numbers are added

such that the resulting sequences have a length of 1000, i.e. the last 968 entries of each sequence are only noise to distract the network. Table 2.2 shows the result for coRNN together with other recently published best case results. We observe that coRNN readily outperforms other RNN architectures on this benchmark, while requiring only 128 hidden units.

Table 2.2: Results on noise padded CIFAR-10 for coRNN as well as LSTM, Incremental RNN, FastRNN, antisymmetric RNN, gated antisymmetric RNN, and LipschitzRNN Erichson et al. [2020], where all other results are taken from Kag et al. [2020], Chang et al. [2019], Erichson et al. [2020]. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
LSTM	11.6%	128	64k
Incremental RNN	54.5%	128	12k
FastRNN	45.8%	128	16k
antisymmetric RNN	48.3%	256	36k
gated antisymmetric RNN	**54.7%**	256	37k
Lipschitz RNN	**55.2%**	256	134k
coRNN	**59.0%**	128	46k

Human activity recognition. This experiment is based on the human activity recognition data set provided by Anguita et al. [2012]. The data set is a collection of tracked human activities, which were measured by an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. Six activities were binarized to obtain two merged classes {Sitting, Laying, Walking_Upstairs} and {Standing, Walking, Walking_Downstairs}, leading to the HAR-2 data set, which was first proposed in Kusupati et al. [2018]. Table 2.3 shows the result for coRNN together with other very recently published best case results on the same data set. We can see that coRNN readily outperforms all other methods. We also ran this experiment on a *tiny coRNN* with very few parameters, i.e. only 1k. We can see that even in this case, the tiny coRNN beats all baselines. We thus conclude that coRNN can efficiently be used on resource-constrained IoT micro-controllers.

Table 2.3: Results on HAR-2 for coRNN as well as GRU, LSTM, FastRNN, FastGRNN, antisymmetric RNN, and incremental RNN, where all other results are taken from Kag et al. [2020], Kusupati et al. [2018]. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
GRU	93.6%	75	19k
LSTM	93.7%	64	16k
FastRNN	94.5%	80	7k
FastGRNN	95.6%	80	7k
antisymmetric RNN	93.2%	120	8k
incremental RNN	**96.3%**	64	4k
coRNN	**97.2%**	64	9k
tiny coRNN	**96.5%**	20	1k

IMDB sentiment analysis. The IMDB data set [Maas et al., 2011] is a collection of 50k movie reviews, where 25k reviews are used for training (with 7.5k of these reviews used for validating) and 25k reviews are used for testing. The aim of this binary sentiment classification task is to decide whether a movie review is positive or negative. We follow the standard procedure by initializing the word embedding with pretrained 100d GloVe [Pennington et al., 2014] vectors and restrict the dictionary to 25k words. Table 2.4 shows the results for coRNN and other recently published models, which are trained similarly and have the same number of hidden units, i.e. 128. We can see that coRNN compares favorable with gated baselines (which are known to perform very well on this task), while at the same time requiring significantly less parameters.

Table 2.4: Results on IMDB for coRNN as well as LSTM, Skip LSTM, GRU, Skip GRU, and ReLU GRU, where all other results are taken from Campos et al. [2018], Dey and Salemt [2017]. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
LSTM	**86.8%**	128	220k
Skip LSTM	**86.6%**	128	220k
GRU	86.2%	128	164k
Skip GRU	**86.6%**	128	164k
ReLU GRU	84.8%	128	99k
coRNN	**87.4%**	128	46k

Chaotic time-series prediction. According to proposition 2.2.5, coRNN does not exhibit chaotic behavior by design. While this property is highly desirable for learning long-term dependencies (a slight perturbation of the input should not result in an unbounded perturbation of the prediction), it impairs the performance on tasks, where the network has to learn actual chaotic dynamics. To test this numerically, we consider the following version of the Lorenz 96 system: [Lorenz, 1996]:

$$x_j' = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, \tag{2.54}$$

where $x_j \in \mathbb{R}$ for all $j = 1, \dots, 5$ and F is an external force controlling the level of chaos in the system. Fig. 2.2 shows a trajectory of the system (2.54) plotted on the x_1x_2-plane for a small external force of $F = 0.9$ as well as a trajectory for a large external force of $F = 8$. We can see that while for $F = 0.9$ the system does not exhibit chaotic behavior, the dynamics for $F = 8$ is already highly chaotic.

Our task consists of predicting the 25-th next state of a trajectory of the system (2.54). We provide 128 trajectories of length 2000 for each of the training, validation and test sets. The trajectories are generated by numerically solving the system (2.54) and evaluating it at 2000 equidistantly distributed discrete time points with distance 0.01. The initial value for each trajectory is chosen uniform at random on $[F - 1/2, F + 1/2]^5$ around the equilibrium point $(F, \dots, F)$ of the system (2.54).

Since LSTMs are known to be able to produce chaotic dynamics, even in the autonomous (zero-entry) case [Laurent and von Brecht, 2017], we expect them to perform significantly better than coRNN if the underlying system exhibits strong chaotic behavior. Table 2.5 shows the normalized root mean square error (NRMSE) (RMSE divided by the root mean square of the target trajectory) on the test set for coRNN and LSTM. We can see that indeed for the non-chaotic case of using an external force of $F = 0.9$ LSTM and coRNN perform similarly. However, when the dynamics get chaotic (in this case using an external force of $F = 8$), the LSTM clearly outperforms coRNN.

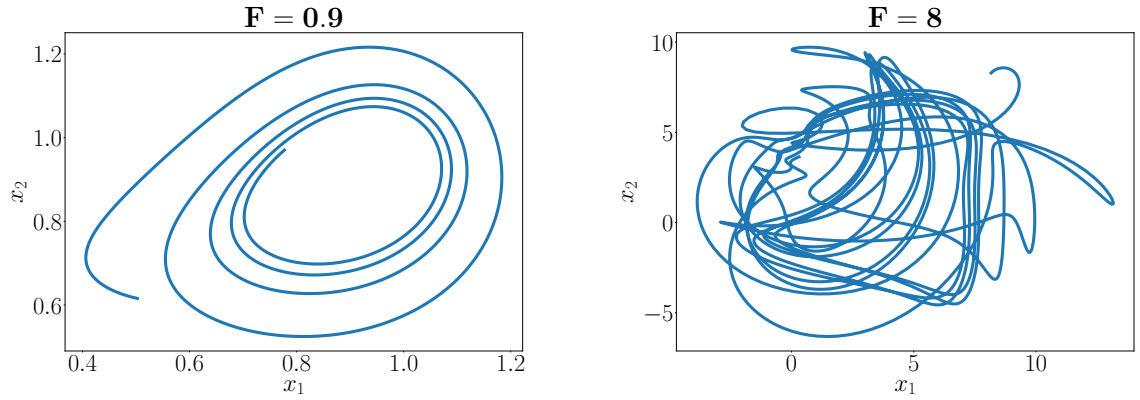

Figure 2.2: Exemplary (x_1, x_2)-trajectories of the Lorenz 96 system (2.54) for different forces F.

Table 2.5: Test NRMSE on the Lorenz 96 system (2.54) for coRNN and LSTM.

Model	$F = 0.9$	$F = 8$	# units	# parameters
LSTM	2.0×10^{-2}	6.8×10^{-2}	44	9k
coRNN	2.0×10^{-2}	9.8×10^{-2}	64	9k

Further experimental results. To shed further light on the performance of coRNN, we consider the following issues. First, the theory suggested that coRNN mitigates the exploding and vanishing gradients problem as long as the assumptions (2.16) on the time step Δt and weight matrices $\mathbf{W}, \mathcal{W}$ hold. Clearly one can choose a suitable Δt to enforce (2.16) before training, but do these assumptions remain valid during training? In 2.2.2, we argue, based on worst-case estimates, that the assumptions will remain valid for possibly a large number of training steps. More pertinently, we can verify experimentally that (2.16) holds during training. This is demonstrated in Fig. 2.4, where we show that (2.16) holds for all long-term dependency tasks during training. Thus, the presented theory applies and one can expect control over hidden state gradients with coRNN. Next, we recall that the frequency parameter γ and damping parameter ϵ play a role for coRNNs (see 2.2.3 for the theoretical dependence and Table 2.8 for best performing values of ϵ, γ for each numerical experiment within the range considered in Table 2.7). How sensitive is the performance of coRNN to the choice of these 2 parameters? To investigate this dependence, we focus on the noise padded CIFAR-10 experiment and show the results of a *sensitivity study* in Fig. 2.5, where the test accuracy for different coRNNs based on a two dimensional hyperparameter grid $(\epsilon, \gamma) \in [0.8, 1.8] \times [5.7, 17, 7]$ (i.e., sufficiently large intervals around the best performing values of ϵ, γ from Table 2.8) is plotted. We observe from the figure that although there are reductions in test accuracy for non-optimal values of (ϵ, γ), there is no large variation and the performance is rather robust with respect to these hyperparameters. Finally, note that we follow standard practice and present best reported results with coRNN as well as other competing RNNs in order to compare the relative performance. However, it is natural to investigate the dependence of these *best* results on the random initial (before training) values of the weight matrices. To this end, in Table 2.6 we report the mean and standard deviation (over 10 retrainings) of the test accuracy with coRNN on various learning tasks and find that the mean value is comparable to the best reported value, with low standard deviations. This indicates further robustness of the performance of coRNNs.

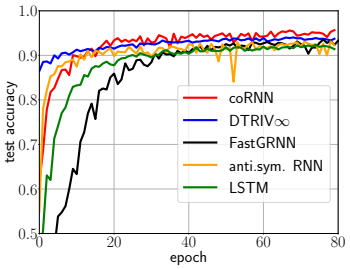

Figure 2.3: Performance on psMNIST for different models, all with 128 hidden units and the same fixed random permutation.

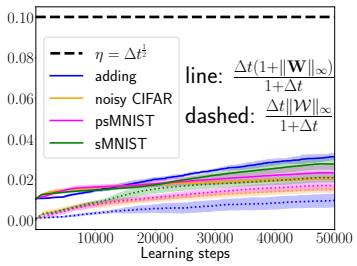

Figure 2.4: Weight assumptions (2.16), with $r = \frac{1}{2}$, evaluated during training for all long-term dependency experiments (mean and standard deviation of 10 different runs for each task).

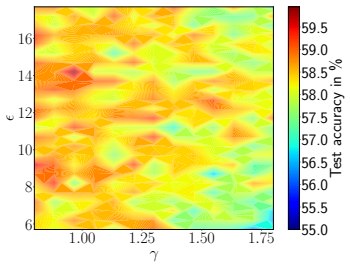

Figure 2.5: Ablation study on the hyperparameters ϵ, γ in (2.3) using the noise padded CIFAR-10 experiment.

Table 2.6: Distributional information (mean and standard deviation) on the results for each classification experiment presented in this chapter based on 10 retrainings of the best performing coRNN using random initialization of the trainable parameters.

Experiment	Mean	Standard deviation
sMNIST (256 units)	99.17%	0.07%
psMNIST (256 units)	96.10%	1.20%
Noise padded CIFAR-10	58.56%	0.35%
HAR-2 (64 units)	96.01%	0.53%
IMDB	86.65%	0.31%

Training details. The IMDB task was conducted on an NVIDIA GeForce GTX 1080 Ti GPU, while all other experiments were run on a Intel Xeon E3-1585Lv5 CPU. The weights and biases of coRNN are randomly initialized according to $\mathcal{U}(-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}})$, where n_{in} denotes the input dimension of each affine transformation. Instead of treating the parameters $\Delta t, \gamma$ and ϵ as fixed hyperparameters, we can also treat them as trainable network parameters by constraining Δt to $[0, 1]$ by using a sigmoidal activation function and $\epsilon, \gamma > 0$ by the use of ReLU for instance. However, in this case no major difference in performance is obtained. The hyperparameters are optimized with a random search algorithm, where the results of the best performing coRNN (based on the validation set) are reported. The ranges of the hyperparameters for the random search algorithm are provided in Table 2.7. Table 2.8 shows the rounded hyperparameters of the best performing coRNN architecture resulting from the random search algorithm for each learning task. We used 100 training epochs for sMNIST, psMNIST and noise padded CIFAR-10 with additional 20 epochs in which the learning rate was reduced by a factor of 10. Additionally, we used 100 epochs for the IMDB task and 250 epochs for the HAR-2 task.

2.4 Discussion

Inspired by many models in physics, biology and engineering, we proposed a novel RNN architecture (2.3) based on a model (2.1) of a *network of controlled forced and damped oscillators*. For this RNN, we

Table 2.7: Setting for the hyperparameter optimization of coRNN. Intervals denote ranges of the corresponding hyperparameter for the grid search algorithm, while fixed numbers mean that no hyperparameter optimization was done in this case.

task	learning rate	batch size	Δt	γ	ϵ
Adding	2×10^{-2}	50	$[10^{-2}, 10^{-1}]$	$[1, 100]$	$[1, 100]$
sMNIST ($n_{hid} = 128$)	$[10^{-4}, 10^{-1}]$	120	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$
sMNIST ($n_{hid} = 256$)	$[10^{-4}, 10^{-1}]$	120	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$
psMNIST ($n_{hid} = 128$)	$[10^{-4}, 10^{-1}]$	120	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$
psMNIST ($n_{hid} = 256$)	$[10^{-4}, 10^{-1}]$	120	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$
Noise padded CIFAR-10	$[10^{-4}, 10^{-1}]$	100	$[10^{-2}, 10^{-1}]$	$[1, 100]$	$[1, 100]$
HAR-2	$[10^{-4}, 10^{-1}]$	64	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$
IMDB	$[10^{-4}, 10^{-1}]$	64	$[10^{-2}, 10^{-1}]$	$[10^{-1}, 10]$	$[10^{-1}, 10]$

Table 2.8: Rounded hyperparameters of the best performing coRNN architecture.

task	learning rate	batch size	Δt	γ	ϵ
Adding ($T = 5000$)	2×10^{-2}	50	1.6×10^{-2}	94.5	9.5
sMNIST ($n_{hid} = 128$)	3.5×10^{-3}	120	5.3×10^{-2}	1.7	4
sMNIST ($n_{hid} = 256$)	2.1×10^{-3}	120	4.2×10^{-2}	2.7	4.7
psMNIST ($n_{hid} = 128$)	3.7×10^{-3}	120	8.3×10^{-2}	1.3×10^{-1}	4.1
psMNIST ($n_{hid} = 256$)	5.4×10^{-3}	120	7.6×10^{-2}	4×10^{-1}	8.0
Noise padded CIFAR-10	7.5×10^{-3}	100	3.4×10^{-2}	1.3	12.7
HAR-2	1.7×10^{-2}	64	10^{-1}	2×10^{-1}	6.4
IMDB	6.0×10^{-4}	64	5.4×10^{-2}	4.9	4.8

rigorously showed that under verifiable hypotheses on the time step and weight matrices, the hidden states are bounded (2.10) and obtained precise bounds on the gradients (Jacobians) of the hidden states, (2.17) and (2.36). Thus by design, this architecture can mitigate the exploding and vanishing gradients problem for RNNs. We present a series of numerical experiments that include sequential image classification, activity recognition and sentiment analysis, to demonstrate that the proposed coRNN keeps hidden states and their gradients under control, while retaining sufficient expressivity to perform complex tasks. Thus, we provide a novel and promising strategy for designing RNN architectures that are motivated by the functioning of natural systems, have rigorous bounds on hidden state gradients and are robust, accurate, straightforward to train and cheap to evaluate.

This work can be extended in different directions. For instance in this chapter, we have mainly focused on the learning of tasks with long-term dependencies and observed that coRNNs are comparable in performance to the best published results in the literature. Given that coRNNs are built with networks of oscillators, it is natural to expect that they will perform very well on tasks with oscillatory inputs/outputs, such as the time series analysis of high-resolution biomedical data, for instance EEG (electroencephalography) and EMG (electromyography) data and seismic activity data from geoscience. Similarly, applications of coRNN to language modeling will be covered in future work.

However, it is essential to point out that coRNNs might not be suitable for every learning task involving sequential inputs/outputs. As a concrete example, we consider the problem of predicting time series corresponding to a chaotic dynamical system. We recall that by construction, the underlying ODE

(2.2) (and the discretization (2.3)) do not allow for super-linear (in time) separation of trajectories for nearby inputs. Thus, we cannot expect that coRNNs will be effective at predicting chaotic time series and it is indeed investigated and demonstrated for a Lorenz-96 ODE in the experimental results section 2.3, where we observe that the coRNN is outperformed by LSTMs in the chaotic regime.

Our main theoretical focus in this chapter was to demonstrate the possible mitigation of the exploding and vanishing gradients problem. On the other hand, we only provided some heuristics and numerical evidence on why the proposed RNN still has sufficient expressivity. A priori, it is natural to think that the proposed RNN architecture might introduce a strong bias towards oscillatory functions. However, the proposed coRNN can be significantly more expressive, as the damping, forcing and coupling of several oscillators modulates nonlinear response to yield a very rich and diverse set of output states. This is also evidenced by the ability of coRNNs to deal with many tasks in our numerical experiments, which do not have an explicit oscillatory structure. This sets the stage for a rigorous investigation of *universality* of the proposed coRNN architecture, as in the case of echo state networks in Grigoryeva and Ortega [2018]. A possible approach would be to leverage the ability of the proposed RNN to convert general inputs into a rich set of superpositions of harmonics (oscillatory wave forms). Moreover, the proposed RNN was based on the simplest model of coupled oscillators (2.1). Much more detailed models of oscillators are available, particularly those that arise in the modeling of biological neurons, Stiefel and Ermentrout [2016] and references therein. An interesting variant of our proposed RNN would be to base the RNN architecture on these more elaborate models, resulting in analogues of the spiking neurons model of Maass [2001] for RNNs.

Chapter 3

Undamped Independent Controlled Oscillatory Recurrent Neural Network

Hamiltonian systems Arnold [1989] are a large class of dynamical systems in physics and engineering that allow for very precise control on the underlying states. Moreover, the fact that the phase space volume is preserved by the trajectories of a Hamiltonian system, makes such systems *invertible* and allows one to significantly reduce the storage requirements. Furthermore, if the resulting hidden state gradients also evolve according to a Hamiltonian dynamical system, one can obtain precise bounds on the hidden state gradients and alleviate the exploding and vanishing gradients problem. Motivated by this, we combine and extend these ideas into an RNN architecture that will allow us to prove rigorous bounds on the hidden states and their gradients, mitigating the exploding and vanishing gradients problem. Moreover, our RNN architecture results in a fast and memory-efficient implementation that attains state-of-the-art performance on a variety of sequential learning tasks with very long time dependencies.

3.1 The proposed RNN

Our proposed RNN is based on the time-discretization of the following system of *second-order ordinary differential equations* (ODEs),

$$\mathbf{y}'' = -[\sigma\left(\mathbf{w} \odot \mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}\right) + \alpha\mathbf{y}]. \tag{3.1}$$

Here, $t \in [0, 1]$ is the (continuous) time variable, $\mathbf{u} = \mathbf{u}(t) \in \mathbb{R}^d$ is the time-dependent *input signal*, $\mathbf{y} = \mathbf{y}(t) \in \mathbb{R}^m$ is the *hidden state* of the RNN with $\mathbf{w} \in \mathbb{R}^m$ is a weight vector, $\mathbf{V} \in \mathbb{R}^{m\times d}$ a weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector and $\alpha \geq 0$ is a control parameter. The operation $\odot$ is the Hadamard product and the function $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is the *activation function* and is applied component wise. For the rest of this chapter, we set $\sigma(u) = \tanh(u)$.

By introducing the auxiliary variable $\mathbf{z} = \mathbf{y}'$, we can rewrite the second order ODE (3.1) as a first order ODE system:

$$\mathbf{y}' = \mathbf{z}, \quad \mathbf{z}' = -[\sigma\left(\mathbf{w} \odot \mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}\right) + \alpha\mathbf{y}]. \tag{3.2}$$

Assuming that $\mathbf{w}_i \neq 0$, for all $1 \leq i \leq m$, it is easy to see that the ODE system (3.2) is a *Hamiltonian system*,

$$\mathbf{y}' = \frac{\partial H}{\partial \mathbf{z}}, \quad \mathbf{z}' = -\frac{\partial H}{\partial \mathbf{y}}, \tag{3.3}$$

with the *time-dependent Hamiltonian*,

$$H(\mathbf{y}, \mathbf{z}, t) = \frac{\alpha}{2}\|\mathbf{y}\|^2 + \frac{1}{2}\|\mathbf{z}\|^2 + \sum_{i=1}^{m} \frac{1}{\mathbf{w}_i} \log(\cosh(\mathbf{w}_i \mathbf{y}_i + (\mathbf{V}\mathbf{u}(t))_i + \mathbf{b}_i)), \tag{3.4}$$

with $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$ denoting the Euclidean norm of the vector $\mathbf{x} \in \mathbb{R}^m$ and $\langle \cdot, \cdot \rangle$ the corresponding inner product.

The next step is to find a discretization of the ODE system (3.2). Given that it is highly desirable to ensure that the discretization respects the Hamiltonian structure of the underlying continuous ODE, the simplest such *structure preserving discretization* is the *symplectic Euler* method Sanz Serna and Calvo [1994], Hairer et al. [2003]. Applying the symplectic Euler method to the ODE (3.2) results in the following discrete dynamical system,

$$\begin{aligned} \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t \mathbf{z}_n, \\ \mathbf{z}_n &= \mathbf{z}_{n-1} - \Delta t[\sigma\left(\mathbf{w} \odot \mathbf{y}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}\right) + \alpha \mathbf{y}_{n-1}], \end{aligned} \tag{3.5}$$

for $1 \leq n \leq N$. Here, $0 < \Delta t < 1$ is the time-step and $\mathbf{u}_n \approx \mathbf{u}(t_n)$, with $t_n = n\Delta t$, is the input signal. It is common to initialize with $\mathbf{y}_0 = \mathbf{z}_0 = \mathbf{0}$.

We see from the structure of the discrete dynamical system (3.5) that there is *no interaction* between the neurons in the hidden layer of (3.5). Such an RNN will have very limited expressivity. Hence, we *stack* more hidden layers to propose the following deep or *multi-layer* RNN,

$$\begin{aligned} \mathbf{y}_n^\ell &= \mathbf{y}_{n-1}^\ell + \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot \mathbf{z}_n^\ell, \\ \mathbf{z}_n^\ell &= \mathbf{z}_{n-1}^\ell - \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot [\sigma(\mathbf{w}^\ell \odot \mathbf{y}_{n-1}^\ell + \mathbf{V}^\ell \mathbf{y}_n^{\ell-1} + \mathbf{b}^l) + \alpha \mathbf{y}_{n-1}^\ell]. \end{aligned} \tag{3.6}$$

Here $\mathbf{y}_n^l, \mathbf{z}_n^l \in \mathbb{R}^m$ are hidden states and $\mathbf{w}^\ell, \mathbf{V}^\ell, \mathbf{b}^\ell$ are weights and biases, corresponding to layer $\ell = 1, \ldots, L$. We set $\mathbf{y}_n^0 = \mathbf{u}_n$ in the multilayer RNN (3.6).

In Fig. 3.1, we present a schematic diagram of the proposed multi-layer recurrent model UnICORNN.

Observe that we use the same step-size Δt for every layer, while multiplying a trainable parameter vector $\mathbf{c} \in \mathbb{R}^m$ to the time step. The action of $\mathbf{c}$ is modulated with the sigmoidal activation function $\hat{\sigma}(u) = 0.5 + 0.5 \tanh(u/2)$, which ensures that the time-step Δt is multiplied by a value between 0 and 1. We remark that the presence of this trainable vector $\mathbf{c}$ allows us to incorporate *multiscale behavior* in the proposed RNN, as the effective time-step is learned during training and can be significantly different from the nominal time-step Δt. It is essential to point out that including this multiscale time stepping is only possible, as each neuron (within the same hidden layer) is independent of the others and can be integrated with a different effective time step. Finally, we also share the control hyperparameter α across the different layers, which results in a memory unit of L layers with a total of only 2 hyperparameters.

Motivation and background. The ODE system (3.2) is a model for a nonlinear system of uncoupled driven oscillators Guckenheimer and Holmes [1990]. To see this, we denote $\mathbf{y}_i(t)$ as the displacement and $\mathbf{z}_i(t)$ as the velocity. Then, the dynamics of the i-th oscillator is determined by the frequency α and also by the *forcing* or *driving* term in the second equation of (3.2), where the forcing acts through the input signal $\mathbf{u}$ and is modulated by the weight $\mathbf{V}$ and bias $\mathbf{b}$. Finally, the weight $\mathbf{w}$ modulates the frequency α and allows each neuron to oscillate with its own frequency, rather than the common frequency α of the system. The structure of $\mathbf{w}$ implies that each neuron is independent of the others. A key element of the oscillator system (3.2) is the absence of any damping or friction term. This allows the system to possess a Hamiltonian structure, with desirable long time behavior. Thus, we term the resulting RNN (3.6), based on the ODE system (3.2) as **Un**damped **I**ndependent **C**ontrolled **O**scillatory **RNN** or **UnICORNN**.

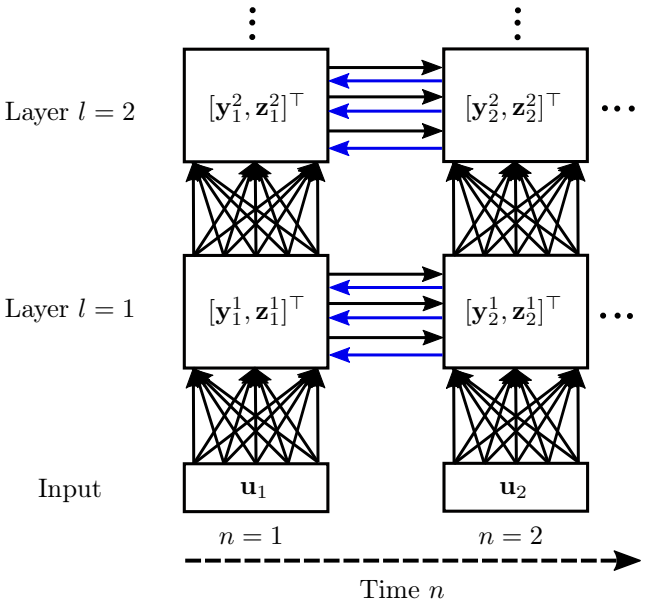

Figure 3.1: Schematic diagram of the multi-layer UnICORNN architecture, where the layers (respectively the input) are densely connected and the hidden states evolve independently in time. The invertibility of UnICORNN is visualized with blue arrows, emphasizing that the hidden states can be reconstructed during the backward pass and do not need to be stored.

We remark that networks of oscillators are very common in science and engineering Guckenheimer and Holmes [1990], Strogatz [2001] with prominent examples being pendulums in mechanics, electrical circuits in engineering, business cycles in economics and functional brain circuits such as cortical columns in neurobiology.

Comparison with related work. UnICORNN lies firmly in the class of ODE-based or ODE-inspired RNNs, which have received considerable amount of attention in the machine learning literature in recent years. Neural ODEs, first proposed in Chen et al. [2018], are a prominent example of using ODEs to construct neural networks. In this architecture, the continuous ODE serves as the learning model and gradients are computed from a sensitivity equation, which allows one to trade accuracy with computing time. Moreover, it is argued that these neural ODEs are invertible and hence, memory efficient. However, it is unclear if a general neural ODE, without any additional structure, can be invertible. Other RNN architectures that are based on discretized ODEs include those proposed in E [2017] and Chang et al. [2019], where the authors proposed an *anti-symmetric* RNN, based on the discretization of a stable ODE resulting from a skew-symmetric hidden weight matrix, thus constraining the gradient dynamics.

Our proposed RNN (3.6) is inspired by two recent RNN architectures. The first one is *coRNN* of Chapter 2, where the underlying RNN architecture was also based on the use of a network of oscillators. As long as a constraint on the underlying weights was satisfied, coRNN was shown to mitigate the exploding and vanishing gradients problem in Section 2.2.2. In contrast to coRNN, UnICORNN does not use a *damping* term. Moreover, each neuron, for any hidden layer, in UnICORNN (3.6) is independent. This is very different from coRNN where all the neurons were coupled together. Finally, UnICORNN is a multi-layer architecture whereas coRNN used a single hidden layer. These innovations allow us to admit

a Hamiltonian structure for UnICORNN and facilitate a fast and memory efficient implementation.

Our proposed architecture was also partly inspired by *IndRNN*, proposed in Li et al. [2018, 2019], where the neurons in each hidden layers were independent of each other and interactions between neurons were mediated by stacking multiple RNN layers, with output of each hidden layer passed on to the next hidden layer, leading to a deep RNN. We clearly use this construction of independent neurons in each layer and stacking multiple layers in UnICORNN (3.6). However in contrast to IndRNN, our proposed RNN is based on a discretized Hamiltonian system and we will not require any constraints on the weights to mitigate the exploding and vanishing gradients problem.

Finally, we would like to point out that discrete Hamiltonian systems have already been used to design RNNs, for instance in Greydanus et al. [2019] and also in Chen et al. [2020c], where a symplectic time-integrator for a Hamiltonian system was proposed as the RNN architecture. However, these approaches are based on underlying time-independent Hamiltonians and are only relevant for mechanical systems as they cannot process time-dependent inputs, which arise in most realistic learning tasks. Moreover, as these methods enforce exact conservation of the Hamiltonian in time, they are not suitable for learning long-time dependencies, see MacKay et al. [2018] for a discussion and experiment on that issue. Although we use a Hamiltonian system as the basis of our proposed RNN (3.6), our underlying Hamiltonian (3.4) is time-dependent and the resulting RNN can readily process any time-dependent input signal.

On the Memory Efficiency of UnICORNN. The standard BPTT training algorithm for RNNs requires one to store all the hidden states at every time step. To see this, we observe that for a standard multi-layer RNN with L layers and a mini-batch size of b (for any mini-batch stochastic gradient descent algorithm), the storage (in terms of floats) scales as $\mathcal{O}(Nbd + LbmN)$, with input and hidden sequences of length N. This memory requirement can be very high. Note that we have ignored the storage of trainable weights and biases for the RNN in the above calculation.

On the other hand, as argued before, our proposed RNN is a symplectic Euler discretization for a Hamiltonian system. Hence, it is invertible. In fact, one can explicitly write the *inverse* of UnICORNN (3.6) as,

$$\begin{aligned} \mathbf{y}_{n-1}^l &= \mathbf{y}_n^l - \Delta t \hat{\sigma}(\mathbf{c}^l) \odot \mathbf{z}_n^l, \\ \mathbf{z}_{n-1}^l &= \mathbf{z}_n^l + \Delta t \hat{\sigma}(\mathbf{c}^l) \odot [\sigma(\mathbf{w}^l \odot \mathbf{y}_{n-1}^l + \mathbf{V}^\ell \mathbf{y}_n^{\ell-1} + \mathbf{b}^l) + \alpha \mathbf{y}_{n-1}^l]. \end{aligned} \tag{3.7}$$

Thus, one can recover all the hidden states in a given hidden layer, only from the *stored* hidden state at the final time step, for that layer. Moreover, only the input signal needs to be stored as the other hidden states can be reconstructed from the formula (3.7). Hence, a straightforward calculation shows that the storage for UnICORNN scales as $\mathcal{O}(Nbd + Lbm)$. As $L << N$, we conclude that UnICORNN allows for a significant saving in terms of storage, when compared to a standard RNN.

3.2 Rigorous analysis of UnICORNN

3.2.1 Continuous-time system

In order to investigate the exploding and vanishing gradients problem for the proposed RNN (3.6), we will first explore the dynamics of the gradients of hidden states $\mathbf{y}, \mathbf{z}$ (solutions of the ODE (3.2)) with respect to the trainable parameters $\mathbf{w}, \mathbf{V}$ and $\mathbf{b}$. Denote any scalar parameter as θ and $f_\theta = \frac{\partial f}{\partial \theta}$, then differentiating the ODE (3.2) with respect to θ results in the ODE,

$$\begin{aligned} \mathbf{y}_\theta' &= \mathbf{z}_\theta, \\ \mathbf{z}_\theta' &= -\sigma'(\mathbf{A}) \odot (\mathbf{w} \odot \mathbf{y}_\theta) - \alpha \mathbf{y}_\theta - \sigma'(\mathbf{A}) \odot \mathbf{C}(t), \end{aligned} \tag{3.8}$$

where $\mathbf{A} = \mathbf{w} \odot \mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}$ is the pre-activation and the coefficient $\mathbf{C} \in \mathbb{R}^m$ is given by $\mathbf{C}_i = \mathbf{y}_i$ if $\theta = \mathbf{w}_i$, $\mathbf{C}_i = \mathbf{u}_j$ if $\theta = \mathbf{V}_{ij}$ and $\mathbf{C}_i = 1$ if $\theta = \mathbf{b}_i$, with all other entries of the vector $\mathbf{C}$ being zero.

It is easy to check that the ODE system (3.8) is a *Hamiltonian system* of form (3.3), with the following time-dependent Hamiltonian;

$$\mathbf{H}\left(\mathbf{y}_\theta, \mathbf{z}_\theta, t\right) := \frac{\alpha}{2}\|\mathbf{y}_\theta\|^2 + \frac{1}{2}\|\mathbf{z}_\theta\|^2 + \frac{1}{2}\sum_{i=1}^{m} \sigma'(\mathbf{A}_i)\mathbf{w}_i((\mathbf{y}_\theta)_i)^2 + \sum_{i=1}^{m} \sigma'(\mathbf{A}_i)\mathbf{C}_i(t)(\mathbf{y}_\theta)_i. \tag{3.9}$$

Thus, by the well-known Liouville's theorem Sanz Serna and Calvo [1994], we know that the phase space volume of (3.8) is preserved. Hence, this system cannot have any asymptotically stable fixed points. This implies that $\{\mathbf{0}, \mathbf{0}\}$ cannot be a stable fixed point for the hidden state gradients $(\mathbf{y}_\theta, \mathbf{z}_\theta)$. Thus, we can expect that the hidden state gradients with respect to the system of oscillators (3.2) do not remain near zero and suggest a possible mechanism for the mitigation of the vanishing gradient problem for UnICORNN (3.6), which is a structure preserving discretization of the ODE (3.2).

3.2.2 Discretized system

We rewrite UnICORNN (3.6) in the following form: for all $1 \leq \ell \leq L$ and for all $1 \leq i \leq m$

$$\begin{aligned}
\mathbf{y}_n^{\ell,i} &= \mathbf{y}_{n-1}^{\ell,i} + \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{z}_n^{\ell,i}, \\
\mathbf{z}_n^{\ell,i} &= \mathbf{z}_{n-1}^{\ell,i} - \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i})\sigma(\mathbf{A}_{n-1}^{\ell,i}) - \alpha \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}, \\
\mathbf{A}_{n-1}^{\ell,i} &= \mathbf{w}^{\ell,i}\mathbf{y}_{n-1}^{\ell,i} + \left(\mathbf{V}^\ell \mathbf{y}_n^{\ell-1}\right)^i + \mathbf{b}^{\ell,i}.
\end{aligned} \tag{3.10}$$

Here, we have denoted the i-th component of a vector $\mathbf{x}$ as $\mathbf{x}^i$.

We follow standard practice and set $\mathbf{y}_0^\ell = \mathbf{z}_0^\ell \equiv 0$, for all $1 \leq \ell \leq L$. Moreover for simplicity of exposition, we set $\alpha > 0$ in the following.

Pointwise bounds on hidden states. We have the following bounds on the discrete hidden states,

Proposition 3.2.1. *Let $\mathbf{y}_n^\ell, \mathbf{z}_n^\ell$ be the hidden states at the n-th time level t_n for UnICORNN (3.10), then under the assumption that the time step $\Delta t << 1$ is sufficiently small, these hidden states are bounded as,*

$$\max_{1\leq i\leq m} |\mathbf{y}_n^{\ell,i}| \leq \sqrt{\frac{2}{\alpha}\left(1 + 2\beta t_n\right)}, \quad \max_{1\leq i\leq m} |\mathbf{z}_n^{\ell,i}| \leq \sqrt{2\left(1 + 2\beta t_n\right)} \quad \forall n, \forall\ 1 \leq \ell \leq L, \tag{3.11}$$

with the constant

$$\beta = \max\{1 + 2\alpha, 4\alpha^2\}.$$

Proof. We fix ℓ, n and multiply the first equation in (3.10) with $\alpha \mathbf{y}_{n-1}^{\ell,i}$ and use the elementary identity

$$b(a-b) = \frac{a^2}{2} - \frac{b^2}{2} - \frac{1}{2}(a-b)^2,$$

to obtain

$$\begin{aligned}
\frac{\alpha(\mathbf{y}_n^{\ell,i})^2}{2} &= \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha}{2}(\mathbf{y}_n^{\ell,i} - \mathbf{y}_{n-1}^{\ell,i})^2 + \alpha \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}, \\
&= \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha \Delta t^2}{2}(\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\mathbf{z}_n^{\ell,i})^2 + \alpha \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}.
\end{aligned} \tag{3.12}$$

Next, we multiply the second equation in (3.10) with $\mathbf{z}_n^{\ell,i}$ and use the elementary identity

$$a(a-b) = \frac{a^2}{2} - \frac{b^2}{2} + \frac{1}{2}(a-b)^2,$$

to obtain

$$\begin{aligned}\frac{(\mathbf{z}_n^{\ell,i})^2}{2} = \frac{(\mathbf{z}_{n-1}^{\ell,i})^2}{2} - \frac{1}{2}(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i})^2 - \Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\sigma(\mathbf{A}_{n-1}^{\ell,i})\left(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i}\right) \\ - \Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\sigma(\mathbf{A}_{n-1}^{\ell,i})\mathbf{z}_{n-1}^{\ell,i} - \alpha\Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}.\end{aligned} \tag{3.13}$$

Adding (3.12) and (3.13) and using Cauchy's inequality yields,

$$\begin{aligned}\frac{\alpha(\mathbf{y}_n^{\ell,i})^2}{2} + \frac{(\mathbf{z}_n^{\ell,i})^2}{2} &\leq \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{(1+\Delta t)(\mathbf{z}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha\Delta t^2}{2}(\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\mathbf{z}_n^{\ell,i})^2 \\ &\quad + (\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\sigma(\mathbf{A}_{n-1}^{\ell,i}))^2\Delta t + \frac{\Delta t - 1}{2}(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i})^2 \\ \Rightarrow \alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2 &\leq \alpha(\mathbf{y}_{n-1}^{\ell,i})^2 + (1+\Delta t)(\mathbf{z}_{n-1}^{\ell,i})^2 + 2\Delta t + \alpha\Delta t^2(\mathbf{z}_n^{\ell,i})^2,\end{aligned}$$

where the last inequality follows from the fact that $|\sigma|, |\hat{\sigma}| \leq 1$ and $\Delta t < 1$. Using the elementary inequality,

$$(a+b+c)^2 \leq 4a^2 + 4b^2 + 2c^2,$$

and substituting for $\mathbf{z}_n^{\ell,i}$ from the second equation of (3.10) in the last inequality leads to,

$$\alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2 \leq (1 + 4\alpha^2\Delta t^4)\alpha(\mathbf{y}_{n-1}^{\ell,i})^2 + (1 + \Delta t + 2\alpha\Delta t^2)(\mathbf{z}_{n-1}^{\ell,i})^2 + 2\Delta t + 4\alpha\Delta t^4.$$

Denoting $H_n = \alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2$ and

$$G := 1 + \beta\Delta t, \quad \beta = \max\{1 + 2\alpha, 4\alpha^2\}$$

yields the following inequality,

$$H_n \leq GH_{n-1} + 2\Delta t(1 + 2\alpha\Delta t^3). \tag{3.14}$$

Iterating the above n-times and using the fact that the initial data is such that $H_0 \equiv 0$ we obtain,

$$\begin{aligned}H_n &\leq \left(2\Delta t + 4\alpha\Delta t^4\right)\sum_{k=0}^{n-1}(1+\beta\Delta t)^k \\ &\leq \frac{(1+\beta\Delta t)^n}{\beta\Delta t}\left(2\Delta t + 4\alpha\Delta t^4\right) \\ &\leq \frac{1}{\beta}\left(1 + 2\beta n\Delta t\right)\left(2 + 4\alpha\Delta t^3\right) \quad \text{as } \Delta t << 1, \\ &\leq 2(1 + 2\beta t_n) \text{ (from definition of } \beta).\end{aligned} \tag{3.15}$$

The definition of H clearly implies the desired bound (3.11). □

On the Exploding Gradient Problem for UnICORNN. We train the RNN (3.10) to minimize the loss function,

$$\mathcal{E} := \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}_n, \quad \mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n^L - \bar{\mathbf{y}}_n\|_2^2, \tag{3.16}$$

with $\bar{\mathbf{y}}$ being the underlying ground truth (training data). Note that the loss function (3.16) only involves the output at the last hidden layer (we set the affine output layer to identity for the sake of simplicity). During training, we compute gradients of the loss function (3.16) with respect to the trainable weights and biases $\mathbf{\Theta} = [\mathbf{w}^\ell, \mathbf{V}^\ell, \mathbf{b}^\ell, \mathbf{c}^\ell]$, for all $1 \leq \ell \leq L$ i.e.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{E}_n}{\partial \theta}, \quad \forall\ \theta \in \mathbf{\Theta}. \tag{3.17}$$

We have the following bound on the gradient (3.17),

Proposition 3.2.2. *Let the time step $\Delta t << 1$ be sufficiently small in the RNN* (3.10) *and let $\mathbf{y}_n^\ell, \mathbf{z}_n^\ell$, for $1 \leq \ell \leq L$, be the hidden states generated by the RNN* (3.10)*. Then, the gradient of the loss function $\mathcal{E}$* (3.16) *with respect to any parameter $\theta \in \mathbf{\Theta}$ is bounded as,*

$$\left| \frac{\partial \mathcal{E}}{\partial \theta} \right| \leq \frac{1 - (\Delta t)^L}{1 - \Delta t} T(1 + 2\gamma T)\overline{\mathbf{V}}(\overline{Y} + \mathbf{F})\mathbf{\Delta}, \tag{3.18}$$

with $\bar{Y} = \max\limits_{1 \leq n \leq N} \|\bar{\mathbf{y}}_n\|_\infty$, be a bound on the underlying training data and other quantities in (3.18) *defined as,*

$$\begin{aligned}
\gamma &= \max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right) + \frac{\left(\max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right)\right)^2}{2}, \\
\overline{\mathbf{V}} &= \prod_{q=1}^{L} \max\{1, \|\mathbf{V}^q\|_\infty\}, \\
\mathbf{F} &= \sqrt{\frac{2}{\alpha}(1 + 2\beta T)}, \\
\mathbf{\Delta} &= \left(2 + \sqrt{(1 + 2\beta T)} + (2 + \alpha)\sqrt{\frac{2}{\alpha}(1 + 2\beta T)}\right).
\end{aligned} \tag{3.19}$$

Proof. For any $1 \leq n \leq N$ and $1 \leq \ell \leq L$, let $\mathbf{X}_n^\ell \in \mathbb{R}^{2m}$ be the augmented hidden state vector defined by,

$$\mathbf{X}_n^\ell = \left[\mathbf{y}_n^{\ell,1}, \mathbf{z}_n^{\ell,1}, \ldots, \mathbf{y}_n^{\ell,i}, \mathbf{z}_n^{\ell,i}, \ldots, \mathbf{y}_n^{\ell,m}, \mathbf{z}_n^{\ell,m}\right]. \tag{3.20}$$

For any $\theta \in \mathbf{\Theta}$, we can apply the chain rule repeatedly to obtain the following extension of the formula of Pascanu et al. [2013] to a deep RNN,

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{\ell=1}^{L} \sum_{k=1}^{n} \underbrace{\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}}_{\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}}. \tag{3.21}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k^\ell$ with respect to the parameter θ, while keeping the other arguments constant.

We remark that the quantity $\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}$ denotes the contribution from the k-recurrent step at the l-th hidden layer of the deep RNN (3.10) to the overall hidden state gradient at the step n.

It is straightforward to calculate that,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} = \left[\mathbf{y}_n^{L,1} - \overline{\mathbf{y}}_n^1, 0, \ldots, \mathbf{y}_n^{L,i} - \overline{\mathbf{y}}_n^i, 0, \ldots, \mathbf{y}_n^{L,m} - \overline{\mathbf{y}}_n^m, 0\right]. \tag{3.22}$$

Repeated application of the chain and product rules yields,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} = \prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \prod_{q=\ell+1}^{L} \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}}. \tag{3.23}$$

For any j, a straightforward calculation using the form of the RNN (3.10) leads to the following representation formula for the matrix $\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \in \mathbb{R}^{2m} \times \mathbb{R}^{2m}$:

$$\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} = \begin{bmatrix} \mathbf{B}_j^{L,1} & 0 & \dots & 0 \\ 0 & \mathbf{B}_j^{L,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{B}_j^{L,m} \end{bmatrix}, \tag{3.24}$$

with the block matrices $\mathbf{B}_j^{L,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{B}_j^{L,i} = \begin{bmatrix} 1 - (\hat{\sigma}(\mathbf{c}^{L,i}))^2 \Delta t^2 \left(\mathbf{w}^{L,i} \sigma'(\mathbf{A}_{j-1}^{L,i}) + \alpha \right) & \hat{\sigma}(\mathbf{c}^{L,i}) \Delta t \\ -\hat{\sigma}(\mathbf{c}^{L,i}) \Delta t \left(\mathbf{w}^{L,i} \sigma'(\mathbf{A}_{j-1}^{L,i}) + \alpha \right) & 1 \end{bmatrix}. \tag{3.25}$$

Similarly for any q, the matrix $\frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \in \mathbb{R}^{2m\times 2m}$ can be readily computed as,

$$\frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} = \begin{bmatrix} \mathbf{D}_{11}^{q,k} & 0 & \mathbf{D}_{12}^{q,k} & 0 & \dots & \dots & \mathbf{D}_{1m}^{q,k} & 0 \\ \mathbf{E}_{11}^{q,k} & 0 & \mathbf{E}_{12}^{q,k} & 0 & \dots & \dots & \mathbf{E}_{1m}^{q,k} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{D}_{m1}^{q,k} & 0 & \mathbf{D}_{m2}^{q,k} & 0 & \dots & \dots & \mathbf{D}_{mm}^{q,k} & 0 \\ \mathbf{E}_{m1}^{q,k} & 0 & \mathbf{E}_{m2}^{q,k} & 0 & \dots & \dots & \mathbf{E}_{mm}^{q,k} & 0 \end{bmatrix}, \tag{3.26}$$

with entries given by,

$$\mathbf{D}_{i,\bar{i}}^{q,k} = -\Delta t^2 (\hat{\sigma}(\mathbf{c}^{q,i}))^2 \sigma' \left(\mathbf{A}_{k-1}^{q,i} \right) \mathbf{V}_{i\bar{i}}^q, \quad \mathbf{E}_{i,\bar{i}}^{q,k} = -\Delta t \hat{\sigma}(\mathbf{c}^{q,i}) \sigma' \left(\mathbf{A}_{k-1}^{q,i} \right) \mathbf{V}_{i\bar{i}}^q. \tag{3.27}$$

A direct calculation with (3.25) leads to,

$$\begin{aligned} \|\mathbf{B}_j^{L,i}\|_\infty &\leq \max\left(1 + \Delta t + (|\mathbf{w}^{L,i}| + \alpha)\Delta t^2, 1 + (|\mathbf{w}^{L,i}| + \alpha)\Delta t\right) \\ &\leq 1 + \max\left(2, |\mathbf{w}^{L,i}| + \alpha\right) \Delta t + \left(\max\left(2, |\mathbf{w}^{L,i}| + \alpha\right)\right)^2 \frac{\Delta t^2}{2}. \end{aligned} \tag{3.28}$$

Using the definition of the L^∞ norm of a matrix, we use (3.28) to the derive the following bound from (3.24),

$$\begin{aligned} \left\| \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \right\|_\infty &\leq 1 + \max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right) \Delta t + \left(\max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right)\right)^2 \frac{\Delta t^2}{2} \\ &\leq 1 + \gamma \Delta t, \end{aligned} \tag{3.29}$$

with γ defined in (3.19).

As $\Delta t < 1$, it is easy to see that,

$$\left\| \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \right\|_\infty \leq \|\mathbf{V}^q\|_\infty \Delta t. \tag{3.30}$$

Combining (3.29) and (3.30) , we obtain from (3.23)

$$\begin{aligned} \left\| \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \right\|_\infty &\leq \prod_{j=k+1}^{n} \left\| \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \right\|_\infty \prod_{q=\ell+1}^{L} \left\| \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \right\|_\infty \\ &\leq \Delta t^{L-\ell} \prod_{q=\ell+1}^{L} \|\mathbf{V}^q\|_\infty (1 + 2\gamma(n-k)\Delta t), \quad (\text{as } \Delta t << 1) \\ &\leq \overline{\mathbf{V}} \Delta t^{L-\ell} (1 + 2\gamma t_n), \end{aligned} \tag{3.31}$$

where the last inequality follows from the fact that $t_n = n\Delta t \leq T$ and the definition of $\overline{\mathbf{V}}$ in (3.19).

Next, we observe that for any $\theta \in \mathbf{\Theta}$

$$\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta} = \left[\frac{\partial^+ \mathbf{y}_k^{\ell,1}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,1}}{\partial \theta} \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{\ell,i}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta}, \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{\ell,m}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,m}}{\partial \theta} \right]^\top. \tag{3.32}$$

For any $1 \leq i \leq m$, a direct calculation with the RNN (3.10) yields,

$$\begin{aligned} \frac{\partial^+ \mathbf{y}_k^{\ell,i}}{\partial \theta} &= \Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \mathbf{z}_k^{\ell,i} + \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta}, \\ \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta} &= -\Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \sigma(\mathbf{A}_{k-1}^{\ell,i}) - \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \sigma'(\mathbf{A}_{k-1}^{\ell,i}) \frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} - \alpha \Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \mathbf{y}_{k-1}^{\ell,i}. \end{aligned} \tag{3.33}$$

Next, we have to compute explicitly $\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta}$ and $\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta}$ in order to complete the expressions (3.33). To this end, we need to consider explicit forms of the parameter θ and obtain,

If $\theta = \mathbf{w}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} \mathbf{y}_{k-1}^{\ell,i}, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{3.34}$$

If $\theta = \mathbf{b}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} 1, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{3.35}$$

If $\theta = \mathbf{V}_{p,\bar{p}}^q$, for some $1 \leq q \leq L$ and $1 \leq p, \bar{p} \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} \mathbf{y}_k^{\ell-1,\bar{p}}, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{3.36}$$

If $\theta = \mathbf{c}^{q,p}$for any $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} \equiv 0. \tag{3.37}$$

Similarly, if $\theta = \mathbf{w}^{q,p}$ or $\theta = \mathbf{b}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, or If $\theta = \mathbf{V}^{q}_{p,\bar{p}}$, for some $1 \leq q \leq L$ and $1 \leq p, \bar{p} \leq m$, then

$$\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \equiv 0. \tag{3.38}$$

On the other hand, if $\theta = \mathbf{c}^{q,p}$for any $1 \leq q \leq L$ and $1 \leq p \leq m$, then

$$\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} = \begin{cases} 1, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{3.39}$$

For any $\theta \in \mathbf{\Theta}$, by substituting (3.34) to (3.39) into (3.33) and doing some simple algebra with norms, leads to the following inequalities,

$$\left| \frac{\partial^{+} \mathbf{z}^{\ell,i}_{k}}{\partial \theta} \right| \leq \Delta t \left(1 + \alpha |\mathbf{y}^{\ell,i}_{k-1}| + \max \left(|\mathbf{y}^{\ell,i}_{k-1}|, |\mathbf{y}^{\ell-1,\bar{p}}_{k}|, 1 \right) \right), \tag{3.40}$$

and,

$$\left| \frac{\partial^{+} \mathbf{y}^{\ell,i}_{k}}{\partial \theta} \right| \leq \Delta t |\mathbf{z}^{\ell,i}_{k}| + \Delta t^2 \left(1 + \alpha |\mathbf{y}^{\ell,i}_{k-1}| + \max \left(|\mathbf{y}^{\ell,i}_{k-1}|, |\mathbf{y}^{\ell-1,\bar{p}}_{k}|, 1 \right) \right), \tag{3.41}$$

for any $1 \leq \bar{p} \leq m$.

By the definition of L^{∞} norm of a vector and some straightforward calculations with (3.41) yields,

$$\left\| \frac{\partial^{+} \mathbf{X}^{\ell}_{k}}{\partial \theta} \right\|_{\infty} \leq \Delta t \left(2 + \|\mathbf{z}^{\ell}_{k}\|_{\infty} + (1+\alpha) \|\mathbf{y}^{\ell}_{k-1}\|_{\infty} + \|\mathbf{y}^{\ell-1}_{k}\|_{\infty} \right). \tag{3.42}$$

From the pointwise bounds (3.11), we can directly bound the above inequality further as,

$$\left\| \frac{\partial^{+} \mathbf{X}^{\ell}_{k}}{\partial \theta} \right\|_{\infty} \leq \Delta t \left(2 + \sqrt{2\left(1 + 2\beta T\right)} + (2+\alpha) \sqrt{\frac{2}{\alpha}\left(1 + 2\beta T\right)} \right). \tag{3.43}$$

By (3.22) and the definition of $\overline{Y}$ as well as the bound (3.11) on the hidden states, it is straightforward to obtain that,

$$\left\| \frac{\partial \mathcal{E}_{n}}{\partial \mathbf{X}^{L}_{n}} \right\|_{\infty} \leq \overline{Y} + \sqrt{\frac{2}{\alpha}\left(1 + 2\beta T\right)} \tag{3.44}$$

From the definition in (3.21), we have

$$\left| \frac{\partial \mathcal{E}^{(n,L)}_{k,\ell}}{\partial \theta} \right| \leq \left\| \frac{\partial \mathcal{E}_{n}}{\partial \mathbf{X}^{L}_{n}} \right\|_{\infty} \left\| \frac{\partial \mathbf{X}^{L}_{n}}{\partial \mathbf{X}^{\ell}_{k}} \right\|_{\infty} \left\| \frac{\partial^{+} \mathbf{X}^{\ell}_{k}}{\partial \theta} \right\|_{\infty}. \tag{3.45}$$

Substituting (3.44), (3.43) and (3.29) into (3.45) yields,

$$\left| \frac{\partial \mathcal{E}^{(n,L)}_{k,\ell}}{\partial \theta} \right| \leq \Delta t^{L-\ell+1} \left(1 + 2\gamma T\right) \overline{\mathbf{V}} (\overline{Y} + \mathbf{F}) \mathbf{\Delta}, \tag{3.46}$$

with $\mathbf{F}$ and $\mathbf{\Delta}$ defined in (3.19).

Therefore, from the fact that $\Delta t < 1, t_n = n\Delta t \leq T$ and (3.21), we obtain

$$\left| \frac{\partial \mathcal{E}_{n}}{\partial \theta} \right| \leq \frac{1 - (\Delta t)^{L}}{1 - \Delta t} T (1 + 2\gamma T) \overline{\mathbf{V}} (\overline{Y} + \mathbf{F}) \mathbf{\Delta}. \tag{3.47}$$

By the definition of the loss function (3.16) and the fact that the right-hand-side of (3.47) is independent of n leads to the desired bound (3.18). □

This proposition demonstrates that as long as the weights $\mathbf{w}^L, \mathbf{V}^q$ are bounded, there is a uniform bound on the hidden state gradients. This bound grows at most as $(N\Delta t)^3$, with N being the total number of time steps. Thus, there is no exponential growth of the gradient with respect to the number of time steps and the *exploding gradient problem* is mitigated for UnICORNN.

On the Vanishing Gradient Problem for UnICORNN. By applying the chain rule repeatedly to the each term on the right-hand-side of (3.17), we obtain

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{\ell=1}^{L} \sum_{k=1}^{n} \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}, \ \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}. \tag{3.48}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k^\ell$ with respect to the parameter θ, while keeping the other arguments constant. The quantity $\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}$ denotes the contribution from the k-recurrent step at the l-th hidden layer of the deep RNN (3.10) to the overall hidden state gradient at the step n. The vanishing gradient problem [Pascanu et al., 2013] arises if $\left| \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} \right|$, defined in (3.48), $\to 0$ exponentially fast in k, for $k << n$ (long-term dependencies). In that case, the RNN does not have long-term memory, as the contribution of the k-th hidden state at the ℓ-th layer to error at time step t_n is infinitesimally small.

As argued before, the vanishing gradient problem for RNNs focuses on the possible smallness of contributions of the gradient over a large number of recurrent steps. As this behavior of the gradient is independent of the number of layers, we start with a result on the vanishing gradient problem for a single hidden layer here. Also, for the sake of definiteness, we set the scalar parameter $\theta = \mathbf{w}^{1,p}$ for some $1 \leq p \leq m$. Similar results also hold for any other $\theta \in \mathbf{\Theta}$. Moreover, we recall the *order*-notation of (2.35),

$$\begin{aligned} &\beta = \mathcal{O}(\gamma), \text{ for } \gamma, \beta \in \mathbb{R}_+ \quad \text{if there exist constants } \overline{C}, \underline{C} \text{ such that } \underline{C}\gamma \leq \beta \leq \overline{C}\gamma. \\ &\mathbf{M} = \mathcal{O}(\gamma), \text{ for } \mathbf{M} \in \mathbb{R}^{d_1 \times d_2}, \gamma \in \mathbb{R}_+ \quad \text{if there exists a constant } \overline{C} \text{ such that } \|\mathbf{M}\| \leq \overline{C}\gamma. \end{aligned} \tag{3.49}$$

Proposition 3.2.3. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN* (3.10)*. Then the gradient for long-term dependencies, i.e. $k << n$, satisfies the representation formula,*

$$\frac{\partial \mathcal{E}_{k,1}^{(n,1)}}{\partial \mathbf{w}^{1,p}} = -\Delta t \hat{\sigma}(\mathbf{c}^{1,p})^2 t_n \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p} \left(\mathbf{y}_n^{1,p} - \overline{\mathbf{y}}_n^p \right) + \mathcal{O}(\Delta t^2). \tag{3.50}$$

Proof. Following the definition (3.48) and as $L = 1$ and $\theta = \mathbf{w}^{1,p}$, we have,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,1)}}{\partial \mathbf{w}^{1,p}} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^1} \frac{\partial \mathbf{X}_n^1}{\partial \mathbf{X}_k^1} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}}. \tag{3.51}$$

We will explicitly compute all three expressions on the right-hand-side of (3.51). To start with, using (3.32), (3.33) and (3.34), we obtain,

$$\begin{aligned} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} &= \left[0, 0, \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \ldots, \ldots, 0, 0 \right]^\top, \\ \left(\frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} \right)_{2p-1} &= \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} = -\Delta t^2 (\hat{\sigma}(\mathbf{c}^{1,p}))^2 \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p}, \\ \left(\frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} \right)_{2p} &= \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} = -\Delta t \hat{\sigma}(\mathbf{c}^{1,p}) \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p}. \end{aligned} \tag{3.52}$$

Using the product rule (3.23) we have,

$$\frac{\partial \mathbf{X}_n^1}{\partial \mathbf{X}_k^1} = \prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}. \tag{3.53}$$

Observing from the expressions (3.24) and (3.25) and using the *order*-notation (3.49), we obtain that,

$$\frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{I}_{2m\times 2m} + \Delta t \mathbf{C}_j^1 + \mathcal{O}(\Delta t^2), \tag{3.54}$$

with $\mathbf{I}_{k\times k}$ is the $k \times k$ Identity matrix and the matrix $\mathbf{C}_j^1$ defined by,

$$\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} = \begin{bmatrix} \mathbf{C}_j^{1,1} & 0 & \dots & 0 \\ 0 & \mathbf{C}_j^{1,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}_j^{1,m} \end{bmatrix}, \tag{3.55}$$

with the block matrices $\mathbf{C}_j^{1,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{C}_j^{1,i} = \begin{bmatrix} 0 & \hat{\sigma}(\mathbf{c}^{1,i}) \\ -\hat{\sigma}(\mathbf{c}^{1,i}) \left(\mathbf{w}^{1,i} \sigma'(\mathbf{A}_{j-1}^{1,i}) + \alpha \right) & 0 \end{bmatrix}. \tag{3.56}$$

By a straightforward calculation and the use of induction, we claim that

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{I}_{2m\times 2m} + \Delta t \mathbf{C}^1 + \mathcal{O}(\Delta t^2), \tag{3.57}$$

with

$$\mathbf{C}^1 = \begin{bmatrix} \mathbf{C}^{1,1} & 0 & \dots & 0 \\ 0 & \mathbf{C}^{1,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}^{1,m} \end{bmatrix}, \tag{3.58}$$

with the block matrices $\mathbf{C}^{1,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{C}^{1,i} = \begin{bmatrix} 0 & (n-k)\hat{\sigma}(\mathbf{c}^{1,i}) \\ -(n-k)\alpha\hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i})\mathbf{w}^{1,i} \sum_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 0 \end{bmatrix}. \tag{3.59}$$

By the assumption that $k << n$ and using the fact that $t_n = n\Delta t$, we have that,

$$\Delta t \mathbf{C}^{1,i} = \begin{bmatrix} 0 & t_n\hat{\sigma}(\mathbf{c}^{1,i}) \\ -t_n\alpha\hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i})\mathbf{w}^{1,i}\Delta t \sum_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 0 \end{bmatrix}. \tag{3.60}$$

Hence, the non-zero entries in the block matrices can be $\mathcal{O}(1)$. Therefore, the product formula (3.57) is modified to,

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{C} + \mathcal{O}(\Delta t), \tag{3.61}$$

with the $2m \times 2m$ matrix $\mathbf{C}$ defined as,

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}^1 & 0 & \dots & 0 \\ 0 & \mathbf{C}^2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}^m \end{bmatrix}, \tag{3.62}$$

and,

$$\mathbf{C}^i = \begin{bmatrix} 1 & t_n \hat{\sigma}(\mathbf{c}^{1,i}) \\ -t_n \alpha \hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i}) \mathbf{w}^{1,i} \Delta t \sum_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 1 \end{bmatrix}. \tag{3.63}$$

Thus by taking the product of (3.61) with (3.52), we obtain that,

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} = \left[0, 0, \dots, \dots, \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} + \mathbf{C}_{12}^p \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \mathbf{C}_{21}^p \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} + \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} \dots, \dots, 0, 0 \right]^\top + \mathcal{O}(\Delta t^2), \tag{3.64}$$

with $\mathbf{C}_{12}^p, \mathbf{C}_{21}^p$ are the off-diagonal entries of the corresponding block matrix, defined in (3.63). Note that the $\mathcal{O}(\Delta t^2)$ remainder term arises from the Δt-dependence in (3.52).

From (3.22), we have that,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^1} = \left[\mathbf{y}_n^{1,1} - \overline{\mathbf{y}}_n^1, 0, \dots, \mathbf{y}_n^{1,i} - \overline{\mathbf{y}}_n^i, 0, \dots, \mathbf{y}_n^{1,m} - \overline{\mathbf{y}}_n^m, 0 \right]. \tag{3.65}$$

Therefore, taking the products of (3.65) and (3.64) and substituting the explicit expressions in (3.52), we obtain the desired identity (3.50). □

It is clear from the representation formula (3.50) that there is no k-dependence for the gradient. In particular, as long as all the weights are of $\mathcal{O}(1)$, the leading-order term in (3.50) is $\mathcal{O}(\Delta t)$. Hence, the gradient can be small but is independent of the recurrent step k. Thus, we claim that the *vanishing gradient problem*, with respect to recurrent connections, is mitigated for UnICORNN (3.10).

On the vanishing gradient problem for the multilayer version of UnICORNN. The explicit representation formula (3.50) holds for 1 hidden layer in (3.10). What happens when additional hidden layers are stacked together as in UnICORNN (3.10)? To answer this question, we consider the concrete case of $L = 3$ layers as this is the largest number of layers that we have used in the context of UnICORNN with fully connected stacked layers. As before, we set the scalar parameter $\theta = \mathbf{w}^{1,p}$ for some $1 \leq p \leq m$. Similar results also hold for any other $\theta \in \mathbf{\Theta}$. We have the following representation formula for the gradient in this case,

Proposition 3.2.4. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN* (3.10)*. The gradient for long-term dependencies satisfies the representation formula,*

$$\frac{\partial \mathcal{E}_{k,1}^{(n,3)}}{\partial \mathbf{w}^{1,p}} = \Delta t^4 \hat{\sigma}(\mathbf{c}^{1,p}) t_n \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} \sum_{i=1}^{m} \bar{\mathbf{G}}_{2i-1,2p-1} \left(\mathbf{y}^{3,i} - \overline{\mathbf{y}}^i \right) + \mathcal{O}(\Delta t^6), \tag{3.66}$$

with the coefficients given by,

$$\begin{aligned}\frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} &= -\Delta t \hat{\sigma}(\mathbf{c}^{1,p})\sigma'(\mathbf{A}_{k-1}^{1,p})\mathbf{y}_{k-1}^{1,p},\\ \bar{\mathbf{G}}_{2i-1,2p-1} = \sum_{j=1}^{m} \mathbf{G}_{ij}^3 \mathbf{G}_{jp}^2, \quad \forall\ 1 \le i \le m, \quad \mathbf{G}_{r,s}^q &= -(\hat{\sigma}(\mathbf{c}^{q,r}))^2 \sigma'\left(\mathbf{A}_{n-1}^{q,r}\right)\mathbf{V}_{rs}^q, \quad q = 2, 3.\end{aligned} \tag{3.67}$$

Proof. Following the definition (3.48) and as $L = 3$ and $\theta = \mathbf{w}^{1,p}$, we have,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,3)}}{\partial \mathbf{w}^{1,p}} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^3}\frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_k^1}\frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}}. \tag{3.68}$$

We will explicitly compute all three expressions on the right-hand-side of (3.68).

In (3.52), we have already explicitly computed the right most expression in the RHS of (3.68). Using the product rule (3.23) we have,

$$\frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_k^1} = \frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_n^2}\frac{\partial \mathbf{X}_n^2}{\partial \mathbf{X}_n^1}\prod_{j=k+1}^{n}\frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}. \tag{3.69}$$

Note that we have already obtained an explicit representation formula for $\prod_{j=k+1}^{n}\frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}$ in (3.61).

Next we consider the matrices $\frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_n^2}$ and $\frac{\partial \mathbf{X}_n^2}{\partial \mathbf{X}_n^1}$. By the representation formula (3.26), we have the following decomposition for any $1 \le q \le n$,

$$\frac{\partial \mathbf{X}_n^q}{\partial \mathbf{X}_n^{q-1}} = \Delta t^2 \mathbf{G}^{q,n} + \Delta t H^{q,n}, \tag{3.70}$$

with,

$$\mathbf{G}^{q,n} = \begin{bmatrix} \mathbf{G}_{11}^{q,n} & 0 & \mathbf{G}_{12}^{q,n} & 0 & \dots & \dots & \mathbf{G}_{1m}^{q,n} & 0 \\ 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{G}_{m1}^{q,n} & 0 & \mathbf{G}_{m2}^{q,n} & 0 & \dots & \dots & \mathbf{G}_{mm}^{q,n} & 0 \\ 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_{i,\bar{i}}^{q,k} = -(\hat{\sigma}(\mathbf{c}^{q,i}))^2 \sigma'\left(\mathbf{A}_{n-1}^{q,i}\right)\mathbf{V}_{i\bar{i}}^q, \tag{3.71}$$

and

$$\mathbf{H}^{q,n} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \mathbf{H}_{11}^{q,n} & 0 & \mathbf{H}_{12}^{q,n} & 0 & \dots & \dots & \mathbf{H}_{1m}^{q,n} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \mathbf{H}_{m1}^{q,n} & 0 & \mathbf{H}_{m2}^{q,n} & 0 & \dots & \dots & \mathbf{H}_{mm}^{q,n} & 0 \end{bmatrix}, \quad \mathbf{H}_{i,\bar{i}}^{q,k} = -\hat{\sigma}(\mathbf{c}^{q,i})\sigma'\left(\mathbf{A}_{n-1}^{q,i}\right)\mathbf{V}_{i\bar{i}}^q. \tag{3.72}$$

It is straightforward to see from (3.72) and (3.71) that,

$$\mathbf{H}^{3,n}\mathbf{H}^{2,n} \equiv \mathbf{0}_{2m\times 2m}, \quad \mathbf{G}^{3,n}\mathbf{H}^{2,n} \equiv \mathbf{0}_{2m\times 2m}, \tag{3.73}$$

and the entries of the $2m \times 2m$ matrix $\bar{\mathbf{G}} = \mathbf{G}^{3,n}\mathbf{G}^{2,n}$ are given by,

$$\bar{\mathbf{G}}_{2r-1,2s-1} = \sum_{j=1}^{m} \mathbf{G}^{3,n}_{r,j}\mathbf{G}^{2,n}_{j,s}, \quad \bar{\mathbf{G}}_{2r-1,2s} = \bar{\mathbf{G}}_{2r,2s-1} = \bar{\mathbf{G}}_{2r,2s} = 0, \quad \forall\ 1 \leq r, s \leq m, \tag{3.74}$$

while the entries of the $2m \times 2m$ matrix $\bar{\mathbf{H}} = \mathbf{H}^{3,n}\mathbf{G}^{2,n}$ are given by

$$\bar{\mathbf{H}}_{2r,2s-1} = \sum_{j=1}^{m} \mathbf{H}^{3,n}_{r,j}\mathbf{G}^{2,n}_{j,s}, \quad \bar{\mathbf{H}}_{2r-1,2s-1} = \bar{\mathbf{H}}_{2r-1,2s} = \bar{\mathbf{H}}_{2r,2s} = 0, \quad \forall\ 1 \leq r, s \leq m. \tag{3.75}$$

Hence we have,

$$\frac{\partial \mathbf{X}^3_n}{\partial \mathbf{X}^2_n}\frac{\partial \mathbf{X}^2_n}{\partial \mathbf{X}^1_n} = \Delta t^4(\bar{\mathbf{G}} + \Delta t^{-1}\bar{\mathbf{H}}). \tag{3.76}$$

Taking the matrix-vector product of (3.76) with (3.64), we obtain

$$\begin{aligned}\frac{\partial \mathbf{X}^3_n}{\partial \mathbf{X}^1_k}\frac{\partial^+ \mathbf{X}^1_k}{\partial \mathbf{w}^{1,p}} &= \Delta t^4 \left(\frac{\partial^+ \mathbf{y}^{1,p}_k}{\partial \mathbf{w}^{1,p}} + \mathbf{C}^p_{12}\frac{\partial^+ \mathbf{z}^{1,p}_k}{\partial \mathbf{w}^{1,p}}\right)\left[\bar{\mathbf{G}}_{1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2,2p-1}, \ldots, \bar{\mathbf{G}}_{2m-1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2m,2p-1}\right]^\top + \mathcal{O}(\Delta t^6) \\ &= \Delta t^4 \mathbf{C}^p_{12}\frac{\partial^+ \mathbf{z}^{1,p}_k}{\partial \mathbf{w}^{1,p}}\left[\bar{\mathbf{G}}_{1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2,2p-1}, \ldots, \bar{\mathbf{G}}_{2m-1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2m,2p-1}\right]^\top + \mathcal{O}(\Delta t^6),\end{aligned} \tag{3.77}$$

where the last identify follows from the fact that $\frac{\partial^+ \mathbf{y}^{1,p}_k}{\partial \mathbf{w}^{1,p}} = \mathcal{O}(\Delta t^2)$.

Therefore, taking the products of (3.65) and (3.77), we obtain the desired identity (3.66).

□

An inspection of the representation formula (3.66) shows that as long as the weights are $\mathcal{O}(1)$ and from the bounds (3.11), we know that $\mathbf{y} \sim \mathcal{O}(1)$, the gradient

$$\frac{\partial \mathcal{E}^{(n,3)}_{k,1}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}(\Delta t^5),$$

where the additional Δt stems from the Δt-term in (3.52). Thus the gradient does not depend on the recurrent step k. Hence, there is no vanishing gradient problem with respect to the number of recurrent connections, even in the multi-layer case.

However, it is clear from the representation formulas (3.50) and (3.66), as well as the proof of proposition 3.2.4 that for L-hidden layers in UnICORNN (3.10), we have,

$$\frac{\partial \mathcal{E}^{(n,L)}_{k,1}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}\left(\Delta t^{2L-1}\right). \tag{3.78}$$

Thus, the gradient can become very small if too many layers are stacked together. This is not at all surprising as such a behavior occurs even if there are no recurrent connections in UnICORNN (3.10). In that case, we simply have a fully connected deep neural network and it is well-known that the gradient can vanish as the number of layers increases, making it harder to train deep networks.

Residual stacking of layers in UnICORNN. Given the above considerations, it makes imminent sense to modify the fully-connected stacking of layers in UnICORNN (3.10) if a moderately large number of layers ($L \geq 4$) are used. It is natural to modify the fully-connected stacking with a residual stacking, see Li et al. [2019]. We use the following form of residual stacking,

$$\mathbf{y}_n^\ell = \mathbf{y}_{n-1}^\ell + \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot \mathbf{z}_n^\ell, \tag{3.79}$$

$$\mathbf{z}_n^\ell = \mathbf{z}_{n-1}^\ell - \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot [\sigma\left(\mathbf{w}^\ell \odot \mathbf{y}_{n-1}^\ell + \mathbf{x}_n^\ell + \mathbf{b}^l\right) + \alpha \mathbf{y}_{n-1}^\ell], \tag{3.80}$$

where the input $\mathbf{x}_n^\ell$ corresponds to a residual connection skipping S layers, i.e.

$$\mathbf{x}_n^\ell = \begin{cases} \mathbf{\Lambda}^\ell \mathbf{y}_n^{\ell-S-1} + \mathbf{V}^\ell \mathbf{y}_n^{\ell-1}, & \text{for } l > S \\ \mathbf{V}^\ell \mathbf{y}_n^{\ell-1}, & \text{for } l \leq S \end{cases}.$$

The number of skipped layers is $2 \leq S$ and $\Lambda^\ell \in \mathbb{R}^{m \times m}$ is a trainable matrix.

The main advantages of using a residual staking such as (3.79) is to alleviate the vanishing gradient problem that arises from stacking multiple layers together and obtain a better scaling of the gradient than (3.78). To see this, we can readily follow the proof of proposition 3.2.4, in particular the product,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_n^1} = \prod_{s=1}^{\nu} \frac{\partial \mathbf{X}_n^{L-(s-1)S}}{\partial \mathbf{X}_k^{L-sS}} \prod_{\ell=2}^{L-\nu S} \frac{\partial \mathbf{X}_n^\ell}{\partial \mathbf{X}_n^{\ell-1}} + \prod_{\ell=1}^{L-1} \frac{\partial \mathbf{X}_n^{\ell+1}}{\partial \mathbf{X}_k^\ell}, \tag{3.81}$$

with,

$$\nu = \begin{cases} \left[\frac{L}{S}\right], & \text{if} \quad L \bmod S \neq 0, \\ \left[\frac{L}{S}\right] - 1, & \text{if} \quad L \bmod S = 0. \end{cases} \tag{3.82}$$

Here $[x] \in \mathbb{N}$ is the largest natural number less than or equal to $x \in \mathbb{R}$.

Given the additive structure in the product of gradients and using induction over matrix products as in (3.73) and (3.74), we can compute that,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_n^1} = \mathcal{O}\left(\Delta t^{2(\nu+L-\nu S-1)}\right) + \mathcal{O}\left(\Delta t^{2(L-1)}\right). \tag{3.83}$$

By choosing S large enough, we clearly obtain that $\nu + L - \nu S - 1 < L - 1$. Hence by repeating the arguments of the proof of proposition 3.2.4, we obtain that to leading order, the gradient of the residual stacked version of UnICORNN scales like,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,L)}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}\left(\Delta t^{2\nu+2L-2\nu S-1}\right). \tag{3.84}$$

Note that (3.84) is far more favorable scaling for the gradient than the scaling (3.78) for a fully connected stacking. As a concrete example, let us consider $L = 7$ i.e., a network of 7 stacked layers of UniCORNN. From (3.78), we see that the gradient scales like $\mathcal{O}(\Delta t^{13})$ in this case. Even for a very moderate values of $\Delta t < 1$, this gradient will be very small and will ensure that the first layer will have very little, if any, influence on the loss function gradients. On the other hand, for the same number of layers $L = 7$, let us consider the residual stacking (3.79) with $S = 3$ skipped connections. In this case $\nu = 2$ and one directly concludes from (3.84) that the gradient scales like $\mathcal{O}(\Delta t^5)$, which is significantly larger than the gradient for the fully connected version of UnICORNN. In fact, it is exactly the same as the gradient scaling for fully connected UnICORNN (3.10) with 3 hidden layers (3.66). Thus, introducing skipped connections enabled the gradient to behave like a shallower fully-connected network, while possibly showing the expressivity of a deeper network.

3.3 Empirical results

Implementation. The structure of UnICORNN (3.6) enables us to achieve a very fast implementation. First, the transformation of the input (i.e. $\mathbf{V}^\ell \mathbf{y}_n^{\ell-1}$ for all $l = 1, \dots, L$), which is the most computationally expensive part of UnICORNN, does not have a sequential structure and can thus be computed in parallel over time. Second, as the underlying ODEs of the UnICORNN are uncoupled for each neuron, the remaining recurrent part of UnICORNN is solved independently for each component. Hence, inspired by the implementation of Simple Recurrent Units (SRU) [Lei et al., 2018] and IndRNN, we present the details of an efficient CUDA implementation. More concretely, we speed up the computation of the forward and backward pass, by parallelizing the input transformation and computing the recurrent part for each independent dimension in an independent CUDA thread. While the forward/backward pass for the input transformation is simply that of an affine transformation, we discuss only the recurrent part. Since we compute the gradients of each dimension of UnICORNN independently and add them up afterwards in order to obtain the full gradient, we simplify the dynamical system (3.6) as follows,

$$\begin{aligned} z_n &= z_{n-1} - \Delta t \hat{\sigma}(c)[\sigma\left(w y_{n-1} + x_n\right) + \alpha y_{n-1}], \\ y_n &= y_{n-1} + \Delta t \hat{\sigma}(c) z_n, \end{aligned}$$

where $x_n = (\mathbf{V}\mathbf{u}_n)_j$ is the transformed input corresponding to the respective dimension $j = 1, \dots, m$.

Since we wish to train the UnICORNN on some given objective

$$\mathcal{E} := \sum_{n=1}^{N} \tilde{\mathcal{E}}(y_n), \tag{3.85}$$

where $\tilde{\mathcal{E}}$ is some loss function taking the hidden states y_n as inputs, e.g. mean-squared distance of (possibly transformed) hidden states y_n to some ground truth. During training, we compute gradients of the loss function (3.85) with respect to the following quantities $\mathbf{\Theta} = [w, \Delta t, x_n]$, i.e.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial \theta}, \quad \forall\ \theta \in \mathbf{\Theta}. \tag{3.86}$$

We provide a recursion formula for computing the gradients in (3.86). We exemplarily provide the formula for the gradient with respect to the hidden weight w (the computation of the gradients with respect to the other quantities follow similarly),

$$\delta_k^z = \delta_{k-1}^z + \delta_{k-1}^y \Delta t \hat{\sigma}(c), \tag{3.87}$$

$$\delta_k^y = \delta_{k-1}^y - \delta_k^z \Delta t \hat{\sigma}(c)[\sigma'(w y_{N-k} + x_{N-k+1}) w + \alpha] + \frac{\partial \tilde{\mathcal{E}}}{\partial y_{N-k}}, \tag{3.88}$$

with initial values $\delta_0^y = \frac{\partial \tilde{\mathcal{E}}}{\partial y_N}$ and $\delta_0^z = 0$. The gradient can then be computed as

$$\frac{\partial \mathcal{E}}{\partial w} = \sum_{k=1}^{N} a_k, \qquad \text{with } a_k = -\delta_k^z \Delta t \hat{\sigma}(c) \sigma'(w y_{N-k} + x_{N-k+1}) y_{N-k}. \tag{3.89}$$

Note that this recursive formula is a direct formulation of the back-propagation through time algorithm [Werbos, 1990] applied to UnICORNN.

We can verify formula (3.87)-(3.89) by explicitly calculating the gradient in (3.86):

$$\frac{\partial \mathcal{E}}{\partial w} = \sum_{n=1}^{N} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} = \sum_{n=1}^{N-1} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + \frac{\partial \tilde{\mathcal{E}}}{\partial y_N} \left[\frac{\partial y_{N-1}}{\partial w} + \Delta t \hat{\sigma}(c) \left(\frac{\partial z_{N-1}}{\partial w} - \Delta t \hat{\sigma}(c)(\sigma'(w y_{N-1} + x_N) \right.\right.$$
$$\left.\left. (y_{N-1} + w \frac{\partial y_{N-1}}{\partial w}) + \alpha \frac{\partial y_{N-1}}{\partial w} \right) \right] = \sum_{n=1}^{N-2} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + a_1 + \delta_1^z \frac{\partial z_{N-1}}{\partial w} + \delta_1^y \frac{\partial y_{N-1}}{\partial w}$$
$$= \sum_{n=1}^{N-2} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + a_1 + \delta_1^y \frac{\partial y_{N-2}}{\partial w} + (\delta_1^y \Delta t \hat{\sigma}(c) + \delta_1^z) \left(\frac{\partial z_{N-2}}{\partial w} - \Delta t \hat{\sigma}(c)(\sigma'(w y_{N-2} + x_{N-1}) \right.$$
$$\left. (y_{N-2} + w \frac{\partial y_{N-2}}{\partial w}) + \alpha \frac{\partial y_{N-2}}{\partial w}) \right) = \sum_{n=1}^{N-3} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + \sum_{k=1}^{2} a_k + \delta_2^z \frac{\partial z_{N-2}}{\partial w} + \delta_2^y \frac{\partial y_{N-2}}{\partial w}.$$

Iterating the same reformulation yields the desired formula (3.87)-(3.89).

We benchmark the training speed of UnICORNN with $L = 2$ layers, against the fastest available RNN implementations, namely the cuDNN implementation [Appleyard et al., 2016] of LSTM (with 1 hidden layer), SRU and IndRNN (both with $L = 2$ layers and with batch normalization). Fig. 3.2 shows the computational time (measured on a GeForce RTX 2080 Ti GPU) of the combined forward and backward pass for each network, averaged over 100 batches with each of size 128, for two different sequence lengths, i.e. $N = 1000, 2000$. We can see that while the cuDNN LSTM is relatively slow, the SRU, IndRNN and the UnICORNN perform similarly fast. Moreover, we also observe that UnICORNN is about $30 - 40$ times faster per combined forward and backward pass, when compared to recently developed RNNs such as expRNN Lezcano-Casado and Martınez-Rubio [2019] and coRNN (Chapter 2). We thus conclude that the UnICORNN is among the fastest available RNN architectures.

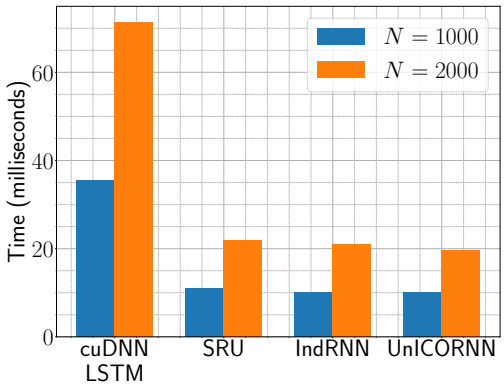

Figure 3.2: Measured computing time for the combined forward and backward pass for the UnICORNN as well as for three of the fastest available RNN implementations.

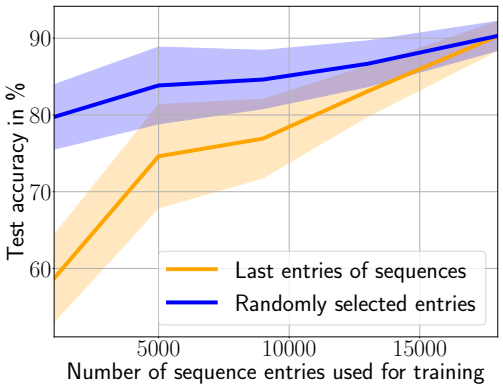

Figure 3.3: Test accuracy (mean and standard deviation) for UnICORNN on EigenWorms for two types of sub-sampling approaches, i.e. using the last entries of the sequences as well as using a random subset of the entries. Both are shown for increasing number of entries used in each corresponding sub-sampling routine.

Permuted sequential MNIST. A well-established benchmark for testing RNNs on input sequences with long-time dependencies is the permuted sequential MNIST (psMNIST) task [Le et al., 2015]. Based on the classical MNIST data set [LeCun et al., 1998], the flattened grey-scale matrices are randomly permuted (based on a fixed random permutation) and processed sequentially by the RNN. This makes the learning task more challenging than sequential MNIST, where one only flattens the MNIST matrices without permuting them. In order to make different methods comparable, we use the same fixed seed for the random permutation, as in Lezcano-Casado and Martınez-Rubio [2019], Casado [2019], Helfrich et al. [2018]. Table 3.1 shows the results for UnICORNN with 3 layers, together with other recently proposed RNNs, which were explicitly designed to learn long-term dependencies as well as two gated baselines. We see that UnICORNN clearly outperforms the other methods.

Table 3.1: Results on permuted sequential MNIST. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
LSTM	92.9%	256	270k
GRU	94.1%	256	200k
expRNN	96.6%	512	127k
coRNN	**97.3%**	256	134k
IndRNN (L=6)	96.0%	128	86k
dense-IndRNN (L=6)	97.2%	128	257k
UnICORNN (L=3)	**97.8%**	128	35k
UnICORNN (L=3)	**98.4%**	256	135k

Noise padded CIFAR-10. A more challenging test for the ability of RNNs to learn long-term dependencies is provided by the recently proposed noise padded CIFAR-10 experiment [Chang et al., 2019]. In it, the CIFAR-10 data points [Krizhevsky et al., 2009] are fed to the RNN row-wise and flattened along the channels resulting in sequences of length 32. To test long term memory, entries of uniform random numbers are added such that the resulting sequences have a length of 1000, i.e. the last 968 entries of each sequences are only noise to distract the RNNs. Table 3.2 shows the result of the UnICORNN with 3 layers together with the results of other recently proposed RNNs, namely for the LSTM, anti.sym. RNN and gated anti.sym. RNN [Chang et al., 2019], Lipschitz RNN [Erichson et al., 2020], Incremental RNN [Kag et al., 2020], FastRNN [Kusupati et al., 2018] and coRNN (Chapter 2). We conclude that the proposed RNN readily outperforms all other methods on this experiment.

EigenWorms. The EigenWorms data set Bagnall et al. [2018] is a collecting of 259 very long sequences, i.e. length of 17984, describing the motion of a worm. The task is, based on the 6-dimensional motion sequences, to classify a worm as either wild-type or one of four mutant types. We use the same train/valid/test split as in Morrill et al. [2020], i.e. 70%/15%/15%. As the length of the input sequences is extremely long for this test case, we benchmark UnICORNN against three sub-sampling based baselines. These include the results of Morrill et al. [2020], which is based on signature sub-sampling routine for neural controlled differential equations. Additionally after a hyperparameter fine-tuning procedure, we perform a random sub-sampling as well as truncated back-propagation through time (BPTT) routine using LSTMs, where the random sub-sampling is based on 200 randomly selected time points of the sequences as well as the BPTT is truncated after the last 500 time points of the sequences. Furthermore, we compare UnICORNN with three leading RNN architectures for solving long-term dependency tasks,

Table 3.2: Results on noise padded CIFAR-10. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
LSTM	11.6%	128	64k
Incremental RNN	54.5%	128	12k
Lipschitz RNN	**55.8%**	256	158k
FastRNN	45.8%	128	16k
anti.sym. RNN	48.3%	256	36k
gated anti.sym. RNN	54.7%	256	37k
coRNN	**59.0%**	128	46k
UnICORNN (L=3)	**62.4%**	128	47k

namely expRNN, IndRNN and coRNN, which are all applied to the full-length sequences. The results, presented in Table 3.3, show that while sub-sampling approaches yield moderate test accuracies, expRNN as well as the IndRNN yield very poor accuracies. In contrast, coRNN performs very well. However, the best results are obtained for UnICORNN as it reaches a test accuracy of more than 90%, while at the same time yielding a relatively low standard deviation, further underlining the robustness of the proposed RNN.

Table 3.3: Results on EigenWorms using 5 retrainings of each best performing network (based on the validation set). The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
t-BPTT LSTM	57.9% ± 7.0%	32	5.3k
sub-samp. LSTM	69.2% ± 8.3%	32	5.3k
sign.-NCDE	**77.8% ± 5.9%**	32	35k
expRNN	40.0% ± 10.1%	64	2.8k
IndRNN (L=2)	49.7% ± 4.8%	32	1.6k
coRNN	**86.7% ± 3.0%**	32	2.4k
UnICORNN (L=2)	**90.3% ± 3.0%**	32	1.5k

As this data set has only recently been proposed as a test for RNNs in learning long-term dependencies, it is unclear if the input sequences truly exhibit very long-time dependencies. To investigate this further, we train UnICORNN on a subset of the entries of the sequences. To this end, we consider using only the last entries as well as using a random subset of the entries. Fig. 3.3 shows the distributional results (10 re-trainings of the best performing UnICORNN) for the number of entries used in each sub-sampling routine, ranging from only using 1000 entries to using the full sequences for training. We can see that in order to reach a test accuracy of 80% when training on the last entries of the sequences, at least the last 10k entries are needed. Moreover, for both sub-sampling methods the test accuracy increases monotonically as the number of entries for training is increased. On the other hand, using a random subset of the entries increases the test accuracy significantly when compared to using only the last entries of the sequences. This indicates that the important entries of the sequences, i.e. information needed in order to classify them correctly, are uniformly distributed throughout the full sequence. We thus conclude that the EigenWorms data set indeed exhibits *very* long-time dependencies.

Healthcare application: Vital signs prediction. We apply UnICORNN on two real-world data sets in health care, aiming to predict the vital signs of a patient, based on PPG and ECG signals. The data sets are part of the TSR archive Tan et al. [2020] and are based on clinical data from the Beth Israel Deaconess Medical Center. The PPG and ECG signals are sampled with a frequency of 125Hz for 8 minutes each. The resulting two-dimensional sequences have a length of 4000. The goal is to predict a patient's respiratory rate (RR) and heart rate (HR) based on these signals. We compare UnICORNN to 3 leading RNN architectures for solving long-term dependencies, i.e. expRNN, IndRNN and coRNN. Additionally, we present two baselines using the LSTM as well as the recently proposed sub-sampling method of computing signatures for neural controlled differential equations (NCDE) Morrill et al. [2020]. Following Morrill et al. [2020], we split the 7949 sequences in a training set, validation set and testing set, using a 70%/15%/15% split. Table 3.4 shows the distributional results of all networks using 5 re-trainings of the best performing RNN. We observe that while the LSTM does not reach a low L^2 testing error in both experiments, the other RNNs approximate the vital signs reasonably well. However, UnICORNN clearly outperforms all other methods on both benchmarks. We emphasize that UnICORNN significantly outperforms all other state-of-the-art methods on estimating the RR, which is of major importance in modern healthcare applications for monitoring hospital in-patients as well as for mobile health applications, as special invasive equipment (for instance capnometry or measurement of gas flow) is normally needed to do so Pimentel et al. [2016].

Table 3.4: Results (L^2 test error) on vital sign prediction using 5 retrainings of each best performing network (based on the validation set), where the respiratory rate (RR) and heart rate (HR) is estimated based on PPG and ECG data. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	respiratory rate	heart rate
sign.-NCDE	1.51 ± 0.08	2.97 ± 0.45
LSTM	2.28 ± 0.25	10.7 ± 2.00
expRNN	1.57 ± 0.16	$\mathbf{1.87 \pm 0.19}$
IndRNN (L=3)	$\mathbf{1.47 \pm 0.09}$	2.10 ± 0.20
coRNN	$\mathbf{1.45 \pm 0.23}$	$\mathbf{1.71 \pm 0.10}$
UnICORNN (L=3)	$\mathbf{1.06 \pm 0.03}$	$\mathbf{1.39 \pm 0.09}$

Sentiment analysis: IMDB. As a final experiment, we test the proposed UnICORNN on the widely used NLP benchmark data set IMDB [Maas et al., 2011], which consists of 50k online movie reviews with 25k reviews used for training and 25k reviews used for testing. This denotes a classical sentiment analysis task, where the model has to decide whether a movie review is positive or negative. We use 30% of the training set (i.e. 7.5k reviews) as the validation set and restrict the dictionary to 25k words. We choose an embedding size of 100 and initialize it with the pretrained 100d GloVe Pennington et al. [2014] vectors. Table 3.5 shows the results for UnICORNN with 2 layers together with other recently proposed RNN architectures and gated baselines (which are known to perform very well on these tasks). The result of ReLU GRU is taken from Dey and Salemt [2017], of coRNN from Chapter 2 and all other results are taken from Campos et al. [2018]. We can see that UnICORNN outperforms the other methods while requiring significantly less parameters. We thus conclude, that the UnICORNN can also be successfully applied to problems, which do not necessarily exhibit long-term dependencies.

Table 3.5: Results on IMDB. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
LSTM	**86.8%**	128	220k
skip LSTM	86.6%	128	220k
GRU	85.2%	128	99k
ReLU GRU	84.8%	128	99k
skip GRU	86.6%	128	165k
coRNN	**87.4%**	128	46k
UnICORNN (L=2)	**88.4%**	128	30k

Chaotic time-series prediction: Lorenz 96 system. It is instructive to explore limitations of the proposed UnICORNN. It is straightforward to prove, along the lines of the proof of proposition 3.2.1, that the UnICORNN architecture does not exhibit chaotic behavior with respect to changes in the input. While this property is highly desirable for many applications where a slight change in the input should not lead to a major (possibly unbounded) change in the output, it impairs the performance on tasks where an actual chaotic system has to be learned.

Following the experiment in Chapter 2, we aim to predict future states of a dynamical system, following the Lorenz 96 system [Lorenz, 1996]:

$$x_j' = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, \tag{3.90}$$

where $x_j \in \mathbb{R}$ for all $j = 1, \dots, 5$ and F is an external force controlling the level of chaos in the system.

We consider two different choices for the external force, namely $F = 0.9$ and $F = 8$. While the first one produces non-chaotic trajectories, the latter leads to a highly chaotic system. We discretize the system exactly along the lines of Chapter 2, resulting in 128 sequences of length 2000 for each the training, validation and testing set. Table 3.6 shows the normalized root mean square error (NRMSE) for

Table 3.6: Test NRMSE on the Lorenz 96 system (3.90) for UnICORNN, coRNN and LSTM.

Model	$F = 0.9$	$F = 8$	# units	# params
LSTM	2.0×10^{-2}	6.8×10^{-2}	44	9k
coRNN	2.0×10^{-2}	9.8×10^{-2}	64	9k
UnICORNN (L=2)	2.2×10^{-2}	3.1×10^{-1}	90	9k

UnICORNN as well as for coRNN and an LSTM, where all models have 9k parameters. We can see that UnICORNN performs comparably to coRNN and LSTM in the chaos-free regime (i.e. $F = 0.9$), while performing poorly compared to an LSTM when the system exhibits chaotic behavior (i.e. $F = 8$). This is not surprising, as LSTMs are shown to be able to exhibit chaotic behavior [Laurent and von Brecht, 2017], while coRNN and UnICORNN are not chaotic by design. This shows also numerically that UnICORNN should not be used for chaotic time-series prediction.

Further experimental results. As stated before, UnICORNN has two hyperparameters, i.e. the maximum allowed time-step Δt and the damping parameter α. It is of interest to examine how sensitive the performance of UnICORNN is with respect to variations of these hyperparameters. To this end, we

consider the noise padded CIFAR-10 experiment and perform an ablation study of the test accuracy with respect to variations of both α and Δt. Both hyperparameters are varied by an order of magnitude and the results of this study are plotted in Fig. 3.4. We observe from this figure, that the results are indeed somewhat sensitive to the maximum allowed time-step Δt and show a variation of approximately 15% with respect to to this hyperparameter. On the other hand, there is very little noticeable variation with respect to the damping parameter α. Thus, it can be set to a default value, for instance $\alpha = 1$, without impeding the performance of the underlying RNN. Next, we recall that the design of UnICORNN (3.6)

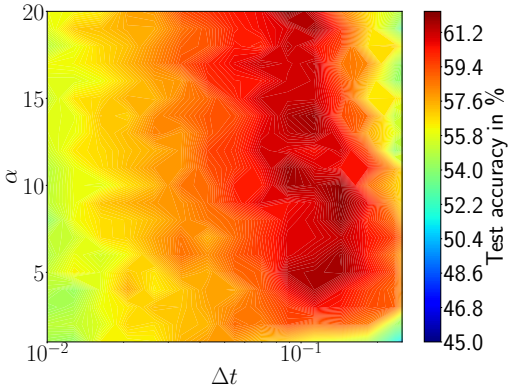

Figure 3.4: Ablation study on the hyperparameters Δt and α of UnICORNN (3.6) using the noise padded CIFAR-10 experiment.

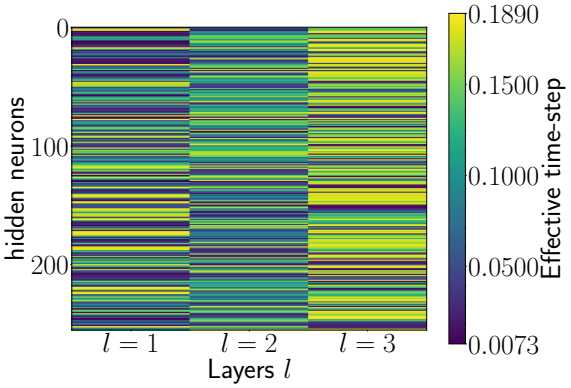

Figure 3.5: Effective time-step $\Delta t\hat{\sigma}(\mathbf{c}_i^l)$ for each hidden neuron $i = 1, \ldots, m$ and each layer $l = 1, \ldots, L$ of UnICORNN, after training on the psMNIST task using $m = 256$ hidden units and $L = 3$ layers.

enables it to learn the effective time step (with a possible maximum of Δt) from data. It is instructive to investigate if this ability to express *multiscale* behavior is realized in practice or not. To this end, we consider the trained UnICORNN of the psMNIST task with 3 layers and 256 neurons. Here, a maximum time step of $\Delta t = 0.19$ was identified by the hyperparameter tuning. In Fig. 3.5, we plot the effective time step $\Delta t\hat{\sigma}(\mathbf{c}_i^l)$, for each hidden neuron $i = 1, \ldots, 256$ and each layer $l = 1, 2, 3$. We observe from this figure that a significant variation of the effective time step is observed, both within the neurons in each layer, as well as between layers. In particular, the minimum effective time step is about 28 times smaller than the maximum allowed time step. Thus, we conclude from this figure, that UnICORNN exploits its design features to learn multiscale behavior that is latent in the data. This perhaps explains the superior performance of UnICORNN on many learning tasks.

Moreover, as we compare the results of the UnICORNN to the results of other recent RNN architecture, where only the best results of each RNN were published for the psMNIST, noise padded CIFAR-10 and IMDB task, we follow the same procedure by showing the best (based on a validation set) obtained results for UnICORNN. However, distributional results, i.e. statistics of several re-trainings of the best performing UnICORNN based on different random initialization of the trainable parameters, provide additional insights into the performance. Table 3.7 shows the mean and standard deviation of 10 re-trainings of the best performing UnICORNN for the psMNIST, noise padded CIFAR-10 and IMDB task. We can see that in all experiments the standard deviation of the re-trainings are relatively low, which underlines the robustness of our presented results.

As emphasized previously in this chapter, naively stacking of many layers for UnICORNN might

Table 3.7: Distributional information (mean and standard deviation) on the results for the classification experiment presented in this chapter, where only the best results is shown, based on 10 re-trainings of the best performing UnICORNN using different random seeds.

Experiment	Mean	Standard deviation
psMNIST (128 units)	97.7%	0.09%
psMNIST (256 units)	98.2%	0.22%
Noise padded CIFAR-10	61.5%	0.52%
IMDB	88.1%	0.19%

result in a vanishing gradient for the deep multi-layer model, due to the vanishing gradient problem of stacking many (not necessarily recurrent) layers. However, one can use skipped residual connections. In this context, we see that the estimate on the gradients scale preferably when using residual connections compared to a naively stacking, when using many layers. To test this also numerically, we train a standard UnICORNN (3.10) as well as a residual UnICORNN (res-UnICORNN) (3.79), with $S = 2$ skipping layers, on the noise padded CIFAR-10 task. Fig. 3.6 shows the test accuracy (mean and standard deviation) of the best resulting model for different number of network layers $L = 3, \dots, 6$, for the standard UnICORNN and res-UnICORNN. We can see that while both models seem to perform comparably for using only few layers, i.e. $L = 3, 4$, the res-UnICORNN with $S = 2$ skipping connections outperforms the standard UnICORNN when using more layers, i.e. $L = 5, 6$. In particular, we can see that the standard UnICORNN is not able to significantly improve the test accuracy when using more layers, while the res-UnICORNN seems to obtain higher test accuracies when using more layers.

Moreover, Fig. 3.6 also shows the test accuracy for a UnICORNN with an untrained time-step vector $\mathbf{c}$, resulting in a UnICORNN without the multiscale property generated by the time-step. We can see that the UnICORNN without the multiscale feature is inferior in performance, to the standard UnICORNN as well as its residual counterpart.

Finally, we recall that the estimate (3.18) on the gradients for UnICORNN (3.10) needs the weights to be bounded, see (3.19). One always initializes the training with bounded weights. However, it might happen that the weights explode during training. To check this issue, in Fig. 3.7, we plot the mean and standard deviation of the norms of the hidden weights $\mathbf{w}^l$ for $l = 1, 2, 3$ during training based on 10 re-trainings of the best performing UnICORNN on the noise padded CIFAR-10 experiment. We can see that none of the norms of the weights explode during training. In fact the weight norms seem to saturate, mostly on account of reducing the learning rate after 250 epochs. Thus, the upper bound (3.18) can be explicitly computed and it is finite, even after training has concluded.

Training details. All experiments were conducted on GPUs, namely NVIDIA GeForce GTX 1080 Ti and NVIDIA GeForce RTX 2080 Ti. The hidden weights $\mathbf{w}$ of UnICORNN are initialized according to $\mathcal{U}(0, 1)$, while all bias values are set to zero. The trained vector $\mathbf{c}$ is initialized according to $\mathcal{U}(-0.1, 0.1)$. The input weights $\mathbf{V}$ are initialized according to the Kaiming uniform initialization [He et al., 2015] based on the input dimension mode and the negative slope of the rectifier set to $a = 8$.

The hyperparameters of the UnICORNN are selected using a random search algorithm based on a validation set. The hyperparameters of the best performing UnICORNN can be seen in Table 3.8. The value for Δt and α is shared across all layers, except for the IMDB task and EigenWorms task, where we use a different Δt value for the first layer and the corresponding Δt value in Table 3.8 for all subsequent layers, i.e. we use $\Delta t = 6.6 \times 10^{-3}$ for IMDB and $\Delta t = 2.81 \times 10^{-5}$ for EigenWorms in the first layer. Additionally, the dropout column corresponds to variational dropout [Gal and Ghahramani, 2016], which

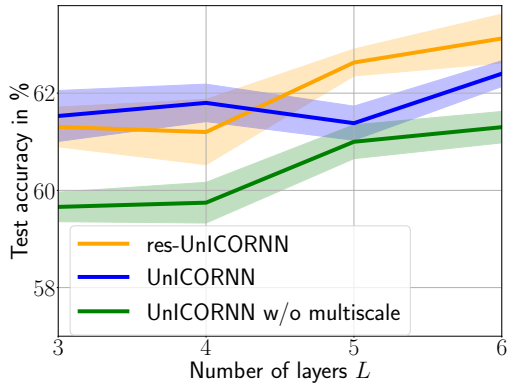

Figure 3.6: Test accuracies (mean and standard deviation of 10 re-trainings of the best performing model) of the standard UnICORNN, res-UnICORNN and UnICORNN without multiscale behavior on the noise padded CIFAR-10 experiment for different number of layers L.

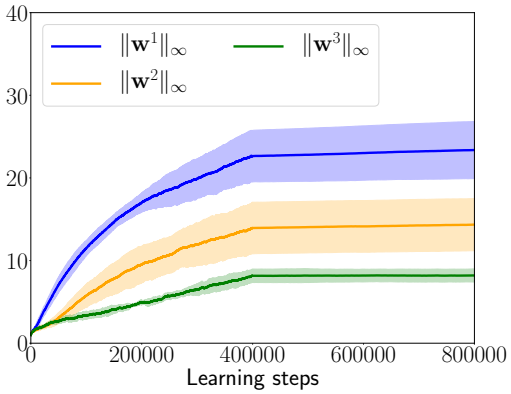

Figure 3.7: Norms (mean and standard deviation of 10 re-trainings) of the hidden weights $\|\mathbf{w}^l\|_\infty$, for $l = 1, 2, 3$, of the UnICORNN during training.

is applied after each consecutive layer. Note that for the IMDB task also an embedding dropout with $p = 0.65$ is used.

We train the UnICORNN for a total of 50 epochs on the IMDB task and for a total of 250 epochs on the EigenWorms task. Moreover, we train UnICORNN for 650 epochs on psMNIST, after which we decrease the learning rate by a factor of 10 and proceed training for 3 times the amount of epochs used before reducing the learning rate. On all other tasks, UnICORNN is trained for 250 epochs, after which we decrease the learning rate by a factor of 10 and proceed training for additional 250 epochs. The resulting best performing networks are selected *based on a validation set*.

Table 3.8: Hyperparameters of the best performing UnICORNN architecture (based on a validation set) for each experiment.

Experiment	learning rate	dropout	batch size	Δt	α
noise padded CIFAR-10	3.14×10^{-2}	1.0×10^{-1}	30	1.26×10^{-1}	13.0
psMNIST (#units = 128)	1.14×10^{-3}	1.0×10^{-1}	64	4.82×10^{-1}	12.53
psMNIST (#units = 256)	2.51×10^{-3}	1.0×10^{-1}	32	1.9×10^{-1}	30.65
IMDB	1.67×10^{-4}	6.1×10^{-1}	32	2.05×10^{-1}	0.0
EigenWorms	8.59×10^{-3}	0.0	8	3.43×10^{-2}	0.0
Healthcare: RR	3.98×10^{-3}	1.0×10^{-1}	32	1.1×10^{-2}	9.0
Healthcare: HR	2.88×10^{-3}	1.0×10^{-1}	32	4.6×10^{-2}	10.0

3.4 Discussion

The design of RNNs that can accurately handle sequential inputs with long-time dependencies is very challenging. This is largely on account of the exploding and vanishing gradients problem. Moreover, there is a significant increase in both computational time as well as memory requirements when long-term dependency tasks have to be processed. Our main aim in this chapter was to present a novel RNN architecture which is fast, memory efficient, *invertible* and mitigates the exploding and vanishing gradients problem. To this end, we proposed UnICORNN (3.6), an RNN based on the symplectic Euler discretization of a Hamiltonian system of second-order ODEs (3.2) modeling a network of independent, undamped, controlled and driven oscillators. In order to gain expressivity, we stack layers of RNNs and also endow this construction with a multiscale feature by training the effective time step in (3.6).

Given the Hamiltonian structure of our continuous and discrete dynamical system, invertibility and volume preservation in phase space are guaranteed. Invertibility enables the proposed RNN to be memory efficient. The independence of neurons within each hidden layer allows us to build a highly efficient CUDA implementation of UnICORNN that is as fast as the fastest available RNN architectures. Under suitable smallness constraints on the maximum allowed time step Δt, we prove rigorous upper bounds (3.18) on the gradients and show that the exploding gradient problem is mitigated for UnICORNN. Moreover, we derive an explicit representation formula (3.50) for the gradients of (3.6), which shows that the vanishing gradient problem is also mitigated. Finally, we have tested UnICORNN on a suite of benchmarks that includes both synthetic as well as realistic learning tasks, designed to test the ability of an RNN to deal with long-time dependencies. In all the experiments, UnICORNN was able to show state-of-the-art performance.

It is instructive to compare UnICORNN with two recently proposed RNN architectures, with which it shares some essential features. First, the use of coupled oscillators to design RNNs was already explored in the case of coRNN (Chapter 2). In contrast to coRNN, neurons in UnICORNN are independent (uncoupled) and as there is no damping, UnICORNN possesses a Hamiltonian structure. This paves the way for invertibility as well as for mitigating the exploding and vanishing gradients problem without any assumptions on the weights whereas the mitigation of exploding and vanishing gradients problem with coRNN was conditional on restrictions on weights. Finally, UnICORNN provides even better performance on benchmarks than coRNN, while being significantly faster. While we also use independent neurons in each hidden layer and stack RNN layers together as in IndRNN Li et al. [2018], our design principle is completely different as it is based on Hamiltonian ODEs. Consequently, we do not impose weight restrictions, which are necessary for IndRNN to mitigate the exploding and vanishing gradients problem. Moreover, in contrast to IndRNNs, our architecture is invertible and hence, memory efficient.

This work can be extended in different directions. First, UnICORNN is a very flexible architecture in terms of stacking layers of RNNs together. We have used a fully connected stacking in (3.6) but other possibilities can be readily explored. Second, the invertibility of UnICORNN can be leveraged in the context of normalizing flows Papamakarios et al. [2019], where the objective is to parametrize a flow such that the resulting Jacobian is readily computable. Finally, our focus in this chapter was on testing UnICORNN on learning tasks with long-time dependencies. Given that the underlying ODE (3.2) models oscillators, one can envisage that UnICORNN will be very competitive with respect to processing different time series data that arise in healthcare AI such as EEG and EMG data, as well as seismic time series from the geosciences.

Chapter 4

Neural Oscillators are Universal

Oscillators are ubiquitous in the sciences and engineering Guckenheimer and Holmes [1990], Strogatz [2015]. Prototypical examples include pendulums in mechanics, feedback and relaxation oscillators in electronics, business cycles in economics and heart beat and circadian rhythms in biology. Particularly relevant to our context is the fact that the neurons in our brain can be thought of as oscillators on account of the periodic spiking and firing of the action potential Stiefel and Ermentrout [2016], Gu et al. [2015]. Consequently, functional brain circuits such as cortical columns are being increasingly analyzed in terms of networks of coupled oscillators Stiefel and Ermentrout [2016].

Given this wide prevalence of (networks of) oscillators in nature and man-made devices, it is not surprising that oscillators have inspired various machine learning architectures in recent years. Prominent examples include the RNN architectures introduced in the previous two chapters, i.e., coRNN in Chapter2 and UnICORNN in Chapter 3. Other examples include Second Order Neural ODEs (SONODEs) [Norcliffe et al., 2020], which can be interpreted as oscillatory neural ODEs, locally coupled oscillatory recurrent networks (LocoRNN) [Keller and Welling, 2023], and Oscillatory Fourier Neural Network (O-FNN) [Han et al., 2022].

Another avenue where machine learning models based on oscillators arise is that of *physical neural networks* (PNNs) Wright et al. [2022] i.e., physical devices that perform machine learning on analog (beyond digital) systems. Such analog systems have been proposed as alternatives or accelerators to the current paradigm of machine learning on conventional electronics, allowing us to significantly reduce the prohibitive energy costs of training state-of-the-art machine learning models. In Wright et al. [2022], the authors propose a variety of physical neural networks which include a mechanical network of multi-mode oscillations on a plate and electronic circuits of oscillators as well as a network of nonlinear oscillators. Coupled with a novel *physics aware training* (PAT) algorithm, the authors of Wright et al. [2022] demonstrated that their nonlinear oscillatory PNN achieved very good performance on challenging benchmarks such as Fashion-MNIST [Xiao et al., 2017]. Moreover, other oscillatory systems such as coupled lasers and spintronic nano-oscillators have also been proposed as possible PNNs, see Velichko et al. [2019] as an example of the use of thermally coupled vanadium dioxide oscillators for image recognition and Romera and et. al [2018], Torrejon and et. al [2017] for the use of spin-torque nano-oscillators for speech recognition and for neuromorphic computing, respectively.

What is the rationale behind the successful use of (networks of) oscillators in many different contexts in machine learning? In Chapter 2 and Chapter 3 we argued that it is because of the inherent stability of oscillatory dynamics, as the state (and its gradients) of an oscillatory system remain within reasonable bounds throughout the time-evolution of the system. However, this is at best a partial explanation, as it does not demonstrate why oscillatory dynamics can learn (approximate) mappings between inputs and

outputs rather than bias the learned states towards oscillatory functions. As an example, consider the problem of classification of MNIST [LeCun et al., 1998] (or Fashion-MNIST) images. It is completely unclear if the inputs (vectors of pixel values), outputs (class probabilities) and the underlying mapping possess any (periodic) oscillatory structure. Consequently, how can oscillatory RNNs (such as CoRNN and UnICORNN) or a network of oscillatory PNNs learn the underlying mapping?

Our main aim in this chapter is to provide an answer to this very question on the ability of neural networks, based on oscillators, to express (to approximate) arbitrary mappings. To this end,

- We introduce an abstract framework of *neural oscillators* that encompasses both sequence models such as CoRNN and UnICORNN, as well as variants of physical neural networks as the ones proposed in Wright et al. [2022]. These neural oscillators are defined in terms of second-order versions of *neural ODEs* Chen et al. [2018], and combine *nonlinear dynamics* with a *linear read-out*.

- We prove a *Universality* theorem for neural oscillators by showing that they can approximate, to any given tolerance, continuous operators between appropriate function spaces.

- Our proof of universality is based on a novel theoretical result of independent interest, termed the *fundamental Lemma*, which implies that a suitable combination of *linear oscillator dynamics* with *nonlinear read-out* suffices for universality.

Such universality results, Barron [1993], Cybenko [1989], Hornik et al. [1989], Pinkus [1999] and references therein, have underpinned the widespread use of traditional neural networks (such as multi-layer perceptrons and convolutional neural networks). Hence, our universality result establishes a firm mathematical foundation for the deployment of neural networks, based on oscillators, in myriad applications. Moreover, our constructive proof provides insight into how networks of oscillators can approximate a large class of mappings.

4.1 Neural Oscillators

General Form of Neural Oscillators. Given $u : [0, T] \to \mathbb{R}^p$ as an input signal, for any final time $T \in \mathbb{R}_+$, we consider the following system of *neural* ODEs for the evolution of dynamic hidden variables $y \in \mathbb{R}^m$, coupled to a linear read-out to yield the output $z \in \mathbb{R}^q$,

$$\begin{cases} \ddot{y}(t) = \sigma\left(Wy(t) + Vu(t) + b\right), & \text{(4.1a)} \\ y(0) = \dot{y}(0) = 0, & \text{(4.1b)} \\ z(t) = Ay(t) + c. & \text{(4.1c)} \end{cases}$$

Equation (4.1) defines an input-/output-mapping $u(t) \mapsto z(t)$, with time-dependent output $z : [0, T] \to \mathbb{R}^q$. Specification of this system requires a choice of the hidden variable dimension m and the activation function σ. The resulting mapping $u \mapsto z$ depends on tunable weight matrices $W \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{m\times p}$, $A \in \mathbb{R}^{q\times m}$ and bias vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^q$. For simplicity of the exposition, we consider only *activation functions* $\sigma \in C^\infty(\mathbb{R})$*, with* $\sigma(0) = 0$ *and* $\sigma'(0) = 1$, such as tanh or sin, although more general activation functions can be readily considered. This general second-order neural ODE system (4.1) will be referred to as a **neural oscillator**.

Multi-layer neural oscillators. As a special case of neural oscillators, we consider the following *much sparser* class of second-order neural ODEs,

$$\begin{cases} y^0(t) := u(t), & \text{(4.2a)} \\ \ddot{y}^\ell(t) = \sigma\left(w^\ell \odot y^\ell(t) + V^\ell y^{\ell-1}(t) + b^\ell\right), \quad (\ell = 1, \dots, L), & \text{(4.2b)} \\ y^\ell(0) = \dot{y}^\ell(0) = 0, & \text{(4.2c)} \\ z(t) = Ay^L(t) + c. & \text{(4.2d)} \end{cases}$$

In contrast to the general neural oscillator (4.1), the above multi-layer neural oscillator (4.2) defines a *hierarchical* structure; The solution $y^\ell \in \mathbb{R}^{m_\ell}$ at level ℓ solves a second-order ODE with driving force $y^{\ell-1}$, and the lowest level, $y^0 = u$, is the input signal. Here, the layer dimensions $m_1, \dots, m_L$ can vary across layers, the weights $w^\ell \in \mathbb{R}^{m_\ell}$ are given by vectors, with $\odot$ componentwise multiplication, $V^\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$ is a weight matrix, and $b^\ell \in \mathbb{R}^{m_\ell}$ the bias. Given the result of the final layer, y^L, the output signal is finally obtained by an affine output layer $z(t) = Ay^L(t) + c$. In the multi-layer neural oscillator, the matrices V^ℓ, A and vectors w^ℓ, b^ℓ and c represent the trainable hidden parameters. The system (4.2) is a special case of (4.1), since it can be written in the form (4.1), with $y := [y^L, y^{L-1}, \dots, y^1]^T$, $b := [b^L, \dots, b^1]^T$, and a (upper-diagonal) block-matrix structure for W:

$$W := \begin{bmatrix} w^L I & V^L & 0 & \dots & 0 \\ 0 & w^{L-1} I & V^{L-1} & \ddots & \vdots \\ \vdots & \ddots & & \ddots & 0 \\ 0 & \dots & 0 & w^2 I & V^2 \\ 0 & \dots & 0 & 0 & w^1 I \end{bmatrix}, \quad V := \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ V^1 \end{bmatrix} \tag{4.3}$$

Given the block-diagonal structure of the underlying weight matrices, it is clear that the multi-layer neural oscillator (4.2) is a much sparser representation of the general neural operator (4.1). Moreover, one can observe from the structure of the neural ODE (4.2) that within each layer, the individual neurons are *independent* of each other.

Assuming that $w_i^\ell \neq 0$, for all $1 \leq i \leq m_\ell$ and all $1 \leq \ell \leq L$, we further highlight that the multi-layer neural oscillator (4.2) is a Hamiltonian system,

$$\dot{y}^\ell = \frac{\partial H}{\partial \dot{y}^\ell}, \quad \ddot{y}^\ell = -\frac{\partial H}{\partial y^\ell}, \tag{4.4}$$

with the *layer-wise time-dependent Hamiltonian*,

$$H(y^\ell, \dot{y}^\ell, t) = \frac{1}{2}\|\dot{y}^\ell\|^2 - \sum_{i=1}^{m_\ell} \frac{1}{w_i^\ell} \hat{\sigma}(w_i^\ell y_i^\ell + (V^\ell y^{\ell-1})_i + b_i^\ell), \tag{4.5}$$

with $\hat{\sigma}$ being the antiderivative of σ, and $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$ denoting the Euclidean norm of the vector $\mathbf{x} \in \mathbb{R}^m$ and $\langle \cdot, \cdot \rangle$ the corresponding inner product. Hence, any symplectic discretization of the multi-layer neural oscillator (4.2) will result in a fully reversible model, which can first be leveraged in the context of normalizing flows [Rezende and Mohamed, 2015], and second leads to a memory-efficient training, as the intermediate states (i.e., $y^\ell(t_0), \dot{y}^\ell(t_0), y_\ell(t_1), \dot{y}_\ell(t_1), \dots, y_\ell(t_N), \dot{y}_\ell(t_N)$, for some time discretization $t_0, t_1, \dots, t_N$ of length N) do not need to be stored and can be reconstructed during the backward pass. This potentially leads to a drastic memory saving of $\mathcal{O}(N)$ during training.

4.1.1 Examples of Neural Oscillators

(Forced) harmonic oscillator. Let $p = m = q = 1$ and we set $W = -\omega^2$, for some $\omega \in \mathbb{R}$, $V = 1, b = 0$ and the activation function to be identity $\sigma(x) = x$. In this case, the neural ODE (4.1) reduces to the ODE modeling the dynamics of a forced simple harmonic oscillator Guckenheimer and Holmes [1990] of the form,

$$\ddot{y} = -\omega^2 y + u, \quad y(0) = \dot{y}(0) = 0. \tag{4.6}$$

Here, y is the displacement of the oscillator, ω the frequency of oscillation and u is a forcing term that forces the motion of the oscillator. Note that (4.6) is also a particular example of the multi-layer oscillator (4.2) with $L = 1$.

This simple example provides justification for our terminology of neural oscillators, as in general, the hidden state y can be thought of as the vector of displacements of m-coupled oscillators, which are coupled together through the weight matrix W and are forced through a forcing term u, whose effect is modulated via V and a bias term b. The nonlinear activation function mediates possible nonlinear feedback to the system on account of large displacements.

CoRNN. We recall that the Coupled Oscillatory RNN (CoRNN) architecture (2.3) of Chapter 2 is given by the neural ODE:

$$\ddot{y} = \sigma\left(Wy + \mathcal{W}\dot{y} + Vu + b\right) - \gamma y - \epsilon \dot{y}.$$

Hence, we can recover the neural oscillator (4.1) as a special case of CoRNN by setting $\mathcal{W} = \mathbf{0}, \gamma = \epsilon = 0$; thus, a universality theorem for neural oscillators immediately implies a corresponding universality result for the CoRNN architecture.

UnICORNN. The Undamped Independent Controlled Oscillatory RNN (UnICORNN) architecture (3.6) of Chapter 3 recovers the multi-layer neural oscillator (4.2) in the case where the fundamental frequencies of UnICORNN are automatically determined inside the weight matrix W in (4.1).

Nonlinear oscillatory PNN of Wright et al. [2022]. In [Wright et al., 2022, SM, Sect. 4.A], the authors propose an analog machine learning device that simulates a network of nonlinear oscillators, for instance realized through coupled pendula. The resulting mathematical model is the so-called simplified Frenkel-Kontorova model Braun and Kivshar [1998] given by the ODE system,

$$M\ddot{\theta} = -K\sin(\theta) - C\sin(\theta) + F,$$

where $\theta = (\theta_1, \ldots, \theta_N)$ is the vector of angles across all coupled pendula, $M = \mathrm{diag}(\mu^1, \ldots, \mu^N)$ is a diagonal mass matrix, F an external forcing, $K = \mathrm{diag}(k^1, \ldots, k^N)$ the "spring constant" for pendula, given by $k^i = \mu^i g/\ell$ with ℓ the pendulum length and g the gravitational acceleration, and where $C = C^T$ is a symmetric matrix, with

$$C_{\ell\ell} = -\sum_{\ell' \neq \ell} C_{\ell\ell'}, \quad \text{so that} \quad [C\sin(\theta)]_\ell = \sum_{\ell' \neq \ell} C_{\ell\ell'}(\sin(\theta_{\ell'}) - \sin(\theta_\ell)), \tag{4.7}$$

which quantifies the coupling between different pendula. We note that this simplified Frenkel-Kontorova system can also model other coupled nonlinear oscillators, such as coupled lasers or spintronic oscillators Wright et al. [2022].

We can bring the above system into a more familiar form by introducing the variable y according to the relationship $Py = \theta$ for a matrix P. Substitution of this ansatz then yields $MP\ddot{y} = -(K + C)\sin(Py) + F$;

choosing $P = M^{-1}(K + C)$, we find

$$\ddot{y} = -\sin(M^{-1}(K + C)y) + F, \tag{4.8}$$

which can be written in the form $\ddot{y} = \sigma(Wy) + F$ for $\sigma = -\sin(\,\cdot\,)$ and $W = M^{-1}(K + C)$. If we now take C in a block-matrix form

$$C := \begin{bmatrix} \gamma^L I & C^L & 0 & \dots & 0 \\ C^{L,T} & \gamma^{L-1} I & \ddots & & \vdots \\ 0 & \ddots & & \ddots & 0 \\ \vdots & \dots & C^{3,T} & \gamma^2 I & C^2 \\ 0 & \dots & 0 & C^{2,T} & \gamma^1 I \end{bmatrix},$$

and with corresponding mass matrix M in block-matrix form $M = \mathrm{diag}(\mu^L I, \mu^{L-1} I, \dots, \mu^1 I)$, then with $\rho^\ell := \gamma^\ell / \mu^\ell$, we have

$$M^{-1}C := \begin{bmatrix} \rho^L I & C^L/\mu^L & 0 & \dots & 0 \\ C^{L,T}/\mu^{L-1} & \rho^{L-1} I & \ddots & & \vdots \\ 0 & \ddots & & \ddots & 0 \\ \vdots & & \ddots & \rho^2 I & C^2/\mu^2 \\ 0 & \dots & 0 & C^{2,T}/\mu^1 & \rho^1 I \end{bmatrix},$$

Introducing an ordering parameter $\epsilon > 0$, and choosing $\gamma^\ell, C^\ell, \mu^\ell \sim \epsilon^\ell$, it follows that ρ^ℓ, $\frac{C^\ell}{\mu^\ell} = O(1)$, and $\frac{C^\ell}{\mu^{\ell-1}} = O(\epsilon)$. Hence, with a suitable ordering of the masses across the different layers, one can introduce an effective one-way coupling, making

$$M^{-1}C = \begin{bmatrix} \rho^L I & V^L & 0 & \dots & 0 \\ 0 & \rho^{L-1} I & V^{L-1} & \ddots & \vdots \\ \vdots & & \ddots & & 0 \\ 0 & \dots & 0 & \rho^2 I & V^2 \\ 0 & \dots & 0 & 0 & \rho^1 I \end{bmatrix} + O(\epsilon),$$

upper triangular, up to small terms of order ϵ. We note that the diagonal entries ρ^ℓ in $M^{-1}C$ are determined by the off-diagonal terms through the identity (4.7). The additional degrees of freedom in the (diagonal) K-matrix in (4.8) can be used to tune the diagonal weights of the resulting weight matrix $W = M^{-1}(K + C)$.

Thus, physical systems such as the Frankel-Kontorova system of nonlinear oscillators can be approximated (to leading order) by multi-layer systems of the form

$$\ddot{y}^\ell = \sigma\left(w^\ell \odot y^\ell + V^\ell y^{\ell-1}\right) + F^\ell, \tag{4.9}$$

with F^ℓ an external forcing, representing a tunable linear transformation of the external input to the system. The only formal difference between (4.9) and (4.2) is (i) the absence of a bias term in (4.9) and (ii) the fact that the external forcing appears outside of the nonlinear activation function σ in (4.9). A bias term could readily be introduced by measuring the angles represented by y^ℓ in a suitably shifted

reference frame; physically, this corresponds to tuning the initial position $y^\ell(0)$ of the pendula, with $y^\ell(0)$ also serving as the reference value. Furthermore, in our proof of universality for (4.2), it makes very little difference whether the external forcing F is applied inside the activation function, as in (4.2b) resp. (4.1a), or outside as in (4.9); indeed, the first layer in our proof of universality will in fact approximate the *linearized dynamics* of (4.2b), i.e. a forced harmonic oscillator (4.6). Consequently, a universality result for the multi-layer neural oscillator (4.2) also implies universality of variants of nonlinear oscillator-based physical neural networks, such as those considered in Wright et al. [2022].

4.2 Universality of Neural Oscillators

In this section, we concisely state the proof for our main result regarding the universality of neural oscillators (4.1) or, more specifically, multi-layer oscillators (4.2). To this end, we start with some mathematical preliminaries to set the stage for the main theorem.

4.2.1 Setting

Input signal. We want to approximate operators $\Phi : u \mapsto \Phi(u)$, where $u = u(t)$ is a time-dependent input signal over a time-interval $t \in [0, T]$, and $\Phi(u)(t)$ is a time-dependent output signal. We will assume that the input signal $t \mapsto u(t)$ is continuous, and that $u(0) = 0$. To this end, we introduce the space

$$C_0([0, T]; \mathbb{R}^p) := \{u : [0, T] \to \mathbb{R}^p \,|\, t \mapsto u(t) \text{ is continuous and } u(0) = 0\}.$$

We will assume that the underlying operator defines a mapping $\Phi : C_0([0, T]; \mathbb{R}^p) \to C_0([0, T]; \mathbb{R}^q)$.

The approximation we discuss in this work are based on oscillatory systems starting from rest. These oscillators are forced by the input signal u. For such systems the assumption that $u(0) = 0$ is necessary, because the oscillator starting from rest takes a (arbitrarily small) time-interval to synchronize with the input signal (to "warm up"); If $u(0) \neq 0$, then the oscillator cannot accurately approximate the output during this warm-up phase. This intuitive fact is also implicit in our proofs. We will provide a further comment on this issue in Remark 4.2.2, below.

Operators of interest. We consider the approximation of an operator $\Phi : C_0([0, T]; \mathbb{R}^p) \to C_0([0, T]; \mathbb{R}^q)$, mapping a continuous input signal $u(t)$ to a continuous output signal $\Phi(u)(t)$. We will restrict attention to the uniform approximation of Φ over a compact set of input functions $K \subset C_0([0, T]; \mathbb{R}^p)$. We will assume that Φ satisfies the following properties:

- Φ is **causal**: For any $t \in [0, T]$, if $u, v \in C_0([0, T]; \mathbb{R}^p)$ are two input signals, such that $u|_{[0,t]} \equiv v|_{[0,t]}$, then $\Phi(u)(t) = \Phi(v)(t)$, i.e. the value of $\Phi(u)(t)$ at time t does not depend on future values $\{u(\tau) \,|\, \tau > t\}$.

- Φ is **continuous** as an operator

  $$\Phi : (C_0([0, T]; \mathbb{R}^p), \|\cdot\|_{L^\infty}) \to (C_0([0, T]; \mathbb{R}^q), \|\cdot\|_{L^\infty}),$$

  with respect to the L^∞-norm on the input-/output-signals.

Note that the class of Continuous and Causal operators are very general and natural in the contexts of mapping between sequence spaces or time-varying function spaces, see Grigoryeva and Ortega [2018], Gonon et al. [2019] and references therein.

4.2.2 Universal approximation Theorem

The universality of neural oscillators is summarized in the following theorem:

Theorem 4.2.1 (Universality of the multi-layer neural oscillator)**.** *Let $\Phi : C_0([0,T];\mathbb{R}^p) \to C_0([0,T];\mathbb{R}^q)$ be a causal and continuous operator. Let $K \subset C_0([0,T];\mathbb{R}^p)$ be compact. Then for any $\epsilon > 0$, there exist hyperparameters L, $m_1, \dots, m_L$, weights $w^\ell \in \mathbb{R}^{m_\ell}$, $V^\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$, $A \in \mathbb{R}^{q \times m_L}$ and bias vectors $b^\ell \in \mathbb{R}^{m_\ell}$, $c \in \mathbb{R}^q$, for $\ell = 1, \dots, L$, such that the output $z : [0,T] \to \mathbb{R}^q$ of the multi-layer neural oscillator* (4.2) *satisfies*

$$\sup_{t\in[0,T]} |\Phi(u)(t) - z(t)| \leq \epsilon, \quad \forall\, u \in K.$$

It is important to observe that the *sparse, independent* multi-layer neural oscillator (4.2) suffices for universality in the considered class. Thus, there is no need to consider the wider class of neural oscillators (4.1), at least in this respect. We remark in passing that Theorem 4.2.1 immediately implies another universality result for neural oscillators, showing that they can also be used to approximate arbitrary continuous functions $F : \mathbb{R}^p \to \mathbb{R}^q$. This extension is explained in detail in Section 4.3.

Remark 4.2.2. *We note that the theorem can be readily extended to remove the requirement on $u(0) = 0$ and $\Phi(u)(0) = 0$. To this end, let $\Phi : C([0,T];\mathbb{R}^p) \to C([0,T];\mathbb{R}^q)$ be an operator between spaces of continuous functions, $u \mapsto \Phi(u)$ on $[0,T]$. Fix a $t_0 > 0$, and extend any input function $u : [0,T] \to \mathbb{R}^p$ to a function $\mathcal{D}(u) \in C_0([-t_0,T];\mathbb{R}^p)$, by*

$$\mathcal{D}(u)(t) := \begin{cases} \frac{(t_0+t)}{t_0} u(0), & t \in [-t_0, 0), \\ u(t), & t \in [0,T]. \end{cases}$$

Our proof of Theorem 4.2.1 can readily be used to show that the oscillator system with forcing $\mathcal{D}(u)$, and initialized at time $-t_0 < 0$, can uniformly approximate $\Phi(u)$ over the entire time interval $[0,T]$, without requiring that $u(0) = 0$, or $\Phi(u)(0) = 0$. In this case, the initial time interval $[-t_0, 0]$ provides the required "warm-up phase" for the neural oscillator.

Remark 4.2.3. *In practice, neural ODEs such as* (4.2) *need to be discretized via suitable numerical schemes. As examples, CoRNN* (2.3) *and UnICORNN* (3.6) *were implemented with implicit-explicit time discretizations. Nevertheless, universality also applies for such discretizations as long as the time-step is small enough, as the underlying discretization is going to be a sufficiently accurate approximation of* (4.2) *and Theorem 4.2.1 can be used for showing universality of the discretized version of the multi-layer neural oscillator* (4.2)*.*

In the following, we present the proof of the universality Theorem 4.2.1. For a given tolerance ϵ, we will explicitly construct the weights and biases of the multi-layer neural oscillator (4.2) such that the underlying operator can be approximated within the given tolerance. This construction takes place in the following steps:

(Forced) Harmonic Oscillators compute a time-windowed sine transform. Recall that the forced harmonic oscillator (4.6) is the simplest example of a neural oscillator (4.1). The following lemma shows that this forced harmonic oscillator actually computes a time-windowed variant of the sine transform at the corresponding frequency:

Lemma 4.2.4. *Assume that $\omega \neq 0$. Then the solution of* (4.6) *is given by*

$$y(t) = \frac{1}{\omega} \int_0^t u(t-\tau) \sin(\omega\tau)\, d\tau. \tag{4.10}$$

Proof. We can rewrite $y(t) = \frac{1}{\omega}\int_0^t u(\tau)\sin(\omega(t-\tau))\,d\tau$. By direct differentiation, one readily verifies that $y(t)$ so defined, satisfies

$$\dot{y}(t) = \int_0^t u(\tau)\cos(\omega(t-\tau))\,d\tau + [u(\tau)\sin(\omega(t-\tau))]_{\tau=t} = \int_0^t u(\tau)\cos(\omega(t-\tau))\,d\tau,$$

in account of the fact that $\sin(0) = 0$. Differentiating once more, we find that

$$\begin{aligned}\ddot{y}(t) &= -\omega\int_0^t u(\tau)\sin(\omega(t-\tau))\,d\tau + [u(\tau)\cos(\omega(t-\tau))]_{\tau=t} \\ &= -\omega^2 y(t) + u(t).\end{aligned}$$

Thus $y(t)$ solves the ODE (4.6), with initial condition $y(0) = \dot{y}(0) = 0$. □

Given the last result, for a function u, we define its **time-windowed sine transform** as follows,

$$\mathcal{L}_t u(\omega) := \int_0^t u(t-\tau)\sin(\omega\tau)\,d\tau. \tag{4.11}$$

Lemma 4.2.4 shows that a forced harmonic oscillator computes (4.11) up to a constant.

Approximation of causal operators from finite realizations of time-windowed sine transforms. The following novel result, termed the *fundamental Lemma*, shows that the time-windowed sine transform (4.11) composed with a suitable nonlinear function can approximate causal operators Φ to desired accuracy; as a consequence, one can conclude that *forced harmonic oscillators* combined with a *nonlinear read-out* defines a universal architecture in the sense of Theorem 4.2.1.

Lemma 4.2.5 (Fundamental Lemma)**.** *Let $\Phi : K \subset C_0([0,T];\mathbb{R}^p) \to C_0([0,T];\mathbb{R}^q)$ be a causal and continuous operator, with $K \subset C_0([0,T];\mathbb{R}^p)$ compact. Then for any $\epsilon > 0$, there exists $N \in \mathbb{N}$, frequencies $\omega_1, \ldots, \omega_N$ and a continuous mapping $\Psi : \mathbb{R}^{p\times N} \times [0, T^2/4] \to \mathbb{R}^q$, such that*

$$|\Phi(u)(t) - \Psi(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t^2/4)| \leq \epsilon,$$

for all $u \in K$.

We prove this fundamental Lemma in the following. Let $[0,T] \subset \mathbb{R}$ be an interval and $K \subset C_0([0,T];\mathbb{R}^p)$ a fixed compact set. Note that for any $u \in K$, we have $u(0) = 0$, and hence K can be identified with a subset of $C((-\infty, T];\mathbb{R}^p)$, consisting of functions with $\mathrm{supp}(u) \subset [0,T]$. We consider the reconstruction of continuous functions $u \in K$. We will show that u can be approximately reconstructed from knowledge of $\mathcal{L}_t(\omega)$. More precisely, we provide a detailed proof of the following result:

Lemma 4.2.6. *Let $K \subset C((-\infty, T];\mathbb{R}^p)$ be compact, such that $\mathrm{supp}(u) \subset [0,T]$ for all $u \in K$. For any $\epsilon, \Delta t > 0$, there exists $N \in \mathbb{N}$, frequencies $\omega_1, \ldots, \omega_N \in \mathbb{R}\setminus\{0\}$, phase-shifts $\vartheta_1, \ldots, \vartheta_N \in \mathbb{R}$ and weights $\alpha_1, \ldots, \alpha_N \in \mathbb{R}$, such that*

$$\sup_{\tau\in[0,\Delta t]} \left| u(t-\tau) - \sum_{j=1}^{N} \alpha_j \mathcal{L}_t u(\omega_j)\sin(\omega_j\tau - \vartheta_j) \right| \leq \epsilon,$$

for all $u \in K$ and for all $t \in [0,T]$.

Proof. **Step 0: (Equicontinuity)** We recall the following fact from topology. If $K \subset C((-\infty, T]; \mathbb{R}^p)$ is compact, then it is equicontinuous; i.e. there exists a continuous modulus of continuity $\phi : [0, \infty) \to [0, \infty)$ with $\phi(r) \to 0$ as $r \to 0$, such that

$$|u(t-\tau) - u(t)| \leq \phi(\tau), \quad \forall\, \tau \geq 0,\ t \in [0, T],\ \forall\, u \in K. \tag{4.12}$$

Step 1: (Connection to Fourier transform) Fix $t_0 \in [0, T]$ and $u \in K$ for the moment. Define $f(\tau) = u(t_0 - \tau)$. Note that $f \in C([0, \infty); \mathbb{R}^p)$, and f has compact support $\text{supp}(f) \subset [0, T]$. We also note that, by (4.12), we have

$$|f(t+\tau) - f(t)| \leq \phi(\tau), \quad \forall\, \tau \geq 0,\ t \in [0, T].$$

We now consider the following odd extension of f to all of $\mathbb{R}$:

$$F(\tau) := \begin{cases} f(\tau), & \text{for } \tau \geq 0, \\ -f(-\tau), & \text{for } \tau \leq 0. \end{cases}$$

Since F is odd, the Fourier transform of F is given by

$$\hat{F}(\omega) := \int_{-\infty}^{\infty} F(\tau) e^{-i\omega\tau}\, d\tau = i \int_{-\infty}^{\infty} F(\tau) \sin(\omega\tau)\, d\tau = 2i \int_0^T f(\tau) \sin(\omega\tau)\, d\tau = 2i \mathcal{L}_{t_0} u(\omega).$$

Let $\epsilon > 0$ be arbitrary. Our goal is to uniformly approximate $F(\tau)$ on the interval $[0, \Delta t]$. The main complication here is that F lacks regularity (is discontinuous), and hence the inverse Fourier transform of $\hat{F}$ does not converge to F uniformly over this interval; instead, a more careful reconstruction based on mollification of F is needed. We provide the details below.

Step 2: (Mollification) We now fix a smooth, non-negative and compactly supported function $\rho : \mathbb{R} \to \mathbb{R}$, such that $\text{supp}(\rho) \subset [0, 1]$, $\rho \geq 0$, $\int_{\mathbb{R}} \rho(t)\, dt = 1$, and we define a mollifier $\rho_\epsilon(t) := \frac{1}{\epsilon}\rho(t/\epsilon)$. In the following, we will assume throughout that $\epsilon \leq T$. We point out that $\text{supp}(\rho_\epsilon) \subset [0, \epsilon]$, and hence, the mollification $F_\epsilon(t) = (F * \rho_\epsilon)(t)$ satisfies, for $t \geq 0$:

$$\begin{aligned} |F(t) - F_\epsilon(t)| &= \left| \int_0^\epsilon (F(t) - F(t+\tau)) \rho_\epsilon(\tau)\, d\tau \right| = \left| \int_0^\epsilon (f(t) - f(t+\tau)) \rho_\epsilon(\tau)\, d\tau \right| \\ &\leq \left\{ \sup_{\tau \in [0,\epsilon]} |f(t) - f(t+\tau)| \right\} \int_0^\epsilon \rho_\epsilon(\tau)\, d\tau \leq \phi(\epsilon). \end{aligned}$$

In particular, this shows that

$$\sup_{t \in [0,T]} |F(t) - F_\epsilon(t)| \leq \phi(\epsilon),$$

can be made arbitrarily small, with an error that depends only on the modulus of continuity ϕ.

Step 3: (Fourier inverse) Let $\hat{F}_\epsilon(\omega)$ denote the Fourier transform of F_ϵ. Since F_ϵ is smooth and compactly supported, it is well-known that we have the identity

$$F_\epsilon(\tau) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{F}_\epsilon(\omega) e^{-i\omega\tau}\, d\omega, \qquad \forall\, t \in \mathbb{R},$$

where $\omega \mapsto \hat{F}_\epsilon(\omega)$ decays to zero very quickly (almost exponentially) as $|\omega| \to \infty$. In fact, since $F_\epsilon = F * \rho_\epsilon$ is a convolution, we have $\hat{F}_\epsilon(\omega) = \hat{F}(\omega)\hat{\rho}_\epsilon(\omega)$, where $|\hat{F}(\omega)| \leq 2\|f\|_{L^\infty} T$ is uniformly bounded, and $\hat{\rho}_\epsilon(\omega)$ decays quickly. In particular, this implies that there exists a $L = L(\epsilon, T) > 0$ *independent of* f, such that

$$\left| F_\epsilon(\tau) - \frac{1}{2\pi} \int_{-L}^{L} \hat{F}(\omega)\hat{\rho}_\epsilon(\omega) e^{-i\omega\tau}\, d\omega \right| \leq 2T\|f\|_{L^\infty} \int_{|\omega|>L} |\hat{\rho}_\epsilon(\omega)|\, d\omega \leq \|f\|_{L^\infty} \epsilon, \qquad \forall\, \tau \in \mathbb{R}. \tag{4.13}$$

Step 4: (Quadrature) Next, we observe that, since F and ρ_ϵ are compactly supported, their Fourier transform $\omega \mapsto \hat{F}(\omega)\hat{\rho}_\epsilon(\omega)e^{-i\omega\tau}$ is smooth; in fact, for $|\tau| \leq T$, the Lipschitz constant of this mapping can be explicitly estimated by noting that

$$\begin{aligned}\frac{\partial}{\partial\omega}\left[\hat{F}(\omega)\hat{\rho}_\epsilon(\omega)e^{-i\omega\tau}\right] &= \frac{\partial}{\partial\omega}\int_{\mathrm{supp}(F_\epsilon)}(F*\rho_\epsilon)(t)e^{i\omega(t-\tau)}\,dt \\ &= \int_{\mathrm{supp}(F_\epsilon)} i(t-\tau)(F*\rho_\epsilon)(t)e^{i\omega(t-\tau)}\,dt.\end{aligned}$$

We next take absolute values, and note that any t in the support of F_ϵ obeys the bound $|t| \leq T+\epsilon \leq 2T$, while $|\tau| \leq T$ by assumption; it follows that

$$\mathrm{Lip}\left(\omega \mapsto \hat{F}(\omega)\hat{\rho}_\epsilon(\omega)e^{-i\omega\tau}\right) \leq (2T+T)\|F\|_{L^\infty}\,\|\rho_\epsilon\|_{L^1} = 3T\|F\|_{L^\infty}, \quad \forall\,\tau\in[0,T].$$

It thus follows from basic results on quadrature that for an equidistant choice of frequencies $\omega_1 < \cdots < \omega_N$, with spacing $\Delta\omega = 2L/(N-1)$, we have

$$\left|\frac{1}{2\pi}\int_{-L}^{L}\hat{F}(\omega)\hat{\rho}_\epsilon(\omega)e^{-i\omega\tau}\,d\omega - \frac{\Delta\omega}{2\pi}\sum_{j=1}^{N}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}\right| \leq \frac{CL^2\,3T\|F\|_{L^\infty}}{N}, \quad \forall\,\tau\in[0,T],$$

for an absolute constant $C>0$, independent of F, T and N. By choosing N to be even, we can ensure that $\omega_j \neq 0$ for all j. In particular, recalling that $L = L(T,\epsilon)$ depends only on ϵ and T, and choosing $N = N(T,\epsilon)$ sufficiently large, we can combine the above estimate with (4.13) to ensure that

$$\left|F_\epsilon(\tau) - \frac{\Delta\omega}{2\pi}\sum_{j=1}^{N}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}\right| \leq 2\|f\|_{L^\infty}\epsilon, \quad \forall\,\tau\in[0,T],$$

where we have taken into account that $\|F\|_{L^\infty} = \|f\|_{L^\infty}$.

Step 5: (Conclusion) To conclude the proof, we recall that $\hat{F}(\omega) = 2i\mathcal{L}_{t_0}u(\omega)$ can be expressed in terms of the sine transform $\mathcal{L}_t u$ of the function u which was fixed at the beginning of Step 1. Recall also that $f(\tau) = u(t_0-\tau)$, so that $\|f\|_{L^\infty} = \|u\|_{L^\infty}$. Hence, we can write the real part of $\frac{\Delta\omega}{2\pi}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau} = \frac{\Delta\omega}{2\pi}2i\mathcal{L}_{t_0}u(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}$, in the form $\alpha_j\mathcal{L}_{t_0}(\omega_j)\sin(\omega_j\tau-\vartheta_j)$ for coefficients $\alpha_j\in\mathbb{R}$ and $\theta_j\in\mathbb{R}$ which depend only on $\Delta\omega$ and $\hat{\rho}_\epsilon(\omega_j)$, but are independent of u. In particular, it follows that

$$\begin{aligned}\sup_{\tau\in[0,\Delta t]}\left|u(t_0-\tau) - \sum_{j=1}^{N}\alpha_j\mathcal{L}_{t_0}u(\omega_j)\sin(\omega_j\tau-\vartheta_j)\right| &= \sup_{t\in[0,\Delta t]}\left|F(\tau) - \Re\left(\frac{\Delta\omega}{2\pi}\sum_{j=1}^{N}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}\right)\right| \\ &\leq \sup_{\tau\in[0,\Delta t]}\left|F(\tau) - \frac{\Delta\omega}{2\pi}\sum_{j=1}^{N}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}\right| \\ &\leq \sup_{\tau\in[0,\Delta t]}|F(\tau)-F_\epsilon(\tau)| \\ &\quad + \sup_{\tau\in[0,\Delta t]}\left|F_\epsilon(\tau) - \frac{\Delta\omega}{2\pi}\sum_{j=1}^{N}\hat{F}(\omega_j)\hat{\rho}_\epsilon(\omega_j)e^{-i\omega_j\tau}\right|.\end{aligned}$$

By Steps 1 and 3, the first term on the right-hand side is bounded by $\leq \phi(\epsilon)$, while the second one is bounded by $\leq 2\sup_{u\in K}\|u\|_{L^\infty}\epsilon \leq C\epsilon$, where $C = C(K) < \infty$ depends only on the compact set

$K \subset C([0,T];\mathbb{R}^p)$. Hence, we have

$$\sup_{\tau\in[0,\Delta t]} \left| u(t_0-\tau) - \sum_{j=1}^{N} \alpha_j \mathcal{L}_{t_0} u(\omega_j) \sin(\omega_j \tau - \vartheta_j) \right| \leq \phi(\epsilon) + C\epsilon.$$

In this estimate, the function $u \in K$ and $t_0 \in [0,T]$ were arbitrary, and the modulus of continuity ϕ as well as the constant C on the right-hand side depend only on the set K. it thus follows that for this choice of α_j, ω_j and ϑ_j, we have

$$\sup_{u\in K} \sup_{t\in[0,T]} \sup_{\tau\in[0,\Delta t]} \left| u(t-\tau) - \sum_{j=1}^{N} \alpha_j \mathcal{L}_t u(\omega_j) \sin(\omega_j \tau - \vartheta_j) \right| \leq \phi(\epsilon) + C\epsilon.$$

Since $\epsilon > 0$ was arbitrary, the right-hand side can be made arbitrarily small. The claim then readily follows. □

The next step in the proof of the fundamental Lemma 4.2.5 needs the following preliminary result in functional analysis,

Lemma 4.2.7. *Let $\mathcal{X}, \mathcal{Y}$ be Banach spaces, and let $K \subset \mathcal{X}$ be a compact subset. Assume that $\Phi : \mathcal{X} \to \mathcal{Y}$ is continous. Then for any $\epsilon > 0$, there exists a $\delta > 0$, such that if $\|u - u^K\|_{\mathcal{X}} \leq \delta$ with $u \in \mathcal{X}$, $u^K \in K$, then $\|\Phi(u) - \Phi(u^K)\|_{\mathcal{Y}} \leq \epsilon$.*

Proof. Suppose not. Then there exists $\epsilon_0 > 0$ and a sequence u_j, u_j^K, $(j \in \mathbb{N})$, such that $\|u_j - u_j^K\|_{\mathcal{X}} \leq j^{-1}$, while $\|\Phi(u_j) - \Phi(u_j^K)\|_{\mathcal{Y}} \geq \epsilon_0$. By the compactness of K, we can extract a subsequence $j_k \to \infty$, such that $u_{j_k}^K \to u^K$ converges to some $u^K \in K$. By assumption on u_j, this implies that

$$\|u_{j_k} - u^K\|_{\mathcal{X}} \leq \|u_{j_k} - u_{j_k}^K\|_{\mathcal{X}} + \|u_{j_k}^K - u^K\|_{\mathcal{X}} \overset{(k\to\infty)}{\longrightarrow} 0,$$

which, by the assumed continuity of Φ, leads to the contradiction that $0 < \epsilon_0 \leq \|\Phi(u_{j_k}) - \Phi(u^K)\|_{\mathcal{Y}} \to 0$, as $k \to \infty$. □

Now, we can prove the fundamental Lemma in the following,

Proof. Let $\epsilon > 0$ be given. We can identify $K \subset C_0([0,T];\mathbb{R}^p)$ with a compact subset of $C((-\infty,T];\mathbb{R}^p)$, by extending all $u \in K$ by zero for negative times, i.e. we set $u(t) = 0$ for $t < 0$. Applying Lemma 4.2.7, with $\mathcal{X} = C_0([0,T];\mathbb{R}^p)$ and $\mathcal{Y} = C_0([0,T];\mathbb{R}^q)$, we can find a $\delta > 0$, such that for any $u \in C_0([0,T];\mathbb{R}^p)$ and $u^K \in K$, we have

$$\|u - u^K\|_{L^\infty} \leq \delta \quad \Rightarrow \quad \|\Phi(u) - \Phi(u^K)\|_{L^\infty} \leq \epsilon. \tag{4.14}$$

By the inverse sine transform Lemma 4.2.6, there exist $N \in \mathbb{N}$, frequencies $\omega_1, \ldots, \omega_N \neq 0$, phase-shifts $\vartheta_1, \ldots, \vartheta_N$ and coefficients $\alpha_1, \ldots, \alpha_N$, such that for any $u \in K$ and $t \in [0,T]$:

$$\sup_{\tau\in[0,T]} \left| u(t-\tau) - \sum_{j=1}^{N} \alpha_j \, \mathcal{L}_t u(\omega_j) \sin(\omega_j \tau - \vartheta_j) \right| \leq \delta.$$

Given $\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N)$, we can thus define a reconstruction mapping $\mathcal{R} : \mathbb{R}^N \times [0,T] \to C([0,T];\mathbb{R}^p)$ by

$$\mathcal{R}(\beta_1, \ldots, \beta_N; t)(\tau) := \sum_{j=1}^{N} \alpha_j \beta_j \sin(\omega_j (t-\tau) - \vartheta_j).$$

Then, for $\tau \in [0,t]$, we have

$$|u(\tau) - \mathcal{R}(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t)(\tau)| \leq \delta.$$

We can now uniquely define $\Psi : \mathbb{R}^N \times [0, T^2/4] \to C_0([0,T]; \mathbb{R}^p)$, by the identity

$$\Psi(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t^2/4) = \Phi\left(\mathcal{R}(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t)\right).$$

Using the short-hand notation $\mathcal{R}_t u = \mathcal{R}(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t)$, we have $\sup_{\tau \in [0,t]} |u(\tau) - \mathcal{R}_t u(\tau)| \leq \delta$, for all $t \in [0,T]$. By (4.14), this implies that

$$\left|\Phi(u)(t) - \Psi(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t^2/4)\right| = |\Phi(u)(t) - \Phi(\mathcal{R}_t u)(t)| \leq \epsilon.$$

□

Given these two results, we can discern a clear strategy to prove the universality Theorem 4.2.1. First, we will show that a general *nonlinear* form of the neural oscillator (4.2) can also compute the time-windowed sine transform at arbitrary frequencies. Then, these outputs need to be processed in order to apply the fundamental Lemma 4.2.5 and approximate the underlying operator Φ. To this end, we will also approximate the function Ψ (mapping finite-dimensional inputs to finite-dimensional outputs) by oscillatory layers. The concrete steps in this strategy are outlined below.

Nonlinear Oscillators approximate the time-windowed sine transform. The building block of multi-layer neural oscillators (4.2) is the nonlinear oscillator of the form,

$$\ddot{y} = \sigma(w \odot y + Vu + b). \tag{4.15}$$

In the following Lemma, we show that even for a nonlinear activation function σ such as tanh or sin, the nonlinear oscillator (4.15) can approximate the time-windowed sine transform.

Lemma 4.2.8. *Fix $\omega \neq 0$. Assume that $\sigma(0) = 0$, $\sigma'(0) = 1$. For any $\epsilon > 0$, there exist $w, V, b, A \in \mathbb{R}$, such that the nonlinear oscillator* (4.15)*, initialized at $y(0) = \dot{y}(0) = 0$, has output*

$$|Ay(t) - \mathcal{L}_t u(\omega)| \leq \epsilon, \qquad \forall u \in K,\ t \in [0,T],$$

with $\mathcal{L}_t u(\omega)$ being the time-windowed sine transform (4.11)*.*

Proof. Let $\omega \neq 0$ be given. For a (small) parameter $s > 0$, we consider

$$\ddot{y}_s = \frac{1}{s}\sigma(-s\omega^2 y_s + su), \quad y_s(0) = \dot{y}_s(0) = 0.$$

Let Y be the solution of

$$\ddot{Y} = -\omega^2 Y + u, \quad Y(0) = \dot{Y}(0) = 0.$$

Then we have, on account of $\sigma(0) = 0$ and $\sigma'(0) = 1$,

$$\begin{aligned} s^{-1}\sigma(-s\omega^2 Y + su) - [-\omega^2 Y + u] &= \frac{\sigma(-s\omega^2 Y + su) - \sigma(0)}{s} - \sigma'(0)[-\omega^2 Y + u] \\ &= \frac{1}{s}\int_0^s \frac{\partial}{\partial \zeta}\left[\sigma(-\zeta\omega^2 Y + \zeta u)\right] d\zeta - \sigma'(0)[-\omega^2 Y + u] \\ &= \frac{1}{s}\left(\int_0^s \left[\sigma'(-\zeta\omega^2 Y + \zeta u) - \sigma'(0)\right] d\zeta\right)\left[-\omega^2 Y + u\right]. \end{aligned}$$

It follows from Lemma 4.2.4 that for any input $u \in K$, with $\sup_{u\in K} \|u\|_{L^\infty} =: B < \infty$, we have a uniform bound $\|Y\|_{L^\infty} \leq BT/\omega$, hence we can estimate

$$|-\omega^2 Y + u| \leq B(\omega T + 1),$$

uniformly for all such u. In particular, it follows that

$$\left|s^{-1}\sigma(-s\omega^2 Y + su) - [-\omega^2 Y + u]\right| \leq B(T\omega + 1) \sup_{|x|\leq sB(T\omega+1)} |\sigma'(x) - \sigma'(0)|.$$

Clearly, for any $\delta > 0$, we can choose $s \in (0, 1]$ sufficiently small, such that the right hand-side is bounded by δ, i.e. with this choice of s,

$$\left|s^{-1}\sigma(-s\omega^2 Y(t) + su(t)) - [-\omega^2 Y(t) + u(t)]\right| \leq \delta, \quad \forall\, t \in [0, T],$$

holds for any choice of $u \in K$. We will fix this choice of s in the following, and write $g(y, u) := s^{-1}\sigma(-s\omega^2 y + su)$. We note that g is Lipschitz continuous in y, for all $|y| \leq BT/\omega$ and $|u| \leq B$, with $\mathrm{Lip}_y(g) \leq \omega^2 \sup_{|\xi|\leq B(\omega T+1)} |\sigma'(\xi)|$.

To summarize, we have shown that Y solves

$$\ddot{Y} = g(Y, u) + f, \qquad Y(0) = \dot{Y}(0) = 0,$$

where $\|f\|_{L^\infty} \leq \delta$. By definition, y_s solves

$$\ddot{y}_s = g(y_s, u), \qquad y_s(0) = \dot{y}_s(0) = 0.$$

It follows from this that

$$\begin{aligned}
|y_s(t) - Y(t)| &\leq \int_0^t \int_0^\tau \{|g(y_s(\theta), u(\theta)) - g(Y(\theta), u(\theta))| + |f(\theta)|\}\, d\theta\, d\tau \\
&\leq \int_0^t \int_0^\tau \left\{\mathrm{Lip}_y(g)|y_s(\theta) - Y(\theta)| + \delta\right\} d\theta\, d\tau \\
&\leq T\omega^2 \sup_{|\xi|\leq B(\omega T+1)} |\sigma'(\xi)| \int_0^t |y_s(\tau) - Y(\tau)|\, d\tau + T^2\delta.
\end{aligned}$$

Recalling that $Y(t) = \mathcal{L}_t u(\omega)$, then by Gronwall's inequality, the last estimate implies that

$$\sup_{t\in[0,T]} |y_s(t) - \mathcal{L}_t u(\omega)| = \sup_{t\in[0,T]} |y_s - Y| \leq C\delta,$$

for a constant $C = C(T, \omega, \sup_{|\xi|\leq B(\omega T+1)} |\sigma'(\xi)|) > 0$, depending only on T, ω, B and σ'. Since $\delta > 0$ was arbitrary, we can ensure that $C\delta \leq \epsilon$. Thus, we have shown that a suitably rescaled nonlinear oscillator approximates the harmonic oscillator to any desired degree of accuracy, and uniformly for all $u \in K$.

To finish the proof, we observe that y solves

$$\ddot{y} = \sigma(-\omega^2 y + su), \qquad y(0) = \dot{y}(0) = 0,$$

if, and only if, $y_s = y/s$ solves

$$\ddot{y}_s = s^{-1}\sigma(-s\omega^2 y_s + su), \qquad y_s(0) = \dot{y}_s(0) = 0.$$

Hence, with $W = -\omega^2$, $V = s$, $b = 0$ and $A = s^{-1}$, we have

$$\sup_{t\in[0,T]} |Ay(t) - \mathcal{L}_t u(\omega)| = \sup_{t\in[0,T]} |y_s(t) - \mathcal{L}_t u(\omega)| \leq \epsilon.$$

This concludes the proof. □

Coupled Nonlinear Oscillators approximate time-delays. The next step in the proof is to show that coupled oscillators can approximate time-delays in the continuous input signal. This fact will be of crucial importance in subsequent arguments. We have the following Lemma,

Lemma 4.2.9. *Let $K \subset C_0([0,T];\mathbb{R}^p)$ be a compact subset. For every $\epsilon > 0$, and $\Delta t \geq 0$, there exist $m \in \mathbb{N}$, $w \in \mathbb{R}^m$, $V \in \mathbb{R}^{m\times p}$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{p\times m}$, such that the oscillator* (4.15)*, initialized at $y(0) = \dot{y}(0) = 0$, has output*

$$\sup_{t\in[0,T]} |u(t-\Delta t) - Ay(t)| \leq \epsilon, \quad \forall\, u \in K,$$

where $u(t)$ is extended to negative values $t < 0$ by zero.

Proof. Let $\epsilon, \Delta t$ be given. By the sine transform reconstruction Lemma 4.2.6, there exists $N \in \mathbb{N}$, frequencies $\omega_1, \ldots, \omega_N$, weights $\alpha_1, \ldots, \alpha_N$ and phase-shifts $\vartheta_1, \ldots, \vartheta_N$, such that

$$\sup_{\tau\in[0,\Delta t]} \left| u(t-\tau) - \sum_{j=1}^{N} \alpha_j \mathcal{L}_t u(\omega_j) \sin(\omega_j \tau - \vartheta_j) \right| \leq \frac{\epsilon}{2}, \quad \forall\, t \in [0,T],\ \forall\, u \in K, \tag{4.16}$$

where any $u \in K$ is extended by zero to negative times. It follows from Lemma 4.2.8, that there exists a coupled oscillator network,

$$\ddot{y} = \sigma(w \odot y + Vu + b), \qquad y(0) = \dot{y}(0) = 0,$$

with dimension $m = pN$, and $w \in \mathbb{R}^m$, $V \in \mathbb{R}^{m\times p}$, and a linear output layer $y \mapsto \tilde{A}y$, $\tilde{A} \in \mathbb{R}^{m\times m}$, such that $[\tilde{A}y(t)]_j \approx \mathcal{L}_t u(\omega_j)$ for $j = 1, \ldots, N$; more precisely, such that

$$\sup_{t\in[0,T]} \sum_{j=1}^{N} |\alpha_j| \left| \mathcal{L}_t u(\omega_j) - [\tilde{A}y]_j(t) \right| \leq \frac{\epsilon}{2}, \quad \forall\, u \in K. \tag{4.17}$$

Composing with another linear layer $B : \mathbb{R}^m \simeq \mathbb{R}^{p\times N} \to \mathbb{R}^p$, which maps $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_N]$ to

$$B\boldsymbol{\beta} := \sum_{j=1}^{N} \alpha_j \beta_j \sin(\omega_j \Delta t - \vartheta_j) \in \mathbb{R}^p,$$

we define $A := B\tilde{A}$, and observe that from (4.16) and (4.17):

$$\begin{aligned}
\sup_{t\in[0,T]} |u(t-\Delta t) - Ay(t)| \leq & \sup_{t\in[0,T]} \left| u(t-\Delta t) - \sum_{j=1}^{N} \alpha_j \mathcal{L}_t u(\omega_j) \sin(\omega_j \Delta t - \vartheta_j) \right| \\
& + \sup_{t\in[0,T]} \sum_{j=1}^{N} |\alpha_j| \left| \mathcal{L}_t u(\omega_j) - [\tilde{A}y]_j(t) \right| |\sin(\omega_j \Delta t - \vartheta_j)| \\
\leq & \ \epsilon.
\end{aligned}$$

□

Two-layer neural oscillators approximate neural networks pointwise. As in the strategy outlined above, the final ingredient in our proof of the universality theorem 4.2.1 is to show that neural oscillators can approximate continuous functions, such as the Ψ in the fundamental lemma 4.2.5, to desired accuracy. To this end, we will first show that neural oscillators can approximate general neural networks (perceptrons) and then use the universality of neural networks in the class of continuous functions to prove the desired result. We have the following lemma,

Lemma 4.2.10. *Let $K \subset C_0([0,T];\mathbb{R}^p)$ be compact. For matrices Σ, Λ and bias γ, and any $\epsilon > 0$, there exists a two-layer ($L = 2$) oscillator (4.2), initialized at $y^\ell(0) = \dot{y}^\ell(0) = 0$, $\ell = 1, 2$, such that*

$$\sup_{t\in[0,T]} \left|\left[Ay^2(t) + c\right] - \Sigma\sigma(\Lambda u(t) + \gamma)\right| \leq \epsilon, \quad \forall u \in K.$$

Proof. Fix Σ, Λ, γ as in the statement of the lemma. Our goal is to approximate $u \mapsto \Sigma\sigma(\Lambda u + \gamma)$.

Step 1: (nonlinear layer) We consider a first layer for a hidden state $y = [y_1, y_2]^T \in \mathbb{R}^{p+p}$, given by

$$\left.\begin{cases} \ddot{y}_1(t) = \sigma(\Lambda u(t) + \gamma) \\ \ddot{y}_2(t) = \sigma(\gamma) \end{cases}\right\}, \quad y(0) = \dot{y}(0) = 0.$$

This layer evidently does not approximate $\sigma(\Lambda u(t) + \gamma)$; however, it does encode this value in the second derivative of the hidden variable y_1. The main objective of the following analysis is to approximately compute $\ddot{y}_1(t)$ through a suitably defined additional layer.

Step 2: (Second-derivative layer) To obtain an approximation of $\sigma(\Lambda u(t) + \gamma)$, we first note that the solution operator

$$\mathcal{S} : u(t) \mapsto \eta(t), \quad \text{where } \ddot{\eta}(t) = \sigma(\Lambda u(t) + \gamma) - \sigma(\gamma), \quad \eta(0) = \dot{\eta}(0) = 0,$$

defines a continuous mapping $\mathcal{S} : C_0([0,T];\mathbb{R}^p) \to C_0^2([0,T];\mathbb{R}^p)$, with $\eta(0) = \dot{\eta}(0) = \ddot{\eta}(0) = 0$. Note that η is very closely related to y_1. The fact that $\ddot{\eta} = 0$ is important to us, because it allows us to *smoothly* extend η to negative times by setting $\eta(t) := 0$ for $t < 0$ (which would not be true for $y_1(t)$). The resulting extension defines a compactly supported function $\eta : (-\infty, 0] \to \mathbb{R}^p$, with $\eta \in C^2((-\infty, T];\mathbb{R}^p)$. Furthermore, by continuity of the operator $\mathcal{S}$, the image $\mathcal{S}(K)$ of the compact set K under $\mathcal{S}$ is compact in $C^2((-\infty, T];\mathbb{R}^p)$. From this, it follows that for small $\Delta t > 0$, the second-order backward finite difference formula converges,

$$\sup_{t\in[0,T]} \left|\frac{\eta(t) - 2\eta(t-\Delta t) + \eta(t-2\Delta t)}{\Delta t^2} - \ddot{\eta}(t)\right| = o_{\Delta t\to 0}(1), \quad \forall \eta = \mathcal{S}(u),\, u \in K,$$

where the bound on the right-hand side is uniform in $u \in K$, due to equicontinuity of $\{\ddot{\eta} \,|\, \eta = \mathcal{S}(u),\, u \in K\}$. In particular, the second derivative of η can be approximated through *linear combinations of time-delays of η*. We can now choose $\Delta t > 0$ sufficiently small so that

$$\sup_{t\in[0,T]} \left|\frac{\eta(t) - 2\eta(t-\Delta t) + \eta(t-2\Delta t)}{\Delta t^2} - \ddot{\eta}(t)\right| \leq \frac{\epsilon}{2\|\Sigma\|}, \quad \forall y = \mathcal{S}(u),\, u \in K,$$

where $\|\Sigma\|$ denotes the operator norm of the matrix Σ. By Lemma 4.2.9, applied to the input set $\tilde{K} = \mathcal{S}(K) \subset C_0([0,T];\mathbb{R}^p)$, there exists a coupled oscillator

$$\ddot{z}(t) = \sigma(w \odot z(t) + V\eta(t) + b), \quad z(0) = \dot{z}(0) = 0, \tag{4.18}$$

and a linear output layer $z \mapsto \tilde{A}z$, such that

$$\sup_{t\in[0,T]} \Big| [\eta(t) - 2\eta(t-\Delta t) + \eta(t-2\Delta t)] - \tilde{A}z(t) \Big| \leq \frac{\epsilon \Delta t^2}{2\|\Sigma\|}, \quad \forall \eta = \mathcal{S}(u),\, u \in K.$$

Indeed, Lemma 4.2.9 shows that time-delays of any given input signal can be approximated with any desired accuracy, and $\eta(t) - 2\eta(t-\Delta) - \eta(t-2\Delta)$ is simply a linear combination of time-delays of the input signal η in (4.18).

To connect $\eta(t)$ back to the $y(t) = [y_1(t), y_2(t)]^T$ constructed in Step 1, we note that

$$\ddot{\eta} = \sigma(Au(t) + b) - \sigma(b) = \ddot{y}_1 - \ddot{y}_2,$$

and hence, taking into account the initial values, we must have $\eta \equiv y_1 - y_2$ by ODE uniqueness. In particular, upon defining a matrix $\tilde{V}$ such that $\tilde{V}y := Vy_1 - Vy_2 \equiv V\eta$, we can equivalently write (4.18) in the form,

$$\ddot{z}(t) = \sigma(w \odot z(t) + \tilde{V}y(t) + b), \quad z(0) = \dot{z}(0) = 0. \tag{4.19}$$

Step 3: (Conclusion)

Composing the layers from Step 1 and 2, we obtain a coupled oscillator

$$\ddot{y}^\ell = \sigma(w^\ell \odot y^\ell + V^\ell y^{\ell-1} + b^\ell), \quad (\ell = 1, 2),$$

initialized at rest, with $y^1 = y$, $y^2 = z$, such that for $A := \Sigma\tilde{A}$ and $c := \Sigma\sigma(\gamma)$, we obtain

$$\begin{aligned}
\sup_{t\in[0,T]} \big| \big[Ay^2(t) + c\big] - \Sigma\sigma(\Lambda u(t) + \gamma) \big| &\leq \|\Sigma\| \sup_{t\in[0,T]} \Big| \tilde{A}z(t) - [\sigma(\Lambda u(t) + \gamma) - \sigma(\gamma)] \Big| \\
&= \|\Sigma\| \sup_{t\in[0,T]} \Big| \tilde{A}z(t) - \ddot{\eta}(t) \Big| \\
&\leq \|\Sigma\| \sup_{t\in[0,T]} \left| \tilde{A}z(t) - \frac{\eta(t) - 2\eta(t-\Delta t) + \eta(t-2\Delta t)}{\Delta t^2} \right| \\
&\quad + \|\Sigma\| \sup_{t\in[0,T]} \left| \frac{\eta(t) - 2\eta(t-\Delta t) + \eta(t-2\Delta t)}{\Delta t^2} - \ddot{\eta}(t) \right| \\
&\leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.
\end{aligned}$$

This concludes the proof. □

As can be seen, the proof is constructive and the neural oscillator that we construct has two layers. The first layer just processes a nonlinear input function through a nonlinear oscillator and the second layer, approximates the second-derivative (in time) from time-delayed versions of the input signal that were constructed in Lemma 4.2.9.

Combining the ingredients to prove the universality theorem 4.2.1. We can now combine the afore-constructed ingredients to prove the universality theorem 4.2.1.

Proof. **Step 1:** By the Fundamental Lemma 4.2.5, there exist N, a continuous mapping Ψ, and frequencies $\omega_1, \ldots, \omega_N$, such that

$$|\Phi(u)(t) - \Psi(\mathcal{L}_t u(\omega_1), \ldots, \mathcal{L}_t u(\omega_N); t^2/4)| \leq \epsilon,$$

for all $u \in K$, and $t \in [0, T]$. Let M be a constant such that

$$|\mathcal{L}_t u(\omega_1)|, \ldots, |\mathcal{L}_t u(\omega_N)|, \frac{t^2}{4} \leq M,$$

for all $u \in K$ and $t \in [0,T]$. By the universal approximation theorem for ordinary neural networks, there exist weight matrices Σ, Λ and bias γ, such that

$$|\Psi(\beta_1, \dots, \beta_N; t^2/4) - \Sigma\sigma(\Lambda\boldsymbol{\beta} + \gamma)| \leq \epsilon, \quad \boldsymbol{\beta} := [\beta_1, \dots, \beta_N; t^2/4]^T,$$

holds for all $t \in [0,T]$, $|\beta_1|, \dots, |\beta_N| \leq M$.

Step 2: Fix $\epsilon_1 \leq 1$ sufficiently small, such that also $\|\Sigma\|\|\Lambda\|\mathrm{Lip}(\sigma)\epsilon_1 \leq \epsilon$, where $\mathrm{Lip}(\sigma) := \sup_{|\xi| \leq \|\Lambda\|M + |\gamma| + 1} |\sigma'(\xi)|$ denotes an upper bound on the Lipschitz constant of the activation function over the relevant range of input values. It follows from Lemma 4.2.8, that there exists an oscillator network,

$$\ddot{y}^1 = \sigma(w^1 \odot y^1 + V^1 u + b^1), \quad y^1(0) = \dot{y}^1(0) = 0, \tag{4.20}$$

of depth 1, such that

$$\sup_{t \in [0,T]} |[\mathcal{L}_t u(\omega_1), \dots, \mathcal{L}_t u(\omega_N); t^2/4]^T - A^1 y^1(t)| \leq \epsilon_1,$$

for all $u \in K$.

Step 3: Finally, by Lemma 4.2.10, there exists an oscillator network,

$$\ddot{y}^2 = \sigma(w^2 \odot y^2 + V^2 y^1 + b^1),$$

of depth 2, such that

$$\sup_{t \in [0,T]} |A^2 y^2(t) - \Sigma\sigma(\Lambda A^1 y^1(t) + \gamma)| \leq \epsilon,$$

holds for all y^1 belonging to the compact set $K_1 := \mathcal{S}(K) \subset C_0([0,T]; \mathbb{R}^{N+1})$, where $\mathcal{S}$ denotes the solution operator of (4.20).

Step 4: Thus, we have for any $u \in K$, and with short-hand $\mathcal{L}_t u(\boldsymbol{\omega}) := (\mathcal{L}_t u(\omega_1), \dots, \mathcal{L}_t u(\omega_N))$,

$$\begin{aligned}
\left|\Phi(u)(t) - A^2 y^2(t)\right| \leq{}& \left|\Phi(u)(t) - \Psi(\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4)\right| \\
&+ \left|\Psi(\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4) - \Sigma\sigma(\Lambda[\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4] + \gamma)\right| \\
&+ \left|\Sigma\sigma(\Lambda[\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4] + \gamma) - \Sigma\sigma(\Lambda A^1 y^1(t) + \gamma)\right| \\
&+ \left|\Sigma\sigma(\Lambda A_1 y_1(t) + \gamma) - A^2 y^2(t)\right|.
\end{aligned}$$

By step 1, we can estimate

$$\left|\Phi(u)(t) - \Psi(\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4)\right| \leq \epsilon, \quad \forall\, t \in [0,T],\ u \in K.$$

By the choice of Σ, Λ, γ, we have

$$\left|\Psi(\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4) - \Sigma\sigma(\Lambda[\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4] + \gamma)\right| \leq \epsilon, \quad \forall\, t \in [0,T],\ u \in K.$$

By construction of y^1 in Step 2, we have

$$\begin{aligned}
&\left|\Sigma\sigma(\Lambda[\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4] + \gamma) - \Sigma\sigma(\Lambda A_1 y_1(t) + \gamma)\right| \\
&\qquad \leq \|\Sigma\|\mathrm{Lip}(\sigma)\|\Lambda\| \left|[\mathcal{L}_t u(\boldsymbol{\omega}); t^2/4] - A^1 y^1(t)\right| \\
&\qquad \leq \|\Sigma\|\mathrm{Lip}(\sigma)\|\Lambda\|\, \epsilon_1 \\
&\qquad \leq \epsilon,
\end{aligned}$$

for all $t \in [0,T]$ and $u \in K$. By construction of y^2 in Step 3, we have

$$\left|\Sigma\sigma(\Lambda A^1 y^1(t) + \gamma) - A^2 y^2(t)\right| \leq \epsilon, \quad \forall\, t \in [0,T],\ u \in K.$$

Thus, we conclude that

$$|\Phi(u)(t) - A^2 y^2(t)| \leq 4\epsilon,$$

for all $t \in [0,T]$ and $u \in K$. Since $\epsilon > 0$ was arbitrary, we conclude that for any causal and continuous operator $\Phi : C_0([0,T];\mathbb{R}^p) \to C_0([0,T];\mathbb{R}^q)$, compact set $K \subset C_0([0,T];\mathbb{R}^p)$ and $\epsilon > 0$, there exists a coupled oscillator of depth 3, which uniformly approximates Φ to accuracy ϵ for all $u \in K$. This completes the proof. □

To summarize the proof of the universality theorem 4.2.1, we explicitly construct a *three-layer* neural oscillator (4.2) which approximates the underlying operator Φ. The first layer follows the construction of Lemma 4.2.8, to approximate the time-windowed sine transform (4.11), for as many frequencies as are required in the fundamental Lemma 4.2.5. The second- and third-layers imitate the construction of Lemma 4.2.10 to approximate a neural network (perception), which in turn by the universal approximation of neural networks, approximates the function Ψ in Lemma 4.2.5 to desired accuracy. Putting the network together leads to a three-layer oscillator that approximates the continuous and casual operator Φ. This construction is depicted in Figure 4.1.

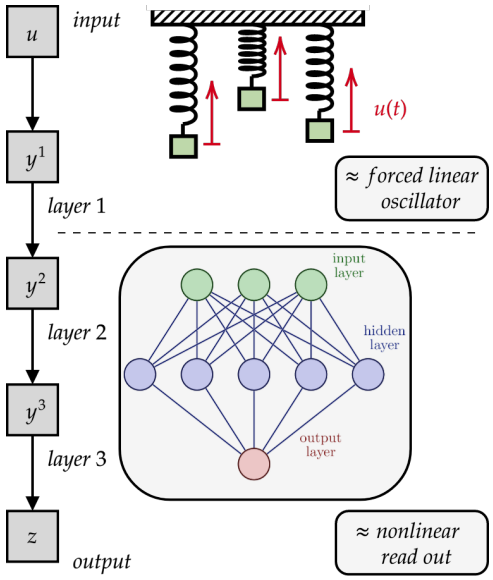

Figure 4.1: Illustration of the universal 3-layer neural oscillator architecture constructed in the proof of Theorem 4.2.1.

4.3 Another universality result for neural oscillators

The universal approximation Theorem 4.2.1 immediately implies another universal approximation results for neural oscillators, as explained next. We consider a continuous map $F : \mathbb{R}^p \to \mathbb{R}^q$; our goal is to show that F can be approximated to given accuracy ϵ by suitably defined neural oscillators. Fix a time interval $[0,T]$ for (an arbitrary choice) $T = 2$. Let $K_0 \subset \mathbb{R}^p$ be a compact set. Given $\xi \in \mathbb{R}^p$, we associate with it a function $u_\xi(t) \in C_0([0,T];\mathbb{R}^p)$, by setting

$$u_\xi(t) := t\xi. \tag{4.21}$$

Clearly, the set $K := \{u_\xi \,|\, \xi \in K_0\}$ is compact in $C_0([0,T];\mathbb{R}^p)$. Furthermore, we can define an operator $\Phi : C_0([0,T];\mathbb{R}^p) \to C_0([0,T];\mathbb{R}^q)$, by

$$\Phi(u)(t) := \begin{cases} 0, & t \in [0,1), \\ (t-1)F(u(1)), & t \in [1,T]. \end{cases} \tag{4.22}$$

where $F : \mathbb{R}^p \to \mathbb{R}^q$ is the given continuous function that we wish to approximate. One readily checks that Φ defines a causal and continuous operator. Note, in particular, that

$$\Phi(u_\xi)(T) = (T-1)F(u_\xi(1)) = F(\xi),$$

is just the evaluation of F at ξ, for any $\xi \in K_0$.

Since neural oscillators can uniformly approximate the operator Φ for inputs $u_\xi \in K$, then as a consequence of Theorem 4.2.1 and (4.3), it follows that, for any $\epsilon > 0$ there exists $m \in \mathbb{N}$, matrices $W \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{m\times p}$ and $A \in \mathbb{R}^{q\times m}$, and bias vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^q$, such that for any $\xi \in K_0$, the neural oscillator system,

$$\begin{cases} \ddot{y}_\xi(t) = \sigma\left(W y_\xi(t) + tV\xi + b\right), & (4.23) \\ y_\xi(0) = \dot{y}_\xi(0) = 0, & (4.24) \\ z_\xi(t) = A y_\xi(t) + c, & (4.25) \end{cases}$$

satisfies

$$|z_\xi(T) - F(\xi)| = |z_\xi(T) - \Phi(u_\xi)(T)| \leq \sup_{t\in[0,T]} |z_\xi(t) - \Phi(u_\xi)(t)| \leq \epsilon,$$

uniformly for all $\xi \in K_0$. Hence, neural oscillators can be used to approximate an arbitrary continuous function $F : \mathbb{R}^p \to \mathbb{R}^q$, uniformly over compact sets. Thus, neural oscillators also provide universal function approximation.

4.4 Discussion

Machine learning architectures, based on networks of coupled oscillators, for instance sequence models such as CoRNN (2.3) and UnICORNN (3.6), and the so-called physical neural networks (PNNs) such as linear and nonlinear mechanical oscillators Wright et al. [2022] and spintronic oscillators Romera and et. al [2018], Torrejon and et. al [2017], are being increasingly used. A priori, it is unclear why machine learning systems based on oscillators can provide competitive performance on a variety of learning benchmarks, e.g. Section 2.3, Section 3.3, and Wright et al. [2022], rather than biasing their outputs towards oscillatory functions. In order to address these concerns about their expressivity, we have investigated the theoretical properties of machine learning systems based on oscillators. Our main aim was to answer a fundamental question: *"are coupled oscillator based machine learning architectures universal?"*. In other words, can these architectures, in principle, approximate a large class of input-output maps to desired accuracy.

To answer this fundamental question, we introduced an abstract framework of *neural oscillators* (4.1) and its particular instantiation, the *multi-layer neural oscillators* (4.2). This abstract class of *second-order neural ODEs* encompasses both sequence models such as CoRNN and UnICORNN, as well as a very general and representative PNN, based on the so-called Frenkel-Kontorova model. The main contribution of this chapter was to prove the universality theorem 4.2.1 on the ability of multi-layer neural oscillators (4.2) to approximate a large class of operators, namely causal and continuous maps between spaces of continuous functions, to desired accuracy. Despite the fact that the considered neural oscillators possess a very specific and constrained structure, not even encompassing general Hamiltonian systems,

the approximated class of operators is nevertheless very general, including solution operators of general ordinary and even time-delay differential equations.

The crucial theoretical ingredient in our proof was the *fundamental Lemma* 4.2.5, which implies that linear oscillator dynamics combined with a pointwise nonlinear read-out suffices for universal operator approximation; our construction can correspondingly be thought of as a large number of linear processors, coupled with nonlinear readouts. This construction could have implications for other models such as *structured state-space models* Gu et al. [2021, 2020] which follow a similar paradigm, and the extension of our universality results to such models could be of great interest.

Our universality result has many interesting implications. To start with, we rigorously prove that an machine learning architecture based on coupled oscillators can approximate a very large class of operators. This provides theoretical support to many widely used sequence models and PNNs based on oscillators. Moreover, given the generality of our result, we hope that such a universality result can spur the design of innovative architectures based on oscillators, particularly in the realm of analog devices as machine learning inference systems or machine learning accelerators Wright et al. [2022].

It is also instructive to lay out some of the limitations of our theoretical findings in this chapter and point to avenues for future work. In this context, our setup currently only considers time-varying functions as inputs and outputs. Roughly speaking, these inputs and outputs have the structure of (infinite) sequences. However, a large class of learning tasks can be reconfigured to take sequential inputs and outputs. These include text (as evident from the tremendous success of large language models Radford et al. [2018]), DNA sequences, images Karpathy and Fei-Fei [2015], timeseries and (offline) reinforcement learning Janner et al. [2021]. Nevertheless, a next step would be to extend such universality results to inputs (and outputs) which have some spatial or relational structure, for instance by considering functions which have a spatial dependence or which are defined on graphs. On the other hand, the class of operators that we consider, i.e., casual and continuous, is not only natural in this setting but very general Grigoryeva and Ortega [2018], Gonon et al. [2019].

Another limitation lies in the *feed forward* structure of the multi-layer neural oscillator (4.2). As mentioned before, most physical (and neurobiological) systems exhibit feedback loops between their constituents. However, this is not common in machine learning systems. In fact, we had to use a *mass ordering* in the Frenkel-Kontorova system of coupled pendula (4.8) in order to recast it in the form of the multi-layer neural oscillator (4.2). Such asymptotic ordering may not be possible for arbitrary physical neural networks. Exploring how such ordering mechanisms might arise in physical and biological systems in order to effectively give rise to a feed forward system could be very interesting. One possible mechanism for coupled oscillators that can lead to a hierarchical structure is that of synchronization Winfree [1967], Strogatz [2001] and references therein. How such synchronization interacts with universality is a very interesting question and will serve as an avenue for future work.

Finally, universality is arguably necessary but far from sufficient to analyze the performance of any machine learning architecture. Other aspects such as trainability and generalization are equally important, and we do not address these issues here. We do mention that trainability of oscillatory systems would profit from the fact that oscillatory dynamics is (gradient) stable and this formed the basis of the proofs of mitigation of the exploding and vanishing gradients problem for CoRNN in Section 2.2 and UnICORNN in Section 3.2. Extending these results to the general second-order neural ODE (4.2), for instance through an analysis of the associated adjoint system, is left for future work.

Chapter 5

Long Expressive Memory

Realistic sequential data sets often contain information arranged according to multiple (time, length, etc., depending on the data and task) scales. If there were only one or two scales over which information correlated, then a simple model with a parameter chosen to correspond to that scale (or, e.g., scale difference) should be able to model the data well. Thus, it is reasonable to expect that a *multiscale model* should be considered to process efficiently such *multiscale data*. To this end, we propose a novel sequence model, *Long Expressive Memory* (LEM), that is based on a suitable time-discretization of a set of multiscale ordinary differential equations (ODEs). For this novel sequence model (proposed in Section 5.1):

- we derive bounds on the hidden state gradients to prove that LEM mitigates the exploding and vanishing gradients problem (Section 5.2.1);
- we rigorously prove that LEM can approximate a very large class of (multiscale) dynamical systems to arbitrary accuracy (Section 5.2.1); and
- we provide an extensive empirical evaluation of LEM on a wide variey of data sets, including image and sequence classification, dynamical systems prediction, keyword spotting, and language modeling, thereby demonstrating that LEM outperforms or is comparable to state-of-the-art RNNs, GRUs and LSTMs in each task (Section 5.3).

We also discuss a small portion of the large body of related work, and we provide a brief discussion of our results in a broader context (Section 5.5).

5.1 The proposed sequence model

We start with the simplest example of a system of *two-scale ODEs*,

$$\frac{d\mathbf{y}}{dt} = \tau_y \left(\sigma\left(\mathbf{W}_y\mathbf{z} + \mathbf{V}_y\mathbf{u} + \mathbf{b}_y\right) - \mathbf{y}\right), \quad \frac{d\mathbf{z}}{dt} = \tau_z \left(\sigma\left(\mathbf{W}_z\mathbf{y} + \mathbf{V}_z\mathbf{u} + \mathbf{b}_z\right) - \mathbf{z}\right). \tag{5.1}$$

Here, $t \in [0, T]$ is the continuous time, $0 < \tau_y \leq \tau_z \leq 1$ are the two time scales, $\mathbf{y}(t) \in \mathbb{R}^{m_y}, \mathbf{z}(t) \in \mathbb{R}^{m_z}$ are the vectors of *slow* and *fast* variables and $\mathbf{u} = \mathbf{u}(t) \in \mathbb{R}^d$ is the *input signal*. For simplicity, we set $m_y = m_z = m$. The dynamic interactions between the neurons are modulated by weight matrices $\mathbf{W}_{y,z}, \mathbf{V}_{y,z}$, bias vectors $\mathbf{b}_{y,z}$ and a *nonlinear* tanh activation function $\sigma(u) = \tanh(u)$.

However, two scales (one fast and one slow), may not suffice in representing a large number of scales that could be present in realistic sequential data sets. Hence, we need to generalize (5.1) to a *multiscale* version. One such generalization is provided by the following set of ODEs,

$$\begin{aligned} \frac{d\mathbf{y}}{dt} &= \hat{\sigma}\left(\mathbf{W}_2\mathbf{y} + \mathbf{V}_2\mathbf{u} + \mathbf{b}_2\right) \odot \left(\sigma\left(\mathbf{W}_y\mathbf{z} + \mathbf{V}_y\mathbf{u} + \mathbf{b}_y\right) - \mathbf{y}\right), \\ \frac{d\mathbf{z}}{dt} &= \hat{\sigma}\left(\mathbf{W}_1\mathbf{y} + \mathbf{V}_1\mathbf{u} + \mathbf{b}_1\right) \odot \left(\sigma\left(\mathbf{W}_z\mathbf{y} + \mathbf{V}_z\mathbf{u} + \mathbf{b}_z\right) - \mathbf{z}\right). \end{aligned} \tag{5.2}$$

In addition to previously defined quantities, we need additional weight matrices $\mathbf{W}_{1,2}, \mathbf{V}_{1,2}$, bias vectors $\mathbf{b}_{y,z}$ and sigmoid activation function $\hat{\sigma}(u) = 0.5(1 + \tanh(u/2))$. Note that $\odot$ refers to the componentwise product of vectors. As $\hat{\sigma}$ is monotone, we can set $\mathbf{W}_{1,2} = \mathbf{V}_{1,2} \equiv 0$ and $(\mathbf{b}_1)_j = b_y, (\mathbf{b}_2)_j = b_z$, for all $1 \leq j \leq m$, with $\hat{\sigma}(b_{y,z}) = \tau_{y,z}$ to observe that the two-scale system (5.1) is a special case of (5.2). One can readily generalize this construction to obtain many different scales in (5.2). Thus, we can interpret $(\boldsymbol{\tau}_z(\mathbf{y}, t), \boldsymbol{\tau}_y(\mathbf{y}, t)) = (\hat{\sigma}\left(\mathbf{W}_1\mathbf{y} + \mathbf{V}_1\mathbf{u} + \mathbf{b}_1\right), \hat{\sigma}\left(\mathbf{W}_2\mathbf{y} + \mathbf{V}_2\mathbf{u} + \mathbf{b}_2\right))$ in (5.2) as input and state dependent gating functions, which endow ODE (5.2) with *multiple time scales*. These scales can be learned adaptively (with respect to states) and dynamically (in time).

Next, we propose a time-discretization of the multiscale ODE system (5.2), providing a circuit to our sequential model architecture. As is common with numerical discretizations of ODEs, doing so properly is important to preserve desirable properties. To this end, we fix $\Delta t > 0$, and we discretize (5.2) with the following implicit-explicit (IMEX) time-stepping scheme to arrive at LEM, written in compact form as,

$$\begin{aligned} \boldsymbol{\Delta}\mathbf{t}_n &= \Delta t\hat{\sigma}(\mathbf{W}_1\mathbf{y}_{n-1} + \mathbf{V}_1\mathbf{u}_n + \mathbf{b}_1), \\ \overline{\boldsymbol{\Delta}\mathbf{t}}_n &= \Delta t\hat{\sigma}(\mathbf{W}_2\mathbf{y}_{n-1} + \mathbf{V}_2\mathbf{u}_n + \mathbf{b}_2), \\ \mathbf{z}_n &= (1 - \boldsymbol{\Delta}\mathbf{t}_n) \odot \mathbf{z}_{n-1} + \boldsymbol{\Delta}\mathbf{t}_n \odot \sigma(\mathbf{W}_z\mathbf{y}_{n-1} + \mathbf{V}_z\mathbf{u}_n + \mathbf{b}_z), \\ \mathbf{y}_n &= (1 - \overline{\boldsymbol{\Delta}\mathbf{t}}_n) \odot \mathbf{y}_{n-1} + \overline{\boldsymbol{\Delta}\mathbf{t}}_n \odot \sigma(\mathbf{W}_y\mathbf{z}_n + \mathbf{V}_y\mathbf{u}_n + \mathbf{b}_y). \end{aligned} \tag{5.3}$$

For steps $1 \leq n \leq N$, the hidden states in LEM (5.3) are $\mathbf{y}_n, \mathbf{z}_n \in \mathbb{R}^m$, with input state $\mathbf{u}_n \in \mathbb{R}^d$. The weight matrices are $\mathbf{W}_{1,2,z,y} \in \mathbb{R}^{m\times m}$ and $\mathbf{V}_{1,2,z,y} \in \mathbb{R}^{m\times d}$ and the bias vectors are $\mathbf{b}_{1,2,z,y} \in \mathbb{R}^m$. We also augment LEM (5.3) with a linear *output state* $\omega_n \in \mathbb{R}^o$ with $\omega_n = \mathcal{W}_y\mathbf{y}_n$, and $\mathcal{W}_y \in \mathbb{R}^{o\times m}$.

Related work. We start by comparing our proposed model, LEM (5.3), to the widely used LSTM of Hochreiter and Schmidhuber [1997]. Observe that $\boldsymbol{\Delta}\mathbf{t}_n, \overline{\boldsymbol{\Delta}\mathbf{t}}_n$ in (5.3) are similar in form to the *input, forget* and *output* gates in an LSTM, and that LEM (5.3) has exactly the same number of parameters (weights and biases) as an LSTM, for the same number of hidden units. Moreover, as detailed in the last paragraph of this section, we show that by choosing very specific values of the LSTM gates and the $\boldsymbol{\Delta}\mathbf{t}_n, \overline{\boldsymbol{\Delta}\mathbf{t}}_n$ terms in LEM (5.3), the two models are equivalent. However, this analysis also reveals key differences between LEM (5.3) and LSTMs, as they are equivalent only under very stringent assumptions. In general, as the different gates in both LSTM and LEM (5.3) are *learned* from data, one can expect them to behave differently. Moreover in contrast to LSTM, LEM stems from a discretized ODE system (5.2), which endows it with (gradient) stable dynamics.

The use of *multiscale* neural network architectures in machine learning has a long history. An early example was provided in Hinton and Plaut [1987], who proposed a neural network with each connection having a fast changing weight for temporary memory and a slow changing weight for long-term learning. More recently, one can view convolutional neural networks as multiscale architectures for processing multiple *spatial* scales in data [Bai et al., 2020].

The use of ODE-based learning architectures has also received considerable attention in recent years with examples such as *continuous-time* neural ODEs [Chen et al., 2018, Queiruga et al., 2020, 2021] and

their recurrent extensions ODE-RNNs [Rubanova et al., 2019], as well as RNNs based on discretizations of ODEs [Chang et al., 2019, Erichson et al., 2020, Chen et al., 2020c, Lim et al., 2021b]. In addition to the specific details of our architecture, we differ from other discretized ODE-based RNNs in the explicit use of multiple (learned) scales in LEM.

Relation between LEM and the Hodgkin-Huxley equations. We observe that the multiscale ODEs (5.2), on which LEM is based, are a special case of the following ODE system,

$$\frac{d\mathbf{z}}{dt} = \mathbf{F}_z\left(\mathbf{y}, t\right) - \mathbf{G}_z(\mathbf{y}, t) \odot \mathbf{z}, \quad \frac{d\mathbf{y}}{dt} = \mathbf{F}_y\left(\mathbf{z}, t\right) \odot \mathbf{H}\left(\mathbf{y}, t\right) - \mathbf{G}_y(\mathbf{y}, t) \odot \mathbf{y}. \tag{5.4}$$

It turns out the well-known Hodgkin-Huxley equations Hodgkin and Huxley [1952], modeling the dynamics of the action potential of a biological neuron can also be written down in the abstract form (5.4), with $m_y = 1$, $m_z = 3$ and the variables $\mathbf{y} = y$ modeling the voltage and $\mathbf{z} = (z_1, z_2, z_3)$ modeling the concentration of Potassium activation, Sodium activation and Sodium inactivation channels.

The exact form of the different functions in (5.4) for the Hodgkin-Huxley equations is given by,

$$\begin{aligned}
\mathbf{F}_z(y) &= \left(\alpha_1(y), \alpha_2(y), \alpha_3(y)\right), \\
\mathbf{G}_z(y) &= \left(\alpha_1(y) + \beta_1(y), \alpha_2(y) + \beta_2(y), \alpha_3(y) + \beta_3(y)\right), \\
\alpha_1(y) &= \frac{0.01(10 + \hat{y} - y)}{e^{\frac{10+\hat{y}-y}{10}} - 1}, \quad \alpha_2(y) = \frac{0.1(25 + \hat{y} - y)}{e^{\frac{25+\hat{y}-y}{10}} - 1}, \quad \alpha_3(y) = 0.07e^{\frac{\hat{y}-y}{20}}, \\
\beta_1(y) &= 0.125e^{\frac{\hat{y}-y}{80}}, \quad \beta_2(y) = 4e^{\frac{\hat{y}-y}{18}}, \quad \beta_3(y) = \frac{1}{1 + e^{1+\frac{\hat{y}-y}{10}}}, \\
\mathbf{F}_y(z, t) &= u(t) + z_1^4 + z_2^3 z_3, \quad \mathbf{H}(y) = c_1(\bar{y} - y) + c_2(\bar{y} - y), \quad \mathbf{G}_y(y) = c_3,
\end{aligned} \tag{5.5}$$

with input current u and constants $\hat{y}, \bar{y}, c_{1,2,3}$, whose exact values can be read from [Hodgkin and Huxley, 1952].

Thus, the multiscale ODEs (5.2) and the Hodgkin-Huxley equations are a special case of the same general family (5.4) of ODEs. Moreover, the *gating functions* $\mathbf{G}_{y,z}(\mathbf{y})$, that model voltage-gated ion channels in the Hodgkin-Huxley equations, are similar in form to $\boldsymbol{\Delta t}_n, \overline{\boldsymbol{\Delta t}}_n$ in (5.2).

It is also worth highlighting the differences between our proposed model LEM (and the underlying ODE system (5.2)) and the Hodgkin-Huxley ODEs modeling the dynamics of the neuronal action potential. Given the complicated form of the nonlinearites $\mathbf{F}_{y,z}, \mathbf{G}_{y,z}, \mathbf{H}$ in the Hodgkin-Huxley equations (5.5), we cannot use them in designing any learning model. Instead, building on the abstract form of (5.4), we propose *bespoke* nonlinearities in the ODE (5.2) to yield a tractable learning model, such as LEM (5.3). Moreover, it should be emphasized that the Hodgkin-Huxley equations only model the dynamics of a single neuron (with a scalar voltage and 3 ion channels), whereas the hidden state dimension d of (5.2) can be arbitrary.

Relation between LEM and LSTM. The well-known LSTM [Hochreiter and Schmidhuber, 1997] (in its mainly-used version using a forget gate [Gers et al., 2000]) is given by,

$$\begin{aligned}
\mathbf{f}_n &= \hat{\sigma}(\mathbf{W}_f \mathbf{h}_{n-1} + \mathbf{V}_f \mathbf{u}_n + \mathbf{b}_f) \\
\mathbf{i}_n &= \hat{\sigma}(\mathbf{W}_i \mathbf{h}_{n-1} + \mathbf{V}_i \mathbf{u}_n + \mathbf{b}_i) \\
\mathbf{o}_n &= \hat{\sigma}(\mathbf{W}_o \mathbf{h}_{n-1} + \mathbf{V}_o \mathbf{u}_n + \mathbf{b}_o) \\
\mathbf{c}_n &= \mathbf{f}_n \odot \mathbf{c}_{n-1} + \mathbf{i}_n \odot \sigma(\mathbf{W}\mathbf{h}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}) \\
\mathbf{h}_n &= \mathbf{o}_n \odot \sigma(\mathbf{c}_n).
\end{aligned} \tag{5.6}$$

Here, for any $1 \leq n \leq N$, $\mathbf{h}_n \in \mathbb{R}^m$ is the hidden state and $\mathbf{c}_n \in \mathbb{R}^m$ is the so-called *cell state*. The vectors $\mathbf{i}_n, \mathbf{f}_n, \mathbf{o}_n \in \mathbb{R}^d$ are the *input, forget* and *output* gates, respectively. $\mathbf{u}_n \in \mathbb{R}^d$ is the input signal and the weight matrices and bias vectors are given by $\mathbf{W}, \mathbf{W}_{f,i,o} \in \mathbb{R}^{m \times m}$, $\mathbf{V}, \mathbf{V}_{f,i,o} \in \mathbb{R}^{m \times d}$ and $\mathbf{b}, \mathbf{b}_{f,i,o} \in \mathbb{R}^m$, respectively.

It is straightforward to relate LSTM (5.6) and LEM (5.3) by first setting the cell state $\mathbf{c}_n = \mathbf{z}_n$, for all $1 \leq n \leq N$ and the hidden state $\mathbf{h}_n = \mathbf{y}_n$.

We further need to assume that the input state $\mathbf{i}_n = \boldsymbol{\Delta}\mathbf{t}_n$ and the forget state has to be $\mathbf{f}_n = 1 - \boldsymbol{\Delta}\mathbf{t}_n$. Finally, the output state of the LSTM (5.6) has to be

$$\mathbf{o}_n = \overline{\boldsymbol{\Delta}\mathbf{t}}_n = 1, \quad \forall 1 \leq n \leq N.$$

Under these assumptions and by setting $\Delta t = 1$, we can readily observe that the LEM (5.3) and LSTM (5.6) are equivalent.

A different interpretation of LEM, in relation to LSTM, is as follows; LEM can be thought of a variant of LSTM but with two cell states $\mathbf{y}_n, \mathbf{z}_n$ per unit and no output gate. The input gates are $\boldsymbol{\Delta}\mathbf{t}_n$ and $\overline{\boldsymbol{\Delta}\mathbf{t}}_n$ and the forget gates are coupled to the input gates. Given that the state $\mathbf{z}_n$ is fed into the update for the state $\mathbf{y}_n$, one can think of one of the cell states sitting above the other, leading to a more sophisticated recursive update for LEM (5.3), when compared to LSTM (5.6).

5.2 Rigorous analysis of LEM

In this section, we provide a rigorous analysis of the proposed LEM architecture. We start with the following simplifying notation for various terms in LEM (5.3),

$$\begin{aligned}
\mathbf{A}_{n-1} &= \mathbf{W}_1 \mathbf{y}_{n-1} + \mathbf{V}_1 \mathbf{u}_n + \mathbf{b}_1, \\
\mathbf{B}_{n-1} &= \mathbf{W}_2 \mathbf{y}_{n-1} + \mathbf{V}_2 \mathbf{u}_n + \mathbf{b}_2, \\
\mathbf{C}_{n-1} &= \mathbf{W}_z \mathbf{y}_{n-1} + \mathbf{V}_z \mathbf{u}_n + \mathbf{b}_z, \\
\mathbf{D}_n &= \mathbf{W}_y \mathbf{z}_n + \mathbf{V}_y \mathbf{u}_n + \mathbf{b}_y.
\end{aligned}$$

Note that for all $1 \leq n \leq N$, $\mathbf{A}_n, \mathbf{B}_n, \mathbf{C}_n, \mathbf{D}_n \in \mathbb{R}^m$. With the above notation, LEM (5.3) can be written componentwise, for each component $1 \leq i \leq d$ as,

$$\begin{aligned}
\mathbf{z}_n^i &= \mathbf{z}_{n-1}^i + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i) - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \mathbf{z}_{n-1}^i, \\
\mathbf{y}_n^i &= \mathbf{y}_{n-1}^i + \Delta t \hat{\sigma}(\mathbf{B}_{n-1}^i) \sigma(\mathbf{D}_n^i) - \Delta t \hat{\sigma}(\mathbf{B}_{n-1}^i) \mathbf{y}_{n-1}^i.
\end{aligned} \tag{5.7}$$

Moreover, we recall the *order*-notation from (2.35),

$$\begin{aligned}
\beta = \mathcal{O}(\alpha), \text{for } \alpha, \beta \in \mathbb{R}_+ \quad &\text{if there exist constants } \overline{C}, \underline{C} \text{ such that } \underline{C}\alpha \leq \beta \leq \overline{C}\alpha. \\
\mathbf{M} = \mathcal{O}(\alpha), \text{for } \mathbf{M} \in \mathbb{R}^{d_1 \times d_2}, \alpha \in \mathbb{R}_+ \quad &\text{if there exists a constant } \overline{C} \text{ such that } \|\mathbf{M}\| \leq \overline{C}\alpha.
\end{aligned} \tag{5.8}$$

5.2.1 On the exploding and vanishing gradients problem

Bounds on hidden states. The structure of LEM (5.3) allows us to prove that its hidden states satisfy the following *pointwise bound*.

Proposition 5.2.1. *Denote $t_n = n\Delta t$ and assume that $\Delta t \leq 1$. Further assume that the initial hidden states are $\mathbf{z}_0 = \mathbf{y}_0 \equiv 0$. Then, the hidden states $\mathbf{z}_n, \mathbf{y}_n$ of LEM (5.3) are bounded pointwise as,*

$$\max_{1 \leq i \leq d} \max\{|\mathbf{z}_n^i|, |\mathbf{y}_n^i|\} \leq \min\left(1, \overline{\Delta}\sqrt{t_n}\right), \quad \forall 1 \leq n, \text{ with } \overline{\Delta} = \frac{1 + \Delta t}{\sqrt{2 - \Delta t}}. \tag{5.9}$$

Proof. The proof of the bound (5.9) is split into 2 parts. We start with the first equation in (5.7) and rewrite it as,

$$\mathbf{z}_n^i = \left(1 - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i)\right) \mathbf{z}_{n-1}^i + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i).$$

Noting that the activation functions are such that $0 \leq \hat{\sigma}(x) \leq 1$, for all x and $-1 \leq \sigma(x) \leq 1$, for all x and using the fact that $\Delta t \leq 1$, we have from the above expression that,

$$\begin{aligned} \mathbf{z}_n^i &\leq \left(1 - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i)\right) \max\left(\mathbf{z}_{n-1}^i, 1\right) + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \max\left(\mathbf{z}_{n-1}^i, 1\right), \\ &\leq \max\left(\mathbf{z}_{n-1}^i, 1\right). \end{aligned}$$

By a symmetric argument, one can readily show that,

$$\mathbf{z}_n^i \geq \min(-1, \mathbf{z}_{n-1}^i).$$

Combining the above inequalities yields,

$$\min(-1, \mathbf{z}_{n-1}^i) \leq \mathbf{z}_n^i \leq \max\left(\mathbf{z}_{n-1}^i, 1\right). \tag{5.10}$$

Iterating (5.10) over n and using $z_0^i = 0$ for all $1 \leq i \leq d$ leads to,

$$-1 \leq \mathbf{z}_n^i \leq 1, \quad \forall n, \quad \forall 1 \leq i \leq d. \tag{5.11}$$

An argument, identical to the derivation of (5.11), but for the hidden state $\mathbf{y}$ yields,

$$-1 \leq \mathbf{y}_n^i \leq 1, \quad \forall n, \quad \forall 1 \leq i \leq d. \tag{5.12}$$

Thus, we have shown that the hidden states remain in the interval $[-1, 1]$, irrespective of the sequence length.

Next, we will use the following elementary identities in the proof,

$$b(a-b) = \frac{a^2}{2} - \frac{b^2}{2} - \frac{1}{2}(a-b)^2, \tag{5.13}$$

for any $a, b \in \mathbb{R}$, and also,

$$ab \leq \frac{\epsilon a^2}{2} + \frac{b^2}{2\epsilon}, \quad \forall \epsilon > 0. \tag{5.14}$$

We fix $1 \leq i \leq d$ and multiply the first equation in (5.7) with $\mathbf{z}_{n-1}^i$ and apply (5.13) to obtain,

$$\begin{aligned} \frac{(\mathbf{z}_n^i)^2}{2} &= \frac{(\mathbf{z}_{n-1}^i)^2}{2} + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i) \mathbf{z}_{n-1}^i - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i)(\mathbf{z}_{n-1}^i)^2 + \frac{(\mathbf{z}_n^i - \mathbf{z}_{n-1}^i)^2}{2} \\ &= \frac{(\mathbf{z}_{n-1}^i)^2}{2} + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i) \mathbf{z}_{n-1}^i - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i)(\mathbf{z}_{n-1}^i)^2 \\ &+ \frac{\Delta t^2}{2} \left(\hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i) - \hat{\sigma}(\mathbf{A}_{n-1}^i) \mathbf{z}_{n-1}^i\right)^2, \quad \text{(from (5.7))} \\ &\leq \frac{(\mathbf{z}_{n-1}^i)^2}{2} + \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i) |\sigma(\mathbf{C}_{n-1}^i)| |\mathbf{z}_{n-1}^i| - \Delta t \hat{\sigma}(\mathbf{A}_{n-1}^i)(\mathbf{z}_{n-1}^i)^2 \\ &+ \frac{\Delta t^2}{2} (\hat{\sigma}(\mathbf{A}_{n-1}^i) \sigma(\mathbf{C}_{n-1}^i))^2 + \frac{\Delta t^2}{2} \hat{\sigma}(\mathbf{A}_{n-1}^i)^2 (\mathbf{z}_{n-1}^i)^2 \\ &+ \Delta t^2 \hat{\sigma}(\mathbf{A}_{n-1}^i)^2 |\sigma(\mathbf{C}_{n-1}^i)| |\mathbf{z}_{n-1}^i| \quad (\text{as } (a-b)^2 \leq a^2 + b^2 + 2|a||b|) \end{aligned}$$

We fix $\epsilon = \frac{2-\Delta t}{1+\Delta t}$ in the elementary identity (5.14) to yield,

$$|\sigma(\mathbf{C}^i_{n-1})||\mathbf{z}^i_{n-1}| \leq \frac{\sigma(\mathbf{C}^i_{n-1})^2}{2\epsilon} + \frac{\epsilon(\mathbf{z}^i_{n-1})^2}{2}$$

Applying this to the inequality for $(\mathbf{z}^i_n)^2$ leads to,

$$\begin{aligned}\frac{(\mathbf{z}^i_n)^2}{2} \leq \frac{(\mathbf{z}^i_{n-1})^2}{2} + \left(\Delta t\hat{\sigma}(\mathbf{A}^i_{n-1}) + \Delta t^2\hat{\sigma}(\mathbf{A}^i_{n-1})^2\right)\frac{\sigma(\mathbf{C}^i_{n-1})^2}{2\epsilon} \\ - \Delta t\hat{\sigma}(\mathbf{A}^i_{n-1})\left[1 - \frac{\epsilon}{2} - \frac{\Delta t\hat{\sigma}(\mathbf{A}^i_{n-1})}{2} - \frac{\Delta t\hat{\sigma}(\mathbf{A}^i_{n-1})\epsilon}{2}\right](\mathbf{z}^i_{n-1})^2.\end{aligned}$$

Using the fact that $0 \leq \hat{\sigma}(x) \leq 1$ for all $x \in \mathbb{R}$, $\sigma^2 \leq 1$ and that $\epsilon = \frac{2-\Delta t}{1+\Delta t}$, we obtain from the last line of the previous equation that,

$$(\mathbf{z}^i_n)^2 \leq (\mathbf{z}^i_{n-1})^2 + \frac{\Delta t + \Delta t^2}{\epsilon} \leq (\mathbf{z}^i_{n-1})^2 + \frac{\Delta t(1+\Delta t)^2}{2-\Delta t}, \quad \forall 1 \leq n.$$

Iterating the above estimate over $n = 1, \ldots, \bar{n}$, for any $1 \leq \bar{n}$ and setting $\bar{n} = n$ yields,

$$\begin{aligned}&(\mathbf{z}^i_n)^2 \leq (\mathbf{z}^i_0)^2 + n\frac{\Delta t(1+\Delta t)^2}{2-\Delta t}, \\ \Rightarrow \quad &(\mathbf{z}^i_n)^2 \leq t_n\frac{(1+\Delta t)^2}{2-\Delta t} \quad \text{as } \mathbf{z}^i_0 = 0,\ t_n = n\Delta t.\end{aligned}$$

Taking a square root in the above inequality yields,

$$|\mathbf{z}^i_n| \leq \overline{\Delta}\sqrt{t_n}, \quad \forall n, \quad \forall 1 \leq i \leq m. \tag{5.15}$$

with $\overline{\Delta}$ defined in the expression (5.9).

We can repeat the above argument with the hidden state $\mathbf{y}$ to obtain,

$$|\mathbf{y}^i_n| \leq \overline{\Delta}\sqrt{t_n}, \quad \forall n, \quad \forall 1 \leq i \leq m. \tag{5.16}$$

Combining (5.15) and (5.16) with the pointwise bounds (5.11) and (5.12) yields the desired bound (5.9). □

On the exploding and vanishing gradients problem. For any $1 \leq n \leq N$, let $\mathbf{X}_n \in \mathbb{R}^{2m}$, denoted the *combined hidden state*, given by $\mathbf{X}_n = \left[\mathbf{z}^1_n, \mathbf{y}^1_n, \ldots\ldots, \mathbf{z}^i_n, \mathbf{y}^i_n, \ldots\ldots, \mathbf{z}^m_n, \mathbf{y}^m_n\right]$. For simplicity of the exposition, we consider a *loss function*: $\mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n - \overline{\mathbf{y}}_n\|^2$, with $\overline{\mathbf{y}}_n$ being the underlying *ground truth*. The training of our proposed model (5.3) entails computing *gradients* of the above loss function with respect to its underlying weights and biases $\theta \in \mathbf{\Theta} = [\mathbf{W_{1,2,y,z}}, \mathbf{V_{1,2,y,z}}, \mathbf{b_{1,2,y,z}}]$, at every step of the gradient descent procedure. Following Pascanu et al. [2013], one uses chain rule to show,

$$\frac{\partial\mathcal{E}_n}{\partial\theta} = \sum_{1\leq k\leq n}\frac{\partial\mathcal{E}^{(k)}_n}{\partial\theta}, \quad \frac{\partial\mathcal{E}^{(k)}_n}{\partial\theta} = \frac{\partial\mathcal{E}_n}{\partial\mathbf{X}_n}\frac{\partial\mathbf{X}_n}{\partial\mathbf{X}_k}\frac{\partial^+\mathbf{X}_k}{\partial\theta}. \tag{5.17}$$

In general, for recurrent models, the partial gradient $\frac{\partial\mathcal{E}^{(k)}_n}{\partial\theta}$, which measures the contribution to the hidden state gradient at step n arising from step k of the model, can behave as $\frac{\partial\mathcal{E}^{(k)}_n}{\partial\theta} \sim \gamma^{n-k}$, for some

$\gamma > 0$ Pascanu et al. [2013]. If $\gamma > 1$, then the partial gradient grows *exponentially* in sequence length, for long-term dependencies $k << n$, leading to the exploding gradient problem. On the other hand, if $\gamma < 1$, then the partial gradient decays *exponentially* for $k << n$, leading to the vanishing gradient problem. Thus, mitigation of the exploding and vanishing gradients problem entails deriving bounds on the gradients. We start with the following upper bound,

Proposition 5.2.2. *Let $\mathbf{z}_n, \mathbf{y}_n$ be the hidden states generated by LEM (5.3). We assume that $\Delta t << 1$ is chosen to be sufficiently small. Then, the gradient of the loss function $\mathcal{E}_n$ with respect to any parameter $\theta \in \mathbf{\Theta}$ is bounded as*

$$\begin{aligned} \left| \frac{\partial \mathcal{E}_n}{\partial \theta} \right| &\leq (1 + \hat{\mathbf{Y}}) t_n + (1 + \hat{\mathbf{Y}}) \Gamma t_n^2, \quad \hat{Y} = \|\overline{\mathbf{y}}_n\|_\infty, \\ \eta = \max\{\|\mathbf{W}_1\|_\infty, \|\mathbf{W}_2\|_\infty, &\|\mathbf{W}_z\|_\infty, \|\mathbf{W}_y\|_\infty\}, \quad \Gamma = 2\,(1+\eta)\,(1+3\eta) \end{aligned} \tag{5.18}$$

Proof. We can apply the chain rule repeatedly (for instance as in Pascanu et al. [2013]) to obtain,

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{1 \leq k \leq n} \underbrace{\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n} \frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} \frac{\partial^+ \mathbf{X}_k}{\partial \theta}}_{\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}}. \tag{5.19}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k$ with respect to the parameter θ, while keeping the other arguments constant.

A straightforward application of the product rule yields,

$$\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} = \prod_{k < \ell \leq n} \frac{\partial \mathbf{X}_\ell}{\partial \mathbf{X}_{\ell-1}}. \tag{5.20}$$

For any $k < \ell \leq n$, a tedious yet straightforward computation yields the following representation formula,

$$\frac{\partial \mathbf{X}_\ell}{\partial \mathbf{X}_{\ell-1}} = \mathbf{I}_{2m \times 2m} + \Delta t \mathbf{E}^{\ell,\ell-1} + \Delta t^2 \mathbf{F}^{\ell,\ell-1}. \tag{5.21}$$

Here $\mathbf{E}^{\ell,\ell-1} \in \mathbb{R}^{2m \times 2m}$ is a matrix whose entries are given below. For any $1 \leq i \leq m$, we have,

$$\begin{aligned} \mathbf{E}^{\ell,\ell-1}_{2i-1,2j-1} &\equiv 0, \quad j \neq i \\ \mathbf{E}^{\ell,\ell-1}_{2i-1,2i-1} &= -\hat{\sigma}(\mathbf{A}^i_{\ell-1}), \\ \mathbf{E}^{\ell,\ell-1}_{2i-1,2j} &= (\mathbf{W}_1)_{i,j} \hat{\sigma}'(\mathbf{A}^i_{\ell-1}) \left(\sigma(\mathbf{C}^i_{\ell-1}) - \mathbf{z}^i_{\ell-1}\right) + (\mathbf{W}_z)_{i,j} \hat{\sigma}(\mathbf{A}^i_{\ell-1}) \sigma'(\mathbf{C}^i_{\ell-1}), \ \forall 1 \leq j \leq d \\ \mathbf{E}^{\ell,\ell-1}_{2i,2j-1} &= (\mathbf{W}_y)_{i,j} \hat{\sigma}(\mathbf{B}^i_{\ell-1}) \sigma'(\mathbf{D}^i_\ell), \ \forall 1 \leq j \leq m \\ \mathbf{E}^{\ell,\ell-1}_{2i,2j} &= (\mathbf{W}_2)_{i,j} \hat{\sigma}'(\mathbf{B}^i_{\ell-1}) \left(\sigma(\mathbf{D}^i_\ell) - \mathbf{y}^i_{\ell-1}\right), \quad j \neq i \\ \mathbf{E}^{\ell,\ell-1}_{2i,2i} &= -\hat{\sigma}(\mathbf{B}^i_{\ell-1}) + (\mathbf{W}_2)_{i,i} \hat{\sigma}'(\mathbf{B}^i_{\ell-1}) \left(\sigma(\mathbf{D}^i_\ell) - \mathbf{y}^i_{\ell-1}\right). \end{aligned} \tag{5.22}$$

Similarly, $\mathbf{F}^{\ell,\ell-1} \in \mathbb{R}^{2m \times 2m}$ is a matrix whose entries are given below. For any $1 \leq i \leq m$, we have,

$$\begin{aligned} \mathbf{F}^{\ell,\ell-1}_{2i-1,j} &\equiv 0, \quad \forall\ 1 \leq j \leq 2m, \\ \mathbf{F}^{\ell,\ell-1}_{2i,2j-1} &= -(\mathbf{W}_y)_{i,j} \hat{\sigma}(\mathbf{A}^j_{\ell-1}) \hat{\sigma}(\mathbf{B}^i_{\ell-1}) \sigma'(\mathbf{D}^i_\ell), \quad 1 \leq j \leq m, \\ \mathbf{F}^{\ell,\ell-1}_{2i,2j} &= \hat{\sigma}(\mathbf{B}^i_{\ell-1}) \sigma'(\mathbf{D}^i_\ell) \sum_{\lambda=1}^{d} (\mathbf{W}_y)_{i,\lambda} \left(\left(\sigma(\mathbf{C}^\lambda_{\ell-1}) - \mathbf{z}^\lambda_{\ell-1}\right) \hat{\sigma}'(\mathbf{A}^\lambda_{\ell-1}) (\mathbf{W}_1)_{\lambda,j} \right. \\ &\quad \left. + \hat{\sigma}(\mathbf{A}^\lambda_{\ell-1}) \sigma'(\mathbf{C}^\lambda_{\ell-1}) (\mathbf{W}_z)_{\lambda,j} \right). \end{aligned} \tag{5.23}$$

Using the fact that,

$$\sup_{x\in\mathbb{R}} \max\left\{|\sigma(x)|, |\sigma'(x)|, |\hat{\sigma}(x)|, |\hat{\sigma}'(x)|\right\} \leq 1,$$

the pointwise bounds (5.9), the notation $t_n = n\Delta t$ for all n, the definition of η (5.18) and the definition of matrix norms, we obtain that,

$$\begin{aligned}\|\mathbf{E}^{\ell,\ell-1}\|_\infty &\leq \max\left\{1 + \|\mathbf{W}_z\|_\infty + (1 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\|\mathbf{W}_1\|_\infty, 1 + \|\mathbf{W}_y\|_\infty + (1 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\|\mathbf{W}_2\|_\infty\right\} \\ &\leq 1 + (2 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\eta.\end{aligned} \tag{5.24}$$

By similar calculations, we obtain,

$$\begin{aligned}\|\mathbf{F}^{\ell,\ell-1}\|_\infty &\leq \|\mathbf{W}_y\|_\infty \left(1 + (1 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\|\mathbf{W}_1\|_\infty + \|\mathbf{W}_z\|_\infty\right) \\ &\leq \eta(1 + (2 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\eta).\end{aligned} \tag{5.25}$$

Applying (5.24) and (5.25) in the representation formula (5.21) and observing that $\Delta t \leq 1$ and $\ell \leq n$, we obtain.

$$\begin{aligned}\left\|\frac{\partial \mathbf{X}_\ell}{\partial \mathbf{X}_{\ell-1}}\right\|_\infty &\leq 1 + \left(1 + (2 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\eta\right)\Delta t + \eta\left(1 + (2 + \min(1, \overline{\Delta}\sqrt{t_\ell}))\eta\right)\Delta t^2, \\ &\leq 1 + \frac{\Gamma}{2}\Delta t,\end{aligned}$$

with

$$\Gamma = 2\,(1+\eta)\,(1+3\eta). \tag{5.26}$$

Using the expression (5.20) with the above inequality yields,

$$\left\|\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k}\right\|_\infty \leq \left(1 + \frac{\Gamma}{2}\Delta t\right)^{n-k}. \tag{5.27}$$

Next, we choose $\Delta t << 1$ small enough such that the following holds,

$$\left(1 + \frac{\Gamma}{2}\Delta t\right)^{n-k} \leq 1 + \Gamma(n-k)\Delta t, \tag{5.28}$$

for any $1 \leq k < n$.

Hence applying (5.28) in (5.27), we obtain,

$$\left\|\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k}\right\|_\infty \leq 1 + \Gamma(n-k)\Delta t. \tag{5.29}$$

For the sake of definiteness, we fix any $1 \leq \alpha, \beta \leq m$ and set $\theta = (\mathbf{W}_y)_{\alpha,\beta}$ in the following. The following bounds for any other choice of $\theta \in \Theta$ can be derived analogously. Given this, it is straightforward to calculate from the structure of LEM (5.3) that entries of the vector $\frac{\partial^+ \mathbf{X}_k}{\partial (\mathbf{W}_y)_{\alpha,\beta}}$ are given by,

$$\begin{aligned}\left(\frac{\partial^+ \mathbf{X}_k}{\partial (\mathbf{W}_y)_{\alpha,\beta}}\right)_j &\equiv 0, \quad \forall\ j \neq 2\alpha, \\ \left(\frac{\partial^+ \mathbf{X}_k}{\partial (\mathbf{W}_y)_{\alpha,\beta}}\right)_{2\alpha} &= \Delta t \hat{\sigma}(\mathbf{B}^\alpha_{k-1})\sigma'(\mathbf{D}^\alpha_k)\mathbf{z}^\beta_k.\end{aligned} \tag{5.30}$$

Hence, by the pointwise bounds (5.9), we obtain from (5.30) that

$$\left\| \frac{\partial^+ \mathbf{X}_k}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right\|_\infty \leq \Delta t \min(1, \overline{\Delta}\sqrt{t_k}). \tag{5.31}$$

Finally, it is straightforward to calculate from the loss function $\mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n - \overline{\mathbf{y}}_n\|^2$ that

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n} = \left[0, \mathbf{y}_n^1 - \bar{\mathbf{y}}^1, \ldots\ldots, 0, \mathbf{y}_n^m - \bar{\mathbf{y}}^m\right]. \tag{5.32}$$

Therefore, using the pointwise bounds (5.9) and the notation $\hat{\mathbf{Y}} = \|\bar{\mathbf{y}}\|_\infty$, we obtain

$$\left\| \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n} \right\|_\infty \leq \hat{\mathbf{Y}} + \min(1, \overline{\Delta}\sqrt{t_n}). \tag{5.33}$$

Applying (5.29), (5.31) and (5.33) in the definition (5.19) yields,

$$\left| \frac{\partial \mathcal{E}_n^{(k)}}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right| \leq \Delta t \min(1, \overline{\Delta}\sqrt{t_k}) \left(\hat{\mathbf{Y}} + \min(1, \overline{\Delta}\sqrt{t_n}) \right) (1 + \Gamma(n-k)\Delta t). \tag{5.34}$$

Observing that $1 \leq k \leq n$, we see that $n - k \leq n$ and $t_k \leq t_n$. Therefore, (5.35) can be estimated for any $1 \leq k \leq n$ by,

$$\left| \frac{\partial \mathcal{E}_n^{(k)}}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right| \leq \Delta t \min(1, \overline{\Delta}\sqrt{t_n}) \left(\hat{\mathbf{Y}} + \min(1, \overline{\Delta}\sqrt{t_n}) \right) (1 + \Gamma t_n), \quad 1 \leq k \leq n. \tag{5.35}$$

Applying the bound (5.35) in (5.19) leads to the following bound on the total gradient,

$$\begin{aligned} \left| \frac{\partial \mathcal{E}_n}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right| &\leq \sum_{k=1}^{n} \left| \frac{\partial \mathcal{E}_n^{(k)}}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right| \\ &\leq t_n \min(1, \overline{\Delta}\sqrt{t_n}) \left(\hat{\mathbf{Y}} + \min(1, \overline{\Delta}\sqrt{t_n}) \right) (1 + \Gamma t_n) \\ &\leq t_n (1 + \hat{\mathbf{Y}})(1 + \Gamma t_n) \\ &\leq (1 + \hat{\mathbf{Y}}) t_n + (1 + \hat{\mathbf{Y}}) \Gamma t_n^2 \end{aligned} \tag{5.36}$$

which is the desired bound (5.18) for $\theta = (\mathbf{W}_y)_{\alpha,\beta}$.

Moreover, for *long-term dependencies* i.e., $k << n$, we can set $t_k = k\Delta t < 1$, with k independent of sequence length n, in (5.34) to obtain the following bound on the partial gradient,

$$\left| \frac{\partial \mathcal{E}_n^{(k)}}{\partial (\mathbf{W}_y)_{\alpha,\beta}} \right| \leq \Delta t^{\frac{3}{2}} \overline{\Delta}\sqrt{k} \left(1 + \hat{\mathbf{Y}} \right) (1 + \Gamma t_n), \quad 1 \leq k << n. \tag{5.37}$$

□

Remark 5.2.3. *The bound* (5.18) *on the total gradient depends on $t_n = n\Delta t$, with n being the sequence length and $\Delta t \leq 1$, a hyperparameter which can either be chosen a priori or determined through a hyperparameter tuning procedure. The proof of the bound* (5.36) *relies on Δt being sufficiently small. It would be natural to choose $\Delta t \sim n^{-s}$, for some $s \geq 0$. Substituting this expression in* (5.18) *leads to a bound of the form,*

$$\left| \frac{\partial \mathcal{E}_n}{\partial \theta} \right| = \mathcal{O}\left(n^{2(1-s)} \right) \tag{5.38}$$

If $s = 1$, then clearly $\left|\frac{\partial \mathcal{E}_n}{\partial \theta}\right| = \mathcal{O}(1)$ i.e., the total gradient is bounded. Clearly, the exploding gradient problem is mitigated in this case.

On the other hand, if s takes another value, for instance $s = \frac{1}{2}$ which is empirically observed during the hyperparameter training (see Section 5.3), then we can readily observe from (5.38) *that $\left|\frac{\partial \mathcal{E}_n}{\partial \theta}\right| = \mathcal{O}(n)$. Thus in this case, the gradient can grow with sequence length n but only linearly and not exponentially. Thus, the exploding gradient problem is also mitigated in this case.*

Following Pascanu et al. [2013], to mitigate the vanishing gradient problem, we need to obtain a more precise characterization of the gradient $\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}$ defined in (5.19). For the sake of definiteness, we fix any $1 \leq \alpha, \beta \leq m$ and set $\theta = (\mathbf{W}_y)_{\alpha,\beta}$ in the following. The following formulas for any other choice of $\theta \in \Theta$ can be derived analogously. Moreover, for simplicity of notation, we set the target function $\bar{\mathbf{X}}_n \equiv 0$.

Proposition 5.2.4. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states generated by LEM* (5.3)*, then we have the following representation formula for the hidden state gradient,*

$$\begin{aligned}
\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} &= \Delta t \hat{\sigma}(\mathbf{B}_{k-1}^{\alpha}) \sigma'(\mathbf{D}_k^{\alpha}) \mathbf{z}_k^{\beta} \left(\mathbf{y}_n^{\alpha} - \bar{\mathbf{y}}_n^{\alpha}\right) \\
&+ \Delta t^2 \hat{\sigma}(\mathbf{B}_{k-1}^{\alpha}) \sigma'(\mathbf{D}_k^{\alpha}) \mathbf{z}_k^{\beta} \left[\sum_{j=1}^{d} \left(\mathbf{y}_n^j - \bar{\mathbf{y}}_n^j\right) \sum_{\ell=k+1}^{n} \hat{\sigma}'(\mathbf{B}_{\ell-1}^j) \left(\sigma(\mathbf{D}_{\ell}^j) - \mathbf{y}_{\ell-1}^j\right) (\mathbf{W}_2)_{j,2\alpha} \right] \\
&+ \Delta t^2 \hat{\sigma}(\mathbf{B}_{k-1}^{\alpha}) \sigma'(\mathbf{D}_k^{\alpha}) \mathbf{z}_k^{\beta} \left[\sum_{\ell=k+1}^{n} \hat{\sigma}(\mathbf{B}_{\ell-1}^{\alpha}) \left(\mathbf{y}_n^{\alpha} - \bar{\mathbf{y}}_n^{\alpha}\right) \right] + \mathcal{O}(\Delta t^3).
\end{aligned} \tag{5.39}$$

Here, the constants in $\mathcal{O}$ could depend on η defined in (5.18)*.*

Proof. The starting point for deriving an asymptotic formula for the hidden state gradient $\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta}$ is to observe from the representation formula (5.21), the bound (5.25) on matrices $\mathbf{F}^{\ell,\ell-1}$ and the order notation (5.8) that,

$$\frac{\partial \mathbf{X}_{\ell}}{\partial \mathbf{X}_{\ell-1}} = \mathbf{I}_{2m \times 2m} + \Delta t \mathbf{E}^{\ell,\ell-1} + \mathcal{O}(\Delta t^2), \tag{5.40}$$

as long as η is independent of Δt.

By using induction and the bounds (5.24),(5.25), it is straightforward to calculate the following representation formula for the product,

$$\frac{\partial \mathbf{X}_n}{\partial \mathbf{X}_k} = \prod_{k<\ell\leq n} \frac{\partial \mathbf{X}_{\ell}}{\partial \mathbf{X}_{\ell-1}} = \mathbf{I}_{2m \times 2m} + \Delta t \sum_{\ell=k+1}^{n} \mathbf{E}^{\ell,\ell-1} + \mathcal{O}(\Delta t^2). \tag{5.41}$$

Recall that we have set $\theta = (\mathbf{W}_y)_{\alpha,\beta}$. Hence, by the expressions (5.32) and (5.30), a direct but tedious

calculation leads to,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n}\mathbf{I}_{2m\times 2m}\frac{\partial^+ \mathbf{X}_k}{\partial \theta} = \Delta t\hat{\sigma}(\mathbf{B}_{k-1}^\alpha)\sigma'(\mathbf{D}_k^\alpha)\mathbf{z}_k^\beta\left(\mathbf{y}_n^\alpha - \bar{\mathbf{y}}_n^\alpha\right),$$

$$\sum_{\ell=k+1}^{n}\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n}\mathbf{E}^{\ell,\ell-1}\frac{\partial^+ \mathbf{X}_k}{\partial \theta} =$$

$$\Delta t\hat{\sigma}(\mathbf{B}_{k-1}^\alpha)\sigma'(\mathbf{D}_k^\alpha)\mathbf{z}_k^\beta\left[\sum_{j=1}^{d}\left(\mathbf{y}_n^j - \bar{\mathbf{y}}_n^j\right)\sum_{\ell=k+1}^{n}\hat{\sigma}'(\mathbf{B}_{\ell-1}^j)\left(\sigma(\mathbf{D}_\ell^j) - \mathbf{y}_{\ell-1}^j\right)(\mathbf{W}_2)_{j,2\alpha} - \sum_{\ell=k+1}^{n}\hat{\sigma}(\mathbf{B}_{\ell-1}^\alpha)\left(\mathbf{y}_n^\alpha - \bar{\mathbf{y}}_n^\alpha\right)\right]. \tag{5.42}$$

Therefore, by substituting the above expression into the representation formula (5.41) yields the desired formula (5.39).

In order to prove the formula (5.45), we focus our interest on long-term dependencies i.e., $k << n$. Then, a closer perusal of the expression in (5.42), together with the pointwise bounds (5.9) which implies that $\mathbf{y}_{k-1} \approx \mathcal{O}(\sqrt{\Delta t})$, results in the following,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n}\mathbf{I}_{2m\times 2m}\frac{\partial^+ \mathbf{X}_k}{\partial \theta} = \mathcal{O}\left(\Delta t^{\frac{3}{2}}\right). \tag{5.43}$$

Similarly, we also obtain,

$$\Delta t\sum_{\ell=k+1}^{n}\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n}\mathbf{E}^{\ell,\ell-1}\frac{\partial^+ \mathbf{X}_k}{\partial \theta} = \mathcal{O}\left(\Delta t^{\frac{3}{2}}\right). \tag{5.44}$$

Combining (5.43) and (5.44) results in the desired asymptotic bound (5.45). □

A straight-forward corollary of proposition 5.2.4 is,

Proposition 5.2.5. *Let $\mathbf{y}_n, \mathbf{z}_n$ be the hidden states generated by LEM* (5.3) *and the ground truth satisfy $\overline{\mathbf{y}}_n \sim \mathcal{O}(1)$. Then, for any $k << n$ (long-term dependencies) we have,*

$$\frac{\partial \mathcal{E}_n^{(k)}}{\partial \theta} = \mathcal{O}\left(\Delta t^{\frac{3}{2}}\right). \tag{5.45}$$

Here, constants in $\mathcal{O}(\Delta t^{\frac{3}{2}})$ depend on only on η (5.18) *and $\overline{\eta} = \|\mathbf{W}_2\|_1$ and are independent of n, k.*

This formula (5.45) shows that although the partial gradient can be small, i.e., $\mathcal{O}(\Delta t^{\frac{3}{2}})$, it is in fact *independent of k*, ensuring that long-term dependencies contribute to gradients at much later steps and mitigating the vanishing gradient problem.

Remark 5.2.6. *The upper bound on the gradient* (5.18) *and the gradient asymptotic formula* (5.45) *impact the choice of the timestep hyperparameter Δt. For sequence length n, if we choose $\Delta t \sim n^{-s}$, with $s \geq 0$, we see from Remark 5.2.3 that the upper bound on the total gradient scales like $\mathcal{O}(n^{2(1-s)})$. On the other hand, from* (5.45)*, the gradient contribution from long-term dependencies will scale like $\mathcal{O}(n^{\frac{-3s}{2}})$.*

Hence, a small value of $s \approx 0$, will ensure that the gradient with respect to long-term dependencies will be $\mathcal{O}(1)$. However, the total gradient will behave like $\mathcal{O}(n^2)$ and possibly blow up fast. Similarly, setting $s \approx 1$ leads to a bounded gradient, while the contributions from long-term dependencies decay as fast as $n^{\frac{-3}{2}}$. Hence, one has to find a value of s that balances both these requirements. Equilibrating them leads to $s = \frac{4}{7}$, ensuring that the total gradient grows sub-linearly while long-term dependencies still contribute with a sub-linear decay. This value is very close to the empirically observed value of $s = \frac{1}{2}$ which also ensures that the total gradient grows linearly and the contribution of long-term dependencies decays sub-linearly in the sequence length n.

5.2.2 Universality of LEM

Universal approximation of general dynamical systems. The above bounds on hidden state gradients show that the proposed model LEM (5.3) mitigates the exploding and vanishing gradients problem. However, this by itself, does not guarantee that it can learn complicated and realistic input-output maps between sequences. To investigate the *expressivity* of the proposed LEM, we will show in the following proposition that it can approximate *any* dynamical system, mapping an input sequence $\mathbf{u}_n$ to an output sequence $\mathbf{o}_n$, of the (very) general form,

$$\boldsymbol{\phi}_n = \mathbf{f}\left(\boldsymbol{\phi}_{n-1}, \mathbf{u}_n\right), \quad \mathbf{o}_n = \mathbf{o}(\boldsymbol{\phi}_n), \quad \forall\ 1 \leq n \leq N, \tag{5.46}$$

with $\boldsymbol{\phi}_n \in \mathbb{R}^{d_h}, \mathbf{o}_n \in \mathbb{R}^{d_o}$ denoting the *hidden* and *output* states, respectively. The input signal is $\mathbf{u}_n \in \mathbb{R}^{d_u}$ and maps $\mathbf{f} : \mathbb{R}^{d_h} \times \mathbb{R}^{d_u} \mapsto \mathbb{R}^{d_h}$ and $\mathbf{o} : \mathbb{R}^{d_h} \mapsto \mathbb{R}^{d_o}$ are Lipschitz continuous. For simplicity, we set the initial state $\boldsymbol{\phi}_0 = 0$.

Proposition 5.2.7. *For all $1 \leq n \leq N$, let $\boldsymbol{\phi}_n, \mathbf{o}_n$ be given by the dynamical system (5.46) with input signal $\mathbf{u}_n$. Under the assumption that there exists a $R > 0$ such that $\max\{\|\boldsymbol{\phi}_n\|, \|\mathbf{u}_n\|\} < R$, for all $1 \leq n \leq N$, then for any given $\epsilon > 0$ there exists a LEM of the form (5.3), with hidden states $\mathbf{y}_n, \mathbf{z}_n \in \mathbb{R}^{d_y}$ and output state $\omega_n = \mathcal{W}_y \mathbf{y}_n \in \mathbb{R}^{d_o}$, for some d_y such that the following holds,*

$$\|\mathbf{o}_n - \omega_n\| \leq \epsilon, \quad \forall 1 \leq n \leq N. \tag{5.47}$$

Proof. To prove this proposition, we have to construct hidden states $\mathbf{y}_n, \mathbf{z}_n$, output state ω_n, weight matrices $\mathbf{W}_{1,2,y,z}, \mathcal{W}_y, \mathbf{V}_{1,2,y,z}$ and bias vectors $\mathbf{b}_{1,2,y,z}$ such that LEM (5.3) with output state $\omega_n = \mathcal{W}_y \mathbf{y}_n$ approximates the dynamical system (5.46).

Let $R^* > R >> 1$ and $\epsilon^* < \epsilon$, be parameters to be defined later. By the theorem for universal approximation of continuous functions with neural networks with the tanh activation function $\sigma = \tanh$ [Barron, 1993], given ϵ^*, there exist weight matrices $W_1 \in \mathbb{R}^{d_1 \times d_h}, V_1 \in \mathbb{R}^{d_1 \times d_u}, W_2 \in \mathbb{R}^{d_h \times d_1}$ and bias vector $b_1 \in \mathbb{R}^{d_1}$ such that the tanh neural network defined by,

$$\mathcal{N}_1(h, u) = W_2 \sigma\left(W_1 h + V_1 u + b_1\right), \tag{5.48}$$

approximates the underlying function $\mathbf{f}$ in the following manner,

$$\max_{\max(\|h\|, \|u\|) < R^*} \|\mathbf{f}(h, u) - \mathcal{N}_1(h, u)\| \leq \epsilon^*. \tag{5.49}$$

Similarly, one can readily approximate the identity function $\mathbf{g}(h, u) = h$ with a tanh neural network of the form,

$$\bar{\mathcal{N}}_2(h) = \bar{W}_2 \sigma\left(\bar{W}_1 h\right), \tag{5.50}$$

such that

$$\max_{\|h\|,\|u\|<R^*} \|\mathbf{g}(h) - \mathcal{N}_2(h)\| \leq \epsilon^*. \tag{5.51}$$

Next, we define the following dynamical system,

$$\begin{aligned} \bar{\mathbf{z}}_n &= W_2\sigma\left(W_1\bar{\mathbf{y}}_{n-1} + V_1\mathbf{u}_n + b_1\right), \\ \bar{\mathbf{y}}_n &= \bar{W}_2\sigma\left(\bar{W}_1\bar{\mathbf{z}}_n\right), \end{aligned} \tag{5.52}$$

with initial states $\bar{\mathbf{z}}_0 = \bar{\mathbf{y}}_0 = 0$.

Using the approximation bound (5.49), we derive the following bound,

$$\begin{aligned} \|\boldsymbol{\phi}_n - \bar{\mathbf{y}}_n\| &= \|\mathbf{f}\left(\boldsymbol{\phi}_{n-1}, \mathbf{u}_n\right) - \bar{\mathbf{z}}_n + \bar{\mathbf{z}}_n - \bar{\mathbf{y}}_n\| \\ &\leq \|\mathbf{f}\left(\boldsymbol{\phi}_{n-1}, \mathbf{u}_n\right) - W_2\sigma\left(W_1\bar{\mathbf{y}}_{n-1} + V_1\mathbf{u}_n + b_1\right)\| + \|\mathbf{g}(\bar{\mathbf{z}}_n) - \bar{W}_2\sigma\left(\bar{W}_1\bar{\mathbf{z}}_n\right)\| \\ &\leq \|\mathbf{f}\left(\boldsymbol{\phi}_{n-1}, \mathbf{u}_n\right) - \mathbf{f}\left(\bar{\mathbf{y}}_{n-1}, \mathbf{u}_n\right)\| + \|\mathbf{f}\left(\bar{\mathbf{y}}_{n-1}, \mathbf{u}_n\right) - W_2\sigma\left(W_1\bar{\mathbf{y}}_{n-1} + V_1\mathbf{u}_n + b_1\right)\| \\ &+ \|\mathbf{g}(\bar{\mathbf{z}}_n) - \bar{W}_2\sigma\left(\bar{W}_1\bar{\mathbf{z}}_n\right)\| \\ &\leq \mathrm{Lip}(\mathbf{f})\|\boldsymbol{\phi}_{n-1} - \bar{\mathbf{y}}_{n-1}\| + 2\epsilon^* \quad \text{(from (5.49),(5.51))}. \end{aligned}$$

Here, $\mathrm{Lip}(\mathbf{f})$ is the Lipschitz constant of the function $\mathbf{f}$ on the compact set $\{(h,u) \in \mathbb{R}^{d_h \times d_u} : \|h\|, \|u\| < \mathbb{R}^*\}$. Note that one can readily prove using the fact that $\bar{\mathbf{y}}_0 = \bar{\mathbf{z}}_0 = 0$, bounds (5.49), (5.51) and the assumption $\|\boldsymbol{\phi}_n\|, \|\mathbf{u}_n\| < R$, that $\|\bar{\mathbf{z}}_n\|, \|\bar{\mathbf{y}}_n\| < R^* = 2R$.

Iterating the above inequality over n leads to the bound,

$$\|\boldsymbol{\phi}_n - \bar{\mathbf{y}}_n\| \leq 2\epsilon^* \sum_{\lambda=0}^{n-1} \mathrm{Lip}(\mathbf{f})^\lambda. \tag{5.53}$$

Hence, using the Lipschitz continuity of the output function $\mathbf{o}$ in (5.46), one obtains,

$$\|\mathbf{o}_n - \mathbf{o}(\bar{\mathbf{y}}_n)\| \leq 2\epsilon^* \mathrm{Lip}(\mathbf{o}) \sum_{\lambda=0}^{n-1} \mathrm{Lip}(\mathbf{f})^\lambda, \tag{5.54}$$

with $\mathrm{Lip}(\mathbf{o})$ being the Lipschitz constant of the function $\mathbf{o}$ on the compact set $\{h \in \mathbb{R}^{d_h} : \|h\| < R^*\}$.

Next, we can use the universal approximation theorem for neural networks again to conclude that given a tolerance $\bar{\epsilon}$, there exist weight matrices $W_3 \in \mathbb{R}^{d_2 \times d_h}, W_4 \in \mathbb{R}^{d_h \times d_2}$ and bias vector $b_2 \in \mathbb{R}^{d_2}$ such that the tanh neural network defined by,

$$\mathcal{N}_3(h) = W_4\sigma\left(W_3 h + b_2\right), \tag{5.55}$$

approximates the underlying output function $\mathbf{o}$ in the following manner,

$$\max_{\|h\|<R^*} \|\mathbf{o}(h) - \mathcal{N}_3(h)\| \leq \bar{\epsilon}. \tag{5.56}$$

Now defining,

$$\bar{\omega}_n = W_4\sigma\left(W_3\bar{\mathbf{y}}_n + b_2\right), \tag{5.57}$$

we obtain from (5.56) and (5.54) that,

$$\|\mathbf{o}_n - \bar{\omega}_n\| \leq \bar{\epsilon} + 2\epsilon^* \mathrm{Lip}(\mathbf{o}) \sum_{\lambda=0}^{n-1} \mathrm{Lip}(\mathbf{f})^\lambda. \tag{5.58}$$

Next, we introduce the notation,

$$\tilde{\mathbf{z}}_n = \sigma\left(W_1\bar{\mathbf{y}}_{n-1} + V_1\mathbf{u}_n + b_1\right), \quad \tilde{\mathbf{y}}_n = \sigma\left(\bar{W}_1\bar{\mathbf{z}}_n\right). \tag{5.59}$$

From (5.52), we see that

$$\bar{\mathbf{z}}_n = W_2\tilde{\mathbf{z}}_n, \quad \bar{\mathbf{y}}_n = \bar{W}_2\tilde{\mathbf{y}}_n \tag{5.60}$$

Thus from (5.60) and (5.58), we have

$$\begin{aligned} \bar{\omega}_n &= W_4\sigma\left(W_3W_2\tilde{\mathbf{y}}_n + b_2\right), \\ &= W_4\sigma\left(W_3W_2\sigma\left(\bar{W}_1W_2\tilde{\mathbf{z}}_n\right) + b_2\right). \end{aligned} \tag{5.61}$$

Define the function $\mathcal{R} : \mathbb{R}^{d_h} \times \mathbb{R}^{d_u} \mapsto \mathbb{R}^{d_o}$ by,

$$\mathcal{R}(z) = W_4\sigma\left(W_3W_2\sigma\left(\bar{W}_1W_2z\right) + b_2\right). \tag{5.62}$$

The function, defined above, is clearly Lipschitz continuous. We can apply the universal approximation theorem for tanh neural networks to find, for any given tolerance $\tilde{\epsilon}$, weight matrices $W_5 \in \mathbb{R}^{d_3\times d_4}, W_6 \in \mathbb{R}^{d_o\times d_3}, V_2 \in \mathbb{R}^{d_3\times d_u}$ and bias vector $b_3 \in \mathbb{R}^{d_3}$ such that the following holds,

$$\max_{\max(\|z\|)<R^*} \|\mathcal{R}(z) - W_6\sigma(W_5z + b_3)\| \leq \tilde{\epsilon}. \tag{5.63}$$

Denote $\tilde{\omega}_n = W_6\sigma(W_5\tilde{\mathbf{z}}_n + b_3)$, then from (5.63) and (5.61), we obtain that

$$\|\bar{\omega}_n - \tilde{\omega}_n\| \leq \tilde{\epsilon}.$$

Combining this estimate with (5.58) yields,

$$\|\mathbf{o}_n - \tilde{\omega}_n\| \leq \tilde{\epsilon} + \bar{\epsilon} + 2\epsilon^*\mathrm{Lip}(\mathbf{o})\sum_{\lambda=0}^{n-1}\mathrm{Lip}(\mathbf{f})^\lambda. \tag{5.64}$$

Now, we collect all ingredients to define the LEM that can approximate the dynamical system (5.46). To this end, we define hidden states $\mathbf{z}_n, \mathbf{y}_n \in \mathbb{R}^{2d_h}$ as

$$\mathbf{z}_n = \left[\tilde{\mathbf{z}}_n, \hat{\mathbf{z}}_n\right], \quad \mathbf{y}_n = \left[\tilde{\mathbf{y}}_n, \hat{\mathbf{y}}_n\right],$$

with $\tilde{\mathbf{z}}_n, \hat{\mathbf{z}}_n, \tilde{\mathbf{y}}_n, \hat{\mathbf{y}}_n \in \mathbb{R}^{d_h}$. These hidden states are evolved by the dynamical system,

$$\begin{aligned} \mathbf{z}_n^\perp &= \sigma\left(\begin{bmatrix} W_1\bar{W}_2 & 0 \\ 0 & 0 \end{bmatrix}\mathbf{y}_{n-1}^\perp + [V_1\mathbf{u}_n, 0]^\perp + [b_1, 0]^\perp\right), \\ \mathbf{y}_n^\perp &= \sigma\left(\begin{bmatrix} \bar{W}_1W_2 & 0 \\ W_5 & 0 \end{bmatrix}\mathbf{z}_n^\perp + [0, 0]^\perp + [0, b_3]^\perp\right) \end{aligned} \tag{5.65}$$

and the output state is calculated by,

$$\omega_n^\perp = [0, W_6]\mathbf{y}_n^\perp. \tag{5.66}$$

Finally, we can recast the dynamical system (5.65), (5.66) as a LEM of the form (5.3) for the hidden states $\mathbf{y}_n, \mathbf{z}_n$, defined in (5.65), with the following parameters, Now, define the hidden states $\bar{\mathbf{y}}_n, \bar{\mathbf{z}}_n \in \mathbb{R}^{d_y}$

for all $1 \leq n \leq N$ by the LEM (5.3) with the following parameters,

$$\begin{aligned}
&\Delta t = 1, \quad d_y = 2d_h, \\
&\mathbf{W}_1 = \mathbf{W}_2 = \mathbf{V}_1 = \mathbf{V}_2 = 0 \\
&\mathbf{b}_1 = \mathbf{b}_2 = \mathbf{b}_\infty, \\
&\mathbf{W}_z = \begin{bmatrix} W_1 \bar{W}_2 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{V}_z = [V_1, 0], \quad \mathbf{b}_z = [b_1, 0] \\
&\mathbf{W}_y = \begin{bmatrix} \bar{W}_1 W_2 & 0 \\ W_5 & 0 \end{bmatrix}, \quad \mathbf{V}_y = 0, \quad \mathbf{b}_z = [0, b_3]. \\
&\mathcal{W}_y = [0, W_6].
\end{aligned} \tag{5.67}$$

Here, $\mathbf{b}_\infty \in \mathbb{R}^{d_h}$ is defined as

$$\mathbf{b}_\infty = [b_\infty, b_\infty, \ldots, \ldots, b_\infty],$$

with $1 << b_\infty$ is such that

$$|1 - \hat{\sigma}(b_\infty)| \leq \delta. \tag{5.68}$$

The nature of the sigmoid function guarantees the existence of such a b_∞ for any δ. As δ decays exponentially fast, we set it to 0 in the following for notational simplicity.

It is straightforward to verify that the output state of the LEM (5.3) with parameters given in (5.67) is $\omega_n = \tilde{\omega}_n$.

Therefore, from (5.64) and by setting $\bar{\epsilon} < \frac{\epsilon}{3}$, $\tilde{\epsilon} < \frac{\epsilon}{3}$ and

$$\epsilon^* < \frac{\epsilon}{6\mathrm{Lip}(\mathbf{o}) \sum_{\lambda=0}^{N-1} \mathrm{Lip}(\mathbf{f})^\lambda},$$

we prove the desired bound (5.47). □

From this proposition, we conclude that, in principle, the proposed LEM (5.3) can approximate a very large class of dynamical systems.

Universal approximation of multiscale dynamical systems. While expressing a general form of input-output maps between sequences, the dynamical system (5.46) does not explicitly model dynamics at multiple scales. Instead, here we consider the following two-scale *fast-slow* dynamical system of the general form,

$$\boldsymbol{\phi}_n = \mathbf{f}(\boldsymbol{\phi}_{n-1}, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n), \quad \boldsymbol{\psi}_n = \tau \mathbf{g}(\boldsymbol{\phi}_n, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n), \quad \mathbf{o}_n = \mathbf{o}(\boldsymbol{\psi}_n). \tag{5.69}$$

Here, $0 < \tau << 1$ and 1 are the slow and fast time scales, respectively. The underlying maps $(\mathbf{f}, \mathbf{g}) : \mathbb{R}^{d_h \times d_h \times d_u} \mapsto \mathbb{R}^{d_h}$ are Lipschitz continuous. In the following proposition we show that LEM (5.3) can approximate (5.69) to desired accuracy.

Proposition 5.2.8. *For any $0 < \tau << 1$, and for all $1 \leq n \leq N$, let $\boldsymbol{\phi}_n, \boldsymbol{\psi}_n, \mathbf{o}_n$ be given by the two-scale dynamical system* (5.69) *with input signal $\mathbf{u}_n$. Under the assumption that there exists a $R > 0$ such that $\max\{\|\boldsymbol{\phi}_n\|, \|\boldsymbol{\psi}_n\|, \|\mathbf{u}_n\|\} < R$, for all $1 \leq n \leq N$, then for any given $\epsilon > 0$, there exists a LEM of the form* (5.3)*, with hidden states $\mathbf{y}_n, \mathbf{z}_n \in \mathbb{R}^{d_y}$ and output state $\omega_n \in \mathbb{R}^{d_o}$ with $\omega_n = \mathcal{W}\mathbf{y}_n$ such that the following holds,*

$$\|\mathbf{o}_n - \omega_n\| \leq \epsilon, \quad \forall 1 \leq n \leq N. \tag{5.70}$$

Moreover, the weights, biases and size (number of neurons) of the underlying LEM (5.3) *are* independent *of the time-scale τ.*

Proof. The proof of this proposition is based heavily on the proof of Proposition 5.2.7. Hence, we will highlight the main points of difference.

As the steps for approximation of a general Lipschitz continuous output map are identical to the corresponding steps in the proof of proposition 5.2.7 (see the steps from Eqns. (5.54) to (5.64)), we will only consider the following linear output map for convenience herein,

$$\mathbf{o}(\boldsymbol{\psi}_n) = \mathcal{W}_c \boldsymbol{\psi}_n. \tag{5.71}$$

Let $R^* > R >> 1$ and $\epsilon^* < \epsilon$, be parameters to be defined later. By the theorem for universal approximation of continuous functions with neural networks with the tanh activation function $\sigma = \tanh$, given ϵ^*, there exist weight matrices $W_1^f, W_2^f \in \mathbb{R}^{d_1 \times d_h}, V_1^f \in \mathbb{R}^{d_1 \times d_u}, W_3^f \in \mathbb{R}^{d_h \times d_1}$ and bias vector $b_1^f \in \mathbb{R}^{d_1}$ such that the tanh neural network defined by,

$$\mathcal{N}_f(h, c, u) = W_3^f \sigma \left(W_1^f h + W_2^f c + V_1^f u + b_1^f \right), \tag{5.72}$$

approximates the underlying function $\mathbf{f}$ in the following manner,

$$\max_{\max(\|h\|, \|c\|, \|u\|) < R^*} \|\mathbf{f}(h, c, u) - \mathcal{N}_f(h, c, u)\| \leq \epsilon^*. \tag{5.73}$$

Next, we define the following map,

$$\mathbf{G}(h, c, u) = \mathbf{g}(h, c, u) + \left(1 - \frac{1}{\tau}\right) c, \tag{5.74}$$

for any $\tau > 0$.

By the universal approximation theorem, given ϵ^*, there exist weight matrices $W_1^g, W_2^g \in \mathbb{R}^{d_2 \times d_h}, V_1^g \in \mathbb{R}^{d_2 \times d_u}, W_3^g \in \mathbb{R}^{d_h \times d_2}$ and bias vector $b_1^g \in \mathbb{R}^{d_2}$ such that the tanh neural network defined by,

$$\mathcal{N}_g(h, c, u) = W_3^g \sigma \left(W_1^g h + W_2^g c + V_1^g u + b_1^g \right), \tag{5.75}$$

approximates the function $\mathbf{G}$ (5.74) in the following manner,

$$\max_{\max(\|h\|, \|c\|, \|u\|) < R^*} \|\mathbf{G}(h, c, u) - \mathcal{N}_f(h, c, u)\| \leq \epsilon^*. \tag{5.76}$$

Note that the sizes of the neural network $\mathcal{N}_g$ can be made independent of the small parameter τ by simply taking the sum of the neural networks approximating the functions g and $\hat{g}(h, c, u) = c$ with tanh neural networks. As neither of these functions depend on the small parameter τ, the sizes of the corresponding neural networks are independent of the small parameter too.

Next, as in the proof of proposition 5.2.7, one can readily approximate the identity function $\hat{f}(h, c, u) = h$ with a tanh neural network of the form,

$$\bar{\mathcal{N}}_f(h) = \bar{W}_2 \sigma \left(\bar{W}_1 h \right), \tag{5.77}$$

such that

$$\max_{\|h\|, \|c\|, \|u\| < R^*} \|\hat{f}(h, c, u) - \mathcal{N}_f(h)\| \leq \epsilon^*, \tag{5.78}$$

and with the same weights and biases, one can approximate the identity function $\hat{g}(h, c, u) = c$ with the tanh neural network,

$$\bar{\mathcal{N}}_g(c) = \bar{W}_2 \sigma \left(\bar{W}_1 c \right), \tag{5.79}$$

such that

$$\max_{\|h\|,\|c\|,\|u\|<R^*} \|\hat{g}(h,c,u) - \mathcal{N}_g(c)\| \leq \epsilon^*. \tag{5.80}$$

Next, we define the following dynamical system,

$$\begin{aligned}
\hat{\mathbf{z}}_n &= W_3^f \sigma\left(W_1^f \tilde{\mathbf{y}}_{n-1} + W_2^f \hat{\mathbf{y}}_{n-1} + V_1^f \mathbf{u}_n + b_1^f\right), \\
\tilde{\mathbf{z}}_n &= \bar{W}_2 \sigma\left(\bar{W}_1 \hat{\mathbf{y}}_{n-1}\right), \\
\hat{\mathbf{y}}_n &= (1-\tau)\hat{\mathbf{y}}_{n-1} + \tau W_3^g \sigma\left(W_1^g \hat{\mathbf{z}}_n + W_2^g \tilde{\mathbf{z}}_n + V_1^g \mathbf{u}_n + b_1^g\right), \\
\tilde{\mathbf{y}}_n &= \bar{W}_2 \sigma\left(\bar{W}_1 \hat{\mathbf{z}}_n\right),
\end{aligned} \tag{5.81}$$

with hidden states $\hat{\mathbf{z}}_n, \tilde{\mathbf{z}}_n, \hat{\mathbf{y}}_n, \tilde{\mathbf{y}}_n \in \mathbb{R}^{d_h}$ and with initial states $\hat{\mathbf{z}}_0 = \tilde{\mathbf{z}}_0 = \hat{\mathbf{y}}_0 = \tilde{\mathbf{y}}_0 = 0$.

We derive the following bounds,

$$\begin{aligned}
\|\boldsymbol{\phi}_n - \hat{\mathbf{z}}_n\| &= \|\mathbf{f}(\boldsymbol{\phi}_{n-1}, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n) - W_3^f \sigma\left(W_1^f \tilde{\mathbf{y}}_{n-1} + W_2^f \hat{\mathbf{y}}_{n-1} + V_1^f \mathbf{u}_n + b_1^f\right)\| \\
&\leq \|\mathbf{f}(\boldsymbol{\phi}_{n-1}, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n) - \mathbf{f}(\tilde{\mathbf{y}}_{n-1}, \hat{\mathbf{z}}_{n-1}, \mathbf{u}_n)\|, \\
&+ \|\mathbf{f}(\tilde{\mathbf{y}}_{n-1}, \hat{\mathbf{z}}_{n-1}, \mathbf{u}_n) - W_3^f \sigma\left(W_1^f \tilde{\mathbf{y}}_{n-1} + W_2^f \hat{\mathbf{y}}_{n-1} + V_1^f \mathbf{u}_n + b_1^f\right)\| \\
&\leq \mathrm{Lip}(\mathbf{f})\left(\|\boldsymbol{\phi}_{n-1} - \hat{\mathbf{z}}_{n-1}\| + 2\|\tilde{\mathbf{y}}_{n-1} - \hat{\mathbf{z}}_{n-1}\| + \|\boldsymbol{\psi}_{n-1} - \tilde{\mathbf{y}}_{n-1}\|\right) + \epsilon^* \quad \text{(by (5.76))} \\
&\leq \mathrm{Lip}(\mathbf{f})\left(\|\boldsymbol{\phi}_{n-1} - \hat{\mathbf{z}}_{n-1}\| + \|\boldsymbol{\psi}_{n-1} - \tilde{\mathbf{y}}_{n-1}\|\right) + (1 + 2\mathrm{Lip}(\mathbf{f}))\,\epsilon^* \quad \text{(by (5.78),(5.81))},
\end{aligned}$$

and

$$\begin{aligned}
\|\boldsymbol{\psi}_n - \hat{\mathbf{y}}_n\| &= \|(1-\tau)(\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}) + \tau\left(\mathbf{G}(\boldsymbol{\phi}_n, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n) - W_3^g \sigma\left(W_2^g \tilde{\mathbf{z}}_n + W_1^g \hat{\mathbf{z}}_n + V_1^g \mathbf{u}_n + b_1^g\right)\right)\| \\
&\leq \|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}\| + \tau\|\mathbf{G}(\boldsymbol{\phi}_n, \boldsymbol{\psi}_{n-1}, \mathbf{u}_n) - \mathbf{G}(\hat{\mathbf{z}}_n, \tilde{\mathbf{z}}_n, \mathbf{u}_n)\| \\
&\quad + \tau\|\mathbf{G}(\hat{\mathbf{z}}_n, \tilde{\mathbf{z}}_n, \mathbf{u}_n) - W_3^g \sigma\left(W_2^g \tilde{\mathbf{z}}_n + W_1^g \hat{\mathbf{z}}_n + V_1^g \mathbf{u}_n + b_1^g\right)\| \\
&\leq \|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1})\| + \tau\mathrm{Lip}(\mathbf{G})\left(\|\boldsymbol{\phi}_n - \hat{\mathbf{z}}_n\| + \|\tilde{\mathbf{z}}_n - \hat{\mathbf{y}}_{n-1}\| + \|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}\|\right) + \tau\epsilon^*, \\
&\leq (1 + \tau\mathrm{Lip}(\mathbf{G}))(1 + \mathrm{Lip}(\mathbf{f}))\|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}\| + \tau\mathrm{Lip}(\mathbf{G})\mathrm{Lip}(\mathbf{f})\|\boldsymbol{\phi}_{n-1} - \hat{\mathbf{z}}_{n-1}\| \\
&\quad + \tau(1 + \mathrm{Lip}(\mathbf{G})(2 + 2\mathrm{Lip}(\mathbf{f})))\epsilon^*,
\end{aligned}$$

where the last inequality follows by using the previous inequality together with (5.81) and (5.80).

As $\tau < 1$, it is easy to see from (5.74) that $\mathrm{Lip}(\mathbf{G}) < \mathrm{Lip}(\mathbf{g}) + \frac{2}{\tau}$. Therefore, the last inequality reduces to,

$$\begin{aligned}
\|\boldsymbol{\psi}_n - \hat{\mathbf{y}}_n\| \leq (3 + \tau\mathrm{Lip}(\mathbf{g}))(1 + \mathrm{Lip}(\mathbf{f}))\|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}\| + (2 + \tau\mathrm{Lip}(\mathbf{g}))\mathrm{Lip}(\mathbf{f})\|\boldsymbol{\phi}_{n-1} - \hat{\mathbf{z}}_{n-1}\| \\
+ (\tau + (2 + \tau\mathrm{Lip}(\mathbf{g}))(2 + 2\mathrm{Lip}(\mathbf{f})))\epsilon^*.
\end{aligned}$$

Adding we obtain,

$$\|\boldsymbol{\phi}_n - \hat{\mathbf{z}}_n\| + \|\boldsymbol{\psi}_n - \hat{\mathbf{y}}_n\| \leq C^*\left(\|\boldsymbol{\phi}_{n-1} - \hat{\mathbf{z}}_{n-1}\| + \|\boldsymbol{\psi}_{n-1} - \hat{\mathbf{y}}_{n-1}\|\right) + D^*\epsilon^*, \tag{5.82}$$

where,

$$\begin{aligned}
C^* &= \max\{(3 + \mathrm{Lip}(\mathbf{g}))\mathrm{Lip}(\mathbf{f}), \mathrm{Lip}(\mathbf{f})(3 + \mathrm{Lip}(\mathbf{g}))(1 + \mathrm{Lip}(\mathbf{f}))\}, \\
D^* &= 1 + (2 + \mathrm{Lip}(\mathbf{g}))(2 + 2\mathrm{Lip}(\mathbf{f})).
\end{aligned} \tag{5.83}$$

Iterating over n leads to the bound,

$$\|\boldsymbol{\phi}_n - \hat{\mathbf{z}}_n\| + \|\boldsymbol{\psi}_n - \hat{\mathbf{y}}_n\| \leq \epsilon^* D^* \sum_{\lambda=0}^{n-1} (C^*)^\lambda. \tag{5.84}$$

Here, $\mathrm{Lip}(\mathbf{f}), \mathrm{Lip}(\mathbf{g})$ are the Lipschitz constants of the functions $\mathbf{f}, \mathbf{g}$ on the compact set $\{(h, c, u) \in \mathbb{R}^{d_h \times d_h \times d_u} : \|h\|, \|c\|, \|u\| < \mathbb{R}^*\}$. Note that one can readily prove using the zero values of initial states, the bounds (5.78), (5.80) and the assumption $\|\boldsymbol{\phi}_n\|, \|\boldsymbol{\psi}_n\|, \|\mathbf{u}_n\| < R$, that $\|\hat{\mathbf{z}}_n\|, \|\tilde{\mathbf{z}}_n\|, \|\hat{\mathbf{y}}_n\|, \|\tilde{\mathbf{y}}_n\| < R^* = 2R$.

Using the definition of the output function (5.70) and the bound (5.84) that,

$$\|\mathbf{o}_n - \mathbf{o}(\hat{\mathbf{y}}_n)\| \leq \|\mathcal{W}_c\| \epsilon^* D^* \sum_{\lambda=0}^{n-1} (C^*)^\lambda. \tag{5.85}$$

Defining the dynamical system,

$$\begin{aligned}
\mathbf{z}_n^* &= \sigma\left(W_1^f \bar{W}_2 \bar{\mathbf{y}}_{n-1} + W_2^f W_3^g \mathbf{y}_{n-1}^* + V_1^f \mathbf{u}_n + b_1^f\right) \\
\bar{\mathbf{z}}_n &= \sigma\left(\bar{W}_1 W_3^g \mathbf{y}_{n-1}^*\right) \\
\mathbf{y}_n^* &= (1-\tau)\mathbf{y}_{n-1}^* + \tau\sigma\left(W_1^g W_3^f \mathbf{z}_n^* + W_2^g \bar{W}_2 \bar{\mathbf{z}}_n + V_1^g \mathbf{u}_n + b_1^g\right), \\
\bar{\mathbf{y}}_n &= \sigma\left(\bar{W}_1 W_3^f \mathbf{z}_n^*\right).
\end{aligned} \tag{5.86}$$

By multiplying suitable matrices to (5.81), we obtain that,

$$\hat{\mathbf{z}}_n = W_3^f \mathbf{z}_n^*, \quad \tilde{\mathbf{z}}_n = \bar{W}_2 \bar{\mathbf{z}}_n, \quad \hat{\mathbf{y}}_n = W_3^g \mathbf{y}_n^*, \quad \tilde{\mathbf{y}}_n = \bar{W}_2 \bar{\mathbf{y}}_n. \tag{5.87}$$

Finally, in addition to b_∞ defined in (5.53), for any given $\tau \in (0, 1]$, we introduce $b_\tau \in \mathbb{R}$ defined by

$$\hat{\sigma}(b_\tau) = \tau. \tag{5.88}$$

The existence of a unique b_τ follows from the fact that the sigmoid function $\hat{\sigma}$ is monotone. Next, we define the two vectors $\mathbf{b}_\infty, \mathbf{b}_\tau \in \mathbb{R}^{2d_h}$ as

$$\begin{aligned}
\mathbf{b}_\infty^i &= b_\infty, \quad \forall\ 1 \leq i \leq 2d_h, \\
\mathbf{b}_\tau^i &= b_\tau, \quad \forall\ 1 \leq i \leq d_h, \\
\mathbf{b}_\tau^i &= b_\infty, \quad \forall\ d_h + 1 \leq i \leq 2d_h.
\end{aligned} \tag{5.89}$$

We are now in a position to define the LEM of form (5.3), which will approximate the two-scale dynamical system (5.69). To this end, we define the hidden states $\mathbf{z}_n, \mathbf{y}_n \in \mathbb{R}^{2d_h}$ such that $\mathbf{z}_n = [\mathbf{z}_n^*, \bar{\mathbf{z}}_n]$ and $\mathbf{y}_n = [\mathbf{y}_n^*, \bar{\mathbf{y}}_n]$. The parameters for the corresponding LEM of form (5.3) given by,

$$\begin{aligned}
&\Delta t = 1, d_y = 2d_h \\
&\mathbf{W}_1 = \mathbf{W}_2 = \mathbf{V}_1 = \mathbf{V}_2 \equiv 0, \\
&\mathbf{b}_1 = \mathbf{b}_\infty, \quad \mathbf{b}_2 = \mathbf{b}_\tau, \\
&\mathbf{W}_z = \begin{bmatrix} W_2^f W_3^g & W_1^f \bar{W}_2 \\ \bar{W}_1 W_3^g & 0 \end{bmatrix}, \quad \mathbf{V}_z = [V_1^f 0], \quad \mathbf{b}_z = [b_1^f, 0], \\
&\mathbf{W}_y = \begin{bmatrix} W_1^g W_3^f & W_2^g \bar{W}_2 \\ \bar{W}_1 W_3^f & 0 \end{bmatrix}, \quad \mathbf{V}_z = [V_1^g 0], \quad \mathbf{b}_z = [b_1^g, 0],
\end{aligned} \tag{5.90}$$

and with following parameters defining the output states,

$$\mathcal{W}_y = [\mathcal{W}_c W_3^g\ 0], \tag{5.91}$$

yields an output state $\omega_n = \mathcal{W}_y \mathbf{y}_n$.

It is straightforward to observe that $\omega_n \equiv \mathbf{o}(\hat{\mathbf{y}}_n)$. Hence, the desired bound (5.70) follows from (5.84) by choosing,

$$\epsilon^* = \frac{\epsilon}{D^* \sum_{\lambda=0}^{N-1} (C^*)^\lambda}.$$

□

The proof of proposition 5.2.8 can be readily extended to prove the following proposition about a general r-scale dynamical system of the form,

$$\begin{aligned}
\boldsymbol{\phi}_n^1 &= \tau_1 \mathbf{f}^1(\boldsymbol{\phi}_{n-1}^1, \boldsymbol{\phi}_{n-1}^2, \ldots, \boldsymbol{\phi}_{n-1}^r \mathbf{u}_n), \\
\boldsymbol{\phi}_n^2 &= \tau_2 \mathbf{f}^2(\boldsymbol{\phi}_{n-1}^1, \boldsymbol{\phi}_{n-1}^2, \ldots, \boldsymbol{\phi}_{n-1}^r \mathbf{u}_n), \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\boldsymbol{\phi}_n^r &= \tau_r \mathbf{f}^r(\boldsymbol{\phi}_{n-1}^1, \boldsymbol{\phi}_{n-1}^2, \ldots, \boldsymbol{\phi}_{n-1}^r \mathbf{u}_n), \\
\mathbf{o}_n &= \mathbf{o}(\boldsymbol{\phi}_n^1, \boldsymbol{\phi}_n^2, \ldots, \boldsymbol{\phi}_n^r).
\end{aligned} \tag{5.92}$$

Here, $\tau_1 \leq \tau_2 \ldots\ldots \leq \tau_r \leq 1$, with $r > 1$, are the r-time scales of the dynamical system (5.92). We assume that the underlying maps $\mathbf{f}^{1,2,\ldots,r}$ are Lipschitz continuous. We can prove the following proposition,

Proposition 5.2.9. *For all $1 \leq n \leq N$, let $\boldsymbol{\phi}_n^{1,2,\ldots,r}, \mathbf{o}_n$ be given by the r-scale dynamical system* (5.92) *with input signal $\mathbf{u}_n$. Under the assumption that there exists a $R > 0$ such that $\max\{\|\boldsymbol{\phi}_n^1\|, |\boldsymbol{\phi}_n^2\|, \ldots, |\boldsymbol{\phi}_n^r\|, \|\mathbf{u}_n\|\} < R$, for all $1 \leq n \leq N$, then for any given $\epsilon > 0$, there exists a LEM of the form* (5.3)*, with hidden states $\mathbf{y}_n, \mathbf{z}_n$ and output state ω_n with $\omega_n = \mathcal{W}\mathbf{y}_n$ such that the following holds,*

$$\|\mathbf{o}_n - \omega_n\| \leq \epsilon, \quad \forall 1 \leq n \leq N. \tag{5.93}$$

Moreover, the weights, biases and size (number of neurons) of the underlying LEM (5.3) *are* independent *of the time-scales $\tau_{1,2,\ldots,r}$.*

Hence, we show that, in principle, the proposed model LEM (5.3) can approximate multiscale dynamical systems, with model size being *independent* of the underlying timescales. These theoretical results for LEM (5.3) point to the ability of this architecture to learn complicated multiscale input-output maps between sequences, while mitigating the exploding and vanishing gradients problem. Although useful prerequisities, these theoretical properties are certainly not sufficient to demonstrate that LEM (5.3) is efficient in practice. To do this, we perform several benchmark evaluations, and we report the results in Section 5.3.

5.2.3 LEMs emulate Heterogeneous multiscale methods for ODEs

Following Kuehn [2015], we consider the following prototypical example of a fast-slow system of ordinary differential equations,

$$\begin{aligned}
\boldsymbol{\phi}'(t) &= \frac{1}{\tau}\left(f(\boldsymbol{\psi}) - \boldsymbol{\phi}\right), \\
\boldsymbol{\psi}'(t) &= g(\boldsymbol{\phi}, \boldsymbol{\psi}).
\end{aligned} \tag{5.94}$$

Here $\boldsymbol{\phi}, \boldsymbol{\psi} \in \mathbb{R}^{d_c}$ are the fast and slow variables respectively and $0 < \tau << 1$ is a small parameter. Note that we have rescaled time and are interested in the dynamics of the slow variable $\boldsymbol{\psi}(t)$ in the time interval $[0, T]$.

A naive time-stepping numerical scheme for (5.94) requires a time step size $\delta t \sim \mathcal{O}(\tau)$. Thus, the computation will entail time updates $N \sim \mathcal{O}(1/\tau)$. Hence, one needs a multiscale ODE solver to approximate the solutions of the system (5.94). One such popular ODE solver can be derived by using the Heterogenous multiscale method (HMM); see Kuehn [2015] and references therein. This in turns, requires using two time stepping schemes, a *macro* solver for the slow variable, with a time step Δt of the form,

$$\boldsymbol{\psi}_n = \boldsymbol{\psi}_{n-1} + \tilde{\Delta} t g(\boldsymbol{\phi}_n, \boldsymbol{\psi}_{n-1}). \tag{5.95}$$

Here, the time step $\tilde{\Delta} t < 1$ is independent of the small parameter τ.

Moreover, the fast variable is updated using a *micro* solver of the form,

$$\begin{aligned} \boldsymbol{\phi}_{n-1}^{(k)} &= \boldsymbol{\phi}_{n-1}^{(k-1)} - \delta t(f(\boldsymbol{\psi}_{n-1}) - \boldsymbol{\phi}_{n-1}^{(k-1)}), \quad 1 \leq k \leq K. \\ \boldsymbol{\phi}_n &= \boldsymbol{\phi}_{n-1}^{K}, \\ \boldsymbol{\phi}_{n-1}^{(0)} &= \boldsymbol{\phi}_{n-1}. \end{aligned} \tag{5.96}$$

Note that the micro time step size δt and the number of micro time steps K are assumed to independent of the small parameter τ.

It is shown in Kuehn [2015] (Chapter 10.8) that for any given small tolerance $\epsilon > 0$, one can choose a macro time step $\tilde{\Delta} t$, a micro time step δt, the number K of micro time steps, the number N of macro time steps, independent of τ, such that the discrete states $\boldsymbol{\psi}_n$ approximate the slow-variable $\boldsymbol{\psi}(t_n)$ (with $t_n = n\tilde{\Delta} t$) of the fast-slow system (5.94) to the desired accuracy of ϵ.

Our aim is to show that we can construct a LEM of the form (5.3) such that the states $\boldsymbol{\phi}_n, \boldsymbol{\psi}_n$, defined in (5.95), (5.96) can be approximated to arbitrary accuracy. By combining this with the accuracy of HMM, we will prove that LEMs can approximate the solutions of the fast-slow system (5.94) to desired accuracy, independent of the small parameter τ in (5.94).

Proposition 5.2.10. *Let $\boldsymbol{\phi}_n, \boldsymbol{\psi}_n \in \mathbb{R}^{d_c}$, for $1 \leq n \leq N$, be the states defined by the HMM dynamical system* (5.95), (5.96). *For any given $\epsilon > 0$, there exists a LEM of the form* (5.3) *with hidden states $[\mathbf{z}_n, \mathbf{y}_n]$, where $\mathbf{z}_n, \mathbf{y}_n \in \mathbb{R}^{d_m}$ and output states ω_n^h, ω_n^c such that the following holds,*

$$\max\left\{\|\boldsymbol{\phi}_n - \omega_n^h\|, \|\boldsymbol{\psi}_n - \omega_n^c\|\right\} \leq \epsilon, \quad \forall 1 \leq n \leq N. \tag{5.97}$$

Proof. We start by using iteration on the micro solver (5.96) from $k = 1$ to $k = K$ to derive the following,

$$\begin{aligned} \boldsymbol{\phi}_n &= \overline{\delta t}\boldsymbol{\phi}_{n-1} + (1 - \overline{\delta t}) f(\boldsymbol{\psi}_{n-1}), \\ \overline{\delta t} &= (1 - \delta t)^K. \end{aligned} \tag{5.98}$$

As $\delta t < 1$, we have that $\overline{\delta t} < 1$.

By the universal approximation theorem for tanh neural networks, for any given tolerance ϵ^*, there exist weight matrices $W_1^f \in \mathbb{R}^{d_1 \times d_c}, W_2^f \in \mathbb{R}^{d_c \times d_1}$ and bias vector $b_1^f \in \mathbb{R}^{d_1}$ such that the tanh neural network defined by,

$$\mathcal{N}_f(c) = W_2^f \sigma\left(W_1^f c + b_1^f\right), \tag{5.99}$$

approximates the underlying function $\mathbf{f}$ in the following manner,

$$\max_{\|c\| < R^*} \|\mathbf{f}(c) - \mathcal{N}_f(c)\| \leq \epsilon^*. \tag{5.100}$$

Next, we define the following map,

$$\mathbf{G}(h, c) = \mathbf{g}(h, c) + c, \tag{5.101}$$

By the universal approximation theorem, given ϵ^*, there exist weight matrices $W_1^g, W_2^g \in \mathbb{R}^{d_2 \times d_c}, W_3^g \in \mathbb{R}^{d_c \times d_2}$ and bias vector $b_1^g \in \mathbb{R}^{d_2}$ such that the tanh neural network defined by,

$$\mathcal{N}_g(h,c) = W_3^g \sigma\left(W_1^g h + W_2^g c + b_1^g\right), \tag{5.102}$$

approximates the function $\mathbf{G}$ (5.101) in the following manner,

$$\max_{\max(\|h\|,\|c\|)<R^*} \|\mathbf{G}(h,c) - \mathcal{N}_g(h,c)\| \leq \epsilon^*. \tag{5.103}$$

Next, as in the proof of propositions 5.2.7 5.2.8, one can readily approximate the identity function $\hat{f}(h,c) = h$ with a tanh neural network of the form,

$$\bar{\mathcal{N}}_f(h) = \bar{W}_2 \sigma\left(\bar{W}_1 h\right), \tag{5.104}$$

such that

$$\max_{\|h\|,\|c\|<R^*} \|\hat{f}(h,c) - \mathcal{N}_f(h)\| \leq \epsilon^*, \tag{5.105}$$

and with the same weights and biases, one can approximate the identity function $\hat{g}(h,c) = c$ with the tanh neural network,

$$\bar{\mathcal{N}}_g(c) = \bar{W}_2 \sigma\left(\bar{W}_1 c\right), \tag{5.106}$$

such that

$$\max_{\|h\|,\|c\|<R^*} \|\hat{g}(h,c) - \mathcal{N}_g(c)\| \leq \epsilon^*. \tag{5.107}$$

Then, we define the following dynamical system,

$$\begin{aligned}
\hat{\mathbf{z}}_n &= \overline{\delta t}\hat{\mathbf{z}}_n + (1-\overline{\delta t}) W_2^f \sigma\left(W_1^f \hat{\mathbf{y}}_{n-1} + b_1^f\right), \\
\tilde{\mathbf{z}}_n &= \bar{W}_2 \sigma\left(\bar{W}_1 \hat{\mathbf{y}}_{n-1}\right), \\
\hat{\mathbf{y}}_n &= (1-\tilde{\Delta}t)\hat{\mathbf{y}}_{n-1} + \tilde{\Delta}t W_3^g \sigma\left(W_1^g \hat{\mathbf{z}}_n + W_2^g \tilde{\mathbf{z}}_n + b_1^g\right), \\
\tilde{\mathbf{y}}_n &= \bar{W}_2 \sigma\left(\bar{W}_1 \hat{\mathbf{z}}_n\right),
\end{aligned} \tag{5.108}$$

with hidden states $\hat{\mathbf{z}}_n, \tilde{\mathbf{z}}_n, \hat{\mathbf{y}}_n, \tilde{\mathbf{y}}_n \in \mathbb{R}^m$ and with initial states $\hat{\mathbf{z}}_0 = \tilde{\mathbf{z}}_0 = \hat{\mathbf{y}}_0 = \tilde{\mathbf{y}}_0 = 0$.

Completely analogously as in the derivation of (5.84), we can derive the following bound,

$$\|\boldsymbol{\phi}_n - \hat{\mathbf{z}}_n\| + \|\boldsymbol{\psi}_n - \hat{\mathbf{y}}_n\| \leq C^* \epsilon^*, \tag{5.109}$$

with constant $C^* = C^*\left(n, \mathrm{Lip}(f), \mathrm{Lip}(g)\right)$.

Defining the dynamical system,

$$\begin{aligned}
\mathbf{z}_n^* &= \overline{\delta t}\mathbf{z}_n^* + (1-\overline{\delta t})\sigma\left(W_1^f W_3^g \hat{\mathbf{y}}_{n-1} + b_1^f\right) \\
\bar{\mathbf{z}}_n &= \sigma\left(\bar{W}_1 W_3^g \mathbf{y}_{n-1}^*\right) \\
\mathbf{y}_n^* &= (1-\tilde{\Delta}t)\mathbf{y}_{n-1}^* + \tilde{\Delta}t\sigma\left(W_1^g W_3^f \mathbf{z}_n^* + W_2^g \bar{W}_2 \tilde{\mathbf{z}}_n + b_1^g\right) \\
\bar{\mathbf{y}}_n &= \sigma\left(\bar{W}_1 W_2^f \mathbf{z}_n^*\right).
\end{aligned} \tag{5.110}$$

By multiplying suitable matrices to (5.110), we obtain that,

$$\hat{\mathbf{z}}_n = W_2^f \mathbf{z}_n^*, \quad \tilde{\mathbf{z}}_n = \bar{W}_2 \bar{\mathbf{z}}_n, \quad \hat{\mathbf{y}}_n = W_3^g \mathbf{y}_n^*, \quad \tilde{\mathbf{y}}_n = \bar{W}_2 \bar{\mathbf{y}}_n. \tag{5.111}$$

In addition to b_∞ defined in (5.53), for $\overline{\delta t} \in (0, 1]$, we introduce $b_\delta \in \mathbb{R}$ defined by

$$\hat{\sigma}(b_\delta) = \overline{\delta t}. \tag{5.112}$$

Similarly for $\tilde{\Delta} t \in (0, 1]$, we introduce $b_\Delta \in \mathbb{R}$ defined by

$$\hat{\sigma}(b_\Delta) = \tilde{\Delta} t. \tag{5.113}$$

The existence of unique b_δ and b_Δ follows from the fact that the sigmoid function $\hat{\sigma}$ is monotone.

Next, we define the two vectors $\mathbf{b}_\infty, \mathbf{b}_\delta, \mathbf{b}_\Delta \in \mathbb{R}^{2d_c}$ as

$$\begin{aligned} \mathbf{b}_\delta^i &= b_\delta, \quad \forall\ 1 \leq i \leq d_c, \\ \mathbf{b}_\delta^i &= b_\infty, \quad \forall\ d_c + 1 \leq i \leq 2d_c, \\ \mathbf{b}_\Delta^i &= b_\Delta, \quad \forall\ 1 \leq i \leq d_c, \\ \mathbf{b}_\Delta^i &= b_\infty, \quad \forall\ d_c + 1 \leq i \leq 2d_c. \end{aligned} \tag{5.114}$$

We define the LEM of form (5.3), which will approximate the HMM (5.95),(5.96). To this end, we define the hidden states $\mathbf{z}_n, \mathbf{y}_n \in \mathbb{R}^{2d_c}$ such that $\mathbf{z}_n = [\mathbf{z}_n^*, \bar{\mathbf{z}}_n^*]$ and $\mathbf{y}_n = [\mathbf{y}_n^*, \bar{\mathbf{y}}_n^*]$. The parameters for the corresponding LEM of form (5.3) given by,

$$\begin{aligned} &\Delta t = 1, d_y = 2d_c \\ &\mathbf{W}_1 = \mathbf{W}_2 = \mathbf{V}_1 = \mathbf{V}_2 \equiv 0, \\ &\mathbf{b}_1 = \mathbf{b}_\delta, \quad \mathbf{b}_2 = \mathbf{b}_\Delta, \\ &\mathbf{W}_z = \begin{bmatrix} W_1^f W_3^g & 0 \\ \bar{W}_1 W_3^g & 0 \end{bmatrix}, \quad \mathbf{V}_z = 0, \quad \mathbf{b}_z = [b_1^f, 0], \\ &\mathbf{W}_y = \begin{bmatrix} W_1^g W_3^f & W_2^g \bar{W}_2 \\ \bar{W}_1 W_2^f & 0 \end{bmatrix}, \quad \mathbf{V}_z = 0, \quad \mathbf{b}_z = [b_1^g, 0]. \end{aligned} \tag{5.115}$$

The output states are defined by,

$$\omega_n^h = W_2^f \mathbf{z}_n^*, \quad \omega_n^h = W_3^g \mathbf{y}_n^* \tag{5.116}$$

It is straightforward to observe that $\omega_n^h = \hat{\mathbf{z}}_n,\ \omega_n^c = \hat{\mathbf{y}}_n$. Hence, the desired bound (5.97) follows from (5.109) by choosing,

$$\epsilon^* = \frac{\epsilon}{C^*}.$$

□

5.3 Empirical results

We present a variety of experiments ranging from long-term dependency tasks to real-world applications as well as tasks which require high expressivity of the model. Details of the training procedure for each experiment can be found at the end of this section. As competing models to LEM, we choose two different types of architectures—LSTMs and GRUs—as they are known to excel at expressive tasks such as language modeling and speech recognition, while not performing well on long-term dependency tasks, possibly due to the exploding and vanishing gradients problem. On the other hand, we choose state-of-the-art RNNs which are tailor-made to learn tasks with long-term dependencies. Our objective is to evaluate the performance of LEM and compare it with competing models.

Very long adding problem. We start with the well-known adding problem [Hochreiter and Schmidhuber, 1997], proposed to test the ability of a model to learn (very) long-term dependencies. The input is a two-dimensional sequence of length N, with the first dimension consisting of random numbers drawn from $\mathcal{U}([0, 1])$ and with two non-zero entries (both set to 1) in the second dimension, chosen at random locations, but one each in both halves of the sequence. The output is the sum of two numbers of the first dimension at positions, corresponding to the two 1 entries in the second dimension. We consider three very challenging cases, namely input sequences with length $N = 2000, 5000$ and 10000. The results of LEM together with competing models including state-of-the-art RNNs, which are explicitly designed to solve long-term dependencies, are presented in Fig. 5.1. We observe in this figure that while baseline LSTM is not able to beat the baseline mean-square error of 0.167 (the variance of the baseline output 1) in any of the three cases, a proper weight initialization for LSTM, the so-called *chrono*-initialization of Tallec and Ollivier [2018] leads to much better performance in all cases. For $N = 2000$, all other architectures (except baseline LSTM) beat the baseline convincingly. However for $N = 5000$, only LEM, chrono-LSTM and coRNN are able to beat the baseline. In the extreme case of $N = 10000$, only LEM and chrono-LSTM are able to beat the baseline. Nevertheless, LEM outperforms chrono-LSTM by converging faster (in terms of number of training steps) and attaining a lower test MSE than chrono-LSTM in all three cases.

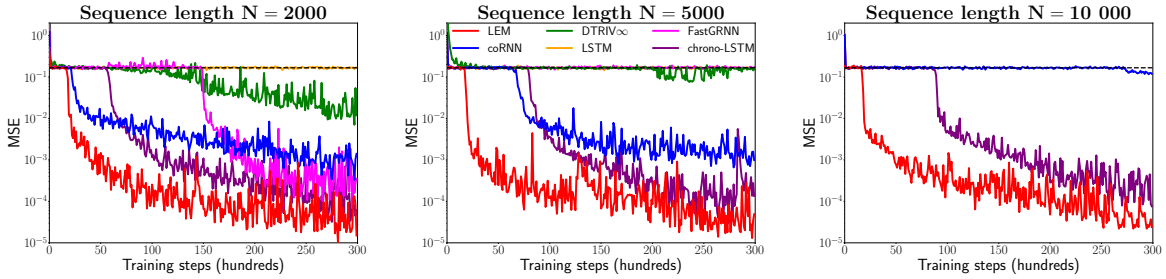

Figure 5.1: Results on the very long adding problem for LEM, coRNN, DTRIV∞ [Casado, 2019], FastGRNN [Kusupati et al., 2018], LSTM and LSTM with chrono initialization [Tallec and Ollivier, 2018] based on three very long sequence lengths N, i.e., $N = 2000$, $N = 5000$ and $N = 10000$.

Sequential image recognition. We consider three experiments based on two widely-used image recognition data sets, i.e., MNIST [LeCun et al., 1998] and CIFAR-10 [Krizhevsky et al., 2009], where the goal is to predict the correct label after reading in the whole sequence. The first two tasks are based on MNIST images, which are flattened along the rows to obtain sequences of length $N = 784$. In sequential MNIST (sMNIST), the sequences are fed to the model one pixel at a time in streamline order, while in permuted sequential MNIST (psMNIST), a fixed random permutation is applied to the sequences, resulting in much longer dependency than for sMNIST. We also consider the more challenging noisy CIFAR-10 (nCIFAR-10) experiment [Chang et al., 2019], where CIFAR-10 images are fed to the model row-wise and flattened along RGB channels, resulting in 96-dimensional sequences, each of length 32. Moreover, a random noise padding is applied after the first 32 inputs to produce sequences of length $N = 1000$. Hence, in addition to classifying the underlying image, a model has to store this result for a long time. In Table 5.1, we present the results for LEM on the three tasks together with other state-of-the-art RNNs, which were explicitly designed to solve long-term dependency tasks, as well as LSTM and GRU baselines. We observe that LEM outperforms all other methods on sMNIST and nCIFAR-10. Additionally on psMNIST, LEM performs as well as coRNN, which has been state-of-the-art

among single-layer RNNs on this task.

Table 5.1: Test accuracies on sMNIST, psMNIST and nCIFAR-10, where M denotes the total number of parameters of the corresponding model. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	MNIST			CIFAR-10	
	sMNIST	psMNIST	# units / M	nCIFAR-10	# units / M
GRU	99.1%	94.1%	256 / 201k	43.8%	128 / 88k
LSTM	98.9%	92.9%	256 / 267k	11.6%	128 / 116k
chrono-LSTM	98.9%	94.6%	128 / 68k	55.9%	128 / 116k
anti.sym. RNN	98.0%	95.8%	128 / 10k	48.3%	256 / 36k
Lipschitz RNN	**99.4%**	**96.3%**	128 / 34k	**57.4%**	128 / 46k
expRNN	98.4%	**96.2%**	360 / 69k	52.9%	360 / 103k
coRNN	**99.3%**	**96.6%**	128/ 34k	**59.0%**	128 / 46k
LEM	**99.5%**	**96.6%**	128 / 68k	**60.5%**	128 / 116k

EigenWorms: Very long sequences for genomics classification. The goal of this task [Bagnall et al., 2018] is to classify worms as belonging to either the wild-type or four different mutants, based on 259 very long sequences (length $N = 17984$) measuring the motion of a worm. In addition to the nominal length, it was empirically shown in Chapter 3 that the EigenWorms sequences exhibit actual very long-term dependencies (i.e., longer than 10k).

Following Morrill et al. [2020] and Chapter 3, we divide the data into a train, validation and test set according to a $70\%, 15\%, 15\%$ ratio. In Table 5.2, we present results for LEM together with other models. As the validation and test sets, each consist of only 39 sequences, we report the mean (and standard deviation of) accuracy over 5 random initializations to rule out lucky outliers. We observe from this table that LEM outperforms all other methods, even the 2-layer UnICORNN architecture, which has been state-of-the-art on this task.

Table 5.2: Test accuracies on EigenWorms using 5 re-trainings of each best performing network (based on the validation set), where all other results are taken from Chapter 3 except that the NRDE result is taken from Morrill et al. [2020] and the results indicated by * are added by us. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
NRDE	$83.8\% \pm 3.0\%$	32	35k
expRNN	$40.0\% \pm 10.1\%$	64	2.8k
IndRNN (2 layers)	$49.7\% \pm 4.8\%$	32	1.6k
LSTM	$38.5\% \pm 10.1\%$*	32	5.3k
BiLSTM+1d-conv	$40.5\% \pm 7.3\%$*	22	5.8k
chrono-LSTM	$82.6\% \pm 6.4\%$*	32	5.3k
coRNN	$\mathbf{86.7\% \pm 3.0\%}$	32	2.4k
UnICORNN (2 layers)	$\mathbf{90.3\% \pm 3.0\%}$	32	1.5k
LEM	$\mathbf{92.3\% \pm 1.8\%}$	32	5.3k

Healthcare application: Heart-rate prediction. In this experiment, one predicts the heart rate from a time-series of measured PPG data, which is part of the TSR archive [Tan et al., 2020] and has been collected at the Beth Isreal Deaconess medical center. The data set, consisting of 7949 sequences, each of length $N = 4000$, is divided into a train, validation and test set according to a 70%,15%,15% ratio (i.e., same es in Section 3.3). The results, presented in Table 5.3, show that LEM outperforms the other competing models, including having a test L^2 error of 35% less than the state-of-the-art UnICORNN.

Table 5.3: Test L^2 error on heart-rate prediction using PPG data. All results are obtained by running the same code and using the same fine-tuning protocol.

Model	test L^2 error	# units	# parameters
LSTM	9.93	128	67k
chrono-LSTM	3.31	128	67k
expRNN	1.63	256	34k
IndRNN (3 layers)	1.94	128	34k
coRNN	**1.61**	128	34k
UnICORNN (3 layers)	**1.31**	128	34k
LEM	**0.85**	128	67k

Multiscale dynamical system prediction. The FitzHugh-Nagumo system [Fitzhugh, 1955]

$$v' = v - \frac{v^3}{3} - w + I_{\text{ext}}, \quad w' = \tau(v + a - bw), \tag{5.117}$$

is a prototypical model for a two-scale fast-slow nonlinear dynamical system, with fast variable v and slow variable w and $\tau << 1$ determining the slow-time scale. This *relaxation-oscillator* is an approximation to the Hodgkin-Huxley model [Hodgkin and Huxley, 1952] of neuronal action-potentials under an external signal $I_{\text{ext}} \geq 0$. With $\tau = 0.02$, $I_{\text{ext}} = 0.5$, $a = 0.7$, $b = 0.8$ and initial data $(v_0, w_0) = (c, 0)$, with c randomly drawn from $\mathcal{U}([-1, 1])$, we numerically approximate (5.117) with the explicit Runge-Kutta method of order 5(4) in the interval $[0, 400]$ and generate 128 training and validation and 1024 test sequences, each of length $N = 1000$, to complete the data set. The results, presented in Table 5.4, show that LEM not only outperforms LSTM by a factor of 6 but also all other methods including coRNN, which is tailormade for oscillatory time-series. This reinforces our theory by demonstrating efficient approximation of multiscale dynamical systems with LEM.

Table 5.4: Test L^2 error on FitzHugh-Nagumo system prediction. All results are obtained by running the same code and using the same fine-tuning protocol. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	error ($\times 10^{-2}$)	# units	# parameters
LSTM	**1.2**	16	1k
expRNN	2.3	50	1k
LipschitzRNN	1.8	24	1k
coRNN	**0.4**	24	1k
LEM	**0.2**	16	1k

Google12 (V2) keyword spotting. The Google Speech Commands data set V2 [Warden, 2018] is a widely used benchmark for keyword spotting, consisting of 35 words, sampled at a rate of 16 kHz from 1 second utterances of 2618 speakers. We focus on the 12-label task (Google12) and follow the pre-defined splitting of the data set into train/validation/test sets and test different sequential models. In order to ensure comparability of different architectures, we do not use performance-enhancing tools such as convolutional filtering or multi-head attention. From Table 5.5, we observe that both LSTM and GRU, widely used models in this context, perform very well with a test accuracy of around 95%. Nevertheless, LEM is able to outperform both on this task and provides the best performance.

Table 5.5: Test accuracies on Google12. All results are obtained by running the same code and using the same fine-tuning protocol. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test accuracy	# units	# parameters
tanh-RNN	73.4%	128	27k
anti.sym. RNN	90.2%	128	20k
LSTM	**94.9%**	128	107k
GRU	**95.2%**	128	80k
FastGRNN	94.8%	128	27k
expRNN	92.3%	128	19k
coRNN	94.7%	128	44k
LEM	**95.7%**	128	107k

Language modeling: Penn Tree Bank corpus. Language modeling with the widely used small scale Penn Treebank (PTB) corpus [Marcus et al., 1993], preprocessed by Mikolov et al. [2010], has been identified as an excellent task for testing the expressivity of recurrent models [Kerg et al., 2019]. To this end, in Table 5.6, we report the results of different architectures, with a similar number of hidden units, on the PTB char-level task and observe that RNNs, designed explicitly for learning long-term dependencies, perform significantly worse than LSTM and GRU. On the other hand, LEM is able to outperform both LSTM and GRU on this task by some margin (a test bpc of 1.25 in contrast with approximately a bpc of 1.36). In fact, LEM provides the smallest test bpc among all reported single-layer recurrent models on this task, to the best of our knowledge. This superior performance is further illustrated in Table 5.7, where the test perplexity for different models on the PTB word-level task is presented. We observe that not only does LEM significantly outperform (by around 40%) LSTM, but it also provides again the best performance among all single layer recurrent models, including the recently proposed TARNN [Kag and Saligrama, 2021]. Moreover, the single-layer results for LEM are better than reported results for multi-layer LSTM models, such as in Gal and Ghahramani [2016] (2-layer LSTM, 1500 units each: 75.2 test perplexity) or Bai et al. [2018] (3-layer LSTM, 700 units each: 78.93 test perplexity).

Training details. All experiments were run on CPU, namely Intel Xeon Gold 5118 and AMD EPYC 7H12, except for Google12, PTB character-level and PTB word-level, which were run on a GeForce RTX 2080 Ti GPU. All weights and biases of LEM (5.3) are initialized according to $\mathcal{U}(-1/\sqrt{m}, 1/\sqrt{m})$, where m is the number of hidden units.

The hyperparameters are selected based on a random search algorithm, where we present the rounded hyperparameters for the best performing LEM model (*based on a validation set*) on each task in Table 5.8.

Table 5.6: Test bits-per-character (bpc) on PTB character-level for single layer LEM and other single layer RNN architectures. Other results are taken from the papers cited accordingly in the table, while the results for coRNN are added by us. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test bpc	# units	# parameters
anti.sym RNN [Erichson et al., 2020]	1.60	1437	1.3M
Lipschitz RNN [Erichson et al., 2020]	1.42	764	1.3M
expRNN [Kerg et al., 2019]	1.51	1437	1.3M
coRNN	1.46	1024	2.3M
nnRNN [Kerg et al., 2019]	1.47	1437	1.3M
LSTM [Krueger et al., 2017]	**1.36**	1000	5M
GRU [Bai et al., 2018]	**1.37**	1024	3M
LEM	**1.25**	1024	5M

Table 5.7: Test perplexity on PTB word-level for single layer LEM and other single layer RNN architectures. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	test perplexity	# units	# parameters
Lipschitz RNN [Erichson et al., 2020]	115.4	160	76k
FastRNN [Kag and Saligrama, 2021]	115.9	256	131k
LSTM [Kag and Saligrama, 2021]	116.9	256	524k
SkipLSTM [Kag and Saligrama, 2021]	**114.2**	256	524k
TARNN [Kag and Saligrama, 2021]	**94.6**	256	524k
LEM	**72.8**	256	524k

We base the training for the PTB experiments on the following language modelling code: https://github.com/deepmind/lamb, where we fine-tune, based on a random search algorithm, only the learning rate, input-, output- and state-dropout, L^2-penalty term and the maximum gradient norm.

We train LEM for 100 epochs on sMNIST, psMNIST and nCIFAR-10, after which we decrease the learning rate by a factor of 10 and proceed training for 20 epochs. Moreover, we train LEM for 50, 60 as well as 400 epochs on EigenWorms, Google12 and FitzHugh-Nagumo. We decrease the learning rate by a factor of 10 after 50 epochs on Google12. On the Healthcare task, we train LEM for 250 epochs, after which we decrease the learning rate by a factor of 10 and proceed training for 250 epochs.

5.4 Further empirical analysis

On the choice of the hyperparameter Δt. The hyperparameter Δt in LEM (5.3) measures the maximum allowed (time) step in the discretization of the multiscale ODE system (5.2). In propositions 5.2.1, 5.2.2 and 5.2.5, this hyperparameter Δt plays a key role in the bounds on the hidden states (5.9) and their gradients (5.18). In particular, setting $\Delta t = \mathcal{O}(N^{-1})$ will lead to hidden states and gradients, that are bounded uniformly with respect to the underlying sequence length N. However, these upper bounds on the hidden states and gradients account for *worst-case* scenarios and can be very pessimistic for the problem at hand. Thus, in practice, we determine Δt through a hyperparameter tuning procedure. To this end, we perform a random search within $\Delta t < 2$ and present the resulting optimal values of Δt

Table 5.8: Rounded hyperparameters of the best performing LEM architecture for each experiment. If no value is given for Δt, it means that Δt is fixed to 1 and no fine-tuning is performed on this hyperparameter.

experiment	learning rate	batch size	Δt
Adding ($N = 10000$)	2.6×10^{-3}	50	2.42×10^{-2}
sMNIST	1.8×10^{-3}	128	2.1×10^{-1}
psMNIST	3.5×10^{-3}	128	1.9×10^{0}
nCIFAR-10	1.8×10^{-3}	120	9.5×10^{-1}
EigenWorms	2.3×10^{-3}	8	1.6×10^{-3}
Healthcare	1.56×10^{-3}	32	1.9×10^{-1}
FitzHugh-Nagumo	9.04×10^{-3}	32	/
Google12	8.9×10^{-4}	100	/
PTB character-level	6.6×10^{-4}	128	/
PTB word-level	6.8×10^{-4}	64	/

for each of the considered data sets in Table 5.8. From this table, we observe that for data sets such as PTB, FitzHugh-Nagumo and Google 12 we do not need any tuning of Δt and a default value of $\Delta t = 1$ resulted in very good empiricial performance. On the other data sets such as sMNIST, nCIFAR-10 and the healthcare example, where the sequence length ($N = \mathcal{O}(10^3)$) is larger, we observe that values of $\Delta t \approx 0.1$ yielded the best performance. The notable exception to this was for the EigenWorms data set, with a very long sequence length of $N = 17984$ as well as demonstrated very long range dependencies in the data, see Chapter 3. Here, a value of $\Delta t = 1.6 \times 10^{-3}$ resulted in the best observed performance. To further investigate the role of the hyperparameter Δt in the EigenWorms experiment, we perform a sensitivity study where the value of Δt is varied and the corresponding accuracy of the trained LEM is observed. The results of this sensitivity study are presented in Fig. 5.2, where we plot the test accuracy (Y-axis) vs. the value of Δt (X-axis). From this figure, we observe that the accuracy is rather poor for $\Delta t \approx 1$ but improves monotonically as Δt is reduced till a value of approximately 10^{-2}, after which it saturates. Thus, in this case, a value of $\Delta t = \mathcal{O}(N^{-\frac{1}{2}})$ (for sequence length N) suffices to yield the best empirical performance.

Given this observation, we further test whether $\Delta t = \mathcal{O}(N^{-\frac{1}{2}})$ suffices for other problems with long-term dependencies. To this end, we consider the adding problem and vary the input sequence length by an order of magnitude, i.e., from $N = 250$ to $N = 2000$. The value of Δt is now fixed at $\Delta t = \frac{1}{\sqrt{N}}$ and the resulting test loss (Y-axis) vs the number of training steps (X-axis) is plotted in Fig. 5.3. We see from this figure that this value of Δt sufficed to yield very small average test errors for this problem for all considered sequence lengths N. Thus, empirically a value of Δt in the range $\frac{1}{\sqrt{N}} \leq \Delta t \leq 1$ yields very good performance.

Even if we set $\Delta t = \mathcal{O}(\frac{1}{\sqrt{N}})$, it can happen for very long sequences $N >> 1$ that the gradient can be quite small from the gradient asymptotic formula (5.45). This might lead to saturation in training, resulting in long training times. However, we do not observe such long training times for very long sequence lengths in our experiment. To demonstrate this, we again consider Fig. 5.3 where the number of training steps (X-axis) is plotted for sequence lengths that vary an order of magnitude. The figure clearly shows that the approximately the same number of training steps are needed to attain a low test error, irrespective of the sequence length. This is further buttressed in Fig. 5.1, where similar number of training steps where needed for obtaining the same very low test error, even for long sequence lengths, with N up to 10000. Moreover, from the training details in section 5.3, we see that the number of epochs for different data sets is independent of the sequence length. For instance, only 50 epochs were necessary

for EigenWorms with a sequence length of $N = 17984$ and $\Delta t = 1.6 \times 10^{-3}$ whereas 400 epochs were required for the FitzHugh-Nagumo system with a $\Delta t = 1$.

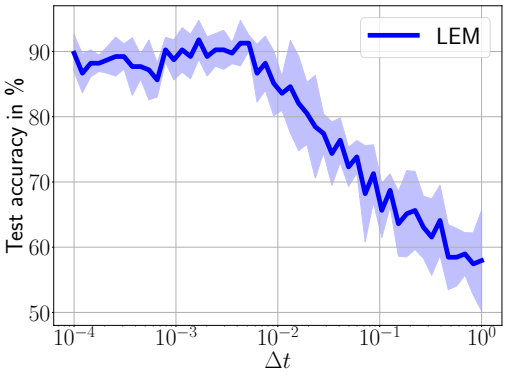

Figure 5.2: Sensitivity study on hyperparameter Δt in (5.3) using the EigenWorms experiment.

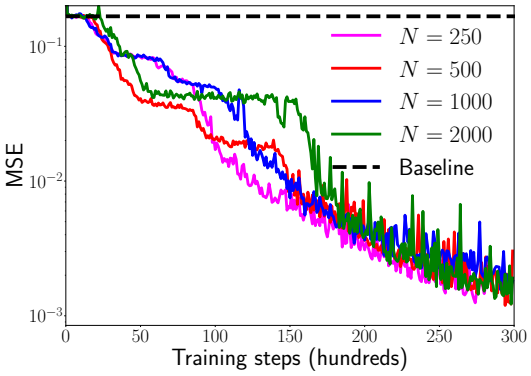

Figure 5.3: Average (over ten different initializations each) test mean-square error on the adding problem of LEM for different sequence lengths N, where the hyperparameter Δt of LEM (5.3) is fixed to $\Delta t = 1/\sqrt{N}$.

Multiscale Behavor of LEM. LEM (5.3) is designed to represent multiple scales, with terms $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ being explicitly designed to learn possible multiple scales. In the following, we will investigate if in practice, LEM learns multiple scales and uses them to yield the observed superior empirical performance with respect to competing models.

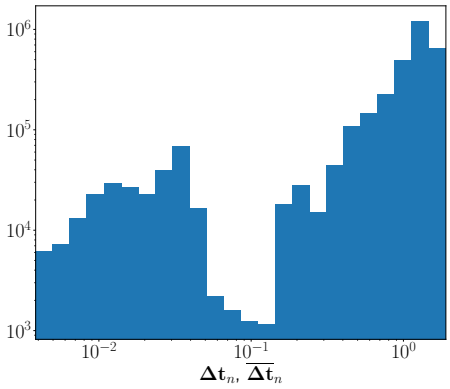

Figure 5.4: Histogram of $(\mathbf{\Delta t}_n)_i$ and $(\overline{\mathbf{\Delta t}}_n)_i$ for all $n = 1, \ldots, N$ and $i = 1, \ldots, m$ of LEM (5.3) after training on the FitzHugh-Nagumo fast-slow system (5.117) using $\Delta t = 2$.

To this end, we start by recalling the proposition 5.2.8 where we showed that in principle, LEM can learn the two underlying timescales of a *fast-slow* dynamical system (see proposition 5.2.9 for a similar result for the universal approximation of a r-time scale (with $r \geq 2$) dynamical system with LEM). Does

this hold in practice ? To further investigate this issue, we consider the FitzHugh-Nagumo dynamical system (5.117) which serves as a prototype for a two-scale dynamical system. We consider this system (5.117) with the two time-scales being $\tau = 0.02$ and 1 and train LEM for this system. In Fig. 5.4, we plot the empirical histogram that bins the ranges of learned scales $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n \leq \Delta t = 2$ (for all n and m) and counts the number of occurrences of $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ in each bin. From this figure, we observe that there is a clear concentration of learned scales around the values 1 and $\tau = 0.02$, which exactly correspond to the underlying fast and slow time scales. Thus, for this model problem, LEM is exactly learning what it is designed to do and is able to learn the underlying time scales for this particular problem.

Nevertheless, one might argue that these learnable mutliple scales $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ are not necessary and a single scale would suffice to provide good empirical performance. We check this possibility on the FitzHugh-Nagumo data set by simply setting $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n \equiv \Delta t \mathbf{1}$ (with $\mathbf{1}$ being the vector with all entries set to 1), for all n and tuning the hyperparameter Δt. The comparative results are presented in Table 5.9. We see from this table by not allowing for learnable $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ and simply setting them to a single scale parameter Δt and tuning this parameter only leads to results that are comparable to the baseline LSTM model. On the other hand, learning $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ resulted in an error that is a factor of 6 less than the baseline LSTM test error. Thus, we demonstrate the importance of the ability of the proposed LEM model to learn multiple scales in this example.

Table 5.9: Test L^2 error on FitzHugh-Nagumo system prediction.

Model	error ($\times 10^{-2}$)	# units	# params
LSTM	1.2	16	1k
LEM w/o multiscale	1.1	16	1k
LEM	0.2	16	1k

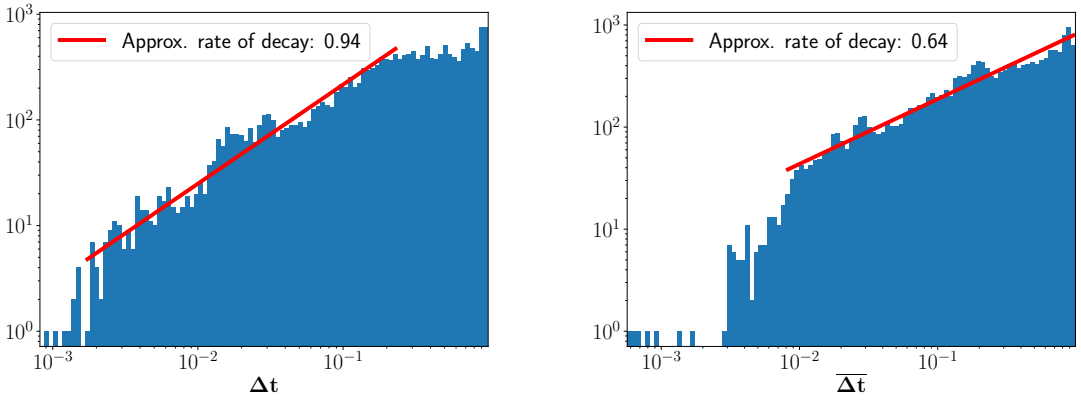

Figure 5.5: Histogram of $(\mathbf{\Delta t}_n)_i$ and $(\overline{\mathbf{\Delta t}}_n)_i$ for all $n = 1, \ldots, N$ and $i = 1, \ldots, m$ of LEM (5.3) after training on the Google12 data set

Hence, the multiscale resolution of LEM seems essential for the fast-slow dynamical system. Does this multiscale resolution also appear for other datasets and can it explain aspects of the observed empirical performance ? To this end, we consider the Google12 Keyword spotting data set and start by pointing out that given the spatial (with respect to hidden dimension m) and temporal (with respect to sequence length

N) heterogeneities, a priori, it is unclear if the underlying data has a multiscale structure. We plot the empirical histograms of $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ in Fig. 5.5 to observe that even for this problem, the terms $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ are expressed over a range of scales, amounting to $2-3$ orders of magnitude. Thus, a range of scales are present in the trained LEM even for this example, but do they affect the empirical performance of LEM ? We investigate this question by performing an ablation study and reporting the results in Fig. 5.6. In this study, we clip the values of $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ to lie within the range $[2^{-i}, 1]$, for $i = 0, 1, \ldots, 7$ and plot the statistics of the observed test accuracy of LEM. We observe from Fig. 5.6 that by clipping $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ to lie near the default (single scale) value of 1 results in very poor empirical performance of an accuracy of $\approx 65\%$. Then the accuracy jumps to around 90% when an order of magnitude range for $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ is considered, before monotonically and slowly increasing to yield the best empirical performance for the largest range of values of $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$, considered in this study. A closer look at the empirical histograms plotted in Fig. 5.5 reveal that the proportion of occurrences of $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$ decays as a *power law*, and not exponentially, with respect to the scale amplitude. This, together with results presented in Fig. 5.6 suggest that not only do a range of scales occur in learned $\mathbf{\Delta t}_n, \overline{\mathbf{\Delta t}}_n$, the small scales also contribute proportionately to the dynamics and enable the increase in performance shown in Fig. 5.6.

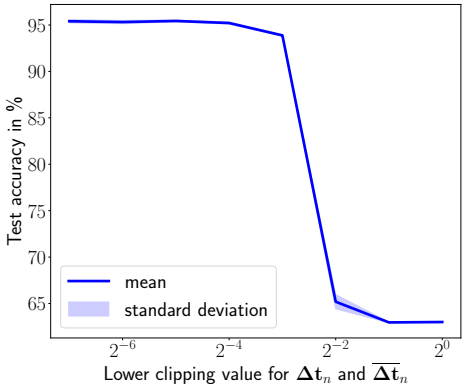

Figure 5.6: Average (and standard deviation of) test accuracies of 5 runs each for LEM on Google12, where $\mathbf{\Delta t}_n$ and $\overline{\mathbf{\Delta t}}_n$ in (5.3) are clipped to the ranges $[\frac{1}{2^i}, 1]$ for $i = 0, \ldots, 7$ during training.

Finally, in Fig. 5.7, we plot the empirical histograms of $\mathbf{\Delta t}_n$ and $\overline{\mathbf{\Delta t}}_n$ for the learned LEM on the sMNIST data set to observe that again a range of scales are observed and the observed occurrences of $\mathbf{\Delta t}_n$ and $\overline{\mathbf{\Delta t}}_n$ at each scale decays as a power law with respect to scale amplitude. Hence, we have sufficient empirical evidence to claim that the multiscale resolution of LEM seems essential to its observed performance. However, further investigation is required to elucidate the precise mechanisms through this multiscale resolution enables superior performance, particularly on problems where the multiscale structure of the underlying data may not be explicit.

On gradient-stable initialization. Specialized weight initialization is a popular tool to increase the performance of RNNs on long-term dependency tasks. One particular approach is the so-called chrono initialization [Tallec and Ollivier, 2018] for LSTMs, where all biases are set to zero except for the bias of the forget gate as well as the input gate ($\mathbf{b}_f$ and $\mathbf{b}_i$ in the LSTM (5.6)), which are sampled from

$$\begin{aligned} \mathbf{b}_f &\sim \log(\mathcal{U}[1, T_{\max} - 1]) \\ \mathbf{b}_i &= -\mathbf{b}_f, \end{aligned}$$

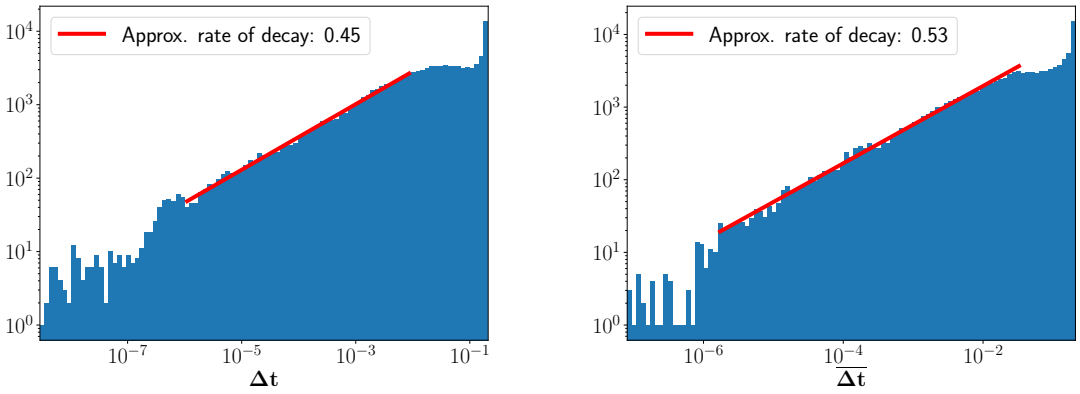

Figure 5.7: Histogram of $(\mathbf{\Delta t}_n)_i$ and $(\overline{\mathbf{\Delta t}}_n)_i$ for all $n = 1, \ldots, N$ and $i = 1, \ldots, m$ of LEM (5.3) after training on the sMNIST data set

where $T_{\max}$ denotes the maximal temporal dependency of the underlying sequential data. We can see in Table 5.2 that the chrono initialization significantly improves the performance of LSTM on the EigenWorms task. Hence, we are interested in extending the chrono initialization to LEMs. One possible manner for doing this is as follows: Initialize all biases of LEM to zero except for $\mathbf{b}_1$ in (5.3), which is sampled from

$$\mathbf{b}_1 \sim -\log(\mathcal{U}[1, T_{\max}\Delta t - 1]).$$

Table 5.10: Test accuracies on EigenWorms using 5 re-trainings of each best performing network (based on the validation set), where we train LSTM and LEM with and without chrono intialization, as well as LEM without chrono initialization but with tuned Δt.

Model	test accuracy	# units	# params	chrono	tuning Δt
LSTM	38.5% ± 10.1%	32	5.3k	NO	/
LSTM	82.6 % ± 6.4%	32	5.3k	YES	/
LEM	57.9% ± 7.7%	32	5.3k	NO	NO
LEM	88.2% ± 6.9%	32	5.3k	YES	NO
LEM	92.3% ± 1.8%	32	5.3k	NO	YES

We test the chrono initialization for LEM on the EigenWorms dataset, where we train LEM (without tuning Δt, i.e., setting $\Delta t = 1$), with and without chrono initialization. We provide the results in Table 5.10, where we show again the results of LSTM with and without chrono initialization as well as the LEM result with tuned Δt and without chrono initialization from Table 5.2 for comparison. We see from Table 5.10 that when Δt is fixed to 1, the chrono initialization significantly improves the result of LEM. However, if we tune Δt, but do not use the chrono initialization, we significantly improve the performance of LEM again. We further remark that tuning Δt as well as using chrono initialization for LEM does not improve the results obtained with simply tuning Δt in LEM. Thus, we conclude that chrono initialization can successfully be adapted to LEM. However, tuning Δt (which controls the gradients) is still advisable in order to obtain the best possible results.

5.5 Discussion

The design of a gradient-based model for processing sequential data that can learn tasks with long-term dependencies while retaining the ability to learn complicated sequential input-output maps is very challenging. In this chapter, we have proposed *Long Expressive Memory* (LEM), a novel recurrent architecture, with a suitable time-discretization of a specific multiscale system of ODEs (5.2) serving as the circuit to the model. By a combination of theoretical arguments and extensive empirical evaluations on a diverse set of learning tasks, we demonstrate that LEM is able to learn long-term dependencies while retaining sufficient expressivity for efficiently solving realistic learning tasks.

It is natural to ask why LEM performs so well. A part of the answer lies in the mitigation of the exploding and vanishing gradients problem. Proofs for gradient bounds (5.18),(5.45) reveal a key role played by the hyperparameter Δt. We observe from Table 5.8 that small values of Δt might be needed for problems with very long-term dependencies, such as the EigenWorms dataset. On the other hand, no tuning of the hyperparameter Δt is necessary for several tasks such as language modeling, keyword spotting and dynamical systems prediction and a default value of $\Delta t = 1$ yielded very good performance. The role and choice of the hyperparameter Δt is investigated extensively in section 5.4. However, mitigation of exploding and vanishing gradients problem alone does not explain high expressivity of LEM. In this context, we proved that LEMs can approximate a very large class of multiscale dynamical systems. Moreover, we provide experimental evidence in section 5.4 to observe that LEM not only expresses a range of scales, as it is designed to do, but also these scales contribute proportionately to the resulting multiscale dynamics. Furthermore, empirical results presented in section 5.4 show that this ability to represent multiple scales correlates with the high accuracy of LEM. We believe that this combination of gradient stable dynamics, specific model structure, and its multiscale resolution can explain the observed performance of LEM.

We conclude with a comparison of LEM and the widely-used gradient-based LSTM model. In addition to having exactly the same number of parameters for the same number of hidden units, our experiments show that LEMs are better than LSTMs on expressive tasks such as keyword spotting and language modeling, while also providing significantly better performance on long-term dependencies. This robustness of the performance of LEM with respect to sequence length paves the way for its application to learning many different sequential data sets where competing models might not perform satisfactorily.

Part II

Physics-inspired Graph Representation Learning

Chapter 6

Introduction to Graph Representation Learning

Learning tasks involving graph structured data arise in a wide variety of problems in science and engineering. Graph Neural Networks (GNNs) [Sperduti, 1994, Goller and Kuchler, 1996, Sperduti and Starita, 1997, Frasconi et al., 1998, Gori et al., 2005, Scarselli et al., 2008, Bruna et al., 2014, Defferrard et al., 2016, Kipf and Welling, 2017, Monti et al., 2017, Gilmer et al., 2017] are a popular deep learning architecture for graph-structured and relational data. GNNs have been successfully applied in domains including computer vision and graphics [Monti et al., 2017], recommender systems [Ying et al., 2018], transportation [Derrow-Pinion et al., 2021], computational chemistry [Gilmer et al., 2017], drug discovery [Gaudelet et al., 2021], particle physics [Shlomi et al., 2020] and social networks. See Zhou et al. [2019], Bronstein et al. [2021] for extensive reviews.

Modern GNNs process graph-structured data based on the so-called message-passing framework, which can be described as propagating node features on the underlying graph by exchanging (passing) information among adjacent nodes. More concretely, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $|\mathcal{V}| = v$ nodes and $|\mathcal{E}| = e$ edges (i.e., unordered pairs of nodes $\{i, j\} \in \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ we denote as $i \sim j$). We define the 1-*neighborhood* of a node i as $\mathcal{N}_i = \{j \in \mathcal{V} : i \sim j\}$. Furthermore, we define an m-dimensional feature vector $\mathbf{X}_i$ for every node $i \in \mathcal{V}$ and arrange all feature vectors into a $v \times m$ matrix $\mathbf{X} = (\mathbf{X}_{ik})$ with $i = 1, \ldots, v$ and $k = 1, \ldots, m$. An N-layer Message-Passing GNN can then be defined via the iterative update rule,

$$\begin{aligned} &\mathbf{X}^n = \sigma(\mathbf{F}_{\theta^n}(\mathbf{X}^{n-1}, \mathcal{G})), \quad \forall n = 1, \ldots, N, \\ &\mathbf{X}^0 = \mathbf{X}, \end{aligned} \tag{6.1}$$

where $\mathbf{F}_{\theta^n}$ is a *learnable* function with parameters θ^n, $\mathbf{X}^n \in \mathbb{R}^{v \times m}$ are the m-dimensional hidden node features, and σ is an element-wise nonlinear activation function. In particular, we consider local (1-neighborhood) coupling of the form $(\mathbf{F}(\mathbf{X}, \mathcal{G}))_i = \mathbf{F}(\mathbf{X}_i, \mathbf{X}_{j \in \mathcal{N}_i})$ operating on the multiset of 1-neighbors of each node, such as Graph Convolutional Networks (GCNs) [Kipf and Welling, 2017] and Graph Attention Networks (GATs) [Velickovic et al., 2018].

Despite the widespread success of GNNs and a plethora of different architectures, several fundamental problems still impede their efficiency on realistic learning tasks. These include bottlenecks on graphs [Alon and Yahav, 2021], oversquashing [Topping et al., 2021], and oversmoothing [Nt and Maehara, 2019, Oono and Suzuki, 2020] phenomena.

Oversmoothing. The phenomenon of oversmoothing refers to the observation that all node features in a deep (multi-layer) GNN tend to converge to the same constant node vector for increasing number of layers. More formally, oversmoothing can be defined as,

Definition 6.0.1 (Oversmoothing)**.** *Let $\mathcal{G}$ be an undirected, connected graph and $\mathbf{X}^n \in \mathbb{R}^{v\times m}$ denote the n-th layer hidden features of an N-layer GNN defined on $\mathcal{G}$. Moreover, we call $\mu : \mathbb{R}^{v\times m} \longrightarrow \mathbb{R}_{\geq 0}$ a* ***node-similarity measure*** *if it satisfies the following axioms:*

1. $\exists \mathbf{c} \in \mathbb{R}^m$ *with* $\mathbf{X}_i = \mathbf{c}$ *for all nodes* $i \in \mathcal{V} \Leftrightarrow \mu(\mathbf{X}) = 0$, *for* $\mathbf{X} \in \mathbb{R}^{v\times m}$
2. $\mu(\mathbf{X} + \mathbf{Y}) \leq \mu(\mathbf{X}) + \mu(\mathbf{Y})$, *for all* $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{v\times m}$

We then define **oversmoothing with respect to** μ *as the layer-wise exponential convergence of the node-similarity measure μ to zero, i.e.,*

3. $\mu(\mathbf{X}^n) \leq C_1 e^{-C_2 n}$, *for* $n = 0, \ldots, N$ *with some constants* $C_1, C_2 > 0$.

Based on definition 6.0.1, oversmoothing can be measured through the Dirichlet energy on graphs,

$$\mathcal{D}(\mathbf{X}^n) = \frac{1}{v} \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{N}_i} \|\mathbf{X}_i^n - \mathbf{X}_j^n\|^2. \tag{6.2}$$

Oversmoothing is then defined with respect to the Dirichlet energy as the exponential convergence to zero of $\mathcal{D}$ for increasing number of GNN layers, i.e.,

$$\mathcal{D}(\mathbf{X}^n) \leq C_1 e^{C_2 n}, \quad \forall n = 1, \ldots, N, \tag{6.3}$$

with some constants $C_1, C_2 > 0$. The main challenge in designing methods that mitigate oversmoothing is to maintain the expressive power of the GNN model, besides preserving the diversity of node features. Thereby, current approaches addressing the oversmoothing issue can be classified into three main strategies: (i) using regularization and normalization Zhou et al. [2021], Zhao and Akoglu [2019], Rong et al. [2020], (ii) leveraging residual connections Chen et al. [2020b], Xu et al. [2018b], Liu et al. [2020], and (iii) changing the dynamics of the message-passing propagation Di Giovanni et al. [2022], Bodnar et al. [2022].

Oversquashing and bottlenecks. The phenomenon of oversquashing Alon and Yahav [2021], Giovanni et al. [2023] relates to the node-wise encoding of (layer-wise) exponentially increasing number of node features through the message-passing framework. This is of particular harm to the GNN in case of bottlenecks in the underlying graph, in which case information cannot be bypassed but rather has to flow through "over-squashed" nodes. In Giovanni et al. [2023], the authors propose to define oversquashing through the second-order mixing of nodes in an underlying graph dataset, connecting oversquashing with the expressive power of GNNs. Finally, to mitigate oversquashing, the majority of recent approaches are based on re-wiring the underlying computational graph [Topping et al., 2021, Deac et al., 2022, Klicpera et al., 2018].

In this part, we focus on constructing novel message-passing frameworks inspired by physical systems (i.e., that of graph-coupled oscillators and multi-rate systems) that are provably able to mitigate oversmoothing and thus allow for building deep GNNs. Moreover, we provide extensive empirical evidence that the proposed frameworks obtain state-of-the-art results on real-world small- /medium- /and large-scale graph dataset with different levels of homophily.

Chapter 7

Graph-Coupled Oscillator Network

Graph-coupled oscillators are often encountered in mechanical, electronic, and biological systems, and have been studied extensively Strogatz [2001], with a prominent example being functional circuits in the brain such as cortical columns Stiefel and Ermentrout [2016]. In these circuits, each neuron oscillates with periodic firing and spiking of the action potential. The network of neurons is coupled in the form of a graph, with neurons representing nodes and edges corresponding to synapses linking neurons.

Motivated by this, we propose a novel physically-inspired approach to learning on graphs. Our framework, termed **GraphCON** (Graph-Coupled Oscillator Network) builds upon suitable time-discretizations of a specific class of ordinary differential equations (ODEs) that model the dynamics of a network of nonlinear controlled and damped oscillators, which are coupled via the adjacency structure of the underlying graph.

Main Contributions. In the subsequent sections, we will demonstrate the following features of GraphCON:

- GraphCON is flexible enough to accommodate any standard GNN layer (such as GAT or GCN) as its coupling function. As timesteps of our discretized ODE can be interpreted as layers of a deep neural network Chen et al. [2018], Haber and Ruthotto [2018], Chamberlain et al. [2021b], one can view GraphCON as a wrapper around any underlying basic GNN layer allowing to build deep GNNs. Moreover, we will show that standard GNNs can be recovered as steady states of the underlying class of ODEs, whereas GraphCON utilizes their dynamic behavior to sample a richer set of states, which leads to better expressive power.
- We mathematically formulate the frequently encountered oversmoothing problem for GNNs Nt and Maehara [2019], Oono and Suzuki [2020] in terms of the stability of zero-Dirichlet energy steady states of the underlying equations. By a careful analysis of the dynamics of the proposed ODEs, we demonstrate that any zero-Dirichlet energy steady states are not (exponentially) stable. Consequently, we show that the oversmoothing problem for GraphCON is mitigated by construction.
- We rigorously prove that GraphCON mitigates the so-called exploding and vanishing gradients problem for the resulting GNN. Hence, GraphCON can greatly improve the trainability of deep multi-layer GNNs.
- We provide an extensive empirical evaluation of GraphCON on a wide variety of graph learning tasks such as transductive and inductive node classification and graph regression and classification, demonstrating that GraphCON achieves competitive performance.

7.1 The proposed graph-learning framework

We recall the $v \times m$-dimensional feature matrix $\mathbf{X}$ defined on the undirected graph $\mathcal{G}$. Based on this and central to our framework, we define a *graph dynamical system* represented by the following nonlinear system of ODEs:

$$\mathbf{X}'' = \sigma(\mathbf{F}_\theta(\mathbf{X}, t, \mathcal{G})) - \gamma\mathbf{X} - \alpha\mathbf{X}'. \tag{7.1}$$

Here, $\mathbf{X}(t)$ denotes the time-dependent $v \times m$-matrix of node features, σ is the activation function, $\mathbf{F}_\theta$ is a general learnable (possibly time-dependent) 1-neighborhood coupling function of the form

$$(\mathbf{F}_\theta(\mathbf{X}, t, \mathcal{G}))_i = \mathbf{F}_\theta\left(\mathbf{X}_i(t), \mathbf{X}_j(t), t\right) \quad \forall i \sim j, \tag{7.2}$$

parametrized with a set of learnable parameters θ.

By introducing the auxiliary *velocity* variable $\mathbf{Y}(t) = \mathbf{X}'(t) \in \mathbb{R}^{v\times m}$, we can rewrite the second-order ODEs (7.1) as a first-order system:

$$\begin{aligned} \mathbf{Y}' &= \sigma(\mathbf{F}_\theta(\mathbf{X}, t, \mathcal{G})) - \gamma\mathbf{X} - \alpha\mathbf{Y}, \\ \mathbf{X}' &= \mathbf{Y}. \end{aligned} \tag{7.3}$$

The key idea of our framework is, given the input node features $\mathbf{X}(0)$ as an initial condition, to use the solution $\mathbf{X}(T)$ at some time T as the output (more generally, one can also apply (linear) transformations (embeddings) to $\mathbf{X}(0)$ and $\mathbf{X}(T)$). As will be shown in the following section, the space of solutions of our system is a rich class of functions that can solve many learning tasks on a graph.

The system (7.3) must be solved by an iterative numerical solver using a suitable time-discretization. It is highly desirable for a time-discretization to preserve the structure of the underlying ODEs (7.3) Hairer et al. [1987]. In this chapter, we use the following IMEX (implicit-explicit) time-stepping scheme, which extends the *symplectic Euler method* Hairer et al. [1987] to systems with an additional damping term,

$$\begin{aligned} \mathbf{Y}^n &= \mathbf{Y}^{n-1} + \Delta t[\sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, t^{n-1}, \mathcal{G})) - \gamma\mathbf{X}^{n-1} - \alpha\mathbf{Y}^{n-1}], \\ \mathbf{X}^n &= \mathbf{X}^{n-1} + \Delta t\mathbf{Y}^n, \end{aligned} \tag{7.4}$$

for $n = 1, \ldots, N$, where $\Delta t > 0$ is a fixed time-step and $\mathbf{Y}^n, \mathbf{X}^n$ denote the hidden node features at time $t^n = n\Delta t$. The iterative scheme (7.4) can be interpreted as an N-layer graph neural network (with potential additional linear input and readout layers, omitted here for simplicity), which we refer to as **GraphCON** (see section 7.2 for the motivation of this nomenclature). The coupling function $\mathbf{F}_\theta$ plays the role of a message-passing mechanism (Gilmer et al. [2017], also referred to, in various contexts, as 'diffusion' or 'neighborhood aggregation') in traditional GNNs.

Choice of the coupling function $\mathbf{F}_\theta$. Our framework allows for any learnable 1-neighborhood coupling to be used as $\mathbf{F}_\theta$, including instances of message-passing mechanisms commonly used in the Graph ML literature such as GraphSAGE [Hamilton et al., 2017], Graph Attention Velickovic et al. [2018], Graph Convolution Defferrard et al. [2016], Kipf and Welling [2017], SplineCNN [Fey et al., 2018], or MoNet [Monti et al., 2017]). In this chapter, we focus on two particularly popular choices:

Attentional message-passing of Velickovic et al. [2018]:

$$\mathbf{F}_\theta(\mathbf{X}^n, t^n, \mathcal{G}) = \mathbf{A}^n(\mathbf{X}^n)\mathbf{X}^n\mathbf{W}^n,$$

with learnable weight matrices $\mathbf{W}^n \in \mathbb{R}^{m\times m}$ and attention matrices $\mathbf{A}^n \in \mathbb{R}^{n\times n}$ following the adjacency structure of the graph $\mathcal{G}$, i.e., $(\mathbf{A}^n(\mathbf{X}^n))_{ij} = 0$ if $j \notin \mathcal{N}_i$ and

$$(\mathbf{A}^n(\mathbf{X}^n))_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}^n\mathbf{X}_i^n || \mathbf{W}^n\mathbf{X}_j^n]))}{\sum\limits_{k\in\mathcal{N}_i} \exp(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}^n\mathbf{X}_i^n || \mathbf{W}^n\mathbf{X}_k^n]))},$$

otherwise (here $\mathbf{X}_i^n$ denotes the i-th row of $\mathbf{X}^n$ and $\mathbf{a} \in \mathbb{R}^{2m}$). We refer to (7.4) based on this attentional 1-neighborhood coupling as **GraphCON-GAT**.

Graph convolution operator of Kipf and Welling [2017]:

$$\mathbf{F}_\theta(\mathbf{X}^n, t^n, \mathcal{G}) = \hat{\mathbf{D}}^{-\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^n \mathbf{W}^n, \tag{7.5}$$

with $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denoting the adjacency matrix of $\mathcal{G}$ with inserted self-loops, diagonal degree matrix $\mathbf{D} = \mathrm{diag}(\sum_{l=1}^n \hat{\mathbf{A}}_{kl})$, and $\mathbf{W}_i^n \in \mathbb{R}^{m \times m}$ being learnable weight matrices. We refer to (7.4) based on this convolutional 1-neighborhood coupling as **GraphCON-GCN**.

Steady States of GraphCON and relation to GNNs. It is straightforward to see that the steady states $\mathbf{X}^*, \mathbf{Y}^*$ of the GraphCON dynamical system (7.4) with an *autonomous* coupling function $\mathbf{F}_\theta = \mathbf{F}_\theta(\mathbf{X}, \mathcal{G})$ (as in GraphCON-GAT or GraphCON-GCN) are given by $\mathbf{Y}^* \equiv \mathbf{0}$ and

$$\mathbf{X}^* = \frac{1}{\gamma} \sigma(\mathbf{F}_\theta(\mathbf{X}^*, \mathcal{G})). \tag{7.6}$$

Using a simple fixed point iteration to find the steady states (7.6) yields a multi-layer GNN of the form;

$$\mathbf{X}^n = \frac{1}{\gamma} \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})), \quad \text{for } n = 1, 2, \ldots, N. \tag{7.7}$$

We observe that (up to a rescaling by the factor $1/\gamma$) equation (7.7) corresponds to the update formula for any standard N-layer message-passing GNN Gilmer et al. [2017], including such popular variants as GAT Velickovic et al. [2018] or GCN Kipf and Welling [2017].

Thus, this interpretation of GraphCON (7.4) clearly brings out its relationship with standard GNNs. Unlike in standard multi-layer GNNs of the generic form (7.7) that can be thought of as steady states of the underlying ODEs (7.3), GraphCON evolves the underlying node features *dynamically in time*. Interpreting the multiple GNN layers as iterations at times $t^n = n\Delta t$ in (7.4), we observe that the node features in GraphCON follow the trajectories of the corresponding dynamical system and can explore a richer sampling of the underlying latent feature space, leading to possibly greater expressive power than standard GNNs (7.7), which might remain in the vicinity of steady states.

Moreover, this interpretation also reveals that, in principle, any GNN of the form (7.7) can be used within the GraphCON framework, offering a very flexible and broad class of architectures. Hence, one can think of GraphCON as an *additional wrapper* on top of any basic GNN layer allowing for a principled and stable design of deep multi-layered GNNs. In the following Section 7.2, we show that such an approach has several key advantages over standard GNNs.

Related Work. Differential equations have historically played a role in designing and interpreting various algorithms in machine learning, including nonlinear dimensionality reduction methods Belkin and Niyogi [2003], Coifman and Lafon [2006] and ranking Page et al. [1999], Chakrabarti [2007] (all of which are related to closed-form solutions of diffusion PDEs). In the context of Deep Learning, differential equations have been used to derive various types of neural networks including Neural ODEs and their variants, that have been used to design and interpret residual Chen et al. [2018] and convolutional Haber and Ruthotto [2018] neural networks. These approaches have recently gained traction in Graph ML, e.g. with ODE-based models for learning on graphs Avelar et al. [2019], Poli et al. [2019], Zhuang et al. [2020], Xhonneux et al. [2020].

Chamberlain et al. [2021b] used parabolic diffusion-type PDEs to design GNNs using graph gradient and divergence operators as the spatial differential operator, a transformer type-attention as a learnable

diffusivity function ('1-neighborhood coupling' in our terminology), and a variety of time stepping schemes to discretize the temporal dimension in this framework. Chamberlain et al. [2021a] applied a non-euclidean diffusion equation ('Beltrami flow') to a joint positional-feature space, yielding a scheme with adaptive spatial derivatives ('graph rewiring'), and Topping et al. [2021] studied a discrete geometric PDE similar to Ricci flow to improve information propagation in GNNs. We can see the contrast between the diffusion-based methods of Chamberlain et al. [2021b,a] and GraphCON in the simple case of identity activation $\sigma(x) = x$. Then, under the further assumption that the second-order time derivative $\mathbf{X}''$ is removed from (7.1) and $\alpha = \gamma = 1$, we recover the graph diffusion-PDEs of Chamberlain et al. [2021b]. Hence, the presence of the temporal second-order derivative distinguishes this approach from diffusion-based PDEs.

Eliasof et al. [2021] proposed a GNN framework arising from a mixture of parabolic (diffusion) and hyperbolic (wave) PDEs on graphs with convolutional coupling operators, which describe dissipative wave propagation. We point out that a particular instance of their model (damped wave equation, also called as the *Telegrapher's equation*) can be obtained as a special case of our model (7.1) with the identity activation function. This is not surprising as the zero grid-size limit of oscillators on a regular grid yields a wave equation. However, given that we use a nonlinear activation function and the specific placement of the activation layer in (7.3), a local PDE interpretation of the general form of our underlying ODEs (7.1) does not appear to be feasible.

Finally, we proposed the explicit use of networks of coupled, controlled oscillators to design machine learning models in context of recurrent neural networks (RNNs) in Chapter 2 and Chapter 3.

7.2 Rigorous analysis of GraphCON

7.2.1 GraphCON dynamics

To gain some insight into the functioning of GraphCON (7.4), we start by setting the hyperparameter $\gamma = 1$ and assuming that the 1-neighborhood coupling $\mathbf{F}_\theta$ is given by either the GAT or GCN type coupling functions. In this case, the underlying ODEs (7.3) takes the following node-wise form,

$$\begin{aligned} \mathbf{X}_i' &= \mathbf{Y}_i, \\ \mathbf{Y}_i' &= \sigma\left(\sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}\mathbf{X}_j\right) - \mathbf{X}_i - \alpha\mathbf{Y}_i, \end{aligned} \tag{7.8}$$

for all nodes $i \in \mathcal{V}$, with $\mathbf{A}_{ij} = \mathbf{A}\left(\mathbf{X}_i(t), \mathbf{X}_j(t)\right) \in \mathbb{R}$ stemming from the attention or convolution operators. Furthermore, the matrices are *right stochastic* i.e., the entries satisfy,

$$\begin{aligned} &0 \leq \mathbf{A}_{ij} \leq 1, \quad \forall j \in \mathcal{N}_i, \quad \forall i \in \mathcal{V}, \\ &\sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij} = 1, \quad \forall i \in \mathcal{V}. \end{aligned} \tag{7.9}$$

Uncoupled case. The simplest case of (7.8), corresponds to setting $\sigma \equiv 0$ and $\alpha = 0$. In this case, all nodes are *uncoupled* from each other and the solutions of the resulting ODEs are of the form,

$$\mathbf{X}_i(t) = \mathbf{X}_i(0)\cos(t) + \mathbf{Y}_i(0)\sin(t). \tag{7.10}$$

Thus, the dynamics of the ODEs (7.3) in this special case correspond to a *system of uncoupled oscillators*, with each node oscillating at unit frequency.

Coupled linear case. Next, we introduce coupling between the nodes that are adjacent on the underlying graph $\mathcal{G}$ and assume identity activation function $\sigma(x) = x$. In this case, (7.8) is a *coupled linear system* and an exact closed form solution, such as (7.10) may not be possible. However, we can describe the dynamics of (7.8) in the form of the following proposition,

Proposition 7.2.1. *Let the node features* $\mathbf{X}, \mathbf{Y}$ *evolve according to the ODEs* (7.8) *with activation function* $\sigma = \mathrm{id}$ *and time-independent matrix* $\mathbf{A}$ *(e.g.* $\mathbf{A}_{ij} = \mathbf{A}(\mathbf{X}_i(0), \mathbf{X}_j(0))$ *using the initial features). Further assume that* $\mathbf{A}$ *is* symmetric *and* $\alpha = 0$*. Then*

$$\begin{aligned}&\sum_{i\in\mathcal{V}} \|\mathbf{Y}_i(t)\|^2 + \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}\|\mathbf{X}_i(t) - \mathbf{X}_j(t)\|^2 \\ &= \sum_{i\in\mathcal{V}} \|\mathbf{Y}_i(0)\|^2 + \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}\|\mathbf{X}_i(0) - \mathbf{X}_j(0)\|^2,\end{aligned} \tag{7.11}$$

holds for all $t > 0$.

Proof. We multiply $\mathbf{Y}_i^\top$ to the second equation of (7.8) and obtain,

$$\mathbf{Y}_i^\top \frac{d\mathbf{Y}_i}{dt} = \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}\mathbf{Y}_i^\top (\mathbf{X}_j - \mathbf{X}_i), \quad \left(\text{as } \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij} = 1\right)$$

Summing over $i \in \mathcal{V}$ and using the symmetry condition $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ in the above expression yields,

$$\begin{aligned}\frac{d}{dt}\sum_{i\in\mathcal{V}} \frac{\|\mathbf{Y}_i\|^2}{2} &= -\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij} (\mathbf{Y}_j - \mathbf{Y}_i)^\top (\mathbf{X}_j - \mathbf{X}_i), \\ &= -\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij} \left(\frac{d(\mathbf{X}_j - \mathbf{X}_i)}{dt}\right)^\top (\mathbf{X}_j - \mathbf{X}_i) \\ \Rightarrow \frac{1}{2}\frac{d}{dt} &\left(\sum_{i\in\mathcal{V}} \|\mathbf{Y}_i\|^2 + \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}\|\mathbf{X}_j - \mathbf{X}_i\|^2\right) = 0.\end{aligned}$$

Integrating the last line in the above expression over time $[0, t]$ yields the desired identity (7.11) □

Thus, in this case, we have shown that the dynamics of the underlying ODEs (7.8) preserves the *energy*,

$$\mathscr{E}(t) := \sum_{i\in\mathcal{V}} \|\mathbf{Y}_i(t)\|^2 + \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}\|\mathbf{X}_i(t) - \mathbf{X}_j(t)\|^2, \tag{7.12}$$

and the trajectories of (7.8) are constrained to lie on a manifold of the node feature space, defined by the level sets of the energy. In particular, energy (7.12) is not produced or destroyed but simply redistributed among the nodes of the underlying graph $\mathcal{G}$. Thus, the dynamics of (7.3) in this setting amounts to the motion of a *linear system of coupled oscillators*.

General nonlinear case. In the general case, we have (i) a nonlinear activation function σ; (ii) time-dependent nonlinear coefficients $\mathbf{A}_{ij} = \mathbf{A}(\mathbf{X}_i(t), \mathbf{X}_j(t))$; and (iii) possible unsymmetrical entries $\mathbf{A}_{ij} \neq \mathbf{A}_{ji}$. All these factors destroy the energy conservation property (7.11) and can possibly lead to unbounded growth of the energy. Hence, we need to add some damping to the system. To this end, the damping term in (7.8) is activated by setting $\alpha > 0$. Moreover, $\gamma \neq 1$ corresponds to controlling

frequencies of the nodes. Thus, the overall dynamics of the underlying ODEs (7.3) amounts to the motion of a nonlinear system of *coupled, controlled and damped oscillators* with the coupling structure being that of the underlying graph. This explains our choice of the name, *Graph-Coupled Oscillatory Neural Network* or 'GraphCON' for short.

We illustrate the dynamics of GraphCON in Fig. 7.1, where the model is applied to the graph of a molecule from the ZINC database Irwin et al. [2012], with features $\mathbf{X}$ denoting the position of the nodes and they are propagated in time through the action of GraphCON (7.4). The oscillatory behavior of the node features, as well as their dependence on the adjacency structure of the underlying graph can be clearly observed in this figure.

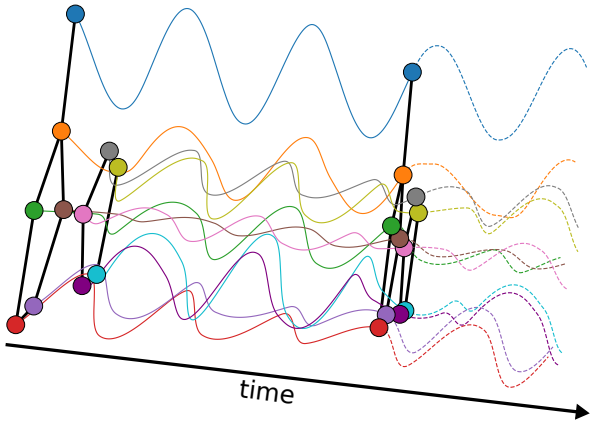

Figure 7.1: Illustration of GraphCON dynamics on a ZINC molecular graph. The initial positions of GraphCON ($\mathbf{X}_0$ in (7.4)) are represented by the 2-dimensional positions of the nodes, while the initial velocities ($\mathbf{Y}_0$ in (7.4)) are set to the initial positions. The positions are propagated forward in time ('layers') using GraphCON-GCN with random weights. The molecular graph is plotted at initial time $t = 0$ as well as at $t = 20$.

7.2.2 On the oversmoothing issue

We will show that GraphCON allows to mitigate the oversmoothing problem in the sense of definition 6.0.1, measured by the Dirichlet energy (6.2). To see this, we focus on the underlying ODEs (7.3). It is trivial to extend the Dirichlet energy-based definition of oversmoothing from the discrete case to the continuous one,

Definition 7.2.2 (Continuous-time oversmoothing)**.** *Let $\mathcal{G}$ be an undirected, connected graph and $\mathbf{X}(t) \in \mathbb{R}^{v\times m}$ denote the normalized hidden node features of a continuous-time GNN (7.1) at time $t \geq 0$ defined on $\mathcal{G}$. Moreover, μ is a node-similarity measure as of Definition 6.0.1. We then define oversmoothing with respect to μ as the exponential convergence in time of the node-similarity measure μ to zero, i.e.,*

$$\mu(\mathbf{X}(t)) \leq C_1 e^{-C_2 t}, \text{ for } t \geq 0, \tag{7.13}$$

with some constants $C_1, C_2 > 0$.

Note that also here we instantiate the node-similarity measure with the Dirichlet energy (6.2).

We have the following simple proposition that characterizes the oversmoothing problem for the underlying ODEs in the standard terminology of dynamical systems Wiggins [2003],

Proposition 7.2.3. *The oversmoothing problem occurs for the ODEs* (7.3) *if and only if the hidden states* $(\mathbf{X}^*, \mathbf{Y}^*) = (\mathbf{c}, \mathbf{0})$ *are* exponentially stable steady states (fixed points) *of the ODE* (7.3)*, for some* $\mathbf{c} \in \mathbb{R}^m$ *and* $\mathbf{0}$ *being the* m*-dimensional vector with zeroes for all its entries.*

Proof. By the definition of the Dirichlet energy (6.2), (7.13) implies that,

$$\lim_{t\to\infty} \mathbf{X}_i(t) \equiv \mathbf{c}, \quad \forall i \in \mathcal{V}, \tag{7.14}$$

for some $\mathbf{c} \in \mathbb{R}^m$. In other words, all the hidden node features converge to the same feature vector $\mathbf{c}$ as time increases. Moreover, by (7.13), this convergence is exponentially fast.

Plugging in (7.14) in to the first equation of the ODE (7.3), we obtain that,

$$\lim_{t\to\infty} \mathbf{Y}_i(t) \equiv \mathbf{0}, \quad \forall i \in \mathcal{G}, \tag{7.15}$$

with $\mathbf{0}$ being the m vector with zeroes for all its entries. Thus, oversmoothing in the sense of definition 7.2.2, amounts to $(\mathbf{c}, \mathbf{0})$ being an exponentially stable fixed point (steady state) for the dynamics of (7.8)

On the other hand, if $(\mathbf{c}, \mathbf{0})$ is an exponentially stable steady state of (7.8), then the trajectories converge to this state exponentially fast satisfying (7.13). Consequently, by the definition of the Dirichlet energy (6.2), we readily observe that the oversmoothing problem, in the sense of definition 7.2.2, occurs in this case. □

In other words, all the trajectories of the ODE (7.3), that start within the corresponding basin of attraction, have to converge exponentially fast in time (satisfy (7.13)) to the corresponding steady state $(\mathbf{c}, \mathbf{0})$ for the oversmoothing problem to occur for this system. Note that the basins of attraction will be different for different values of $\mathbf{c}$.

Given this characterization, the key questions are a) whether $(\mathbf{c}, \mathbf{0})$ are fixed points for the ODE (7.3), and b) whether these fixed points are exponentially stable. We answer these questions for the ODEs (7.8) in the following

Proposition 7.2.4. *Assume that the activation function* σ *in the ODEs* (7.8) *is ReLU. Then, for any* $\mathbf{c} \in \mathbb{R}^m$ *such that each entry of the vector* $\mathbf{c}_\ell \geq 0$*, for all* $1 \leq \ell \leq m$*, the hidden state* $(\mathbf{c}, \mathbf{0})$ *is a steady state for the ODEs* (7.8)*. However under the additional assumption of* $\alpha \geq \frac{1}{2}$*, this fixed point is* not exponentially stable*.*

The fact that $(\mathbf{c}, \mathbf{0})$ is a steady state of (7.8), for any *positive* $\mathbf{c}$ is straightforward to see from the structure of (7.8) and the definition of the ReLU activation function. We can already observe from the energy identity (7.11) for the simplified symmetric linear system that the energy (7.12) for the small perturbations around the steady state $(\mathbf{c}, \mathbf{0})$ is conserved in time. Hence, these small perturbations do not decay at all, let alone, exponentially fast in time. Thus, these steady states are not exponentially stable.

An extension of this analysis to the nonlinear time-dependent, possibly non-symmetric system (7.8) is more subtle. Therefore, the main aim of the remaining section is to prove proposition 7.2.4 by showing that steady states of (7.8), of the form $(\mathbf{c}, \mathbf{0})$ are not exponentially stable.

To this end, we fix $\mathbf{c}$ and start by considering small perturbations around the fixed point $(\mathbf{c}, \mathbf{0})$. We define,

$$\hat{\mathbf{X}}_i = \mathbf{X}_i - \mathbf{c}, \hat{\mathbf{Y}}_i = \mathbf{Y}_i,$$

and evolve these perturbations by the linearized ODE,

$$\begin{aligned} \hat{\mathbf{X}}_i' &= \hat{\mathbf{Y}}_i, \\ \hat{\mathbf{Y}}_i{}' &= \sigma'(\mathbf{c}) \sum_{j\in\mathcal{N}_i} \hat{\mathbf{A}}_{i,j}\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i - \alpha\hat{\mathbf{Y}}_i, \end{aligned} \tag{7.16}$$

As $\sigma(x) = \max(x, 0)$ and $\mathbf{c} \geq 0$, we have that $\sigma'(\mathbf{c}) = ID$ and linearized system (7.17) reduces to,

$$\begin{aligned} \hat{\mathbf{X}}_i' &= \hat{\mathbf{Y}}_i, \\ \hat{\mathbf{Y}}_i{}' &= \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i - \alpha \hat{\mathbf{Y}}_i, \end{aligned} \tag{7.17}$$

with

$$\begin{aligned} \hat{\mathbf{A}}_{ij} &= \mathbf{A}_{ij}(\mathbf{c}, \mathbf{c}), \quad \forall j \in \mathcal{N}_i, \quad \forall i \in \mathcal{G}, \\ 0 \leq \hat{A}_{ij} &\leq 1, \quad \sum_{j \in \mathcal{N}_i} \hat{A}_{ij} = 1. \end{aligned} \tag{7.18}$$

We have the following proposition on the dynamics of linearized system (7.17) with respect to perturbations of the fixed point $(\mathbf{c}, \mathbf{0})$,

Proposition 7.2.5. *Perturbations $\hat{\mathbf{X}}(t), \hat{\mathbf{Y}}(t)$ of the fixed point $(\mathbf{c}, \mathbf{0})$, which evolve according to (7.17) satisfy the following identity,*

$$\begin{aligned} &\frac{1}{v}\left(\sum_{i \in \mathcal{V}} \|\hat{\mathbf{Y}_i}(t)\|^2 + \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \frac{\hat{\mathbf{A}}_{ij} + \hat{\mathbf{A}}_{ji}}{2} \left(\|\hat{\mathbf{X}}_j(t) - \hat{\mathbf{X}}_i(t)\|^2\right)\right) = T_1(t) + T_2(t) + T_3(t), \\ T_1(t) &= \frac{1}{v} \sum_{i \in \mathcal{V}} \left(\|\hat{\mathbf{Y}_i}(0)\|^2\right) e^{-2\alpha t} + \frac{1}{v} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \frac{\hat{\mathbf{A}}_{ij} + \hat{\mathbf{A}}_{ji}}{2} \left(\|\hat{\mathbf{X}}_j(0) - \hat{\mathbf{X}}_i(0)\|^2\right) e^{-2\alpha t} \\ T_2(t) &= \frac{\alpha}{v} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \left(\hat{\mathbf{A}}_{ij} + \hat{\mathbf{A}}_{ji}\right) \int_0^t \|\hat{\mathbf{X}}_j(s) - \hat{\mathbf{X}}_i(s)\|^2 e^{2\alpha(s-t)} ds \\ T_3(t) &= \frac{1}{v} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \left(\hat{\mathbf{A}}_{ij} - \hat{\mathbf{A}}_{ji}\right) \int_0^t \left(\hat{\mathbf{Y}}_i(s) + \hat{\mathbf{Y}}_j(s)\right)^\top \left(\hat{\mathbf{X}}_j(s) - \hat{\mathbf{X}}_i(s)\right) e^{2\alpha(s-t)} ds \end{aligned} \tag{7.19}$$

Proof. Multiplying the second equation in (7.17) with $\hat{\mathbf{Y}}_i^\top$ and using the fact that $\sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} = 1$, we obtain,

$$\begin{aligned} &\frac{d}{dt} \frac{\|\hat{\mathbf{Y}}_i\|^2}{2} + \alpha \|\hat{\mathbf{Y}}_i\|^2 = \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \hat{\mathbf{Y}}_i^\top \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right), \\ &= \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \frac{\left(\hat{\mathbf{Y}}_i + \hat{\mathbf{Y}}_j\right)^\top}{2} \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right) - \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \frac{\left(\hat{\mathbf{Y}}_j - \hat{\mathbf{Y}}_i\right)^\top}{2} \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right), \\ &= \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \frac{\left(\hat{\mathbf{Y}}_i + \hat{\mathbf{Y}}_j\right)^\top}{2} \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right) - \sum_{j \in \mathcal{N}_i} \frac{\hat{\mathbf{A}}_{ij}}{2} \frac{d}{dt} \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^\top \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right), \end{aligned} \tag{7.20}$$

where we have used the first equation of (7.17) in the last line of (7.20). Consequently, we have for all $i \in \mathcal{V}$,

$$\begin{aligned} &\frac{d}{dt} \frac{\|\hat{\mathbf{Y}}_i\|^2}{2} + \alpha \|\hat{\mathbf{Y}}_i\|^2 + \frac{d}{dt} \sum_{j \in \mathcal{N}_i} \frac{\hat{\mathbf{A}}_{ij}}{2} \frac{\|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\|^2}{2} \\ &\qquad = \sum_{j \in \mathcal{N}_i} \hat{\mathbf{A}}_{ij} \frac{\left(\hat{\mathbf{Y}}_i + \hat{\mathbf{Y}}_j\right)^\top}{2} \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right) \end{aligned} \tag{7.21}$$

Summing (7.21) over all nodes $i \in \mathcal{V}$ yields,

$$\begin{aligned}\frac{d}{dt}\sum_{i\in\mathcal{V}}\frac{\|\hat{\mathbf{Y}}_i\|^2}{2} + \alpha\sum_{i\in\mathcal{V}}\|\hat{\mathbf{Y}}_i\|^2 + \frac{d}{dt}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\frac{\|\hat{\mathbf{X}}_j-\hat{\mathbf{X}}_i\|^2}{2} \\ = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}-\hat{\mathbf{A}}_{ji}}{2}\left(\hat{\mathbf{Y}}_i+\hat{\mathbf{Y}}_j\right)^\top\left(\hat{\mathbf{X}}_j-\hat{\mathbf{X}}_i\right)\end{aligned} \tag{7.22}$$

Multiplying $e^{2\alpha t}$ to both sides of (7.22) and using the chain rule, we readily obtain,

$$\begin{aligned}&\frac{d}{dt}\sum_{i\in\mathcal{V}}e^{2\alpha t}\left(\frac{\|\hat{\mathbf{Y}}_i\|^2}{2}+\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\frac{\|\hat{\mathbf{X}}_j-\hat{\mathbf{X}}_i\|^2}{2}\right) \\ &= \alpha e^{2\alpha t}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\|\hat{\mathbf{X}}_j-\hat{\mathbf{X}}_i\|^2 \\ &+ e^{2\alpha t}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}-\hat{\mathbf{A}}_{ji}}{2}\left(\hat{\mathbf{Y}}_i+\hat{\mathbf{Y}}_j\right)^\top\left(\hat{\mathbf{X}}_j-\hat{\mathbf{X}}_i\right)\end{aligned} \tag{7.23}$$

Integrating (7.23) over the time interval $[0,t]$ yields,

$$\begin{aligned}&\sum_{i\in\mathcal{V}}\left(\frac{\|\hat{\mathbf{Y}_i}(t)\|^2}{2}\right)e^{2\alpha t}+\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\left(\frac{\|\hat{\mathbf{X}}_j(t)-\hat{\mathbf{X}}_i(t)\|^2}{2}\right)e^{2\alpha t} \\ &= \sum_{i\in\mathcal{V}}\left(\frac{\|\hat{\mathbf{Y}_i}(0)\|^2}{2}\right)+\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\left(\frac{\|\hat{\mathbf{X}}_j(0)-\hat{\mathbf{X}}_i(0)\|^2}{2}\right) \\ &+ \alpha\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}+\hat{\mathbf{A}}_{ji}}{2}\int_0^t\|\hat{\mathbf{X}}_j(s)-\hat{\mathbf{X}}_i(s)\|^2e^{2\alpha s}ds \\ &+ \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{\hat{\mathbf{A}}_{ij}-\hat{\mathbf{A}}_{ji}}{2}\int_0^t\left(\hat{\mathbf{Y}}_i(s)+\hat{\mathbf{Y}}_j(s)\right)^\top\left(\hat{\mathbf{X}}_j(s)-\hat{\mathbf{X}}_i(s)\right)e^{2\alpha s}ds\end{aligned} \tag{7.24}$$

We readily obtain the desired identity (7.19) from (7.24). □

Next, we observe that the right-hand side of the nonlinear ODEs (7.8) is *globally Lipschitz*. Therefore, solutions exist for all time $t > 0$, are unique and depend continuously on the data.

We assume that the initial perturbations around the steady state $(\mathbf{c}, \mathbf{0})$ are small i.e., they satisfy

$$\begin{aligned}\|\hat{\mathbf{X}}_i(0)-\hat{\mathbf{X}}_j(0)\| \le \epsilon, \quad \forall j \in \mathcal{N}_i, \quad \forall i \in \mathcal{V}, \\ \|\hat{\mathbf{Y}}_i(0)\| \le \epsilon, \quad \forall i \in \mathcal{V},\end{aligned}$$

for some $0 < \epsilon << 1$.

Hence, there exists a small time $\tau > 0$ such that the time-evolution of these perturbations can be approximated to arbitrary accuracy by solutions of the linearized system (7.17).

Next, we see from the identity (7.19) that the evolution of the perturbations $\hat{\mathbf{X}}, \hat{\mathbf{Y}}$ from the fixed point $(\mathbf{c}, \mathbf{0})$ for the linearized system (7.17) is balanced by three terms $T_{1,2,3}$. The term T_1 is clearly a *dissipative* term and says that the initial perturbations are damped exponentially fast in time.

On the other hand, the term T_2, which has a positive sign, is a *production* term and says that the initial perturbations will *grow* with time t. Given the continuous dependence of the dynamics evolved by the ODE (7.17), there exists a time, still called τ by choosing it even smaller than the τ encountered before, such that

$$\begin{aligned} \|\hat{\mathbf{X}}_i(t) - \hat{\mathbf{X}}_j(t)\| \sim \mathcal{O}(\epsilon), \quad \forall j \in \mathcal{N}_i, \quad \forall i \in \mathcal{V}, \quad \forall t \in [0, \tau], \\ \|\hat{\mathbf{Y}}_i(t)\| \sim \mathcal{O}(\epsilon), \quad \forall i \in \mathcal{V}, \forall t \in [0, \tau]. \end{aligned} \tag{7.25}$$

Plugging the above expression into the term T_2 in (7.19) and using the right-stochasticity of the matrix $\hat{\mathbf{A}}$, we obtain that,

$$T_2(t) \sim \mathcal{O}(\epsilon^2) \left(1 - e^{-2\alpha t}\right), \quad \forall t \leq \tau \tag{7.26}$$

Thus, the leading term in T_2 *grows* algebraically with respect to the initial perturbations.

Next we turn our attention to the term T_3 in (7.19). This term is proportional to the *asymmetry* in the graph-coupling matrix $\hat{\mathbf{A}} = \mathbf{A}(\mathbf{c}, \mathbf{c})$. If this matrix were symmetric, then T_3 vanishes. On the other hand, for many 1-neighborhood couplings considered in this chapter, the matrix $\hat{\mathbf{A}}$ is not symmetric. In fact, one can explicitly compute that for the GAT and Transformers attention and GCN-couplings, we have,

$$\hat{\mathbf{A}}_{ij} = \frac{1}{\deg(i)}, \quad \forall j \in \mathcal{N}_i, \quad \forall i \in \mathcal{V}. \tag{7.27}$$

Here, deg refers to the degree of the node, with possibly inserted self-loops.

As the ordering of nodes of the graph $\mathcal{G}$ is arbitrary, we can order them in such a manner that $\hat{\mathbf{A}}_{ij} > \hat{\mathbf{A}}_{ji}$. Even with this ordering, as long as the matrix $\hat{\mathbf{A}}$ is not symmetric, the term T_3 is of *indefinite sign*. If it is positive, then we have additional growth with respect to time in (7.19). On the other hand, if T_3 is negative, it will have a *dissipative effect*. The rate of this dissipation can be readily calculated for a short time $t \leq \tau$ under the assumption (7.25) to be,

$$|T_3(t)| \sim \frac{\overline{D} - \underline{D}}{\overline{D}\underline{D}} \left(\frac{1 - e^{-2\alpha t}}{2\alpha}\right) \mathcal{O}(\epsilon^2). \tag{7.28}$$

Here, we define,

$$\overline{D} = \max_{i \in \mathcal{V}} \deg(i), \quad \underline{D} = \min_{i \in \mathcal{V}} \deg(i) \tag{7.29}$$

Thus by combining (7.26) with (7.28), we obtain,

$$T_2 + T_3 \sim \left(1 - \frac{\overline{D} - \underline{D}}{2\alpha \overline{D}\underline{D}}\right) \left(1 - e^{-2\alpha t}\right) \mathcal{O}(\epsilon^2) \tag{7.30}$$

In particular for $\alpha \geq 1/2$, we see from (7.30), that the overall balance (7.19) leads to an algebraic growth, rather than exponential decay, of the initial perturbations of the fixed point $(\mathbf{c}, \mathbf{0})$. Thus, we have shown that this steady state is not exponentially stable and small perturbations will take the trajectories of the ODE (7.17) away from this fixed point, completing the proof of Proposition 7.2.4.

Remark 7.2.6. *We see from the above proof, the condition $\alpha \geq \frac{1}{2}$ is only a sufficient condition for the proof of Proposition 7.2.4, we can readily replace it by,*

$$\alpha \geq \frac{\overline{D} - \underline{D}}{2\overline{D}\underline{D}}$$

To summarize the careful analysis, small perturbations can grow polynomially in time (at least for short time periods) and do not decay exponentially. Consequently, the fixed point $(\mathbf{c}, \mathbf{0})$ is not stable. This shows that the oversmoothing problem, in the sense of definition 7.2.2, is mitigated for the ODEs (7.3) and structure preserving time-discretizations of it such as (7.4), from which, in simple words it follows that *GraphCON mitigates oversmoothing by construction.*

This analysis also illustrates the rich dynamics of (7.3) as we show that even if the trajectories reach a steady state of the form $(\mathbf{c}, \mathbf{0})$, very small perturbations will grow and the trajectory will veer away from this steady state, possibly towards other constant steady states which are also not stable. Thus, the trajectories can sample large parts of the latent space, contributing to the expressive power of the model.

We remark here that the use of ReLU activation function in proposition 7.2.5 is purely for definiteness. Any other widely used activation function can be used in σ, with corresponding zero Dirichlet energy steady states being specified by the roots of the algebraic equation $\sigma(\mathbf{c}) = \mathbf{c}$ and an analogous result can be derived. For instance, the zero-Dirichlet energy steady state corresponding to the Tanh activation function is given by $(\mathbf{0}, \mathbf{0})$.

7.2.3 On the exploding and vanishing gradients problem

The mitigation of oversmoothing by GraphCON has a great bearing on increasing the expressivity of the resulting deep GNN. In addition, it turns out that using graph-coupled oscillators can also facilitate training of the underlying GNNs. To see this, we will consider a concrete example of the coupling function in (7.4) to be GCN (7.5). Other coupling functions such as GAT can be considered analogously. For simplicity of exposition and without any loss of generality, we consider *scalar node features* by setting $m = 1$. We also set $\alpha, \gamma = 1$. With these assumptions, a N-layer deep GraphCON-GCN reduces to the following explicit (node-wise) form,

$$\begin{aligned} \mathbf{Y}_i^n &= (1 - \Delta t)\mathbf{Y}_i^{n-1} + \Delta t \sigma\left(\mathbf{C}_i^{n-1}\right) - \Delta t \mathbf{X}_i^{n-1}, \\ \mathbf{C}_i^{n-1} &= \frac{\omega^n}{d_i}\mathbf{X}_i^{n-1} + \sum_{j \in \mathcal{N}_i} \frac{\omega^n \mathbf{X}_j^{n-1}}{\sqrt{d_i d_j}}, \\ \mathbf{X}_i^n &= \mathbf{X}_i^{n-1} + \Delta t \mathbf{Y}_i^n, \quad \forall 1 \leq n \leq N, \quad \forall 1 \leq i \leq v. \end{aligned} \tag{7.31}$$

Here, $d_i = \deg(i)$, denoting the degree of a node $i \in \mathcal{V}$ and $\omega^n \in \mathbb{R}$, denoting the learnable weight.

Moreover, we are in a setting where the learning task is for the GNN to approximate the *ground truth* vector $\overline{\mathbf{X}} \in \mathbb{R}^v$. Consequently, we set up the following loss-function,

$$\mathbf{J}(\mathbf{w}) := \frac{1}{2v} \sum_{i \in \mathcal{V}} |\mathbf{X}_i^N - \overline{\mathbf{X}}_i|^2, \tag{7.32}$$

with $\mathbf{w} = [\omega^1, \omega^2, \cdots, \omega^N]$ denoting the concatenated learnable weights in (7.31). During training, one computes an approximate minimizer of the loss-function (7.32) with a (stochastic) gradient descent (SGD) procedure. At every step of gradient descent, we need to compute the gradient $\partial_{\mathbf{w}}\mathbf{J}$. For definiteness, we fix layer $1 \leq \ell \leq N$ and consider the learnable weight ω^ℓ. Thus, in a SGD step, one needs to compute gradient, $\frac{\partial \mathbf{J}}{\partial \omega^\ell}$. By chain rule, one readily proves the following identity (see for instance Pascanu et al. [2013]),

$$\frac{\partial \mathbf{J}}{\partial \omega^\ell} = \frac{\partial \mathbf{J}}{\partial \mathbf{Z}^N} \frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} \frac{\partial \mathbf{Z}^\ell}{\partial \omega^\ell}. \tag{7.33}$$

Here,

$$\mathbf{Z}^n = [\mathbf{X}_1^n, \mathbf{Y}_1^n, \mathbf{X}_2^n, \mathbf{Y}_2^n, \cdots, \mathbf{X}_i^n, \mathbf{Y}_i^n, \cdots, \mathbf{X}_v^n, \mathbf{Y}_v^n],$$

is the concatenated node-feature vector at the layer $1 \leq n \leq N$.

Furthermore, by using the product rule, we see that,

$$\frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} = \prod_{n=\ell+1}^{N} \frac{\partial \mathbf{Z}^n}{\partial \mathbf{Z}^{n-1}}. \tag{7.34}$$

In other words, the gradient $\frac{\partial \mathbf{J}}{\partial \omega^\ell}$ measures the contribution made by the node k in the ℓ-th hidden layer to the learning process.

If we assume that the partial gradient behaves as $\frac{\partial \mathbf{Z}^n}{\partial \mathbf{Z}^{n-1}} \sim \lambda$, for all n, then, the long-product structure of (7.34) implies that $\frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} \sim \lambda^{N-\ell}$. If on average, $\lambda > 1$, then we observe that the total gradient (7.33) can grow *exponentially* in the number of layers, leading to the exploding gradients problem. Similarly, if on average, $\lambda < 1$, then the total gradient (7.33) can decay *exponentially* in the number of layers, leading to the vanishing gradients problem. Either of these situations can lead to failure of training as the gradient step either blows up or does not change at all. Hence, for very deep GNN architectures, it is essential to investigate if the exploding and vanishing gradients problem can be mitigated. We start by showing the following upper bound on the gradients. To this end, we first establish an upper bound on the hidden node features of the following general form of GraphCON (7.4), written node-wise as,

$$\begin{aligned} \mathbf{C}_i^{n-1} &= (\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G}))_i, \\ \mathbf{Y}_i^n &= \mathbf{Y}_i^{n-1} + \Delta t \sigma(\mathbf{C}_i^{n-1}) - \gamma \Delta t \mathbf{X}_i^{n-1} - \alpha \Delta t \mathbf{Y}_i^{n-1}, \\ \mathbf{X}_i^n &= \mathbf{X}_i^{n-1} + \Delta t \mathbf{Y}_i^n. \end{aligned} \tag{7.35}$$

We derive following upper bound on the resulting hidden node features,

Proposition 7.2.7. *For all n, let $t_n = n\Delta t$ and the time step Δt satisfy,*

$$\Delta t < \min\left(\frac{\alpha}{\gamma}, \frac{1}{\alpha}\right)$$

Let $\mathbf{X}_i^n$ denote the hidden state vector at any node $i \in \mathcal{V}$ which evolves according to GraphCON (7.35), then the hidden states satisfy the following bound,

$$\begin{aligned} \|\mathbf{X}_i^n\|^2 \leq \|\mathbf{X}_i^0\|^2 + \frac{1}{\gamma}\|\mathbf{Y}_i^0\|^2 \\ + \frac{m\beta^2 t_n}{2\gamma(\alpha - \gamma\Delta t)} \end{aligned} \tag{7.36}$$

where β is the global bound on the underlying activation function σ (7.41).

Proof. We multiply $\gamma(\mathbf{X}_i^{n-1})^\top$ to the third equation of (7.35) and $(\mathbf{Y}_i^n)^\top$ to the second equation of (7.35) and repeatedly use the following elementary identities,

$$\begin{aligned} \mathbf{a}^\top(\mathbf{a} - \mathbf{b}) &= \frac{\|\mathbf{a}\|^2}{2} - \frac{\|\mathbf{b}\|^2}{2} + \frac{1}{2}\|\mathbf{a} - \mathbf{b}\|^2, \\ \mathbf{b}^\top(\mathbf{a} - \mathbf{b}) &= \frac{\|\mathbf{a}\|^2}{2} - \frac{\|\mathbf{b}\|^2}{2} - \frac{1}{2}\|\mathbf{a} - \mathbf{b}\|^2, \end{aligned}$$

to obtain,

$$\begin{aligned}\gamma\frac{\|\mathbf{X}_i^n\|^2}{2}+\frac{\|\mathbf{Y}_i^n\|^2}{2} &= \gamma\frac{\|\mathbf{X}_i^{n-1}\|^2}{2}+\frac{\|\mathbf{Y}_i^{n-1}\|^2}{2}\\ &+\Delta t(\mathbf{Y}_i^n)^\top\sigma(\mathbf{C}_i^{n-1})\\ &+\Delta t\left(\frac{\gamma\Delta t}{2}-\alpha+\frac{\alpha}{2}\right)\|\mathbf{Y}_i^n\|^2\\ &-\frac{\alpha\Delta t}{2}\|\mathbf{Y}_i^{n-1}\|^2\\ &+\left(\frac{\alpha\Delta t-1}{2}\right)\|\mathbf{Y}_i^n-\mathbf{Y}_i^{n-1}\|^2\end{aligned}$$

As we have assumed that the time step Δt is chosen such that

$$\Delta t<\min\left(\frac{\alpha}{\gamma},\frac{1}{\alpha}\right)$$

we obtain from the above inequality that,

$$\begin{aligned}\gamma\frac{\|\mathbf{X}_i^n\|^2}{2}+\frac{\|\mathbf{Y}_i^n\|^2}{2} &\le \gamma\frac{\|\mathbf{X}_i^{n-1}\|^2}{2}+\frac{\|\mathbf{Y}_i^{n-1}\|^2}{2}\\ &+\Delta t(\mathbf{Y}_i^n)^\top\sigma(\mathbf{C}_i^{n-1})\\ &-\Delta t\left(\frac{\alpha-\gamma\Delta t}{2}\right)\|\mathbf{Y}_i^n\|^2\end{aligned}$$

Next we use the elementary identity

$$\mathbf{a}^\top\mathbf{b}\le\frac{\epsilon\|\mathbf{a}\|^2}{2}+\frac{\|\mathbf{b}\|^2}{2\epsilon},$$

with $\epsilon=\alpha-\gamma\Delta t$ in the above inequality to obtain,

$$\begin{aligned}\gamma\frac{\|\mathbf{X}_i^n\|^2}{2}+\frac{\|\mathbf{Y}_i^n\|^2}{2} &\le \gamma\frac{\|\mathbf{X}_i^{n-1}\|^2}{2}+\frac{\|\mathbf{Y}_i^{n-1}\|^2}{2}\\ &+\frac{\Delta t}{2(\alpha-\gamma\Delta t)}\|\sigma(\mathbf{C}_i^{n-1})\|^2\end{aligned}\tag{7.37}$$

Now from the bound (7.41) on the activation function, we obtain from (7.37) that,

$$\begin{aligned}\gamma\frac{\|\mathbf{X}_i^n\|^2}{2}+\frac{\|\mathbf{Y}_i^n\|^2}{2} &\le \gamma\frac{\|\mathbf{X}_i^{n-1}\|^2}{2}+\frac{\|\mathbf{Y}_i^{n-1}\|^2}{2}\\ &+\frac{m\Delta t\beta^2}{2(\alpha-\gamma\Delta t)}\end{aligned}\tag{7.38}$$

Iterating (7.38) over n yields,

$$\begin{aligned}\gamma\|\mathbf{X}_i^n\|^2+\|\mathbf{Y}_i^n\|^2 &\le \gamma\|\mathbf{X}_i^0\|^2+\|\mathbf{Y}_i^0\|^2\\ &+\frac{mn\Delta t\beta^2}{2(\alpha-\gamma\Delta t)},\end{aligned}\tag{7.39}$$

which readily yields the desired inequality (7.36). □

Based on this bound, we can prove the following upper bound on the gradients,

Proposition 7.2.8. *Let $\mathbf{X}^n, \mathbf{Y}^n$ be the node features, generated by Graphcon-GCN* (7.31)*. We assume that $\Delta t << 1$ is chosen to be sufficiently small. Then, the gradient of the loss function* $\mathbf{J}$ (7.32) *with respect to any learnable weight parameter ω^ℓ, for some $1 \leq \ell \leq N$ is bounded as*

$$\left|\frac{\partial \mathbf{J}}{\partial \omega^\ell}\right| \leq \frac{\beta' \hat{D}(\hat{d}+1)\Delta t(1+\Gamma N \Delta t)}{v}\left(\max_{1\leq i\leq v}(|\mathbf{X}_i^0|+|\mathbf{Y}_i^0|+\max_{1\leq i\leq v}|\overline{\mathbf{X}}_i|+\beta\sqrt{N\Delta t}\right)^2. \tag{7.40}$$

Here,

$$\begin{aligned} &\beta = \max_x |\sigma(x)|, \quad \beta' = \max_x |\sigma'(x)|, \quad \hat{d} = \max_{i\in\mathcal{V}} d_i \\ &\hat{D} = \max_{i,j\in\mathcal{V}} \frac{1}{\sqrt{d_i d_j}}, \quad \Gamma := 6 + 4\beta'\hat{D} \max_{1\leq n\leq N} |\omega^n|. \end{aligned} \tag{7.41}$$

Proof. For any $\ell \leq n \leq N$, a tedious yet straightforward computation yields the following representation formula,

$$\frac{\partial \mathbf{Z}^n}{\partial \mathbf{Z}^{n-1}} = \mathrm{I}_{2v\times 2v} + \Delta t \mathbf{E}^{n,n-1} + \Delta t^2 \mathbf{F}^{n,n-1}. \tag{7.42}$$

Here $\mathbf{E}^{n,n-1} \in \mathbb{R}^{2v\times 2v}$ is a matrix whose entries are given below. For any $1 \leq i \leq v$, we have,

$$\begin{aligned} \mathbf{E}^{n,n-1}_{2i-1,2i} &= 1, \\ \mathbf{E}^{n,n-1}_{2i-1,j} &= 0, \quad \forall j \neq 2i, \\ \mathbf{E}^{n,n-1}_{2i,2i} &= -1, \\ \mathbf{E}^{n,n-1}_{2i,2i-1} &= -1 + \frac{\sigma'(\mathbf{C}_i^{n-1})\omega^n}{d_i}, \\ \mathbf{E}^{n,n-1}_{2i,2j} &= 0, \quad \forall 1 \leq j \leq v, \text{ and } j \neq i, \\ \mathbf{E}^{n,n-1}_{2i,2j-1} &= \frac{\sigma'(\mathbf{C}_j^{n-1})\omega^n}{\sqrt{d_i d_j}}, \forall j \in \mathcal{N}_i, \\ \mathbf{E}^{n,n-1}_{2i,2j-1} &= 0, \quad \forall j \notin \mathcal{N}_i \text{ and } j \neq i. \end{aligned}$$

Similarly, $\mathbf{F}^{n,n-1} \in \mathbb{R}^{2v\times 2v}$ is a matrix whose entries are given below. For any $1 \leq i \leq v$, we have,

$$\begin{aligned} \mathbf{F}^{n,n-1}_{2i,j} &= 0, \quad \forall j, \\ \mathbf{F}^{n,n-1}_{2i-1,2i-1} &= -1 + \frac{\sigma'(\mathbf{C}_i^{n-1})\omega^n}{d_i}, \\ \mathbf{F}^{n,n-1}_{2i-1,2j-1} &= \frac{\sigma'(\mathbf{C}_j^{n-1})\omega^n}{\sqrt{d_i d_j}}, \forall j \in \mathcal{N}_i, \\ \mathbf{F}^{n,n-1}_{2i-1,2j-1} &= 0, \quad \forall j \notin \mathcal{N}_i \text{ and } j \neq i. \end{aligned}$$

Using (7.41), it is straightforward to compute that,

$$\begin{aligned} \|\mathbf{E}^{n,n-1}\|_\infty &\leq 2 + \beta'\hat{D}|\omega^n|, \\ \|\mathbf{F}^{n,n-1}\|_\infty &\leq 1 + \beta'\hat{D}|\omega^n|, \end{aligned} \tag{7.43}$$

Then using $\Delta t \leq 1$ and definition (7.41), we have from (7.42) that,

$$\left\|\frac{\partial \mathbf{Z}^n}{\partial \mathbf{Z}^{n-1}}\right\|_\infty \leq 1 + \frac{\Gamma}{2}\Delta t, \quad \forall n.$$

Therefore, from the identity (7.34), we obtain,

$$\left\| \frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} \right\|_\infty \leq \left(1 + \frac{\Gamma}{2}\Delta t\right)^{N-\ell}.$$

Now choosing $\Delta t << 1$ small enough such that the following inequality holds,

$$\left(1 + \frac{\Gamma}{2}\Delta t\right)^{N-\ell} \leq 1 + (N-\ell)\Gamma\Delta t, \tag{7.44}$$

leads to the following bound,

$$\left\| \frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} \right\|_\infty \leq 1 + (N-\ell)\Gamma\Delta t \leq 1 + N\Gamma\Delta t \tag{7.45}$$

A straight-forward differentiation of the loss function (7.32) yields,

$$\frac{\partial \mathbf{J}}{\partial \mathbf{Z}^N} = \frac{1}{v}\left[\mathbf{X}_1^N - \overline{\mathbf{X}}_1, 0, \mathbf{X}_2^N - \overline{\mathbf{X}}_2, 0, \cdots, \mathbf{X}_v^N - \overline{\mathbf{X}}_v, 0\right]. \tag{7.46}$$

Hence,

$$\left\| \frac{\partial \mathbf{J}}{\partial \mathbf{Z}^N} \right\|_\infty \leq \frac{1}{v}\left(\max_{1\leq i\leq v} |\mathbf{X}_i^N| + \max_{1\leq i\leq v} |\overline{\mathbf{X}}_i|\right) \tag{7.47}$$

Applying the pointwise upper bound (7.36) to (7.47), we obtain,

$$\left\| \frac{\partial \mathbf{J}}{\partial \mathbf{Z}^N} \right\|_\infty \leq \frac{1}{v}\left(\max_{1\leq i\leq v} (|\mathbf{X}_i^0| + |\mathbf{Y}_i^0|) + \max_{1\leq i\leq v} |\overline{\mathbf{X}}_i| + \beta\sqrt{N\Delta t}\right) \tag{7.48}$$

Finally, a direct calculation provides the following characterization of the vector $\frac{\partial \mathbf{Z}^\ell}{\partial \omega^\ell} \in \mathbb{R}^{2v}$,

$$\begin{aligned}
\left(\frac{\partial \mathbf{Z}^\ell}{\partial \omega^\ell}\right)_{2j} &= \Delta t \sigma'(\mathbf{C}_j^\ell)\left(\frac{\mathbf{X}_j^{\ell-1}}{d_j} + \sum_{k\in\mathcal{N}_j} \frac{\mathbf{X}_k^{\ell-1}}{\sqrt{d_j d_k}}\right), \\
\left(\frac{\partial \mathbf{Z}^\ell}{\partial \omega^\ell}\right)_{2j-1} &= \Delta t^2 \sigma'(\mathbf{C}_j^\ell)\left(\frac{\mathbf{X}_j^{\ell-1}}{d_j} + \sum_{k\in\mathcal{N}_j} \frac{\mathbf{X}_k^{\ell-1}}{\sqrt{d_j d_k}}\right).
\end{aligned} \tag{7.49}$$

Therefore using the pointwise bound (7.36), one can readily calculate that,

$$\begin{aligned}
\left\| \frac{\partial \mathbf{Z}^\ell}{\partial \omega^\ell} \right\|_\infty &\leq \Delta t \beta' \hat{D}(\hat{d}+1)\left(\max_{1\leq i\leq v} (|\mathbf{X}_i^0| + |\mathbf{Y}_i^0|) + \beta\sqrt{\ell\Delta t}\right) \\
&\leq \Delta t \beta' \hat{D}(\hat{d}+1)\left(\max_{1\leq i\leq v} (|\mathbf{X}_i^0| + |\mathbf{Y}_i^0|) + \max_{1\leq i\leq v} |\overline{\mathbf{X}}_i| + \beta\sqrt{N\Delta t}\right)
\end{aligned} \tag{7.50}$$

Multiplying (7.45), (7.48) and (7.50) and using the product rule (7.34) yields the desired upper bound (7.40). □

The upper bound (7.40) clearly shows that the total gradient is globally bounded, independent of the number of layers N, if $\Delta t \sim N^{-1}$, thus mitigating the exploding gradients problem. Even if the small parameter Δt is chosen independently of the number of layers N, the total gradient in (7.40) only grows, at most quadratically in the number of layers, thus preventing exponential blowup of gradients and mitigating the exploding gradients problem. However, this upper bound (7.40) does not necessarily rule out the vanishing gradients problem. To this end, we derive the following formula for the gradients,

Proposition 7.2.9. *For $1 \leq n \leq N$, let $\mathbf{X}^n$ be the node features generated by GraphCON-GCN (7.31), Then for sufficiently small $\Delta t << 1$, the gradient $\frac{\partial \mathbf{J}}{\partial \omega^\ell}$, for any ℓ satisfies the following expression,*

$$\begin{aligned}\frac{\partial \mathbf{J}}{\partial \omega^\ell} =& \frac{\Delta t^2}{v} \sum_{j \in \mathcal{V}} \sum_{k \in \mathcal{N}_j \cup \{j\}} \frac{\sigma'(\mathbf{C}_j^{\ell-1}) \mathbf{X}_k^{\ell-1} \left(\mathbf{X}_j^N - \overline{\mathbf{X}}_j\right)}{\sqrt{d_j d_k}} \\ &+ \mathcal{O}(\Delta t^3),\end{aligned} \tag{7.51}$$

with the order notation being defined in (2.35).

Proof. The key ingredient in the proof is the following representation formula,

$$\frac{\partial \mathbf{Z}^N}{\partial \mathbf{Z}^\ell} = \mathrm{I}_{2v \times 2v} + \Delta t \sum_{n=\ell+1}^{N} \mathbf{E}^{n,n-1} + \mathcal{O}(\Delta t^2), \tag{7.52}$$

the proof of which follows directly from the identity (7.42) and the boundedness of the matrices $\mathbf{E}, \mathbf{F}$ in (7.42).

Then, (7.51) follows from a multiplication of (7.46), (7.52) and (7.49) and a straightforward rearrangement of the terms, □

One readily observes from the formula (7.51), that to leading order in the small parameter Δt, the gradient $\frac{\partial \mathbf{J}}{\partial \omega^\ell}$ is *independent* of the number of layers N of the underlying GNN. Thus, although the gradient can be small (due to small Δt), it will not vanish by increasing the number of layers, mitigating the vanishing gradient problem. Even if small parameter Δt depends on the number of layers, as long as this dependence is polynomial i.e., $\Delta t \sim N^{-s}$, for some s, the gradient cannot decay exponentially in N, alleviating the vanishing gradients problem in this case too.

7.3 Experimental results

We present a detailed experimental evaluation of the proposed framework on a variety of graph learning tasks. We test two settings of GraphCON: GraphCON-GCN (using graph convolution as the 1-neighborhood coupling in (7.4)) and GraphCON-GAT (using the attentional coupling). Since in most experiments, these two configurations already outperform the state-of-the-art, we only apply GraphCON with more involved coupling functions in a few particular tasks.

Evolution of Dirichlet Energy. We start by illustrating the dynamics of the Dirichlet energy (6.2) of GraphCON for an undirected graph representing a 2-dimensional 10×10 regular grid with 4-neighbor connectivity. The node features $\mathbf{X}$ are randomly sampled from $\mathcal{U}([0, 1])$ and then propagated through 100-layer GNNs (with random weights): GAT, GCN, and their GraphCON-stacked versions (GraphCON-GAT and GraphCON-GCN) for two different values of the damping parameter $\alpha = 0, 0.5$ in (7.4) and with fixed $\gamma = 1$. In Fig. 7.2, we plot the (logarithm of) Dirichlet energy of each layer's output with respect to (logarithm) of the layer number. It can clearly be seen that GAT and GCN suffer from the oversmoothing problem as the Dirichlet energy converges exponentially fast to zero, indicating that the node features become constant, while GraphCON is devoid of this behavior. This holds true even for non-zero value of the damping parameter α, where the Dirichlet energy stabilizes after an initial decay.

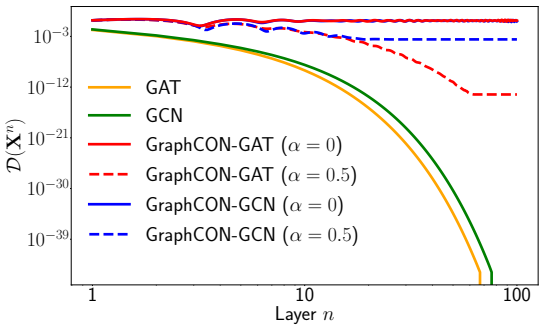

Figure 7.2: Dirichlet energy $\mathcal{D}(\mathbf{X}^n)$ of layer-wise node features $\mathbf{X}^n$ propagated through a GAT and GCN as well as their GraphCON-stacked versions (GraphCon-GAT and GraphCON-GCN) for two different values of $\alpha = 0, 0.5$ in (7.4) and fixed $\gamma = 1$.

Transductive node classification. We evaluate GraphCON on both homophilic and heterophilic datasets, where high homophily implies that the features in a node are similar to those of its neighbors. The homophily level reported in Table 7.1 and Table 7.2 is the measure proposed by Pei et al. [2020].

Homophilic datasets: We consider three widely used node classification tasks, based on the citation networks Cora [McCallum et al., 2000], Citeseer [Sen et al., 2008] and Pubmed [Namata et al., 2012]. We follow the evaluation protocols and training, validation, and test splits of Shchur et al. [2018], Chamberlain et al. [2021b], using only on the largest connected component in each network.

Table 7.1 compares GraphCON with standard GNN baselines: GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018], MoNet [Monti et al., 2017], GraphSAGE (GS) [Hamilton et al., 2017], CGNN [Xhonneux et al., 2020], GDE [Poli et al., 2019], and GRAND Chamberlain et al. [2021b]. We observe that GraphCON-GCN and GraphCON-GAT outperform pure GCN and GAT consistently. We also provide results for GraphCON based on the propagation layer used in GRAND i.e., transformer Vaswani et al. [2017] based graph attention, referred to as GraphCON-Tran, which also outperforms the basic underlying model. Overall, GraphCON models show the best performance on all these datasets.

Heterophilic datasets: We also evaluate GraphCON on the heterophilic graphs; Cornell, Texas and Wisconsin from the WebKB dataset. Here, the assumption on neighbor feature similarity does not hold. Many GNN models were shown to struggle in this settings as can be seen by the poor performance of baseline GCN and GAT in Table 7.2. On the other hand, we see from Table 7.2 that not only do GraphCON-GCN and GraphCON-GAT dramatically outperform the underlying GCN and GAT models (e.g. for the most heterophilic Texas graph, GraphCON-GCN and GraphCON-GAT have mean accuracies of 85.4% and 82.2%, compared to accuracies of 55.1% and 52.2% for GCN and GAT), the GraphCON models also provide the best performance, outperforming recent baselines that are specifically designed for heterophilic graphs.

Inductive node classification. In this experiment, we consider the Protein-Protein-Interaction (PPI) dataset of Zitnik and Leskovec [2017], using the protocol of Hamilton et al. [2017]. Table 7.3 shows the test performance (micro-average F1) of GraphCON and several standard GNN baselines. We can see that GraphCON significantly improves the performance of the underling models (GAT from 97.4% to 99.4% and GCN from 98.5% to 99.6%, which is the top result on this benchmark).

Table 7.1: Transductive node classification test accuracy (MAP in %) on homophilic datasets. Mean and standard deviation are obtained using 20 random initializations on 5 random splits each. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

	Cora	**Citeseer**	**Pubmed**
Homophily level	**0.81**	**0.74**	**0.80**
GAT-ppr	81.6 ± 0.3	68.5 ± 0.2	76.7 ± 0.3
MoNet	81.3 ± 1.3	71.2 ± 2.0	78.6 ± 2.3
GraphSage-mean	79.2 ± 7.7	71.6 ± 1.9	77.4 ± 2.2
GraphSage-maxpool	76.6 ± 1.9	67.5 ± 2.3	76.1 ± 2.3
CGNN	81.4 ± 1.6	66.9 ± 1.8	66.6 ± 4.4
GDE	78.7 ± 2.2	71.8 ± 1.1	73.9 ± 3.7
GCN	81.5 ± 1.3	71.9 ± 1.9	77.8 ± 2.9
GraphCON-GCN	81.9 ± 1.7	72.9 ± 2.1	**78.8 ± 2.6**
GAT	81.8 ± 1.3	71.4 ± 1.9	78.7 ± 2.3
GraphCON-GAT	**83.2 ± 1.4**	**73.2 ± 1.8**	**79.5 ± 1.8**
GRAND	**83.6 ± 1.0**	**73.4 ± 0.5**	**78.8 ± 1.7**
GraphCON-Tran	**84.2 ± 1.3**	**74.2 ± 1.7**	**79.4 ± 1.3**

Table 7.2: Transductive node classification test accuracy (MAP in %) on heterophilic datasets. All results represent the average performance of the respective model over 10 fixed train/val/test splits, which are taken from Pei et al. [2020]. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

	Texas	**Wisconsin**	**Cornell**
Homophily level	**0.11**	**0.21**	**0.30**
GPRGNN	78.4 ± 4.4	82.9 ± 4.2	80.3 ± 8.1
H2GCN	**84.9 ± 7.2**	**87.7 ± 5.0**	**82.7 ± 5.3**
GCNII	77.6 ± 3.8	80.4 ± 3.4	77.9 ± 3.8
Geom-GCN	66.8 ± 2.7	64.5 ± 3.7	60.5 ± 3.7
PairNorm	60.3 ± 4.3	48.4 ± 6.1	58.9 ± 3.2
GraphSAGE	**82.4 ± 6.1**	81.2 ± 5.6	76.0 ± 5.0
MLP	80.8 ± 4.8	85.3 ± 3.3	81.9 ± 6.4
GAT	52.2 ± 6.6	49.4 ± 4.1	61.9 ± 5.1
GraphCON-GAT	82.2 ± 4.7	**85.7 ± 3.6**	**83.2 ± 7.0**
GCN	55.1 ± 5.2	51.8 ± 3.1	60.5 ± 5.3
GraphCON-GCN	**85.4 ± 4.2**	**87.8 ± 3.3**	**84.3 ± 4.8**

Molecular graph property regression. We reproduce the benchmark proposed in Dwivedi et al. [2020], regressing the constrained solubility of 12K molecular graphs from the ZINC dataset [Irwin et al., 2012]. We follow verbatim the settings of Dwivedi et al. [2020], Beani et al. [2021]: make no use of edge features and constrain the network sizes to $\sim$100K parameters. Table 7.4 summarizes the performance of GraphCON and standard GNN baselines. Both GraphCON-GAT and GraphCON-GCN outperform GAT

Table 7.3: Test micro-averaged F_1 score on Protein-Protein Interactions (PPI) data set. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	Micro-averaged F_1
VR-GCN [Chen et al., 2017]	97.8
GraphSAGE [Hamilton et al., 2017]	61.2
PDE-GCN [Eliasof et al., 2021]	99.2
GCNII [Chen et al., 2020b]	**99.5**
Cluster-GCN [Chiang et al., 2019]	**99.4**
GeniePath Liu et al. [2019b]	98.5
JKNet [Xu et al., 2018b]	97.6
GAT [Velickovic et al., 2018]	97.3
GraphCON-GAT	**99.4**
GCN [Kipf and Welling, 2017]	98.5
GraphCON-GCN	**99.6**

and GCN respectively, by a factor of 2. Moreover, the performance of GraphCON-GCN is on par with the recent state-of-the-art method DGN [Beani et al., 2021] with significantly lower standard deviation.

Table 7.4: Test mean absolute error (MAE, averaged over 4 runs on different initializations) on ZINC (**without edge features, small 12k version**) restricted to small network sizes of $\sim 100k$ parameters. Baseline results are taken from Beani et al. [2021]. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	Test MAE
GIN [Xu et al., 2018a]	0.41 ± 0.008
GatedGCN [Bresson and Laurent, 2017]	0.42 ± 0.006
GraphSAGE Hamilton et al. [2017]	0.41 ± 0.005
MoNet [Monti et al., 2017]	0.41 ± 0.007
PNA [Corso et al., 2020]	$\mathbf{0.32 \pm 0.032}$
DGN [Beani et al., 2021]	$\mathbf{0.22 \pm 0.010}$
GCN [Kipf and Welling, 2017]	0.47 ± 0.002
GraphCON-GCN	$\mathbf{0.22 \pm 0.004}$
GAT [Velickovic et al., 2018]	0.46 ± 0.002
GraphCON-GAT	$\mathbf{0.23 \pm 0.004}$

MNIST Superpixel graph classification. This experiment, first suggested by Monti et al. [2017], is based on the MNIST dataset [LeCun et al., 1998], where the grey-scale images are transformed into irregular graphs, as follows: the vertices in the graphs represent superpixels (large blobs of similar color), while the edges represent their spatial adjacency. Each graph has a fixed number of 75 superpixels (vertices). We use the standard splitting of using 55K-5K-10K for training, validation, and testing.

Table 7.5 shows that GraphCON-GCN dramatically improves the performance of a pure GCN (test accuracy of 88.89% vs 98.70%). We stress that both models share the parameters over all layers, i.e.

GraphCON-GCN does not have more parameters despite being a deeper model. Thus, the better performance of GraphCON-GCN over GCN can be attributed to the use of more 'layers' (iterations) and not to a higher number of parameters (see Table 7.8 for accuracy vs. number of layers for this testcase). Finally, Table 7.5 also shows that GraphCON-GAT outperforms all other methods, including the recently proposed PNCNN Finzi et al. [2021], reaching a nearly-perfect test accuracy of 98.91%.

Table 7.5: Test accuracy in % on MNIST Superpixel 75. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

Model	Test accuracy
ChebNet [Defferrard et al., 2016]	75.62
MoNet [Monti et al., 2017]	91.11
PNCNN [Finzi et al., 2021]	**98.76**
SplineCNN [Fey et al., 2018]	95.22
GIN [Xu et al., 2018a]	97.23
GraphCON-GIN	98.53
GatedGCN [Bresson and Laurent, 2017]	97.95
GraphCON-GatedGCN	98.27
GCN [Kipf and Welling, 2017]	88.89
GraphCON-GCN	**98.68**
GAT [Velickovic et al., 2018]	96.19
GraphCON-GAT	**98.91**

Training details. All experiments were run on NVIDIA GeForce GTX 1080 Ti, RTX 2080 Ti as well as RTX 2080 Ti GPUs. The tuning of the hyperparameters was done using a standard random search algorithm. We fix the time-step Δt in (7.4) to 1 in all experiments. The damping parameter α as well as the frequency control parameter γ are set to 1 for all Cora, Citeseer and Pubmed experiments, while we set them to 0 for all experiments based on the Texas, Cornell and Wisconsin network graphs. For all other experiments we include α and γ to the hyperparameter search-space. The tuned values can be found in Table 7.6.

Table 7.6: Hyperparameters α and γ of GraphCON (7.4) for each best performing GraphCON model (based on a validation set).

Model	**Experiment**	α	γ
GraphCON-GCN	**PPI**	0.242	1.0
GraphCON-GAT		0.785	1.0
GraphCON-GCN	**ZINC**	0.215	1.115
GraphCON-GAT		1.475	1.324
GraphCON-GCN	**MNIST (superpixel)**	1.0	0.76
GraphCON-GAT		0.76	0.105

7.4 Further empirical analysis

Performance of GraphCON with respect to number of layers. GraphCON is designed to be a deep GNN architecture with many layers. Depth could enhance the expressive power of GraphCON and we investigate this issue in three of the datasets, presented in Section 7.3. In the first two experiments, we will focus on the GraphCON-GCN model and compare and contrast its performance, with respect to increasing depth, with the baseline GCN model.

We start with the molecular graph property regression example for the ZINC dataset of Irwin et al. [2012]. In Table 7.7, we present the mean absolute error (MAE) of the model on the test set with respect to increasing number of layers (up to 20 layers) of the respective GNNs. As observed from this table, the MAE with standard GCN increases with depth. On the other hand, the MAE with GraphCON decreases as more layers are added.

Table 7.7: Test mean absolute errors of GraphCON-GCN as well as its baseline model GCN on the ZINC task for different number of layers $N = 5, 10, 15, 20$.

Model	Layers			
	5	10	15	20
GraphCON-GCN	0.241	0.233	0.228	0.214
GCN	0.442	0.463	0.478	0.489

Next, we consider the MNIST Superpixel graph classification task and present the test accuracy with increasing depth (number of layers) for both GCN and GraphCON-GCN. As in the previous example, we observe that increasing depth leads to worsening of the test accuracy for GCN. On the other hand, the test accuracy for GraphCON-GCN increases as more layers (up to 32 layers) are added to the model.

Table 7.8: Test accuracies in % of GraphCON-GCN as well as its baseline model GCN on the MNIST Superpixel 75 task for different number of layers $N = 4, 8, 16, 32$.

Model	Layers			
	4	8	16	32
GraphCON-GCN	97.78	98.51	98.55	98.68
GCN	88.09	87.26	86.78	85.67

Additionally, we compare the performance of GraphCON-GCN to GCN with EdgeDrop [Rong et al., 2020] (GCN+EdgeDrop), which has been specifically designed to mitigate the oversmoothing phenomenon for deeper GNN models. We consider the Cora node-based classification task in the semi-supervised setting, where we compare GraphCON-GCN to GCN+DropEdge for increasing number of layers $N = 2, 4, 8, 16, 32, 64$. We observe in Table 7.9 that GraphCON improves (or retains) performance for a large increase in the number of layers, in contrast to plain GCN+DropEdge on this task. Thus, all three experiments demonstrate that GraphCON leverages more depth to improve performance.

Table 7.9: Test accuracies in % of GraphCON-GCN as well as of GCN+DropEdge on cora (semi-supervised setting) for different number of layers $N = 2, 4, 8, 16, 32, 64$. The GCN+DropEdge results are taken from https://github.com/DropEdge/DropEdge

Model	Layers					
	2	4	8	16	32	64
GraphCON-GCN	82.20	82.78	83.53	84.85	82.95	82.12
GCN+DropEdge	82.80	82.00	75.80	75.70	62.50	49.50

Sensitivity of performance of GraphCON to hyperparameters α and γ. We recall that GraphCON (7.4) has two additional hyperparameters, namely the damping parameter $\alpha \geq 0$ and the frequency control parameter $\gamma > 0$. In Table 7.6, we present the values of α, γ that led to the best performance of the resulting GraphCON models. It is natural to ask how sensitive the performance of GraphCON is to the variation of these hyperparameters. To this end, we choose the MNIST Superpixel graph classification task and perform a sensitivity study of the GraphCON-GCN model with respect to these hyperparameters. First, we fix a value of $\gamma = 0.76$ (corresponding to the best results in Table 7.6) and vary α in the range of $\alpha \in [0, 2]$. The results are plotted in Fig. 7.3 and show that the accuracy is extremely robust to a very large parameter range in α. Only for large values $\alpha > 1.6$, we see that the accuracy deteriorates when the damping is too high.

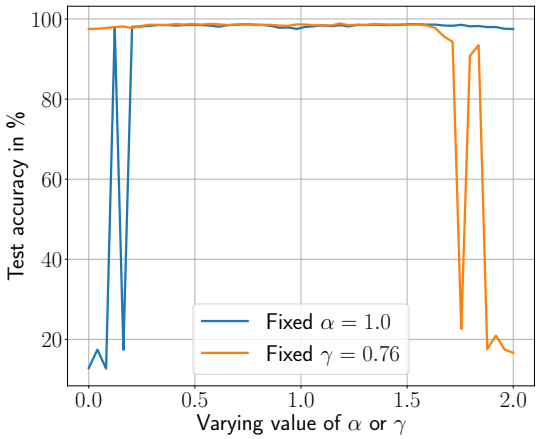

Figure 7.3: Sensitivity (measured as test accuracy) plot for α and γ hyperparameters of GraphCON-GCN (with 32 layers) trained on MNIST superpixel 75 experiment. First, $\alpha = 1.0$ is fixed and γ is varied in $[0, 2]$. Second, $\gamma = 0.76$ is fixed and α is varied in $[0, 2]$. The fixed α, γ are taken from the best performing GraphCON-GCN on the MNIST superpixel 75 task (Table 7.6)

Next for this model and task, we fix $\alpha = 1$ (which provides the best performance as reported in Table 7.6) and vary $\gamma \in [0, 2]$. Again, for a large range of values corresponding to $\gamma \in [0.2, 2]$, the accuracy is

very robust. However, for very small values of γ, the accuracy falls significantly. This is to be expected as the model loses its interpretation as system of oscillators for $\gamma \approx 0$.

Thus, these sensitivity results demonstrate that GraphCON performs very robustly with respect to variations of the parameters α, γ, within a reasonable range.

7.5 Discussion

In conclusion, we proposed a novel framework for designing deep Graph Neural Networks called GraphCON, based on suitable time discretizations of ODEs (7.1) that model the dynamics of a network of controlled and damped oscillators. The coupling between the nodes is conditioned on the structure of the underlying graph.

One can readily interpret GraphCON as a framework to propagate information through multiple layers of a deep GNN, where each hidden layer has the same structure as standard GNNs such as GAT, GCN etc. Unlike in canonical constructions of deep GNNs, which stack hidden layers in a straightforward iterative fashion (7.7), GraphCON stacks them in a more involved manner using the dynamics of the ODE (7.3). Hence, in principle, any GNN hidden layer can serve as the coupling function $\mathbf{F}_\theta$ in GraphCON (7.4), offering it as an attractive framework for constructing very deep GNNs.

The well-known oversmoothing problem for GNNs was described mathematically in terms of the stability of zero Dirichlet energy steady states of the underlying ODE (7.3). We showed that such zero Dirichlet energy steady states of (7.3), which lead to constant node features, are not (exponentially) stable. Even if a trajectory reaches a feature vector that is constant across all nodes, very small perturbations will nudge it away and the resulting node features will deviate from each other. Thus, by construction, we demonstrated that the oversmoothing problem, in the sense of definition 7.2.2, is mitigated for GraphCON.

In addition to increasing expressivity by mitigating the oversmoothing problem, GraphCON was rigorously shown to mitigate the exploding and vanishing gradients problem. Consequently, using coupled oscillators also facilitates efficient training of the resulting GNNs.

Finally, we extensively test GraphCON on a variety of node- and graph-classification and regression tasks, including heterophilic datasets known to be challenging for standard GNN models. From these experiments, we observed that (i) GraphCON models significantly outperform the underlying base GNN such as GCN or GAT and (ii) GraphCON models are either on par with or outperform state-of-the-art models on these tasks. This shows that ours is a novel, flexible, easy to use framework for constructing deep GNNs with theoretical guarantees and solid empirical performance.

Chapter 8

Gradient Gating

Standard Message-Passing GNN architectures (MPNNs), such as GCN [Kipf and Welling, 2017] or GAT [Velickovic et al., 2018], update each node at exactly the *same rate* within every hidden layer. Yet, realistic learning tasks might benefit from having different rates of propagation (flow) of information on the underlying graph. This insight leads to a novel *multi-rate message-passing* scheme capable of learning these underlying rates. Moreover, we also propose a novel procedure that harnesses graph gradients to ameliorate the oversmoothing problem. Combining these elements leads to a new architecture, which we term **Gradient Gating** ($\mathbf{G}^2$).

Main Contributions. We will demonstrate the following advantages of the proposed approach:

- $\mathbf{G}^2$ is a flexible framework wherein any standard message-passing layer (such as GAT, GCN, GIN, or GraphSAGE) can be used as the coupling function. Thus, it should be thought of as a framework into which one can plug existing GNN components. The use of multiple rates and gradient gating facilitates the implementation of deep GNNs and generally improves performance.
- $\mathbf{G}^2$ can be interpreted as a discretization of a dynamical system governed by nonlinear differential equations. By investigating the stability of zero-Dirichlet energy steady states of this system, we rigorously prove that our gradient gating mechanism prevents oversmoothing. To complement this, we also prove a partial converse, that the lack of gradient gating can lead to oversmoothing.
- We provide extensive empirical evidence demonstrating that $\mathbf{G}^2$ achieves state-of-the-art performance on a variety of graph learning tasks, including on large heterophilic graph datasets.

8.1 The proposed Gradient Gating framework

We recall the $v \times m$-dimensional feature matrix $\mathbf{X}$ defined on the undirected graph $\mathcal{G}$. Based on this, a typical residual Message-Passing GNN (MPNN) updates the node features by performing several iterations of the form,

$$\mathbf{X}^n = \mathbf{X}^{n-1} + \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})), \tag{8.1}$$

where $\mathbf{F}_\theta$ is a *learnable* function with parameters θ, and σ is an element-wise nonlinear activation function. Here $n \geq 1$ denotes the n-th hidden layer with $n = 0$ being the input.

One can interpret (8.1) as a discrete dynamical system in which $\mathbf{F}$ plays the role of a *coupling function* determining the interaction between different nodes of the graph. In particular, we consider local

(1-neighborhood) coupling of the form $\mathbf{Y}_i = (\mathbf{F}(\mathbf{X}, \mathcal{G}))_i = \mathbf{F}(\mathbf{X}_i, \{\!\{\mathbf{X}_{j\in\mathcal{N}_i}\}\!\})$ operating on the multiset of 1-neighbors of each node. Examples of such functions used in the graph machine learning literature [Bronstein et al., 2021] are *graph convolutions* $\mathbf{Y}_i = \sum_{j\in\mathcal{N}_i} c_{ij}\mathbf{X}_j$ (GCN, [Kipf and Welling, 2017]) and *graph attention* $\mathbf{Y}_i = \sum_{j\in\mathcal{N}_i} a(\mathbf{X}_i, \mathbf{X}_j)\mathbf{X}_j$ (GAT, [Velickovic et al., 2018]).

We observe that in (8.1), at each hidden layer, every node and every feature channel gets updated with exactly the same rate. However, it is reasonable to expect that in realistic graph learning tasks one can encounter multiple rates for the flow of information (node updates) on the graph. Based on this observation, we propose a **multi-rate (MR)** generalization of (8.1), allowing updates to each node of the graph and feature channel with different rates,

$$\mathbf{X}^n = (1 - \boldsymbol{\tau}^n) \odot \mathbf{X}^{n-1} + \boldsymbol{\tau}^n \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})), \tag{8.2}$$

where $\boldsymbol{\tau}$ denotes a $v \times m$ matrix of rates with elements $\boldsymbol{\tau}_{ik} \in [0, 1]$. Rather than fixing $\boldsymbol{\tau}$ prior to training, we aim to learn the different update rates based on the node data $\mathbf{X}$ and the local structure of the underlying graph $\mathcal{G}$, as follows

$$\boldsymbol{\tau}^n(\mathbf{X}^{n-1}, \mathcal{G}) = \bar{\sigma}(\hat{\mathbf{F}}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})), \tag{8.3}$$

where $\hat{\mathbf{F}}_{\hat{\theta}}$ is another learnable 1-neighborhood coupling function, and $\bar{\sigma}$ is a sigmoidal logistic activation function to constrain the rates to lie within $[0, 1]$. Since the multi-rate message-passing scheme (8.2) using (8.3) does not necessarily prevent oversmoothing (for any choice of the coupling function), we need to further constrain the rate matrix $\boldsymbol{\tau}^n$. To this end, we note that the *graph gradient* of scalar node features $\mathbf{y}$ on the underlying graph $\mathcal{G}$ is defined as $(\nabla\mathbf{y})_{ij} = \mathbf{y}_j - \mathbf{y}_i$ at the edge $i \sim j$ [Lim, 2015]. Next, we will use graph gradients to obtain the proposed **Gradient Gating** ($\mathbf{G}^2$) framework given by

$$\begin{aligned}
\hat{\boldsymbol{\tau}}^n &= \sigma(\hat{\mathbf{F}}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})), \\
\boldsymbol{\tau}^n_{ik} &= \tanh\left(\sum_{j\in\mathcal{N}_i} |\hat{\boldsymbol{\tau}}^n_{jk} - \hat{\boldsymbol{\tau}}^n_{ik}|^p\right), \\
\mathbf{X}^n &= (1 - \boldsymbol{\tau}^n) \odot \mathbf{X}^{n-1} + \boldsymbol{\tau}^n \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})),
\end{aligned} \tag{8.4}$$

where $\hat{\boldsymbol{\tau}}^n_{jk} - \hat{\boldsymbol{\tau}}^n_{ik} = (\nabla\hat{\boldsymbol{\tau}}^n_{*k})_{ij}$ denotes the graph-gradient and $\hat{\boldsymbol{\tau}}^n_{*k}$ is the k-th column of the rate matrix $\hat{\boldsymbol{\tau}}^n$ and $p \geq 0$. Since $\sum_{j\in\mathcal{N}_i} |\hat{\boldsymbol{\tau}}^n_{jk} - \hat{\boldsymbol{\tau}}^n_{ik}|^p \geq 0$ for all $i \in \mathcal{V}$, it follows that $\boldsymbol{\tau}^n \in [0, 1]^{v\times m}$ for all n, retaining its interpretation as a matrix of rates. The sum over the neighborhood $\mathcal{N}_i$ in (8.4) can be replaced by any permutation-invariant aggregation function (e.g., mean or max). Moreover, any standard message-passing procedure can be used to define the coupling functions $\mathbf{F}$ and $\hat{\mathbf{F}}$ (and, in particular, one can set $\hat{\mathbf{F}} = \mathbf{F}$). As an illustration, Fig. 8.1 shows a schematic diagram of the layer-wise update of the proposed $\mathbf{G}^2$ architecture.

The intuitive idea behind gradient gating in (8.4) is the following: If for any node $i \in \mathcal{V}$ local oversmoothing occurs, i.e., $\lim_{n\to\infty} \sum_{j\in\mathcal{N}_i} \|\mathbf{X}^n_i - \mathbf{X}^n_j\| = 0$, then $\mathbf{G}^2$ ensures that the corresponding rate $\boldsymbol{\tau}^n_i$ goes to zero (at a faster rate), such that the underlying hidden node feature $\mathbf{X}_i$ is no longer updated. This prevents oversmoothing by *early-stopping* of the message-passing procedure.

Related Work. Gating is a key component of our proposed framework. The use of gating (i.e., the modulation between 0 and 1) of hidden layer outputs has a long pedigree in neural networks and sequence modeling. In particular, classical recurrent neural network (RNN) architectures such as LSTM [Hochreiter and Schmidhuber, 1997] and GRU [Cho et al., 2014] rely on gates to modulate information propagation in the RNN. Given the connections between RNNs and early versions of GNNs [Zhou et al., 2019], it is not

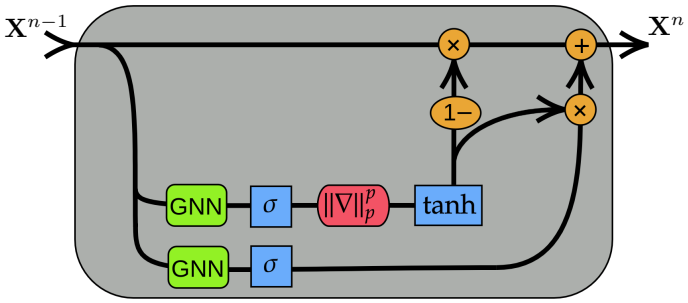

Figure 8.1: Schematic diagram of $\mathbf{G}^2$ (8.4) showing the layer-wise update of the latent node features $\mathbf{X}$ (at layer n). The norm of the graph-gradient (i.e., sum in second equation in (8.4)) is denoted as $\|\nabla\|_p^p$.

surprising that the idea of gating has been used in designing GNNs Bresson and Laurent [2017], Li et al. [2016], Zhang et al. [2018]. However, to the best of our knowledge, the use of local graph-gradients to further modulate gating in order to alleviate the oversmoothing problem is novel, and so is its theoretical analysis.

The multi-rate gating procedure used in $\mathbf{G}^2$ is a particular example of *multiscale* mechanisms. The use of multiscale neural network architectures has a long history. An early example is Hinton and Plaut [1987], who proposed a neural network with each connection having a fast changing weight for temporary memory and a slow changing weight for long-term learning. The classical convolutional neural networks (CNNs, LeCun et al. [1989]) can be viewed as multiscale architectures for processing multiple *spatial* scales in images [Bai et al., 2020]. Moreover, there is a close connection between our multi-rate mechanism (8.4) and the use of multiple time scales in recently proposed sequence models such as UnICORNN (Chapter 3) and LEM (Chapter 5).

Ordinary and partial differential equations (ODEs and PDEs) are playing an increasingly important role in designing, interpreting, and analyzing novel graph machine learning architectures Avelar et al. [2019], Poli et al. [2019], Zhuang et al. [2020], Xhonneux et al. [2020]. Chamberlain et al. [2021b] designed attentional GNNs by discretizing parabolic diffusion-type PDEs. Di Giovanni et al. [2022] interpreted GCNs as gradient flows minimizing a generalized version of the Dirichlet energy. Chamberlain et al. [2021a] applied a non-Euclidean diffusion equation ("Beltrami flow") yielding a scheme with adaptive spatial derivatives ("graph rewiring"), and Topping et al. [2021] studied a discrete geometric PDE similar to Ricci flow to improve information propagation in GNNs. Eliasof et al. [2021] proposed a GNN framework arising from a mixture of parabolic (diffusion) and hyperbolic (wave) PDEs on graphs with convolutional coupling operators, which describe dissipative wave propagation. Finally, in Chapter 7 we used systems of nonlinear oscillators coupled through the associated graph structure to rigorously overcome the oversmoothing problem. In line with these works, one contribution of this chapter is a continuous version of $\mathbf{G}^2$ (8.9), which we use for a rigorous analysis of the oversmoothing problem. Understanding whether this system of ODEs has an interpretation as a known physical model is a topic for future research.

8.2 Rigorous analysis of Gradient Gating

$\mathbf{G}^2$ is a flexible framework. An important aspect of $\mathbf{G}^2$ (8.4) is that it can be considered as a "wrapper" around any specific MPNN architecture. In particular, the hidden layer update for *any* form of message-passing (e.g., GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018], GIN [Xu et al., 2018a] or GraphSAGE [Hamilton et al., 2017]) can be used as the coupling functions $\mathbf{F}, \hat{\mathbf{F}}$ in (8.4). By setting $\boldsymbol{\tau} \equiv \mathbf{I}$, (8.4) reduces to

$$\mathbf{X}^n = \sigma\left(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})\right), \tag{8.5}$$

a standard (non-residual) MPNN. As we will show in the following, the use of a non-trivial gradient-gated *learnable* rate matrix $\boldsymbol{\tau}$ allows implementing very deep architectures that avoid oversmoothing.

Maximum Principle for node features. Node features produced by $\mathbf{G}^2$ satisfy the following Maximum Principle.

Proposition 8.2.1. *Let $\mathbf{X}^n$ be the node feature matrix generated by iteration formula* (8.4)*. Then, the features are bounded as follows:*

$$\min\left(-1, \underline{\sigma}\right) \leq \mathbf{X}^n_{ik} \leq \max\left(1, \overline{\sigma}\right), \quad \forall 1 \leq n, \tag{8.6}$$

where the scalar activation function is bounded by $\underline{\sigma} \leq \sigma(z) \leq \overline{\sigma}$ for all $z \in \mathbb{R}$.

The proof follows readily from writing (8.4) component-wise and using the fact that $0 \leq \boldsymbol{\tau}^n_{ik} \leq 1$, for all $1 \leq i \leq v$, $1 \leq k \leq m$ and $1 \leq n$.

Continuous limit of $\mathbf{G}^2$. It has recently been shown (see Avelar et al. [2019], Poli et al. [2019], Zhuang et al. [2020], Xhonneux et al. [2020], Chamberlain et al. [2021b], Eliasof et al. [2021], Chamberlain et al. [2021a], Topping et al. [2021] and references therein) that interesting properties of GNNs (with residual connections) can be understood by taking the continuous (infinite-depth) limit and analyzing the resulting differential equations.

In this context, we can derive a continuous version of (8.4) by introducing a small-scale $0 < \Delta t < 1$ and rescaling the rate matrix $\boldsymbol{\tau}^n$ to $\Delta t \boldsymbol{\tau}^n$ leading to

$$\mathbf{X}^n = (1 - \Delta t \boldsymbol{\tau}^n) \odot \mathbf{X}^{n-1} + \Delta t \boldsymbol{\tau}^n \odot \sigma\left(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})\right). \tag{8.7}$$

Rearranging the terms in (8.7), we obtain

$$\frac{\mathbf{X}^n - \mathbf{X}^{n-1}}{\Delta t} = \boldsymbol{\tau}^n \odot \left(\sigma\left(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})\right) - \mathbf{X}^{n-1}\right). \tag{8.8}$$

Interpreting $\mathbf{X}^n \approx \mathbf{X}(n\Delta t) = \mathbf{X}(t^n)$, i.e., marching in time, corresponds to increasing the number of hidden layers. Letting $\Delta t \to 0$, one obtains the following system of graph-coupled ordinary differential equations (ODEs):

$$\begin{aligned}
\frac{d\mathbf{X}(t)}{dt} &= \boldsymbol{\tau}(t) \odot \left(\sigma\left(\mathbf{F}_\theta(\mathbf{X}(t), \mathcal{G})\right) - \mathbf{X}(t)\right), \quad \forall t \geq 0, \\
\boldsymbol{\tau}_{ik}(t) &= \tanh\left(\sum_{j \in \mathcal{N}_i} |\hat{\boldsymbol{\tau}}_{ik}(t) - \hat{\boldsymbol{\tau}}_{jk}(t)|^p\right), \\
\hat{\boldsymbol{\tau}}(t) &= \hat{\sigma}(\hat{\mathbf{F}}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})).
\end{aligned} \tag{8.9}$$

We observe that the iteration formula (8.4) acts as a *forward Euler* discretization of the ODE system (8.9). Hence, one can follow Chamberlain et al. [2021b] and design more general (e.g., higher-order, adaptive, or implicit) discretizations of the ODE system (8.9). All these can be considered as design extensions of (8.4).

Oversmoothing. Using the interpretation of (8.4) as a discretization of the ODE system (8.9), we can leverage the same mathematical framework that we developed in chapter 7 to study the oversmoothing problem for $\mathbf{G}^2$.

To this end, one can prove the following proposition further characterizing oversmoothing with the standard terminology of dynamical systems [Wiggins, 2003].

Proposition 8.2.2. *The oversmoothing problem occurs for the ODEs* (8.9) *iff the hidden states* $\mathbf{X}_i^* = \mathbf{c}$*, for all* $i \in \mathcal{V}$ *are* exponentially stable steady states (fixed points) *of the ODE* (8.9)*, for some* $\mathbf{c} \in \mathbb{R}^m$*.*

In other words, for the oversmoothing problem to occur for this system, all the trajectories of the ODE (8.9) that start within the corresponding basin of attraction have to converge exponentially fast in time (according to (7.13)) to the corresponding steady state $\mathbf{c}$. Note that the basins of attraction will be different for different values of $\mathbf{c}$. The proof of Proposition 8.2.2 is a straightforward adaptation of the proof of Proposition 7.2.3.

Given this precise formulation of oversmoothing, we will investigate whether and how gradient gating in (8.9) can prevent oversmoothing. For simplicity, we set $m = 1$ to consider only scalar node features (extension to vector node features is straightforward). Moreover, we assume coupling functions of the form $\mathbf{F}(\mathbf{X}) = \mathbf{A}(\mathbf{X})\mathbf{X}$, expressed element-wise as,

$$(\mathbf{F}(\mathbf{X}))_i = \sum_{j \in \mathcal{N}_i} \mathbf{A}(\mathbf{X}_i, \mathbf{X}_j)\mathbf{X}_j. \tag{8.10}$$

Here, $\mathbf{A}(\mathbf{X})$ is a matrix-valued function whose form covers many commonly used coupling functions stemming from the graph attention (GAT, where $\mathbf{A}_{ij} = \mathbf{A}(\mathbf{X}_i, \mathbf{X}_j)$ is learnable) or convolution operators (GCN, where $\mathbf{A}_{ij}$ is fixed). Furthermore, the matrices are *right stochastic*, i.e., the entries satisfy

$$0 \leq \mathbf{A}_{ij} \leq 1, \ \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij} \quad = 1. \tag{8.11}$$

Finally, as the multi-rate feature of (8.9) has no direct bearing on the oversmoothing problem, we focus on the contribution of the gradient feedback term. To this end, we deactivate the multi-rate aspects and assume that $\hat{\boldsymbol{\tau}}_i = \mathbf{X}_i$ for all $i \in \mathcal{V}$, leading to the following form of (8.9):

$$\begin{aligned} \frac{d\mathbf{X}_i(t)}{dt} &= \boldsymbol{\tau}_i(t) \left(\sigma \left(\sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij} \mathbf{X}_j(t) \right) - \mathbf{X}_i(t) \right), \quad \forall t \geq 0, \\ \boldsymbol{\tau}_i(t) &= \tanh \left(\sum_{j \in \mathcal{N}_i} \|\mathbf{X}_j(t) - \mathbf{X}_i(t)\|^p \right). \end{aligned} \tag{8.12}$$

Lack of $\mathbf{G}^2$ can lead to oversmoothing. We first consider the case where the Gradient Gating is switched off by setting $p = 0$ in (8.12). This yields a standard GNN in which node features are evolved through message-passing between neighboring nodes, without any explicit information about graph gradients. We further assume that the activation function is ReLU i.e., $\sigma(x) = \max(x, 0)$. Given this setting, we have the following proposition on oversmoothing:

Proposition 8.2.3. *Assume the underlying graph $\mathcal{G}$ is connected. For any $c \geq 0$, let $\mathbf{X}_i^* \equiv c$, for all $i \in \mathcal{V}$ be a (zero-Dirichlet energy) steady state of the ODEs* (8.12)*. Moreover, assume no Gradient Gating ($p = 0$ in* (8.12)*) and*

$$\mathbf{A}_{ij}(c, c) = \mathbf{A}_{ji}(c, c), \text{ and } \mathbf{A}_{ij}(c, c) \geq \underline{a}, \quad 1 \leq i, j \leq v, \tag{8.13}$$

with $0 < \underline{a} \leq 1$ and that there exists at least one node denoted w.l.o.g. with index 1 such that $\mathbf{X}_1(t) \equiv c$, for all $t \geq 0$. Then, the steady state $\mathbf{X}_i^* = \mathbf{c}$, for all $i \in \mathcal{V}$, of (8.12) *is* exponentially stable.

Proposition 8.2.2 implies that without gradient gating ($\mathbf{G}^2$), (8.9) can lead to oversmoothing. In order to prove 8.2.3, we first have to establish the following technical result, i.e., a Poincare inequality on connected graphs.

Poincare inequalities for functions [Evans, 2010] bound function values in terms of their gradients. Similar bounds on node values in terms of graph-gradients can be derived and a particular instance is given below,

Proposition 8.2.4. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{D})$ be a connected graph and the corresponding (scalar) node features are denoted by $\mathbf{Y}_i \in \mathbb{R}$, for all $i \in \mathcal{V}$. Let $\mathbf{Y}_1 = 0$. Then, the following bound holds,*

$$\sum_{i \in \mathcal{V}} \mathbf{Y}_i^2 \leq \overline{d} \Delta_1 \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} |\mathbf{Y}_j - \mathbf{Y}_i|^2, \tag{8.14}$$

where $\overline{d} = \max_{i \in \mathcal{V}} \deg(i)$ and Δ_1 is the eccentricity of the node 1.

Proof. Fix a node $i \in \mathcal{V}$. By assumption, the graph $\mathcal{G}$ is connected. Hence, there exists a path connecting i and the node 1. Denote the shortest path as $\mathcal{P}(i, 1)$. This path can be expressed in terms of the nodes $\ell_{i,1}$ with $0 \leq \ell \leq \delta$, where $0_{i,1} = 1$ and $\delta_{i,1} = i$. For any ℓ, we require $\ell_{i,1} \sim (\ell + 1)_{i,1}$. Moreover, $\delta_{i,1}$ is the graph distance between the nodes i and 1 and $\Delta_1 = \max_{i \in \mathcal{V}} \delta_{i,1}$ is the eccentricity of the node 1.

Given the node feature $\mathbf{Y}_i$, we can rewrite it as,

$$\mathbf{Y}_i = \mathbf{Y}_1 + \sum_{\ell=0}^{\delta-1} \mathbf{Y}_{(\ell+1)_{i,1}} - \mathbf{Y}_{\ell_{i,1}} = \sum_{\ell=0}^{\delta-1} \mathbf{Y}_{(\ell+1)_{i,1}} - \mathbf{Y}_{\ell_{i,1}},$$

as by assumption $\mathbf{Y}_1 = 0$.

Using Cauchy-Schwartz inequality on the previous identity yields,

$$\mathbf{Y}_i^2 \leq \Delta_1 \sum_{\ell=0}^{\delta-1} \left(\mathbf{Y}_{(\ell+1)_{i,1}} - \mathbf{Y}_{\ell_{i,1}} \right)^2.$$

Summing the above inequality over $i \in \mathcal{V}$ and using the fact that $\ell_{i,1} \sim (\ell + 1)_{i,1}$, we obtain the desired Poincare inequality (8.14).

□

Leveraging this Poincare inequality on connected graphs, we proceed to prove proposition 8.2.3.

Proof. By the definition of exponential stability, we consider a small perturbation around the steady state $\mathbf{c}$ and study whether this perturbation grows or decays in time. To this end, define the perturbation as,

$$\hat{\mathbf{X}}_i = \mathbf{X}_i - c, \quad 1 \leq i \leq v. \tag{8.15}$$

A tedious but straightforward calculation shows that these perturbations evolve by the following *linearized* system of ODEs,

$$\frac{d\hat{\mathbf{X}}_i(t)}{dt} = \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right), \quad \forall t,\ \forall i \in \mathcal{V}. \tag{8.16}$$

Multiplying $\hat{\mathbf{x}}_i$ to both sides of (8.16) yields,

$$\begin{aligned}
\hat{\mathbf{X}}_i \frac{d\hat{\mathbf{X}}_i(t)}{dt} &= \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\hat{\mathbf{X}}_i\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right), \\
\Rightarrow \quad \frac{d\hat{\mathbf{X}}_i^2(t)}{dt} &= \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2\right) - \sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^2.
\end{aligned}$$

Summing the above identity over all nodes $i \in \mathcal{V}$ yields,

$$\begin{aligned}
\frac{d}{dt}\sum_{i\in\mathcal{V}} \hat{\mathbf{X}}_i^2(t) &= \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2\right) - \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^2 \\
&= \frac{1}{2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \underbrace{\left(\mathbf{A}_{ij}(c,c) - \mathbf{A}_{j,i}(c,c)\right)}_{=0\ (8.13)}\left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2\right) - \frac{1}{2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \underbrace{\left(\mathbf{A}_{ij}(c,c) + \mathbf{A}_{j,i}(c,c)\right)}_{=2\mathbf{A}_{ij}\ (8.13)}\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^2, \\
&= -\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \mathbf{A}_{ij}(c,c)\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^2, \\
&\leq -\underline{a}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right)^2, \quad \text{(by (8.13))}, \\
&\leq -\frac{\underline{a}}{\overline{d}\Delta_1}\sum_{i\in\mathcal{V}} \hat{\mathbf{X}}_i^2.
\end{aligned}$$

Here, the last inequality comes from applying the Poincare inequality (8.14) for the perturbations $\hat{\mathbf{X}}$ and from the fact that by assumption $\hat{\mathbf{X}}_1 = 0$.

Applying Grönwall's inequality yields,

$$\sum_{i\in\mathcal{V}} \hat{\mathbf{X}}_i^2(t) \leq \sum_{i\in\mathcal{V}} \hat{\mathbf{X}}_i^2(0) e^{-\frac{\underline{a}}{\overline{d}\Delta_1}t}. \tag{8.17}$$

Thus, the initial perturbations around the steady state $\mathbf{c}$ are damped down exponentially fast and the steady state $\mathbf{c}$ is exponentially stable implying that this architecture will lead to oversmoothing. □

$\mathbf{G}^2$ prevents oversmoothing. We next investigate the effect of Gradient Gating in the same setting of Proposition 8.2.3. The following Proposition shows that gradient gating prevents oversmoothing:

Proposition 8.2.5. *Assume the underlying graph $\mathcal{G}$ is connected. For any $c \geq 0$ and for all $i \in \mathcal{V}$, let $\mathbf{X}_i^* \equiv c$ be a (zero-Dirichlet energy) steady state of the ODEs* (8.12)*. Moreover, assume Gradient Gating ($p > 0$) and that the matrix $\mathbf{A}$ in* (8.12) *satisfies* (8.13) *and that there exists at least one node denoted w.l.o.g. with index* 1 *such that $\mathbf{X}_1(t) \equiv c$, for all $t \geq 0$. Then, the steady state $\mathbf{X}_i^* = \mathbf{c}$, for all $i \in \mathcal{V}$ is* not exponentially stable*.*

Proof. As in the proof of Proposition 8.2.3, we consider small perturbations of form (8.15) of the steady state $\mathbf{c}$ and investigate how these perturbations evolve in time. Assuming that the initial perturbations

are small, i.e., that there exists an $0 < \epsilon << 1$ such that $\max_{i \in \mathcal{V}} |\hat{\mathbf{x}}_i(0)| \leq \epsilon$, we perform a straightforward calculation to obtain that the perturbations (for a short time) evolve with the following *quasi-linearized* system of ODEs,

$$\begin{aligned} \frac{d\hat{\mathbf{X}}_i(t)}{dt} &= \overline{\boldsymbol{\tau}}_i(t) \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right), \ \forall i \in \mathcal{V}, \\ \overline{\boldsymbol{\tau}}_i(t) &= \sum_{j \in \mathcal{N}_i} |\hat{\mathbf{X}}_j(t) - \hat{\mathbf{X}}_i(t)|^p, \ \forall i \in \mathcal{V}. \end{aligned} \tag{8.18}$$

Note that we have used the fact that $\sigma'(x) = 1$ and $\tanh'(0) = 1$ in obtaining (8.18) from (8.12).

Next, we multiply $\hat{\mathbf{x}}_i$ to both sides of (8.18) to obtain,

$$\begin{aligned} \hat{\mathbf{X}}_i \frac{d\hat{\mathbf{X}}_i(t)}{dt} &= \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \overline{\boldsymbol{\tau}}_i \hat{\mathbf{X}}_i \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right), \\ \Rightarrow \quad \frac{d\hat{\mathbf{X}}_i^2(t)}{dt} &= \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \overline{\boldsymbol{\tau}}_i \left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2 \right) - \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \overline{\boldsymbol{\tau}}_i \left(\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right)^2 \end{aligned} \tag{8.19}$$

Trivially,

$$|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i|^p \leq \overline{\boldsymbol{\tau}}_i, \ \forall j \in \mathcal{N}_i, \quad \forall i.$$

Applying this inequality to the last line of the identity (8.19), we obtain,

$$\frac{d\hat{\mathbf{X}}_i^2(t)}{dt} \leq \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \overline{\boldsymbol{\tau}}_i \left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2 \right) - \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \left| \hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right|^{p+2}.$$

Summing the above inequality over $i \in \mathcal{V}$ leads to,

$$\begin{aligned} \frac{d}{dt} \sum_{i \in \mathcal{V}} \hat{\mathbf{X}}_i^2(t) &\leq \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \overline{\boldsymbol{\tau}}_i \left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2 \right) - \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \left| \hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right|^{p+2} \\ &\leq \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \left(\overline{\boldsymbol{\tau}}_i - \overline{\boldsymbol{\tau}}_j \right) \left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2 \right) \quad (\mathbf{A}_{ij} = \mathbf{A}_{j,i}) \\ &\quad - \underline{a} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \left| \hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right|^{p+2} \quad \text{(from (8.13))}. \end{aligned}$$

Therefore, we have the following inequality,

$$\begin{aligned} \frac{d}{dt} \sum_{i \in \mathcal{V}} \hat{\mathbf{X}}_i^2(t) &\leq T_1 - T_2, \\ T_1 &= \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbf{A}_{ij}(c,c) \left(\overline{\boldsymbol{\tau}}_i - \overline{\boldsymbol{\tau}}_j \right) \left(\hat{\mathbf{X}}_j^2 - \hat{\mathbf{X}}_i^2 \right) \\ T_2 &= \underline{a} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \left| \hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i \right|^{p+2}. \end{aligned} \tag{8.20}$$

We analyze the differential inequality (8.20) by starting with the term T_1 in (8.20). We observe that this term does not have a definite sign and can be either positive or negative. However, we can upper bound this term in the following manner. Given that the right-hand side of the ODE system (8.18) is Lipschitz continuous, the well-known Cauchy-Lipschitz theorem states that the solutions $\hat{\mathbf{x}}$ depend continuously on

the initial data. Given that $\max_{i\in\mathcal{V}}|\hat{\mathbf{X}}_i(0)| \leq \epsilon << 1$ and the bounds on the hidden states (8.1), there exists a time $\bar{t} > 0$ such that

$$\max_{i\in\mathcal{V}}|\hat{\mathbf{X}}_i(t)| \leq 1, \forall t \in [0, \bar{t}].$$

Using the definitions of $\overline{\boldsymbol{\tau}}$ and the right stochasticity of the matrix $\mathbf{A}$, we easily obtain the following bound,

$$|T_1| \leq 2^{p+1}\overline{d}^2 v, \tag{8.21}$$

where $\overline{d} = \max_{i\in\mathcal{V}} \deg(i)$.

On the other hand, the term T_2 in (8.20) is clearly positive. Hence, the solutions of resulting ODE,

$$\frac{d}{dt}\sum_{i\in\mathcal{V}}\hat{\mathbf{X}}_i^2(t) \leq -T_2, \tag{8.22}$$

will clearly decay in time. The key question is whether or not the decay is *exponentially fast*. We answer this question below.

To this end, we have the following calculation using the Hölder's inequality,

$$\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right|^2 \leq (\overline{d}v)^{\frac{p}{p+2}}\left(\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right|^{p+2}\right)^{\frac{2}{p+2}},$$

$$\Rightarrow \quad \frac{1}{(\overline{d}v)^{\frac{p}{2}}}\left(\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right|^2\right)^{\frac{p+2}{2}} \leq \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right|^{p+2}.$$

Observing that $\hat{\mathbf{X}}_1 = 0$ by assumption, we can applying the Poincare inequality (8.14) in the above inequality to further obtain,

$$\frac{1}{\overline{d}^{p+1}v^{\frac{p}{2}}\Delta_1^{\frac{p+2}{2}}}\left(\sum_{i\in\mathcal{V}}|\hat{\mathbf{X}}_i|^2\right)^{\frac{p+2}{2}} \leq \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left|\hat{\mathbf{X}}_j - \hat{\mathbf{X}}_i\right|^{p+2}.$$

Hence, from the definition of T_2 (8.20), we have,

$$T_2 \geq \frac{\underline{a}}{\overline{d}^{p+1}v^{\frac{p}{2}}\Delta_1^{\frac{p+2}{2}}}\left(\sum_{i\in\mathcal{V}}|\hat{\mathbf{X}}_i|^2\right)^{\frac{p+2}{2}}. \tag{8.23}$$

Therefore, the differential inequality (8.22) now reduces to,

$$\frac{d}{dt}\sum_{i\in\mathcal{V}}\hat{\mathbf{X}}_i^2(t) \leq -\frac{\underline{a}}{\overline{d}^{p+1}v^{\frac{p}{2}}\Delta_1^{\frac{p+2}{2}}}\left(\sum_{i\in\mathcal{V}}|\hat{\mathbf{X}}_i|^2\right)^{\frac{p+2}{2}}. \tag{8.24}$$

The differential inequality (8.24) can be explicitly solved to obtain,

$$\sum_{i\in\mathcal{V}}\hat{\mathbf{X}}_i^2(t) \leq \left(2 + pt\frac{\underline{a}}{\overline{d}^{p+1}v^{\frac{p}{2}}\Delta_1^{\frac{p+2}{2}}}\sum_{i\in\mathcal{V}}\hat{\mathbf{X}}_i^2(0)^{\frac{p}{2}}\right)^{-\frac{2}{p}}\sum_{i\in\mathcal{V}}\hat{\mathbf{X}}_i^2(0). \tag{8.25}$$

From (8.25), we see that the initial perturbations decay but only *algebraically* at a rate of $t^{-\frac{2}{p}}$ in time. For instance, the decay is only linear in time for $p = 2$ and even slower for higher value of p.

Combining the analysis of the terms $T_{1,2}$ in the differential inequality (8.20), we see that the one of the terms can lead to a growth in the initial perturbations whereas the second term only leads to polynomial decay. Even if the contribution of the term $T_1 \equiv 0$, the decay of initial perturbations is only polynomial. Thus, the steady state $\mathbf{c}$ is not exponentially stable. □

This proof clearly elucidates the role of gradient gating by showing that the energy associated with the quasi-linearized evolution equations (8.18) is balanced by two terms (8.20)), both resulting from the introduction of gradient gating by setting $p > 0$ in (8.12). One of them is of indefinite sign and can even cause *growth* of perturbations around a steady state $\mathbf{c}$. The other decays initial perturbations. However, the rate of this decay is at most *polynomial* (Eqn. (8.25)). For instance, the decay is merely linear for $p = 2$ and slower for higher values of p. Thus, the steady state $\mathbf{c}$ cannot be exponentially stable and oversmoothing is prevented. This justifies the intuition behind gradient gating, namely, if oversmoothing occurs around a node i, i.e., $\lim_{n\to\infty} \sum_{j\in\mathcal{N}_i} \|\mathbf{X}_i^n - \mathbf{X}_j^n\| = 0$, then the corresponding rate $\boldsymbol{\tau}_i^n$ goes to zero (at a faster rate), such that the underlying hidden node feature $\mathbf{X}_i$ stops getting updated.

Remark 8.2.6. *We note that the Proposition 8.2.5 assumes a certain structure of the matrix* $\mathbf{A}$ *in* (8.12)*. A careful perusal of the proof presented above reveals that these assumptions can be further relaxed. To start with, if the matrix* $\mathbf{A}(c, c)$ *is not symmetric, then there will be an additional term in the inequality* (8.20)*, which would be proportional to* $\mathbf{A}_{ij} - \mathbf{A}_{ji}$*. This term will be of indefinite sign and can cause further growth in the perturbations of the steady state c. In any case, it can only further destabilize the quasi-linearized system. The assumption that the entries of* $\mathbf{A}$ *are uniformly positive amounts to assuming positivity of the weights of the underlying GNN layer. This can be replaced by requiring that the corresponding eigenvalues are uniformly positive. If some eigenvalues are negative, this will cause further instability and only strengthen the conclusion of lack of (exponential) stability. Finally, the assumption that one node is not perturbed during the quasi-linearization is required for the Poincare inequality* (8.14)*. If this is not true, an additional term, of indefinite sign, is added to the inequality* (8.20)*. This term can cause further growth of the perturbations and will only add instability to the system. Hence, all the assumptions in Proposition 8.2.5 can be relaxed and the conclusion of lack of exponential stability of the zero-Dirichlet energy steady state still holds.*

8.3 Empirical results

In this section, we present an experimental study of $\mathbf{G}^2$ on both synthetic and real datasets. We use $\mathbf{G}^2$ with three different coupling functions: GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018] and GraphSAGE [Hamilton et al., 2017].

Effect of $\mathbf{G}^2$ on Dirichlet energy. Given that oversmoothing relates to the decay of Dirichlet energy (6.2), we follow the experimental setup proposed in Chapter 7 to probe the dynamics of the Dirichlet energy of Gradient-Gated GNNs, defined on a 2-dimensional 10×10 regular grid with 4-neighbor connectivity. The node features $\mathbf{X}$ are randomly sampled from $\mathcal{U}([0, 1])$ and then propagated through a 1000-layer GNN with random weights. We compare GAT, GCN and their gradient-gated versions ($\mathbf{G}^2$-GAT and $\mathbf{G}^2$-GCN) in this experiment. Fig. 8.2 depicts on log-log scale the Dirichlet energy of each layer's output with respect to the layer number. We clearly observe that GAT and GCN *oversmooth* as the underlying Dirichlet energy converges exponentially fast to zero, resulting in the node features becoming indistinguishable. In practice, the Dirichlet energy for these architectures is ≈ 0 after just ten hidden layers. On the other

hand, and as suggested by the theoretical results of the previous section, adding $\mathbf{G}^2$ decisively prevents this behavior and the Dirichlet energy remains (near) constant, even for very deep architectures (up to 1000 layers).

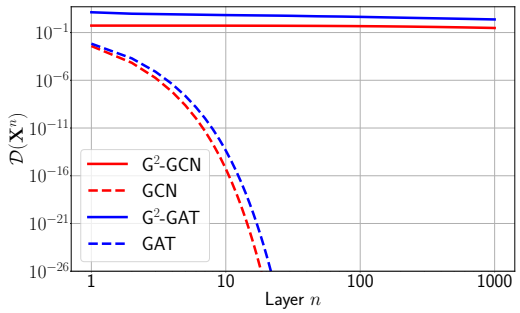

Figure 8.2: Dirichlet energy $\mathcal{D}(\mathbf{X}^n)$ of layer-wise node features $\mathbf{X}^n$ propagated through a GAT, GCN and their gradient gated versions ($\mathbf{G}^2$-GAT, $\mathbf{G}^2$-GCN).

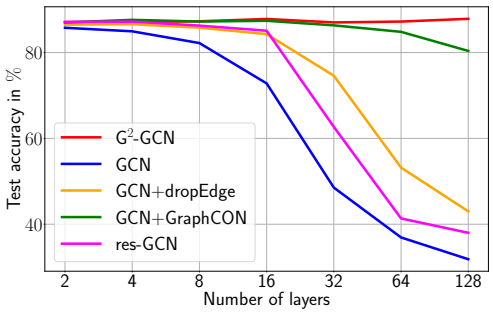

Figure 8.3: Test accuracies of GCN with gradient gating ($\mathbf{G}^2$-GCN) as well as plain GCN and GCN combined with other methods on the Cora dataset for increasing number of layers.

$\mathbf{G}^2$ for very deep GNNs. Oversmoothing inhibits the use of large number of GNN layers. As $\mathbf{G}^2$ is designed to alleviate oversmoothing, it should allow very deep architectures. To test this assumption, we reproduce the experiment considered in Chamberlain et al. [2021b]: a node-level classification task on the Cora dataset using increasingly deeper GCN architectures. In addition to $\mathbf{G}^2$, we also compare with two recently proposed mechanisms to alleviate oversmoothing, DropEdge [Rong et al., 2020] and GraphCON (Chapter 7). The results are presented in Fig. 8.3, where we plot the test accuracy for all the models with the number of layers ranging from 2 to 128. While a plain GCN seems to suffer the most from oversmoothing (with the performance rapidly deteriorating after 8 layers), GCN+DropEdge as well as GCN+GraphCON are able to mitigate this behavior to some extent, although the performance eventually starts dropping (after 16 and 64 layers, respectively). In contrast, $\mathbf{G}^2$-GCN exhibits a small but noticeable *increase* in performance for increasing number of layers, reaching its peak performance for 128 layers. This experiment suggests that $\mathbf{G}^2$ can indeed be used in conjunction with deep GNNs, potentially allowing performance gains due to depth.

$\mathbf{G}^2$ for multiscale node-level regression. We test the multi-rate nature of $\mathbf{G}^2$ on node-level regression tasks, where the target node values exhibit multiple scales. Due to a lack of widely available node-level regression tasks, we propose regression experiments based on the Wikipedia article networks Chameleon and Squirrel, [Rozemberczki et al., 2021]. While Chameleon and Squirrel are already widely used as heterophilic node-level classification tasks, the original datasets consist of continuous node targets (average monthly web-page traffic). We normalize the provided webpage traffic values for every node between 0 and 1 and note that the resulting node values exhibit values on a wide range of different scales ranging between 10^{-5} and 1 (see Fig. 8.4). Table 8.1 shows the test normalized mean-square error (mean and standard deviation based on the ten pre-defined splits in Pei et al. [2020]) for two standard GNN architectures (GCN and GAT) with and without $\mathbf{G}^2$. We observe from Table 8.1 that adding $\mathbf{G}^2$ to the baselines significantly reduces the error, demonstrating the advantage of using multiple update rates.

Table 8.1: Normalized test MSE on multiscale node-level regression tasks. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

	Chameleon	**Squirrel**
#Nodes	2,277	5,201
#Edges	31,421	198,493
GCNII	**0.170 ± 0.034**	**0.093 ± 0.031**
PairNorm	0.207 ± 0.038	0.140 ± 0.040
GCN	0.207 ± 0.039	0.143 ± 0.039
GAT	0.207 ± 0.038	0.143 ± 0.039
$\mathbf{G}^2$-GCN	**0.137 ± 0.033**	**0.070 ± 0.028**
$\mathbf{G}^2$-GAT	**0.136 ± 0.029**	**0.069 ± 0.029**

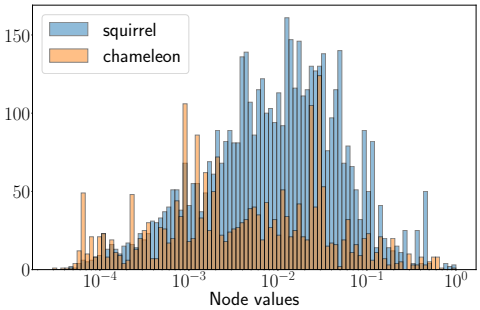

Figure 8.4: Histogram of the target node values of the Chameleon and Squirrel node-level regression tasks.

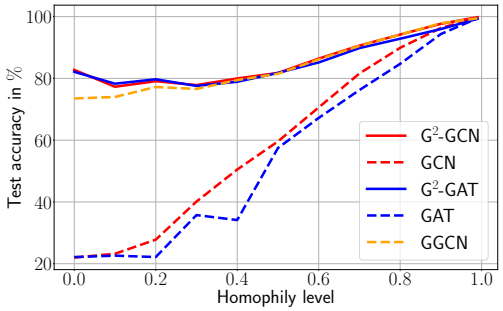

Figure 8.5: Test accuracy of GCN and GAT with / without gradient gating ($\mathbf{G}^2$) on synthetic Cora with a varying level of true label homophily.

$\mathbf{G}^2$ for varying homophily (Synthetic Cora). We test $\mathbf{G}^2$ on a node-level classification task with varying levels of homophily on the synthetic Cora dataset Zhu et al. [2020]. Standard GNN models are known to perform poorly in heterophilic settings. This can be seen in Fig. 8.5, where we present the classification accuracy of GCN and GAT on the synthetic-Cora dataset with a level of homophily varying between 0 and 0.99. While these models succeed in the homophilic case (reaching nearly perfect accuracy), their performance drops to $\approx 20\%$ when the level of homophily approaches 0. Adding $\mathbf{G}^2$ to GCN or GAT mitigates this phenomenon: the resulting models reach a test accuracy of over 80%, even in the most heterophilic setting, thus leading to a four-fold increase in the accuracy of the underlying GCN or GAT models. Furthermore, we notice an increase in performance even in the homophilic setting. Moreover, we compare with a state-of-the-art model GGCN [Yan et al., 2021], which has been recently proposed to explicitly deal with heterophilic graphs. From Fig. 8.5 we observe that $\mathbf{G}^2$ performs on par and slightly better than GGCN in strongly heterophilic settings.

Heterophilic datasets. In Table 8.2, we test the proposed framework on several real-world heterophilic graphs (with a homophily level of ≤ 0.30) [Pei et al., 2020, Rozemberczki et al., 2021] and benchmark it against baseline models GraphSAGE [Hamilton et al., 2017], GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018] and MLP [Goodfellow et al., 2016], as well as recent state-of-the-art models on heterophilic graph datasets, i.e., GGCN [Yan et al., 2021], GPRGNN [Chien et al., 2020], H2GCN [Zhu et al., 2020], FAGCN [Bo et al., 2021], F^2GAT [Wei et al., 2022], MixHop [Abu-El-Haija et al., 2019],

Table 8.2: Results on heterophilic graphs. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**
Hom level	**0.11**	**0.21**	**0.22**	**0.22**	**0.23**	**0.30**
#Nodes	183	251	7,600	5,201	2,277	183
#Edges	295	466	26,752	198,493	31,421	280
#Classes	5	5	5	5	5	5
GGCN	**84.86 ± 4.55**	86.86 ± 3.29	**37.54 ± 1.56**	55.17 ± 1.58	**71.14 ± 1.84**	85.68 ± 6.63
GPRGNN	78.38 ± 4.36	82.94 ± 4.21	34.63 ± 1.22	31.61 ± 1.24	46.58 ± 1.71	80.27 ± 8.11
H2GCN	**84.86 ± 7.23**	**87.65 ± 4.98**	35.70 ± 1.00	36.48 ± 1.86	60.11 ± 2.15	82.70 ± 5.28
FAGCN	82.43 ± 6.89	82.94 ± 7.95	34.87 ± 1.25	42.59 ± 0.79	55.22 ± 3.19	79.19 ± 9.79
F^2GAT	82.70 ± 5.95	**87.06 ± 4.13**	36.65 ± 1.13	47.32 ± 2.43	67.81 ± 2.05	83.51 ± 6.70
MixHop	77.84 ± 7.73	75.88 ± 4.90	32.22 ± 2.34	43.80 ± 1.48	60.50 ± 2.53	73.51 ± 6.34
GCNII	77.57 ± 3.83	80.39 ± 3.40	**37.44 ± 1.30**	38.47 ± 1.58	63.86 ± 3.04	77.86 ± 3.79
Geom-GCN	66.76 ± 2.72	64.51 ± 3.66	31.59 ± 1.15	38.15 ± 0.92	60.00 ± 2.81	60.54 ± 3.67
PairNorm	60.27 ± 4.34	48.43 ± 6.14	27.40 ± 1.24	50.44 ± 2.04	62.74 ± 2.82	58.92 ± 3.15
LINKX	74.60 ± 8.37	75.49 ± 5.72	36.10 ± 1.55	**61.81 ± 1.80**	68.42 ± 1.38	77.84 ± 5.81
GloGNN	84.32 ± 4.15	**87.06 ± 3.53**	**37.35 ± 1.30**	**57.54 ± 1.39**	**69.78 ± 2.42**	83.51 ± 4.26
GraphSAGE	82.43 ± 6.14	81.18 ± 5.56	34.23 ± 0.99	41.61 ± 0.74	58.73 ± 1.68	75.95 ± 5.01
ResGatedGCN	80.00 ± 5.57	81.57 ± 5.35	36.02 ± 1.19	37.60 ± 1.80	49.82 ± 2.71	73.51 ± 4.95
GCN	55.14 ± 5.16	51.76 ± 3.06	27.32 ± 1.10	31.52 ± 0.71	38.44 ± 1.92	60.54 ± 5.30
GAT	52.16 ± 6.63	49.41 ± 4.09	27.44 ± 0.89	36.77 ± 1.68	48.36 ± 1.58	61.89 ± 5.05
MLP	80.81 ± 4.75	85.29 ± 3.31	36.53 ± 0.70	28.77 ± 1.56	46.21 ± 2.99	81.89 ± 6.40
$\mathbf{G}^2$-GAT	**84.59 ± 5.55**	**87.65 ± 4.64**	37.30 ± 0.87	46.48 ± 1.41	64.12 ± 1.96	**87.30 ± 4.84**
$\mathbf{G}^2$-GCN	**84.86 ± 3.24**	**87.06 ± 3.19**	37.09 ± 1.16	39.62 ± 2.91	55.83 ± 2.88	**86.49 ± 5.27**
$\mathbf{G}^2$-GraphSAGE	**87.57 ± 3.86**	**87.84 ± 3.49**	37.14 ± 1.01	**64.26 ± 2.38**	**71.40 ± 2.38**	**86.22 ± 4.90**

GCNII [Chen et al., 2020b], Geom-GCN [Pei et al., 2020], PairNorm [Zhao and Akoglu, 2019]. We can observe that $\mathbf{G}^2$ added to GCN, GAT or GraphSAGE outperforms all other methods (in particular recent methods such as GGCN, GPRGNN, H2GCN that were explicitly designed to solve heterophilic tasks). Moreover, adding $\mathbf{G}^2$ to the underlying base GNN model improves the results on average by 45.75% for GAT, 45.4% for GCN and 18.6% for GraphSAGE.

Large-scale graphs. Given the exceptional performance of $\mathbf{G}^2$-GraphSAGE on small and medium sized heterophilic graphs, we test the proposed $\mathbf{G}^2$ (applied to GraphSAGE, i.e., $\mathbf{G}^2$-GraphSAGE) on large-scale datasets. To this end, we consider three different experiments based on large graphs from Lim et al. [2021a], which range from highly heterophilic (homophily level of 0.07) to fairly homophilic (homophily level of 0.61). The sizes range from large graphs with ∼170K nodes and ∼1M edges to a very large graph with ∼3M nodes and ∼14M edges.

Table 8.3 shows the results of $\mathbf{G}^2$-GraphSAGE together with other standard GNNs, as well as recent state-of-the-art models, i.e., MLP[Goodfellow et al., 2016], GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018], MixHop [Abu-El-Haija et al., 2019], LINK(X) [Lim et al., 2021a], GCNII [Chen et al., 2020b], APPNP [Klicpera et al., 2018], GloGNN [Li et al., 2022], GPR-GNN [Chien et al., 2020] and ACM-GCN [Luan et al., 2021]. We can see that $\mathbf{G}^2$-GraphSAGE significantly outperforms current state-of-the-art (by up to 13%) on the two heterophilic graphs (i.e., snap-patents and arXiv-year). Moreover, $\mathbf{G}^2$-GraphSAGE is on-par with the current state-of-the-art on the homophilic graph dataset genius.

We conclude that the proposed gradient gating method can successfully be scaled up to large graphs,

reaching state-of-the-art performance, in particular on heterophilic graph datasets.

Table 8.3: Results on large-scale datasets. The three best performing methods are highlighted in **red** (First), **blue** (Second), and **violet** (Third).

	snap-patents	**arXiv-year**	**genius**
Hom level	**0.07**	**0.22**	**0.61**
#Nodes	2,923,922	169,343	421,961
#Edges	13,975,788	1,166,243	984,979
#Classes	5	5	2
MLP	31.34 ± 0.05	36.70 ± 0.21	86.68 ± 0.09
GCN	45.65 ± 0.04	46.02 ± 0.26	87.42 ± 0.37
GAT	45.37 ± 0.44	46.05 ± 0.51	55.80 ± 0.87
MixHop	52.16 ± 0.09	51.81 ± 0.17	90.58 ± 0.16
LINKX	**61.95 ± 0.12**	**56.00 ± 1.34**	**90.77 ± 0.27**
LINK	60.39 ± 0.07	53.97 ± 0.18	73.56 ± 0.14
GCNII	37.88 ± 0.69	47.21 ± 0.28	90.24 ± 0.09
APPNP	32.19 ± 0.07	38.15 ± 0.26	85.36 ± 0.62
GloGNN	**62.09 ± 0.27**	**54.68 ± 0.34**	**90.66 ± 0.11**
GPR-GNN	40.19 ± 0.03	45.07 ± 0.21	90.05 ± 0.31
ACM-GCN	55.14 ± 0.16	47.37 ± 0.59	80.33 ± 3.91
$\mathbf{G}^2$-GraphSAGE	**69.50 ± 0.39**	**63.30 ± 1.84**	**90.85 ± 0.64**

Training details. All small and medium-scale experiments have been run on NVIDIA GeForce RTX 2080 Ti, GeForce RTX 3090, TITAN RTX and Quadro RTX 6000 GPUs. The large-scale experiments have been run on Nvidia Tesla A100 (40GiB) GPUs.

All hyperparameters were tuned using random search. Table 8.4 shows the ranges of each hyperparameter as well as the random distribution used to randomly sample from it. Moreover, Table 8.5 shows the rounded hyperparameter p in $\mathbf{G}^2$ (8.4) of each best performing network.

Table 8.4: Hyperparameter ranges.

	range	**rand. distribution**
learning rate	$[10^{-4}, 10^{-2}]$	log uniform
hidden size m	$\{32, 64, 128, 256, 512\}$	disc. uniform
dropout input	$[0, 0.9]$	uniform
dropout output	$[0, 0.9]$	uniform
weight decay	$[10^{-8}, 10^{-2}]$	log uniform
$\mathbf{G}^2$-exponent p	$[1, 5]$	uniform
Usage of $\hat{\mathbf{F}}_{\hat{\theta}}$ in (8.4)	{YES, NO}	disc. uniform

8.4 Further empirical analysis

On the multi-rate effect of $\mathbf{G}^2$. Here, we analyze the performance of $\mathbf{G}^2$ on the multiscale node-level regression task of Section 8.3. As we see in Section 8.3, $\mathbf{G}^2$ applied to GCN or GAT outperforms their plain counterparts (GCN and GAT) on the multiscale node-level regression task by more than

Table 8.5: Rounded hyperparameter p in $\mathbf{G}^2$ of each best performing network.

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**snap-patents**	**arXiv-year**	**genius**
$\mathbf{G}^2$-GAT	3.06	1.68	1.23	3.48	3.54	3.54	/	/	/
$\mathbf{G}^2$-GCN	3.93	2.92	3.79	1.99	1.08	3.87	/	/	/
$\mathbf{G}^2$-GraphSAGE	4.47	1.14	2.89	3.04	2.00	3.27	1.60	3.40	4.40

50% on Chameleon and more than 100% on Squirrel. The question therefore arises whether this better performance can be explained by the multi-rate nature of gradient gating.

To empirically analyse this, we begin by adding a control parameter α to $\mathbf{G}^2$ (8.4) as follows,

$$\mathbf{X}^n = (1 - (\boldsymbol{\tau}^n)^\alpha) \odot \mathbf{X}^{n-1} + (\boldsymbol{\tau}^n)^\alpha \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})).$$

Clearly, setting $\alpha = 1$ recovers the original gradient gating message-passing update,

$$\mathbf{X}^n = (1 - \boldsymbol{\tau}^n) \odot \mathbf{X}^{n-1} + \boldsymbol{\tau}^n \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})),$$

while setting $\alpha = 0$ disables any explicit multi-rate behavior and a plain message-passing scheme is recovered,

$$\mathbf{X}^n = \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})).$$

Note that by continuously changing α from 0 to 1 controls the level of multi-rate behavior in the proposed gradient gating method.

In Fig. 8.6 we plot the test NMSE of the best performing $\mathbf{G}^2$-GCN and $\mathbf{G}^2$-GAT on the Chameleon multiscale node-level regression task for increasing values of $\alpha \in [10^{-3}, 1]$ in log-scale. We can see that the test NMSE monotonically decreases (lower error means better performance) for both $\mathbf{G}^2$-GCN and $\mathbf{G}^2$-GAT for increasing values of α, i.e., increasing level of multi-rate behavior. We can conclude that the multi-rate behavior of $\mathbf{G}^2$ is instrumental in successfully learning multiscale regression tasks.

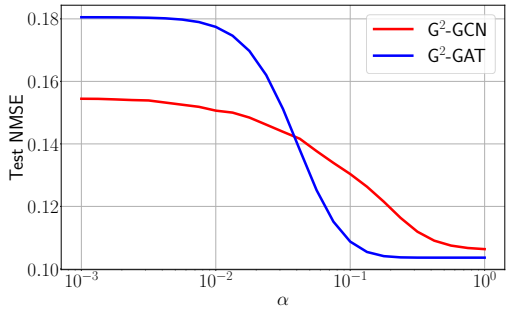

Figure 8.6: Test NMSE on the multiscale chameleon node-level regression task of $\mathbf{G}^2$-GCN and $\mathbf{G}^2$-GAT for continuously decreasing level of multi-rate behavior.

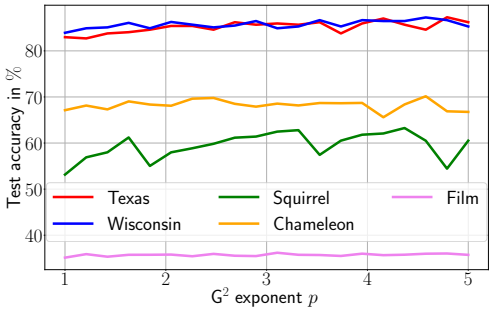

Figure 8.7: Test accuracies of $\mathbf{G}^2$-GraphSAGE on Texas, Squirrel, Film, Wisconsin and Chameleon graph datasets for varying values of p in (8.4).

On the sensitivity of performance of $\mathbf{G}^2$ to the hyperparameter p. The proposed gradient gating model implicitly depends on the hyperparameter p, which defines the multiple rates $\boldsymbol{\tau}$, i.e.,

$$\hat{\boldsymbol{\tau}}^n = \hat{\sigma}(\hat{\mathbf{F}}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})),$$
$$\boldsymbol{\tau}_{ik}^n = \tanh\left(\sum_{j \in \mathcal{N}_i} |\hat{\tau}_{ik}^n - \hat{\tau}_{jk}^n|^p\right).$$

While any value $p > 0$ can be used in practice, a standard hyperparameter tuning procedure on p has been applied in every experiment included in this chapter. Thus, it is natural to ask how sensitive the performance of $\mathbf{G}^2$ is with respect to different values of the hyperparameter p.

To answer this question, we trained different $\mathbf{G}^2$-GraphSAGE models on a variety of different graph datasets (i.e., Texas, Squirrel, Film, Wisconsin and Chameleon) for different values of $p \in [1, 5]$. Fig. 8.7 shows the resulting performance of $\mathbf{G}^2$-GraphSAGE. We can see that different values of p do not significantly change the performance of the model. However, including the hyperparameter p to the hyperparameter fine-tuning procedure will further improve the overall performance of $\mathbf{G}^2$.

On the sensitivity of performance of $\mathbf{G}^2$ to the number of parameters. All results of $\mathbf{G}^2$ provided in this chapter are obtained using standard hyperparameter tuning (i.e., random search). Those hyperparameters include the number of hidden channels for each hidden node of the graph, which directly correlates with the total number of parameters used in $\mathbf{G}^2$. It is thus natural to ask how $\mathbf{G}^2$ performs compared to its plain counter-version (e.g. $\mathbf{G}^2$-GCN to GCN) for the exact same number of total parameters of the underlying model. To this end, Fig. 8.8 shows the test accuracies of $\mathbf{G}^2$-GCN and GCN for increasing number of total parameters in its corresponding model. We can see that first, using more parameters has only a slight effect on the overall performance of both models. Second, and most importantly, $\mathbf{G}^2$-GCN constantly reaches significantly higher test accuracies for the exact same number of total parameters. We can thus rule out that the better performance of $\mathbf{G}^2$ compared to its plain counter-versions is explained by the usage of more parameters.

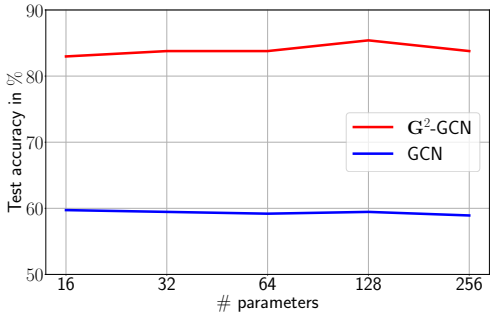

Figure 8.8: Test accuracies of plain GCN and $\mathbf{G}^2$-GCN on Texas for varying number of total parameters in the GNN.

Ablation of $\hat{\mathbf{F}}_{\hat{\theta}}$ in $\mathbf{G}^2$. In its most general form $\mathbf{G}^2$ (8.4) uses an additional GNN $\hat{\mathbf{F}}_{\hat{\theta}}$ to construct the multiple rates $\boldsymbol{\tau}^n$. Is this additional GNN needed ? To answer this question, Table 8.6 shows in which of the provided experiments (using $\mathbf{G}^2$-GraphSAGE) we actually used an additional GNN $\hat{\mathbf{F}}_{\hat{\theta}}$ (as

suggested by our hyperparameter tuning protocol). We can see that on small-scale experiments having an additional GNN is not needed. However, on the considered medium and large-scale graph datasets it is beneficial to use it. Motivated by this, Table 8.7 shows the results for $\mathbf{G}^2$-GraphSAGE on the three medium-scale graph datasets (Film, Squirrel and Chameleon) without using an additional GNN in (8.4) (i.e., $\mathbf{F}_\theta = \hat{\mathbf{F}}_{\hat{\theta}}$) as well as with using an additional GNN. We can see that while $\mathbf{G}^2$-GraphSAGE without an additional GNN (i.e., w/o $\hat{\mathbf{F}}_{\hat{\theta}}$) yields competitive results, using an additional GNN is needed in order to obtain state-of-the-art results on these three datasets.

Table 8.6: Usage of $\hat{\mathbf{F}}_{\hat{\theta}}$ in $\mathbf{G}^2$ (8.4) for each result with $\mathbf{G}^2$-GraphSAGE (YES indicates the usage of $\hat{\mathbf{F}}_{\hat{\theta}}$, while NO indicates that no additional GNN is used to construct the multiple rates, i.e., $\mathbf{F}_\theta = \hat{\mathbf{F}}_{\hat{\theta}}$)

Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**snap-patents**	**arXiv-year**	**genius**
NO	NO	YES	YES	YES	NO	YES	YES	YES

Table 8.7: Test accuracies of $\mathbf{G}^2$-GraphSAGE with and without additional GNN (i.e., w/ $\hat{\mathbf{F}}_{\hat{\theta}}$ and w/o $\hat{\mathbf{F}}_{\hat{\theta}}$ in (8.4)) on Film, Squirrel and Chameleon graph dataset.

	Film	**Squirrel**	**Chameleon**
$\mathbf{G}^2$-GraphSAGE w/ $\hat{\mathbf{F}}_{\hat{\theta}}$	37.14 ± 1.01	64.26 ± 2.38	71.40 ± 2.38
$\mathbf{G}^2$-GraphSAGE w/o $\hat{\mathbf{F}}_{\hat{\theta}}$	36.83 ± 1.26	55.78 ± 1.61	65.04 ± 2.27

Ablation of multi-rate channels in $\mathbf{G}^2$. The corner stone of the proposed $\mathbf{G}^2$ is the multi-rate matrix $\boldsymbol{\tau}^n$ in (8.4), which automatically solves the oversmoothing issue for any given GNN (Proposition 8.2.3). This multi-rate matrix learns different rates for every node but also for every channel of every node. It is thus natural to ask if the multi-rate property for the channels is necessary, or if having multiple rates for the different nodes is sufficient, i.e., having a **multi-rate vector** $\boldsymbol{\tau}^n \in \mathbb{R}^v$. A direct construction of such multi-rate vector (derived from our proposed $\mathbf{G}^2$) is:

$$\begin{aligned} \hat{\boldsymbol{\tau}}^n &= \sigma(\hat{\mathbf{F}}_{\hat{\theta}}(\mathbf{X}^{n-1}, \mathcal{G})), \\ \boldsymbol{\tau}_i^n &= \tanh\left(\sum_{j \in \mathcal{N}_i} \|\hat{\boldsymbol{\tau}}_j^n - \hat{\boldsymbol{\tau}}_i^n\|_p\right), \\ \mathbf{X}^n &= (1 - \boldsymbol{\tau}^n) \odot \mathbf{X}^{n-1} + \boldsymbol{\tau}^n \odot \sigma(\mathbf{F}_\theta(\mathbf{X}^{n-1}, \mathcal{G})). \end{aligned} \tag{8.26}$$

Note that the only difference to our proposed $\mathbf{G}^2$ is in the second equation of (8.26), where we sum over the node-wise p-norms of the differences of adjacent nodes. This way, we compute a single scalar $\boldsymbol{\tau}_i^n$ for every node $i \in \mathcal{V}$.

Table 8.8 shows the results of our proposed $\mathbf{G}^2$-GraphSAGE as well as the single-rate channels ablation of $\mathbf{G}^2$ (eq. (8.26)) on the Film, Squirrel and Chameleon graph datasets. As a baseline, we also include the results of a plain GraphSAGE. We can see that while $\mathbf{G}^2$ with single-scale channels outperforms the base GraphSAGE model, our proposed $\mathbf{G}^2$ with multi-rate channels vastly outperforms the single-rate channels version of $\mathbf{G}^2$.

Table 8.8: Test accuracies of plain GraphSAGE, $\mathbf{G}^2$-GraphSAGE with multi-rate channels for each node (i.e., standard $\mathbf{G}^2$ (8.4)) as well as with only a single rate for every channel on Film, Squirrel and Chameleon.

	Film	**Squirrel**	**Chameleon**
GraphSAGE	34.23 ± 0.99	41.61 ± 0.74	58.73 ± 1.68
$\mathbf{G}^2$-GraphSAGE w/ multi-rate channels $\mathbf{G}^2$ (8.4)	37.14 ± 1.01	64.26 ± 2.38	71.40 ± 2.38
$\mathbf{G}^2$-GraphSAGE w/ single-rate channels $\mathbf{G}^2$ (8.26)	36.67 ± 0.56	44.03 ± 1.01	60.29 ± 3.45

Alternative measures of oversmoothing. The proof of Proposition 8.2.2 and Proposition 8.2.3 as well as the first experiment in Section 8.3 is based on the definition of oversmoothing using the Dirichlet energy. However, there exist alternative measures to describe the oversmoothing phenomenon in deep GNNs. One such measure is the mean average distance (MAD), which was proposed in Chen et al. [2020a]. In order to check if our proposed $\mathbf{G}^2$ mitigates oversmoothing defined through the MAD measure we repeat the first experiment in Section 8.3 and plot the MAD instead of the Dirichlet energy for increasing number of layers in Fig. 8.9. We can see that while the MAD of a plain GCN and GAT converges exponentially with increasing number of layers, it remains constant for $\mathbf{G}^2$-GCN and $\mathbf{G}^2$-GAT. We can thus conclude that $\mathbf{G}^2$ mitigates oversmoothing defined through the MAD measure.

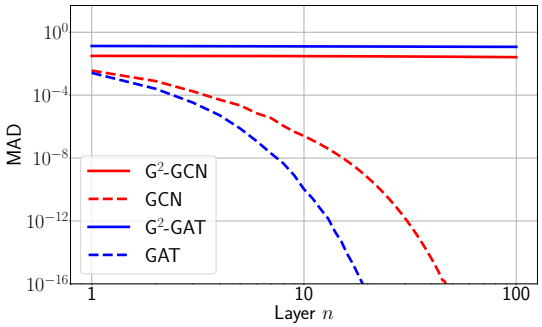

Figure 8.9: Mean average distance (MAD) of layer-wise node features $\mathbf{X}^n$ propagated through a GAT, GCN and their gradient gated versions ($\mathbf{G}^2$-GAT, $\mathbf{G}^2$-GCN).

8.5 Discussion

We have proposed a novel framework, termed $\mathbf{G}^2$, for efficient learning on graphs. $\mathbf{G}^2$ builds on standard MPNNs, but seeks to overcome their limitations. In particular, we focus on the fact that for standard MPNNs such as GCN or GAT, each node (in every hidden layer) is updated at the same *rate*. This might inhibit efficient learning of tasks where different node features would need to be updated at different rates. Hence, we equip a standard MPNN with *gates* that amount to a multi-rate modulation for the hidden layer output in (8.4). This enables multiple rates (or scales) of flow of information across a graph. Moreover, we leverage local (graph) gradients to further constrain the gates. This is done to alleviate

oversmoothing where node features become indistinguishable as the number of layers is increased.

By combining these ingredients, we present a very flexible framework (dubbed $\mathbf{G}^2$) for graph machine learning wherein *any* existing MPNN hidden layer can be employed as the coupling function and the multi-rate gradient gating mechanism can be built on top of it. Moreover, we also show that $\mathbf{G}^2$ corresponds to a time-discretization of a system of ODEs (8.9). By studying the (in)-stability of the corresponding zero-Dirichlet energy steady states we rigorously prove that gradient gating can mitigate the oversmoothing problem, paving the way for the use of very deep GNNs within the $\mathbf{G}^2$ framework. In contrast, the lack of gradient gating is shown to lead to oversmoothing.

We also present an extensive empirical evaluation to illustrate different aspects of the proposed $\mathbf{G}^2$ framework. Starting with synthetic, small-scale experiments, we demonstrate that i) $\mathbf{G}^2$ can prevent oversmoothing by keeping the Dirichlet energy constant, even for a very large number of hidden layers, ii) this feature allows us to deploy very deep architectures and to observe that the accuracy of classification tasks can *increase* with increasing number of hidden layers, iii) the multi-rate mechanism significantly improves performance on node regression tasks when the node features are distributed over a range of scales, and iv) $\mathbf{G}^2$ is very accurate at classification on *heterophilic* datasets, witnessing an increasing gain in performance with increasing heterophily.

This last feature was more extensively investigated, and we observed that $\mathbf{G}^2$ can significantly outperform baselines as well as recently proposed methods on both benchmark medium-scale and large-scale heterophilic datasets, achieving state-of-the-art performance. Thus, by a combination of theory and experiments, we demonstrate that the $\mathbf{G}^2$-framework is a promising approach for learning on graphs.

Chapter 9

Conclusion

Physics-inspired machine learning can be seen as incorporating structure from physical systems (e.g., given by ordinary or partial differential equations) into machine learning methods to obtain models with better inductive biases. Examples of such systems include oscillatory systems, Hamiltonian systems, multiscale systems, gradient flows, reaction-diffusion systems, and Langevin dynamics, to name just a few. The main goals of physics-inspired machine learning are: **(i)** address important and central issues in machine learning from a physics perspective, **(ii)** improve the performance of machine learning models applied to problems in the physical sciences by incorporating structure reasoned from the problem at hand (i.e., inductive bias), and **(iii)** provide better theoretical understanding of the proposed machine learning models by leveraging insights from the underlying physical principles. In this thesis, we have proposed and theoretically analysed novel physics-inspired machine learning models in the fields of sequence modeling and graph representation learning.

The first part of this thesis introduces recurrent sequence models derived from specific physical systems, namely that of nonlinear oscillators. Motivated by the need of expressive and fast sequence models that are able to process sequential data with time-dependent interactions over (very) long time-scales, we first propose to leverage the structure of fully coupled nonlinear driven oscillators in this context by constructing the coupled oscillatory RNN (CoRNN) in Chapter 2. We show that CoRNN is able to learn long-term dependencies by mitigating the exploding and vanishing gradients problem, a central issue for deep (or recurrent) neural networks prohibiting information to flow over long time scales. Moreover, we provide extensive empirical evidence on challenging synthetic as well as real-world datasets, demonstrating that CoRNN performs comparably to state-of-the-art recurrent sequence models. While fully coupled oscillatory systems (i.e., CoRNN) perform very well on sequential data, even faster methods (in terms of training and inference time) are needed to process sequential data with very long sequence length (i.e., sequence length > 10k). To this end, in Chapter 3 we propose the undamped independent controlled oscillatory RNN (UnICORNN), which is based on independent (i.e., uncoupled) nonlinear oscillators. This enables a very fast implementation on GPUs, where each independent neuron (i.e., dimension of the underlying dynamical system) is solved in a (possibly independent) CUDA thread. Moreover, since the underlying dynamics of UnICORNN denotes a time-dependent Hamiltonian systems, we can apply Liouville's theorem on the preservation of the phase space volume to show that UnICORNN is perfectly invertible in time. This makes UnICORNN very memory efficient, as no intermediate layers have to be stored for the backpropagation through time algorithm but can simply be reconstructed based on the very last hidden state. This drastically reduces the memory requirement to train UnICORNN. Thus, UnICORNN can potentially be used for continuous online training on memory-constrained edge devices. Finally, we rigorously show that UnICORNN mitigates the exploding and vanishing gradients

problem and can thus successfully be applied to sequential data exhibiting long-term dependencies. We subsequently show that UnICORNN reaches state-of-the-art performance and outperforms every other recurrent sequence model on a variety of versatile sequence tasks, where the length of the sequences range from 1k up to $\sim$ 18k. While coRNN and UnICORNN are empirically very successful, it is unclear if these models are universal. Moreover, based on the oscillatory inductive bias, it is natural to ask if these models are biased towards oscillatory functions. To answer this question empirically we note that both CoRNN as well as UnICORNN have successfully been applied to a large variety of different datasets that certainly do not dominantly exhibit oscillatory representations (e.g., datasets in computer vision and NLP). However, we also provide a rigorous theoretical answer to this question in Chapter 4. In this chapter, we generalize CoRNN and UnICORNN to neural oscillators and rigorously prove that these models are indeed universal approximators for continuous and causal operators.

In the second part of the thesis, we discuss the necessity of machine learning models to be able to process (real-world) data potentially exhibiting multiple scales. This motivates LEM in Chapter 5, a novel physics-inspired sequence model that we derive from dynamical systems with multiple time-adaptive learnable scales. We rigorously prove that LEM is able to learn long-term dependencies. Moreover, we show that LEM is a universal approximator for general (Lipschitz continuous) dynamical systems. On top of that, we show that LEM is even universal within the class of multiscale dynamical systems, where the size of LEM is independent of the different scales, which is in stark contrast to standard RNNs. Finally, we show that LEM does not only reach state-of-the-art results among recurrent models on tasks exhibiting long-term dependencies, but also on tasks that require high expressive power of the underlying model (e.g., in natural language understanding).

The third part of this thesis introduces graph representation learning in Chapter 6. In particular it highlights a central issue impairing the expressive power of deep GNNs, namely oversmoothing. This phenomenon describes the exponential convergence of all node features towards the same constant node vector with respect to increasing number of GNN layers. Motivated by unsynchronized graph-coupled oscillators to preserve node feature diversity over time, we present GraphCON in Chapter 7. This physics-inspired framework for learning on graphs is derived from graph-dynamical systems modelling interactions of nonlinear oscillators coupled through a graph. GraphCON can be seen as a general framework allowing to stack many GNN layers and propagating information according to the dynamics of graph-coupled nonlinear oscillators. We subsequently prove that GraphCON indeed mitigates the oversmoothing issue and allows for the construction of deep GNNs. Finally, we show that GraphCON outperforms state-of-the-art graph-learning models on a variety of different graph datasets, namely heterophilic and homophilic node-level tasks as well as graph-level tasks.

Last but not least we present gradient gating ($\mathbf{G}^2$) in Chapter 8. This framework is based on gating the output of GNN layers with a mechanism for multi-rate flow of message-passing information across nodes of the underlying graph. Local gradients are harnessed to further modulate message-passing updates. Our framework flexibly allows one to use any basic GNN layer as a wrapper around which the multi-rate gradient gating mechanism is built. Moreover, $\mathbf{G}^2$ can be used as a wrapper around other GNN architectures that perform poorly on heterophilic data (e.g., GCN, GAT), and turns them into powerful methods on such data. We rigorously prove that $\mathbf{G}^2$ alleviates the oversmoothing problem and allows the design of deep GNNs. Finally, we show that $\mathbf{G}^2$ vastly outperforms state-of-the-art models in particular on heterophilic large-scale graph datasets.

The results in this thesis thus represent a collection of very early approaches in the exciting field of physics-inspired machine learning. In particular, it emphasizes the symbiosis of machine learning and physics, and provides a recipe on how to tackle problems in machine learning research from a physics perspective. Moreover, it demonstrates its potential to provide not only empirical solutions but also theoretical answers to challenging problems in various aspects of modern machine learning.

Future research directions. Although the presented models and methods have been extensively tested on synthetic and real-world datasets, we have mainly applied them to datasets in classical machine learning, e.g. in image recognition, speech recognition, NLP, time series, and citation networks. However, only few experiments have been done in the context of physics and scientific computing. Since these models are based and inspired by physical systems (and thus exhibit physics-based inductive biases), one would expect these models to excel on problems in the physical sciences. Hence, an important future research direction is to test the proposed methods on their ability to solve problems in the physical sciences and analyse the role of their physics-based inductive bias therein.

Another important open research direction is the theoretical understanding of the proposed inductive biases. For instance, in 5.4 we saw that LEM exhibits multiscale behavior after training, as the proportion of occurrences of the learned scales decays as a power law. However, it is unclear how this power law and the corresponding multiscale predictions correlate with the scales in the underlying dataset, in particular for real-world data such as images or natural text. Thus, a theoretical analysis of this phenomenon is needed, which potentially gives rise to a better understanding of natural and real-world data from a multiscale perspective.

While this thesis focuses on sequence modelling and graph representation learning, there are many more fields of machine learning research that can profit from physics-inspired approaches. For instance generative modelling, a field of machine learning in which, instead of fitting functions to data, the distribution that generates the data is learned. In this context, neural oscillators as of Chapter 4 denote promising models, as they are invertible (and potentially symplectic) by design. This makes neural oscillators suitable candidates for normalizing flows [Rezende and Mohamed, 2015]. Moreover, current state-of-the-art methods in generative modelling are diffusion models [Yang et al., 2022], that incrementally corrupt input data with random noise (i.e., diffuse the input data), and generate new data points by inverting this process. Since diffusion models are already based on a physical process, namely diffusion, it is no surprise that recent advances in this field are based on physical insights into the underlying diffusion process [Dockhorn et al., 2021, Lai et al., 2022, Salimans and Ho, 2021]. However, little work has been done on the theoretical understanding of these models. Therefore, an important future research direction is the theoretical understanding of (score-based) diffusion models (possibly with further physical inductive biases) along the same lines of Chapter 4 providing in-depth theoretical understanding of oscillatory neural architectures.

A very recent area of modern machine learning research deals with the development of generalist learners – a single model capable of performing various different (possibly out-of-distribution) tasks. Examples of such include recent advances in large language models [Brown et al., 2020, Chowdhery et al., 2022], neural algorithmic learning [Ibarz et al., 2022], and general-purpose agents [Reed et al., 2022]. Since physics represents a mathematical generalist description of natural phenomena and engineering systems, physics-inspired machine learning models denote a promising approach for setting up novel and powerful generalist learners.

The author hopes that the presented work in this thesis may serve as a basis for future research on physics-inspired machine learning, as well as a guidance for tackling and addressing challenging problems in machine learning research from a physics perspective.

Bibliography

S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

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

D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop on Ambient Assisted Living*, pages 216–223. Springer, 2012.

J. Appleyard, T. Kocisky, and P. Blunsom. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*, 2016.

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

V. I. Arnold. *Mathematical methods of classical mechanics*. Springer Verlag, New York, 1989.

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

A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 770–778, 2020.

A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory.*, 39(3):930–945, 1993.

S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):2453, 2022.

D. Beani, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò. Directional graph networks. In *ICML*. PMLR, 2021.

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

D. Bo, X. Wang, C. Shi, and H. Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3950–3957, 2021.

C. Bodnar, F. Di Giovanni, B. P. Chamberlain, P. Liò, and M. M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *arXiv preprint arXiv:2202.04579*, 2022.

G. E. Box and G. C. Tiao. *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011.

O. M. Braun and Y. Kivshar. Nonlinear dynamics of the frenkel-kontorova model. *Physics Reports*, 306: 1–108, 1998.

X. Bresson and T. Laurent. Residual gated graph convnets. *arXiv:1711.07553*, 2017.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478*, 2021.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

V. Campos, B. Jou, X. Giró-i-Nieto, J. Torres, and S. Chang. Skip RNN: learning to skip state updates in recurrent neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

M. L. Casado. Trivializations for gradient-based optimization on manifolds. In *Advances in Neural Information Processing Systems*, pages 9154–9164, 2019.

S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, 2007.

B. Chamberlain, J. Rowbottom, D. Eynard, F. Di Giovanni, X. Dong, and M. Bronstein. Beltrami flow and neural diffusion on graphs. In *NeurIPS*, 2021a.

B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. M. Bronstein, S. Webb, and E. Rossi. GRAND: graph neural diffusion. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 1407–1418. PMLR, 2021b.

B. Chang, M. Chen, E. Haber, and E. H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020a.

J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv:1710.10568*, 2017.

M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *ICML*. PMLR, 2020b.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. Symplectic recurrent neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020c.

W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

J. Choi, S. Hong, N. Park, and S.-B. Cho. Gread: Graph neural reaction-diffusion equations. *arXiv preprint arXiv:2211.14208*, 2022.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

R. R. Coifman and S. Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. Principal neighbourhood aggregation for graph nets. *arXiv:2004.05718*, 2020.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL `https://doi.org/10.1007/BF02551274`.

A. Deac, M. Lackenby, and P. Veličković. Expander graph propagation. *arXiv preprint arXiv:2210.02997*, 2022.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Veličković. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

R. Dey and F. M. Salemt. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

F. Di Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022.

T. Dockhorn, A. Vahdat, and K. Kreis. Score-based generative modeling with critically-damped langevin diffusion. *arXiv preprint arXiv:2112.07068*, 2021.

V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *arXiv:2003.00982*, 2020.

W. E. A proposal on machine learning via dynamical systems. *Commun. Math. Stat*, 5:1–11, 2017.

M. Eliasof, E. Haber, and E. Treister. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. In *NeurIPS*, 2021.

N. B. Erichson, O. Azencot, A. Queiruga, and M. W. Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

L. C. Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *CVPR*, 2018.

M. A. Finzi, R. Bondesan, and M. Welling. Probabilistic numeric convolutional neural networks. In *9th International Conference on Learning Representations, ICLR*, 2021.

R. Fitzhugh. Mathematical models of threshold phenomena in the nerve membrane. *Bull. Math. Biophysics*, 17:257–278, 1955.

P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Trans. Neural Networks*, 9(5):768–786, 1998.

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29:1019–1027, 2016.

T. Gaudelet, B. Day, A. R. Jamasb, J. Soman, C. Regep, G. Liu, J. B. Hayter, R. Vickers, C. Roberts, J. Tang, et al. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics*, 22(6), 2021.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

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

F. D. Giovanni, T. K. Rusch, M. M. Bronstein, A. Deac, M. Lackenby, S. Mishra, and P. Veličković. How does over-squashing affect the power of gnns? *arXiv preprint arXiv:2306.03589*, 2023.

C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *ICNN*, 1996.

L. Gonon, L. Grigoryeva, and J.-P. Ortega. Risk bounds for reservoir computing. *arXiv:1910.13886*, 2019.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IJCNN*, 2005.

S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15379–15389, 2019.

L. Grigoryeva and J.-P. Ortega. Echo state networks are universal. *Neural Networks*, 108:495 – 508, 2018. ISSN 0893-6080.

A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re. Hippo:recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems*, pages 1474–1487, 2020.

A. Gu, K. Goel, and C. Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

B.-M. Gu, H. vanRijn, and W. K. Meck. Oscillatory multiplexing of neural population codes for interval timing and working memory. *Neuroscience and Behaviorial reviews,*, 48:160–185, 2015.

J. Guckenheimer and P. Holmes. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Springer Verlag, New York, 1990.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34, 2018.

E. Hairer, S. P. Norsett, and G. Wanner. *Solving ordinary differential equations I*. Springer, 1987.

E. Hairer, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the störmer-verlet method. *Acta Numerica*, 14:399–450, 2003.

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

B. Han, C. Wang, and K. Roy. Oscillatory fourier neural network: A compact and efficient architecture for sequential processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(6), pages 6838–6846, 2022.

R. Hasani, M. Lechner, T.-H. Wang, M. Chahine, A. Amini, and D. Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. Helfrich, D. Willmott, and Q. Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978. PMLR, 2018.

M. Henaff, A. Szlam, and Y. LeCun. Recurrent orthogonal networks and long-memory tasks. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2034–2042, 2016.

G. Hinton and D. Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the cognitive science society*, pages 177–186, 1987.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. Hodgkin and A. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500–544, 1952.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `http://www.sciencedirect.com/science/article/pii/0893608089900208`.

B. Ibarz, V. Kurin, G. Papamakarios, K. Nikiforou, M. Bennani, R. Csordás, A. J. Dudzik, M. Bošnjak, A. Vitvitskyi, Y. Rubanova, et al. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, pages 2–1. PMLR, 2022.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *arXiv:2106.02039*, 2021.

A. Kag and V. Saligrama. Time adaptive recurrent neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15149–15158, June 2021.

A. Kag, Z. Zhang, and V. Saligrama. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

T. A. Keller and M. Welling. Locally coupled oscillatory recurrent networks learn traveling waves and topographic organization. In *Cosyne abstracts*, 2023.

G. Kerg, K. Goyette, M. P. Touzel, G. Gidel, E. Vorontsov, Y. Bengio, and G. Lajoie. Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics. In *Advances in Neural Information Processing Systems*, pages 13591–13601, 2019.

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

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. C. Courville, and C. J. Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

C. Kuehn. *Multiple time scale dynamics*, volume 191. Springer, 2015.

A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pages 9017–9028, 2018.

C.-H. Lai, Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon. Regularizing score-based models with score fokker-planck equations. *arXiv preprint arXiv:2210.04296*, 2022.

S. Lanthaler, T. K. Rusch, and S. Mishra. Neural oscillators are universal. In *Conference on Neural Information Processing Systems*, 2023.

T. Laurent and J. von Brecht. A recurrent neural network without chaos. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

M. Lezcano-Casado and D. 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.

S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466, 2018.

S. Li, W. Li, C. Cook, Y. Gao, and C. Zhu. Deep independently recurrent neural network (indrnn). *arXiv preprint arXiv:1910.06251*, 2019.

X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian. Finding global homophily in graph neural networks when meeting heterophily. *arXiv preprint arXiv:2205.07308*, 2022.

Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. In Y. Bengio and Y. LeCun, editors, *ICLR*, 2016.

D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021a.

L.-H. Lim. Hodge laplacians on graphs. *arXiv preprint arXiv:1507.05379*, 2015.

S. H. Lim, N. B. Erichson, L. Hodgkinson, and M. W. Mahoney. Noisy recurrent neural networks. *arXiv preprint arXiv:2102.04877*, 2021b.

M. Liu, H. Gao, and S. Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019a.

Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*, 2019b.

E. N. Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on Predictability*, volume 1, 1996.

S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint arXiv:2109.05641*, 2021.

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 142–150. Association for Computational Linguistics, 2011.

W. Maass. Fast sigmoidal networks via spiking neurons. *Neural Computation*, 9:279–304, 2001.

M. MacKay, P. Vicol, J. Ba, and R. B. Grosse. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 9029–9040, 2018.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

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

N. McGreivy and A. Hakim. Convolutional layers are not translation equivariant. *arXiv preprint arXiv:2206.04979*, 2022.

T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, 2017.

J. Morrill, P. Kidger, C. Salvi, J. Foster, and T. Lyons. Neural cdes for long time series via the log-ode method. *arXiv preprint arXiv:2009.08295*, 2020.

G. Namata, B. London, L. Getoor, B. Huang, and U. EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

A. Norcliffe, C. Bodnar, B. Day, N. Simidjievski, and P. Liò. On second order behaviour in augmented neural odes. *Advances in neural information processing systems*, 33:5911–5921, 2020.

H. Nt and T. Maehara. Revisiting graph neural networks: all we have is low pass filters. *arXiv:1812.08434v4*, 2019.

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

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762v1*, 2019.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28 of *ICML'13*, page III–1310–III–1318. JMLR.org, 2013.

H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

M. A. Pimentel, A. E. Johnson, P. H. Charlton, D. Birrenkott, P. J. Watkinson, L. Tarassenko, and D. A. Clifton. Toward a robust estimation of respiratory rate from pulse oximeters. *IEEE Transactions on Biomedical Engineering*, 64(8):1914–1923, 2016.

A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8(1):143–195, 1999.

M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, and J. Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

A. Queiruga, N. B. Erichson, L. Hodgkinson, and M. W. Mahoney. Compressing deep ode-nets using basis function expansions. *arXiv preprint arXiv:2106.10820*, 2021.

A. F. Queiruga, N. B. Erichson, D. Taylor, and M. W. Mahoney. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

M. Romera and et. al. Vowel recognition with four coupled spin-torque nano-oscillators. *Nature*, 588: 230–234, 2018.

D. W. Romero, R.-J. Bruintjes, J. M. Tomczak, E. J. Bekkers, M. Hoogendoorn, and J. C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*, 2021a.

D. W. Romero, A. Kuzina, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021b.

Y. Rong, W. Huang, T. Xu, and J. Huang. Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

F. Rosenblatt. Principles of neurodynamics: perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

B. Rozemberczki, C. Allen, and R. Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

T. K. Rusch and S. Mishra. Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021a.

T. K. Rusch and S. Mishra. Unicornn: A recurrent model for learning very long time dependencies. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9168–9178. PMLR, 2021b.

T. K. Rusch, B. Chamberlain, J. Rowbottom, S. Mishra, and M. Bronstein. Graph-coupled oscillator networks. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18888–18909. PMLR, 2022a.

T. K. Rusch, S. Mishra, N. B. Erichson, and M. W. Mahoney. Long expressive memory for sequence modeling. In *International Conference on Learning Representations*, 2022b.

T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

T. Salimans and J. Ho. Should ebms model the energy or the score? In *Energy Based Models Workshop-ICLR 2021*, 2021.

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

J. Sanz Serna and M. Calvo. *Numerical Hamiltonian problems*. Chapman and Hall, London, 1994.

V. G. Satorras, E. Hoogeboom, and M. Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2008.

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

J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

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

J. Shlomi, P. Battaglia, and J.-R. Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

J. T. Smith, A. Warrington, and S. W. Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

A. Sperduti. Encoding labeled graphs by labeling RAAM. In *NIPS*, 1994.

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

K. Stachenfeld, D. B. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, P. Battaglia, and A. Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021.

I. Steinwart and A. Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

K. M. Stiefel and G. B. Ermentrout. Neurons as oscillators. *Journal of Neurophysiology*, 116:2950–2960, 2016.

S. Strogatz. *Nonlinear Dynamics and Chaos*. Westview, Boulder CO, 2015.

S. H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 2001.

C. Tallec and Y. Ollivier. Can recurrent neural networks warp time? In *6th International Conference on Learning Representations, ICLR*, 2018.

C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb. Monash university, uea, ucr time series regression archive. *arXiv preprint arXiv:2006.10996*, 2020.

J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv:2111.14522*, 2021.

J. Torrejon and et. al. Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547: 428–431, 2017.

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

A. Velichko, M. Belyaev, and P. Boriskov. A model of an oscillatory neural network with multilevel neurons for pattern recognition and computing. *Electronics*, 8, 2019.

P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR*, 2018.

P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.

P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

L. Wei, H. Zhao, and Z. He. Designing the topology of graph neural networks: A novel feature fusion perspective. In *Proceedings of the ACM Web Conference 2022*, pages 1381–1391, 2022.

P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981*, pages 762–770. Springer, 1982.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990.

S. Wiggins. *Introduction to nonlinear dynamical systems and chaos*. Springer, 2003.

A. T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 16:15–42, 1967.

S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022.

L.-P. Xhonneux, M. Qu, and J. Tang. Continuous graph neural networks. In *ICML*. PMLR, 2020.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv:1810.00826*, 2018a.

K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*. PMLR, 2018b.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3): 645–678, 2005.

Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, Y. Shao, W. Zhang, B. Cui, and M.-H. Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022.

R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, , C. Li, and M. Sun. Graph neural networks: a review of methods and applications. *arXiv:1812.08434v4*, 2019.

K. Zhou, X. Huang, D. Zha, R. Chen, L. Li, S.-H. Choi, and X. Hu. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021.

J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

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

M. Zitnik and J. Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.