Neural Differential Equations for Learning to Program Neural Nets Through Continuous Learning Rules

Kazuki Irie[1] **Francesco Faccio**[1] **Jürgen Schmidhuber**[1,2]
[1]The Swiss AI Lab, IDSIA, USI & SUPSI, Lugano, Switzerland
[2]AI Initiative, KAUST, Thuwal, Saudi Arabia
{kazuki, francesco, juergen}@idsia.ch

Abstract

Neural ordinary differential equations (ODEs) have attracted much attention as continuous-time counterparts of deep residual neural networks (NNs), and numerous extensions for recurrent NNs have been proposed. Since the 1980s, ODEs have also been used to derive theoretical results for NN learning rules, e.g., the famous connection between Oja's rule and principal component analysis. Such rules are typically expressed as additive iterative update processes which have straightforward ODE counterparts. Here we introduce a novel combination of learning rules and Neural ODEs to build continuous-time sequence processing nets that learn to manipulate short-term memory in rapidly changing synaptic connections of other nets. This yields continuous-time counterparts of Fast Weight Programmers and linear Transformers. Our novel models outperform the best existing Neural Controlled Differential Equation based models on various time series classification tasks, while also addressing their fundamental scalability limitations. Our code is public.[1]

1 Introduction

Neural ordinary differential equations (NODEs) [1] have opened a new perspective on continuous-time computation with neural networks (NNs) as a practical framework for machine learning based on differential equations. While the original approach—proposed as a continuous-depth version of deep feed-forward residual NNs [2, 3]—only covers autonomous ODEs entirely determined by the initial conditions, more recent extensions deal with sequential data (reviewed in Sec. 2.1) in a way similar to what is typically done with standard recurrent NNs (RNNs) in the discrete-time scenario. This potential for continuous-time (CT) sequence processing (CTSP) is particularly interesting, since there are many applications where datapoints are observed at irregularly spaced time steps, and CT sequence models might better deal with such data than their discrete-time counterparts. However, the development of NODEs for CTSP is still at an early stage. For example, a popular approach of Neural Controlled Differential Equations [4] (NCDEs; also reviewed in Sec. 2.1) has in practice only one architectural variant corresponding to the "vanilla" RNN [5]. Discrete-time processing, however, exploits many different RNN architectures as well as Transformers [6].

While it is not straightforward to transform the standard Transformer into a CT sequence processor, we'll show that the closely related Fast Weight Programmers (FWPs) [7, 8, 9, 10] and linear Transformers [11] (reviewed in Sec. 2.3) have direct CT counterparts. In FWPs, temporal processing of short-term memory (stored in fast weight matrices) uses learnable sequences of *learning rules*. Hence CT versions of FWPs will require differential equations to model the learning rules. This relates to a trend of the 1980s/90s. Among many old connections between NNs and dynamical systems described by ODEs (e.g., [12, 13, 14, 15, 16, 17]), the theoretical analysis of NN learning rules in the ODE

[1] https://github.com/IDSIA/neuraldiffeq-fwp

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

framework has been particularly fruitful. Consider the famous example of Oja's rule [18] (briefly reviewed in Sec. 2.2): many results on its stability, convergence, and connection to Principal Component Analysis [19, 20] were obtained using its ODE counterpart (e.g., [18, 21, 22, 23, 24, 25, 26, 27]).

Here we propose a novel combination of Neural ODEs and learning rules, to obtain a new class of sequence processing Neural ODEs which are continuous-time counterparts of Fast Weight Programmers and linear Transformers. The resulting models are general-purpose CT sequence-processing NNs, which can directly replace the standard Neural CDE models typically used for supervised CT sequence processing tasks. To the best of our knowledge, there is no previous work on Neural ODE-based Transformer families for CT sequence processing, despite their popularity in important types of discrete time computation such as Natural Language Processing and beyond. We also show how our approach solves the fundamental limitation of existing Neural CDEs in terms of model size scalability.

We conduct experiments on three standard time series classification tasks covering various scenarios (regularly sampled, irregularly sampled with missing values, and very long time series). We demonstrate that our novel models outperform existing Neural ODE-based sequence processors, in some cases by a large margin.

2 Background

We briefly review the main background concepts this work builds upon: NODEs for sequence processing (Sec. 2.1), NN learning rules and their connection to ODEs (Sec. 2.2), and Fast Weight Programmers whose memory update is based on learning rules controlled by an NN (Sec. 2.3).

2.1 Neural ODEs (NODEs) and Their Extensions for Sequence Processing

Here we review the core idea of NODEs [1]. In what follows, let n, N, d, d_{in} denote positive integers, T be a positive real number, and θ denote an arbitrary set of real numbers. We consider a residual layer (say, the n-th layer with a dimension d) in an N-layer deep NN which transforms an input $\boldsymbol{h}_{n-1} \in \mathbb{R}^d$ to an output $\boldsymbol{h}_n \in \mathbb{R}^d$ with a parameterised function $\boldsymbol{f}_\theta : \mathbb{R}^d \to \mathbb{R}^d$ as follows:

$$\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + \boldsymbol{f}_\theta(\boldsymbol{h}_{n-1}) \tag{1}$$

This coincides [28, 29, 30, 31, 32, 33, 34, 1] with the following equation for $\epsilon = 1$

$$\boldsymbol{h}(t_n) = \boldsymbol{h}(t_{n-1}) + \epsilon \boldsymbol{f}_\theta(\boldsymbol{h}(t_{n-1})) \tag{2}$$

where $\boldsymbol{h} : [0, T] \to \mathbb{R}^d$ is a function such that $\boldsymbol{h}(t_n) = \boldsymbol{h}_n$ holds for all $n : 0 \leq n \leq N$ and $t_n \in [0, T]$ such that $t_n - t_{n-1} = \epsilon > 0$ if $n \geq 1$. This equation is a forward Euler discretisation of the ordinary differential equation defined for all $t \in (t_0, T]$ as

$$\boldsymbol{h}'(t) = \boldsymbol{f}_\theta(\boldsymbol{h}(t)) \quad \text{or} \quad \boldsymbol{h}(t) = \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{f}_\theta(\boldsymbol{h}(s))ds \tag{3}$$

where $\boldsymbol{h}'$ denotes the first order derivative. This establishes the connection between the ODE and the deep residual net with parameters θ shared across layers[2]: given the initial condition $\boldsymbol{h}(t_0) = \boldsymbol{h}_0$, the solution to this equation evaluated at time T, i.e., $\boldsymbol{h}(T)$, corresponds to the output of this deep residual NN, which can be computed by an ODE solver. We denote it as a function $\mathrm{ODESolve}$ taking four variables: $\boldsymbol{h}(T) = \mathrm{ODESolve}(\boldsymbol{f}_\theta, \boldsymbol{h}_0, t_0, T)$. During training, instead of backpropagating through the ODE solver's operations, the continuous *adjoint sensitivity method* [35] (which essentially solves another ODE but backward in time) can compute gradients with $O(d)$ memory requirement, constant w.r.t. T [1].

A natural next step is to extend this formulation for RNNs, i.e., the index n now denotes the time step, and we assume an external input $\boldsymbol{x}_n \in \mathbb{R}^{d_{\text{in}}}$ at each step n to update the hidden state $\boldsymbol{h}_{n-1}$ to $\boldsymbol{h}_n$ as

$$\boldsymbol{h}_n = \boldsymbol{f}_\theta(\boldsymbol{h}_{n-1}, \boldsymbol{x}_n) \tag{4}$$

Depending on the property of external inputs $(\boldsymbol{x}_n)_{n=1}^N = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$, there are different ways of defining NODEs for sequence processing. We mainly distinguish three cases.

[2] Or we make θ dependent of t such that parameters are "depth/layer-dependent" as in standard deep nets.

First, when there is a possibility to construct a *differentiable* control signal $\boldsymbol{x} : t \mapsto \boldsymbol{x}(t) \in \mathbb{R}^{d_{\text{in}}}$ for $t \in [t_0, T]$ from the inputs $(\boldsymbol{x}_n)_{n=1}^N$; an attractive approach by Kidger et al. [4] handles the corresponding dynamics in a *neural controlled differential equation* (NCDE):

$$\boldsymbol{h}(t) = \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{h}(s)) d\boldsymbol{x}(s) = \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{h}(s)) \boldsymbol{x}'(s) ds \tag{5}$$

where $\boldsymbol{F}_\theta$ is a parameterised function (typically a few-layer NN) which maps a vector $\boldsymbol{h}(s) \in \mathbb{R}^d$ to a matrix $\boldsymbol{F}_\theta(\boldsymbol{h}(s)) \in \mathbb{R}^{d \times d_{\text{in}}}$ (we already relate this component to Recurrent Fast Weight Programmers below) and thus, $\boldsymbol{F}_\theta(\boldsymbol{h}(s)) d\boldsymbol{x}(s)$ denotes a matrix-vector multiplication. There are several methods to construct the *control* $\boldsymbol{x} : [t_0, T] \to \mathbb{R}^{d_{\text{in}}}$ based on the discrete data points $(\boldsymbol{x}_n)_{n=1}^N$, such that its differentiability is guaranteed. In this work, we follow Kidger et al. [4] and mainly use natural cubic splines over all data points (which, however, makes it incompatible with auto-regressive processing); for better alternatives, we refer to Morrill et al. [36]. Since the final equation is again an NODE with a vector field of form $\boldsymbol{g}_{\theta,\boldsymbol{x}'}(s, \boldsymbol{h}(s)) = \boldsymbol{F}_\theta(\boldsymbol{h}(s)) \boldsymbol{x}'(s)$, all methods described above are applicable: ODE solver for evaluation and continuous adjoint method for memory efficient training. A notable extension of Neural CDEs is the use of *log-signatures* to sub-sample the input sequence [37]. The resulting NCDEs are called *neural rough differential equations* (NRDEs), which are relevant for processing long sequences. One fundamental limitation of the NCDEs above is the lack of scalability of $\boldsymbol{F}_\theta : \mathbb{R}^d \to \mathbb{R}^{d \times d_{\text{in}}}$. For example, if we naively parameterise $\boldsymbol{F}_\theta$ using a linear layer, the size of its weight matrix is $d^2 * d_{\text{in}}$ which quadratically increases with the hidden state size d. Previous attempts [38] do not successfully resolve this issue without performance degradation. In Sec. 5, we'll discuss how our models (Sec. 3.2) naturally circumvent this limitation while remaining powerful NCDEs.

On a side note, the NCDE is often referred to as the "continuous-time analogue" to RNNs [4], but this is a bit misleading: discrete-time RNN equations corresponding to the continuous-time Eq. 5 do not reflect the standard RNN of Eq. 4 but:

$$\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + \boldsymbol{W}_{n-1}(\boldsymbol{x}_n - \boldsymbol{x}_{n-1}) \tag{6}$$

$$\boldsymbol{W}_n = \boldsymbol{F}_\theta(\boldsymbol{h}_n) \tag{7}$$

where one network (Eq. 6) learns to translate the variation of inputs $(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})$ into a change in the state space, using a weight matrix $\boldsymbol{W}_{n-1}$ which itself is generated by another network ($\boldsymbol{F}_\theta : \mathbb{R}^d \to \mathbb{R}^{d \times d_{\text{in}}}$; Eq. 7) on the fly from the hidden state. This model is thus a kind of Recurrent FWP [39, 40, 10].

Second, even if $\boldsymbol{x}$ is not differentiable, having access to $\boldsymbol{x}$ defined and bounded over an interval of interest $[t_0, T]$ is enough to define a sequence processing NODE, by making it part of the vector field:

$$\boldsymbol{h}(t) = \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{f}_\theta(\boldsymbol{h}(s), \boldsymbol{x}(s)) ds \tag{8}$$

where the vector field $\boldsymbol{f}_\theta(\boldsymbol{h}(t), \boldsymbol{x}(t)) = \boldsymbol{g}_{\theta,\boldsymbol{x}}(t, \boldsymbol{h}(t))$ can effectively be evaluated at any time $t \in [t_0, T]$. We refer to this second approach as a *direct NODE* method. While Kidger et al. [4] theoretically and empirically show that this approach is less expressive than the NCDEs above, we'll show how in our case of learning rules one can derive interesting models within this framework, which empirically perform on par with the CDE variants.

Finally, when no control function with one of the above properties can be constructed, a mainstream approach dissociates the continuous-time hidden state update via ODE for the time between two observations (e.g., Eq. 9 below) from integration of the new data (Eq. 10 below). Notable examples of this category include ODE-RNNs [41, 42] which transform the hidden states $\boldsymbol{h}_{n-1}$ to $\boldsymbol{h}_n$ for each observation $\boldsymbol{x}_n$ available at time t_n as follows:

$$\boldsymbol{u}_n = \text{ODESolve}(\boldsymbol{f}_{\theta_1}, \boldsymbol{h}_{n-1}, t_{n-1}, t_n) \tag{9}$$

$$\boldsymbol{h}_n = \boldsymbol{\phi}_{\theta_2}(\boldsymbol{x}_n, \boldsymbol{u}_n) \tag{10}$$

where Eq. 9 autonomously updates the hidden state between two observations using a function $\boldsymbol{f}_{\theta_1}$ parameterised by θ_1, while in Eq. 10, function $\boldsymbol{\phi}_{\theta_2}$ parameterised by θ_2 integrates the new input $\boldsymbol{x}_n$ into the hidden state. In Latent ODE-RNN [41], a popular extension of this approach to the variational setting, the initial recurrent state $\boldsymbol{h}_0$ is sampled from a prior (during training, an additional encoder is trained to map sequences of inputs to parameters of the prior). While this third case is not our focus, we'll also show how to use FWPs in this scenario in Sec. 3.3 for the sake of completeness.

2.2 Learning Rules and Their Connections to ODEs

Learning rules of artificial NNs describe the process which modifies their weights in response to some inputs. This includes the standard backpropagation rule (also known as the reverse mode of automatic differentiation) derived for the case of supervised learning, as well as rules inspired by Hebb's informal rule [43] in "unsupervised" settings. Here we focus on the latter. Let n, d_{in} d_{out} be positive integers. Given a linear layer with a weight matrix $\boldsymbol{W}_n \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ (the single output neuron case $d_{\text{out}} = 1$ is the focus of the classic works) at time n which transforms input $\boldsymbol{x}_n \in \mathbb{R}^{d_{\text{in}}}$ to output $\boldsymbol{y}_n \in \mathbb{R}^{d_{\text{out}}}$ as

$$\boldsymbol{y}_n \quad = \quad \boldsymbol{W}_{n-1}\boldsymbol{x}_n \tag{11}$$

the pure Hebb-style additive learning rule modifies the weights according to

$$\boldsymbol{W}_n \quad = \quad \boldsymbol{W}_{n-1} + \eta_n \boldsymbol{y}_n \otimes \boldsymbol{x}_n \tag{12}$$

where $\otimes$ denotes outer product and $\eta_n \in \mathbb{R}_+$ is a learning rate at time n.

Oja [18] proposed stability improvements to this rule through a decay term

$$\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \eta_n \boldsymbol{y}_n \otimes (\boldsymbol{x}_n - \boldsymbol{W}_{n-1}^\top \boldsymbol{y}_n) \tag{13}$$

whose theoretical analysis has since the 1980s been a subject of many researchers covering stability, convergence, and relation to Principal Component Analysis [18, 21, 22, 23, 24, 25, 26, 27, 44]. One key approach for such theoretical analysis is to view the equation above as a discretisation of the following ODE:

$$\boldsymbol{W}'(t) = \eta(t)\boldsymbol{y}(t) \otimes (\boldsymbol{x}(t) - \boldsymbol{W}(t-1)^\top \boldsymbol{y}(t)) \tag{14}$$

On a related note, the study of RNNs has also profited from ODEs (e.g., [45, 46]).

2.3 Fast Weight Programmers & Linear Transformers

Fast Weight Programmers (FWP; [7, 8, 9, 10]) are general-purpose (auto-regressive) sequence processing NNs. In general, an FWP is a system of two NNs: a *slow* NN, the *programmer*, rapidly generates during runtime weight changes of another neural network, the *fast* NN. The (slow) weights of the slow net are typically trained by gradient descent. Variants of FWPs whose weight generation is based on outer products between keys and values [7] have been shown [9] to be equivalent to Linear Transformers [11] (using the mathematical equivalence known from perceptron/kernel machine duality [47, 48]). These FWPs use sequences of learning rules to update short-term memory in form of a fast weight matrix. A practical example of such FWPs is the DeltaNet [9] which transforms an input $\boldsymbol{x}_n \in \mathbb{R}^{d_{\text{in}}}$ into an output $\boldsymbol{y}_n \in \mathbb{R}^{d_{\text{out}}}$ at each time step n while updating its fast weight matrix $\boldsymbol{W}_{n-1} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}$ as follows:

$$\beta_n, \boldsymbol{q}_n, \boldsymbol{k}_n, \boldsymbol{v}_n \quad = \quad \boldsymbol{W}_{\text{slow}}\boldsymbol{x}_n \tag{15}$$

$$\boldsymbol{W}_n \quad = \quad \boldsymbol{W}_{n-1} + \sigma(\beta_n)(\boldsymbol{v}_n - \boldsymbol{W}_{n-1}\phi(\boldsymbol{k}_n)) \otimes \phi(\boldsymbol{k}_n) \tag{16}$$

$$\boldsymbol{y}_n \quad = \quad \boldsymbol{W}_n \phi(\boldsymbol{q}_n) \tag{17}$$

where the slow net (Eq. 15; with weights $\boldsymbol{W}_{\text{slow}} \in \mathbb{R}^{(1+2*d_{\text{key}}+d_{\text{out}}) \times d_{\text{in}}}$) generates key/value vectors $\boldsymbol{k}_n \in \mathbb{R}^{d_{\text{key}}}$ and $\boldsymbol{v}_n \in \mathbb{R}^{d_{\text{out}}}$ as well as a scalar $\beta_n \in \mathbb{R}$ to obtain a dynamic learning rate by applying a sigmoid function σ, and ϕ is an element-wise activation function whose output elements are positive and sum up to one (typically softmax). These fast dynamic variables generated by a slow NN are used in a learning rule (Eq. 16) akin to the classic delta rule [49] to update the fast weight matrix. The output is finally produced by the forward computation of the fast NN, i.e., by *querying* the fast weight matrix by the generated query vector $\boldsymbol{q}_n \in \mathbb{R}^{d_{\text{key}}}$ (Eq. 17). An intuitive interpretation of the fast weight matrix is a key-value associative memory with write and read operations defined by Eq. 16 and 17, respectively. This encourages intuitive thoughts about memory capacity (limited by the number of "keys" we can store without interference) [9]. For instance, if we replace the learning rule (i.e., memory writing operation) of Eq. 16 by a pure additive Hebb-style rule (and a fixed learning rate of 1.0): $\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \boldsymbol{v}_n \otimes \phi(\boldsymbol{k}_n)$, we obtain the Linear Transformer [11] (we refer to prior work [9] for further explanations of the omission of attention normalisation). Such a purely additive learning rule often suffers from long term dependencies, unlike the delta rule [9]. We'll confirm this trend also in the CT models (using the EigenWorms dataset). For later convenience, we introduce a notation FWP which denotes generic FWP operations: $\boldsymbol{y}_n, \boldsymbol{W}_n = \text{FWP}(\boldsymbol{x}_n, \boldsymbol{W}_{n-1}; \boldsymbol{W}_{\text{slow}})$.

3 Continuous-Time Fast Weight Programmers

We propose continuous-time counterparts of Fast Weight Programmers (Sec. 2.3) which naturally combine ODEs for learning rules (Sec. 2.2) and existing approaches for sequence processing with NODEs (Sec. 2.1). We present three types of these CT FWP models in line with the categorisation of Sec. 2.1 while the main focus of this work is on the two first cases.

3.1 Direct NODE-based FWPs

In the *direct NODE* approach (reviewed in Sec. 2.1), we assume a (bounded) control signal $\boldsymbol{x} : t \mapsto \boldsymbol{x}(t)$ defined over an interval $[t_0, T]$. We make it part of the vector field to define an ODE describing a *continuous-time learning rule* for a fast weight matrix $\boldsymbol{W}(t)$:

$$\boldsymbol{W}(t) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s))ds \tag{18}$$

where $\boldsymbol{W} : t \mapsto \boldsymbol{W}(t) \in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}$ is a function defined on $[t_0, T]$, and $\boldsymbol{F}_\theta$ is an NN parameterised by θ which maps onto $\mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}$. This is a neural differential equation for learning to program a neural net through continuous learning rules, that is, to train a fast weight matrix $\boldsymbol{W}(t)$ of a fast NN (Eq. 20 below) for each sequential control $\boldsymbol{x}$. Like in the discrete-time FWPs (Sec. 2.3), the output $\boldsymbol{y}(T) \in \mathbb{R}^{d_{\text{out}}}$ is obtained by *querying* this fast weight matrix[3] (e.g., at the last time step T):

$$\boldsymbol{q}(T) = \boldsymbol{W}_q \boldsymbol{x}(T) \tag{19}$$

$$\boldsymbol{y}(T) = \boldsymbol{W}(T)\boldsymbol{q}(T) \tag{20}$$

where $\boldsymbol{W}_q \in \mathbb{R}^{d_{\text{key}} \times d_{\text{in}}}$ is a slow weight matrix used to generate the query $\boldsymbol{q}(T) \in \mathbb{R}^{d_{\text{key}}}$ (Eq. 19). Now we need to specify $\boldsymbol{F}_\theta$ in Eq. 18 to fully define the learning rule. We focus on three variants:

$$\boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s)) = \sigma(\beta(s)) \begin{cases} \boldsymbol{k}(s) \otimes \boldsymbol{v}(s) & \text{Hebb-style} \\ \boldsymbol{v}(s) \otimes \big(\boldsymbol{k}(s) - \boldsymbol{W}(s)^\top \boldsymbol{v}(s)\big) & \text{Oja-style} \\ \big(\boldsymbol{v}(s) - \boldsymbol{W}(s)\boldsymbol{k}(s)\big) \otimes \boldsymbol{k}(s) & \text{Delta-style} \end{cases} \tag{21}$$

where $[\beta(s), \boldsymbol{k}(s), \boldsymbol{v}(s)] = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}(s)$ with a slow weight matrix $\boldsymbol{W}_{\text{slow}} \in \mathbb{R}^{(1+d_{\text{key}}+d_{\text{out}}) \times d_{\text{in}}}$. As in the discrete-time FWP (Sec. 2.3), the slow NN generates $\beta(s) \in \mathbb{R}$ (to which we apply the sigmoid function σ to obtain a learning rate), key $\boldsymbol{k}(s) \in \mathbb{R}^{d_{\text{key}}}$ and value $\boldsymbol{v}(s) \in \mathbb{R}^{d_{\text{out}}}$ vectors from input $\boldsymbol{x}(s)$. These variants are inspired by the respective classic learning rules of the same name, while they are crucially different from the classic ones in the sense that all variables involved (key, value, learning rate) are continually generated by the slow NN. In the experimental section, we'll comment on how some of these design choices can result in task-dependent performance gaps. In practice, we use the *multi-head* version of the operations above (i.e., by letting H be a positive integer denoting the number of heads, query/key/value vectors are split into H sub-vectors and Eqs. 20-21 are conducted independently for each head). The output is followed by the standard feed-forward block like in Transformers [6]. Possible extensions for deeper models are discussed in Appendix C.1.

3.2 NCDE-based FWPs

Here we present models based on NCDEs (reviewed in Sec. 2.1). We assume availability of a differentiable control signal $\boldsymbol{x}(t)$, whose first order derivative is denoted by $\boldsymbol{x}'(t)$. Given the NCDE formulation of Eq. 5, the most straight-forward approach to obtain a CT Fast Weight Programmer is to extend the dimensionality of the recurrent hidden state, i.e., we introduce a parameterised function F_θ which maps a matrix $\boldsymbol{W}(t) \in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}$ to a third-order tensor $\mathsf{F}_\theta(\boldsymbol{W}(t)) \in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}} \times d_{\text{in}}}$:

$$\boldsymbol{W}(t) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \mathsf{F}_\theta(\boldsymbol{W}(s))d\boldsymbol{x}(s) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \mathsf{F}_\theta(\boldsymbol{W}(s))\boldsymbol{x}'(s)ds \tag{22}$$

However, this approach is obviously not scalable since the input and output dimensions ($d_{\text{out}} \times d_{\text{key}}$ and $d_{\text{out}} \times d_{\text{key}} \times d_{\text{in}}$) of F_θ can be too large in practice. A more tractable CDE-based approach can

[3] In practice, we also apply element-wise activation functions to query/key/value vectors where appropriate, which we omit here for readability. We refer to Appendix A for further details.

be obtained by providing $\boldsymbol{x}$ and/or $\boldsymbol{x}'$ to the vector field:

$$\boldsymbol{W}(t) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \mathbf{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s))\boldsymbol{x}'(s)ds \tag{23}$$

While this equation still remains a CDE because of the multiplication from the right by $d\boldsymbol{x} = \boldsymbol{x}'(s)ds$, the additional inputs to the vector field offer a way of making use of various learning rules, as in the case of direct NODE approach above (Sec. 3.1). To be specific, either $\boldsymbol{x}$ and $\boldsymbol{x}'$ or only $\boldsymbol{x}'$ is required in the vector field to obtain these tractable CDEs. Here we present the version which uses both $\boldsymbol{x}$ and $\boldsymbol{x}'$ [4]. The resulting vector fields for different cases are:

$$\mathbf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s) = \sigma(\beta(s)) \begin{cases} \boldsymbol{W}_k\boldsymbol{x}(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Hebb} \\ \big(\boldsymbol{W}_k\boldsymbol{x}(s) - \boldsymbol{W}(s)^\top\boldsymbol{W}_v\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Oja} \\ \big(\boldsymbol{W}_v\boldsymbol{x}(s) - \boldsymbol{W}(s)\boldsymbol{W}_k\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \text{Delta} \end{cases} \tag{24}$$

As can be seen above, the use of CDEs to describe a continuous fast weight learning rule thus naturally results in a key/value memory where $\boldsymbol{x}'$ is used to generate either key or value vectors. Because of the multiplication from the right by $\boldsymbol{x}'$, the role of $\boldsymbol{x}'$ changes depending on the choice of learning rule: $\boldsymbol{x}'$ is used to generate the key in the Delta case but the value vector in the case of Oja. In the case of Hebb, the choice made in Eq. 24 of using $\boldsymbol{x}$ for keys and $\boldsymbol{x}'$ for values is arbitrary since Eq. 24 is symmetric in terms of roles of keys and values (see an ablation study in Appendix C.2 for the other case where we use $\boldsymbol{x}'$ to generate the key and $\boldsymbol{x}$ for the value). The querying operation (analogous to Eqs. 19-20 for the direct NODE case) is also modified accordingly, depending on the choice of learning rule, such that the same input ($\boldsymbol{x}$ or $\boldsymbol{x}'$) is used to generate both key and query:

$$\boldsymbol{y}(T) = \begin{cases} \boldsymbol{W}(T)^\top\boldsymbol{W}_q\boldsymbol{x}(T) & \text{Hebb and Oja} \\ \boldsymbol{W}(T)\boldsymbol{W}_q\boldsymbol{x}'(T) & \text{Delta} \end{cases} \tag{25}$$

Note that since the proposed vector field $\mathbf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s)$ is more general than the one used in the original NCDE $\mathbf{F}_\theta\big(\boldsymbol{W}(s)\big)\boldsymbol{x}'(s)$, any theoretical results on the CDE remain valid (which, however, does not tell us anything about the best choice for its exact parameterisation).

3.3 ODE-RFWP and Latent ODE-RFWP

The main focus of this work is the setting of Kidger et al. [4] where we assume the existence of some control signal $\boldsymbol{x}$ (Sec. 3.1 and 3.2 above). However, here we also show a way of using FWPs in the third/last case presented in Sec. 2.1 where no control $\boldsymbol{x}(t)$ is available (or can be constructed), i.e., we only have access to discrete observations $(\boldsymbol{x}_n)_{n=0}^N$. Here we cannot directly define the vector field involving continuous transformations using the inputs. We follow the existing approaches (ODE-RNN or Latent ODE; Sec. 2.1) which use two separate update functions: A discrete recurrent state update is executed every time a new observation is available to the model, while a continuous update using an autonomous ODE is conducted in between observations. Unlike with standard recurrent state vectors, however, it is not practical to autonomously evolve high-dimensional fast weight matrices[5]. We therefore opt for using a Recurrent FWP (RFWP) [10] and combine it with an ODE:

$$\boldsymbol{u}_n = \text{ODESolve}(\boldsymbol{f}_{\theta_1}, \boldsymbol{h}_{n-1}, t_{n-1}, t_n) \tag{26}$$

$$\boldsymbol{h}_n, \boldsymbol{W}_n = \text{FWP}([\boldsymbol{x}_n, \boldsymbol{u}_n], \boldsymbol{W}_{n-1}; \theta_2) \tag{27}$$

where we keep the fast weight learning rule itself discrete (Eq. 27), but evolve the recurrent state vector $\boldsymbol{u}_n$ using an ODE (Eq. 26) such that the information to be read/written to the fast weight matrix is controlled by a variable which is continuously updated between observations. We refer to this model as ODE-RFWP and its variational variant as Latent ODE-RFWP.

Since we focus on continuous-time learning rules, the case above is not of central interest as the learning rule remains discrete here (Eq. 27). Nevertheless, in Appendix C.3, we also provide some experimental results for model-based reinforcement learning settings corresponding to this case.

[4] The equations for the version using only $\boldsymbol{x}'$ can be obtained by replacing $\boldsymbol{x}$ by $\boldsymbol{x}'$ in Eq. 24. We provide an ablation in Appendix C.2. As a side note, we also obtain the equation for the CDE using only $\boldsymbol{x}'$ by replacing $\boldsymbol{x}$ by $\boldsymbol{x}'$ in Eq. 18 for the direct NODE case.

[5] Such an approach would require a computationally expensive matrix-to-matrix transforming NN, whose scalability is limited.

Table 1: **Accuracy (%) on the Speech Commands classification task** and **AUC ($\times 10^2$) on the PhysioNet Sepsis prediction task**. PhysioNet has two cases: with (OI) or without (no-OI) observational intensity (see text for details). Numbers marked by * are taken from Kidger et al. [4]. Mean and standard deviation (std) are computed over 5 runs.

Type	Model	Speech Commands	PhysioNet Sepsis	
			OI	no-OI
Direct NODE	GRU-ODE [4]*	47.9 (2.9)	85.2 (1.0)	77.1 (2.4)
	Hebb	82.8 (1.1)	**90.4 (0.4)**	82.9 (0.7)
	Oja	**85.4 (0.9)**	88.9 (1.4)	82.9 (0.5)
	Delta	81.5 (3.8)	89.8 (1.0)	**84.5 (2.9)**
CDE	NCDE [4]*	89.8 (2.5)	88.0 (0.6)	77.6 (0.9)
	Hebb	89.5 (0.3)	89.9 (0.6)	**85.7 (0.3)**
	Oja	90.0 (0.7)	**91.2 (0.4)**	85.1 (2.5)
	Delta	**90.2 (0.2)**	90.9 (0.2)	84.5 (0.7)

4 Experiments

We consider three datasets covering three types of time series which are regularly sampled (Speech Commands [50]), irregularly sampled with partially missing features (PhysioNet Sepsis [51]), or very long (EigenWorms [52]). We compare the proposed direct NODE and CDE based FWP models (Sec. 3.1 & 3.2) to NODE baselines previously reported on the same datasets [4, 36]. Appendix B provides further experimental details including hyper-parameters.

Speech Commands. The Speech Commands [50] is a single word speech recognition task. The datapoints are regularly sampled, and the sequence lengths are relatively short ($\leq$ 160 frames), which makes this task a popular sanity check. Following prior work on NCDEs [4], we use 20 mel frequency cepstral coefficients as speech features and classify the resulting sequence to one out of ten keywords. The middle column of Table 1 shows the results. The table is split into the direct NODE (top) and CDE (bottom) based approaches. We first observe that among the direct NODE approaches, all our FWPs largely outperform ($\geq$ 80% accuracy) the baseline GRU-ODE performance of 47.9% (the best direct NODE baseline from Kidger et al. [4]). This demonstrates that with a good parameterisation of the vector field, the direct NODE approach can achieve competitive performance. On the other hand, all CDE-based approaches yield similar performance. We also only see slight differences in terms of performance among different learning rules, without a clear winner for this task. This may indicate that the ordinary nature of this task (regularly sampled; short sequences) does not allow for differentiating among these CDE models, including the baseline.

PhysioNet Sepsis. The PhysioNet Sepsis is a dataset of the sepsis prediction task from the PhysioNet challenge 2019 [51]. This is again a dataset used by Kidger et al. [4] to evaluate NCDEs. The task is a binary prediction of sepsis from a time series consisting of measurements of 34 medical features (e.g., respiration rate) of patients' stays at an ICU. Each sequence is additionally labelled by five static features of the patient (e.g., age) which are fed to the model to generate the initial state of the ODE. Sequences are relatively short ($\leq$ 72 frames) but datapoints are irregularly sampled and many entries are missing, which makes this task challenging. It comes in two versions: with and without the so-called *observation intensity* information (denoted as "OI" and "no-OI") which is one extra input feature indicating each observation's time stamp (providing the models with information on measurement frequency). This distinction is important since the prior work [4] has reported that existing ODE/CDE-based approaches struggle with the no-OI case of this task. Following the previous work, we report the performance in terms of Area Under the ROC Curve (AUC). The right part of Table 1 shows the results. We obtain large improvements in the no-IO case (from 77.6 to 85.7% for the CDEs and from 77.1 to 84.5% for the direct NODEs), while also obtaining small improvements in the OI case (from 85.2 to 90.4% for direct NODEs, and from 88.0 to 91.2% for CDEs). The no-OI performance of our models is also comparable to the best overall performance reported by Kidger [38]: 85.0 % (1.3) achieved by GRU-D [53]. This demonstrates the efficacy of

Table 2: **Classification Accuracy (%) on the EigenWorms task.** Numbers marked by * are taken from Morrill et al. [37]. Mean and standard deviation (std) are computed over 5 runs. "Sig-Depth" indicates the depth of the signature (with this number equal to 1, an RDE is reduced to a CDE). To facilitate comparisons to prior work [37], we also add the column "Step" indicating the sequence down-sampling factor (even if we fix it to the best value [37] of 4).

Model	Sig-Depth	Step	Test Acc. [%]
NRDE [37]*	2	4	83.8 (3.0)
Hebb	2	4	45.6 (5.9)
Oja			46.7 (7.5)
Delta			**87.7 (1.9)**
NCDE [37]*	1	4	66.7 (11.8)
Hebb	1	4	41.0 (6.5)
Oja			49.7 (9.9)
Delta			**91.8** (3.4)

CT FWP model variants for handling irregularly sampled data with partially missing features even in the case without frequency information. Differences between various learning rules are rather small again. In some cases, we observe performance to be very sensitive to hyper-parameters. For example, the best Oja-CDE configuration achieves 85.1% (2.5) with a learning rate of 6e-5, while this goes down to 79.6% (4.7) when the learning rate is changed to 5e-5.

EigenWorms. The EigenWorms dataset (which is part of the UEA benchmark [52]) is a 5-way classification of roundworm types based on time series tracking their movements. To be more specific, motions of a worm are represented by six features corresponding to their projections to six template movement shapes, called "eigenworms." While this dataset contains only 259 examples, it is notable for its very long sequences (raw sequence lengths exceed 17 K) and long-span temporal dependencies [37, 54, 55]. We use the same train/validation/test split ratio as the prior work [37] which reports Neural RDEs (NRDEs) as achieving the best NODE model performance on this dataset. The equations of our CT FWPs for the RDE case can be straightforwardly obtained by replacing the input $\boldsymbol{x}$ in Eqs. 18-19 of the direct NODE formulation (or, $\boldsymbol{x}$ and $\boldsymbol{x}'$ in Eqs. 23-25 of the NCDEs) by the corresponding log-signatures. Table 2 shows the results, where "Step" denotes the time sub-sampling rate which is fixed to 4 for which the prior work [37] reports the best NRDE and NCDE performance. "Sig-Depth" denotes the depth of the log-signature (the deeper, the more log-signature terms we take into account, thus ending up with a larger input feature vector; we refer to the original paper [37] for further details). We consider two values for this parameter: 1 and 2. When set to 1, the input feature contains only the first derivative $\boldsymbol{x}'(s)$ and thus the NRDE is reduced to an NCDE (with controls constructed via linear interpolation). We take the best NCDE performance from Morrill et al. [37] as the depth-1 baseline. Morrill et al. [37] report the best overall performance for the depth-2 NRDE (depth-2 baseline in our table). In both cases, we first note a large performance gap between models with different learning rules. While the naive Hebb and Oja based models struggle with this very long sequence processing (sequence length still exceeds 4 K with a down-sampling step size of 4), the Delta rule performs very well. This confirms the prior result in the discrete-time domain [9] which motivated the Delta rule design by its potential for handling long sequences (we refer to prior work [9] for further explanations). Since its performance on other tasks is comparable to the one of Hebb and Oja variants, the Delta rule is a natural default choice for parameterising the CT FWPs.

In both the depth-1 and depth-2 cases, we obtain large improvements compared to the respective baselines. It is counter-intuitive to find certain depth-2 models underperforming their depth-1 counterparts, but this trend has also been observed in the original NRDEs [36]. Our best overall performance is obtained in the depth-1 case: 91.8 % (3.4) exceeds the previous best NRDE based model performance of 83.8 % (3.0) [36]. This almost matches the state-of-the-art accuracy of 92.8 % (1.8) reported by Rusch et al. [55] (using an ODE-inspired discrete-time model). Our model's performance variance is high (best single seed performance is 97.4% while the standard deviation is 3.4). The

wall clock time is similar for our best model (last row in Table 2) and the NRDE baseline (36s/epoch on a GeForce RTX 2080 Ti) and their sizes are comparable (87 K vs. 65 K parameters respectively).

5 Discussions

Scalability Advantage Compared to Standard NCDEs. In addition to the good empirical results shown above, our FWP approach also addresses an important limitation of existing NCDEs [4]: their scalability in terms of model size. The vector field in standard NCDEs (Eq. 5) requires an NN $\boldsymbol{F}_\theta$ which takes a vector $\boldsymbol{h}(s) \in \mathbb{R}^d$ as an input to produce a matrix of size $\mathbb{R}^{d \times d_{\text{in}}}$. This can be very challenging when d_{in} or/and d is large. Actually, the same bottleneck is present in the weight generation of FWPs [7]. The use of outer products can remediate this issue in discrete FWPs as well as in CT FWPs: the computations in our FWP-based NODE/NCDEs only involve "first-order" dimensions (i.e., no multiplication between different dimensions, such as $d \times d_{\text{in}}$) for NN outputs. This can scale well with increased model size, making feasible larger scale tasks infeasible for existing NCDEs. On the other hand, Kidger et al. [4] report that using outer products (in their "Sec. 6.1 on limitations") in standard NCDEs does not perform well. Why do outer products work well in our models but not in the original NCDEs? The answer may be simple. In the original NCDEs (Eq. 5), multiplications occur at each (infinitesimal) time step between the generated rank-one weight matrix $\boldsymbol{F}_\theta(\boldsymbol{h}(s))$ and $\boldsymbol{x}'(s)$ before the sum. All these transformations are thus of rank one while we expect expressive transformations to be necessary to translate $\boldsymbol{x}'(s)$ into changes in the state space. In contrast, in our CT FWPs, the ODE only parameterises the weight generation process of another net, and thus the rank-one matrices are never used in isolation: they are summed up over time (Eq. 18 or 23) to form an expressive weight matrix which is only then used for matrix multiplication (Eq. 20 or 25. The proposed FWP-NODE/NCDEs thus offer scalable alternatives to existing NCDEs, also yielding good empirical performance (Sec. 4).

Importance of Memory Efficient Backpropagation for FWPs. Memory efficiency of continuous adjoint backpropagation may be not so important for standard NCDEs of state size $O(d)$, but is crucial for FWPs of state size $O(d^2)$ which can quickly become prohibitive for long sequences, as naive backpropagation stores all states used in the forward pass. Prior works on discrete FWPs [9, 56, 10] solve this problem by a custom memory-efficient implementation. Here, the continuous adjoint method naturally addresses this problem.

Limitations. Our ablation studies and hyper-parameter tuning focus on optimising the model configuration/architecture. From the NODE perspective, other parameters may further improve performance or alleviate performance variability/stability issues observed in some cases (Sec. 4). For example, we use the numerical solver configurations of the baselines (see Appendix B) without tuning them. Similarly, we use natural cubic splines to construct the differentiable control signals for NCDEs in the Speech Commands and PhysioNet Sepsis tasks, following the original NCDE paper [4]. Morrill et al. [36] report performance enhancements by improving the corresponding interpolation methods (e.g., 93.7% on Speech Commands). Such further optimisation is not conducted here.

Generally speaking, real benefits of using continuous-time sequence processing models are yet to be proved. While we achieve improvements over the best existing NODE/NCDE models on multiple datasets, discrete-time models tailored to the corresponding problem still perform as well or even better than our improved CT models (e.g., Rusch et al. [55] for EigenWorms and Che et al. [53] for PhysioNet Sepsis; see Sec. 4).

Related Work on Parameter/Weight ODEs. Other works use ODEs to parameterise the time-evolving weights of some model. However, they are limited to autonomous ODEs (i.e., no external control $\boldsymbol{x}$ is involved). Zhang et al. [57] and Choromanski et al. [58] study coupled ODEs where one ODE is used for temporal evolution of parameters of the main Neural ODE. The scope of these two works is limited to autonomous ODEs corresponding to continuous-depth residual NNs with different parameters per depth. Deleu et al. [59] consider an ODE version of a gradient descent learning process for adaptation, but also formulated as an autonomous ODE. In contrast, our focus is really on *sequence processing* where the model continuously receives external controls $\boldsymbol{x}$ and translates them into weight changes of another network.

6 Conclusion

We introduced novel continuous-time sequence processing neural networks that learn to use sequences of ODE-based continuous learning rules as elementary programming instructions to manipulate short-term memory in rapidly changing synaptic connections of another network. The proposed models are continuous-time counterparts of Fast Weight Programmers and linear Transformers. Our new models experimentally outperform by a large margin existing Neural ODE based sequence processors on very long or irregularly sampled time series. Our Neural ODE/CDE based FWPs also address the fundamental scalability problem of the original Neural CDEs, which is highly promising for future applications of ODE based sequence processors to large scale problems.

7 Acknowledgements

We would like to thank Kidger et al. [4], Morrill et al. [37] and Du et al. [60] for their public code. This research was partially funded by ERC Advanced grant no: 742870, project AlgoRNN, and by Swiss National Science Foundation grant no: 200021_192356, project NEUSYM. We are thankful for hardware donations from NVIDIA and IBM. The resources used for this work were partially provided by Swiss National Supercomputing Centre (CSCS) project s1145 and s1154.

References

[1] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 6572–6583, Montréal, Canada, December 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016.

[3] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *the Deep Learning workshop at Int. Conf. on Machine Learning (ICML)*, Lille, France, July 2015.

[4] Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. Neural controlled differential equations for irregular time series. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual Only, December 2020.

[5] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, Long Beach, CA, USA, December 2017.

[7] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991.

[8] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[9] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.

[10] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2021.

[11] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2020.

[12] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[13] Alan Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. *Technical Report No. LA-UR-87-2662*, 1987.

[14] Fernando J Pineda. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

[15] Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[16] Masa-aki Sato and Yoshihiko Murakami. Learning nonlinear dynamics by recurrent neural. In *Some Problems on the Theory of Dynamical Systems in Applied Sciences-Proceedings of the Symposium*, volume 10, page 49, 1991.

[17] Ramiro Rico-Martinez, K Krischer, Ioannis Kevrekidis, MC Kube, and JL Hudson. Discrete-vs. continuous-time nonlinear signal processing of Cu electrodissolution data. *Chemical Engineering Communications*, 118(1):25–48, 1992.

[18] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[19] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[20] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[21] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106 (1):69–84, 1985.

[22] Erkki Oja. Neural networks, principal components, and subspaces. *International journal of neural systems*, 1(01):61–68, 1989.

[23] Mark D Plumbley. Lyapunov functions for convergence of principal component algorithms. *Neural Networks*, 8(1):11–23, 1995.

[24] Kurt Hornik and C-M Kuan. Convergence analysis of local feature extraction algorithms. *Neural Networks*, 5(2):229–240, 1992.

[25] Terence D Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6):459–473, 1989.

[26] John L. Wyatt Jr. and Ibrahim M. Elfadel. Time-domain solutions of Oja's equations. *Neural Computation*, 7(5):915–922, 1995.

[27] Jean-Claude Fort and Gilles Pages. Convergence of stochastic algorithms: From the Kushner–Clark theorem to the Lyapounov functional method. *Advances in applied probability*, 28(4): 1072–1094, 1996.

[28] Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11, March 2017.

[29] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, December 2017.

[30] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales - multiscale methods for convolution neural networks. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 3142–3148, New Orleans, LA, USA, February 2018.

[31] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2811–2818, New Orleans, LA, USA, February 2018.

[32] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 3282–3291, Stockholm, Sweden, July 2018.

[33] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. In *Int. Conf. on Learning Representations (ICLR)*, Vancouver, Canada, April 2018.

[34] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino J. Gomez. NAIS-Net: Stable deep networks from non-autonomous differential equations. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 3029–3039, Montréal, Canada, December 2018.

[35] Liev S. Pontryagin, VG Boltyanskii, RV Gamkrelidze, and EF Mishchenko. *LS Pontryagin Selected Works: The Mathematical Theory of Optimal Processes*. 1962.

[36] James Morrill, Patrick Kidger, Lingyi Yang, and Terry Lyons. Neural controlled differential equations for online prediction tasks. *Preprint arXiv:2106.11028*, 2021.

[37] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 7829–7838, Virtual only, July 2021.

[38] Patrick Kidger. *On Neural Differential Equations*. PhD thesis, Mathematical Institute, University of Oxford, 2021.

[39] Jürgen Schmidhuber. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In *International Conference on Artificial Neural Networks (ICANN)*, pages 460–463, Amsterdam, Netherlands, September 1993.

[40] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *Int. Conf. on Learning Representations (ICLR)*, Toulon, France, April 2017.

[41] Yulia Rubanova, Tian Qi Chen, and David Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 5321–5331, Vancouver, Canada, December 2019.

[42] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 7377–7388, Vancouver, Canada, December 2019.

[43] Donald Olding Hebb. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology*, 62:78, 1949.

[44] Chi-Ning Chou and Mien Brabeeba Wang. ODE-inspired analysis for the biological version of Oja's rule in solving streaming PCA. In *Proc. Conf. on Learning Theory (COLT)*, pages 1339–1343, Virtual only, July 2020.

[45] Huaguang Zhang, Zhanshan Wang, and Derong Liu. A comprehensive review of stability analysis of continuous-time recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 25(7):1229–1262, 2014.

[46] Adeline Fermanian, Pierre Marion, Jean-Philippe Vert, and Gérard Biau. Framing RNN as a kernel method: A neural ODE approach. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2021.

[47] Mark A. Aizerman, Emmanuil M. Braverman, and Lev I. Rozonoer. Theoretical foundations of potential function method in pattern recognition. *Automation and Remote Control*, 25(6): 917–936, 1964.

[48] Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. The dual form of neural networks revisited: Connecting test time predictions to training patterns via spotlights of attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, Baltimore, MD, USA, July 2022.

[49] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. In *Proc. IRE WESCON Convention Record*, pages 96–104, Los Angeles, CA, USA, August 1960.

[50] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *Preprint arXiv:1804.03209*, 2018.

[51] Matthew A. Reyna, Christopher Josef, Salman Seyedi, Russell Jeter, Supreeth P. Shashikumar, M. Brandon Westover, Ashish Sharma, Shamim Nemati, and Gari D. Clifford. Early prediction of sepsis from clinical data: the PhysioNet/computing in cardiology challenge 2019. In *Proc. Computing in Cardiology (CinC)*, pages 1–4, Singapore, September 2019.

[52] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The UEA multivariate time series classification archive, 2018. *Preprint arXiv:1811.00075*, 2018.

[53] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[54] T Konstantin Rusch and Siddhartha Mishra. UnICORNN: A recurrent model for learning very long time dependencies. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 9168–9178, Virtual only, July 2021.

[55] T Konstantin Rusch, Siddhartha Mishra, N Benjamin Erichson, and Michael W Mahoney. Long expressive memory for sequence modeling. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

[56] Kazuki Irie and Jürgen Schmidhuber. Training and generating neural networks in compressed weight space. In *ICLR Neural Compression Workshop*, Virtual only, May 2021.

[57] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E. Gonzalez, Kurt Keutzer, Michael W. Mahoney, and George Biros. ANODEV2: A coupled neural ODE framework. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 5152–5162, Vancouver, Canada, December 2019.

[58] Krzysztof Marcin Choromanski, Jared Quincy Davis, Valerii Likhosherstov, Xingyou Song, Jean-Jacques E. Slotine, Jacob Varley, Honglak Lee, Adrian Weller, and Vikas Sindhwani. Ode to an ODE. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2020.

[59] Tristan Deleu, David Kanaa, Leo Feng, Giancarlo Kerg, Yoshua Bengio, Guillaume Lajoie, and Pierre-Luc Bacon. Continuous-time meta-learning with forward mode differentiation. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

[60] Jianzhun Du, Joseph Futoma, and Finale Doshi-Velez. Model-based reinforcement learning for semi-Markov decision processes with neural ODEs. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, 2020.

[61] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language modeling with deep Transformers. In *Proc. Interspeech*, pages 3905–3909, Graz, Austria, September 2019.

[62] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5026–5033, Vilamoura, Portugal, October 2012. IEEE.

[63] Sahil Sharma, Aravind S. Lakshminarayanan, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *Int. Conf. on Learning Representations (ICLR)*, Toulon, France,, April 2017.

[64] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. 1994.

[65] Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

[66] Eduardo F Camacho and Carlos Bordons. *Model predictive control*. 2013.

[67] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[68] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Int. Conf. on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016.

[69] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proc. Int. Conf. on Machine Learning (ICML)*, Beijing, China, June 2014.

[70] Richard S Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[71] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1008–1014, Denver, CO, USA, November 1999.

[72] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. In *Proc. European Conference on Machine Learning (ECML)*, pages 280–291, Porto, Portugal, October 2005.

[73] Çagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We comment on a relatively high variance in the final performance in two sub-paragraphs "PhysioNet Sepsis" and "EigenWorms" under the experimental section.
   (c) Did you discuss any potential negative societal impacts of your work? [No] because we do not see any such impacts for this work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code is included in the supplemental material and it will be public if the paper is accepted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] in the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report means and standard deviations obtained for five seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Some essential numbers are reported in the main text. There is more in the appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See details in the appendix.
   (b) Did you mention the license of the assets? [Yes] Licence files are included in the code/supplemental material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, our own code.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use standard public benchmark datasets.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Further Model Specifications

For better readability, Sec. 3 omitted element-wise activation functions for query/key/value vectors. For example, for Eq. 21, the generation of key and value vectors, $\boldsymbol{k}$ and $\boldsymbol{v}$, can be fully described as

$$\beta(s), \boldsymbol{x_k}(s), \boldsymbol{x_v}(s) = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}(s) \tag{28}$$

$$\boldsymbol{k}(s) = \text{softmax}(\boldsymbol{x_k}(s)) \tag{29}$$

$$\boldsymbol{v}(s) = \tanh(\boldsymbol{x_v}(s)) \tag{30}$$

Eq. 19 for the query generation has to be replaced by

$$\boldsymbol{q}(T) = \text{softmax}(\boldsymbol{W}_q\boldsymbol{x}(T)) \tag{31}$$

These specifications are analogous in the CDE cases, i.e., in Eqs. 24-25.

While softmax has already been identified as a crucial component for stability in prior works on discrete-time FWPs [9], we apply an additional $\tanh$ to the value vectors (Eq. 30). Such usage of $\tanh$ in the output of vector fields for CDEs has been advocated by Kidger et al. [4] to improve the stability of the ODE solver. Indeed, we also generally found this beneficial, and sometimes crucial, for stable training of our models. For the Delta rule (Eq. 24), since we modify the value vector after projection (to take into account the "value" which is currently associated with the key vector), we consider two different ways of applying $\tanh$ as follows:

$$\boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s)) = \sigma(\beta(s)) \begin{cases} \big(\tanh(\boldsymbol{x_v}(s)) - \boldsymbol{W}(s)\boldsymbol{k}(s)\big) \otimes \boldsymbol{k}(s) & \text{pre-delta} \\ \tanh(\boldsymbol{x_v}(s) - \boldsymbol{W}(s)\boldsymbol{k}(s)) \otimes \boldsymbol{k}(s) & \text{post-delta} \end{cases} \tag{32}$$

For the task involving very long sequences ($>$ 4000 frames), we found the "post-delta" version to be crucial for successful training.

B Further Experimental Details

Our basic settings for data preparation and training are based on those used by prior works [4, 37] (we use their public implementations) for fair comparisons with the corresponding baselines.

Dataset/Task details. The essential information about the datasets has already been presented in Sec. 4. The number of sequences in the training set is about 24 K for the Speech Commands dataset and about 28 K for the PhysioNet Sepsis dataset.

Model/Training details. As mentioned in Sec. 3, all our FWP models make use of multiple computational heads, and their NODE/NCDE layer is followed by the standard Transformer feedforward block. The number of heads n_{head} and the feedforward inner dimension d_{ff} are hyper-parameters of the model, in addition to the size d_{model} of the NODE/NCDE layer. For all models considered in the main text, the number of layers is one (see Sec. C.1 below for a discussion of deeper models). We conducted hyper-parameter search in the following ranges, and selected the best configuration for each setting based on its validation performance.

For Speech Commands: $d_{\text{model}} \in \{32, 80, 128, 160\}$, $n_{\text{head}} \in \{4, 8, 16, 32\}$ and $d_{\text{ff}} = \{80, 4{*}d_{\text{model}}\}$, with a learning rate $\eta \in \{1e-5, 5e-5\}$ and a batch size of 1024. The best Delta CDE model was obtained for ($d_{\text{model}} = 80$, $n_{\text{head}} = 4$, $d_{\text{ff}} = 320$, $\eta = 5e-5$).

For PhysioNet Sepsis: $d_{\text{model}} \in \{32, 80, 128, 160\}$, $n_{\text{head}} \in \{8, 16, 32\}$ and $d_{\text{ff}} \in \{64, 4 * d_{\text{model}}\}$, with a learning rate $\eta \in \{1e-5, 2e-5, 3e-5, 4e-5, 5e-5, 6e-5, 8e-5, 1e-4\}$ and a batch size of 1024. The best Delta CDE model was obtained for ($d_{\text{model}} = 80$, $n_{\text{head}} = 16$, $d_{\text{ff}} = 64$, $\eta = 3e-5$) in the "OI" case, and for ($d_{\text{model}} = 160$, $n_{\text{head}} = 32$, $d_{\text{ff}} = 64$, $\eta = 4e-5$) in the "no-OI" setting.

For EigenWorms: $d_{\text{model}} \in \{64, 128, 160\}$, $n_{\text{head}} \in \{8, 16, 32\}$ and $d_{\text{ff}} \in \{32, 64, 80, 128\}$, with a learning rate $\eta \in \{5e-5, 1e-4, 3e-4, 5e-4\}$ and a batch size set to 181 to include all training sequences (70% of the 259 total sequences in this dataset, again following the prior work [37]). The best overall Delta CDE model was obtained for ($d_{\text{model}} = 128$, $n_{\text{head}} = 16$, $d_{\text{ff}} = 64$, $\eta = 1e-4$) in the log-signature depth-1 case.

The best hyper-parameters for all models are listed in Table 3.

Table 3: Hyper-parameters. Their definitions can be found in Appendix B. H/O/D denote Hebb/Oja/Delta variants respectively. The top part shows hyper-parameters of the direct NODE models for Speech Commands and PhysioNet Sepsis, and those of the NRDEs for EigenWorms. The bottom part is for the NCDE models.

		Speech Commands			PhysioNet Sepsis OI			PhysioNet Sepsis no-OI			EigenWorms		
Type	Params	H	O	D	H	O	D	H	O	D	H	O	D
O/RDE	d_{model}	128	128	160	160	160	128	32	160	80	160	160	128
	n_{head}	32	32	16	32	32	16	8	32	16	16	16	16
	d_{ff}	512	512	80	64	64	64	64	64	64	128	128	64
	η (1e-5)	5	5	1	6	6	2	10	4	4	30	30	10
CDE	d_{model}	128	128	80	32	80	80	160	160	160	160	128	128
	n_{head}	32	32	4	8	16	16	32	32	32	16	16	16
	d_{ff}	512	512	320	256	64	64	64	64	64	128	64	64
	η (1e-5)	5	5	5	6	2	3	4	6	4	30	30	10

Numerical solver details. For all tasks, we use the numerical solver settings of the baseline models. For all supervised learning experiments in the main text, the '`rk4`' Runge-Kutta variant with default tolerance parameters in `https://github.com/rtqichen/torchdiffeq` [1] is used: relative/absolute tolerance values are 1e-7/1e-9. For the RL experiment (Sec. C.3), '`dopri5`' is used instead, with relative/absolute tolerance values of 1e-5/1e-6. As mentioned in Sec. 5, we do not tune these configurations specifically for our models. Further enhancements in terms of performance or stability may be obtained by tuning them.

For any further details, we refer to our code.

C Extra Experiments

C.1 Deeper Models

As the datasets used here were rather small, one-layer models yielded satisfactory performance. In general, however, deeper architectures are common for Transformers [61]. Here we show ways of increasing the depth of FWP-based NODE/NCDE models. While the possibility of stacking multiple NCDE layers has been mentioned previously [38], no practical result has been reported.

The following equations are for the direct NODE case of our FWP models (the CDE case is analogous). Let L and l denote positive integers. The forward computation of layer l in an L-layer model at step t is as follows

$$\boldsymbol{W}(t,l) = \boldsymbol{W}(t_0,l) + \int_{s=t_0}^{t} \boldsymbol{F}_{\theta_l}^{(l)}(\boldsymbol{W}(s,l), \boldsymbol{x}(s,l-1))ds \tag{33}$$

$$\boldsymbol{x}(t,l) = \text{FFN}\left(\boldsymbol{W}(t,l)\text{softmax}(\boldsymbol{W}_q\boldsymbol{x}(t,l-1))\right) \tag{34}$$

where FFN denotes the standard Transformer feedforward block, $\boldsymbol{W}(t,l)$ denotes the fast weight matrix, $\boldsymbol{F}_{\theta_l}^{(l)}$ is the vector field with parameters θ_l, and $\boldsymbol{x}(t,l)$ denotes the output of layer $l \geq 1$ while $\boldsymbol{x}(t,0)$ is the external control signal. The output of layer l, $\boldsymbol{x}(t,l)$, thus becomes the control signal for the next layer $l+1$, and so on. This yields a system of L *coupled ODEs*. The fast weight matrix for each layer at time T can be obtained by calling the ODE solver for the corresponding coupled ODEs

$$\boldsymbol{W}(T,1),...,\boldsymbol{W}(T,L) = \text{ODESolve}([\boldsymbol{F}_{\theta_1}^{1},...,\boldsymbol{F}_{\theta_L}^{L}],[\boldsymbol{W}(t_0,1),...,\boldsymbol{W}(t_0,L)],t_0,T) \tag{35}$$

The final output $\boldsymbol{x}(T,L)$ can be computed recursively via Eq. 34 for $t=T$ starting from $l=1$.

We conducted experiments in the EigenWorms dataset with the goal of improving our best single-layer model of Table 2. We tuned the 2-layer model using the same hyper-parameter search space used for the 1-layer model. However, the best obtained 2-layer performance is 86.7 % (4.1) which

is worse than the best 1-layer performance of 91.8 % (3.4). A natural next step towards obtaining potential performance gains from deeper/larger models is to apply regularisation techniques to our models (we leave that to future work).

C.2 Ablation Studies on CDE-based Models

In Sec. 3.2 on NCDE based FWPs, we mentioned two optional model variations skipped in the main text: the Hebb variant which uses $\boldsymbol{x}'$ to generate keys and $\boldsymbol{x}$ to generate values, and NCDE based models which only use $\boldsymbol{x}'$ (instead of $\boldsymbol{x}$ and $\boldsymbol{x}'$) in the vector field (see footnote 4). Here we provide the corresponding ablation studies.

Using $\boldsymbol{x}'$ for key and $\boldsymbol{x}$ for value generation in the Hebb variant. In Sec. 3.2 on FWP based CDEs, we noted that there are two possible formulations for the Hebb variant. The following equations highlight the difference between the two formulations (depending on the variable, $\boldsymbol{x}$ or $\boldsymbol{x}'$, used to generate keys/values):

$$\mathsf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s) = \sigma(\beta(s)) \begin{cases} \boldsymbol{W}_k\boldsymbol{x}(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \boldsymbol{x}\text{-keys, } \boldsymbol{x}'\text{-values as in Eq. 24} \\ \boldsymbol{W}_v\boldsymbol{x}(s) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \boldsymbol{x}\text{-values, } \boldsymbol{x}'\text{-keys} \end{cases} \tag{36}$$

We also use different equations for the fast net forward computation where, for consistency, we generate the query from the same input ($\boldsymbol{x}$ or $\boldsymbol{x}'$) used to generate keys (even if this is not a strict requirement):

$$\boldsymbol{y}(T) = \begin{cases} \boldsymbol{W}(T)^\top \boldsymbol{W}_q\boldsymbol{x}(T) & \boldsymbol{x}\text{-keys, } \boldsymbol{x}'\text{-values} \\ \boldsymbol{W}(T)\boldsymbol{W}_q\boldsymbol{x}'(T) & \boldsymbol{x}\text{-values, } \boldsymbol{x}'\text{-keys} \end{cases} \tag{37}$$

Experimentally, we found this second variant ("$\boldsymbol{x}$-values, $\boldsymbol{x}'$-keys") to perform worse on the Speech Commands task. It obtained a test accuracy of 83.9 % (0.4) compared to 89.5 % (0.3) achieved by the first variant we reported in the main text (Table 1).

FWP based NCDEs using only $\boldsymbol{x}'$. In Sec. 3.2, we noted that tractable FWP based NCDEs can also be obtained by using only $\boldsymbol{x}'$ in the vector field (instead of $\boldsymbol{x}$ and $\boldsymbol{x}'$). The equations for the corresponding models can be obtained by replacing $\boldsymbol{x}$ by $\boldsymbol{x}'$ in Eq. 24:

$$\mathsf{F}_\theta\big(\boldsymbol{W}(s), \boldsymbol{x}'(s)\big)\boldsymbol{x}'(s) = \sigma(\beta(s)) \begin{cases} \boldsymbol{W}_k\boldsymbol{x}'(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Hebb} \\ \big(\boldsymbol{W}_k\boldsymbol{x}'(s) - \boldsymbol{W}(s)^\top\boldsymbol{W}_v\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Oja} \\ \big(\boldsymbol{W}_v\boldsymbol{x}'(s) - \boldsymbol{W}(s)\boldsymbol{W}_k\boldsymbol{x}'(s)\big) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \text{Delta} \end{cases} \tag{38}$$

On the Speech Commands task, we found it crucial to use both $\boldsymbol{x}$ and $\boldsymbol{x}'$. The second variants using only $\boldsymbol{x}'$ perform much worse (< 65 %) than the variants using both $\boldsymbol{x}$ and $\boldsymbol{x}'$, across all three learning rule schemes.

C.3 Model-based Reinforcement Learning (RL)

The main focus of this paper is the one of Kidger et al. [4] where it is assumed that at least some bounded control signal $\boldsymbol{x}(t)$ defined on $[t_0, T]$ is available. However, for the sake of completeness, in Sec. 3.3, we also present the ODE-RFWP and its latent variant in the case where we only have access to discrete inputs. Here we provide some experimental results with working examples of such models in a model-based RL setting previously proposed by Du et al. [60].

Settings. We use the MuJoCo environments [62]. Our setting follows the version with action repetitions [63] proposed by Du et al. [60], formalised as semi-Markov decision processes [64]. The core difference to the standard setting is that there is a (state-dependent) time gap $\tau > 0$ between the time when an agent takes an action in state $\boldsymbol{s}$ and when it observes a new state $\boldsymbol{s}'$ (and can take a new action), resulting in irregularly timed observations. The time gap τ is a deterministic function of the state $\boldsymbol{s}$. Du et al. [60] proposed a model-based planning approach where an environmental

model is parameterised as an ODE-RNN or Latent ODE-RNN such that an ODE is used to model the latent state transitions between observations. The model is trained to predict state transitions (using mean squared error for ODE-RNNs and negative evidence lower bounds for Latent ODE-RNNs) and time gaps between observations, and it is used for model predictive control (MPC) [65, 66, 67]. The agent is trained by Deep Deterministic Policy Gradient (DDPG) [68, 69] with actor-critic [70, 71, 72] where the predictions of policy and value networks are conditioned on the recurrent (latent) state of the environment model. In Du et al. [60]'s work, the Latent ODE-RNN was shown to outperform the ODE-RNN and the model-free baseline in terms of sample efficiency in *Hopper*, *Swimmer* and *Half Cheetah* environments. Therefore, we compare our Latent ODE-RFWP to the baseline Latent ODE-RNN. We conducted our experiments in *Hopper* and *Swimmer* environments (we excluded *Half Cheetah* which was reported [60] to require larger models and thus more compute). On a side note, there are other works [73] with a focus on RL in continuous-time environments. However, the available environments are still limited in terms of complexity (e.g., CartPole).

Training/Model details. Each training iteration consists of interactions with the environment to collect trajectories, training of the environmental model, and policy optimisation. Following Du et al. [60], each iteration consists of 5 K environmental steps, and we train for up to 400 K steps (corresponding to a total of 80 iterations) in addition to the initial random interactions of 15 K steps. The planning horizon for MPC is set to 10 steps. We set the hidden state size of our RFWP models to 128 like in the baseline. We use 16 computational heads, and a feedforward block size of 512. We found that we can use the same RNN-based encoder used in the baseline also for our RFWP models without notable difference in terms of performance compared to an FWP-based encoder. The same training/model configurations are used for both *Swimmer* and *Hopper*. They are identical to those used by Du et al. [60] (we use their base implementation), apart from the fact that we use the continuous adjoint method to train all models. For further details, we refer to our code.

Results. Figures 1a and 1b show learning curves of baseline Latent ODE-RNN and our RFWP variant in *Swimmer* and *Hopper* environments, respectively. Given a high variability of results across seeds, we present an average over ten training runs. We observe that final performance and sample efficiency of both models are comparable on *Swimmer*, while the RFWP model yields better performance on *Hopper*. These results indicate that the Latent ODE-RFWP model presented in Sec. 3.3 can effectively in practice be used as a replacement of the standard Latent ODE-RNN. We note, however, that the final performance of baseline Latent ODE-RNN is below the one reported by Du et al. [60] (which exceeds an expected return of 350). The only change we introduced to the original setting is to consistently use the continuous adjoint method to train all models (we may need further ablation studies to evaluate the impact of this change), and to use ten seeds instead of four.

(a) Swimmer

(b) Hopper

Figure 1: Return over training iterations in the **Swimmer** and **Hopper** environments, averaged over five test episodes. Smoothing is applied on overlapping windows of size 20. One iteration corresponds to 5 K interactions with the environment. Mean/std are computed for ten runs.