SEQ2TENS: AN EFFICIENT REPRESENTATION OF SEQUENCES BY LOW-RANK TENSOR PROJECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequential data such as time series, video, or text can be challenging to analyse as the ordered structure gives rise to complex dependencies. At the heart of this is non-commutativity, in the sense that reordering the elements of a sequence can completely change its meaning. We use a classical mathematical object – the free algebra – to capture this non-commutativity. To address the innate computational complexity of this algebra, we use compositions of low-rank tensor projections. This yields modular and scalable building blocks that give state-of-the-art performance on standard benchmarks such as multivariate time series classification and generative models for video.

1 INTRODUCTION

A central task of learning is to find representations of the underlying data that efficiently and faithfully capture their structure. In the case of sequential data, one data point consists of a sequence of objects. This is a rich and non-homogeneous class of data and includes classical uni- or multi-variate time series (sequences of scalars or vectors), video (sequences of images), and text (sequences of letters). Particular challenges of sequential data are that each sequence entry can itself be a highly structured object and that data sets typically include sequences of different length which makes naive vectorization troublesome.

Contribution. Our main result is a generic method that takes a *static feature map* for a class of objects (e.g. a feature map for vectors, images, or letters) as input and turns this into a feature map for sequences of arbitrary length of such objects (e.g. a feature map for time series, video, or text). We call this feature map for sequences Seq2Tens for reasons that will become clear; among its attractive properties are that it (i) provides a structured, parsimonious description of sequences; generalizing classical methods for strings, (ii) comes with theoretical guarantees such as universality, (iii) can be turned into modular and flexible neural network (NN) layers for sequence data. The key ingredient to our approach is to embed the feature space of the static feature map into a larger linear space that forms an algebra (a vector space equipped with a multiplication). The product in this algebra is then used to "stitch together" the static features of the individual sequence entries in a structured way. The construction that allows to do all this is classical in mathematics, and known as the *free algebra* (over the static feature space).

Outline. Section 2 formalizes the main ideas of Seq2Tens and introduces the free algebra $\mathrm{T}(V)$ over a space V as well as the associated product, the so-called *convolution tensor product*. It shows how low rank (LR) constructions combined with sequence-to-sequence transforms allows one to efficiently use this rich algebraic structure. Section 3 applies the results of Section 2 to build modular and scalable NN layers. Section 4 demonstrates the flexibility and modularity of this approach on both discriminative and generative benchmarks. Section 6 makes connections with previous work and summarizes this article. In the appendices we provide mathematical background, extensions, and detailed proofs for our theoretical results.

2 CAPTURING ORDER BY NON-COMMUTATIVE MULTIPLICATION

We denote the set of sequences of elements in a set $\mathcal{X}$ by

$$\mathrm{Seq}(\mathcal{X}) = \{\mathbf{x} = (\mathbf{x}_i)_{i=1,\ldots,L} : \mathbf{x}_i \in \mathcal{X},\, L \geq 1\} \tag{1}$$

where $L \geq 1$ is some arbitrary length. Even if $\mathcal{X}$ itself is a linear space, e.g. $\mathcal{X} = \mathbb{R}$, $\mathrm{Seq}(\mathcal{X})$ is never a linear space since there is no natural addition of two sequences of different length.

Seq2Tens in a nutshell. Given any vector space V we may construct the so-called free algebra $\mathrm{T}(V)$ over V. We describe the space $\mathrm{T}(V)$ in detail below, but as for now the only thing that is important is that $\mathrm{T}(V)$ is also a vector space that includes V, and that it carries a non-commutative product, which is, in a precise sense, "the most general product" on V.

The main idea of Seq2Tens is that any "static feature map" for elements in $\mathcal{X}$

$$\phi : \mathcal{X} \to V$$

can be used to construct a new feature map $\Phi : \mathrm{Seq}(\mathcal{X}) \to \mathrm{T}(V)$ for sequences in $\mathcal{X}$ by using the algebraic structure of $\mathrm{T}(V)$: the non-commutative product on $\mathrm{T}(V)$ makes it possible to "stitch together" the individual features $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_L) \in V \subset \mathrm{T}(V)$ of the sequence $\mathbf{x}$ in the larger space $\mathrm{T}(V)$ by multiplication in $\mathrm{T}(V)$. With this we may define the feature map $\Phi(\mathbf{x})$ for a sequences $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(\mathcal{X})$ as follows

(i) lift the map $\phi : \mathcal{X} \to V$ to a map $\varphi : \mathcal{X} \to \mathrm{T}(V)$,

(ii) map $\mathrm{Seq}(\mathcal{X}) \to \mathrm{Seq}(\mathrm{T}(V))$ by $(\mathbf{x}_1, \ldots, \mathbf{x}_L) \mapsto (\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_L))$,

(iii) map $\mathrm{Seq}(\mathrm{T}(V)) \to \mathrm{T}(V)$ by multiplication $(\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_L)) \mapsto \varphi(\mathbf{x}_1) \cdots \varphi(\mathbf{x}_L)$.

In a more concise form, we define Φ as

$$\Phi : \mathrm{Seq}(\mathcal{X}) \to \mathrm{T}(V), \quad \Phi(\mathbf{x}) = \prod_{i=1}^{L} \varphi(\mathbf{x}_i) \tag{2}$$

where $\prod$ denotes multiplication in $\mathrm{T}(V)$. We refer to the resulting map Φ as the Seq2Tens map, which stands short for ***Seq**uences-**2**-**Tens**ors*. Why is this construction a good idea? First note, that step (i) is always possible since $V \subset \mathrm{T}(V)$ and we discuss the simplest such lift before Theorem 2.1 as well as other choices in Appendix B. Further, if ϕ, respectively φ, provides a faithful representation of objects in $\mathcal{X}$, then there is no loss of information in step (ii). Finally, since step (iii) uses "the most general product" to multiply $\varphi(\mathbf{x}_1) \cdots \varphi(\mathbf{x}_L)$ one expects that $\Phi(\mathbf{x}) \in \mathrm{T}(V)$ faithfully represents the sequence $\mathbf{x}$ as an element of $\mathrm{T}(V)$.

Indeed in Theorem 2.1 below we show an even stronger statement, namely that if the static feature map $\phi : \mathcal{X} \to V$ contains enough non-linearities so that non-linear functions from $\mathcal{X}$ to $\mathbb{R}$ can be approximated as *linear functions* of the static feature map ϕ, then the above construction extends this property to functions of sequences. Put differently, *if ϕ is a universal feature map for $\mathcal{X}$, then Φ is a universal feature map for* $\mathrm{Seq}(\mathcal{X})$; that is, any non-linear function $f(\mathbf{x})$ of a sequence $\mathbf{x}$ can be approximated as a linear functional of $\Phi(\mathbf{x})$, $f(\mathbf{x}) \approx \langle \ell, \Phi(\mathbf{x}) \rangle$. We also emphasize that the domain of Φ is the space $\mathrm{Seq}(\mathcal{X})$ of sequences of *arbitrary* (finite) length. The remainder of this Section gives more details about steps (i),(ii),(iii) for the construction of Φ.

The free algebra $\mathrm{T}(V)$ over a vector space V. Let V be a vector space. We denote by $\mathrm{T}(V)$ the set of sequences of tensors indexed by their degree m,

$$\mathrm{T}(V) := \{\mathbf{t} = (\mathbf{t}_m)_{m \geq 0} \,|\, \mathbf{t}_m \in V^{\otimes m}\} \tag{3}$$

where by convention $V^{\otimes 0} = \mathbb{R}$. For example, if $V = \mathbb{R}^d$ and $\mathbf{t} = (\mathbf{t}_m)_{m \geq 0}$ is some element of $\mathrm{T}(\mathbb{R}^d)$, then its degree $m = 1$ component is a d-dimensional vector $\mathbf{t}_1$, its degree $m = 2$ component is a $d \times d$ matrix $\mathbf{t}_2$, and its degree $m = 3$ component is a degree 3 tensor $\mathbf{t}_3$. By defining addition and scalar multiplication as

$$\mathbf{s} + \mathbf{t} := (\mathbf{s}_m + \mathbf{t}_m)_{m \geq 0}, \quad c \cdot \mathbf{t} = (c\mathbf{t}_m)_{m \geq 0} \tag{4}$$

the set $\mathrm{T}(V)$ becomes a linear space. By identifying $v \in V$ as the element $(0, v, 0, \ldots, 0) \in \mathrm{T}(V)$ we see that V is a linear subspace of $\mathrm{T}(V)$. Moreover, while V is only a linear space, $\mathrm{T}(V)$ carries a product that turns $\mathrm{T}(V)$ into an algebra. This product is the so-called *tensor convolution product*, and is defined for $\mathbf{s}, \mathbf{t} \in \mathrm{T}(V)$ as

$$\mathbf{s} \cdot \mathbf{t} := \big(\sum_{i=0}^{m} \mathbf{s}_i \otimes \mathbf{t}_{m-i}\big)_{m \geq 0} = \big(1, \mathbf{s}_1 + \mathbf{t}_1, \mathbf{s}_2 + \mathbf{s}_1 \otimes \mathbf{t}_1 + \mathbf{t}_2, \ldots\big) \in \mathrm{T}(V) \tag{5}$$

where $\otimes$ denotes the usual outer tensor product; e.g. for vectors $u = (u_i), v = (v_i) \in \mathbb{R}^d$ the outer tensor product $u \otimes v$ is the $d \times d$ matrix $(u_i v_j)_{i,j=1,\ldots,d}$. We emphasize that like the outer tensor product $\otimes$, the tensor convolution product $\cdot$ is non-commutative, i.e. $\mathbf{s} \cdot \mathbf{t} \neq \mathbf{t} \cdot \mathbf{s}$. In a mathematically precise sense, $\mathrm{T}(V)$ *is the most general algebra that contains V; it is a "free construction"*. Since $\mathrm{T}(V)$ is realized as series of tensors of increasing degree, the *free algebra* $\mathrm{T}(V)$ is also known as the *tensor algebra* in the literature. Appendix A contains background on tensors and further examples.

Lifting static feature maps. Step (i) in the construction of Φ requires turning a given feature map $\phi : \mathcal{X} \to V$ into a map $\varphi : \mathcal{X} \to \mathrm{T}(V)$. Throughout the rest of this article we use the lift

$$\varphi(\mathbf{x}) = (1, \phi(\mathbf{x}), 0, 0 \ldots) \in \mathrm{T}(V). \tag{6}$$

We discuss other choices in Appendix B, but attractive properties of the lift 6 are that (a) the evaluation of Φ against low rank tensors becomes a simple recursive formula (Proposition 3.3, (b) it is a generalization of sequence sub-pattern matching as used in string kernels (Appendix B.3, (c) despite its simplicity it performs exceedingly well in practice (Section 4).

Extending to sequences of arbitrary length. So far we have specified in steps (i) and (ii) how the map $\Phi : \mathcal{X} \to \mathrm{T}(V)$ behaves on single observations. Step (iii) in the construction amounts to the requirement that for any two sequences $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_K), \mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_L) \in \mathrm{Seq}(V)$, their concatenation defined as $\mathbf{z} = (\mathbf{x}_1, \ldots, \mathbf{x}_K, \mathbf{y}_1, \ldots, \mathbf{y}_L) \in \mathrm{Seq}(V)$ can be understood in the feature space as (non-commutative) multiplication of their corresponding features

$$\Phi(\mathbf{z}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}). \tag{7}$$

Using equation 7 we may inductively extend the lift φ to sequences of arbitrary length by starting from sequences consisting of a single observation, which is equivalent to equation 2. By repeatedly using the definition of the tensor convolution product in equation 5 this may also be written as

$$\Phi_m(\mathbf{x}) = \sum_{1 \leq i_1 < \cdots < i_m \leq L} \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_m} \in V^{\otimes m}, \;\; \Phi(\mathbf{x}) = (\Phi_m(\mathbf{x}))_{m \geq 0}, \tag{8}$$

where $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(V)$ and the summation is over non-contiguous subsequences of $\mathbf{x}$.

Some intuition: generalized pattern matching. Our derivation of the feature map $\Phi(\mathbf{x}) = (1, \Phi_1(\mathbf{x}), \Phi_2(\mathbf{x}), \ldots) \in \mathrm{T}(V)$ was guided by general algebraic principles, but equation 8 provides an intuitive interpretation. It shows that for each $m \geq 1$, the entry $\Phi_m(\mathbf{x}) \in V^{\otimes m}$ constructs a summary of a long sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(V)$ based on subsequences $(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m})$ of $\mathbf{x}$ of length m. It does this by taking the usual outer tensor product $\mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_m} \in V^{\otimes m}$ and summing over all possible subsequences. This is completely analogous to how string kernels provide a structured description of text by looking at non-contiguous substrings of length m (indeed, Appendix B.3 makes this rigorous). However, the main difference is that the above construction works for arbitrary sequences and not just sequences of discrete letters. Readers with less mathematical background might simply take this as motivation and regard equation 8 as definition. However, the algebraic background allows to prove that Φ is universal, see Theorem 2.1 below.

Universality. A function $\phi : \mathcal{X} \to V$ is said to be *universal for* $\mathcal{X}$ if all continuous functions on $\mathcal{X}$ can be approximated as linear functions on the image of ϕ. One of the most powerful features of neural nets is their universality (Hornik, 1991). A very attractive property of Φ is that it preserves universality: if $\phi : \mathcal{X} \to V$ is universal for $\mathcal{X}$, then $\Phi : \mathrm{Seq}(X) \to \mathrm{T}(V)$ is universal for $\mathrm{Seq}(\mathcal{X})$. To make this precise, note that $V^{\otimes m}$ is a linear space and therefore any $\ell = (\ell_0, \ell_1, \ldots, \ell_M, 0, 0, \ldots) \in$

$\mathrm{T}(V)$ consisting of M tensors $\ell_m \in V^{\otimes m}$, yields a linear functional on $\mathrm{T}(V)$; e.g. if $V = \mathbb{R}^d$ and we identify ℓ_m in coordinates as $\ell_m = (\ell_m^{i_1,\ldots,i_m})_{i_1,\ldots,i_m \in \{1,\ldots,d\}}$ then

$$\langle \ell, \mathbf{t} \rangle := \sum_{m=0}^{M} \langle \ell_m, \mathbf{t}_m \rangle = \sum_{m=0}^{M} \sum_{i_1,\ldots,i_m \in \{1,\ldots,d\}} \ell_m^{i_1,\ldots,i_m} \mathbf{t}_m^{i_1,\ldots,i_m}. \tag{9}$$

Thus linear functionals of the feature map Φ, are real-valued functions of sequences. Theorem 2.1 below shows that any continuous function $f : \mathrm{Seq}(\mathcal{X}) \to \mathbb{R}$ can by arbitrary well approximated by a $\ell \in \mathrm{T}(V)$, $f(\mathbf{x}) \approx \langle \ell, \Phi(\mathbf{x}) \rangle$.

Theorem 2.1. *Let $\phi : \mathcal{X} \to V$ be a universal map with a lift that satisfies some mild constraints, then the following map is universal:*

$$\Phi : \mathrm{Seq}(\mathcal{X}) \to \mathrm{T}(V), \quad \mathbf{x} \mapsto \Phi(\mathbf{x}). \tag{10}$$

A detailed proof and the precise statement of Theorem 2.1 is given in Appendix B.

3 Approximation by low-rank linear functionals

The combinatorial explosion of tensor coordinates and what to do about it. The universality of Φ suggests the following approach to represent a function $f : \mathrm{Seq}(\mathcal{X}) \to \mathbb{R}$ of sequences: First compute $\Phi(\mathbf{x})$ and then optimize over ℓ (and possibly also the hyperparameters of ϕ) such that $f(\mathbf{x}) \approx \langle \ell, \Phi(\mathbf{x}) \rangle = \sum_{m=0}^{M} \langle \ell_m, \Phi_m(\mathbf{x}) \rangle$. Unfortunately, tensors suffer from a combinatorial explosion in complexity in the sense that even just storing $\Phi_m(\mathbf{x}) \in V^{\otimes m} \subset \mathrm{T}(V)$ requires $O(\dim(V)^m)$ real numbers. Below we resolve this computational bottleneck as follows: in Proposition 3.3 we show that for a special class of low-rank elements $\ell \in \mathrm{T}(V)$, the functional $\mathbf{x} \mapsto \langle \ell, \Phi(\mathbf{x}) \rangle$ can be efficiently computed in both time and memory. This is somewhat analogous to a kernel trick since it shows that $\langle \ell, \Phi(\mathbf{x}) \rangle$ can be cheaply computed without explicitly computing the feature map $\Phi(\mathbf{x})$. However, Theorem 2.1 guarantees universality under no restriction on ℓ, thus restriction to rank-1 functionals limits the class of functions $f(\mathbf{x})$ that can be approximated. Nevertheless, by iterating these "low-rank functional" constructions in the form of sequence-to-sequence transformations this can be ameliorated. We give the details below but to gain intuition, we invite the reader to think of this iteration analogous to stacking layers in neural network: each layer is a relatively simple non-linearity (e.g. a sigmoid composed with an affine function) but by composing such layers, complicated functions can be efficiently approximated.

Rank-1 functionals are computationally cheap. Degree $m = 2$ tensors are matrices and LR approximations of matrices are widely used in practice (Udell & Townsend, 2019) to address the quadratic complexity. The definition below generalizes the rank of matrices (tensors of degree $m = 2$) to tensors of any degree m.

Definition 3.1. The *rank* (also called *CP rank* (Carroll & Chang, 1970)) of a degree m tensor $\mathbf{t}_m \in V^{\otimes m}$ is the smallest number $r \geq 0$ such that one may write

$$\mathbf{t}_m = \sum_{i=0}^{r} \mathbf{v}_i^1 \otimes \cdots \otimes \mathbf{v}_i^m, \quad \mathbf{v}_i^1, \ldots, \mathbf{v}_i^m \in V. \tag{11}$$

We say that $\mathbf{t} = (\mathbf{t}_m)_{m \geq 0} \in \mathrm{T}(V)$ has rank-1 (and degree M) if each $\mathbf{t}_m \in V^{\otimes m}$ is a rank-1 tensor and $\mathbf{t}_i = 0$ for $i > M$.

Remark 3.2. For $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(V)$, the rank $r_m \in \mathbb{N}$ of $\Phi_m(\mathbf{x})$ satisfies $r_m \leq \binom{L}{m}$, while the rank and degree $r, d \in \mathbb{N}$ of $\Phi(\mathbf{x})$ satisfy $r \leq \binom{L}{K}$ for $K = \lfloor \frac{L}{2} \rfloor$ and $d \leq L$.

A direct calculation shows that if ℓ is of rank-1, then $\langle \ell, \Phi(\mathbf{x}) \rangle$ can be computed very efficiently by inner product evaluations in V.

Proposition 3.3. *Let $\ell = (\ell_m)_{m \geq 0} \in \mathrm{T}(V)$ be of rank-1 and degree M. If ϕ is lifted to φ as in equation 6, then*

$$\langle \ell, \Phi(\mathbf{x}) \rangle = \sum_{m=0}^{M} \sum_{1 \leq i_1 < \cdots < i_m \leq L} \prod_{k=1}^{m} \langle \mathbf{v}_k^m, \phi(\mathbf{x}_{i_k}) \rangle \tag{12}$$

where $\ell_m = \mathbf{v}_1^m \otimes \cdots \otimes \mathbf{v}_m^m \in V^{\otimes m}$, $\mathbf{v}_i^m \in V$ and $m = 0, \ldots, M$.

Note that the inner sum is taken over all non-contiguous subsequences of $\mathbf{x}$ of length m, analogous to m-mers of strings and we make this connection precise in Appendix B.3; the proof of Proposition 3.3 is given in Appendix B.1.1. While equation 12 looks expensive, by casting it into a recursive formulation over time, it can be computed in $O(M^2 \cdot L \cdot d)$ time and $O(M^2 \cdot (L+c))$ memory, where d is the inner product evaluation time on V, while c is the memory footprint of a $v \in V$. This can further be reduced to $O(M \cdot L \cdot d)$ time and $O(M \cdot (L+c))$ memory by an efficient parametrization of the rank-1 element $\ell \in \mathrm{T}(V)$. We give further details in Appendices D.2, D.3, D.4.

Low-rank Seq2Tens maps. The composition of a linear map $\mathcal{L} : \mathrm{T}(V) \to \mathbb{R}^N$ with Φ can be computed cheaply in parallel using equation 12 when $\mathcal{L}$ is specified through a collection of $N \in \mathbb{N}$ rank-1 elements $\ell^1, \ldots, \ell^N \in \mathrm{T}(V)$ such that

$$\tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1, \ldots, \mathbf{x}_L) := \mathcal{L} \circ \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L) = (\langle \ell^j, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)\rangle)_{j=1}^N \in \mathbb{R}^N. \tag{13}$$

We call the resulting map $\tilde{\Phi}_{\tilde{\theta}} : \mathrm{Seq}(\mathcal{X}) \to \mathbb{R}^N$ a ***Low-rank Seq2Tens*** map of width-N and order-M, where $M \in \mathbb{N}$ is the maximal degree of $\ell^1, \ldots, \ell^N$ such that $\ell^j_i = 0$ for $i > M$. The LS2T map is parametrized by (1) the component vectors $\mathbf{v}^k_{j,m} \in V$ of the rank-1 elements $\ell^j_m = \mathbf{v}^1_{j,m} \otimes \cdots \otimes \mathbf{v}^m_{j,m}$, (2) by any parameters θ that the static feature map $\phi_\theta : \mathcal{X} \to V$ may depend on. We jointly denote these parameters by $\tilde{\theta} = (\theta, \ell^1, \ldots, \ell^N)$. In addition, by the subsequent composition of $\tilde{\Phi}_{\tilde{\theta}}$ with a linear functional $\mathbb{R}^N \to \mathbb{R}$, we get the following function subspace as hypothesis class for the LS2T

$$\tilde{\mathcal{H}} = \big\{ \langle \sum_{j=1}^N \alpha_j \ell^j, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)\rangle \,|\, \alpha_j \in \mathbb{R} \big\} \subsetneq \mathcal{H} = \big\{ \langle \ell, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L) \,|\, \ell \in \mathrm{T}(V) \big\} \tag{14}$$

Hence, we acquire an intuitive explanation of the (hyper)parameters: the width of the LS2T, $N \in \mathbb{N}$ specifies the maximal rank of the low-rank linear functionals of Φ that the LS2T can represent, while the span of the rank-1 elements, $\mathrm{span}(\ell^1, \ldots, \ell^N)$ determine an N-dimensional subspace of the dual space of $\mathrm{T}(V)$ consisting of at most rank-N functionals.

Recall now that without rank restrictions on the linear functionals of Seq2Tens features, Theorem 2.1 would guarantee that any real-valued function $f : \mathrm{Seq}(\mathcal{X}) \to \mathbb{R}$ could be approximated by $f(\mathbf{x}) \approx \langle \ell, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)\rangle$. As pointed out before, the restriction of the hypothesis class to low-rank linear functionals of $\Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)$ would limit the class of functions of sequences that can be approximated. To ameliorate this, we propose to use LS2T transforms in a sequence-to-sequence fashion that allows us to stack such low-rank functionals, significantly recovering expressiveness.

Sequence-to-sequence transforms. We can use LS2T to build sequence-to-sequence transformations in the following way: fix the static map $\phi_\theta : \mathcal{X} \to V$ parametrized by θ and rank-1 elements such that $\tilde{\theta} = (\theta, \ell^1, \ldots, \ell^N)$ and apply the resulting LS2T map $\tilde{\Phi}_{\tilde{\theta}}$ over expanding windows of $\mathbf{x}$:

$$\mathrm{Seq}(\mathcal{X}) \to \mathrm{Seq}(\mathbb{R}^N), \quad \mathbf{x} \mapsto \big(\tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1), \tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1, \mathbf{x}_2), \ldots, \tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1, \ldots, \mathbf{x}_L)\big). \tag{15}$$

The cost of computing the expanding window sequence-to-sequence transform in equation 15 is no more expensive than computing $\tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1, \ldots, \mathbf{x}_L)$ itself due to the recursive nature of our algorithms.

Stacking sequence-to-sequence transforms. Inspired by the empirical successes of stacked RNNs (Graves et al., 2013b;a; Sutskever et al., 2014), we iterate the transformation 15 D-times:

$$\mathrm{Seq}(\mathcal{X}) \to \mathrm{Seq}(\mathbb{R}^{N_1}) \to \mathrm{Seq}(\mathbb{R}^{N_2}) \to \cdots \to \mathrm{Seq}(\mathbb{R}^{N_D}). \tag{16}$$

Each of these mappings $\mathrm{Seq}(\mathbb{R}^{N_i}) \to \mathrm{Seq}(\mathbb{R}^{N_{i+1}})$ is parametrized by the parameters $\tilde{\theta}_i$ of a static feature map ϕ_{θ_i} and a linear map $\mathcal{L}_i$ specified by N_i rank-1 elements of $\mathrm{T}(V)$; these parameters are collectively denoted by $\tilde{\theta}_i = (\theta_i, \ell^1_i, \ldots, \ell^{N_i}_i)$. Evaluating the final sequence in $\mathrm{Seq}(\mathbb{R}^{N_D})$ at last observation-time $t = L$, we get the D-times stacked LS2T map

$$\tilde{\Phi}_{\tilde{\theta}_1, \ldots, \tilde{\theta}_D} : \mathrm{Seq}(\mathcal{X}) \to \mathbb{R}^{n_D}. \tag{17}$$

Making precise how the stacking of such low-rank sequence-to-sequence transformations approximates general functions requires more tools from algebra, and we provide a rigorous quantitative statement in Appendix C. Here, we just appeal to the analogy made with adding depth in neural networks mentioned earlier and empirically validate this in our experiments in Section 4.

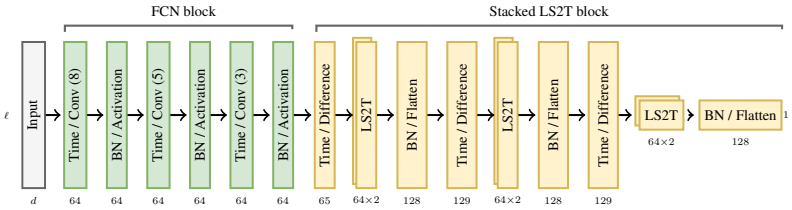

Figure 1: Depiction of the proposed models for time series classification. LS2T^3 only consists of a stacked Seq2Tens block (yellow), while FCN-LS2T^3 also precedes it with an FCN block (green).

4 Building neural networks with LS2T layers

The Seq2Tens map Φ built from a static feature map ϕ is universal if ϕ is universal, Theorem 2.1. NNs form a flexible class of universal feature maps with strong empirical success for data in $\mathcal{X} = \mathbb{R}^d$, and thus make a natural choice for ϕ. Combined with standard deep learning constructions, the framework of Sections 2 and 3 can build modular and expressive layers for sequence learning.

Neural LS2T layers. The simplest choice among many is to use as static feature map $\phi : \mathcal{X} = \mathbb{R}^d \to \mathbb{R}^M$ a feedforward network with depth-P, $\phi = \phi_P \circ \cdots \circ \phi_1$ where $\phi_j(\mathbf{x}) = \sigma(\mathbf{W}_j \mathbf{x} + \mathbf{b}_j)$ for $\mathbf{W}_j \in \mathbb{R}^{M \times d}$, $\mathbf{b}_j \in \mathbb{R}^M$. We can then lift this to a map $\varphi : \mathbb{R}^d \to \mathrm{T}(\mathbb{R}^M)$ as prescribed in equation 6. Hence, the resulting LS2T layer $\mathbf{x} \mapsto (\tilde{\Phi}_{\tilde{\theta}}(\mathbf{x}_1, \ldots, \mathbf{x}_i))_{i=1,\ldots,L}$ is a sequence-to-sequence transform $\mathrm{Seq}(\mathbb{R}^d) \to \mathrm{Seq}(\mathbb{R}^M)$ that is parametrized by $\tilde{\theta} = (\mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_P, \mathbf{b}_P, \ell_1^1, \ldots, \ell_1^{N_1})$.

Bidirectional LS2T layers. The transformation in equation 15 is completely causal in the sense that each step of the output sequence depends only on past information. For generative models, it can behove us to make the output depend on both past and future information, see Graves et al. (2013a); Baldi et al. (1999); Li & Mandt (2018). Similarly to bidirectional RNNs and LSTMs (Schuster & Paliwal, 1997; Graves & Schmidhuber, 2005), we may achieve this by defining a bidirectional layer,

$$\tilde{\Phi}^{\mathrm{b}}_{(\tilde{\theta}_1, \tilde{\theta}_2)}(\mathbf{x}) : \mathrm{Seq}(\mathbb{R}^d) \to \mathrm{Seq}(\mathbb{R}^{N+N'}), \quad \mathbf{x} \mapsto (\tilde{\Phi}_{\tilde{\theta}_1}(\mathbf{x}_1, \ldots, \mathbf{x}_i), \tilde{\Phi}_{\tilde{\theta}_2}(\mathbf{x}_i, \ldots, \mathbf{x}_L))_{i=1}^L. \tag{18}$$

The sequential nature is kept intact by making the distinction between what classifies as past (the first N coordinates) and future (the last N' coordinates) information. This amounts to having a form of precognition in the model, and has been applied in e.g. dynamics generation (Li & Mandt, 2018), machine translation (Sundermeyer et al., 2014), and speech processing (Graves et al., 2013a).

Convolutions and LS2T. We motivate to replace the time-distributed feedforward layers proposed in the paragraph above by temporal convolutions (CNN) instead. Although theory only requires the preprocessing layer of the LS2T to be a state-space nonlinearity, we find that it is beneficial to capture some of the sequential information in the preprocessing layer as well, e.g. using CNNs or RNNs. From a mathematical point of view, CNNs are a straightforward extension since they can be interpreted as time-distributed feedforward layers applied to the input sequence augmented with a $p \in \mathbb{N}$ number of its lags for CNN kernel size p (see Appendix D.1 for further discussion).

In the following, we precede our stacked LS2T blocks by one or more CNN layers. Intuitively, CNNs and LS2Ts are similar in that both transformations operate on subsequences of their input sequence. The main difference between the two lies in that *CNNs operate on contiguous subsequences*, and therefore, capture local, short-range nonlinear interactions between timesteps; *while LS2Ts (equation 12) use all non-contiguous subsequences*, and hence, learn global, long-range interactions in time. This observation motivates that the inductive biases of the two types of layers (local/global time-interactions) are highly complementary in nature, and we propose that the improvement in the experiments on the models containing vanilla CNN blocks are due to this complementarity.

5 Experiments

We demonstrate the modularity and flexibility of the above LS2T and its variants by applying it to (i) multivariate time series classification, (ii) mortality prediction in healthcare, (iii) generative modelling of sequential data. In all cases, we take a strong baseline model (FCN and GP-VAE, as detailed below) and upgrade it with LS2T layers. As Thm. 2.1 requires the Seq2Tens to be preceded by at

Table 1: Posterior probabilities given by a Bayesian signed-rank test comparison of the proposed methods against the baselines. $\{>\}$, $\{<\}$, $\{=\}$ refer to the respective events that the row method is better, the column method is better, or that they are equivalent.

MODEL	LS2T^3			FCN-LS2T^3		
	$p(>)$	$p(=)$	$p(<)$	$p(>)$	$p(=)$	$p(<)$
LS2T^3	–	–	–	0.000	0.017	**0.983**
SMTS (BAYDOGAN & RUNGER, 2015A)	0.369	0.000	**0.631**	0.005	0.000	**0.995**
LPS (BAYDOGAN & RUNGER, 2015B)	0.477	0.002	**0.520**	0.001	0.000	**0.999**
MVARF (TUNCEL & BAYDOGAN, 2018)	0.021	0.168	**0.811**	0.000	0.089	**0.911**
DTW (SAKOE & CHIBA, 1978)	0.086	0.000	**0.914**	0.000	0.000	**1.000**
ARKERNEL (CUTURI & DOUCET, 2011)	0.136	0.104	**0.760**	0.000	0.044	**0.955**
GRSF (KARLSSON ET AL., 2016)	**0.851**	0.008	0.141	0.063	0.024	**0.913**
MLSTMFCN (KARIM ET AL., 2019)	**0.947**	0.038	0.014	0.342	0.190	**0.469**
MUSE (SCHÄFER & LESER, 2017)	**0.606**	0.194	0.200	0.021	0.197	**0.781**
FCN (WANG ET AL., 2017)	**0.971**	0.000	0.029	0.176	0.033	**0.791**
RESNET (WANG ET AL., 2017)	**0.962**	0.000	0.038	0.122	0.000	**0.878**

least a universal state-space nonlinearity, we expect these layers to perform best as an add-on on top of other models, which however can be quite simple, such as an FCN. The additional computation time is negligible (in fact, for FCN it allows us to reduce the number of parameters significantly), but it can yield substantial improvements. This is remarkable, since the results of the original models are already state-of-the-art on well-established (frequentist and Bayesian) benchmarks.

5.1 MULTIVARIATE TIME SERIES CLASSIFICATION

First, we consider multivariate time series classification (TSC) on a semi-standardized collection of benchmark datasets unified in Baydogan (2015). A wide range of previous publications report results on this archive, which makes it possible to compare against a wide range of baseline methods without bias in parameter settings. A subset of this archive was also considered in a recent, highly cited review paper on deep learning (DL) for TSC (Ismail Fawaz et al., 2019) from where we also borrow the best performing models as baselines. We provide further details on the experiment, baselines and datasets in Appendix E.1. As our own models, we introduce two simple architectures that utilize LS2T layers: (i) LS2T^3 stacks 3 LS2T layers of order 2 and width 64; (ii) FCN-LS2T^3 precedes the LS2T^3 block by a CNN block of 3 convolutional layers of width 64 and filter sizes of $8, 5, 3$; a downsized version of the FCN model of (Wang et al., 2017) with time-embeddings (Liu et al., 2018a) and without a final global average pooling (GAP) layer, see Fig. 1. The hyperparameters, i.e. CNN/LS2T width $N = 64$, LS2T order $M = 2$ and LS2T depth $D = 3$ were tuned using a small subset of the datasets from $N \in \{16, 64, 128\}$, $M \in \{2, 3, 4\}$ and $D \in \{1, 2, 3\}$.

Results. We trained the introduced models on each dataset 5 times while the baseline results were borrowed from their respective publications. In Appendix E.1, Figure 4 depicts the results as a box-plot of distributions and a CD diagram using the Nemenyi test (Nemenyi, 1963), while Table 7 shows the full list of means and standard deviations. Since mean-ranks based tests raise some paradoxical issues (Benavoli et al., 2016), it is customary to conduct pairwise comparisons using frequentist (Demšar, 2006) or Bayesian (Benavoli et al., 2017) hypothesis tests. We adopted the Bayesian signed-rank test from Benavoli et al. (2014), which is showcased in Table 1, while the Bayesian posteriors are visualized on Figure 5 in App. E.1. The results of the signed-rank test indicate that while LS2T^3 is better with moderate probability ($p \geq 0.6$) than 4 of the baselines, FCN-LS2T^3 performs better than all with high probability ($p \geq 0.75$), except for MLSTMFCN which is only slightly outperformed by FCN-LS2T^3. Note however that *FCN-LS2T^3 has fewer parameters than MLSTMFCN by more than* 60%, see Table 6 for a parameter comparison. Also, as MLSTMFCN is the concatenation of an FCN with an LSTM (Karim et al., 2019), the LSTM layer can bottleneck the computations for very long sequences as it scales linearly, while *LS2T layers effectively scale sublinearly in sequence length*, Appendix D.4 for a computational comparison. In conclusion, we have verified that LS2T^3 already performs well on some classification tasks, and by preceding it with an FCN, its performance is elevated to outperforming all baseline models. This observation is not surprising, as Thm. 2.1 warrants, the full strength of the LS2T layer shines when it is preceded by at least some state-space nonlinearities. Hence, we expect LS2Ts to find their best use as a modular

building block being part of larger models. We also suggest the improvement on the vanilla FCN composed of CNN blocks is due to the LS2T enhancing the model's ability to pick up long-range interactions in time as previously discussed at the end of Section 4.

5.2 MORTALITY PREDICTION

We consider the 2012 PhysioNet challenge dataset (Goldberger et al., 2000) for mortality prediction, that consists of medical time series of 12,000 ICU stays over at least 48 hours. Overall, 6 static features were recorded and potentially up to 37 TS values were measured both irregularly and asynchronously for each stay with certain dimensions completely missing in some cases. The task is to predict whether a patient died during their hospital stay. This is a trickier dataset due to missingness, low signal-to-noise ratio, and unbalanced class distributions with a prevalence ratio of around 14%.

We extend the experiments conducted in Horn et al. (2020) containing very strong baselines, and under the same experimental setting we train on this dataset our model FCN-LS2T[3] and the FCN as control. We preprocess the data using the method introduced in Che et al. (2018, eq. (9)), which includes adding the observation mask and time since last measurement taken as extra TS dimensions, while we handle static features by tiling them along the time axis and also adding them as extra dims. For the models FCN-LS2T[3] and FCN, the same hyperparameters are used as in the TSC experiment with the additional introduction of a `SpatialDropout1D` layer following each CNN and LS2T layer using a dropout rate of $r = 0.1$ in order to deal with the low SNR of the dataset.

Results. Table 2 compares the results of our model, FCN-LS2T[3] with that of FCN and the baselines from Horn et al. (2020) through 3 metrics: (1) classification accuracy, (2) area under the precision-recall curve (AUPRC), (3) area under the ROC curve (AUROC). We observe that FCN-LS2T[3] significantly outperforms FCN according to all metrics. FCN-LS2T[3] is also competitive with the SOTA methods, such as the TRANSFORMER (Vaswani et al., 2017), SEFT (Horn et al., 2020), IP-NETS (Shukla & Marlin, 2019) and GRU-D (Che et al., 2018), achieving the best accuracy and taking fourth place with respect to both AUPRC and AUROC.

Table 2: Comparison of methods on mortality prediction with baselines lifted from Horn et al. (2020).

MODEL	ACCURACY	AUPRC	AUROC
FCN-LS2T[3]	$\mathbf{85.9 \pm 0.6}$	51.8 ± 0.7	85.1 ± 0.6
FCN	79.4 ± 0.6	50.2 ± 0.5	84.5 ± 0.4
GRU-D	80.0 ± 2.9	$\mathbf{53.7 \pm 0.9}$	$\mathbf{86.3 \pm 0.3}$
GRU-SIMPLE	82.2 ± 0.2	42.2 ± 0.6	80.8 ± 1.1
IP-NETS	79.4 ± 0.3	51.0 ± 0.6	$\mathit{86.0 \pm 0.2}$
PHASED-LSTM	76.8 ± 5.2	38.7 ± 1.5	79.0 ± 1.0
TRANSFORMER	$\mathit{83.7 \pm 3.5}$	$\mathit{52.8 \pm 2.2}$	$\mathbf{86.3 \pm 0.8}$
LATENT-ODE	76.0 ± 0.1	50.7 ± 1.7	85.7 ± 0.6
SEFT-ATTN.	75.3 ± 3.5	52.4 ± 1.1	85.1 ± 0.4

5.3 GENERATING SEQUENTIAL DATA

As another benchmarking experiment, we demonstrate that LS2T layers do not only provide good representations of sequences in discriminative, but also generative models. The considered problem is sequential data imputation for time series and video.

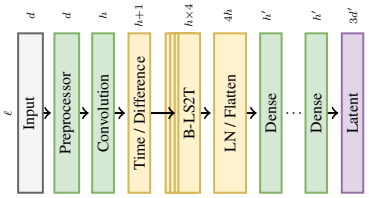

Figure 2: Encoder in GP-VAE (B-LS2T).

The GP-VAE model. In this experiment, we take as base model the recent GP-VAE (Fortuin et al., 2020), that provides state-of-the-art results for probabilistic sequential data imputation. The GP-VAE is essentially based on the HI-VAE (Nazabal et al., 2018) for handling missing data in variational autoencoders (VAEs) (Kingma & Welling, 2013) adapted to the handling of time series data by the use of a Gaussian process (GP) prior (Williams & Rasmussen, 2006) across time in the latent sequence space to capture temporal dynamics. Since the GP-VAE is a highly advanced model, its in-depth description is deferred to Appendix E.2.We extend the experiments conducted in Fortuin et al. (2020), and we make one simple change to the GP-VAE architecture without changing any other hyperparameters or aspects: we introduce a single bidirectional LS2T layer (B-LS2T) into the encoder network that is used in the amortized representation of the means and covariances of the variational posterior. The B-LS2T layer is preceded by a time-embedding and differencing block, and succeeded by channel flattening and layer normalization as depicted in Figure 2. The idea behind this experiment is to see if we can improve the performance of a highly complicated model that is composed of many interacting submodels, by the naive introduction of LS2T layers.

Table 3: Performance comparison of GP-VAE (B-LS2T) with the baseline methods

METHOD	HMNIST			SPRITES	PHYSIONET
	NLL	MSE	AUROC	MSE	AUROC
MEAN IMPUTATION	-	0.168 ± 0.000	0.938 ± 0.000	0.013 ± 0.000	0.703 ± 0.000
FORWARD IMPUTATION	-	0.177 ± 0.000	0.935 ± 0.000	0.028 ± 0.000	0.710 ± 0.000
VAE	0.599 ± 0.002	0.232 ± 0.000	0.922 ± 0.000	0.028 ± 0.000	0.677 ± 0.002
HI-VAE	0.372 ± 0.008	0.134 ± 0.003	$\mathbf{0.962 \pm 0.001}$	0.007 ± 0.000	0.686 ± 0.010
GP-VAE	0.350 ± 0.007	0.114 ± 0.002	$\mathbf{0.960 \pm 0.002}$	$\mathbf{0.002 \pm 0.000}$	0.730 ± 0.006
GP-VAE (B-LS2T)	$\mathbf{0.251 \pm 0.008}$	$\mathbf{0.092 \pm 0.003}$	$\mathbf{0.962 \pm 0.001}$	$\mathbf{0.002 \pm 0.000}$	$\mathbf{0.743 \pm 0.007}$
BRITS	-	-	-	-	$\mathbf{0.742 \pm 0.008}$

Results. To make the comparison, we ceteris paribus re-ran all experiments the authors originally included in their paper (Fortuin et al., 2020), which are imputation of Healing MNIST, Sprites, and Physionet 2012. The results are in Table 3, which report the same metrics as used in Fortuin et al. (2020), i.e. negative log-likelihood (NLL, lower is better), mean squared error (MSE, lower is better) on test sets, and downstream classification performance of a linear classifier (AUROC, higher is better). For all other models beside our GP-VAE (B-LS2T), the results were borrowed from Fortuin et al. (2020). We observe that simply adding the B-LS2T layer improved the result in almost all cases, except for Sprites, where the GP-VAE already achieved a very low MSE score. Additionally, when comparing GP-VAE to BRITS on Physionet, the authors argue that although the BRITS achieves a higher AUROC score, the GP-VAE should not be disregarded as it fits a generative model to the data that enjoys the usual Bayesian benefits of predicting distributions instead of point predictions. The results display that by simply adding our layer into the architecture, we managed to elevate the performance of GP-VAE to the same level while retaining these same benefits. We believe the reason for the improvement is a tighter amortization gap in the variational approximation (Cremer et al., 2018) achieved by increasing the expressiveness of the encoder by the LS2T allowing it to pick up on long-range interactions in time. We provide further discussion in Appendix E.2.

6 RELATED WORK AND SUMMARY

Related Work. The literature on tensor models in ML is vast. Related to our approach we mention pars-pro-toto Tensor Networks (Cichocki et al., 2016), that use classical LR decompositions, such as CP (Carroll & Chang, 1970), Tucker (Tucker, 1966), tensor trains (Oseledets, 2011) and tensor rings (Zhao et al., 2019); further, CNNs have been combined with LR tensor techniques (Cohen et al., 2016; Kossaifi et al., 2017) and extended to RNNs (Khrulkov et al., 2019); Tensor Fusion Networks (Zadeh et al., 2017) and its LR variants (Liu et al., 2018b; Liang et al., 2019; Hou et al., 2019); tensor-based gait recognition (Tao et al., 2007). Our main contribution to this literature is the use of the free algebra $\mathrm{T}(V)$ with its convolution product $\cdot$, instead of $V^{\otimes m}$ with the outer product $\otimes$ that is used in the above papers. While counter-intuitive to work in a larger space $\mathrm{T}(V)$, the additional algebra structure of $(\mathrm{T}(V), \cdot)$ is the main reason for the nice properties of Φ (*universality, making sequences of arbitrary length comparable, convergence in the continuous time limit*; see Appendix B) which we believe are in turn the main reason for the *strong benchmark performance*. Stacked LR sequence transforms allow to exploit this rich algebraic structure with little computational overhead. Another related literature are path signatures in ML (Lyons, 2014; Chevyrev & Kormilitzin, 2016; Graham, 2013; Bonnier et al., 2019; Toth & Oberhauser, 2020). These arise as special case of Seq2Tens (Appendix B) and our main contribution to this literature is that Seq2Tens resolves a well-known computational bottleneck in this literature since it *never needs to compute and store a signature*, instead it *directly and efficiently learns the functional of the signature*.

Summary. We used a classical non-commutative structure to construct a feature map for sequences of arbitrary length. By stacking sequence transforms we turned this into scalable and modular NN layers for sequence data. The main novelty is the use of the free algebra $\mathrm{T}(V)$ constructed from the static feature space V. While free algebras are classical in mathematics, their use in ML seems novel and underexplored. We would like to re-emphasize that $(\mathrm{T}(V), \cdot)$ is not a mysterious abstract space: if you know the outer tensor product $\otimes$ then you can easily switch to the tensor convolution product $\cdot$ by taking sums of outer tensor products, as defined in equation 5. As our experiments show, the benefits of this algebraic structure are not just theoretical but can significantly elevate performance of already strong-performing models.

REFERENCES

Pierre Baldi, Søren Brunak, Paolo Frasconi, Giovanni Soda, and Gianluca Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11):937–946, 1999.

Robert Bamler and Stephan Mandt. Dynamic word embeddings. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 380–389. JMLR. org, 2017.

Mustafa Baydogan. Multivariate time series classification datasets. `http://mustafabaydogan.com`, 2015. [Accessed: 2020-06-11].

Mustafa Gokce Baydogan and George Runger. Learning a symbolic representation for multivariate time series classification. *Data Mining and Knowledge Discovery*, 29(2):400–422, 2015a.

Mustafa Gokce Baydogan and George C. Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30:476–509, 2015b.

Alessio Benavoli, Giorgio Corani, Francesca Mangili, Marco Zaffalon, and Fabrizio Ruggeri. A bayesian wilcoxon signed-rank test based on the dirichlet process. In *International conference on machine learning*, pp. 1026–1034, 2014.

Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, January 2016. ISSN 1532-4435.

Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

David M Blei and John D Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pp. 113–120, 2006.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

P Bonnier, C Liu, and H Oberhauser. Adapted topologies and higher rank signatures. *arXiv preprint arXiv:2005.08897*, 2020.

Patric Bonnier, Patrick Kidger, Imanol Perez Arribas, Cristopher Salvi, and Terry Lyons. Deep signature transforms. *33rd Conference on Neural Information Processing Systems, NeurIPS*, 2019.

Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6775–6785. Curran Associates, Inc., 2018.

J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

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.

K. T. Chen. Iterated integrals and exponential homomorphisms. *Proc. London Math. Soc, 4, 502–512*, 1954.

K. T. Chen. Integration of paths, geometric invariants and a generalized Baker-Hausdorff formula. *Ann. of Math. (2), 65:163–178*, 1957.

K. T. Chen. Integration of paths - a faithful representation of paths by non-commutative formal power series. *Trans. Amer. Math. Soc. 89 (1958), 395–407*, 1958.

I. Chevyrev and A. Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.

Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on learning theory*, pp. 698–728, 2016.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1078–1086, 2018.

N Cristianini and J Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge, 2000.

Marco Cuturi and Arnaud Doucet. Autoregressive Kernels For Time Series. *arXiv e-prints*, art. arXiv:1101.0673, Jan 2011.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

J Diehl, K Ebrahimi-Fard, and N Tapia. Time-warping invariants of multidimensional time series. *arXiv preprint arXiv:1906.05823*, 2019.

Garoe Dorta, Sara Vicente, Lourdes Agapito, Neill DF Campbell, and Ivor Simpson. Structured uncertainty prediction networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5477–5485, 2018.

K. Ebrahimi-Fard and F. Patras. Cumulants, free cumulants and half-shuffles. *Proceedings of the Royal Society*, 2015.

Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. GP-VAE: Deep probabilistic time series imputation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1651–1661. PMLR, 2020.

Samuel J. Gershman and Noah D. Goodman. Amortized inference in probabilistic reasoning. *Cognitive Science*, 36, 2014.

R Giles. A generalization of the strict topology. *Transactions of the American Mathematical Society*, 1971.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

AL Goldberger, LAN Amaral, L Glass, JM Hausdorff, P Ch Ivanov, RG Mark, JE Mietus, GB Moody, CK Peng, and HE Stanley. Components of a new research resource for complex physiologic signals. *PhysioBank, PhysioToolkit, and Physionet*, 2000.

Benjamin Graham. Sparse arrays of signatures for online character recognition. *arXiv preprint arXiv:1308.0371*, 2013.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602 – 610, 2005.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE workshop on automatic speech recognition and understanding*, pp. 273–278. IEEE, 2013a.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649. IEEE, 2013b.

I. Higgins, Loïc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. Set functions for time series. In *ICML*, 2020.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

Ming Hou, Jiajia Tang, Jianhai Zhang, Wanzeng Kong, and Qibin Zhao. Deep multimodal multilinear fusion with high-order polynomial pooling. In *Advances in Neural Information Processing Systems*, pp. 12136–12145, 2019.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Jul 2019. ISSN 1573-756X.

Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237 – 245, 2019. ISSN 0893-6080.

Isak Karlsson, Panagiotis Papapetrou, and Henrik Boström. Generalized random shapelet forests. *Data Min. Knowl. Discov.*, 30(5):1053–1085, September 2016. ISSN 1384-5810.

Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to SGD. *arXiv preprint arXiv:1712.07628*, 2017.

Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. Generalized tensor models for recurrent neural networks. *arXiv preprint arXiv:1901.10801*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Franz J Király and Harald Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 2019.

Jean Kossaifi, Zachary C Lipton, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *arXiv preprint arXiv:1707.08308*, 2017.

Serge Lang. *Algebra*. Springer-Verlag New York, 2002.

C Leslie and R Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 2004.

Yingzhen Li and Stephan Mandt. Disentangled sequential autoencoder, 2018.

Paul Pu Liang, Zhun Liu, Yao-Hung Hubert Tsai, Qibin Zhao, Ruslan Salakhutdinov, and Louis-Philippe Morency. Learning representations from imperfect time series data via tensor rank regularization. *arXiv preprint arXiv:1907.01011*, 2019.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 9628–9639, Red Hook, NY, USA, 2018a. Curran Associates Inc.

Zhun Liu, Ying Shen, Varun Bharadhwaj Lakshminarasimhan, Paul Pu Liang, Amir Zadeh, and Louis-Philippe Morency. Efficient low-rank multimodal fusion with modality-specific factors. *arXiv preprint arXiv:1806.00064*, 2018b.

Terry Lyons. Rough paths, signatures and the modelling of functions on streams. *arXiv preprint arXiv:1405.4537*, 2014.

Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes. *arXiv preprint arXiv:1807.03653*, 2018.

P. Nemenyi. *Distribution-free Multiple Comparisons*. Princeton University, 1963. URL `https://books.google.nl/books?id=nhDMtgAACAAJ`.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

C Reutenauer. *Free Lie Algebras*. Clarendon press – Oxford, 1993.

W. Rudin. *Principles of Mathematical Analysis*. Cambridge University Press, 1965.

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

Tim Sauer, James A Yorke, and Martin Casdagli. Embedology. *Journal of statistical Physics*, 65 (3-4):579–616, 1991.

Patrick Schäfer and Ulf Leser. Multivariate time series classification with weasel+muse. *ArXiv*, abs/1711.11343, 2017.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Satya Narayan Shukla and Benjamin M Marlin. Interpolation-prediction networks for irregularly sampled time series. *arXiv preprint arXiv:1909.07782*, 2019.

Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker, and Hermann Ney. Translation modeling with bidirectional recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14–25, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pp. 366–381. Springer, 1981.

Dacheng Tao, Xuelong Li, Xindong Wu, and Stephen J Maybank. General tensor discriminant analysis and gabor features for gait recognition. *IEEE transactions on pattern analysis and machine intelligence*, 29(10):1700–1715, 2007.

C Toth and H Oberhauser. Bayesian learning from sequential data using gaussian processes with signature covariances. *ICML*, 2020.

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, 1966.

Kerem Sinan Tuncel and Mustafa Gokce Baydogan. Autoregressive forests for multivariate time series modeling. *Pattern Recognition*, 73:202–215, 2018.

Madeleine Udell and Alex Townsend. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Z. Wang, W. Yan, and T. Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, 2017.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Amir Zadeh, Minghai Chen, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Tensor fusion network for multimodal sentiment analysis. *arXiv preprint arXiv:1707.07250*, 2017.

Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

Qibin Zhao, Masashi Sugiyama, Longhao Yuan, and Andrzej Cichocki. Learning efficient tensor representations with ring-structured networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8608–8612. IEEE, 2019.

HOW TO USE THIS APPENDIX

For practitioners, we recommend a look at Section A for a refresher on tensor notation and an introduction to $\mathrm{T}(V)$; further, the introduction of Section B contains a brief summary of the main theoretical properties of Seq2Tens that make it an attractive feature map for sequence data. Sections D and E contain details on algorithms and experiments.

For theoreticians, we recommend Section B for a proof that Φ is universal (Theorem B.3), how the Seq2Tens map behaves in the high-frequency limit as one goes from discrete to continuous time (Proposition B.10), and to Section C for a quantitative statement of low-rank functionals can be turned into high-rank functionals with sequence-to-sequence transformations. We re-emphasize that these more algebra-heavy sections are not needed for practitioners.

A TENSORS AND THE FREE ALGEBRA

This section recalls some basics on the tensor product $\otimes$ and the convolution product that turns the linear space $\mathrm{T}(V)$ into an algebra – the so-called *tensor algebra* or *free algebra over* V. We refer to (Lang, 2002, Chapter 16) for more on tensors, to Reutenauer (1993) for free algebras. Put briefly, for any linear space V there exists a linear space $\mathrm{T}(V)$ that contains V but that also carries a non-commutative product. The space $\mathrm{T}(V)$ can be realized as series of tensors of increasing degree, hence the name tensor algebra.

Tensor products on $\mathbb{R}^d$. If $x = (x_1, \dots, x_d) \in \mathbb{R}^d$ and $y = (y_1, \dots, y_e) \in \mathbb{R}^e$ are two vectors, then their *tensor product* $x \otimes y$ is defined as the $(d \times e)$-matrix, or degree 2 tensor, with entries $(x \otimes y)_{i,j} = x_i y_j$. This is also commonly called the *outer product* of the two vectors. The space $\mathbb{R}^d \otimes \mathbb{R}^e$ is defined as the linear span of all degree 2 tensors $x \otimes y$ for $x \in \mathbb{R}^d, y \in \mathbb{R}^e$. If $z \in \mathbb{R}^f$ is another vector, then one may form a degree 3 tensor $x \otimes y \otimes z$ with shape $(d \times e \times f)$ defined to have entries $(x \otimes y \otimes z)_{i,j,k} = x_i y_j z_k$. The space $\mathbb{R}^d \otimes \mathbb{R}^e \otimes \mathbb{R}^f$ is analogously defined as the linear span of all degree 3 tensors $x \otimes y \otimes z$ for $x \in \mathbb{R}^d, y \in \mathbb{R}^e, z \in \mathbb{R}^f$.

The tensor product of two general vector spaces V and W can be defined even if they are infinite dimensional, see (Lang, 2002, Chapter 16), but we invite readers unfamiliar with general tensor spaces to think of V as $\mathbb{R}^d$ below.

The tensor algebra $\mathrm{T}(V)$. Ultimately we are not only interested in tensors of some fixed degree m – that is an element of $V^{\otimes m}$ – but sequences of tensors of increasing degree. Given some linear space V, the linear space $\mathrm{T}(V)$ is defined as set of all tensors of any degree over V. Formally

$$\mathrm{T}(V) := \prod_{m \geq 0} V^{\otimes m} = \{\mathbf{t} = (\mathbf{t}_m)_{m \geq 0} \,|\, \mathbf{t} \in V^{\otimes m}\} \tag{19}$$

where we use the notation $V^{\otimes 1} = V$, $V^{\otimes 2} = V \otimes V$, $V^{\otimes 3} = V \otimes V \otimes V$ and so on; by convention we let $V^{\otimes 0} = \mathbb{R}$. We normally write elements of $\mathrm{T}(V)$ as $\mathbf{t} = (\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \dots)$ such that $\mathbf{t}_m \in V^{\otimes m}$, that is, $\mathbf{t}_0$ is a scalar, $\mathbf{t}_1$ is a vector, $\mathbf{t}_2$ is a matrix, $\mathbf{t}_3$ is a 3-tensor and so on. Note that $\mathrm{T}(V)$ is again a linear space if we define addition and scalar multiplication as

$$\mathbf{s} + \mathbf{t} = (\mathbf{s}_m + \mathbf{t}_m)_{m \geq 0} \in \mathrm{T}(V) \text{ and } c \cdot \mathbf{t} = (c\mathbf{t}_m)_{m \geq 0} \in \mathrm{T}(V) \tag{20}$$

for $\mathbf{s}, \mathbf{t} \in \mathrm{T}(V)$ and $c \in \mathbb{R}$.

Example A.1. Let $V = \mathbb{R}^d$. For $\mathbf{v} = (\mathbf{v}_i)_{i=1,\dots,d} \in \mathbb{R}^d$ consider $\mathbf{t} = (\mathbf{v}^{\otimes m})_{m \geq 0} \in \mathrm{T}(\mathbb{R}^d)$ where we denote for brevity

$$\mathbf{t}_m := \mathbf{v}^{\otimes m} := \underbrace{\mathbf{v} \otimes \cdots \otimes \mathbf{v}}_{m \text{ many tensor products } \otimes} \in (\mathbb{R}^d)^{\otimes m} \text{ and by convention we set } \mathbf{v}^{\otimes 0} := 1 \in (\mathbb{R}^d)^{\otimes 0}.$$

That is, $\mathbf{t}_1 = \mathbf{v}^{\otimes 1} = \mathbf{v}$ is a d-dimensional vector, with the i coordinate equal to $\mathbf{v}_i$; $\mathbf{t}_2 = \mathbf{v}^{\otimes 2}$ is $d \times d$-matrix with the (i,j)-coordinate equal to $\mathbf{v}_i \mathbf{v}_j$; $\mathbf{t}_3 = \mathbf{v}^{\otimes 3}$ is degree 3-tensor with the (i,j,k)-coordinate equal to $\mathbf{v}_i \mathbf{v}_j \mathbf{v}_k$. In this special case, the element $\mathbf{t} \in \mathrm{T}(\mathbb{R}^d)$ consists of entries $\mathbf{t}_m = \mathbf{v}^{\otimes m} \in (\mathbb{R}^d)^{\otimes m}$ that are symmetric tensors, that is the $(i_1, \dots, i_m)$-th coordinate is the same as the $(i_{\sigma(1)}, \dots, i_{\sigma(d)})$ coordinate if σ is a permutation of $\{1, \dots, d\}$. However, we emphasize that in general an element of $\mathrm{T}(\mathbb{R}^d)$ does not need to be made up of symmetric tensors.

A product on $\mathrm{T}(V)$. Key to our approach is that $\mathrm{T}(V)$ is not only a linear space, but what distinguishes it as a feature space for sequences is that it carries a non-commutative product. In other words, $\mathrm{T}(V)$ is not just a vector space but a (non-commutative) algebra (an algebra is a vector space where one can multiply elements). This is the so-called *tensor convolution product* and defined as follows

$$\mathbf{s} \cdot \mathbf{t} := \big(\sum_{i=0}^{m} \mathbf{s}_i \otimes \mathbf{t}_{m-i} \big)_{m \geq 0} = \big(1, \mathbf{s}_1 + \mathbf{t}_1, \mathbf{s}_2 + \mathbf{s}_1 \otimes \mathbf{t}_1 + \mathbf{t}_2, \dots \big). \tag{21}$$

In a precise mathematical sense, $\mathrm{T}(V)$ is the most general algebra containing V, namely $\mathrm{T}(V)$ *is the "free algebra" that contains* V; see (Lang, 2002, Chapter 16) for the precise definition of free objects.

B A UNIVERSAL FEATURE MAP FOR SEQUENCES OF ARBITRARY LENGTH

Recall from Section 2, that given a map defined on a set $\mathcal{X}$

$$\phi : \mathcal{X} \to V$$

we lift ϕ to a map $\varphi : \mathcal{X} \to \mathrm{T}(V)$ and define the Seq2Tens feature map for sequences in $\mathcal{X}$ of arbitrary length as

$$\Phi : \mathrm{Seq}(\mathcal{X}) \to \mathrm{T}(V), \quad \mathbf{x} \to \prod_{i=1}^{T} \varphi(\mathbf{x}_i).$$

The remainder of Section B makes the following statements mathematically rigorous:

(i) Φ is a universal feature map whenever ϕ is a universal (Section B.1 and B.2),

(ii) Φ makes sequences of different length comparable analogous to how m-mers make strings of different length comparable (Section B.3),

(iii) Φ converges to a well-defined object when we go from discrete to continuous time (sequences converge to paths) (Section B.4).

B.1 THE UNIVERSALITY OF Φ.

Definition B.1. Let $\mathcal{X}$ be a topological space (the "data space") and W a linear space ("the feature space"). We say that a function $f : \mathcal{X} \to W$ is universal (to $C_b(\mathcal{X})$) if the the set of functions

$$\{x \mapsto \langle \ell, f(x) \rangle \, : \, \ell \in W'\} \subseteq C_b(\mathcal{X}) \tag{22}$$

is dense in $C_b(\mathcal{X})$.

Example B.2. Classic examples of this in ML are

- For $\mathcal{X} \subset \mathbb{R}^d$ bounded and $W = \mathrm{T}(\mathbb{R}^d)$, the polynomial map $p : \mathbb{R}^d \to \mathrm{T}(\mathbb{R}^d), \mathbf{x} \mapsto (1, \mathbf{x}, \mathbf{x}^{\otimes 2}, \mathbf{x}^{\otimes 3}, \mathbf{x}^{\otimes 4}, \dots)$ is universal (Rudin, 1965).
- The 1-layer neural net map $\mathbf{x} \mapsto \prod_\theta N_\theta(\mathbf{x})$ where θ runs over all configurations of parameters is universal under some very mild conditions (Hornik, 1991).

We now prove the main result of this section

Theorem B.3. *Let $\varphi : \mathcal{X} \to \mathrm{T}(V), \mathbf{x} \mapsto (\varphi_m(\mathbf{x}))_{m \geq 0}, \varphi_m(\mathbf{x}) \in V^{\otimes m}$ be such that:*

1. *For any $n \geq 1$ the support of $(\varphi_0, \varphi_1, \dots, \varphi_{m_1})^{\otimes n}$ and φ_{m_2} are disjoint if $1 \leq m_1 < m_2$.*
2. *$\varphi_0 = 1$ and $\varphi_1 : \mathcal{X} \to V$ is a bounded universal map with at least one constant term.*

Then

$$\Phi : \mathrm{Seq}(\mathcal{X}) \to \mathrm{T}(V), \quad (\mathbf{x}_1, \dots, \mathbf{x}_L) \mapsto \prod_{i=1}^{L} \varphi(\mathbf{x}_i) \tag{23}$$

is universal.

Remark B.4.

(i) TheoremB.3 implies that $\mathbf{x} \mapsto \prod_{i=1}^{L}(1, \phi(\mathbf{x}), 0, \dots)$ is universal whenever $\phi : \mathcal{X} \to V$ is universal. This is the lift we use throughout the main text, see equation 6.

(ii) By taking $\varphi : \mathbb{R}^d \to \mathrm{T}(\mathbb{R}^d), \mathbf{x} \mapsto (1, \mathbf{x}, \frac{\mathbf{x}^{\otimes 2}}{2!}, \frac{\mathbf{x}^{\otimes 3}}{3!}, \dots)$ one recovers Chen's signature (Chen, 1954; 1957; 1958) as used in rough paths.

(iii) By taking $\varphi : \mathbb{R}^d \to \mathrm{T}(V)$, $\varphi_1(\mathbf{x})$ the polynomial map and $\varphi_m(\mathbf{x}) = 0$ for $m \geq 2$ one recovers the iterated sums of Diehl et al. (2019) and Király & Oberhauser (2019).

(iv) By taking each φ_m to be a trainable Neural Network one gets a trainable universal map Φ for sequences that includes all of the above,

B.1.1 The algebra of linear functionals on Φ.

The proof of Theorem B.3 uses that if φ is universal, then the space of linear functionals on $\Phi(\mathbf{x})$ forms a commutative algebra, that is for two linear functionals ℓ_1, ℓ_2 there exists another linear functional ℓ such that

$$\langle \ell_1, \Phi(\mathbf{x}) \rangle \langle \ell_2, \Phi(\mathbf{x}) \rangle = \langle \ell, \Phi(\mathbf{x}) \rangle. \tag{24}$$

This new functional ℓ is constructed in explicit way from ℓ_1 and ℓ_2, with a so-called quasi-shuffle product. In the remainder of this section B.1, we prepare and give the proof of Theorem B.3: subsection B.1.1 introduces the quasi-shuffle product, and subsection B.1.1 uses this to prove Theorem B.3.

We spell out the proof for the case $\varphi = (1, \phi, 0, 0, \dots) \in \mathrm{T}(V)$ since this is the form we use in the main text, Proposition 2.1, and the other cases follow similarly. In fact, without loss of generality we can take $\phi = \mathrm{id}$ since this does not change the algebraic structure in any way. That is, we take

$$\Phi : \mathrm{Seq}(V) \to \mathrm{T}(V), \quad \Phi(\mathbf{x}_1, \dots, \mathbf{x}_L) := \prod_{i=1}^{L} \varphi(\mathbf{x}_i) \tag{25}$$

with $\varphi(\mathbf{x}) = (1, \mathbf{x}, 0, 0, \dots)$. By using the definition of the product in $\mathrm{T}(V)$ and expanding equation 25 we get

$$\Phi(\mathbf{x}_1, \dots, \mathbf{x}_L) = (1, \sum_{i=1}^{L} \underbrace{\mathbf{x}_i}_{V}, \sum_{1 \leq i_1 < i_2 \leq L} \underbrace{\mathbf{x}_{i_1} \otimes \mathbf{x}_{i_2}}_{V^{\otimes 2}}, \sum_{1 \leq i_1 < i_2 < i_3 \leq L} \underbrace{\mathbf{x}_{i_1} \otimes \mathbf{x}_{i_2} \otimes \mathbf{x}_{i_3}}_{V^{\otimes 3}}, \cdots) \tag{26}$$

In general, writing $\Phi_m(\mathbf{x})$ for the projection of $\Phi(\mathbf{x})$ onto $V^{\otimes m}$, we have

$$\Phi_m(\mathbf{x}) = \sum_{1 \leq i_1 < \dots < i_m \leq L} \mathbf{x}_{i_1} \otimes \dots \otimes \mathbf{x}_{i_m}. \tag{27}$$

So if $\ell = (0, 0, \dots, \mathbf{v}_1 \otimes \dots \otimes \mathbf{v}_m, 0, \dots)$ with $\mathbf{v}_1, \dots, \mathbf{v}_m \in V$, then

$$\langle \ell, \Phi_m(\mathbf{x}) \rangle = \langle \mathbf{v}_1 \otimes \dots \otimes \mathbf{v}_m, \sum_{1 \leq i_1 < \dots < i_m \leq L} \mathbf{x}_{i_1} \otimes \dots \otimes \mathbf{x}_{i_m} \rangle \tag{28}$$

$$= \sum_{1 \leq i_1 < \dots < i_m \leq L} \langle \mathbf{v}_1 \otimes \dots \otimes \mathbf{v}_m, \mathbf{x}_{i_1} \otimes \dots \otimes \mathbf{x}_{i_m} \rangle = \sum_{1 \leq i_1 < \dots < i_m \leq L} \langle \mathbf{v}_1, \mathbf{x}_{i_1} \rangle \cdots \langle \mathbf{v}_m, \mathbf{x}_{i_m} \rangle. \tag{29}$$

Hence $\langle \ell, \Phi_m(\mathbf{x}) \rangle$ can be computed efficiently without computing $\Phi(\mathbf{x})$. Proposition 3.3 follows by linearity since by definition $\langle \ell, \Phi(\mathbf{x}) \rangle = \sum_{m \geq 0} \langle \ell_m, \Phi(\mathbf{x}) \rangle$ and for each of the terms we can use the above formula when $\ell = (\ell_0, \ell_1, \ell_2, \dots, \ell_M, 0, \dots)$ is of rank-1 and of degree M (Definition 3.1).

Non-linear functionals acting on Φ. We now investigate what happens when one applies non-linear functions to $\Phi(\mathbf{x})$. To do this, we first note that since $\mathrm{T}(V)$ is a vector space, we may form the free algebra over $\mathrm{T}(V)$, denoted by $\mathrm{T}(\mathrm{T}(V))$, or $\mathrm{T}^2(V)$. It may be decomposed as

$$\mathrm{T}^2(V) = \prod_{n_1, \dots, n_k \geq 0} V^{\otimes n_1} \big| \cdots \big| V^{\otimes n_k} \tag{30}$$

where we use the notation $\otimes$ for the tensor product on V and the bar $|$ for the tensor product on $\mathrm{T}(V)$. See Ebrahimi-Fard & Patras (2015) for more on $\mathrm{T}^2(V)$ and the bar notation.

Definition B.5. If $x \in V$ is a vector, we denote by $x^\star$ its extension

$$x^\star := (x^{\otimes m})_{m \geq 0} = (1, x, x^{\otimes 2}, x^{\otimes 3}, \ldots) \tag{31}$$

and if $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(V)$ is a sequence, then

$$\mathbf{x}^\star := (\mathbf{x}_1^\star, \ldots, \mathbf{x}_L^\star) \in \mathrm{Seq}(\mathrm{T}(V)) \tag{32}$$

Since $\mathbf{x}^\star$ is a sequence in $\mathrm{T}(V)$, we may compute $\Phi(\mathbf{x}^\star)$ which takes values in $\mathrm{T}(\mathrm{T}(V)) = \mathrm{T}^2(V)$.

The reason for the above definition is that when products of linear functions in $\mathrm{T}(V)$ act on $\Phi(\mathbf{x})$, they may be described as linear functions in $\mathrm{T}^2(V)$ acting on $\Phi(\mathbf{x}^\star)$. That is, $\mathrm{T}(V)$ is not big enough to capture all non-linear functions acting on $\Phi(\mathbf{x})$, but $\mathrm{T}^2(V)$ is.

Definition B.6. Assume that V has basis $e_1, \ldots, e_d$. The *quasi-shuffle* product

$$\star : \mathrm{T}^2(V) \times \mathrm{T}^2(V) \to \mathrm{T}^2(V) \tag{33}$$

is defined inductively on rank 1 elements $\ell_1 = e_{i_1}|\cdots|e_{i_m}, \ell_2 = e_{j_1}|\cdots|e_{j_n}$ by

$$(\ell_1|e_i) \star (\ell_2|e_j) = (\ell_1|e_i \star \ell_2)|e_j + (\ell_1 \star \ell_2|e_j)|e_i + (\ell_1 \star \ell_2)|(e_i \otimes e_j). \tag{34}$$

By linearity $\star$ extends to a product on all of $\mathrm{T}(V)$.

Lemma B.7. *The map Φ satisfies the following*

$$\langle \ell_1, \Phi(\mathbf{x})\rangle \langle \ell_2, \Phi(\mathbf{x})\rangle = \langle \ell_1 \star \ell_2, \Phi(\mathbf{x}^\star)\rangle. \tag{35}$$

Proof. By writing out equation 27 in coordinates we get

$$\langle e_{i_1}|\cdots|e_{i_m}, \Phi(\mathbf{x})\rangle = \sum_{1 \leq k_1 < \cdots < k_m \leq L} \langle e_{i_1}, \mathbf{x}_{k_1}\rangle \cdots \langle e_{i_m}, \mathbf{x}_{k_m}\rangle. \tag{36}$$

which shows that Φ satisfies a recurrence equation. The proof follows by induction. □

The space $\mathrm{T}^2(V)$ might seem very large and difficult to work with at first. The power of this representation comes from the fact that one may leverage this in proving strong statements about the original map $\Phi : \mathrm{Seq}(V) \to \mathrm{T}(V)$, and we will use this in the next subsection.

B.2 Proof of Theorem B.3.

We prepare the proof of Theorem B.3 with the following lemma.

Lemma B.8. *Let $\mathrm{Seq}^1(V)$ be the set of all $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(V)$ with the form $x_i = (1, x_i^1, \ldots, x_i^d)$. That is, all sequences where one of the terms is constant. Then the map*

$$\mathrm{Seq}^1(V) \to \mathrm{T}(V), \quad (\mathbf{x}_1, \ldots, \mathbf{x}_L) \to \prod_{i=1}^{L} (1 + \mathbf{x}_i) \tag{37}$$

is injective.

Proof. Follows from an induction argument over L. For $L = 1$ it is clear since

$$\langle e_i, \Phi(\mathbf{x})\rangle = x_i. \tag{38}$$

Assume that it is true for L, let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{L+1}), \mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_{L+1})$, where we may assume that both have length $L + 1$ by taking any number of components to be 0 if necessary. Let ℓ_1 be some linear function that separates $\Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)$ and $\Phi(\mathbf{y}_1, \ldots, \mathbf{y}_L)$ and ℓ_2 some linear function that separates $\Phi(\mathbf{x}_2, \ldots, \mathbf{x}_{L+1})$ and $\Phi(\mathbf{y}_2, \ldots, \mathbf{y}_{L+1})$, then by fixing some $\gamma \in \mathbb{R}$:

$$\langle \ell_1 \otimes e_0 + \gamma e_0 \otimes \ell_2, \Phi(\mathbf{x}) - \Phi(\mathbf{y})\rangle \tag{39}$$

$$= \langle \ell_1, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L) - \Phi(\mathbf{y}_1, \ldots, \mathbf{y}_L)\rangle + \gamma \langle \ell_2, \Phi(\mathbf{x}_2, \ldots, \mathbf{x}_{L+1}) - \Phi(\mathbf{y}_2, \ldots, \mathbf{y}_{L+1})\rangle. \tag{40}$$

Since neither $\langle \ell_1, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L) - \Phi(\mathbf{y}_1, \ldots, \mathbf{y}_L)\rangle$ nor $\langle \ell_2, \Phi(\mathbf{x}_2, \ldots, \mathbf{x}_{L+1}) - \Phi(\mathbf{y}_2, \ldots, \mathbf{y}_{L+1})\rangle$ are 0 by assumption there exists some $\gamma \in \mathbb{R}$ such that $\langle \ell_1 \otimes e_0 + \gamma e_0 \otimes \ell_2, \Phi(\mathbf{x}) - \Phi(\mathbf{y})\rangle \neq 0$. This shows the assertion. □

We now have everything to give a proof of Theorem B.3.

Proof of Theorem B.3. We will show that linear functionals on Φ are dense in the *strict topology* (Giles, 1971). By Theorem (Giles, 1971, Theorem 3.1) it is enough to show that linear functions on Φ form an algebra since by Lemma B.8 they separates the points of $\mathrm{Seq}(\mathcal{X})$. Since they clearly form a vector space it is enough to show that they are closed under point-wise multiplication. Let ℓ_1, ℓ_2 be two such, then by Lemma B.7

$$\langle \ell_1, \Phi(\mathbf{x})\rangle\langle \ell_2, \Phi(\mathbf{x})\rangle = \langle \ell_1 \star \ell_2, \prod_{i=1}^{L} \phi(\mathbf{x}_i)^\star\rangle \tag{41}$$

so it is enough to show that $\ell_1 \star \ell_2$ also is a linear function on $\Phi(\mathbf{x})$. Note that inductively it is enough to show that if e_i, e_j are unit vectors, then $e_i \otimes e_j$ is a linear function on $\Phi(\mathbf{x})$. By assumption ϕ is bounded and universal, so the continuous bounded function $\mathbf{x} \mapsto \langle e_i, \phi(\mathbf{x}_k)\rangle\langle e_j, \phi(\mathbf{x}_k)\rangle$ is approximately linear, and we may write

$$\langle e_i, \phi(\mathbf{x}_k)\rangle\langle e_j, \phi(\mathbf{x}_k)\rangle = \langle h, \phi(\mathbf{x}_k)\rangle + \varepsilon(\mathbf{x}_k) \tag{42}$$

where $\varepsilon(\mathbf{x}_k)$ can be made arbitrarily small in the strict topology. The assertion now follows since

$$\langle e_i \otimes e_j, \prod_{i=1}^{L} \phi(\mathbf{x}_i)^\star)\rangle = \sum_{k=1}^{L} \langle e_i, \phi(\mathbf{x}_k)\rangle\langle e_j, \phi(\mathbf{x}_k)\rangle = \sum_{k=1}^{L} \langle h, \phi(\mathbf{x}_k)\rangle + \varepsilon(\mathbf{x}_k) \tag{43}$$

$$= \langle e_h, \Phi(\mathbf{x})\rangle + n \max_{1\le k\le n} \varepsilon(\mathbf{x}_k). \tag{44}$$

□

B.3 SEQ2TENS MAKES SEQUENCES OF DIFFERENT LENGTH COMPAREABLE

The simplest kind of a sequence is a string, that is a sequence of letters. Strings are determined by

(i) what letters appear in them,

(ii) in what order the letters appear.

A classical way to produce a graded description of strings is by counting their non-contiguous substrings. These are the so-called m-*mers*; for example,

$$\text{The string "aabc" has the 2-mers } \{aa, ab, ac, ab, ac, bc\}. \tag{45}$$

Measuring similarity between strings by counting how many substrings they have in common is a sensible similarity measure, even if the strings have different length; we refer to Leslie & Kuang (2004) for applications and to (Cristianini & Shawe-Taylor, 2000, Chapter 11) for detailed introduction to the use of substrings in ML.

Our Seq2Tens feature map can be regarded as a vast generalization of such subpattern matching: if $\Phi(\mathbf{x}) = (\Phi_m(\mathbf{x}))_{m\ge 0} \in \mathrm{T}(V)$ then the tensor $\Phi_m(\mathbf{x}) \in V^{\otimes m}$ represents non-contiguous subsequences of $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ of length m and thus comparing $\Phi_m(\mathbf{x})$ and $\Phi_m(\mathbf{y})$ is meaningful even when $\mathbf{x}$ and $\mathbf{y}$ are of different length. It is instructive to spell out in detail how m-mers are a special case of Seq2Tens, Example B.9, and how it generalizes, Example B.11.

Example B.9. Let $\mathcal{X} = \{a, b, c\}$ and $\phi : \mathcal{X} \to V = \mathbb{R}^3$ defined by mapping $a, b, c \in \mathcal{X}$ to the unit vectors $e_1, e_2, e_3 \in V$, so that $\varphi(a) = (1, e_1, 0, 0, ...) \in \mathrm{T}(V)$, $\varphi(b) = (1, e_2, 0, \ldots) \in \mathrm{T}(V)$, and $\varphi(c) = (1, e_3, 0, \ldots) \in \mathrm{T}(V)$. For the sequence $\mathbf{x} = (a, a, b, c) \in \mathrm{Seq}(\mathcal{X})$ we get

$$\Phi(\mathbf{x}) = \varphi(a)\varphi(a)\varphi(b)\varphi(c) \tag{46}$$

$$= (1, e_1, 0, 0, \ldots,) \cdot (1, e_1, 0, 0, \ldots) \cdot (1, e_2, 0, 0, \ldots) \cdot (1, e_3, 0, 0, \ldots) \tag{47}$$

$$= (1, \underbrace{2e_1 + e_2 + e_3}_{\in V}, \underbrace{e_1 \otimes e_1 + 2e_1 \otimes e_2 + 2e_1 \otimes e_3 + e_2 \otimes e_3}_{\in V^{\otimes 2}}, \tag{48}$$

$$\underbrace{e_1 \otimes e_1 \otimes e_2 + e_1 \otimes e_1 \otimes e_3, e_1 \otimes e_1 \otimes e_3 \otimes e_4}_{\in V^{\otimes 3}}, \underbrace{e_1 \otimes e_1 \otimes e_2 \otimes e_3}_{\in V^{\otimes 4}}, 0, \ldots). \tag{49}$$

We see that the tensor $\Phi(\mathbf{x}) \in V^{\otimes m}$ of degree m contains the m-mers, that is the coordinates of $\Phi_m(\mathbf{x})$ count how often a subsequence of length m in $\mathbf{x} = (a, a, b, c)$ appears; e.g. the coordinate $e_1 \otimes e_2$ of $\Phi_2(\mathbf{x})$ equals 2 because the substring "a,b" appears twice but the $e_1 \otimes e_1$ coordinate of $\Phi_2(\mathbf{x})$ equals 1 since the substring "a,a" appears only once in (a, a, b, c), etc. Similarly, the only non-zero coordinate of $\Phi_4(\mathbf{x})$ is $e_1 \otimes e_1 \otimes e_2 \otimes e_3$ since "a,a,b,c" is the only substring of (a, a, b, c) and consequently all coordinate of $\Phi_m(\mathbf{x})$ are 0 for $m \geq 5$.

An analogous calculation shows that for continuous domain such as $\mathcal{X} = \mathbb{R}^d$ the coordinates of $\Phi_m(\mathbf{x}) \in (\mathbb{R}^d)^{\otimes m}$ measure movement patterns in these coordinates. Section B.4 below shows that this interpretation also holds when we go from discrete time (sequences) to continuous time (paths); Example B.11.

B.4 Convergence from discrete to continuous time

A common source of of sequence data is to measure quantity $(x(t))_{t\in[0,T]}$ that evolves in continuous time at fixed times $t_1, \ldots, t_L$ to produce a sequence $\mathbf{x} = (x(t_1), \ldots, x(t_L)) \in \mathrm{Seq}(\mathcal{X})$. Often the measurements are of high-frequency ($|t_{i+1} - t_i| \to 0$ and $L \to \infty$) and is interesting to understand how our Seq2Tens approach behaves in this limiting case. As it turns out, when combined with taking finite differences[1] our feature map $\Phi(\mathbf{x})$ converges to a classical object in analysis, the so-called *signature of the path* x in this limit, (Chevyrev & Kormilitzin, 2016) . For brevity, we spell it out here for smooth paths and with the lift $\varphi(x) = (1, \phi(x), 0, \ldots)$, but readers familiar with rough paths will notice that the result generalizes even to non-smooth paths such as Brownian motion.

Proposition B.10. *Let $x \in C^1([0, 1], \mathbb{R}^d)$ and for every $L \geq 1$ define*

$$\mathbf{x}^L := (x(t_0^L), x(t_1^L) - x(t_0^L), \ldots, x(t_L^L) - x(t_{L-1}^L)) \tag{50}$$

where $t_i := \frac{i}{L}$ for $i = 0, \ldots, L$. Then for every $m \geq 0$ we have

$$\Phi_m(\mathbf{x}^L) \to \int_{0\leq t_1\leq\cdots\leq t_m\leq 1} \frac{dx}{dt}(t_1) \otimes \cdots \otimes \frac{dx}{dt}(t_m) dt_1 \cdots dt_m \text{ as } L \to \infty. \tag{51}$$

Proof. Using the recurrence relation from the proof of Lemma B.7 we may write

$$\Phi_m(\mathbf{x}^L) = \sum_{i=1}^{L} \Phi_{m-1}(\mathbf{x}_i^L) \otimes \big(x(t_{i+1}) - x(t_i)\big) \tag{52}$$

where $\mathbf{x}_i^L$ denotes the sequence $(x(t_0^L), x(t_1^L) - x(t_0^L), \ldots, x(t_i^L) - x(t_{i-1}^L))$. By a Taylor expansion this is equal to

$$\Phi_m(\mathbf{x}^L) = \sum_{i=1}^{L} \Phi_{m-1}(\mathbf{x}_i^L) \otimes \frac{dx}{dt}(t_i)(t_{i+1} - t_i) + O(1/L), \tag{53}$$

so by unravelling the recurrence relation we may write

$$\Phi_m(\mathbf{x}^L) = \sum_{1\leq i_1<\ldots<i_m\leq L} \frac{dx}{dt}(t_{i_1}) \otimes \cdots \otimes \frac{dx}{dt}(t_{i_m})(t_{i_1+1} - t_{i_1}) \cdots (t_{i_m+1} - t_{i_m}) + O(1/L), \tag{54}$$

which is a Riemann sum, and thus converges to the asserted limit. □

The interpretation of $\Phi_m(\mathbf{x})$ as counting sub-patterns remains even in the continuous time case:

Example B.11. Let $x \in C^1([0, 1], \mathbb{R}^d)$ and $e_1, \ldots, e_d$ the standard basis of $V = \mathbb{R}^d$. For $m = 1$ the e_1 coordinate equals

$$\langle e_1, \int_{t=0}^{1} \frac{dx}{dt}(t) dt\rangle = \langle e_1, x(1) - x(0)\rangle$$

[1]This is necessary to counteract the fact the magnitude of Φ grows with the sum of the elements in the sequence

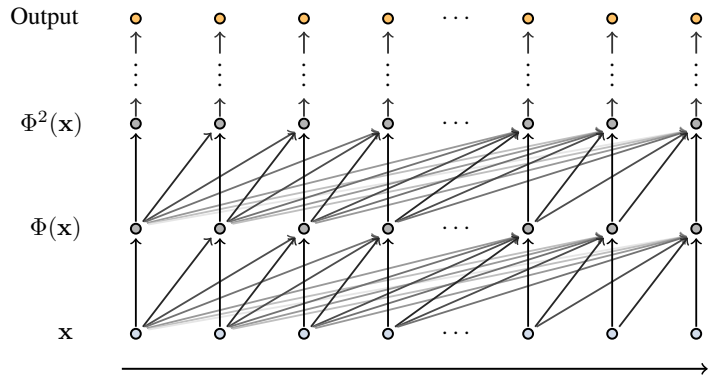

Figure 3: The sequence-to-sequence transformation. The bottom row is the original sequence $\mathbf{x}$ and subsequent ones apply the map Φ in the sequence-to-sequence manner.

which measures the total movement of the path x in the direction e_1. Analogously, for $m = 2$ the $e_1 \otimes e_2$ coordinate equals

$$\langle e_1 \otimes e_2, \int_{0 \le t_1 \le \cdots \le t_m \le 1} \frac{x(t_1)}{dt_1} \otimes \frac{x(t_2)}{dt_2} dt_1 dt_2 = \int_{0 \le t_1 \le t_2 \le 1} \langle e_1, \frac{x(t_1)}{dt_1} \rangle \langle e_2, \frac{x(t_2)}{dt_2} \rangle dt_1 dt_2. \quad (55)$$

which measure the number of ordered tuples (t_1, t_2), $t_1 < t_2$, such that x moves in direction e_1 at time t_1 and subsequently in direction e_2 at time t_2.

C Stacking sequence-to-sequence transforms

As Φ applies to sequences of any length, we may use it to map the original sequence to another sequence in feature space,

$$\mathrm{Seq}(V) \to \mathrm{Seq}(\mathrm{T}(V)) \quad (56)$$

$$(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_L) \mapsto \Big(\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1, \mathbf{x}_2), \Phi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3), \ldots, \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_L)\Big). \quad (57)$$

Since $\mathrm{T}(V)$ is again a linear space, we can repeat this procedure to map $\mathrm{Seq}(\mathrm{T}(V))$ to $\mathrm{Seq}(\mathrm{T}(\mathrm{T}(V)))$. By repeating this D times, we have constructed sequence-to-sequence transforms

$$\mathrm{Seq}(V) = \mathrm{Seq}(\mathrm{T}(V)) \to \mathrm{Seq}(\mathrm{T}(\mathrm{T}(V))) \to \cdots \to \mathrm{Seq}(\underbrace{T(\cdots T}_{D \text{ times}}(V) \cdots)). \quad (58)$$

See Figure 3 for an illustration. We emphasize that in each step of the iteration, the newly created sequence evolves in a much richer space than in the previous step. To make this precise we now introduce the *higher rank tensor algebras*.

Higher rank tensor algebras. Just like in Appendix B we need to enlarge the ambient space $\mathrm{T}(V)$. Recall that we defined $\mathrm{T}^2(V) := \mathrm{T}(\mathrm{T}(V))$. This construction can be iterated indefinitely and leads to the higher rank tensor algebras, recursively defined as follows:

Definition C.1. Define the spaces

$$\mathrm{T}^0(V) = V, \quad \mathrm{T}^D(V) = \prod_{m \ge 0} \big(\mathrm{T}^{D-1}(V)\big)^{\otimes m}. \quad (59)$$

We use the notation $\otimes_{(D)}$ for the tensor product on $\mathrm{T}^{D-1}(V)$ which makes $\big(\mathrm{T}^D(V), +, \otimes_{(D)}\big)$ into a multi-graded algebra over $\boldsymbol{s}$. See Bonnier et al. (2020) for more on this iterated construction.

Half-shuffles. By iterating the sequence-to-sequence D times, one gets a map

$$\Phi^D : \mathrm{Seq}(V) \to \mathrm{T}^D(V). \quad (60)$$

These are very large spaces, but as we will see in Proposition C.3 below, linear functionals on the full map $\mathrm{Seq}(V) \to \mathrm{T}^D(V)$ can be de-constructed into so called *half-quasi-shuffle* on the original map $\Phi : \mathrm{Seq}(V) \to \mathrm{T}(V)$.

Just like in Appendix B we consider the sequence $\mathbf{x}$ as its extension $\mathbf{x}^\star$ taking values in $\mathrm{T}(V)$. Hence linear functionals can be written as linear combinations of elements of the form $e_{i_1} \otimes_{(2)} \cdots \otimes_{(2)} e_{i_n}$.

Definition C.2. The half-quasi-shuffle product is defined on rank 1 tensors by

$$\ell_1 \prec (\ell_2 \otimes_{(2)} e_i) = (\ell_1 \star \ell_2) \otimes_{(2)} e_i \tag{61}$$

and extends by bi-linearity to a map $\mathrm{T}^2(V) \times \mathrm{T}^2(V) \to \mathrm{T}^2(V)$.

Proposition C.3 shows that by composing Φ with itself, low degree tensors on the second level can be rewritten as higher degree tensors on the first level. This indicates that iterated compositions of Φ can be much more efficient than computing everything on the first level. We show this for the first level, but by iterating the statement it can be applied for any number $D \geq 2$.

Proposition C.3. *Let $\mathbf{\Phi}$ be the sequence-to-sequence transformation:*

$$\mathrm{Seq}(V) \to \mathrm{Seq}(\mathrm{T}(V)), \quad (\mathbf{x}_1, \ldots, \mathbf{x}_L) \mapsto (\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}, \ldots, \mathbf{x}_L)) \tag{62}$$

and let $\Delta(\mathbf{x}_0, \ldots, \mathbf{x}_L) = (\mathbf{x}_1 - \mathbf{x}_0, \ldots, \mathbf{x}_L - \mathbf{x}_{L-1})$. Then

$$\langle e_{\ell_1} \otimes_{(3)} e_{\ell_2}, \Phi(\Delta\mathbf{\Phi}(\mathbf{x}))\rangle = \langle \ell_1 \prec \ell_2, \Phi(\mathbf{x})\rangle \tag{63}$$

Proof. We use the notation $\Phi(\mathbf{x})_k = \Phi(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. By induction:

$$\langle e_{\ell_1} \otimes_{(3)} e_{\ell_2 \otimes_{(2)} e_i}, \Phi(\Delta\mathbf{\Phi}(\mathbf{x}))\rangle \tag{64}$$

$$= \sum_{1 \leq k_1 < k_2 \leq L} \big(\langle \ell_1, \Phi(\mathbf{x})_{k_1} - \langle \ell_1, \Phi(\mathbf{x})_{k_1 - 1}\big)\rangle \big(\langle \ell_2 \otimes_{(2)} e_i, \Phi(\mathbf{x})_{k_2}\rangle - \langle \ell_2 \otimes_{(2)} e_i, \Phi(\mathbf{x})_{k_2 - 1}\rangle\big) \tag{65}$$

$$= \sum_{k=1}^{L-1} \langle \ell_1, \Phi(\mathbf{x})_k\rangle \big(\langle \ell_2 \otimes_{(2)} e_i, \Phi(x)_{k+1}\rangle - \langle \ell_2 \otimes_{(2)} e_i, \Phi(\mathbf{x})_k\rangle\big) \tag{66}$$

$$= \sum_{k=1}^{L-1} \langle \ell_1, \Phi(\mathbf{x})_k\rangle \big(\sum_{1 \leq l \leq k} \langle \ell_2, \Phi(\mathbf{x})_l\rangle \mathbf{x}^i_{l+1} - \langle \ell_2, \Phi(\mathbf{x})_{l-1}\rangle \mathbf{x}^i_l\big) \tag{67}$$

$$= \sum_{k=1}^{L-1} \langle \ell_1, \Phi(\mathbf{x})_k\rangle \langle \ell_2, \Phi(\mathbf{x})_k\rangle \mathbf{x}^i_{k+1} = \sum_{k=1}^{L-1} \langle \ell_1 \star \ell_2, \Phi(\mathbf{x})_k\rangle \mathbf{x}^i_{k+1} \tag{68}$$

$$= \langle (\ell_1 \star \ell_2) \otimes_{(2)} e_i, \Phi(\mathbf{x})_L\rangle. \tag{69}$$

□

D DETAILS ON ALGORITHMS

Here we give further information on the implementation of LS2T layers detailed in the main text. For simplicity, we fix the state-space of sequences to be $V = \mathbb{R}^d$ from here onwards.

D.1 VARIATIONS

Truncation degree. To reiterate from Section 2, for a given static feature map $\phi : \mathbb{R}^d \to V$ the Seq2Tens feature map $\Phi : \mathrm{Seq}(\mathbb{R}^d) \to \mathrm{T}(V)$ represents a sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(\mathbb{R}^d)$ as a tensor in $T(V)$,

$$\Phi(\mathbf{x}) = (\Phi_m(\mathbf{x}))_{m \geq 0}, \quad \Phi_m(\mathbf{x}) = \sum_{1 \leq i_1 < \cdots < i_m \leq L} \phi(\mathbf{x}_{i_1}) \otimes \cdots \otimes \phi(\mathbf{x}_{i_m}), \tag{70}$$

where $\Phi_m : \mathrm{Seq}(\mathbb{R}^d) \to V^{\otimes m}$ is given by a summation over all noncontiguous length-m subsequences of $\mathbf{x}$ with non-repeating indices. Therefore, for a sequence of length $L \in \mathbb{N}$, Φ_m can have

potentially non-zero terms for $m \leq L$. An empirical observation is that for most datasets computing everything up to the Lth level is redundant in the sense that usually the first $M \in \mathbb{N}$ levels already contain most of the information a discriminative or a generative model picks up on where $M \ll L$. It is thus better treated as a hyperparameter, which we call "order" in the main text.

Below we take for brevity $\phi(\mathbf{x}_i) = \mathbf{x}_i \in V = \mathbb{R}^d$ since with other maps ϕ simply amounts to replacing $\mathbf{x}_i \in \mathbb{R}^d$ by $\phi(\mathbf{x}_i) \in \mathbb{R}^e$.

Separating functionals across levels. Now let us consider the low-rank Seq2Tens map $\tilde{\Phi}_\theta$, each output coordinate of which is given by a linear functional of Φ, i.e. $\tilde{\Phi}_\theta(\mathbf{x}) = \big(\langle \ell^1, \Phi(\mathbf{x})\rangle, \ldots, \langle \ell^n, \Phi(\mathbf{x}), \rangle\big)$ for a sequence $\mathbf{x} \in \mathrm{Seq}(\mathbb{R}^d)$ and a collection of rank-1 weight tensors $\theta = (\ell^k)_{k=1}^n \subset T(\mathbb{R}^d)$. Then, a single output coordinate of $\tilde{\Phi}_\theta$ may be written for $1 \leq k \leq n$ as

$$\langle \ell^k, \Phi(\mathbf{x})\rangle = \sum_{m=0} \langle \ell^k_m, \Phi_m(\mathbf{x})\rangle, \tag{71}$$

for $\ell^k = (\ell^k_m)_{m\geq 0}$, i.e. we take inner products of tensors that are on the same level of the tensor algebra, and then sum them all up. We found that rather than taking the summation across tensor levels, it is beneficial to treat the linear functional on each level as an independent output to have

$$\tilde{\Phi}_{m,\theta}(\mathbf{x}) = (\langle \ell^1_m, \Phi_m(\mathbf{x}), \ldots, \langle \ell^n_m, \Phi_m(\mathbf{x})\rangle) \quad \text{and} \quad \tilde{\Phi}_\theta(\mathbf{x}) = (\tilde{\Phi}_{m,\theta}(\mathbf{x}))_{m\geq 0}, \tag{72}$$

where now $\tilde{\Phi}_\theta$ has output dimensionality $(M \times n)$ with $M \in \mathbb{N}$ the truncation degree of Φ as detailed in the previous paragraph. Hence this modification scales the output dimension by M, but it will be important for the next step we discuss. It is for this modification that in Figure 1, each output of a LS2T layer has dimensionality 64×2, since we set $n = 64$ and $M = 2$, while in Figure 2 the B-LS2T layer has output dimensionality $h \times 4$, since we set $n = h$ and $M = 4$.

The need for normalization. Here we motivate the need to follow each LS2T layer by some form of normalization. Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(\mathbb{R}^d)$ be a sequence. Let $\alpha \in \mathbb{R}$ be a scalar and define $\mathbf{y} = \alpha\mathbf{x} \in \mathrm{Seq}(\mathbb{R}^d)$ a scaled version of $\mathbf{x}$. Let us investigate how the features change:

$$\Phi_m(\mathbf{y}) = \sum_{1\leq i_1 < \cdots < i_M \leq L} \mathbf{y}_{i_1} \otimes \cdots \otimes \mathbf{y}_{i_m} = \sum_{1\leq i_1 < \cdots < i_M \leq L} (\alpha\mathbf{x}_{i_1}) \otimes \cdots \otimes (\alpha\mathbf{x}_{i_m}) \tag{73}$$

$$= \alpha^m \sum_{1\leq i_1 < \cdots < i_M \leq L} \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_m}, \tag{74}$$

and therefore we have $\Phi(\mathbf{y}) = (\Phi_m(\mathbf{y}))_{m\geq 0} = (\alpha^m \Phi_m(\mathbf{x}))_{m\geq 0}$, which analogously translates into the low-rank Seq2Tens map since

$$\tilde{\Phi}_{m,\theta}(\mathbf{y}) = (\langle \ell^1_m, \Phi_m(\mathbf{y})\rangle, \ldots, \langle \ell^n_m, \Phi_m(\mathbf{y})\rangle) = (\langle \ell^1_m, \alpha^m\Phi_m(\mathbf{x})\rangle, \ldots, \langle \ell^n_m, \alpha^m\Phi_m(\mathbf{x})\rangle) \tag{75}$$

$$= \alpha^m(\langle \ell^1_m, \Phi_m(\mathbf{x})\rangle, \ldots, \langle \ell^n_m, \Phi_m(\mathbf{x})\rangle). \tag{76}$$

From this point alone, it is easy to see that Φ_m, and thus $\tilde{\Phi}_{m,\theta}$ will move across wildly different scales for different values of m, which is inconvenient for the training of neural networks. To counterbalance this, we used a batch normalization layer after each LS2T layer in Section 5.1 that computes mean and variance statistics across time and the batch itself, while in Section 5.3 we used a layer normalization that computes the statistics only across time.

Sequence differencing. In both Section 5.1 and Section 5.3, we precede each LS2T layer by a differencing layer and a time-embedding layer.

Let $\Delta : \mathrm{Seq}(\mathbb{R}^d) \to \mathrm{Seq}(\mathbb{R}^d)$ be the discrete difference operator defined for a sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(\mathbb{R}^d)$ as

$$\Delta\mathbf{x} := (\mathbf{x}_1, \mathbf{x}_2 - \mathbf{x}_1, \ldots, \mathbf{x}_L - \mathbf{x}_{L-1}) \in \mathrm{Seq}(\mathbb{R}^d), \tag{77}$$

where we made the simple identification that $\mathbf{x}_0 \equiv 0$, i.e. for all sequences we first concatenate a $\mathbf{0}$ observation along the time axis. This step is beneficial for two reasons: (i) now Δ preserves the length L of a sequence, (ii) now Δ is one-to-one, since otherwise Δ would be translation invariant, i.e. it would map all sequences which are translations of each other to the same output sequence.

To motivate differencing, first let us consider $\Phi_m(\mathbf{x})$ for $m = 1$, and for brevity denote $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$ for $i = 1, \dots, L$ and the convention $\mathbf{x}_0 = 0$. Then, we may write

$$\Phi_1(\mathbf{x}) = \sum_{i=1}^{L} \Delta\mathbf{x}_i = \mathbf{x}_L, \tag{78}$$

which means that now the first level of the Seq2Tens map is simply point-wise evaluation at the last observation time, and when used as a sequence-to-sequence transformation over expanding windows (i.e. equation 15), it is simply the identity map of the sequence.

Analogously, for the low-rank map we have

$$\tilde{\Phi}_{1,\theta}(\mathbf{x}) = \sum_{i=1}^{L} (\langle \ell_1^1, \Delta\mathbf{x}_i \rangle, \dots, \langle \ell_1^n, \Delta\mathbf{x}_i \rangle) = (\langle \ell_1^1, \mathbf{x}_L \rangle, \dots, \langle \ell_1^n, \mathbf{x}_L \rangle), \tag{79}$$

which is simply a linear map applied to $\mathbf{x}$ in an observation-wise manner. The higher order terms, $\Phi_m(\mathbf{x})$ and $\tilde{\Phi}_{m,\theta}(\mathbf{x})$ can generally be written as

$$\Phi_m(\mathbf{x}) = \sum_{1 \le i_1 < \dots i_m \le L} \Delta\mathbf{x}_{i_1} \otimes \dots \otimes \Delta\mathbf{x}_{i_m}, \tag{80}$$

and

$$\tilde{\Phi}_{m,\theta} = \sum_{1 \le i_1 < \dots < i_m \le L} (\langle \mathbf{z}_{m,1}^1, \Delta\mathbf{x}_{i_1} \rangle \cdots \langle \mathbf{z}_{m,m}^1, \Delta\mathbf{x}_{i_m} \rangle, \dots, \langle \mathbf{z}_{m,1}^n, \Delta\mathbf{x}_{i_1} \rangle \cdots \langle \mathbf{z}_{m,m}^n, \Delta\mathbf{x}_{i_m} \rangle) \tag{81}$$

for some rank-1 degree-m tensors $\ell_m^k = \mathbf{z}_{m,1}^k \otimes \dots \otimes \mathbf{z}_{m,m}^k$ for $k = 1, \dots, n$. We observed that this way the higher order terms are relatively stable across time as the length of a sequence increases, while without differencing they can become unstable, exhibit high oscillations, or simply blow-up.

An additional benefit of taking differences is that the maps Φ and $\tilde{\Phi}_\theta$ become padding invariant, that is, invariant to repetitions of elements. It is easy to see this by checking that if $\mathbf{x}_i = \mathbf{x}_{i-1}$ then $\Delta\mathbf{x}_i = \mathbf{0}$ and all the corresponding terms in the summations equation 80 and equation 81 are zeros. This makes sequences of different length comparable without any explicit need for masking, by simply pre- or post-padding each sequence with repetitions of the first/last observation.

Time-embeddings. By time-embedding, we mean adding as an extra coordinate to an input sequence the observation times $(t_i, \mathbf{x}_i)_{i=1,\dots,L} \in \mathrm{Seq}(\mathbb{R}^{d+1})$. Some datasets already come with a pre-specified observation-grid, in which case we can use that as a time-coordinate at every use of a time-embedding layer. If there is no pre-specified observation grid, we can simply add a normalized and equispaced coordinate, i.e. $t_i = i/L$.

Time-embeddings can be beneficial preceding both convolutional layers and LS2T layers. For convolutions, it allows to learn features that are not translation invariant (Liu et al., 2018a). For the LS2T layer, the interpretation is slightly different. Note that in both Sections 5.1 and 5.3, we employ the time-embedding before the differencing block. This can be equivalently reformulated as after differencing adding an additional constant coordinate to the sequence, i.e. $(t_i - t_{i-1}, \mathbf{x}_i - \mathbf{x}_{i-1})_{i=1,\dots,L} \in \mathrm{Seq}(\mathbb{R}^{d+1})$, where $t_i - t_{i-1} = 1/L$ is simply a constant. This is motivated by Lemma B.8, which states that the map $\Phi : \mathrm{Seq}(\mathbb{R}^d) \to T(V)$ is injective for sequences with a constant coordinate. Thus, the time-embedding before the differencing block is equivalent to adding a constant coordinate after the differencing block, and its purpose is to guarantee injectivity of Φ.

Delay embeddings and convolutions. A useful preprocessing technique for time series are delay embeddings, which simply amount to augmenting the state-space of sequences with a certain number of previous observations, motivated by Takens' theorem (Takens, 1981; Sauer et al., 1991), which

ensures that a high-dimensional dynamical system can be reconstructed from low-dimensional observations. Theorem 2.1 guarantees that if ϕ is a universal feature map on the state-space then Φ is universal. The most straightforward nonlinearity to take as ϕ is a multilayer perceptron (MLP), that is, $\phi = \phi_D \circ \cdots \circ \phi_1$ with $\phi_j(\mathbf{x}) = \sigma(\mathbf{W}_j \mathbf{x} + \mathbf{b}_j)$. By combining such dense layers with delay embeddings (lags), one recovers a temporal convolution layer, i.e. $\phi_j(\mathbf{x}_i^l) = \sigma\left(\sum_{k=0}^{l} \mathbf{W}_{j,k}\mathbf{x}_{i-k} + \mathbf{b}_j\right)$, which motivates the use of convolutions in the preprocessing layers.

D.2 Recursive computations

Next, we show how the computation of the maps Φ_m and $\Phi_{m,\theta}$ can be formulated as a joint recursion over the tensor levels and the sequence itself.

Since Φ_m is given by a summation over all noncontiguous length-m subsequences with non-repetitions of a sequence $\mathbf{x} \in \mathrm{Seq}(\mathbb{R}^d)$, simple reasoning shows that Φ_m obeys the recursion across m and time for $2 \leq l \leq L$ and $1 \leq m$

$$\Phi_m(\mathbf{x}_1, \ldots \mathbf{x}_l) = \Phi_m(\mathbf{x}_1, \ldots \mathbf{x}_{l-1}) + \Phi_{m-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \otimes \mathbf{x}_l, \tag{82}$$

with the initial conditions $\Phi_0 \equiv 1$, $\Phi_1(\mathbf{x}_1) = \mathbf{x}_1$ and $\Phi_m(\mathbf{x}_1) = \mathbf{0}$ for $m \geq 2$.

Let $\ell = (\ell_m)_{m \geq 0} \in T(\mathbb{R}^d)$ be a sequence of rank-1 tensors with $\ell_m = \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m} \in (\mathbb{R}^d)^{\otimes m}$ a rank-1 tensor of degree-m. Then, $\langle \ell_m, \Phi_m(\mathbf{x}) \rangle$ may be computed analogously to equation 82 using the recursion for $2 \leq l \leq L$, $1 \leq m$

$$\begin{aligned}
\langle \ell_m, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_l) \rangle =& \langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m}, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_l) \rangle && (83)\\
=& \langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m}, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle && (84)\\
&+ \langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m-1}, \Phi_{m-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle \langle \mathbf{z}_{m,m}, \mathbf{x}_l \rangle && (85)\\
=& \langle \ell_m, \Phi_m(\mathbf{x}_1, \ldots \mathbf{x}_{l-1}) \rangle && (86)\\
&+ \langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m-1}, \Phi_{m-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle \langle \mathbf{z}_{m,m}, \mathbf{x}_l \rangle && (87)
\end{aligned}$$

and the initial conditions can be rewritten as the identities $\langle z_{0,0}, \Phi_0 \rangle = 1$, $\langle \mathbf{z}_{m,1}, \Phi_1(\mathbf{x}) \rangle = \langle \mathbf{z}_{m,1}, \mathbf{x} \rangle$ and $\langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m}, \Phi_m(\mathbf{x}_1) \rangle = \mathbf{0}$ for $2 \leq m$.

A slight inefficiency of the previous recursion given in Lines equation 86, equation 87 is that one generally cannot substitute $\langle \ell_{m-1}, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle$ for the $\langle \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m-1}, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle \rangle$ term in Line equation 87, since $\ell_{m-1} \neq \mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m-1}$ generally. This means that to construct the degree-m linear functional $\langle \ell_m, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_l) \rangle$, one has to start from scratch by first constructing the degree-1 term $\langle \mathbf{z}_{m,1}, \Phi_1 \rangle$ first, then the degree-2 term $\langle \mathbf{z}_{m,1} \otimes \mathbf{z}_{m,2}, \Phi_2 \rangle$, and so forth. This further means in terms of complexities that while Formula equation 82 has linear complexity in the largest value of m, henceforth denoted by $M \in \mathbb{N}$, Formula equation 86, equation 87 has a quadratic complexity in M due to the non-recursiveness of the rank-1 tensors $(\ell_m)_m = (\mathbf{z}_{m,1} \otimes \cdots \otimes \mathbf{z}_{m,m})_m$.

The previous observation indicates that an even more memory and time efficient recursion can be devised by parametrizing the rank-1 tensors $(\ell_m)_m$ in a recursive way as follows: let $\ell_1 = \mathbf{z}_1 \in \mathbb{R}^d$ and define $\ell_m = \ell_{m-1} \otimes \mathbf{z}_m \in (\mathbb{R}^d)^{\otimes m}$ for $2 \leq m$, i.e. $\ell_m = \mathbf{z}_1 \otimes \cdots \otimes \mathbf{z}_m$ for $\{\mathbf{z}_1, \ldots \mathbf{z}_m\} \subset \mathbb{R}^d$. This parametrization indeed allows to substitute ℓ_{m-1} in Line equation 87, which now becomes

$$\langle \ell_m, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_l) \rangle = \langle \ell_m, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1} \rangle + \langle \ell_{m-1}, \Phi_{m-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{l-1}) \rangle \langle \mathbf{z}_m, \mathbf{x}_l \rangle, \tag{88}$$

and hence, due to the recursion across m for both ℓ_m and Φ_m, it is now linear in the maximal value of m, denoted by $M \in \mathbb{N}$. This results in a less flexible, but more efficient variant of Seq2Tens, due to the additional added recursivity constraint on the rank-1 weight tensors. We refer to this version as the *recursive variant*, while to the non-recursive construction as the *independent variant*.

Next, we show how the previous computations can be rewritten as a simple RNN-like discrete dynamical system. For simplicity, we consider the recursive formulation, but the independent variant can also be formulated as such with a larger latent state size. Let $(\ell^j)_{j=1,\ldots,n}$ be $n \in \mathbb{N}$ different rank-1 recursive weight tensors, i.e. $\ell^j = (\ell_m^j)_{m \geq 0}$, $\ell_m^j = \mathbf{z}_1^j \otimes \ldots \mathbf{z}_m^j$ for $\mathbf{z}_m^j \in \mathbb{R}^d$, $m \geq 0$ and $j = 1, \ldots, n$. Also, denote $h_{m,i}^j := \langle \ell_m^j, \Phi_m(\mathbf{x}_1, \ldots, \mathbf{x}_i) \rangle \in \mathbb{R}$, a scalar corresponding to the output of the jth linear functional on the mth tensor level for the sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$. We collect all such functionals for given m and i into $\mathbf{h}_{m,i} := (h_{m,i}^1, \ldots h_{m,i}^n) \in \mathbb{R}^n$, i.e. $\mathbf{h}_{m,i} = \tilde{\Phi}_{m,\theta}(\mathbf{x}_1, \ldots, \mathbf{x}_i)$.

Algorithm 1 Computing the LS2T layer with independent tensors across levels

Input: Sequences $(\mathbf{x}^j)_{j=1,\dots,n_{\mathbf{x}}} = (\mathbf{x}_1^j, \dots, \mathbf{x}_L^j)_{j=1,\dots,n_{\mathbf{x}}} \subset \mathrm{Seq}(\mathbb{R}^d)$, rank-1 tensors $(\ell^k)_{k=1,\dots,n_\ell} = (\mathbf{z}_{m,1}^k \otimes \cdots \otimes \mathbf{z}_{m,m}^k)_{m=1,\dots,M}^{k=1,\dots,n_\ell} \subset T(\mathbb{R}^d)$, truncation degree $M \in \mathbb{N}$
Compute $M[m,i,j,l,k] \leftarrow \langle \mathbf{z}_{m,k}^j, \mathbf{x}_l^i \rangle$ for $m \in \{1,\dots,M\}$, $i \in \{1,\dots,n_{\mathbf{x}}\}$, $j \in \{1,\dots,n_\ell\}$, $l \in \{1,\dots,L\}$ and $k \in \{1,\dots,m\}$
for $m = 1$ **to** M **do**
  Assign $R \leftarrow M[m,:,:,:,1]$
  for $k = 2$ **to** m **do**
    Iterate $R \leftarrow M[m,:,:,:,k] \odot R[:,:,\boxplus + 1]$
  end for
  Save $Y_m \leftarrow \cdot R[:,:,\boxplus]$
end for
Output: Sequences $(Y_1, \dots, Y_M)$ each of shape $(n_{\mathbf{x}} \times L \times n_\ell)$

Algorithm 2 Computing the LS2T layer with recursive tensors across levels

Input: Sequences $(\mathbf{x}^j)_{j=1,\dots,n_{\mathbf{x}}} = (\mathbf{x}_1^j, \dots, \mathbf{x}_L^j)_{j=1,\dots,n_{\mathbf{x}}} \subset \mathrm{Seq}(\mathbb{R}^d)$, rank-1 tensors $(\ell^k)_{k=1,\dots,n_\ell} = (\mathbf{z}_1^k \otimes \cdots \otimes \mathbf{z}_m^k)_{m=1,\dots,M}^{k=1,\dots,n_\ell} \subset T(\mathbb{R}^d)$, truncation degree $M \in \mathbb{N}$
Compute $M[m,i,j,l] \leftarrow \langle \mathbf{z}_m^j, \mathbf{x}_l^i \rangle$ for $m \in \{1,\dots,M\}$, $i \in \{1,\dots,n_{\mathbf{x}}\}$, $j \in \{1,\dots,n_\ell\}$ and $l \in \{1,\dots,L\}$
Assign $R \leftarrow M[1,:,:,:]$
Save $Y_1 \leftarrow R[:,:,\boxplus]$
for $m = 2$ **to** M **do**
  Update $R \leftarrow M[m,:,:,:] \odot R[:,:,\boxplus + 1]$
  Save $Y_m \leftarrow R[:,:,\boxplus]$
end for
Output: Sequences $(Y_1, \dots, Y_M)$ each of shape $(n_{\mathbf{x}} \times L \times n_\ell)$

Additionally, we collect all weight vectors $\mathbf{z}_m^j \in \mathbb{R}^d$ for a given $m \in \mathbb{N}$ into the matrix $\mathbf{Z}_m := (\mathbf{z}_m^1, \dots, \mathbf{z}_m^n)^\top \in \mathbb{R}^{n \times d}$. Then, we may write the following vectorized version of equation equation 88:

$$\mathbf{h}_{1,i} = \mathbf{h}_{1,i-1} + \mathbf{Z}_1 \mathbf{x}_i, \tag{89}$$

$$\mathbf{h}_{m,i} = \mathbf{h}_{m,i-1} + \mathbf{h}_{m-1,i-1} \odot \mathbf{Z}_m \mathbf{x}_i \quad \text{for } m \geq 2, \tag{90}$$

with the initial conditions $\mathbf{h}_{m,0} = \mathbf{0} \in \mathbb{R}^n$ for all $m \geq 1$, and $\odot$ denoting the Hadamard product.

D.3 Algorithms

We have shown previously that one may compute $\Phi_\theta(\mathbf{x}_1, \dots, \mathbf{x}_i) = (\Phi_{m,\theta}(\mathbf{x}_1, \dots, \mathbf{x}_i))_{m \geq 0}$ recursively in a vectorized way for a given sequence $(\mathbf{x}_1, \dots, \mathbf{x}_i) \in \mathrm{Seq}(\mathbb{R}^d)$. Now, in Algorithms 1 and 2, we additionally show how to further vectorize the previous computations across time and the batch. For this purpose, let $(\mathbf{x}^j)_{j=1,\dots,n_{\mathbf{X}}} \subset \mathrm{Seq}(\mathbb{R}^d)$ be $n_{\mathbf{X}} \in \mathbb{N}$ sequences in $\mathbb{R}^d$ and $(\ell^k)_{k=1,\dots,n_\ell} \subset T(\mathbb{R}^d)$ be n_ℓ be rank-1 tensors in $T(\mathbb{R}^d)$.

Additionally, we adopted the notation for describing algorithms from Király & Oberhauser (2019). For arrays, 1-based indexing is used. Let A and B be k-dimensional arrays with size $(n_1 \times \cdots \times n_k)$, and let $i_j \in \{1, \dots, n_j\}$ for $j \in \{1, \dots, k\}$. Then, the following operations are defined:

(i) The cumulative sum along axis j:

$$A[\dots,:,\boxplus,:,\dots][\dots,i_{j-1},i_j,,i_{j+1}\dots] := \sum_{\kappa=1}^{i_j} A[\dots,i_{j-1},\kappa,i_{j+1},\dots].$$

Table 4: Forward pass computation time in seconds on a Gefore 2080Ti GPU for varying sequence length L, fixed batch size $N = 32$, state-space dimension $d = 64$ and output dimension $h = 64$.

L	Conv1D	LSTM	S2T			S2T-R		
			$M = 2$	$M = 6$	$M = 10$	$M = 2$	$M = 6$	$M = 10$
32	8.1×10^{-4}	1.2×10^{-1}	1.7×10^{-3}	4.5×10^{-3}	9.9×10^{-3}	1.7×10^{-3}	2.5×10^{-3}	3.4×10^{-3}
64	8.5×10^{-4}	2.3×10^{-1}	1.8×10^{-3}	4.5×10^{-3}	9.9×10^{-3}	1.8×10^{-3}	2.6×10^{-3}	3.4×10^{-3}
128	9.7×10^{-4}	4.6×10^{-1}	2.1×10^{-3}	4.9×10^{-3}	1.0×10^{-2}	2.1×10^{-3}	2.9×10^{-3}	3.8×10^{-3}
256	1.1×10^{-3}	9.3×10^{-1}	2.4×10^{-3}	5.2×10^{-3}	1.1×10^{-2}	2.4×10^{-3}	3.2×10^{-3}	4.0×10^{-3}
512	1.3×10^{-3}	1.8×10^{0}	3.2×10^{-3}	6.0×10^{-3}	1.1×10^{-2}	3.0×10^{-3}	4.0×10^{-3}	4.8×10^{-3}
1024	1.9×10^{-3}	3.7×10^{0}	4.4×10^{-3}	7.1×10^{-3}	1.2×10^{-2}	4.4×10^{-3}	5.1×10^{-3}	6.0×10^{-3}

(ii) The slice-wise sum along axis j:

$$A[\dots, :, \Sigma, :, \dots][\dots, i_{j-1}, i_{j+1}, \dots] := \sum_{\kappa=1}^{n_j} A[\dots, i_{j-1}, \kappa, i_{j+1}, \dots].$$

(iii) The shift along axis j by $+m$ for $m \in \mathbb{N}$:

$$A[\dots, :, +m, :, \dots][\dots, i_{j-1}, i_j, i_{j+1}, \dots] := \begin{cases} A[\dots, i_{j-1}, i_j - m, i_{j+1}, \dots], & \text{if } i_j > m, \\ 0, & \text{if } i_j \leq m. \end{cases}$$

(iv) The Hadamard product of arrays A and B:

$$(A \odot B)[i_1, \dots, i_k] := A[i_1, \dots, i_k] \cdot B[i_1, \dots, i_k].$$

D.4 Complexity analysis

We give a complexity analysis of Algorithms 1 and 2. Inspection of Algorithm 1 says that it has $O(M^2 \cdot n_{\mathbf{x}} \cdot L \cdot n_\ell)$ complexity in both time and memory with an additional memory cost of storing the $O(M^2 \cdot n_\ell \cdot d)$ number of parameters, the rank-1 weight tensors $(\ell_m^k)_m$, which are stored in terms of their components $\mathbf{z}_{m,j}^k \in \mathbb{R}^d$. In contrast, Algorithm 1 has a time and memory cost of $O(M \cdot n_{\mathbf{x}} \cdot L \cdot n_\ell)$, thus linear in M, and the recursive rank-1 weight tensors are now only an additional $O(M \cdot n_\ell \cdot d)$ number of parameters.

Additionally to the big-O bounds on complexities, another important question is how well the computations can be parallelized, which can have a larger impact on computations when e.g. running on GPUs. Observing the algorithms, we can see that they are not *completely* parallelizable due to the cumsum ($\boxplus$) operations in Lines 6, 8 (Algorithm 1) and Lines 4, 6 (Algorithm 2). The cumulative sum operates recursively on the whole time axis, therefore it is not parallelizable, but can be computed very efficiently on modern architectures.

To gain further intuition about what kind of performance one can expect for our LS2T layers, we benchmarked the computation time of a forward pass for varying sequence lengths and varying hyperparameters of the model. For comparison, we ran the same experiment with an LSTM layer and a Conv1D layer with a filter size of 32. The input is a batch of sequences of shape $(n_{\mathbf{X}} \times L \times d)$, while the output has shape $(n_{\mathbf{X}} \times L \times h)$, where $d \in \mathbb{N}$ is the state-space dimension of the input sequences, while $h \in \mathbb{N}$ is simply the number of channels or hidden units in the layer. For our layers, we used our own implementation in Tensorflow, while for LSTM and Conv1D, we used the Keras implementation using the Tensorflow backend.

In Table 4, we report the average computation time of a forward pass over 100 trials, for fixed batch size $n_{\mathbf{X}} = 32$, state-space dimension $d = 64$, output dimension $h = 64$ and varying sequence lengths $L \in \{32, 64, 128, 256, 512, 1024\}$. LS2T and S2T-R respectively refer to the independent and recursive variants, and $M \in \mathbb{N}$ denotes the truncation degree. We can observe that while the LSTM practically scales linearly in L, the scaling of LS2T is sublinear for all practical purposes, exhibiting a growth rate that is more close to that of the Conv1D layer, that is fully parallelizable. Specifically, while the LSTM takes 3.7 seconds to make a forward pass for $L = 1024$, all variants of the LS2T layer take less time than that *by a factor of at least a 100*. This suggests that its

computations are *highly parallelizable* across time. Additionally, we observe that LS2T exhibits a more aggressive growth rate with respect to the parameter M due to the quadratic complexity in M (although the numbers show only linear growth), while S2T-R scales very favourably in M as well due to the linear complexity (the results indicate a sublinear growth rate).

D.5 Initialization

Below we detail the initialization procedure used by our models for the parameters θ of the LS2T layer, where $\theta = (\ell^k)_{k=1}^{n_\ell} \subset T(\mathbb{R}^d)$. As before, each $\ell^k = (\ell^k_m)_{m\geq 0} \in T(\mathbb{R}^d)$ is given as a sequence of rank-1 tensors, such that $\ell^k_m = \mathbf{z}^k_{m,1} \otimes \cdots \otimes \mathbf{z}^k_{m,m}$ with $\mathbf{z}^k_{m,j} \in \mathbb{R}^d$ for the independent variant, while $\ell^k_m = \mathbf{z}^k_1 \otimes \cdots \otimes \mathbf{z}^k_m$ with $\mathbf{z}_m \in \mathbb{R}^d$ for the recursive variant. Hence, by initialization, we mean the initialization of the components $\mathbf{z}^k_{m,j}$ or $\mathbf{z}^k_m$.

To find a satisfactory initialization scheme, we took as starting point the Glorot (Glorot & Bengio, 2010) initialization, which specifies that for a layer with input dimension n_{in} and output dimension n_{out}, the weights should be independently drawn from a centered distribution with variance $2/(n_{in}+n_{out})$, where the distribution that is used is usually a uniform or a Gaussian.

Independent variant. We first consider the independent variant of the algorithm. The weights are given as the rank-1 tensors

$$\ell^k_m = \mathbf{z}^k_{m,1} \otimes \cdots \otimes \mathbf{z}^k_{m,m} \in (\mathbb{R}^d)^{\otimes m} \quad \text{for} \quad k = 1, \ldots, n_\ell \quad \text{and} \quad m \geq 0. \tag{91}$$

Denote $\mathbf{z}^k_{m,j} = (z^k_{m,j,1}, \ldots, z^k_{m,j,d}) \in \mathbb{R}^d$, and assume that for a given $m \in \mathbb{N}$ that each of $z^k_{m,j,p}$ are drawn independently from some distribution with

$$\mathbb{E}[z^k_{m,j,p}] = 0 \quad \text{and} \quad \mathbb{E}[z^k_{m,j,p}]^2 = \sigma^2_m \quad \text{for} \quad j = 1, \ldots, m \quad \text{and} \quad p = 1, \ldots, d. \tag{92}$$

Then, for a given multi-index $\mathbf{i} = (i_1, \ldots, i_m) \in \{1, \ldots, d\}^m$, the $\mathbf{i}$th coordinate of ℓ^k_m is given as

$$\ell^k_{m,\mathbf{i}} = z^k_{m,1,i_1} \cdots z^k_{m,m,i_m} \tag{93}$$

and has as its first two moments

$$\mathbb{E}[\ell^k_{m,\mathbf{i}}] = 0 \quad \text{and} \quad \mathbb{E}[\ell^k_{m,\mathbf{i}}]^2 = \sigma^{2m}_m \tag{94}$$

due to the independence of the corresponding terms in the product. Therefore, to have $\mathbb{E}[\ell^k_{m,\mathbf{i}}]^2 = 2/(d^m + n_\ell)$, where we made the substitutions $n_{in} = d^m$ and $n_{out} = n_\ell$, we can simply set

$$\sigma^2_m = \sqrt[m]{\frac{2}{d^m + n_\ell}}. \tag{95}$$

Recursive variant. In the recursive variant, the weights themselves are constructed recursively as

$$\ell^k_m = \mathbf{z}^k_1 \otimes \cdots \otimes \mathbf{z}^k_m \quad \text{for} \quad k = 1, \ldots, n_\ell \quad \text{and} \quad m \geq 0. \tag{96}$$

Thus, for $\mathbf{i} = (i_1, \ldots, i_m) \in \{1, \ldots, d\}^m$, the $\mathbf{i}$th component of ℓ^k_m is given as

$$\ell^k_{m,\mathbf{i}} = z^k_{1,i_1} \cdots z^k_{m,i_m}, \tag{97}$$

and if we assume that for a given $m \in \mathbb{N}$, z^k_{m,i_m} is drawn from a centered distribution with variance σ^2_m, then we have

$$\mathbb{E}[\ell^k_{m,\mathbf{i}}] = 0 \quad \text{and} \quad \mathbb{E}[\ell^k_{m,\mathbf{i}}]^2 = \sigma^2_1 \cdots \sigma^2_m, \tag{98}$$

which means that now our goal is to have $\sigma^2_1 \cdots \sigma^2_m = 2/(d^m + n_\ell)$ for all $m \geq 1$, which is achievable inductively by

$$\sigma^2_1 = \frac{2}{d + n_\ell} \quad \text{and} \quad \sigma^2_{m+1} = \frac{d^m + n_\ell}{d^{m+1} + n_\ell} \quad \text{for} \quad m \geq 1. \tag{99}$$

For both variants of the algorithms, we used the above described initialization schemes, where the tensor component $\mathbf{z}$'s were drawn from a centered uniform or a Gaussian distribution with the specified variances. Although the resulting weight tensors ℓ^k_m were of neither distribution, they had

the pre-specified first two moments, that seemed sufficient to successfully train models with LS2T layers when combined with succeeding normalization layers as described in Appendix D.1.

However, we remark that when Glorot & Bengio (2010) derived their weight initialization scheme, they considered a fully linear regime across layers, while for $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_L) \in \mathrm{Seq}(\mathbb{R}^d)$, the features $\Phi_m(\mathbf{x}_1, \ldots \mathbf{x}_L)$ on which the weight tensors ℓ_m^k act will be highly nonlinear for any given $m \geq 2$ with increasing levels of nonlinearity for larger values of m. Therefore, it is highly likely that better initialization schemes can be derived by studying the distribution of $\Phi_m(\mathbf{x})$, that might also make it possible to abandon the normalization layers succeeding the LS2T layers and still retain the layers' ability to learn relevant features of the data. Alternatively, data dependent initializations could also prove useful here, such as the LSUV initialization (Mishkin & Matas, 2015).

E Details on experiments

In the following, we give details on the time series classification (Appendix E.1) and sequential data imputation (Appendix E.2) experiments. For running all experiments, we used GPU-based computations on a set of computing nodes, that were equipped with 11 NVIDIA GPUs in total: 4 Tesla K40Ms, 5 Geforce 2080 TIs and 2 Quadro GP100 GPUs.

E.1 Time series classification

Problem formulation. Classification is a traditional task in discriminative supervised machine learning: let $\mathcal{X}$ be the data space and $\mathcal{Y} = \{1, \ldots, c\}$ the discrete output space that consists of only categorical values with $c \in \mathbb{N}$ the total number of classes. The problem is then to predict the corresponding labels of a set of unlabelled examples $\mathbf{X}^\star = (\mathbf{x}_i^\star)_{i=1}^{n_{\mathbf{X}^\star}}$ given a set of labelled examples $(\mathbf{X}, \mathbf{y}) = (\mathbf{x}_i, y_i)_{i=1}^{n_{\mathbf{X}}} \subseteq \mathcal{X} \times \mathcal{Y}$. In the context of time series classification (TSC), the data space $\mathcal{X} = \mathrm{Seq}(\mathbb{R}^d)$ is the space of multivariate sequences, i.e. $\mathbf{x}_i = (\mathbf{x}_{i,j})_{j=1}^{l_{\mathbf{x}_i}}$ where $l_{\mathbf{x}_i} \in \mathbb{N}$ is the length of the sequence $\mathbf{x}_i$ that can change from instance from instance.

Datasets. We use a collection of multivariate TSC datasets available from Baydogan (2015) to benchmark our models as several previous publications report test set accuracies on this archive that we can use as baselines. Table 5 details the datasets from Baydogan (2015) that were used for the TSC experiment. The columns are defined as follows: n_c denotes the number of classes, d the dimension of the state space, $l_{\mathbf{x}}$ the range of sequence lengths, $n_{\mathbf{X}}$ and $n_{\mathbf{X}_\star}$ respectively denote the number of examples in the prespecified training the testing sets. As preprocessing, the state space dimensions were normalized to zero mean and unit variance.

Table 5: Specification of datasets used for benchmarking

Dataset	n_c	d	$l_{\mathbf{x}}$	$n_{\mathbf{X}}$	$n_{\mathbf{X}_\star}$
Arabic Digits	10	13	4–93	6600	2200
AUSLAN	95	22	45–136	1140	1425
Char. Traj.	20	3	109–205	300	2558
CMUsubject16	2	62	127–580	29	29
DigitShapes	4	2	30–98	24	16
ECG	2	2	39–152	100	100
Jap. Vowels	9	12	7–29	270	370
Kick vs Punch	2	62	274–841	16	10
LIBRAS	15	2	45	180	180
NetFlow	2	4	50–997	803	534
PEMS	7	963	144	267	173
PenDigits	10	2	8	300	10692
Shapes	3	2	52–98	18	12
UWave	8	3	315	896	3582
Wafer	2	6	104–198	298	896
Walk vs Run	2	62	128–1918	28	16

Baselines. The benefit of the Baydogan TSC archive (Baydogan, 2015) is that there exist several publications which report the test set results of their respective TSC models, which makes it possible to compare against them without bias in parameter setting. We included all results among the comparison that we are aware of: $\mathrm{DTW_i}$ (Sakoe & Chiba, 1978), ARKernel (Cuturi & Doucet, 2011), SMTS (Baydogan & Runger, 2015a), LPS (Baydogan & Runger, 2015b), gRSF (Karlsson et al., 2016), mvARF (Tuncel & Baydogan, 2018), MUSE (Schäfer & Leser, 2017), MLSTMFCN (Karim et al., 2019). Additionally, to have more deep learning models among the baselines we compare against, we selected the two best performing architectures from Ismail Fawaz et al. (2019), FCN and ResNet. Since the latter two models were tested only on a subset of the multivariate TSC datasets in

Baydogan (2015), we re-ran them, that is, FCN and ResNet on all the datasets with the same settings as they were used previously, which we discuss below.

The FCN is a vanilla CNN model consisting of three temporal convolution layers with layer widths of $128, 256, 128$ and filter sizes of $8, 5, 3$, where each layer is succeeded by a batch normalization before the `relu` activation. We took this simple model as the starting point to upgrade with our own LS2T layers. The ResNet is a more complex model consisting of three similar FCN blocks as previously discussed with residual connections in-between, where the width of the convolutional layers in each FCN block are uniform and respectively are $64, 128, 128$. For more details, refer to Ismail Fawaz et al. (2019). We also discuss the MLSTMFCN (Karim et al., 2019) that we have not ran ourselves, but is closely related to the FCN. The MLSTMFCN is effectively the concatenation of an FCN model as detailed previously with the representation of an LSTM, where the number of units in the LSTM was selected by gridsearch from $\{8, 64, 128\}$. The authors also succeed each convolutional layer in the FCN block by a squeeze-and-excite block, and precede the LSTM layer with a dimension-shuffle block, for more details we refer to Karim et al. (2019). For MLSTMFCN (Karim et al., 2019), we report the same results as the ones in Schäfer & Leser (2017).

Implementation. Our two models, LS2T[3] and FCN-LS2T[3], are structured as depicted on Figure 1, where LS2T[3] only consists of the stacked Seq2Tens block, while FCN-LS2T[3] precedes it with an FCN block. In the FCN block, we also precede each convolutional layer by a time-embedding layer and succeed it by a batch normalization layer and a `relu` activation. In the LS2T[3] block, each LS2T layer is preceded by time embedding and differencing and succeeded by a flattening[2] and a batch normalization layer, as motivated in Appendix D.1. The LS2T layers in both LS2T[3] and FCN-LS2T[3] use the recursive formulation of the low-rank LS2T layers, as detailed in Appendix D.2. We implemented all models in Keras using the Tensorflow blackend, while for the LS2T layers we used our own Tensorflow implementation.

Parameter comparison. Table 6 depicts the median number of parameters for the deep learning models and their absolute deviation. We can see that while LS2T[3] has about half as many parameters as FCN-LS2T[3] due to the added CNN block in the latter, FCN-LS2T[3] still has less parameters than FCN by a factor of 3 as it is a much thinner network. ResNet has the largest number of parameters with about twice as many as FCN. The absolute deviation is relatively small for all as the only source variation is the size of the kernel in the first layer due to varying state space dimensionality. Furthermore, although we did not include MLSTMFCN in the parameter comparison, another deep learning architecture among the baselines, it is straightforward to see that it has at least as many parameters as FCN, since it is given by the concatenation of the FCN model with an LSTM layer.

Table 6: Number of trainable parameters

Model	Trainable parameters	
	Median	Abs. Dev.
LS2T[3]	3.6×10^4	1.6×10^3
FCN-LS2T[3]	7.9×10^4	2.4×10^3
FCN	2.7×10^5	3.7×10^3
ResNet	5.1×10^5	2.5×10^3

Training details. Here we give details on the training methodology used for our models. Our main model, FCN-LS2T[3], is overall a smaller (less parameters), but deeper and thinner network than the other deep learning baselines, such as FCN, ResNet and MLSTMFCN. It is an empirical observation that such models are more difficult to train than shallow and wide architectures. In order to be able to use the same architecture on all datasets without running the risk of overfitting on the smaller datasets, a minibatch size of $n = 4$ was used as an implicit regularizer. As optimizer, we employed the SWATS optimizer[3] (Keskar & Socher, 2017), which starts with Adam (Kingma & Ba, 2015) and switches to vanilla SGD at an automatically determined switchover point, that in our experience provided better results than using Adam just by itself and converged faster and more reliably than SGD. The initial learning rate was set to $\alpha = 1 \times 10^{-3}$, and we employed a learning rate decay of $\beta = 1/\sqrt{2}$ after 50 epochs of no improvement in the training loss, and stopping early after no improvement over 300 epochs, after which the lowest training loss parameter set was used. The baseline models, FCN and ResNet, were trained with the same methodology as in Wang et al. (2017); Ismail Fawaz et al. (2019).

[2]By flattening we mean merging all axes except the batch and time axes.

[3]We used the Keras implementation available at `https://github.com/cainmagi/MDNT`.

Table 7: Classifier accuracies on multivariate time series classification datasets with the highest for each row highlighted in bold.

Dataset	SMTS	LPS	mvARF	DTW	ARKernel	gRSF	MLSTMFCN	MUSE	FCN	ResNet	LS2T^3	FCN-LS2T^3
ArabicDigits	0.964	0.971	0.952	0.908	0.988	0.975	0.990	0.992	0.996(0.001)	**0.997**(0.001)	0.968(0.003)	0.993(0.001)
AUSLAN	0.947	0.754	0.934	0.727	0.918	0.955	0.950	0.970	0.976(0.003)	0.977(0.002)	0.969(0.004)	**0.993**(0.001)
Char. Traj.	0.992	0.965	0.928	0.948	0.900	**0.994**	0.990	0.937	0.991(0.001)	0.990(0.002)	0.957(0.005)	**0.994**(0.001)
CMUsubject16	0.997	**1.000**	**1.000**	0.930	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)
DigitShapes	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)
ECG	0.818	0.820	0.785	0.790	0.820	**0.880**	0.870	**0.880**	0.852(0.023)	0.858(0.015)	0.842(0.004)	0.860(0.021)
Jap. Vowels	0.969	0.951	0.959	0.962	0.984	0.800	**1.000**	0.976	0.986(0.003)	0.990(0.004)	0.979(0.001)	0.980(0.002)
Kick vs Punch	0.820	0.900	0.976	0.600	0.927	**1.000**	0.900	**1.000**	0.960(0.049)	0.780(0.271)	**1.000**(0.000)	**1.000**(0.000)
LIBRAS	0.909	0.903	0.945	0.888	0.952	0.911	**0.970**	0.894	0.967(0.005)	0.964(0.007)	0.773(0.017)	0.957(0.007)
NetFlow	**0.977**	0.968	nan	0.976	nan	0.914	0.950	0.961	0.958(0.005)	0.863(0.043)	0.793(0.135)	0.960(0.004)
PEMS	0.896	0.844	nan	0.832	0.750	**1.000**	nan	nan	0.770(0.021)	0.793(0.024)	0.747(0.015)	0.857(0.013)
PenDigits	0.917	0.908	0.923	0.927	0.952	0.932	**0.970**	0.912	0.968(0.001)	0.964(0.003)	0.954(0.003)	0.953(0.004)
Shapes	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.617(0.100)	0.917(0.091)	**1.000**(0.000)	**1.000**(0.000)
UWave	0.941	**0.980**	0.952	0.916	0.904	0.929	0.970	0.916	0.979(0.001)	0.979(0.001)	0.938(0.003)	0.969(0.002)
Wafer	0.965	0.962	0.931	0.974	0.968	**0.992**	0.990	0.997	0.989(0.002)	0.986(0.004)	0.981(0.002)	0.989(0.001)
Walk vs Run	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)	**1.000**(0.000)
Avg. acc.	0.945	0.933	0.949	0.899	0.938	0.955	**0.970**	0.962	0.938	0.941	0.931	0.969
Med. acc.	0.964	0.964	0.952	0.929	0.952	0.984	0.990	0.976	0.978	0.978	0.969	**0.991**
Sd. acc.	0.059	0.073	0.055	0.111	0.073	0.058	0.039	0.043	0.106	0.076	0.089	0.046
Avg. rank	7.281	7.375	8.214	8.750	7.433	5.500	4.700	5.967	4.906	5.281	6.969	**4.281**
Med. rank	8.000	7.500	7.750	9.000	6.500	6.250	5.500	5.500	4.750	5.250	6.750	**5.000**
Sd. rank	3.082	3.122	2.622	3.022	2.441	3.136	2.419	3.259	2.603	3.104	2.356	1.591

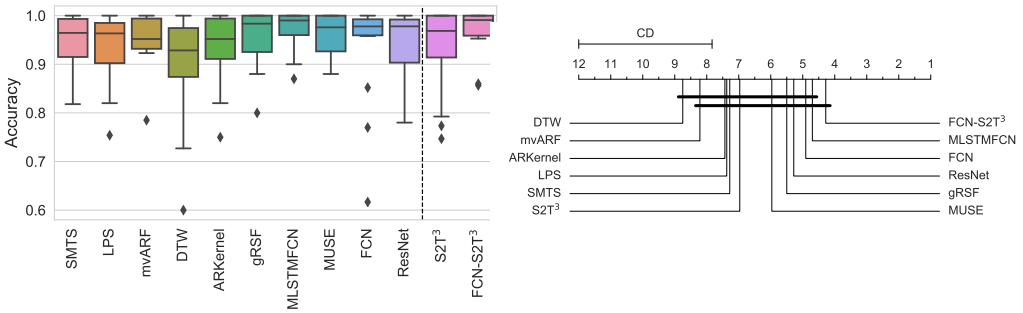

Figure 4: Box-plot of classification accuracies (left) and critical difference diagram (right).

Results. The full table of results are visible in Table 7, where for the models that we have run ourselves (FCN, ResNet, LS2T^3, FCN-LS2T^3), we report the mean and standard deviation over 5 model trains with different seeds. In Section 4, Figure 4 depicts the box-plot distributions of classification accuracies and the corresponding critical difference (CD) diagram. The CD diagram depicts the mean ranks of each method averaged over datasets with a calculated CD region using the Nemenyi test (Nemenyi, 1963). For ranking, an `average` ranking method was used, as opposed to a `dense` ranking, which in a group of ties assigns the average rank within the group. For the Bayesian signed-rank test (Benavoli et al., 2014), we used the implementation from `https://github.com/janezd/baycomp`, and the resulting posterior probabilities are compared in Table 1, Section 4. The posterior distributions themselves are visualized on Figure 5 The region of practical equivalence (rope) was set to $\texttt{rope} = 1 \times 10^{-3}$, that is, two accuracies were practically equivalent if they are at most the given distance from each other. For the visualizations and computation of probabilities, the posteriors were evaluated using $n = 10^5$ Monte Carlo samples.

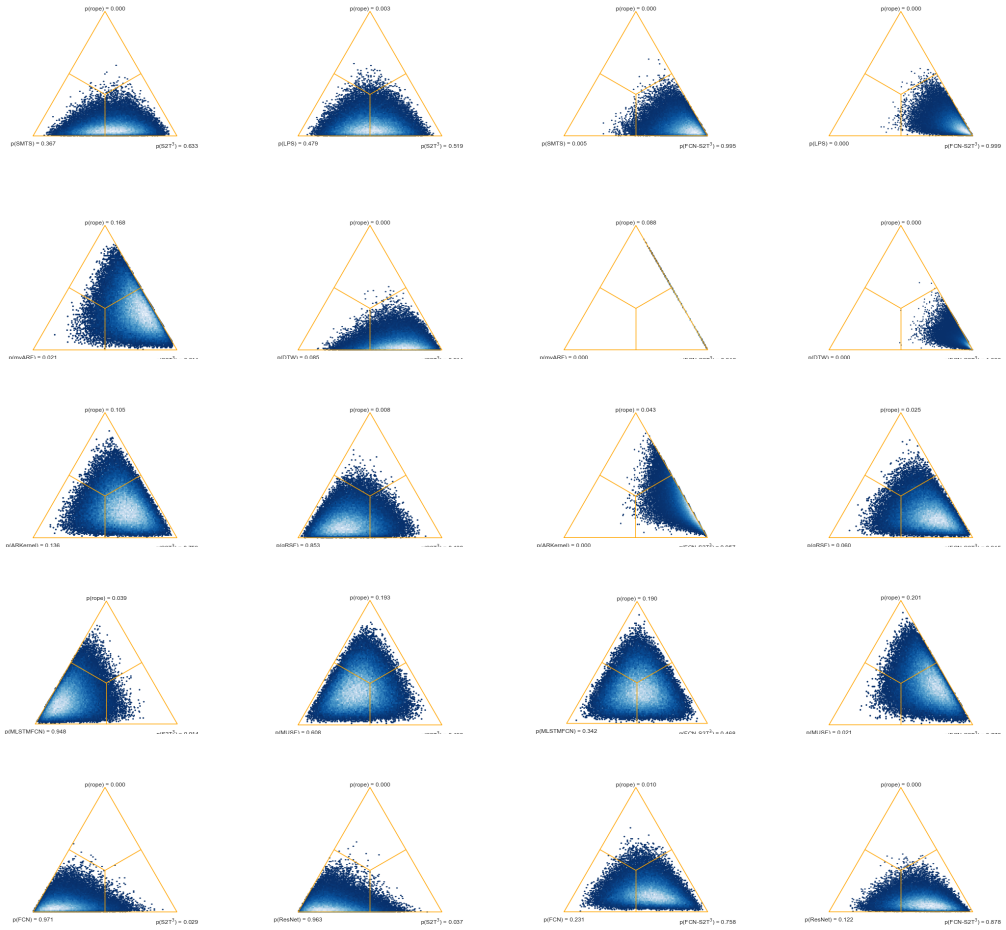

Figure 5: Posterior distribution plots of pairwise Bayesian signed-rank test comparisons

E.2 GENERATIVE SEQUENTIAL DATA IMPUTATION

Problem formulation. Imputation of sequential data can be formulated as a problem of generative unsupervised learning. The input space is given as $\mathcal{X} = \text{Seq}(\mathbb{R}^d)$ and we are given a number of examples $\mathbf{X} = (\mathbf{x}_i)_{i=1}^{n_\mathbf{X}} \subset \text{Seq}(\mathbb{R}^d)$ with $\mathbf{x}_i = (\mathbf{x}_{i,t_j})_{j=1}^{L_i}$. However, now it is not guaranteed that all coordinates if $\mathbf{x}_{i,t_j}$ are observed for a given time-point t_j. Therefore, we are also given for every $\mathbf{x}_i \in \mathbf{X}$ an additional observation mask $\mathbf{m}_i = (\mathbf{m}_{i,t_j})_{j=1}^{L_i} \in \text{Seq}(\{0,1\}^d)$, that specifies whether a given coordinate of $\mathbf{x}_i$ was observed at time t_j or not. The task is then to model the distribution of the unobserved coordinates given the observed coordinates potentially at different time-points.

Model details. We expand on the GP-VAE (Fortuin et al., 2020) model in details. Let $\mathbf{x} = (\mathbf{x}_i)_{i=1,\dots,L} \in \text{Seq}(\mathbb{R}^d)$ be a sequence of length $L \in \mathbb{N}$. The model assumes that $\mathbf{x}$ is noisily generated time-point-wise conditioned on discrete-time realizations of a latent process denoted by $\mathbf{z} = (\mathbf{z}_i)_{i=1,\dots,L} \in \text{Seq}(\mathbb{R}^{d'})$,

$$p_\theta(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i \mid g_\theta(\mathbf{z}_i), \sigma^2 \mathbf{I}_d), \tag{100}$$

where $g_\theta : \mathbb{R}^{d'} \to \mathbb{R}^d$ is the time-point-wise decoder network parametrized by θ, while $\sigma^2 \in \mathbb{R}$ is the observation noise variance. The temporal interdependencies are modelled in the latent space by assigning independent Gaussian process (GP) priors (Williams & Rasmussen, 2006) to the coordinate processes of $\mathbf{z}$, i.e. denoting $\mathbf{z}_i = (z_i^j)_{j=1,\dots,d'} \in \mathbb{R}^{d'}$, it is assumed that $z^j \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$, where $m : \mathbb{R} \to \mathbb{R}$ and $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ are the mean and covariance functions. The authors propose to use the Cauchy kernel as covariance function, defined as

$$k(\tau, \tau') = \tilde{\sigma}^2 \left(1 + \frac{(\tau - \tau')^2}{l^2}\right)^{-1}, \tag{101}$$

which can be seen as an infinite mixture of RBF kernels, allowing one to model temporal dynamics on multiple length scales. For the variational approximation (Blei et al., 2017; Zhang et al., 2018), an amortized Gaussian (Gershman & Goodman, 2014) is used that factorizes across the latent space dimensions, but not across the observation times:

$$q_\psi(\mathbf{z}_1, \dots, \mathbf{z}_L \mid \mathbf{x}_1, \dots \mathbf{x}_L) = q_\psi(z_1^1, \dots z_L^1 \mid \mathbf{x}_1, \dots, \mathbf{x}_L) \cdots, q_\psi(z_1^{d'}, \dots z_L^{d'} \mid \mathbf{x}_1, \dots \mathbf{x}_L) \tag{102}$$

$$= \mathcal{N}(z_1^1, \dots, z_L^1 \mid \mathbf{m}_1, \mathbf{A}_1) \cdots \mathcal{N}(z_1^{d'}, \dots, z_L^{d'} \mid \mathbf{m}_{d'}, \mathbf{A}_{d'}), \tag{103}$$

where $\mathbf{m}_j \in \mathbb{R}^L$ are the posterior means and $\mathbf{A}_j \in \mathbb{R}^{L \times L}$ are the posterior covariance matrices for $j = 1, \dots, d'$. In general, estimating the full covariance matrices $\mathbf{A}_j \in \mathbb{R}^{L \times L}$ from a single data example $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_L) \in \text{Seq}(\mathbb{R}^d)$ is an ill-posed problem. To circumvent the curse of dimensionality in the matrix estimation while allowing for long-range correlations in time, a structured precision matrix representation is used, such that $\mathbf{A}_j^{-1} = \mathbf{B}_j \mathbf{B}_j^\top$ with $\mathbf{B}_j \in \mathbb{R}^{L \times L}$ a lower bidiagonal matrix, such as in Dorta et al. (2018); Blei & Lafferty (2006); Bamler & Mandt (2017), which results in a tridiagonal precision matrix and a potentially dense covariance matrix.

Training across the whole dataset $\mathbf{X} = (\mathbf{x}_i)_{i=1}^{n_\mathbf{X}} \subset \text{Seq}(\mathbb{R}^d)$ is coupled through the decoder and encoder parameters θ and ψ, and the ELBO is computed as

$$\frac{1}{n_\mathbf{X}} \sum_{i=1}^{n_\mathbf{X}} \log p(\mathbf{x}_i) \geq \frac{1}{n_\mathbf{X}} \sum_{i=1}^{n_\mathbf{X}} \Big(\sum_{j=1}^{L_i} \mathbb{E}_{q_\psi(\mathbf{z}_{i,j} \mid \mathbf{x}_i)} [\log p_\theta(\mathbf{x}_{i,j} \mid \mathbf{z}_{i,j})] \tag{104}$$

$$- \beta D_{\text{KL}} q_\psi(\mathbf{z}_i \mid \mathbf{x}_i) p(\mathbf{z}_i)) \Big), \tag{105}$$

where the log-likelihood term is only computed across observed features as was done in Nazabal et al. (2018). Similarly to β-VAEs (Higgins et al., 2017), β is used to rebalance the KL term, now to account for the missingness rate.

Baselines. Additionally to the baseline GP-VAE, the reported baseline results are the same ones as in Fortuin et al. (2020), which are mean imputation, forward imputation, VAE (Kingma & Welling, 2013), HI-VAE (Nazabal et al., 2018) and BRITS (Cao et al., 2018). Among these, the VAE based models are Bayesian and provide a probability measure on possible imputations, while the mean/forward imputation methods and the RNN based BRITS only provide a single imputation.

Datasets. Table 8 details the datasets used, which are the same ones as considered in Fortuin et al. (2020). The columns are defined as: $n_c \in \mathbb{N}$ denotes the number of classes if the dataset is labelled, $m \in (0, 1)$ denotes ratio of missing data, $d \in \mathbb{N}$ denotes the state space dimension of sequences, $L_{\mathbf{X}} \in \mathbb{N}$ denotes the sequence length, $n_{\mathbf{X}}, n_{\mathbf{X}_\star} \in \mathbb{N}$ denote the number of examples in the respective training and testing sets. For Sprites no labels are available, while for Physionet all examples are in the training set and no ground truth values are available. For HMNIST, the MNAR version was used, the most difficult missingness mechanism according to Fortuin et al. (2020).

Table 8: Specification of datasets used for imputation

Dataset	n_c	m	d	$L_{\mathbf{X}}$	$n_{\mathbf{X}}$	$n_{\mathbf{X}_\star}$
HMNIST	10	0.45	28×28	10	60000	10000
Sprites	-	0.6	$64 \times 64 \times 3$	8	9000	2664
Physionet	2	0.82	35	48	3997	-

Experiment details. As depicted in Figure 2, the difference between the original GP-VAE model and ours is that is that we additionally employ a single bidirectional Seq2Tens block (B-LS2T) in the encoder network following the convolutional layer, but preceding the time-distributed dense layers. The motivation for this is that the original encoder only takes local sequential structure into account using the convolutional layer. Hence, it does not exploit global information about the sequence, which might limit flexibility and expressiveness of the encoder network. This limitation can lead to suboptimal inference, due to the fact that the encoder is not able to represent a rich enough subset of the variational family of distributions. This is called the amortization gap in the literature (Cremer et al., 2018).

We have thus hypothesized that by incorporating a bidirectional LS2T layer into the model that takes sequential structure into account not only locally, but globally, we can improve the expressiveness of the encoder network, that can in turn improve on the variational approximation. However, it should be noted that according to the findings of Cremer et al. (2018), a larger encoder network can potentially result in the variational parameters being overfitted to the training data, and can degrade the generalization on unseen data examples. Therefore, the main question is whether increasing the expressiveness of the encoder will improve the quality of the variational approximation on both seen and unseen examples, or will it lead to overfitting to the seen examples?

Another interesting question that we have not considered experimentally, but could lead to improvements is the following. The time-point-wise decoder function assumes that $d' \in \mathbb{N}$ is large enough, so that $\mathbf{z} \in \mathrm{Seq}(\mathbb{R}^{d'})$ is able to fully represent $\mathbf{x} \in \mathrm{Seq}(\mathbb{R}^d)$ in a time-point-wise manner including its dynamics. Although in theory the GP prior should be able to learn the temporal dynamics in the latent space, this might again only be possible for a large enough latent state size d'. In practice, it could turn out to be more efficient to use some of the contextual information in the decoder network as well, either locally, using e.g. a CNN, or globally, using e.g. LS2T layers or RNNs/LSTMs.

Implementation. For the implementation of the GP-VAE, we used the same one as in Fortuin et al. (2020), which implements it using Keras and Tensorflow. The bidirectional LS2T layer used our own implementation based on the same frameworks. The hyperparameters of the models, which are depicted in Appendix A in Fortuin et al. (2020), were left unchanged. The only change we concocted is the B-LS2T layer in the encoder network as depicted in Figure 2. The width of the B-LS2T layer was set to be the same as the convolutional layer, and $M = 4$ tensor levels were used. The parametrization of the low-rank LS2T layer used the independent formulation as detailed in Appendix D.2.

We also made a simple change to how the data is fed into the encoder. In the original model, the missing values were imputed with 0, while we instead used the forward imputed values. This was necessary due to the difference operation preceding the B-LS2T layer in Figure 2. With the zero imputation, the coordinates with missing values exhibited higher oscillations after differencing, while with forward imputation the missing values were more well-behaved. A simple way to see this is that, if there were no preprocessing and convolutional layers in Figure 2 preceding the difference block, then this step would be equivalent to imputing the missing values with zero after differencing.

Result details. Table 3 shows the achieved performance on the datasets with our upgraded model, GP-VAE (B-LS2T), compared against the original GP-VAE (Fortuin et al., 2020) and the baselines. The reported results are negative log-likelihood (NLL), mean squared error (MSE) and AUROC on HMNIST, while on Sprites the MSE is reported and on Physionet the AUROC score. As Sprites is unlabeled, downstream classification performance (AUROC) is undefined on this dataset, while Physionet does not have ground truth values for the missing entries, and reconstruction error (MSE,NLL) is not defined. The only missing entry is Sprites NLL, which was omitted to preserve space. We observe that increasing the expressiveness of the encoder did manage to improve the results on HMNIST and Physionet. The only case where no improvement was observable is Sprites, where the GP-VAE already achieved a very low MSE score of $MSE = 2 \times 10^{-3}$.

To gain some intuition whether the lack of improvement on Sprites was due to the GP-VAE's performance already being maxed out, or there was some other pathology in the model, we further investigated the Sprites dataset and found a bottleneck in both the original and enhanced GP-VAE models. Due to the high dimensionality of the state space of input sequences, $d = 12288$, the width of the first convolutional layer in the encoder network was set to $h = 32$ in order to keep the number of parameters in the layer manageable and be able to train the model with a batch size of $n = 64$, while all subsequent layers had a width of $h' = 256$. Thus, to see if this was indeed an information bottleneck, we increased the convolution width to $h = 256$ and decreased the batch size to $n = 16$, with all other hyperparameters unchanged. Then, we trained using this modification both the baseline GP-VAE and our GP-VAE (B-LS2T) five times on the Sprites datasets. The achieved MSE scores were (i) GP-VAE (base): $MSE = 1.4 \times 10^{-3} \pm 4.1 \times 10^{-5}$, (ii) GP-VAE (B-LS2T): $MSE = 1.3 \times 10^{-3} \pm 4.9 \times 10^{-5}$. Therefore, the smaller convolutional layer was indeed causing an information bottleneck, and by increasing its width to be on par with the other layers in the encoder, we managed to improve the performance of both models. The improvement on the GP-VAE (B-LS2T) was larger, which can be explained by the observation that lifting the information bottleneck additionally allowed the benefits of the B-LS2T layer to kick in, as detailed previously.

To sum up, we have empirically validated the hypothesis that capturing global information in the encoder was indeed beneficial, and managed to improve on the results even on unseen examples in all cases. The experiment as a whole supports that our introduced LS2T layers can serve as useful building blocks in a wide range of models, not only discriminative, but also generative ones.