Supplementary material

A. Tensors

We recall classical constructions with tensors.

A.1. Tensor products of vector spaces

If $u = (u_1, \dots, u_d) \in \mathbb{R}^d$ and $v = (v_1, \dots, v_e) \in \mathbb{R}^e$ then $u \otimes v \in \mathbb{R}^d \otimes \mathbb{R}^e$ is the $(d \times e)$-matrix with indices $i \in \{1, \dots, d\}$, $j \in \{1, \dots, e\}$ and the (i, j)-th entry given as $(u \otimes v)_{i,j} = u_i v_j$. Similarly, for $u \in \mathbb{R}^d$,$v \in \mathbb{R}^e$, $w \in \mathbb{R}^f$, the tensor $u \otimes v \otimes w \in \mathbb{R}^d \otimes \mathbb{R}^e \otimes \mathbb{R}^f$ has indices $i \in \{1, \dots, d\}$, $j \in \{1, \dots, e\}$, $j \in \{1, \dots, f\}$ and its (i, j, k)-th entry is given as $(u \otimes v \otimes w)_{i,j,k} = u_i v_j w_k$, etc.

In the paragraph about variations in Section 4, we mention that one can also lift the sequence to a path evolving in an infinite-dimensional space V rather than $\mathbb{R}^d$ before computing its signatures. Since $\int dx^{\otimes m} \in V^{\otimes m}$ this requires to take a tensor product of an infinite-dimensional space V. Since this might be less known in ML, let us briefly recall a coordinate-free definition of the tensor product: If U and V are vector spaces (not necessarily finite dimensional) then there exists a linear space $U \otimes V$ and a bilinear map $\iota : U \times V \to U \otimes V$ such that any other bilinear map on $U \times V$ factors through $U \otimes V$, that is given any bilinear map $B : U \times V \to Z$ into a vector space Z, there exists a linear map $\hat{B} : U \otimes V \to Z$ such that $\hat{B} \circ \iota = B$. Further, the vector space $U \otimes V$ is unique up to isomorphism. If U, V are finite dimensional it is easy to verify that one recovers the coordinate-wise definition we recalled at the beginning of this section. If $U = V$ then we write $V^{\otimes 2}$ instead of $V \otimes V$; further by convention we define $V^{\otimes 0} := \{1\}$.

A.2. Sequences of tensors $\prod_{m=0}^{M} V^{\otimes m}$

The direct product $\prod_{m \geq 0} V_m$ of vector spaces $V_1, V_2, \dots$ is the set of sequences

$$\prod_{m \geq 0} V_m := \{(t_0, t_1, t_2, \dots,) : t_m \in V_m\}.$$

In our setting, we apply this when V is a vector space and $V_m := V^{\otimes m}$, the get the space

$$\prod_{m \geq 0} V^{\otimes m}.$$

That is, an element $t = (t_m)_{m \geq 0} \in \bigoplus_{m \geq 0} V^{\otimes m}$ is a sequence of tensors of increasing depth, that is $t_0 = 1$ since by convention $V^{\otimes 0} = \{1\}$, $t_1 \in V$ is a vector, $t_2 \in V^{\otimes 2}$, etc.

The space $\prod_{m \geq 0} V^{\otimes m}$ is itself a vector space if one defines addition and scalar multiplication coordinate-wise: for $s = (s_m)_{m \geq 0}$,$t = (t_m)_{m \geq 0} \in \prod_{m \geq 0} V^{\otimes m}$

$$s + t = (s_0 + t_0, s_1 + t_1, s_2 + t_2, \dots)$$
$$\lambda \cdot s = (\lambda s_0, \lambda s_1, \lambda s_2, \dots)$$

That is, if $V = \mathbb{R}^d$ we add vectors to vectors, matrices to matrices, etc. We note that the space $\prod_{m \geq 0} V^{\otimes m}$ is not just a vector space but has also a natural algebra structure and the space $\prod_{m \geq 0} V^{\otimes m}$ is often referred to as the tensor algebra over V.

A.3. Inner products of tensors

We have seen how to build out of a linear space V another linear space $\prod V^{\otimes m}$ of tensors. If V also carries an inner product, $\langle \cdot, \cdot \rangle_V$ this extends canonically to an inner product on subset of $\prod_{m \geq 0} V^{\otimes m}$; set

$$\langle v_1 \otimes \cdots \otimes v_m, w_1 \otimes \cdots \otimes w_m \rangle := \prod_{i=1}^{m} \langle v_i, w_i \rangle_V$$

and extend linearly to $\{t \in \prod_{m \geq 0} V^{\otimes m} : \langle t, t \rangle < \infty\}$. In particular, we can take linear functionals of t.

A.4. Example: the classic polynomial features

Take $\mathbb{R}^d$ with the standard Euclidean inner product. In Section 2 we recalled the classic "polynomial feature map" that takes a point in $\mathbb{R}^d$ to monomials in coordinates in x,

$$\varphi : \mathbf{x} \mapsto (\mathbf{x}^{\otimes m})_{m \geq 0} \in \prod_{m \geq 0} (\mathbb{R}^d)^{\otimes m}. \tag{11}$$

We can build a functions $f : V \to \mathbb{R}$ by taking linear functionals of φ, that is for $\ell \in \prod_{m=0}^{M} (\mathbb{R}^d)^{\otimes m}$ define

$$f : \mathbf{x} \mapsto \langle \ell, \varphi(\mathbf{x}) \rangle.$$

It might be helpful for readers less familiar with tensor products to spell out the definition of f in coordinates: by definition of the inner product we have

$$\langle \ell, \Phi(x) \rangle = \sum_{m=1}^{M} \langle \ell_m, \mathbf{x}^{\otimes m} \rangle.$$

Spelled out in coordinates, $\mathbf{x} = (x_1, \dots, x_d)$ and $\ell_m = (\ell_m^{i_1, \dots, i_m})_{i_1, \dots, i_m \in \{1, \dots, d\}}$, the terms in the sum read as

$$\langle \ell_m, \mathbf{x}^{\otimes} \rangle = \sum_{i_1, \dots, i_m \in \{1, \dots, d\}} \ell_m^{i_1, \dots, i_m} x_{i_1} \cdots x_{i_m}.$$

Thus formulated in coordinates one has

$$\begin{aligned} f(x) = \ell_0 &+ \ell_1^1 x_1 + \cdots \ell_1^1 x_d \\ &+ \ell_2^{1,1} x_1^2 + \ell_2^{1,2} x_1 x_2 + \cdots + \ell_2^{2,2} x_d^2 \\ &+ \vdots \\ &+ \ell_m^{1,\dots,1} x_1^m + \cdots + \ell_m^{d,\dots,d} x_d^m \end{aligned}$$

which is how the polynomial feature map is often represented, see (Rasmussen & Williams, 2006).

B. Signature features

In this Section we give background on signature features. Signature features can be seen as a natural generalization of the polynomial feature map, but instead of mapping a point in $\mathbb{R}^d$ to a sequence of tensors, they map paths $\mathcal{X}_{path}$ to a sequence of tensors. They generalize many of the nice properties of polynomial features such as universality and simulatenuously give the option to ignore the time-parametrization without an explicit search over all possible time changes (like in DTW approaches).

B.1. Definition

By Definition (3), the signature features are given as iterated integrals

$$\Phi(\mathbf{x}) = (1, \int_0^{t_\mathbf{x}} d\mathbf{x}, \int_0^{t_\mathbf{x}} d\mathbf{x}^{\otimes 2}, \dots, \int_0^{t_\mathbf{x}} d\mathbf{x}_\tau^{\otimes m})$$

where $\int_0^t d\mathbf{x}^{\otimes(m+1)} := \int_0^{t_\mathbf{x}} \int_0^s d\mathbf{x}^{\otimes m} \otimes d\mathbf{x}(s) \in (\mathbb{R}^d)^{\otimes m}$ and by convention $\int_0^t d\mathbf{x}^{\otimes 1} := \mathbf{x}(t)$. Hence, $\Phi(\mathbf{x}) \in \prod_{m=0}^M (\mathbb{R}^d)^{\otimes m}$.

B.2. Examples

Coordinate-wise. For a path $x : t \mapsto (x_1(t), \dots, x_d(t))$ that evolves in $\mathbb{R}^d$, one can spell this out in coordinates: the m-th signature feature $\int dx^{\otimes m} \in (\mathbb{R}^d)^{\otimes m}$ is the tensor that has as its $(i_1, \dots, i_m) \in \{1, \dots, d\}^m$-th coordinate entry the real number computed by a Riemann–Stieltjes integral

$$\int dx_{i_1}(t_1) \cdots dx_{i_m}(t_m) = \int \dot{x}_{i_1}(t_1) \cdots \dot{x}_{i_m}(t_m) dt_1 \cdots dt_m$$

where the integration is taken over $0 \le t_1 < \cdots < t_m \le t_\mathbf{x}$ and $\dot{x}_i(t) := \frac{dx_i(t)}{dt}$.

Linear paths. Consider the path $x : [0, 1] \to \mathbb{R}^d$ that just runs along a straight line

$$x(t) = t \mapsto tv \tag{12}$$

where $v \in \mathbb{R}^d$ is a given vector. Plugging (12) into the definition of the iterated integrals, we get by a direct calculation that

$$\int dx^{\otimes m} = \frac{v^{\otimes m}}{m!} \in (\mathbb{R}^d)^{\otimes m}.$$

We see that for this special case of a path x that is fully described by its increment $x(t_\mathbf{x}) - x(0) = v$, the signature features $\Phi(\mathbf{x})$ equal the polynomial features $\varphi(\mathbf{x}(t_\mathbf{x}) - \mathbf{x}(0))$ of the total increment $v = \mathbf{x}(t_\mathbf{x}) - \mathbf{x}(0)$ up to a rescaling by a constant $\frac{1}{m!}$. (This is one of the many reasons why signature features are regarded as "polynomials of paths").

Piecewise linear paths. In general, these integrals need to be computed by standard integration techniques but for a piecewise linear path $\mathbf{x}$, that is $[0, t]$ is partitioned into L disjoint intervals, $[0, t_\mathbf{x}] = \bigsqcup_{i=0}^{L-1} [t_i, t_{i+1}]$, and $\mathbf{x}$ is piecewise linear on each of these pieces, $\mathbf{x}(t) = t \cdot v_i$ for $t \in [t_i, t_{i+1}]$ for a vector $v_i \in \mathbb{R}^d$, then these iterated integrals just reduce to iterated sums, $\int dx^{\otimes m}$ equals

$$\sum_\mathbf{i} c(\mathbf{i}) v_{i_1} \otimes \cdots \otimes v_{i_m} \cdot (t_{i_1+1} - t_{t_1}) \cdots (t_{t_{i_m}+1} - t_{t_{i_m}})$$

where the sum is taken over all tuples $\mathbf{i} = (i_1, \dots, i_m) \in \{1, \dots, L\}^m$ and $c(\mathbf{i})$ is the inverse of the natural number $|\{p : \{1, \dots, L\} \to \{1, \dots, m\}, p(i+1) \ge p(i), p(\mathbf{i}) = \mathbf{i}\}|$!.

B.3. Parametrization invariance.

A classic result going back to Chen (Chen, 1958) shows that the map $\mathbf{x} \mapsto \Phi_0(\mathbf{x})$ is injective up to tree-like equivalence. Loosely speaking, tree-like equivalence is from a purely analytic point of view more natural to work with than reparametrization since tree-like equivalence between paths is analogous to Lebesgue almost sure equivalence between sets. Howevever, we emphasize that from a practical point of view, the difference between paths that are tree-like equivalent and paths that differ by a reparametrization is negligible and we invite the reader to use them as synonyms throughout this article. Nevertheless, we give the precise definition below and refer the interested reader to (Hambly & Lyons, 2010) for a detailed discussion.

Definition 1. *A bounded variation path $\mathbf{x} : [0, t_\mathbf{x}] \to V$ is* tree-like *if there exists a continuous function $h : [0, t_\mathbf{x}] \to [0, \infty)$ such that $h(0) = h(T) = 0$ and such that for all $s < t$*

$$|\mathbf{x}(t) - \mathbf{x}(s)| \le h(s) + h(t) - 2 \inf_{u \in [s,t]} h(u).$$

Theorem 2. *Let $\mathbf{x} : [0, t_\mathbf{x}] \to V$ and $\mathbf{y} : [0, t_\mathbf{y}] \to V$ be two paths of bounded variation. Then*

$$\Phi(\mathbf{x}) = \Phi(\mathbf{y})$$

if and only if $\mathbf{x} \star \overleftarrow{\mathbf{y}}$ is tree-like where $\star$ denotes path concatenation and $\overleftarrow{\mathbf{y}}(t) := \mathbf{y}(t_\mathbf{y} - t)$ denotes time-reversal.

In particular this implies that for any function of the form

$$f(\mathbf{x}) = \langle \ell, \Phi_0 f(\mathbf{x}) \rangle$$

$f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x}$ and $\mathbf{y}$ differ by parametrization (strictly speaking, by a tree-like equivalence). This ability to factor out time-invariance can be very powerful since the space of all possible time reparametrization is huge and we never make an explicit search over all possible time changes like in the calculation of DTW distance.

B.4. Parametrization variance.

Often the functions of sequences $f(\mathbf{x})$ one is interested in, are invariant up to a certain degree of reparametrization but not invariant to extreme reparametrizations. For a stylized example consider TS that arise as blood pressure measurements from patients responding to medication: some patients respond slower, some faster, depending on metabolism and many other factors. Up to a certain degree of time-reparameterisation one should observe a similar shaped TS if the medication works. However, the feature map should allow to distinguish extreme cases, e.g. where the blood pressure is rapidly falling.

To address, we added an extra coordinate to the path $\mathbf{x}$ before computing the signature features of this enhanced path $\mathbf{x}_\tau(t) = (\tau \cdot t, \mathbf{x}(t)) \in \mathbb{R}^{d+1}$, for $\tau > 0$. The enhanced path $\mathbf{x}_\tau$ is never tree-like since the first coordinate $t \mapsto t \cdot \tau$ is strictly increasing. Formulated, differently: this "trick" makes the parametrization part of the trajectory. Hence, the map

$$\mathcal{X}_{paths} \ni \mathbf{x} \mapsto \Phi_\tau(\mathbf{x}) := \Phi(\mathbf{x}_\tau) \in \prod_{m=0}^{M} (\mathbb{R}^{1+d})^{\otimes m}$$

is injective for $\tau > 0$.

B.5. Universality.

One of the most attractive properties of the classical polynomial feature map $\mathbf{x} \to \varphi(\mathbf{x})$ for vectors $\mathbf{x} \in \mathcal{X} = \mathbb{R}^d$, (11), is that any continuous function $f : \mathcal{X} = \mathbb{R}^d \to \mathbb{R}$ can be uniformly approximated on compact sets as a linear functional of φ, that is $f(\mathbf{x}) \approx \langle \ell, \varphi(\mathbf{x}) \rangle$ for some ℓ. The reason is that linear combinations of monomials (polynomials) form an algebra and the Stone–Weierstrass theorem applies. Such approximation properties of feature maps are usually referred to as "universality" in the ML literature.

One of the most attractive properties of the signature feature map $\mathbf{x} \mapsto \Phi_\tau(\mathbf{x})$ for paths $\mathbf{x} \in \mathcal{X}_{paths}$ is that a universality result holds. For every continuous $f : \mathcal{X}_{paths} \to \mathbb{R}$, $K \subset \mathcal{X}_{paths}$ compact, $\epsilon > 0$ there exists a $M \geq 1$, $\ell \in \prod_{m=0}^{M} V^{\otimes m}$ such that

$$\sup_{\mathbf{x} \in K} |f(\mathbf{x}) - \langle \ell, \Phi_\tau(\mathbf{x}) \rangle| < \epsilon.$$

The analogous result holds for $\tau = 0$ when we replace the domain $\mathcal{X}_{paths}$ by equivalence classes of paths (under reparameterisation/tree-like equivalence). For a proof and many extensions, see (Chevyrev & Oberhauser, 2018).

B.6. High-frequency sampling

One way to think about the embedding of $\mathcal{X}_{seq} \hookrightarrow \mathcal{X}_{paths}$ is that $\mathcal{X}_{paths}$ represents the "real-world" where quantities evolve in continuous time but due to pratical reasons such as storage cost we only have access to their preimage in $\mathcal{X}_{seq}$. A natural question is what happens when the sampling gets finer and finer. We believe such consistency in the high-frequency sampling limit is important for the same reason, consistency in the number of samples $n_{\mathbf{X}}$ is important: although in practice we only deal with finite numbers (finite number of samples, sequences rather than paths), we want that our method makes sense as we get more and more information. In the context of learning with sequences this does not only require to study $n_{\mathbf{X}} \to \infty$ but also the limit as the mesh size of that sampling grid converges to 0.

Consistency. More formally, given $\mathbf{x} \in \mathcal{X}_{paths}$ consider a sequence (π_k) of partitions

$$\pi_k = \{(t_1^k, \dots, t_n^k) : 0 \leq t_1^k < \cdots < t_n^k \leq t_{\mathbf{x}}\}$$

with vanishing mesh

$$\mathrm{mesh}(\pi_k) := \max |t_{i+1}^k - t_i^k| \to 0 \text{ as } k \to \infty.$$

Each partition π_k gives rise to sequence $\mathbf{x}^k$ by sampling γ along the time points in π_k. Following our convention we identify $\mathbf{x}^k$ as a piecewise linear path in $\mathcal{X}_{paths}$ and it is easy to verify that $\|\mathbf{x} - \mathbf{x}^k\| \to 0$ as $k \to \infty$. Informally, as $k \to \infty$ we go from discrete to continuous time. One of the nice properties of our GP covariance, is that it is consistent under such limits: given $\mathbf{x}, \mathbf{y} \in \mathcal{X}_{paths}$, $k(\mathbf{x}^k, \mathbf{y}^k) \to k(\mathbf{x}, \mathbf{y})$ as $k \to \infty$. Having a well-defined GP on paths that is consistent under such approximations from discrete to continuous time guarantee that no constants blow up as the sequences gets longer (sampling gets high frequent).

Rough paths. So far we assumed that $\mathcal{X}_{seq}$ consists of bounded variation paths but in the "real-world", the evolution of quantities is often subject to noise, e.g. a classical model in physics and engineering is

$$\mathbf{x}(t) := a(t) + B(t)$$

where a is a bounded variation path but B is a Brownian sample path. Since Brownian sample paths are not of bounded variation, $\mathbf{x}$ is not of bounded variation. However, the same consistency arguments as above go through but one has to replace the iterated Riemann–Stieltjes integrals by Ito–Stratonovich integrals in the definition of $\Phi(\mathbf{x})$. Even rougher trajectories such as fractional Brownian motion and non-Markovian processes can be handled that way with so-called rough path integrals. This is well-beyond the scope of the present article but we refer the interest reader to (Chevyrev & Oberhauser, 2018) for such results.

C. GPs with Signature Covariances

We specified a covariance function k on the set $\mathcal{X}_{paths}$ as inner product of the signature map. This guarantees that

$(\mathbf{x},\mathbf{y}) \mapsto k(\mathbf{x},\mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle$ is a positive definite function and from the general theory of stochastic processes the existence of a centered GP $(f_{\mathbf{x}})_{\mathbf{x}\in\mathcal{X}_{paths}}$ such that $\mathbb{E}[f_{\mathbf{x}}f_{\mathbf{y}}] = k(\mathbf{x},\mathbf{y})$ follows. However, this does not guarantee that the sample paths $\mathbf{x} \mapsto f_{\mathbf{x}}$ are continuous. Seminal work of Dudley (Dudley, 2010) showed that such regularity estimates can be derived by bounding the growth of the covering number of the index set of the GP f under the semi-metric

$$\begin{aligned} d_k(\mathbf{x},\mathbf{y}) &= \sqrt{\mathbb{E}[|f_{\mathbf{x}} - f_{\mathbf{y}}|^2]} \\ &= \sqrt{k(\mathbf{x},\mathbf{x}) - 2k(\mathbf{x},\mathbf{y}) + k(\mathbf{y},\mathbf{y})}. \end{aligned}$$

Already when when the index set is finite dimensional "nice" covariance functions can lead to discontinuous GPs, see e.g. Section 1.4. in (Adler & Taylor, 2009). Our GP has as index set the space of bounded variation paths $\mathcal{X}_{paths}$ which is infinite-dimensional so some caution is needed. However, as we show below we can cover this space by lattice paths and derive covering number estimates that imply continuity.

Theorem 3. *For $L > 0$ and $\epsilon > 0$ denote with $N(\epsilon, L)$ the covering number of the set*

$$\mathcal{X}^L_{paths} := \{x \in \mathcal{X}_{paths} : \|x\|_{bv} \le L\}$$

of bounded variation paths of length less or equal than L under the d_k pseudo-metric. Then

$$\log_2 N(\epsilon, L) \le 2(d+1)L\frac{\sqrt{M}}{\epsilon}$$

Proof. By definition of the metric d_k

$$\begin{aligned} d_k(x,y) &\equiv \sqrt{\langle \Phi(\mathbf{x}) - \Phi(\mathbf{y}), \Phi(\mathbf{x}) - \Phi(\mathbf{y})\rangle} \\ &= \|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\|. \end{aligned}$$

By definition Φ and of the norm $\|\cdot\|$ on $\prod_{m=0}^M (\mathbb{R}^d)^{\otimes m}$ this reads

$$\begin{aligned} d_k^2(\mathbf{x},\mathbf{y}) &= \sum_{m=1}^M \| \int d\mathbf{x}^{\otimes m} - \int d\mathbf{y}^{\otimes m}\|^2 \\ &\le M \max_{m=1,\dots,M} \Delta_m^2(\mathbf{x},\mathbf{y}) \end{aligned}$$

where we denote $\Delta_m(\mathbf{x},\mathbf{y}) := \| \int d\mathbf{x}^{\otimes m} - \int d\mathbf{y}^{\otimes m}\|$. Let $\mathcal{X}^{s,L}_{lattice} \subset \mathcal{X}^L_{paths}$ be the set of lattice paths starting at $0 \in \mathbb{R}^d$ that take steps of size s and that are of total length at most L. By the results in Section 4 of (Lyons & Xu, 2011), for every $\mathbf{x} \in \mathcal{X}^L_{paths}$ and every $n \ge 1$ there exists a $\mathbf{y} \in \mathcal{X}^{L2^{-n},L}_{lattice}$ such that for every $m \ge 1$,

$$\Delta_m(\mathbf{x},\mathbf{y}) \le \frac{d}{2^{n-1}}\frac{4L^{m-1}}{(m-1)!}. \tag{13}$$

Since $\frac{L^{m-1}}{(m-1)!} \le e^L - 1$ we can apply (13) with $n = n(\epsilon) := 1 - \log_2 \frac{\epsilon}{d\sqrt{M}4(e^L-1)}$ to get $\Delta_m(\mathbf{x},\mathbf{y}) \le \epsilon$. Hence, there exists a lattice path $\mathbf{y} \in \mathcal{X}^{L2^{-n(\epsilon)},L}_{lattice}$ such that

$$d_k(\mathbf{x},\mathbf{y}) \le \epsilon.$$

Further, the set $\mathcal{X}_{paths}$ is finite and we can bound it by

$$\begin{aligned} |\mathcal{X}^{L2^{-n(\epsilon)},L}_{lattice}| &\le (2^d+1)^{L2^{n(\epsilon)}} \le 2^{(d+1)L2^{n(\epsilon)}} \\ &= 2^{2(d+1)L2^{n(\epsilon)-1}} = 2^{2(d+1)L\frac{\sqrt{M}}{\epsilon}} \end{aligned}$$

where the first inequality follows since a lattice path has at every step 2^d directions to choose from and in addition can choose not to make a step. The last equality follows from the definition of $n(\epsilon)$. Since $\mathbf{x} \in \mathcal{X}^L_{paths}$ was chosen arbitrary it follows that $\mathcal{X}^L_{paths}$ can be covered by $2^{2(d+1)L\frac{\sqrt{M}}{\epsilon}}$ balls of radius ϵ centered at lattice paths. □

Theorem 3 combined with Dudley's celebrated entropy estimates gives regularity results for samples of our GP. In fact, this even yields a modulus of continuity for our GP.

Theorem 4. *There exists a centered GP $(f_{\mathbf{x}})_{\mathbf{x}\in\mathcal{X}^L_{paths}}$ that has a covariance $\mathbb{E}[f_{\mathbf{x}}f_{\mathbf{y}}]$ the signature covariance function $k(\mathbf{x},\mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle$. Moreover, if we denote its modulus of continuity on $\mathcal{X}^L_{paths}$ with*

$$\omega(\delta) := \sup_{\substack{\mathbf{x},\mathbf{y}\in\mathcal{X}^L_{paths} \\ d_k(\mathbf{x},\mathbf{y})<\delta}} |f_{\mathbf{x}} - f_{\mathbf{y}}|$$

then it holds with probability one that

$$\limsup_{\delta\to 0} \frac{\omega(\delta)}{\sqrt{\delta}4\sqrt{(d+1)L\sqrt{M}} + c\delta\sqrt{\ln\ln\frac{1}{\delta}}} \le 24 \tag{14}$$

where $c > 0$ denotes a universal constant.

Proof. The existence of a centered GP $(\hat{f}_{\mathbf{x}})_{\mathbf{x}}$ with covariance k follows from general results about Gaussian processes. The existence of a continuous modification $(f_{\mathbf{x}})_{\mathbf{x}\in\mathcal{X}_{paths,L}}$ of follows from Dudley's theorem if

$$\int_0^1 \sqrt{\log_2 N(\epsilon, L)}d\epsilon < \infty$$

but by Theorem 3 we have

$$\int_0^1 \sqrt{\log_2 N(\epsilon, L)}d\epsilon \le \sqrt{2(d+1)L\sqrt{M}}\int_0^1 \frac{1}{\sqrt{\epsilon}}d\epsilon < \infty.$$

Dudley's results immediately yield a modulus of continuity in probability. By standard arguments this can be strengthened to give an almost sure modulus of continuity. Concretely, we use the formulation given in Theorem 2.7.1 in

Chapter 5 of (Khoshnevisan, 2002) which guarantees that

$$\limsup_{\delta\to 0} \frac{\omega(\delta)}{\int_0^\delta \sqrt{N(\frac{\epsilon}{2}, L)}d\epsilon + c\delta\sqrt{\ln\ln\frac{1}{\delta}}} \leq 24.$$

The bound (14) follows immediately since first term in the denominator equals

$$\begin{aligned}\int_0^\delta \sqrt{\log_2 N\left(\frac{\epsilon}{2}, L\right)}d\epsilon &= \sqrt{2(d+1)L\sqrt{M}}2\sqrt{2}\sqrt{\delta} \\ &= \sqrt{\delta}4\sqrt{(d+1)L\sqrt{M}}.\end{aligned}$$

□

D. Further algorithms

D.1. Notation for computations.

We define notation based on (Kiraly & Oberhauser, 2019) for concisely describing vectorized computations. We use 1-based indexing for arrays to keep in line with the notation of the main text. Let A and B be k-fold arrays of size $(n_1 \times \cdots \times n_k)$, indexed by $i_j \in \{1, \ldots, n_j\}$ for $j \in \{1, \ldots, k\}$. We define the following operations.

(i) The cumulative sum along axis j as:

$$\begin{aligned}&A[:, \ldots, :, \boxplus, :, \ldots, :][i_1, \ldots, i_{j-1}, i_j, i_{j+1}, \ldots i_k] \\ &:= \sum_{\kappa=1}^{i_j} A[i_1, \ldots, i_{j-1}, \kappa, i_{j+1}, \ldots, i_k].\end{aligned}$$

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

$$\begin{aligned}&A[:, \ldots, :, \Sigma, :, \ldots, :][i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k] \\ &:= \sum_{\kappa=1}^{n_j} A[i_1, \ldots, i_{j-1}, \kappa, i_{j+1}, \ldots i_k].\end{aligned}$$

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

$$\begin{aligned}&A[:, \ldots, :, +m, :, \ldots, :][i_1, \ldots, i_j, \ldots, i_k] \\ &:= \begin{cases} A[i_1, \ldots, i_j - m, \ldots i_k], & \text{if } i_j > m, \\ 0, & \text{if } i_j \leq m. \end{cases}\end{aligned}$$

(iv) The element-wise product of arrays A and B as:

$$A \odot B[i_1, \ldots, i_k] := A[i_1, \ldots, i_k] \cdot B[i_1, \ldots, i_k].$$

Additionally, note that the use of the cumulative sum, $\boxplus$, in conjunction with the shift by 1 operator, $+1$, along the same axis is equivalent to an exclusive cumulative sum, where in the new array the i_jth index contains the sum of the original array's elements from 1 to $i_j - 1$.

Algorithm 3 Computing covariances at sequences, $K_{\mathbf{XX}}$

Input: Sequences $\mathbf{X} = (\mathbf{x}_i)_{i=1,\ldots,n_\mathbf{X}} \subset \mathcal{X}_{seq}$, scalars $(\sigma_0^2, \sigma_1^2, \ldots, \sigma_M^2)$, depth $M \in \mathbb{N}$
Compute $K[i, j, l, k] \leftarrow \langle \Delta x_{i,t_l}, \Delta x_{j,t_k} \rangle$ for $i, j \in \{1, \ldots, n_\mathbf{X}\}$, $l, k \in \{1, \ldots, l_\mathbf{X}\}$
Initialize $R[i, j] \leftarrow \sigma_0^2$ for $i, j \in \{1, \ldots, n_\mathbf{X}\}$
Update $R \leftarrow R + \sigma_1^2 \cdot K[:, :, \Sigma, \Sigma]$
Assign $A \leftarrow K$
for $m = 2$ **to** M **do**
  Iterate $A \leftarrow K \odot A[:, :, \boxplus + 1, \boxplus + 1]$
  Update $R \leftarrow R + \sigma_n^2 \cdot A[:, :, \Sigma, \Sigma]$
end for
Output: Matrix of covariances $K_{\mathbf{XX}} \leftarrow R$

D.2. Covariances between sequences and sequences

We describe in Algorithm 3 the computation of the covariance matrix $K_{\mathbf{XX}}$ of $n_\mathbf{X}$ for sequences $\mathbf{X} = (\mathbf{x}_i)_{i=1,\ldots,n_\mathbf{X}} \subset \mathcal{X}_{seq}$, which is a modification of Algorithm 3 from (Kiraly & Oberhauser, 2019). The observant reader will notice that for the vectorization a requirement is that all sequences in $\mathbf{X}$ have the same length, $l_\mathbf{X} := \sup_{\mathbf{x}\in\mathbf{X}} l_\mathbf{x}$. In practice, this is only a computational restriction and can be circumvented by tabulating each sequence to be the same length, e.g. by repeating the last observation as required. The convenience of the parametrization invariance of signatures is that the results remain unchanged.

Simple inspection says that the complexity of Algorithm 3 is of $O((M+d)\cdot n_\mathbf{X}^2 \cdot l_\mathbf{X}^2)$ in time and $O(d \cdot n_\mathbf{X} \cdot l_\mathbf{X} + n_\mathbf{X}^2 \cdot l_\mathbf{X}^2)$ in memory. Although, note that for factorizing likelihoods the computation of the ELBO and making inference about unseen examples $\mathbf{x}_* \in \mathcal{X}_{seq}$ with credible intervals only requires the diagonals of $K_{\mathbf{XX}}$, i.e. $K_\mathbf{X} := [k(\mathbf{x}, \mathbf{x})]_{\mathbf{x}\in\mathbf{X}}$. Hence, for convenience, we give vectorized pseudo-code in Algorithm 4 for computing $K_\mathbf{X}$, which has complexities $O((M+c)\cdot n_\mathbf{X} \cdot l_\mathbf{X}^2)$ in time and $O(d \cdot n_\mathbf{X} \cdot l_\mathbf{X} + n_\mathbf{X} \cdot l_\mathbf{X}^2)$.

Algorithm 4 Computing variances at sequences, $K_\mathbf{X}$

Input: Sequences $\mathbf{X} = (\mathbf{x}_i)_{i=1,\ldots,n_\mathbf{X}} \subset \mathcal{X}_{seq}$, scalars $(\sigma_0^2, \sigma_1^2, \ldots, \sigma_M^2)$, depth $M \in \mathbb{N}$
Compute $K[i, l, k] \leftarrow \langle \Delta x_{i,t_l}, \Delta x_{i,t_k} \rangle$ for $i \in \{1, \ldots, n_\mathbf{X}\}$, $l, k \in \{1, \ldots, l_\mathbf{X}\}$
Initialize $R[i] \leftarrow \sigma_0^2$ for $i \in \{1, \ldots, n_\mathbf{X}\}$
Update $R \leftarrow R + \sigma_1^2 \cdot K[:, \Sigma, \Sigma]$
Assign $A \leftarrow K$
for $m = 2$ **to** M **do**
  Iterate $A \leftarrow K \odot A[:, \boxplus + 1, \boxplus + 1]$
  Update $R \leftarrow R + \sigma_n^2 \cdot A[:, \Sigma, \Sigma]$
end for
Output: Vector of variances $K_\mathbf{X} \leftarrow R$

E. Further details on experiments

E.1. Implementation details

The implementation of all considered GP models are available at GITHUBAUTHOR. Here, we detail the technicalities related to the implementation of each model.

GP-Sig. This is the standard GP model with the signature kernel over sequences. This is built on top of GPflow (de G. Matthews et al., 2017), and other than a few tweaks, they interface with GPflow models in a straightforward manner. Particularly for the kernel, there are several variants available with different state space embeddings, including RBF and Matérn static kernels. The hyperparameters of the kernel which are learnt from the data are: (1) the lengthscales corresponding to each state space dimension, (2) the scaling parameters that multiply each signature level, allowing to strengthen or weaken its effect, (3) the lag values by which the additional lagged versions of each coordinate are shifted, that is a continuous parameter and is applied using linear interpolation and flat extrapolation (i.e. when the queried time-point is negative then the value at time 0 is used). In Section 5, we denoted the augmented sequence with a time coordinate and p lags by $\hat{\mathbf{x}} := (t_i, x_{t_i}, x_{t_i - s_1}, \dots, x_{t_i - s_p})_{i=1,\dots,l_{\mathbf{x}}}$. The lagged coordinates use the same lengthscales as the original ones, which in many cases leads to better generalization compared to not using lags (e.g. Takens' theorem (Takens, 1981)). The signature kernel is also normalized using the standard kernel normalization $\tilde{\mathrm{k}}(\mathbf{x}, \mathbf{y}) := \mathrm{k}(\mathbf{x}, \mathbf{y}) / \sqrt{\mathrm{k}(\mathbf{x}, \mathbf{x})\,\mathrm{k}(\mathbf{y}, \mathbf{y})}$, which we apply individually to each signature level. The supported inducing variables are `InducingTensors` and `InducingSequences` corresponding to the two variants described in the main text.

GP-Sig-LR. As previously mentioned, there exists a low-rank variant of the signature kernel as introduced in (Kiraly & Oberhauser, 2019), which aims to approximate the feature map using a low-rank approximation, rather than computing inner product of signature features directly. Our implementation first uses the Nyström approximation to find a low-dimensional approximation of the state-space embedding, and then uses the primal formulation of the signature algorithms (see Algorithm 5 in (Kiraly & Oberhauser, 2019)) to compute the signature kernel, while keeping the size of the low-rank factors manageable with sparse randomized projections (Li et al., 2006). Its advantage is that it extends to very long time series due to linear complexity in the time series length $l_{\mathbf{X}} \in \mathbb{N}$, while the quadratic complexity of the full-rank kernel needs to be addressed another way. We did not include this variant among the experiments because overall it performed much worse than the full-rank variant. There were two main issues: (i) on several datasets it failed to fit the dataset due to being less flexible and noise, (ii) even when the predictive means are good, it can still give severely miscalibrated uncertainties similarly to classic kernel approximation techniques (Nyström, RFF), since an LR covariance matrix results in a degenerate GP prior.

GP(-Sig)-LSTM/GRU. The RNN based models with a GP layer placed on top use the Keras implementation of the RNN architectures (Chollet et al., 2015), while the GP parts use the GPflow API, which is possible as both packages can define the computational graph using the Tensorflow backend. However, since none of the packages supports the other, the resulting models have to be trained somewhat manually using the slightly more primitive Tensorflow API, and therefore are not very user friendly. It is up to future work to build a more user friendly API that makes it possible to deploy models that combine neural networks and sparse variational GPs in a convenient manner.

GP-KConv1D. The 1-dimensional convolutional kernel essentially uses the same code as (van der Wilk et al., 2017) included in the GPflow package, with some tweaks that allow different length time series to be compared by padding each sequence with nans and masking the nan entries during the computation. We also normalize the features corresponding to this kernel to unit length in the feature space using the standard kernel normalization. In the experiments, we set the window size to $w = 10$, but a few datasets have $\min_{\mathbf{x} \in \mathbf{X}} l_{\mathbf{x}} < 10$, and in those cases we set $w = \min(10, \min_{\mathbf{x} \in \mathbf{X}} l_{\mathbf{x}})$. Also, as the sequence length $l_{\mathbf{x}}$, and hence, the number of windows can vary from instance to instance, the weighted version of the convolutional kernel from (van der Wilk et al., 2017) is not applicable in this case, and the translation invariant version is used.

E.2. Datasets details

Table 3 details the datasets from (Baydogan, 2015) that we used for benchmarking. Here c denotes the number of classes, d the dimension of the sequence 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 pre-specified training and testing sets. In the experiments, all state space dimensions were normalized to zero mean and unit variance. For the models GP-Sig(-LSTM/GRU), GP-KConv1D, we subsampled very long time series to $l_{\mathbf{X}} = 500$, in order to deal with the quadratic complexity of kernel evaluations and be able to fit within GPU memory limitations.

E.3. Training details

Initialization. For all models considered in the main text in Section 5, the RBF kernel was used as static kernel, which has lengthscale parameters $(l_1, \dots, l_d)$, i.e. the RBF kernel

Table 3: Specification of datasets used for benchmarking

DATASET	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

over $\mathbb{R}^d$ is up to rescaling given by

$$\kappa(\mathbf{x}, \mathbf{x}') := \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \Sigma^{-1}(\mathbf{x} - \mathbf{x}')\right)$$

with $\Sigma_{ii} := l_i^2$ a diagonal matrix. We used the initialization $l_i^{(0)} := \sqrt{\mathbb{E}[(x_i - x_i')^2] \cdot d}$, where x_i, x_i' are two independent copies of the i-th input space coordinate, and we used a stochastic estimator of this with typically $n = 1000$ observation samples from the data.

All considered models in Section 5 used some form of inducing variables. For the signature models, they were placed in the feature space of the signature map in the form of inducing tensors. These inducing tensors given in (4.1) are tensor products of elements in V. As detailed at the end of Section 4, although the state space of a sequence is $\mathbb{R}^d$, we can embed this sequence into a path that evolves in a linear space V that does not have to be $\mathbb{R}^d$. One way to do this is to use an observation-wise state space embedding given by a kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and map a sequence $\mathbf{x} = (t_i, x_i)$ to a sequence $\kappa_{\mathbf{x}} = (t_i, \kappa_{x_i})$ that evolves in the RKHS V of κ; here $\kappa_x := \kappa(x, \cdot) \in V$. Therefore signatures of depth M now live in the space $\prod_{m=0}^M V^{\otimes m}$, which is for most kernels κ a genuine infinite-dimensional space. However, all computations from Sections 3 and 4 carry on mutatis mutandis, with the difference being that we do not have the flexibility to represent the inducing tensors as tensor products of arbitrary elements in V, which are generally infinite dimensional. In this case, we take

$$\mathbf{z} = (z_m)_{m=0,\dots,M} \in \prod_{m=0}^M V_0^{\otimes m} \tag{15}$$

with $z_0 \in \mathbb{R}$ and $z_m = \kappa(x_{m,1}, \cdot) \otimes \cdots \otimes \kappa(x_{m,m}, \cdot)$ with $x_{i,j} \in \mathbb{R}^d$ for $1 \le j \le i$, $1 \le i \le m$, $1 \le m \le M$, where $V_0 := \{\kappa(x, \cdot) : x \in \mathbb{R}^d\}$. Put differently, the inducing tensors are also constrained to being tensor products of only such elements in V which arise as reproducing kernels[6] associated to vectors in $\mathbb{R}^d$. Hence, the complexity of evaluating $\langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle$ is the same as in $\mathbb{R}^d$, and storing an element $\kappa(x, \cdot) \in V_0$ is the same memory. Now, the initialization of the inducing tensors is simply done by sampling random observations from the input sequences in a two step manner: (1) a random input sequence is selected, (2) from the sequence a time-increasing subset of its observations are selected and plugged into the tensor products given in (15), and this procedure is repeated $n_{\mathbf{Z}}$ times.

Other forms of inducing variables used by the models in Section 5 are inducing points for the GP-RNNs and inducing patches for GP-KConv1D. The inducing points are initialized randomly by selecting a $\mathbf{x} \in \mathbf{X}$ and computing its RNN-image $\phi_\theta(\mathbf{x})$, which is then used as an inducing point, and repeated for all $n_{\mathbf{Z}}$. The inducing patches are also initialized in two steps: (1) select a random input sequence $\mathbf{x} \in \mathbf{X}$, (2) select a random window from $\mathbf{x}$, $(x_i, x_{i+1}, \dots, x_{i+w-1})$, where $1 \le w \le \min_{\mathbf{x} \in \mathbf{X}} l_{\mathbf{x}}$ denotes the window length in the convolutional kernel.

For the alternative sparse inference scheme for signatures described in Section 5, denoted the method of inducing sequences, we use the same initialization as for the inducing patches: select a random sequence, and select a random window, and repeat for all $n_{\mathbf{Z}}$.

The means and covariances of the inducing points used the usual whitening transformation, that is, reparametrization in terms of the Cholesky factor L of $K_{\mathbf{ZZ}}$, $K_{\mathbf{ZZ}} = LL^\top$, and parameters initialized from zeros and identity.

The RNNs use the usual initializations, that is, Glorot initialization for the weights (Glorot & Bengio, 2010), orthogonal initialization for the recurrent weights (Saxe et al., 2014), and zeros for the bias.

Optimization details. The training for the benchmarking experiment in Section 4 was performed on 11 GPUs overall: 4 Tesla K40Ms, 5 Geforce 2080 TIs and 2 Quadro GP100 graphics cards. All models were trained 5 times for the benchmarking and the RNN based models an additional 6 times for the grid-search. Thus, the training of overall 480 models required extensive computational resources.

In all experiments in Section 5, we used similar optimization details, that is, optimization with early stopping and checkpointing by optimizing on 80% of the training data and monitoring the nlpp[7] on a 20% validation set. We used

[6] The reproducing kernel associated to a point $x \in \mathcal{X}$ is simply the kernel function evaluated in one of its arguments at x, i.e. $\kappa(x, \cdot) \in \mathcal{H}_0 \subset \mathcal{H}$ for a kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

[7] We found that monitoring the validation nlpp rather than the

a minibatch size of 50, fixed learning rate $\alpha = 1 \times 10^{-3}$, and a patience value of $n = 500$ epochs. As optimizer, GP-Sig and GP-KConv1D used Nadam (Dozat, 2015), while the RNN based models used Adam (Kingma & Ba, 2014). Additionally, as is well-known for SVGPs (Bauer et al., 2016), first fixing the hyperparameters and only optimizing over the variational approximation for a fixed number of epochs is beneficial which we follow. Furthermore, after the main training phase of the hyperparameters has finished, to learn the rest of the validation data that was excluded from the optimization, we re-merge the validation set into the training set, fix the hyperparameters, and optimize only over the variational parameters again to assimilate the remaining information into the variational approximation.

Hence, the training for all models is split into the following phases (1) partition the data in an $80-20$ ratio for optimization and monitoring, (2) with fixed kernel hyperparameters initialized as described previously, train the variational parameters for fixed n epochs to tighten the ELBO bound; (3) unfix the hyperparameters and train by monitoring the nlpp on the validation set, stopping after no improvement for n epochs, and restoring and best model; (4) re-merge the validation set into the training data and train the variational distribution again only for a fixed n epochs with the kernel hyperparameters fixed. In all scenarios, we used $n = 500$.

For GP-Sig, the insertion of an additional optimization phase was found to be beneficial. Particularly, we reparametrize the scaling parameters for the signature levels $\sigma = (\sigma_0, \ldots, \sigma_M)$ as $\sigma = (\beta \cdot \sigma'_0, \ldots, \beta \cdot \sigma'_M)$, where $\beta \in \mathbb{R}^+$. Then, phase (3) is split into two steps: first, train with unfixing all kernel hyperparameters except $(\sigma'_0, \ldots \sigma'_M)$, which are a-priori all set as 1; secondly, now unfixing *all* parameters, continue training with early stopping. This trick allows to calibrate the overall variance of the GP using β in the first step, while fixing $\sigma_0 = \cdots = \sigma_M$. The intution why this works is that the signature levels in general contain complementary information about a given sequence, and fixing them to be equal first enforces the model to find a fit of the data for all signature levels jointly, i.e. in some sense this is an implicit regularization step. The second step allows to slightly adjust the contribution of each level without relying too heavily on any one of them. On the RNN-based signature models this trick did not give substantial improvements, possibly because the variance of the RNN layer generally outweights the variance of the signature layer.

In our experience, when using GP-Sig on datasets with a larger $n_{\mathbf{X}}$, it can yield a further improvement to gradually increase the learning to rate to $\alpha = 1 \times 10^{-2}$ to allow the optimizer to explore the space in more depth, and then decrease it back to $\alpha = 1 \times 10^{-3}$ to drive it to the closest local optima. However, on the smaller datasets this was found to be counterproductive, and in the experiments we chose to stick with a unified scheme that worked consistently on all datasets. However, we also remark that without applying any of the previously described techniques, and training from front to back all parameters jointly with a small learning rate (e.g. $\alpha = 1 \times 10^{-3}$) gives good results already, but a few percents of test set accuracy can be gained on some datasets by using them.

validation accuracy leads to better generalization behaviour.

Architecture search. Table 4 details each of the architectures used for the models containing an RNN layer, where H denotes the number of hidden units used, and D is a boolean trigger, that specifies whether dropout was used for the given experiment or not. In the case $D = 1$, we used the settings `dropout = 0.25` and `recurrent_dropout = 0.05`, otherwise both were set to 0. To find the best performing architecture, we conducted a grid-search among 6 considered architectures, that is, $H \in [8, 32, 128]$ and $D \in [0, 1]$. For the grid-search, only the training data was used, and the data was split in a $60-20-20$ fashion, using 60% for training, 20% for early stopping and checkpointing, and the last 20% was used to evaluate the performance. The training itself was carried out using the same initialization and schedule as described, and was performed only once for each method and setting pair, due to the large number of datasets that we considered.

E.4. Benchmark results

We report in Table 5 and Table 6 the negative log-predictive probabilities and accuracies of the GP models considered in Section 5. For each method-dataset pair, 5 models were trained with the initialization described in Appendix E.3. The variance of the results is therefore due to random initialization of some parameters, and the minibatch randomness while training. The RNN based models used the architectures detailed in Table 4. As non-Bayesian baselines, we report the results of recent frequentist TS classification methods from the respective publications, that is, (Cuturi & Doucet, 2011; Baydogan & Runger, 2015a;b; Karlsson et al., 2016; Tuncel & Baydogan, 2018; Schäfer & Leser, 2017; Karim et al., 2019). Particularly for MLSTMFCN, we report the same results as in (Schäfer & Leser, 2017). In Figure 5, we visualize the box-plot distributions of (1) negative log-predictive probabilities of the GPs, (2) classification accuracies of both the GPs and the frequentist baselines.

Table 4: List of architectures used for the RNN based models

DATASET	GP-SIG-LSTM		GP-SIG-GRU		GP-LSTM		GP-GRU	
ARABIC DIGITS	$H = 128$	$D = 1$	$H = 128$	$D = 1$	$H = 32$	$D = 1$	$H = 128$	$D = 1$
AUSLAN	$H = 128$	$D = 0$	$H = 128$	$D = 0$	$H = 128$	$D = 0$	$H = 32$	$D = 0$
CHARACTER TRAJ.	$H = 8$	$D = 1$	$H = 128$	$D = 1$	$H = 128$	$D = 1$	$H = 128$	$D = 1$
CMUSUBJECT16	$H = 32$	$D = 1$	$H = 32$	$D = 1$	$H = 32$	$D = 1$	$H = 32$	$D = 1$
DIGITSHAPES	$H = 8$	$D = 1$	$H = 128$	$D = 1$	$H = 128$	$D = 1$	$H = 32$	$D = 1$
ECG	$H = 128$	$D = 0$	$H = 128$	$D = 1$	$H = 128$	$D = 0$	$H = 8$	$D = 1$
JAP. VOWELS	$H = 128$	$D = 0$	$H = 128$	$D = 1$	$H = 128$	$D = 1$	$H = 128$	$D = 0$
KICK VS PUNCH	$H = 8$	$D = 1$	$H = 8$	$D = 0$	$H = 128$	$D = 1$	$H = 128$	$D = 1$
LIBRAS	$H = 128$	$D = 0$	$H = 128$	$D = 0$	$H = 32$	$D = 0$	$H = 32$	$D = 0$
NETFLOW	$H = 8$	$D = 0$	$H = 32$	$D = 0$	$H = 32$	$D = 0$	$H = 8$	$D = 1$
PEMS	$H = 32$	$D = 0$	$H = 8$	$D = 1$	$H = 32$	$D = 1$	$H = 32$	$D = 0$
PENDIGITS	$H = 128$	$D = 0$	$H = 128$	$D = 1$	$H = 128$	$D = 1$	$H = 128$	$D = 1$
SHAPES	$H = 8$	$D = 1$	$H = 8$	$D = 1$	$H = 8$	$D = 1$	$H = 8$	$D = 1$
UWAVE	$H = 32$	$D = 1$	$H = 128$	$D = 0$	$H = 32$	$D = 0$	$H = 32$	$D = 0$
WAFER	$H = 128$	$D = 0$	$H = 128$	$D = 1$	$H = 32$	$D = 0$	$H = 32$	$D = 0$
WALK VS RUN	$H = 128$	$D = 1$	$H = 8$	$D = 1$	$H = 8$	$D = 1$	$H = 32$	$D = 1$

Table 5: Mean and standard deviation of negative predictive log-probabilities (nlpp) on test sets over 5 independent runs

DATASET	GP-SIG-LSTM	GP-SIG-GRU	GP-SIG	GP-LSTM	GP-GRU	GP-KCONV1D
ARABIC DIGITS	0.047 ± 0.030	0.023 ± 0.006	0.071 ± 0.021	0.082 ± 0.022	0.066 ± 0.010	0.050 ± 0.003
AUSLAN	0.106 ± 0.007	0.123 ± 0.045	0.550 ± 0.114	0.650 ± 0.071	0.248 ± 0.063	1.900 ± 0.139
CHARACTER TRAJ.	0.031 ± 0.007	0.258 ± 0.265	0.108 ± 0.005	2.506 ± 1.007	3.523 ± 0.635	0.409 ± 0.141
CMUSUBJECT16	0.088 ± 0.020	0.040 ± 0.009	0.089 ± 0.027	0.270 ± 0.080	0.089 ± 0.039	0.255 ± 0.002
DIGITSHAPES	0.008 ± 0.001	0.035 ± 0.051	0.021 ± 0.001	0.013 ± 0.002	0.727 ± 0.569	0.035 ± 0.003
ECG	0.402 ± 0.023	0.431 ± 0.037	0.356 ± 0.008	0.496 ± 0.018	0.601 ± 0.137	0.543 ± 0.019
JAP. VOWELS	0.080 ± 0.031	0.053 ± 0.009	0.069 ± 0.003	0.061 ± 0.029	0.052 ± 0.005	0.067 ± 0.001
KICK VS PUNCH	0.301 ± 0.109	0.493 ± 0.128	0.224 ± 0.014	0.696 ± 0.046	0.674 ± 0.037	0.662 ± 0.017
LIBRAS	0.320 ± 0.045	0.346 ± 0.091	0.259 ± 0.021	0.911 ± 0.056	1.110 ± 0.248	1.608 ± 0.311
NETFLOW	0.218 ± 0.009	0.259 ± 0.078	0.189 ± 0.014	0.251 ± 0.041	0.194 ± 0.011	0.168 ± 0.081
PEMS	0.704 ± 0.130	1.100 ± 0.064	0.520 ± 0.058	1.194 ± 0.308	0.784 ± 0.111	0.537 ± 0.010
PENDIGITS	0.289 ± 0.127	0.399 ± 0.206	0.146 ± 0.007	0.185 ± 0.027	0.187 ± 0.043	0.181 ± 0.005
SHAPES	0.014 ± 0.004	0.012 ± 0.004	0.011 ± 0.002	0.016 ± 0.008	0.168 ± 0.142	0.012 ± 0.001
UWAVE	0.113 ± 0.011	0.121 ± 0.017	0.140 ± 0.004	0.745 ± 0.151	1.168 ± 1.063	0.189 ± 0.008
WAFER	0.048 ± 0.021	0.081 ± 0.011	0.105 ± 0.010	0.105 ± 0.086	0.029 ± 0.011	0.085 ± 0.002
WALK VS RUN	0.030 ± 0.008	0.030 ± 0.008	0.023 ± 0.007	0.048 ± 0.040	0.028 ± 0.000	0.066 ± 0.001
MEAN NLPP.	0.175	0.238	0.180	0.514	0.603	0.423
MED. NLPP.	0.097	0.122	0.124	0.261	0.221	0.185
SD. NLPP.	0.183	0.273	0.161	0.623	0.841	0.542
MEAN RANK ($n_{\mathbf{X}} < 300$)	2.800	2.900	2.200	4.700	4.000	4.400
MEAN RANK ($n_{\mathbf{X}} \geq 300$)	2.333	3.333	2.833	4.833	4.333	3.333
MEAN RANK (ALL)	2.625	3.062	2.438	4.750	4.125	4.000

Table 6: Mean and standard deviation of accuracies on test sets over 5 independent runs

DATASET	GP-SIG-LSTM	GP-SIG-GRU	GP-SIG	GP-LSTM	GP-GRU	GP-KCONV1D
ARABIC DIGITS	0.992 ± 0.003	0.994 ± 0.002	0.979 ± 0.004	0.985 ± 0.004	0.986 ± 0.005	0.984 ± 0.001
AUSLAN	0.983 ± 0.003	0.978 ± 0.006	0.925 ± 0.014	0.880 ± 0.012	0.949 ± 0.014	0.784 ± 0.012
CHARACTER TRAJ.	0.991 ± 0.003	0.925 ± 0.078	0.979 ± 0.002	0.233 ± 0.331	0.114 ± 0.050	0.941 ± 0.013
CMUSUBJECT16	1.000 ± 0.000	1.000 ± 0.000	0.979 ± 0.017	0.924 ± 0.051	0.993 ± 0.014	0.897 ± 0.000
DIGITSHAPES	1.000 ± 0.000	0.988 ± 0.025	1.000 ± 0.000	1.000 ± 0.000	0.812 ± 0.153	1.000 ± 0.000
ECG	0.816 ± 0.029	0.832 ± 0.012	0.848 ± 0.010	0.782 ± 0.032	0.734 ± 0.033	0.760 ± 0.018
JAP. VOWELS	0.981 ± 0.005	0.985 ± 0.004	0.982 ± 0.005	0.982 ± 0.004	0.986 ± 0.005	0.986 ± 0.002
KICK VS PUNCH	0.900 ± 0.063	0.820 ± 0.098	0.900 ± 0.000	0.620 ± 0.075	0.600 ± 0.110	0.700 ± 0.089
LIBRAS	0.921 ± 0.013	0.899 ± 0.031	0.923 ± 0.004	0.776 ± 0.019	0.742 ± 0.050	0.698 ± 0.026
NETFLOW	0.931 ± 0.002	0.921 ± 0.012	0.937 ± 0.003	0.928 ± 0.011	0.926 ± 0.012	0.945 ± 0.027
PEMS	0.763 ± 0.016	0.775 ± 0.019	0.820 ± 0.014	0.745 ± 0.044	0.769 ± 0.020	0.794 ± 0.008
PENDIGITS	0.928 ± 0.030	0.902 ± 0.048	0.955 ± 0.002	0.953 ± 0.008	0.951 ± 0.008	0.946 ± 0.001
SHAPES	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.867 ± 0.163	1.000 ± 0.000
UWAVE	0.970 ± 0.004	0.968 ± 0.006	0.964 ± 0.001	0.870 ± 0.029	0.763 ± 0.225	0.947 ± 0.002
WAFER	0.988 ± 0.005	0.978 ± 0.005	0.965 ± 0.004	0.966 ± 0.037	0.994 ± 0.002	0.984 ± 0.001
WALK VS RUN	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MEAN ACC.	0.948	0.935	0.947	0.853	0.824	0.898
MED. ACC.	0.982	0.973	0.964	0.926	0.896	0.946
SD. ACC.	0.068	0.070	0.052	0.193	0.218	0.107
MEAN RANK ($n_{\mathbf{x}} < 300$)	3.000	3.100	2.800	4.250	4.250	3.600
MEAN RANK ($n_{\mathbf{x}} \geq 300$)	2.167	3.500	3.000	4.167	4.333	3.833
MEAN RANK (ALL)	2.688	3.250	2.875	4.219	4.281	3.688

Table 7: Accuracies of frequentist time series classification methods

DATASET	SMTS	LPS	MVARF	DTW	ARKERNEL	GRSF	MLSTMFCN	MUSE
ARABIC DIGITS	0.964	0.971	0.952	0.908	0.988	0.975	0.990	0.992
AUSLAN	0.947	0.754	0.934	0.727	0.918	0.955	0.950	0.970
CHARACTER TRAJ.	0.992	0.965	0.928	0.948	0.900	0.994	0.990	0.937
CMUSUBJECT16	0.997	1.000	1.000	0.930	1.000	1.000	1.000	1.000
DIGITSHAPES	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
ECG	0.818	0.820	0.785	0.790	0.820	0.880	0.870	0.880
JAP. VOWELS	0.969	0.951	0.959	0.962	0.984	0.800	1.000	0.976
KICK VS PUNCH	0.820	0.900	0.976	0.600	0.927	1.000	0.900	1.000
LIBRAS	0.909	0.903	0.945	0.888	0.952	0.911	0.970	0.894
NETFLOW	0.977	0.968	NA	0.976	NA	0.914	0.950	0.961
PEMS	0.896	0.844	NA	0.832	0.750	1.000	NA	NA
PENDIGITS	0.917	0.908	0.923	0.927	0.952	0.932	0.970	0.912
SHAPES	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
UWAVE	0.941	0.980	0.952	0.916	0.904	0.929	0.970	0.916
WAFER	0.965	0.962	0.931	0.974	0.968	0.992	0.990	0.997
WALK VS RUN	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
MEAN ACC.	0.945	0.933	0.949	0.899	0.938	0.955	0.970	0.962
MED. ACC.	0.964	0.964	0.952	0.929	0.952	0.984	0.990	0.976
SD. ACC.	0.059	0.073	0.055	0.111	0.073	0.058	0.039	0.043

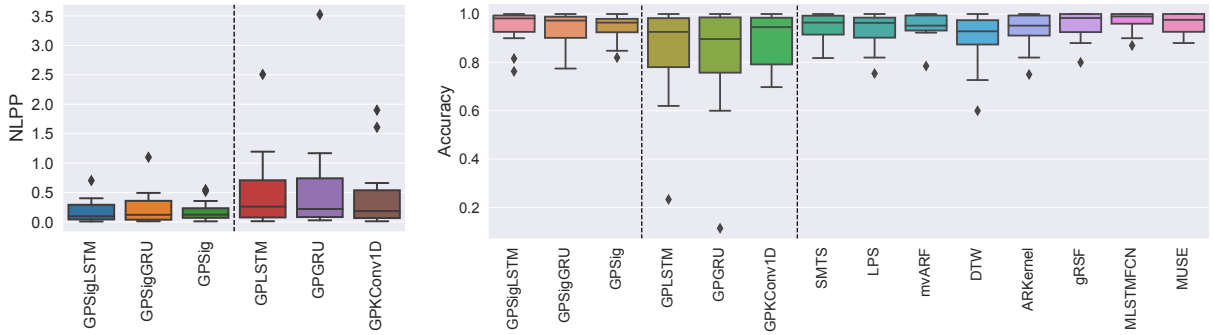

Figure 5: Box-plots of negative log-predictive probabilities (left) and classification accuracies (right) on 16 TSC datasets