SIGNATORY: DIFFERENTIABLE COMPUTATIONS OF THE SIGNATURE AND LOGSIGNATURE TRANSFORMS, ON BOTH CPU AND GPU

Anonymous authors
Paper under double-blind review

ABSTRACT

Signatory is a library for calculating and performing functionality related to the signature and logsignature transforms. The focus is on machine learning, and as such includes features such as CPU parallelism, GPU support, and backpropagation. To our knowledge it is the first GPU-capable library for these operations. Signatory implements new features not available in previous libraries, such as efficient precomputation strategies. Furthermore, several novel algorithmic improvements are introduced, producing substantial real-world speedups even on the CPU without parallelism. The library operates as a Python wrapper around C++, and is compatible with the PyTorch ecosystem. It may be installed directly via `pip`. Source code, documentation, examples, benchmarks and tests may be found at `https://github.com/[redacted]` (see supplementary material). The license is Apache-2.0.

1 INTRODUCTION

The *signature transform*, sometimes referred to as the *path signature* or simply *signature*, is a central object in rough path theory (Lyons, 1998; 2014). It is a transformation on differentiable paths[1], and may be thought of as loosely analogous to the Fourier transform. However whilst the Fourier transform extracts information about frequency, treats each channel separately, and is linear, the signature transform exacts information about order and area, explicitly considers combinations of channels, and is in a precise sense 'universally nonlinear' (Bonnier et al., 2019, Proposition A.6).

The *logsignature transform* (Liao et al., 2019) is a related transform, that we will also consider. In both cases, by treating sequences of data as continuous paths, then the (log)signature transform may be applied for use in problems with sequential structure, such as time series. Indeed there is a significant body of work using the (log)signature transform in machine learning, with examples ranging from handwriting identification to sepsis prediction, see for example Morrill et al. (2019); Fermanian (2019); Király & Oberhauser (2019); Toth & Oberhauser (2019); Morrill et al. (2020b).

Earlier work often used the signature and logsignature transforms as a feature transformation. See Levin et al. (2013); Chevyrev & Kormilitzin (2016); Yang et al. (2016a;b); Kormilitzin et al. (2016); Li et al. (2017); Perez Arribas et al. (2018) for a range of examples. In this context, when training a model on top, it is sufficent to simply preprocess the entire dataset with the signature or logsignature transform, and then save the result.

However, recent work has focused on embedding the signature and logsignature transforms within neural networks. Recent work includes Bonnier et al. (2019); Liao et al. (2019); Moor et al. (2020); Morrill et al. (2020a); Kidger et al. (2020) among others. In this context, the signature and logsignature transforms are evaluated many times throughout a training procedure, and as such efficient and differentiable implementations are crucial. Previous libraries (Lyons, 2017; Reizenstein & Graham, 2018) have been CPU-only and single-threaded, and quickly become the major source of slowdown when training and evaluating these networks.

[1] And may be extended to paths of bounded variation, or merely finite p-variation (Lyons et al., 2004).

1.1 CONTRIBUTIONS

We introduce Signatory, a CPU- and GPU-capable library for calculating and performing functionality related to the signature and logsignature transforms. To our knowledge it is the first GPU-capable library for these operations. The focus is on machine learning applications.

Signatory is significantly faster than previous libraries (whether run on the CPU or the GPU), due to a combination of parallelism and novel algorithmic improvements. In particular the latter includes both uniform and asymptotic rate improvements over previous algorithms. Additionally, Signatory provides functionality not available in previous libraries, such as precomputation strategies for efficient querying of the (log)signature transform over arbitrary overlapping intervals.

The library integrates with the open source PyTorch ecosystem, is compatible with both Python 2 and Python 3, and runs on Mac, Linux and Windows. Documentation, examples, benchmarks and tests form a part of the project.

Much of the code is written in C++ primitives and the CPU implementation utilises OpenMP. The backward operations are handwritten for both speed and memory efficiency, and do not rely on the autodifferentiation provided by PyTorch.

The source code is located at `https://github.com/[redacted]` *(see supplementary material)*, documentation and examples are available at `https://[redacted].readthedocs.io` *(see supplementary material)*, and the project may be installed directly via `pip`.

This paper is not a guide to using Signatory—for that we refer to the documentation. This is meant as a technical exposition of its innovations.

1.2 APPLICATIONS

Signatory has already seen a rapid uptake amongst the signature community. Recent work using Signatory include Morrill et al. (2020b); Perez Arribas et al. (2020) who involve signatures in neural differential equations, or Moor et al. (2020); Min & Ichiba (2020) who study deep signature models (Bonnier et al., 2019). Meanwhile Ni et al. (2020) apply Signatory to hybridise signatures with GANs, and Morrill et al. (2020a) create a generalised framework for the "signature method". As a final example, Signatory is now itself a dependency for other libraries (Kidger, 2020).

2 BACKGROUND

We begin with some exposition on theory of the signature and logsignature transforms. Also see Reizenstein & Graham (2018) for an introduction focusing on computational concern, and Lyons et al. (2004) and Hodgkinson et al. (2020) for pedagogical introductions to the motivating theory of rough paths.

2.1 THE SIGNATURE TRANSFORM

Definition 1. Let $\mathbb{R}^{d_1} \otimes \mathbb{R}^{d_2} \otimes \cdots \otimes \mathbb{R}^{d_n}$ denote the space of all real tensors with shape $d_1 \times d_2 \times \cdots \times d_n$. There is a corresponding binary operation $\otimes$, called the tensor product, which maps a tensor of shape $(d_1, \ldots, d_n)$ and a tensor of shape $(e_1, \ldots, e_m)$ to a tensor of shape $(d_1, \ldots, d_n, e_1, \ldots, e_m)$ via $(A_{i_1,\ldots,i_n}, B_{j_1,\ldots,j_m}) \mapsto A_{i_1,\ldots,i_n} B_{j_1,\ldots,j_m}$. For example when applied to two vectors, it reduces to the outer product.

Let $\left(\mathbb{R}^d\right)^{\otimes k} = \mathbb{R}^d \otimes \cdots \otimes \mathbb{R}^d$, and $v^{\otimes k} = v \otimes \cdots \otimes v$ for $v \in \mathbb{R}^d$, in each case with $k-1$ many $\otimes$.

Definition 2. Let $N \in \mathbb{N}$. The *signature transform to depth* N is defined as

$$\mathrm{Sig}^N \colon \left\{ f \in C([0,1]; \mathbb{R}^d) \;\middle|\; f \text{ differentiable} \right\} \to \prod_{k=1}^{N} \left(\mathbb{R}^d\right)^{\otimes k},$$

$$\mathrm{Sig}^N(f) = \left(\int \cdots \int_{0 < t_1 < \cdots < t_k < 1} \frac{\mathrm{d}f}{\mathrm{d}t}(t_1) \otimes \cdots \otimes \frac{\mathrm{d}f}{\mathrm{d}t}(t_k) \, \mathrm{d}t_1 \cdots \mathrm{d}t_k \right)_{1 \leq k \leq N}. \tag{1}$$

Most texts define the signature transform using the notation of stochastic calculus. Here, we sacrifice some generality (that is not needed in this context) in favour of more widely-used notation.[2]

The signature transform may naturally be extended to sequences of data.

Definition 3. The space of sequences of data over a set V is

$$\mathcal{S}(V) = \{\mathbf{x} = (x_1, \ldots, x_L) \mid L \in \mathbb{N},\, x_i \in V \text{ for all } i\}\ .$$

An *interval* of $(x_1, \ldots, x_L) \in \mathcal{S}(V)$ is $(x_i, \ldots, x_j) \in \mathcal{S}(V)$ for some $1 \leq i < j \leq L$.

Definition 4. Let $\mathbf{x} = (x_1, \ldots, x_L) \in \mathcal{S}(\mathbb{R}^d)$ with $L \geq 2$. Let $f\colon [0,1] \to \mathbb{R}^d$ be the unique continuous piecewise affine function such that $f(\frac{i-1}{L-1}) = x_i$ for all i, and is affine on the pieces in between. Let $N \in \mathbb{N}$. Then define $\mathrm{Sig}^N(\mathbf{x}) = \mathrm{Sig}^N(f)$. In this way we interpret Sig^N as a map

$$\mathrm{Sig}^N\colon \mathcal{S}(\mathbb{R}^d) \to \prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k}\ .$$

Note that the choice of $\frac{i-1}{L-1}$ is unimportant; any L points in $[0,1]$ would suffice, and in fact the definition is invariant to this choice (Bonnier et al., 2019, Definition A.10).

2.2 The grouplike structure

With $A_0 = B_0 = 1 \in \mathbb{R}$ on the right hand side, define $\boxtimes$ by[3]

$$\boxtimes\colon \left(\prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k}\right) \times \left(\prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k}\right) \to \prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k},$$

$$(A_1, \ldots A_N) \boxtimes (B_1, \ldots, B_N) \mapsto \left(\sum_{j=0}^{k} A_j \otimes B_{k-j}\right)_{1 \leq k \leq N}.$$

Chen's identity (Lyons et al., 2004, Theorem 2.9) states that the image of the signature transform forms a noncommutative group with respect to $\boxtimes$. That is, given a sequence of data $(x_1, \ldots, x_L) \in \mathcal{S}(\mathbb{R}^d)$ and some $j \in \{2, \ldots, L-1\}$, then

$$\mathrm{Sig}^N((x_1, \ldots, x_L)) = \mathrm{Sig}^N((x_1, \ldots, x_j)) \boxtimes \mathrm{Sig}^N((x_j, \ldots, x_L)). \tag{2}$$

Furthermore the signature of a sequence of length two may be computed explicitly from the definition. Letting

$$\exp\colon \mathbb{R}^d \to \prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k}, \qquad \exp\colon v \to \left(v, \frac{v^{\otimes 2}}{2!}, \frac{v^{\otimes 3}}{3!}, \ldots, \frac{v^{\otimes N}}{N!}\right),$$

then

$$\mathrm{Sig}^N((x_1, x_2)) = \exp(x_2 - x_1).$$

With Chen's identity, this implies that the signature transform may be computed by evaluating

$$\mathrm{Sig}^N((x_1, \ldots, x_L)) = \exp(x_2 - x_1) \boxtimes \exp(x_3 - x_2) \boxtimes \cdots \boxtimes \exp(x_L - x_{L-1}). \tag{3}$$

2.3 The logsignature, inverted signature, and inverted logsignature

The group inverse we denote $^{-1}$. Additionally a notion of logarithm may be defined (Liao et al., 2019), where

$$\log\colon \mathrm{image}\left(\mathrm{Sig}^N\right) \to \prod_{k=1}^{N} (\mathbb{R}^d)^{\otimes k}\ . \tag{4}$$

[2] Additionally, many texts also include a $k = 0$ term, which is defined to equal one. We omit this as it does not carry any information, and is therefore irrelevant to the task of machine learning.

[3] Most texts use $\otimes$ rather than $\boxtimes$ to denote this operation, as it may be regarded as an generalisation of the tensor product. That will not be important to us, however, so we use differing notation to aid interpretation.

This then defines the notions of *inverted signature transform*, *logsignature transform* and *inverted logsignature transform* as

$$\begin{aligned}\mathrm{InvertSig}^N(\mathbf{x}) &= \mathrm{Sig}^N(\mathbf{x})^{-1},\\ \mathrm{LogSig}^N(\mathbf{x}) &= \log\left(\mathrm{Sig}^N(\mathbf{x})\right),\\ \mathrm{InvertLogSig}^N(\mathbf{x}) &= \log\left(\mathrm{Sig}^N(\mathbf{x})^{-1}\right)\end{aligned}$$

respectively. We emphasise that the inverted signature or logsignature transforms are not the inverse maps of the signature or the logsignature transforms.

The logsignature transform extracts the same information as the signature transform, but represents the information in a much more compact way, as $\mathrm{image}(\log)$ is a proper subspace[4] of $\prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k}$. Its dimension is $w(d,N) = \sum_{k=1}^N \frac{1}{k}\sum_{i|k}\mu\left(\frac{k}{i}\right)d^i$, which is known as Witt's formula (Lothaire, 1997). μ is the Möbius function.

2.4 Signatures in machine learning

In terms of the tensors used by most machine learning frameworks, then the (inverted) signature and logsignature transforms of depth N may both be thought of as consuming a tensor of shape (b, L, d), corresponding to a batch of b different sequences of data, each of the form $(x_1, \ldots, x_L)$ for $x_i \in \mathbb{R}^d$. The (inverted) signature transform then produces a tensor of shape $(b, \sum_{k=1}^N d^k)$, whilst the (inverted) logsignature transform produces a tensor of shape $(b, w(d,N))$.

All of these transforms are in fact differentiable with respect to $\mathbf{x}$, and so may be backpropagated through. These transforms may thus be thought of as differentiable operations between tensors, in the way usually performed by machine learning frameworks.

3 Algorithmic improvements

We present several noveral algorithmic improvements for computing signatures and logsignatures.

3.1 Fused multiply-exponentiate

Recall from equation (3) that the signature may be computed by evaluating several exponentials and several $\boxtimes$. We begin by finding that it is beneficial to compute

$$\begin{aligned}\left(\prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k}\right) \times \mathbb{R}^d &\to \prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k},\\ A, z &\mapsto A \boxtimes \exp(z)\end{aligned}$$

as a fused operation. Doing so has uniformly (over d, N) fewer scalar multiplications than the composition of the individual exponential and $\boxtimes$, and in fact reduces the asymptotic complexity of this operation from $\mathcal{O}(Nd^N)$ to $\mathcal{O}(d^N)$. Furthermore this rate is now optimal, as the size of result (an element of $\prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k}$), is itself of size $\mathcal{O}(d^N)$.

The bulk of a signature computation may then be sped up by writing it in terms of this fused operation. See equation (3): a single exponential is required at the start, followed by a reduction with respect to this fused multiply-exponentiate. This gives substantial real-world speedups; see the benchmarks of Section 5.

The fusing is done by expanding

$$A \boxtimes \exp(z) = \left(\sum_{i=0}^k A_i \otimes \frac{z^{\otimes(k-i)}}{(k-i)!}\right)_{1 \le k \le N},$$

at which point the k-th term may be computed by a scheme in the style of Horner's method:

[4] $\log$ is actually a bijection. $\mathrm{image}\left(\mathrm{Sig}^N\right)$ is some curved manifold in $\prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k}$, and $\log$ is the map that straightens it out into a linear subspace.

$$\sum_{i=0}^{k} A_i \otimes \frac{z^{\otimes(k-i)}}{(k-i)!} = \left(\left(\cdots\left(\left(\frac{z}{k} + A_1\right) \otimes \frac{z}{k-1} + A_2\right) \otimes \frac{z}{k-2} + \cdots\right) \otimes \frac{z}{2} + A_{k-1}\right) \otimes z + A_k. \tag{5}$$

See Appendix A.1 for the the mathematics, including proofs of both the asymptotic complexity and the uniformly fewer multiplications.

3.2 Improved precomputation strategies

Given a sequence of data $\mathbf{x} = (x_1, \ldots, x_L)$, it may be desirable to query $\mathrm{Sig}^N((x_i, \ldots, x_j))$ for many different pairs i, j. We show that this query may be computed in just $\mathcal{O}(1)$ (in L) time and memory by using $\mathcal{O}(L)$ precomputation and storage. Previous theoretical work has achieved only $\mathcal{O}(\log L)$ inference with $\mathcal{O}(L \log L)$ precomputation (Chafai & Lyons, 2005).

Doing so is surprisingly simple. Precompute $\mathrm{Sig}^N((x_1, \ldots, x_j))$ and $\mathrm{InvertSig}^N((x_1, \ldots, x_j))$ for all j. This may be done in only $\mathcal{O}(L)$ work, by iteratively computing each signature via

$$\mathrm{Sig}^N((x_1, \ldots, x_j)) = \mathrm{Sig}^N((x_1, \ldots, x_{j-1})) \boxtimes \mathrm{Sig}^N((x_{j-1}, x_j)), \tag{6}$$

with a similar relation for the inverted signature. Then, at inference time use the group-like structure

$$\mathrm{Sig}^N((x_i, \ldots, x_j)) = \mathrm{InvertSig}^N((x_1, \ldots, x_i)) \boxtimes \mathrm{Sig}^N((x_1, \ldots, x_j)),$$

followed by a $\log$ if it is a logsignature that is desired. As a single operation this is $\mathcal{O}(1)$ in L.

We do remark that this should be used with caution, and may suffer from numerical stability issues when used for large i, j.

3.3 More efficient logsignature basis

The logsignature transform of a path has multiple possible representations, corresponding to different possible bases of the ambient space, which is typically interpreted as a free Lie algebra (Reutenauer, 1993). The *Lyndon basis* is a typical choice (Reizenstein & Graham, 2018).

We show that there exists a more computationally efficient basis. It is mathematically unusual, as it is not constructed as a Hall basis. But if doing deep learning, then the choice of basis is (mostly) unimportant if the next operation is a learnt linear transformation.

The Lyndon basis uses *Lyndon brackets* as its basis elements. Meanwhile our new basis uses basis elements that, when written as a sum of Lyndon brackets and expanded as a sum of words, have precisely one word as a Lyndon word. This means that the coefficient of this basis element can be found cheaply, by extracting the coeffient of that Lyndon word from the tensor algebra representation of the logsignature. See Appendix A.2 for the full exposition.

4 New features

Signatory provides several features not available in previous libraries.

4.1 Inverted signatures and logsignatures

Signatory provides the capability to compute inverted signatures and logsignatures, via the optional `inverse` argument to the `signature` and `logsignature` functions.

4.2 Parallelism

There are two main levels of parallelism. First is naïve parallelism over the batch dimension. Second, we observe that equation (3) takes the form of a noncommutative reduction with respect to the

fused multiply-exponentiate. The operation $\boxtimes$ is associative, and so this may be parallelised in the usual way for reductions, by splitting the computation up into chunks.

Parallelism on the CPU is implemented with OpenMP. For speed the necessary operations are written in terms of C++ primitives, and then bound into PyTorch.

4.3 GPU SUPPORT

An important feature is GPU support, which is done via the functionality available through LibTorch. It was a deliberate choice not to write CUDA code; this is due to technical reasons for distributing the library in a widely-compatible manner. See Appendix B.

There are again two levels of parallelism, as described in the previous section.

When the (log)signature transform is part of a deep learning model trained on the GPU, then GPU support offers speedups not just from the use of a GPU over the CPU, but also obviates the need for copying the data to and from the GPU.

4.4 BACKPROPAGATION

Crucial for any library used in deep learning is to be able to backpropagate through the provided operations. Signatory provides full support for backpropagation through every provided operation. There has previously only been limited support for backpropagation through a handful of simple operations, via the `iisignature` library (Reizenstein & Graham, 2018).

The backpropagation computations are handwritten, rather than being generated autodifferentiably. This improves the speed of the computation by using C++ primitives rather than high-level tensors, and furthermore allows for improved memory efficiency, by exploiting a reversibility property of the signature (Reizenstein, 2019, Section 4.9.3).

4.5 EXPLOITING THE GROUPLIKE STRUCTURE

It is often desirable to compute (inverted) (log)signatures over multiple intervals of the same sequence of data. These calculations may jointly be accomplished more efficiently than by evaluating the signature transform over every interval separately. In some cases, if the original data has been discarded and only its signature is now known, exploiting this structure is the only way to perform the computation.

Here we detail several notable cases, and how Signatory supports them. In all cases the aim is to provide a flexible set of tools that may be used together, so that wherever possible unecessary recomputation may be elided. Their use is also discussed in the documentation, including examples.

4.5.1 COMBINING ADJACENT INTERVALS

Recall equation (2). If the two signatures on the right hand side of the equation are already known, then the signature of the overall sequence of data may be computed using only a single $\boxtimes$ operation, without re-iterating over the data.

This operation is provided for by the `multi_signature_combine` and `signature_combine` functions.

4.5.2 EXPANDING INTERVALS

Given a sequence of data $(x_1, \ldots, x_L) \in \mathcal{S}\left(\mathbb{R}^d\right)$, a scenario that is particularly important for its use in Section 3.2 is to compute the signature of expanding intervals of the data,[5]

$$(\mathrm{Sig}^N((x_1, x_2)), \mathrm{Sig}^N((x_1, x_2, x_3)), \ldots, \mathrm{Sig}^N((x_1, \ldots, x_L))).$$

This may be interpreted as a sequence of signatures, that is to say an element of $\mathcal{S}\left(\prod_{k=1}^N \left(\mathbb{R}^d\right)^{\otimes k}\right)$.

[5]Note that we start with $\mathrm{Sig}^N((x_1, x_2))$, as two is the shortest a sequence of data can be to define a path; see Definition 4.

By equation (6), this may be done efficiently in only $\mathcal{O}(L)$ work, and with all the earlier signatures available as byproducts, for free, of computing the final element $\mathrm{Sig}^N((x_1, \ldots, x_L))$.

This is handled by the optional `stream` argument to the `signature` function and to the `logsignature` function.

4.5.3 ARBITRARY INTERVALS

Given a sequence of data $\mathbf{x} = (x_1, \ldots, x_L)$, it may be desirable to query $\mathrm{Sig}^N((x_i, \ldots, x_j))$ for many i, j such that $1 \leq i < j \leq L$, as in Section 3.2. Using the efficient precomputation strategy described there, Signatory provides this capability via the `Path` class.

4.5.4 KEEPING THE SIGNATURE UP-TO-DATE

Suppose we have a sequence of data $(x_1, \ldots, x_L) \in \mathcal{S}\left(\mathbb{R}^d\right)$ whose signature $\mathrm{Sig}^N((x_1, \ldots, x_L))$ has already been computed. New data subsequently arrives, some $(x_{L+1}, \ldots, x_{L+M}) \in \mathcal{S}\left(\mathbb{R}^d\right)$, and we now wish to update our computed signature, for example to compute the sequence of signatures

$$(\mathrm{Sig}^N((x_1, \ldots, x_{L+1})), \ldots, \mathrm{Sig}^N((x_1, \ldots, x_{L+M}))). \tag{7}$$

This could done as in Section 4.5.1, by computing the signatures over $(x_{L+1}, \ldots, x_{L+M})$, and combining them together. However, if these signatures (over $(x_{L+1}, \ldots, x_{L+M})$) are not themselves of interest, then this approach may be improved upon, as Section 4.5.1 only exploits the grouplike structure, but not the fused multiply-exponentiate described in Section 3.1.

Computing them in this more efficient way may be handled via the `basepoint` and `initial` arguments to the `signature` function, and via the `update` method of the `Path` class.

5 BENCHMARK PERFORMANCE

We are aware of two existing software libraries providing similar functionality, `esig` (Lyons, 2017) and `iisignature` (Reizenstein & Graham, 2018). We ran a series of benchmarks against the latest versions of both of these libraries, namely `esig` 0.6.31 and `iisignature` 0.24. The computer used was equipped with a Xeon E5-2960 v4 and a Quadro GP100, and was running Ubuntu 18.04 and Python 3.7.

For `esig` and `iisignature` we report run time on the CPU, whilst for Signatory we report run time on the GPU, CPU with parallelism, and CPU without parallelism. As it is in principle possible to parallelise these alternative libraries using Python's `multiprocessing` module[6], the most important comparisons are to Signatory on the CPU without parallelism (representing like-for-like computational resources), and on the GPU (representing the best possible performance).

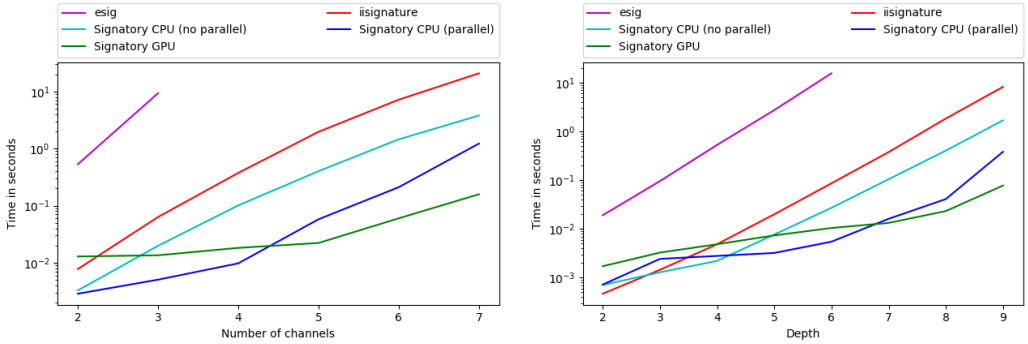
We begin with a benchmark for the forward operation through the signature transform. We consider a batch of 32 sequences, of length 128. We then investigate the scaling as we vary either the number of channels (over 2–7) in the input sequences, or the depth (over 2–9) of the signature transform. For varying channels, the depth was fixed at 7. For varying depths, the channels was fixed at 4.

Every test case is repeated 50 times and the fastest time taken. See Figure 1. Note the logarithmic scale.

We observe that `iisignature` is Signatory's strongest competitor in all cases. Signatory and `iisignature` are comparable for the very smallest of computations. As the computation increases in size, then the CPU implementations of Signatory immediately overtake `iisignature`, followed by the GPU implementation.

For larger computations, Signatory can be orders of magnitude faster. For example, to compute the signature transform with depth and number of channels both equal to 7, then `iisignature` takes 20.9 seconds to perform the computation. In contrast, running on the CPU without parallelism, Signatory takes only 3.8 seconds, which represents a 5.5× speedup. We emphasise that the same computational resources (including lack of parallelism) were used for both. To see the benefits

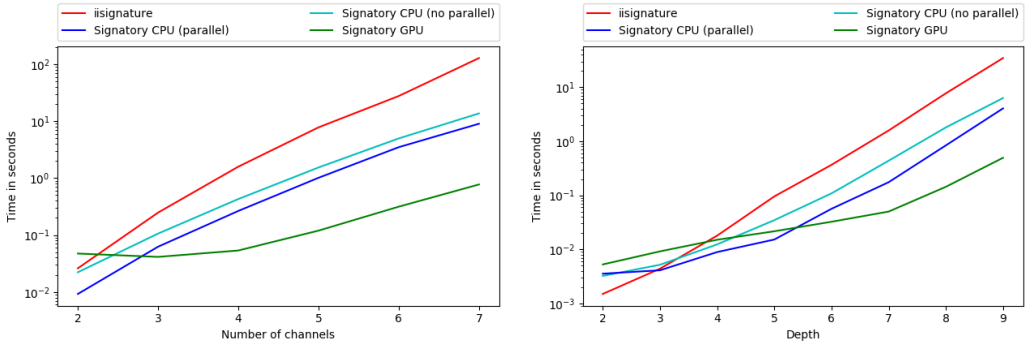
[6]Subject to nontrivial overhead.



(a) Signature forward, varying channels

(b) Signature forward, varying depths

Figure 1: Time taken on benchmark computations to compute the signature transform. `esig` is only shown for small operations as it is incapable of larger operations. Note the logarithmic scale.



(a) Signature backward, varying channels

(b) Signature backward, varying depths

Figure 2: Time taken on benchmark computations to backpropagate through the signature transform. `esig` is not shown as it is incapable of computing backward operations. Note the logarithmic scale.

of a GPU implementation over a CPU implementation—the primary motivation for Signatory's existence—then we observe that Signatory takes only 0.16 seconds to compute this same operation. Compared to the best previous alternative in `iisignature`, this represents a 132× speedup.

Next, we consider the backward operation through the signature transform. We vary over multiple inputs as before. See Figure 2. We again observe the same behaviour. `iisignature` is Signatory's strongest competitor, but is still orders of magnitude slower on anything but the very smallest of problems. For example, to backpropagate through the signature transform, with depth and number of channels both equal to 7, then Signatory on the CPU without parallelism takes 13.7 seconds. Meanwhile, `iisignature` takes over 2 minutes – 128 seconds – to perform this computation on like-for-like computational resources. Running Signatory on the GPU takes a fraction of a second, specifically 0.772 seconds. These represent speedups of 9.4× and 166× respectively.

For further speed benchmarks, a discussion on memory usage benchmarks, the precise numerical values of the graphs presented here, and code to reproduce these benchmarks, see Appendix C. We observe the same consistent improvements on these additional benchmarks.

6 Conclusion

We have introduced Signatory, a library for performing functionality related to the signature and logsignature transforms, with a particular focus on applications to machine learning. Notable contributions are the speed of its operation, its GPU support, differentiability of every provided operation, and its novel algorithmic innovations.

REFERENCES

Patric Bonnier, Patrick Kidger, Imanol Pérez Arribas, Cristopher Salvi, and Terry Lyons. Deep signature transforms. In *Neural Information Processing Systems*, 2019.

Djalil Chafai and Terry Lyons. Coropa project, 2005. `https://coropa.sourceforge.io/rde/`.

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

Adeline Fermanian. Embedding and learning with signatures. *arXiv:1911.13211*, 2019.

Liam Hodgkinson, Chris van der Heide, Fred Roosta, and Michael Mahoney. Stochastic Normalizing Flows. *arXiv:2002.09547*, 2020.

Patrick Kidger. `torchcde`, 2020. `https://github.com/patrick-kidger/torchcde`.

Patrick Kidger, James Morrill, and Terry Lyons. Generalised Interpretable Shapelets for Irregular Time Series. *arXiv:2005.13948*, 2020.

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

A. B. Kormilitzin, K. E. A. Saunders, P. J. Harrison, J. R. Geddes, and T. J. Lyons. Application of the signature method to pattern recognition in the cequel clinical trial. *arXiv:1606.02074*, 2016.

Pierre Lalonde and Arun Ram. Standard lyndon bases of lie algebras and enveloping algebras. *Transactions of the American Mathematical Society*, 347(5):1821–1830, 1995.

D. Levin, T. Lyons, and H. Ni. Learning from the past, predicting the statistics for the future, learning an evolving system. *arXiv:1309.0260*, 2013.

Chenyang Li, Xin Zhang, and Lianwen Jin. LPSNet: a novel log path signature feature based hand gesture recognition framework. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 631–639, 2017.

Shujian Liao, Terry Lyons, Weixin Yang, and Hao Ni. Learning stochastic differential equations using rnn with log signature features. 2019.

M. Lothaire. Combinatorics on words. 1997.

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

Terry Lyons. esig, 2017. URL `https://esig.readthedocs.io/`.

Terry Lyons, Michael Caruana, and Thierry Levy. *Differential equations driven by rough paths*. Springer, 2004. École d'Été de Probabilités de Saint-Flour XXXIV - 2004.

Terry J Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14(2):215–310, 1998.

Ming Min and Tomoyuki Ichiba. Convolutional Signatures for Sequential Data. *arXiv:2009.06719*, 2020.

Michael Moor, Max Horn, Christian Bock, Karsten Borgwardt, and Bastian Rieck. Path Imputation Stratgies for Signature Models. *arXiv:2005.12359*, 2020.

James Morrill, Adeline Fermanian, Patrick Kidger, and Terry Lyons. A Generalised Signature Method for Time Series. *arXiv:2006.00873*, 2020a.

James Morrill, Patrick Kidger, Cristopher Salvi, James Foster, and Terry Lyons. Neural CDEs for Long Time-Series via the Log-ODE Method. *arXiv:2009.08295*, 2020b.

James Lewis Morrill, Andrey Kormilitzin, Alejo J Nevado-Holgado, Sumanth Swaminathan, Sam Howison, and Terry Lyons. The signature-based model for early detection of sepsis from electronic health records in the intensive care unit. In *International Conference in Computing in Cardiology 2019 (CINC 2019)*, 2019.

Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional Sig-Wasserstein GANs for Time Series Generation. *arXiv:2006.05421*, 2020.

Imanol Perez Arribas, Guy M. Goodwin, John R. Geddes, Terry Lyons, and Kate E. A. Saunders. A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder. *Translational Psychiatry*, 8(1):274, 2018.

Imanol Perez Arribas, Cristopher Salvi, and Lukasz Szpruch. Sig-SDEs model for quantitative finance. *arXiv:2006.00218*, 2020.

Jeremy Reizenstein. Iterated-integral signatures in machine learning. 2019. URL `http://wrap.warwick.ac.uk/131162/`.

Jeremy Reizenstein and Benjamin Graham. The iisignature library: efficient calculation of iterated-integral signatures and log signatures. *arXiv:1802.08252*, 2018.

Christophe Reutenauer. *Free Lie Algebras*. Clarendon Press, 1993.

Csaba Toth and Harald Oberhauser. Variational Gaussian Processes with Signature Covariances. *arXiv:1906.08215*, 2019.

Weixin Yang, Lianwen Jin, and Manfei Liu. Deepwriterid: An end-to-end online text-independent writer identification system. *IEEE Intelligent Systems*, 31(2):45–53, 2016a.

Weixin Yang, Lianwen Jin, Hao Ni, and Terry Lyons. Rotation-free online handwritten character recognition using dyadic path signature features, hanging normalization, and deep neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 4083–4088. IEEE, 2016b.