SigKAN: Signature-Weighted Kolmogorov-Arnold Networks for Time Series

Hugo Inzirillo¹ and Rémi Genet²

¹CREST-ENSAE, Institut Polytechnique de Paris ²DRM, Université Paris Dauphine - PSL

June 27, 2024

Abstract

We propose a novel approach that enhances multivariate function approximation using learnable path signatures and Kolmogorov-Arnold networks (KANs). We enhance the learning capabilities of these networks by weighting the values obtained by KANs using learnable path signatures, which capture important geometric features of paths. This combination allows for a more comprehensive and flexible representation of sequential and temporal data. We demonstrate through studies that our SigKANs with learnable path signatures perform better than conventional methods across a range of function approximation challenges. By leveraging path signatures in neural networks, this method offers intriguing opportunities to enhance performance in time series analysis and time series forecasting, among other fields.

1 Introduction

Researchers are very interested in multivariate time series forecasting Wei [2018], Marcellino et al. [2006], and it has grown to be a significant field of study for many businesses. Since there is a growing amount of data available, it makes sense to suggest increasingly sophisticated prediction frameworks. Multivariate time series include multiple interdependent variables, as opposed to univariate time series, which evaluate a single variable over time. The learning task is made considerably more challenging by these relationships. Complex models that can capture temporal dependencies and dynamic interactions across several dimensions and time horizons are necessary for handling multivariate time series data. Researchers investigated a range of techniques to address the difficulties related to multivariate time series forecasting, such as neural networks Tang et al. [1991], state-space models Hamilton [1994], Kim [1994], vector autoregressive models Zivot and Wang [2006], Lütkepohl [2013], and neural networks

Binkowski et al. [2018], Ma and Leung [2019]. Due to its capacity to manage non-linearities and long-range dependencies in data, deep learning techniques like Long Short-Term Memory (LSTM) networks and attention mechanisms have demonstrated encouraging results Hochreiter and Schmidhuber [1997], Vaswani et al. [2017]. Recently, the Kolmogorov-Arnold Network (KAN) Liu et al. [2024] stands out due to its theoretical foundation in the Kolmogorov-Arnold representation theorem Kolmogorov [1961] This theorem ensures that any multivariate continuous function can be decomposed into a sum of continuous one-dimensional functions, which is used by KAN to effectively approximate complex functions. Despite their theoretical robustness, there is still much room to improve the expressiveness and efficiency of KANs, especially when processing sequential and temporal data. Even more recently, some research proposed to extend KANs using wavelets Bozorgasl and Chen [2024], or proposed new framework to incorporate KAN within RRNs Genet and Inzirillo [2024b]. Some researcher proposed framework to use KANs in time series analysis Vaca-Rubio et al. [2024], and model architecture for time series forecasting using attention Genet and Inzirillo [2024a]. When dealing with sequential and temporal data, the importance of context and feature representation is crucial for accurate and reliable predictions. These highly complex models require a "context" form of representation, summarizing the compact information in order to improve predictions. Path signatures, originating from rough path theory, offer a powerful method for encoding the essential features of paths or trajectories. These signatures capture the underlying geometry of the paths through a sequence of iterated integrals, making them particularly well-suited for tasks involving complex sequential data. Integrating path signatures into neural network architectures has shown promise in various applications, but their potential has not been fully explored in the context of KANs. Path signatures, first described by Chen [1958], are collections of iterated integrals of (transformed) time series, in this case, a series of prices for digital assets. We redirect the reader to Lyons [2014, 1998], Chevyrev and Kormilitzin [2016] for a thorough explanation of path signatures as a trustworthy representation or a set of characteristics for unparameterized paths. A path signature, also referred to as a signature for short, is essentially a mathematical expression that "succinctly" captures the structure of a path. While it originated from stochastic analysis and rough path theory, it has recently been applied to a wide range of other fields, including computer vision (Yang et al. [2022], Li et al. [2017]), time series analysis (Gyurkó et al. [2013], Dyer et al. [2021]), machine learning (Chevyrev and Kormilitzin [2016], Perez Arribas et al. [2018], Fermanian [2021]), etc. In this paper, we introduce learnable path signatures as a novel improvement to the original KAN layer. We propose a learnable path signature layer that uses learnable parameters to compute path signatures for each input path. The KAN framework then combines these path characteristics with conventional linear transformations. The goal of this integration is to enhance KANs' approximation skills by utilizing the rich geometric information that path signatures provide. Codes are available at SigKAN and can be installed using the following command: pip install sigkan. Additionaly we release a package needed to compute the path signature using Tensorfow v2.0. iisignaturetensorflow-2 and can be installed using the following command: pip install iisignature-tensorflow-2. This package allows signatures to be trainable in custom tensorflow layer, although it does not support GPU utilization.

2 Kolmogorov-Arnold Networks (KANs)

Multi-Layer Perceptrons (MLPs) Hornik et al. [1989] extension of original percetron proposed by Rosenblatt [1958], are inspired by the universal approximation theorem Cybenko [1989] which states that a feed-forward network (FFN) with single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of \mathbb{R}^n . Kolmogorov-Arnold Network (KAN) focus on the Kolmogorov-Arnold representation theorem Kolmogorov [1961]. The Kolmogorov-Arnold representation theorem states that any multivariate continuous function can be represented as a composition of univariate functions and the addition operation,

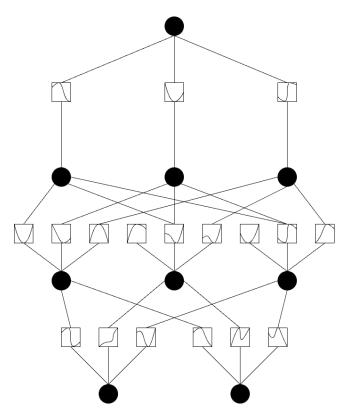


Figure 1: KAN: Kolmogorov Arnold Networks

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$$
 (1)

where $\phi_{q,p}$ are univariates functions that map each input variable (x_p) such $\phi_{q,p}:[0,1]\to\mathbb{R}$ and $\phi_q:\mathbb{R}\to\mathbb{R}$. Since all functions to be learned are univariate functions, we can parametrize each 1D function as a B-spline curve, with learnable coefficients of local B-spline basis functions. The key insight comes when we see the similarities between MLPs and KAN. In MLPs, a layer includes a linear transformation followed by nonlinear operations, and you can make the network deeper by adding more layers. Let us define what a KAN layer is

$$\Phi = {\phi_{q,p}}, \quad p = 1, 2, \dots, n_{\text{in}}, \quad q = 1, 2, \dots, n_{\text{out}},$$
(2)

where $\phi_{q,p}$ are parametrized function of learnable parameters. In the Kolmogov-Arnold theorem, the inner functions form a KAN layer with $n_{\rm in}=n$ and $n_{\rm out}=2n+1$, and the outer functions form a KAN layer with $n_{\rm in}=2n+1$ and $n_{\rm out}=1$. So the Kolmogorov-Arnold representations in Eq. (1) are simply compositions of two KAN layers. Now it becomes clear what it means to have Deep Kolmogorov-Arnold representation. Taking the notation from Liu et al. [2024] let us define a shape of KAN $[n_0, n_1, \cdots, n_L]$, where n_i is the number of nodes in the $i^{\rm th}$ layer of the computational graph. We denote the $i^{\rm th}$ neuron in the $l^{\rm th}$ layer by (l,i), and the activation value of the (l,i)-neuron by $x_{l,i}$. Between layer l and layer l+1, there are $n_l n_{l+1}$ activation functions: the activation function that connects (l,i) and (l+1,j) is denoted by

$$\phi_{l,i,i}, \quad l = 0, \dots, L - 1, \quad i = 1, \dots, n_l, \quad j = 1, \dots, n_{l+1}.$$
 (3)

The pre-activation of $\phi_{l,j,i}$ is simply $x_{l,i}$; the post-activation of $\phi_{l,j,i}$ is denoted by $\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i})$. The activation value of the (l+1,j) neuron is simply the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \qquad j = 1, \dots, n_{l+1}.$$
 (4)

Rewriting it under the matrix form will give,

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_{l}}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_{l}}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_{l}}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_{l}} \mathbf{x}_{l}, \tag{5}$$

where Φ_l is the function matrix corresponding to the l^{th} KAN layer. A general KAN network is a composition of L layers: given an input vector $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, the output of KAN is

$$KAN(\mathbf{x}) = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}. \tag{6}$$

3 Architecture

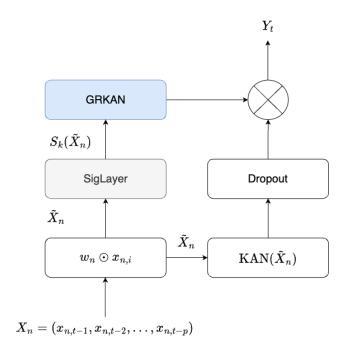


Figure 2: SigKAN

In this section, we will dive into the various components of the network architecture. We will detail each layer and mechanism. We aim to provide a comprehensive understanding of how the model operates and processes data to achieve its predictive capabilities. Before starting we define the main components of the SigKAN

- Gated Residual KAN: This layer allows to modulate the information flow during the learning task.
- Learnable path signature layer: this layer will compute the path signatures for each path with learnable scaling parameters.

In simple words, the Signature KANs (SigKANs) is composed of SigLayer and KAN layer. As path signature capture important geometric features of paths, combine the set of information with the output of KAN layer will enhance the power of prediction. Let us define a N-dimensional path $(X_t)_{t \in [0,T]}$. In other words, $X_t = (X_{1,t}, ..., X_{N,t})$. The 1-dimensional coordinate paths will be denoted as $(X_{n,t})$, $n \in \{1, ..., N\}$. To build the path signature of (X_t) , we first consider the increments of $X^1, ..., X^N$ over any interval [0,t], $t \in [0,T]$, which

are denoted $S(X)^1_{0,t},\dots,S(X)^N_{0,t}$ and defined as follows:

$$S(X)_{0,t}^{n} = \int_{0}^{t} dX_{s}^{n}.$$
 (7)

 $S(X)_{0,t}^n$ is the first stage to calculate the signature of an unidimensional path. It is a sequence of real numbers, each of these numbers corresponding to an iterated integral of the path. Then, the next signature coefficients will involve two paths: a coordinate path (X_t^m) and the "increment path $(S(X)_{0,t}^n)$ associated to the coordinate path (X_t^n) . There are N^2 such second order integrals, which are denoted $S(X)_{0,t}^{1,1}\ldots,S(X)_{0,t}^{N,N}$, where

$$S(X)_{0,t}^{n,m} = \int_0^t S(X)_{0,s}^n dX_s^m, \ n, m \in \{1, \dots, N\}.$$
 (8)

The set of first (resp. second) order integrals involves N (resp. N^2) integrals. The latter first and second order integrals are called the first and second levels of path signatures respectively. Iteratively, we obtain N^k integrals of order k, which is denoted $S(X)_{0,t}^{i_1,\dots,i_k}$ for the k-th level of path signatures, when $i_j \in \{1,\dots,N\}$ and $j \in \{1,\dots,k\}$. More precisely, the k-th level signatures can be written as follows:

$$S(X)_{0,t}^{i_1,\dots,i_k} = \int_0^t S(X)_{0,s}^{i_1,\dots,i_{k-1}} dX_s^{i_k}.$$

The path signature $S(X)_{0,T}$ is finally the infinite ordered set of such terms when considering all levels $k \geq 1$ and the path on the whole interval [0,T]:

$$S(X)_{0,T} = (1, S(X)_{0,T}^{1}, S(X)_{0,T}^{2}, \dots, S(X)_{0,T}^{N}, S(X)_{0,T}^{1,1}, S(X)_{0,T}^{1,2}, \dots, S(X)_{0,T}^{N,N}, S(X)_{0,T}^{1,1,1}, \dots).$$

$$(9)$$

3.1 Gated Residual KANs

The relationship between temporal data is a key issue. Gated Residual Networks (GRNs) offer an efficient and flexible way of modelling complex relationships in data. They allow to control the flow of information and facilitate the learning tasks. They are particularly useful in areas where non-linear interactions and long-term dependencies are crucial. In our model we use the Gated Residual Kolmogorov Arnold Networks (GRKAN) inspired from the GRN proposed by Lim et al. [2021], we kept the same architecture. We propose a novel approach using two KAN layer, to control the information flow while briging more interpretability. Using GRKAN there is no more need for context, however, an additional linear layer is required to match the signature transform and the ouput of the gating mechanism.

$$GRN_{\omega}(x) = LayerNorm(x + GLU_{\omega}(\eta_1)),$$
 (10)

$$\eta_1 = KAN(\varphi_{n_1}(.), \eta_2), \tag{11}$$

$$\eta_2 = KAN(\varphi_{\eta_2}(.), x), \tag{12}$$

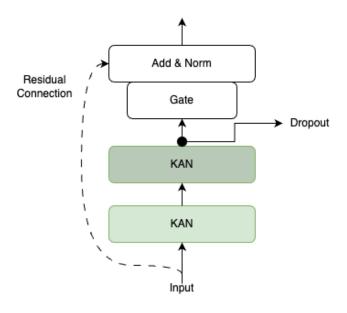


Figure 3: Gated Residual KAN (GRKAN)

In this context, the we used activation functions for KAN layers denoted, $\varphi_{\eta_1}(.)$ and $\varphi_{\eta_2}(.)$, SiLU Elfwing et al. [2018] and ELU Clevert et al. [2015], respectively, while $\eta_1 \in \mathbb{R}^{d_{model}}$ and $\eta_2 \in \mathbb{R}^{d_{model}}$ represent intermediate layers. The standard layer normalization LayerNorm is that described in Ba et al. [2016], and ω is an index used to indicate weight sharing. When the expression KAN(x) is largely positive, ELU activation works as an identity function. On the other hand, when this expression is largely negative, ELU activation produces a constant output, thus behaving like a linear layer. We use Gated Linear Units (GLUs) Dauphin et al. [2017] to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset. Letting $\gamma \in \mathbb{R}^{d_{model}}$ be the input, the GLU then takes the form:

$$GLU_{\omega}(\gamma) = \sigma(W_{4,\omega} \ \gamma + b_{4,\omega}) \odot (W_{5,\omega} \ \gamma + b_{5,\omega}), \tag{13}$$

where $\sigma(.)$ is the sigmoid activation function, $W_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}$, $b_{(.)} \in \mathbb{R}^{d_{model}}$ are the weights and biases, \odot is the element-wise Hadamard product, and d_{model} is the hidden state size. GLU allows to control the extent to which the GRN contributes to the original input x – potentially skipping over the layer entirely if necessary as the GLU outputs could be all close to 0 in order to suppress the nonlinear contribution.

3.2 Learnable Path Signature

For a path X_n represented as a sequence of points $(x_{n,1}, x_{n,2}, \ldots, x_{n,N})$, we transform each point with learnable coefficients:

$$\tilde{X}_{n,i} = w_n \odot x_{n,i} \tag{14}$$

where w_n is a learnable vector of coefficients and \odot denotes element-wise multiplication. The k-th order signature can be expressed as:

$$S(\tilde{X}) = \left(1, \left(\int_{0}^{1} d\tilde{X}_{t_{1}}^{i_{1}}\right)_{1 \leq i_{1} \leq N}, \left(\int_{0}^{1} \int_{0}^{t_{1}} d\tilde{X}_{t_{2}}^{i_{2}} d\tilde{X}_{t_{1}}^{i_{1}}\right)_{1 \leq i_{1}, i_{2} \leq N}, \dots,$$

$$\left(\int_{0}^{1} \int_{0}^{t_{1}} \cdots \int_{0}^{t_{k-1}} d\tilde{X}_{t_{k}}^{i_{k}} \cdots d\tilde{X}_{t_{2}}^{i_{2}} d\tilde{X}_{t_{1}}^{i_{1}}\right)_{1 \leq i_{1}, i_{2}, \dots, i_{k} \leq N}, \dots\right)$$

$$(15)$$

After performing this step, we will compute the path signature of each sequence of transformed points $(\tilde{X}_1,...,\tilde{X}_N)$. The transformed path \tilde{X}_1 is now a vector of transformed points $(\tilde{x}_{1,1},\tilde{x}_{1,2},...,\tilde{x}_{1,N})$. For N paths $(X_1,...,X_N)$ the path signature transformation

$$S(\tilde{X}) = \left[S(\tilde{X})_1, S(\tilde{X})_2, S(\tilde{X})_3, \dots \right], \tag{16}$$

The path signature obtained provides a detailed representation of the underlying path, capturing patterns and dependencies across the data. This can be highly informative for the prediction, and potentially leading to better performance. To do so, the ouput from (16) will be the input of a GRKAN 3, however it is possible to use another gated mechanism to control the flow of information, enabling the networks to focus on the most relevant features and suppress less useful ones.

$$h_s = f(S_k(\tilde{X})) \tag{17}$$

where f(.) is a succession of operations, in our case we choose to use a Gated Residual KAN (GRKAN), the output of this channel will weight the output of the KAN layers.

3.3 Output

We denoted seq and d_{out} , the sequence length and the output dimension, respectively. The output of the GRKAN is used as a weighting applied on the output of a KAN layer on (\tilde{X}) . To do so let's denote the output of the GRKAN as h_s , this one is then passed through a softmax activation to obtain:

$$\psi = \text{SoftMax}(h_s) \tag{18}$$

The output of a SigKAN layer then:

$$\psi \odot KAN(\tilde{X}) \tag{19}$$

It is to note that \tilde{X} is a matrix of shape (seq, d_{out}) , while $KAN(\tilde{X}) \in \mathbb{R}^{(seq, d_{out})}$ where the number of output is a parameter given to the layer. The KAN

transformation applied is the same on each element of the sequence. On the another hand $\psi \in \mathbb{R}^{d_{out}}$ is a vector obtained by a transformation of the signature in the GRKAN, and thoose weights are applied similarly at each elements of the sequences. The output of the layers still have the sequence dimension the SigKAN layers are thus stackable. Thus, we can generalize a SigKANs network such:

$$h_0 = x$$

 $h_j = \operatorname{SoftMax}(\operatorname{GRKAN}_j(S(\hat{h_{j-1}}))) \odot \operatorname{KAN}_j(h_{j-1}), \quad j = 1, 2, \dots, L \quad (20)$
 $y = h_L$

The output of the SigKANs still have the sequence dimension, and it would require to either flatten the output, only select the last outputs on this dimension, or apply a RNN that only return the last hidden state in order to remove this dimension in the network after it. Even if they are stackable, we have observed in our experiments that most of the time this is non-desireable as it doesn't yield to increase in performance, as well as increasing the computational cost quickly. Indeed as one can note the size of the signature depends on the input shape by a power of the degree used, this added to the fact that we do not have GPU supports on the signature parts clearly limits the interests of stacking many SigKAN layers in real application, even if this is possible and as described in the learning task beneficial in case where we do not use KAN layers but standard fully connected ones.

4 Learning Task

To demonstrate the performance benefits of our architecture, we have used two different tasks, one being the same as in Genet and Inzirillo [2024b] and Genet and Inzirillo [2024a] for easy comparison, the other being a task where these two tasks are more difficult to perform.

4.1 Task definitions and dataset

The first task we use involves predicting the notional amount traded on the market several times in advance. This is the task used in Genet and Inzirillo [2024b] and Genet and Inzirillo [2024a], and is interesting because volumes exhibit multiple patterns such as seasonality and autocorrelation. The second task changes only the target value; instead of trying to estimate future volume, we aim to predict future absolute return over several time steps in advance. The

tasks focus solely on the Binance exchange, so our dataset only contains data from this exchange, which has been the most important market for many years. We have also only used USDT markets, as this is the most widely used stablecoin, and all notionals are therefore intended in USDT. For the first task, our dataset

consists of the notional amounts traded each hour on several assets: BTC, ETH,

ADA, XMR, EOS, MATIC, TRX, FTM, BNB, XLM, ENJ, CHZ, BUSD, ATOM, LINK, ETC, XRP, BCH and LTC, which are to be used to predict just one of them, BTC. For the second task, we took the absolute percentage change between closing prices only on BTC to predict BTC. For both, the data period spans 3 years from January 1, 2020 to December 31, 2022.

4.2 Data preprocessing

Data preparation is a very important task in any machine learning task, and can largely alter the quality of the estimate obtained. It is important to have features with the same scale between them, or to have the same target scale over time. To achieve this for the first task, we use two-stage scaling. The first step is to divide the values in the series by the moving median of the last two weeks. This moving median window is also shifted by the number of steps forward we want to predict, so as not to include foresight. This first preprocessing aims to make the series more stationary over time. The second pre-processing applied is a simple MinMaxScaling per asset, even if here the minimum of the series is 0, it's simply a matter of dividing by the maximum value. The aim is to scale the data in the 0, 1 interval to avoid an explosive effect when learning due to the power exponent. However, this pre-processing is adjusted on the training set, the adjustment being limited to finding the maximum value for each series, which is then used directly on the test set. This means that on the test set, it is possible to have data greater than 1, but as no optimization is used, this is not a problem. For the second tasks, as there is only one asset and volatility is a more stationary process over time than volumes, we divide the values only by the maximum value of the ream, in order to have values between 0 and 1 only. Finally, we split our data for both set into a training set and a test set, with a

standard proportion of 80-20. This represents over 21,000 points in the training set and 5,000 in the test set.

4.3 Loss Function for Model Training

Since we have a numerical prediction problem for both tasks, we have opted to optimize our model using the root mean square error (RMSE) as the loss function, whose formula is simple:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)} \right)^{2},$$

where N represents the number of samples in the dataset, $\hat{X}_{t+1}^{(i)}$ denotes the predicted notional values of Bitcoin at time t+1 for the i-th sample, and $X_{t+1}^{(i)}$ are the corresponding true values. The first reason is that this is the most widely used and standard method in machine learning for this type of problem. The second reason is the metric we want to use to display the results, namely the R-squared R^2 . The R^2 is interesting as a metric because it not only gives

information on the error but also on the error given the variance of the estimated series, which means it's much easier to tell whether the model is performing well or not. It is also a measure widely used by econometricians and practitioners for this very reason. However, minimizing the MSE is exactly the same as maximizing the R^2 , as its formula indicates:

$$R^{2} = 1 - \frac{\sum_{i=1}^{N} (\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)})^{2}}{\sum_{i=1}^{N} (X_{t+1}^{(i)} - \bar{X}_{t+1})^{2}}.$$

As we can see, the upper terms of the quotient are equivalent to the sum of the squared errors, the other two components being fixed, the optimization of the mean squared error is totally similar to the optimization of \mathbb{R}^2 .

4.3.1 Note on training details

As in Genet and Inzirillo [2024b] and Genet and Inzirillo [2024a], metrics are calculated directly on scaled data and not on unscaled data. There are two reasons for this: firstly, MinMax scaling has no impact on the metric since the minimum is 0 and the data interval is [0, 1]; rescaling would simply have involved multiplying all the points, which would not have changed the R-squared. In terms of optimizing model details, we used the Adam optimizer, one of the most popular choices, used 20% of our training set as a validation set and included two training callbacks. The first is an early learning stop, which interrupts training after 6 consecutive periods without improvement on the validation set, and restores the weights associated with the best loss obtained on the validation set. The second is a reduction in the plateau learning rate, which halves the learning rate after 3 consecutive periods without improvement on the validation set. Finally, the sequence length given to all models for both tasks is set at a maximum between 45 and 5 times the number of steps forward. We set this lower limit at 45, as we observed that this positively improved the performance of all models. The results for predictions at 1, 3 and 6 steps ahead are therefore different from those in the TKAN and TKAT papers.

4.4 Task 1: Predicting Volumes

4.4.1 Model Architecture and Benchmarks

As these tasks are the same as those used in the TKAN and TKAT articles, we decided to use the same benchmarks for comparison. We also compared SigKAN with a variant called SigDense, in which the KANLinear is replaced by a fully connected dense layer, and the GRKAN is replaced by a standard GRN layer.

- 1. For the SigKAN and SigDense:
 - (a) A variant with one SigKAN (resp. SigDense) layer with 100 units, which outputs is flatten and passed to a Dense 100 with activation 'relu' before a last linear layer. They are referred as SK-1 and SD-1 in the following tables.

(b) A variant with two consecutive SigKAN (resp. SigDense) layers with 20 units, which outputs is flatten and passed to a Dense 100 with activation 'relu' before a last linear layer. They are referred as SK-2 and SD-2 in the following tables.

For all of them, we used a signature level of 2, as this is the lowest level possible to capture the interraction between inputs, while also being the least expensive in terms of computation, which quickly becomes limiting due to the non-GPU support of signature operations.

2. For the TKAN, GRU and LSTM:

- (a) An initial recurrent layer of 100 units that returns complete sequences
- (b) intermediate recurrent layer of 100 units, which returns only the last hidden state.
- (c) A final dense layer with linear activation, with as many units as there is timesteps to predict ahead
- 3. For the TKAT use a number of hidden units of 100 in each layer, with the exception of the embedding layer, which has only one unit per feature. The number of heads used is 4.

4.4.2 Results

The results are separated into two tables, the first being the average R^2 over 5 runs obtained for the SigKAN and SigDense variants, followed by their references. The second is the standard deviation of R^2 obtained over these 5 runs, in order to analyze the stability of the model in relation to its initialization over several runs.

Table 1: R^2 Average: SigKAN and SigDense

Time	SK-1	SD-1	SK-2	SD-2
1	0.36225	0.33055	0.30807	0.31907
3	0.21961	0.21532	0.20580	0.21066
6	0.16361	0.15544	0.15351	0.15836
9	0.13997	0.12768	0.12684	0.13338
12	0.12693	0.11628	0.11826	0.11814
15	0.11861	0.11097	0.11448	0.11065

Table 2: \mathbb{R}^2 Average: Benchmarks

Time	TKAT	TKAN	GRU	LSTM
1	0.30519	0.33736	0.36513	0.35553
3	0.21801	0.21227	0.20067	0.06122
6	0.17955	0.13784	0.08250	-0.22583
9	0.16476	0.09803	0.08716	-0.29058
12	0.14908	0.10401	0.01786	-0.47322
15	0.14504	0.09512	0.03342	-0.40443

The average of \mathbb{R}^2 shows that the proposed SigKAN model outperforms all the simple reference models, namely TKAN, GRU and LSTM, while not reaching transformer performance for this task over a longer period. The main advantage is that while TKAT and TKAN offer inferior performance for one-step prediction, the SigKAN model outperforms all the other models.

This indeed seems logical on short terms prediction given the very simple architecture of the SigKAN model compared to the Kolmogorov-Arnold time transformer, but shows that although it contains no recurrence calculus, it manages to outperform all the simple recurrent networks to which it is compared. The results also show that using the signature as a weighting mechanism is

effective, since the SigDense and SigKAN versions outperform most recurrent models in the results. However, using KAN layers instead of dense layers shows an increase in performance, indicating that both components are interesting in the proposed layer. Finally, we can observe that stacking several layers does not

seem to be effective in improving the performance of the KAN version on this task, while improving that of the SigDense version on longer predictions. Using KAN in this case allows us to reduce the depth of our model for this case.

Table 3: \mathbb{R}^2 Standard Deviation: SigKAN and SigDense

Time	SK-1	SD-1	SK-2	SD-2
1	0.02378	0.01220	0.01102	0.01476
3	0.01435	0.00342	0.00319	0.00429
6	0.00376	0.00820	0.00438	0.00659
9	0.00346	0.01074	0.00665	0.00385
12	0.00369	0.00106	0.00830	0.00215
15	0.00171	0.00442	0.00216	0.00103

Table 4: R^2 Standard Deviation: Benchmarks

Time	TKAT	TKAN	GRU	LSTM
1	0.01886	0.00704	0.00833	0.01116
3	0.00906	0.00446	0.00484	0.08020
6	0.00654	0.01249	0.02363	0.06271
9	0.00896	0.02430	0.01483	0.05272
12	0.00477	0.00132	0.08638	0.08574
15	0.01014	0.00701	0.02407	0.09272

What's very interesting about the SigKAN model is that it seems to give more stable results over different runs compared with models based on recurrent networks for longer prediction tasks. So, while increasing performance compared with simple recurrent models, the addition of the signature eliminates the recurrent part, which is probably one of the drivers of the greatest instability.

Table 5: R^2 Number of parameters: SigKAN and SigDense

Time	SK-1	SK-2	SD-1	SD-2
1	957,846	266,311	558,946	125,791
3	958,048	$266,\!513$	$559,\!148$	125,993
6	$958,\!351$	$266,\!816$	$559,\!451$	$126,\!296$
9	$958,\!654$	$267,\!119$	559,754	126,599
12	$1,\!108,\!972$	$297,\!452$	710,072	156,932
15	$1,\!259,\!290$	327,785	860,390	$187,\!265$

Table 6: \mathbb{R}^2 Number of parameters: Benchmarks

Time	TKAN	TKAT	GRU	MLP	LSTM
1	119,396	1,047,865	97,001	95,801	128,501
3	119,598	1,057,667	97,203	96,003	128,703
6	119,901	1,073,870	$97,\!506$	$96,\!306$	129,006
9	120,204	1,091,873	97,809	$96,\!609$	$129,\!309$
12	$120,\!507$	$1,\!129,\!676$	$98,\!112$	$125,\!412$	129,612
15	120,810	$1,\!178,\!279$	$98,\!415$	$154,\!215$	$129,\!915$

We also think it's worth examining the number of model parameters. While much simpler than the TKAT models, the SigKAN model used here displayed a much higher number of parameters. This is due to the flattening of its output, which is transmitted to the last fully connected layer, rather than to the layer itself, which has a limited number of parameters. Nevertheless, the layer itself has its own number of parameters which can increase sharply with the number

of inputs, so it's often wise to have a smaller input size to feed it. For example, with 15 prediction steps, the SigKAN layers contain 513,663 weights, while the remaining 751,615 are on the intermediate dense layers connected to the flattened output. Changing the architecture to avoid having such flat inputs could be a way of radically reducing the size of the model.

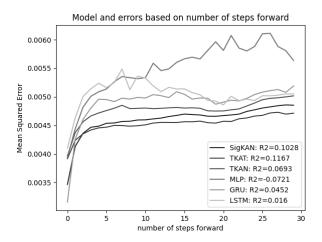


Figure 4: Model and errors based on number of steps forward

Finally, we trained all models once to predict the next 30 steps, and displayed for each step the ratio \mathbb{R}^2 to the actual value. As the results above show, SigKAN lies between TKAN and TKAT in terms of performance. There's also an MLP here, which is the one described in the following tasks, also tested here.

4.5 Predicting Absolute Returns: Benchmarks and Results

4.5.1 Model Architecture and Benchmarks

As the task here is different, with only one feature to predict, we have chosen to keep the same TKAN, GRU and LSTM benchmarks as before, but replace TKAT with a simple MLP model. The SigKAN architecture here refers to SigKAN with 100 hidden units as in the first task, and the MLP refers to a model where sequential inputs are first flattened, then pass through 2 dense layers of 100 units with relu activation, before being passed to a final linear unit with the number of steps forward to predict the units. Volatility being a highly autocorrelated process, we also added a very simple benchmark that uses the average of previous values as a predictor of the next. The window selected for the benchmark is the best one between 1 and 50 selected directly from the test set, which represents the best (unrealistic) result possible with this simple method.

4.5.2 Results

As before the results are obtained over 5 runs for each and displayed in an average and standard deviation table.

Table 7: R^2 Average: SigKAN versus Benchmarks

Time	SK-1	MLP	GRU	LSTM	TKAN	TKAT	Bench
1	0.1670	0.1458	0.1616	0.1581	0.09063	0.125312	0.1476
3	0.1497	0.1335	0.1452	0.1488	0.09063	0.132425	0.1387
6	0.1349	0.1232	0.1360	0.1246	-0.0351	0.132983	0.1293
9	0.1252	0.1172	0.1289	0.1288	0.00708	0.125162	0.1225
12	0.1164	0.1074	0.1163	0.1175	-0.0716	0.107760	0.1169
15	0.1165	0.1085	0.1229	0.1169	-0.0496	0.099868	0.1114

It appears that for this task, SigKAN is able to outperform all other models on one-step-ahead predictions by a wide margin, while giving more sensible but no better predictions on longer-term predictions. The tasks also show that for these single-input tasks, the TKAN and TKAT models have much more difficulty than the other models, which may be a sign of less versatility than the SigKAN given the task. This is to be contrasted with LSTMs, which have almost opposite results to TKAN and TKAT, which while achieving the poorest results on the volume tasks are able to perform much better on the volatility task than the other two.

Table 8: R^2 Standard Deviation: SigKAN versus Benchmarks

Time	SK-1	MLP	GRU	LSTM	TKAN	TKAT	Bench
1	0.0013	0.0050	0.0008	0.0003	0.0517	0.0558	0
3	0.0036	0.0080	0.0073	0.0012	0.0265	0.0087	0
6	0.0005	0.0037	0.0010	0.0020	0.0195	0.0022	0
9	0.0014	0.0026	0.0007	0.0022	0.0792	0.0028	0
12	0.0006	0.0074	0.0023	0.0006	0.0643	0.0060	0
15	0.0013	0.0057	0.0008	0.0001	0.0583	0.0070	0

What is also apparent is the real strength of this model, namely its much greater stability, not only in terms of the tasks it can handle, but also in terms of stability over several calibrations, which is an important property. Finally, we also displayed the total number of parameters for each model used, and we can see that, as before, the number of trainable parameters in SigKAN is much higher than in all the others, due to the flattening of the time dimension leading to a large, dense hidden layer. Here, for 15-step forward forecasting

tasks, the number of weights held by the SigKAN layers is only 114,711, while the remaining 751,615 are due to the following layers. Here again, the reduction

Table 9: R^2 Number of parameters: SigKAN versus Benchmarks

Time	SK-1	TKAN	TKAT	GRU	MLP	LSTM
1	563,646	113,906	485,743	91,601	14,801	121,301
3	563,848	114,108	$495,\!545$	91,803	15,003	$121,\!503$
6	$564,\!151$	114,411	511,748	$92,\!106$	$15,\!306$	121,806
9	$564,\!454$	114,714	529,751	$92,\!409$	15,609	$122,\!109$
12	714,772	$115,\!017$	$567,\!554$	92,712	$17,\!412$	$122,\!412$
15	865,090	$115,\!320$	$616,\!157$	93,015	19,215	122,715

in the number of parameters could be achieved by modifying the way the layer output is used, by not just flattening it and fully connecting all outputs to the next hidden layer.

5 Conclusion

The adoption of path signatures as an improvement on traditional KANs was presented in this paper. It was achieved through an innovative combination with Kolmogorov-Arnold networks, a technique never before realized. This new development significantly improves the ability of these networks to handle tasks involving approximation functions thanks to the use of path signatures containing rich geometric information. The results of our experimental evaluations demonstrate that SigKANs not only outperform other conventional methods for multivariate time series data, but also offer a simple robust framework for modeling complex temporal relationships, which can be applied on a large scale without too much difficulty. The fusion itself holds great promise for future research and applications: it could find its place in a variety of fields, including financial modeling or time series analysis, where such sophisticated tools are most needed. With this effort, we are going beyond mere academic interest in KANs; thanks to what we have developed here, new horizons can be opened up for machine learning, practical applications, and more generally, even other fields outside AI that could benefit from such developments.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Mikolaj Binkowski, Gautier Marti, and Philippe Donnat. Autoregressive convolutional neural networks for asynchronous time series. In *International Conference on Machine Learning*, pages 580–589. PMLR, 2018.

Zavareh Bozorgasl and Hao Chen. Wav-kan: Wavelet kolmogorov-arnold networks. arXiv preprint arXiv:2405.12832, 2024.

- Kuo-Tsai Chen. Integration of paths—a faithful representation of paths by non-commutative formal power series. *Transactions of the American Mathematical Society*, 89(2):395–407, 1958.
- Ilya Chevyrev and Andrey Kormilitzin. A primer on the signature method in machine learning. arXiv preprint arXiv:1603.03788, 2016.
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015.
- George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.
- Joel Dyer, Patrick W Cannon, and Sebastian M Schmon. Deep signature statistics for likelihood-free time-series models. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.
- Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.
- Adeline Fermanian. Embedding and learning with signatures. Computational Statistics & Data Analysis, 157:107148, 2021.
- Remi Genet and Hugo Inzirillo. A temporal kolmogorov-arnold transformer for time series forecasting. arXiv preprint arXiv:2406.02486, 2024a.
- Remi Genet and Hugo Inzirillo. Tkan: Temporal kolmogorov-arnold networks. arXiv preprint arXiv:2405.07344, 2024b.
- Lajos Gergely Gyurkó, Terry Lyons, Mark Kontkowski, and Jonathan Field. Extracting information from the signature of a financial data stream. arXiv preprint arXiv:1307.7244, 2013.
- James D Hamilton. State-space models. *Handbook of econometrics*, 4:3039–3080, 1994.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Chang-Jin Kim. Dynamic linear models with markov-switching. *Journal of econometrics*, 60(1-2):1–22, 1994.

- Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables. American Mathematical Society, 1961.
- 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 Workshops*, pages 631–639, 2017.
- Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.
- Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. arXiv preprint arXiv:2404.19756, 2024.
- Helmut Lütkepohl. Vector autoregressive models. In *Handbook of research methods and applications in empirical macroeconomics*, pages 139–164. Edward Elgar Publishing, 2013.
- Terry Lyons. Rough paths, signatures and the modelling of functions on streams. arXiv preprint arXiv:1405.4537, 2014.
- Terry J Lyons. Differential equations driven by rough signals. Revista Matemática Iberoamericana, 14(2):215–310, 1998.
- King Ma and Henry Leung. A novel lstm approach for asynchronous multivariate time series prediction. In 2019 International Joint Conference on Neural Networks (IJCNN), pages 1–7. IEEE, 2019.
- Massimiliano Marcellino, James H Stock, and Mark W Watson. A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series. *Journal of econometrics*, 135(1-2):499–526, 2006.
- Imanol Perez Arribas, Guy M Goodwin, John R Geddes, Terry Lyons, and Kate EA Saunders. A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder. *Translational psychiatry*, 8(1):274, 2018.
- F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. ISSN 0033-295X.
- Zaiyong Tang, Chrys De Almeida, and Paul A Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *Simulation*, 57(5):303–310, 1991.
- Cristian J Vaca-Rubio, Luis Blanco, Roberto Pereira, and Màrius Caus. Kolmogorov-arnold networks (kans) for time series analysis. arXiv preprint arXiv:2405.08790, 2024.

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.
- William WS Wei. Multivariate time series analysis and applications. John Wiley & Sons, 2018.
- Weixin Yang, Terry Lyons, Hao Ni, Cordelia Schmid, and Lianwen Jin. Developing the path signature methodology and its application to landmark-based human action recognition. In *Stochastic Analysis*, *Filtering*, and *Stochastic Optimization*: A Commemorative Volume to Honor Mark HA Davis's Contributions, pages 431–464. Springer, 2022.
- Eric Zivot and Jiahui Wang. Vector autoregressive models for multivariate time series. *Modeling financial time series with S-PLUS®*, pages 385–429, 2006.