Functional Autoencoder for Smoothing and Representation Learning

Sidi Wu[1], Cédric Beaulac[2], Jiguo Cao[1*]

[1]Department of Statistics and Actuarial Science, Simon Fraser University, Burnaby, BC, Canada.
[2]Département de Mathématiques, Université du Québec à Montréal, Montréal, Québec, Canada.

*Corresponding author(s). E-mail(s): jiguo_cao@sfu.ca;
Contributing authors: sidi_wu@sfu.ca; beaulac.cedric@uqam.ca;

Abstract

A common pipeline in functional data analysis is to first convert the discretely observed data to smooth functions, and then represent the functions by a finite-dimensional vector of coefficients summarizing the information. Existing methods for data smoothing and dimensional reduction mainly focus on learning the linear mappings from the data space to the representation space, however, learning only the linear representations may not be sufficient. In this study, we propose to learn the nonlinear representations of functional data using neural network autoencoders designed to process data in the form it is usually collected without the need of preprocessing. We design the encoder to employ a projection layer computing the inner product of the functional data and functional weights over the observed timestamp, and the decoder to apply a recovery layer that maps the finite-dimensional vector extracted from the functional data back to functional space using a set of predetermined basis functions. The developed architecture can accommodate both regularly and irregularly spaced data. Our experiments demonstrate that the proposed method outperforms functional principal component analysis in terms of prediction and classification, and maintains superior smoothing ability and better computational efficiency in comparison to the conventional autoencoders under both linear and nonlinear settings.

Keywords: Functional data analysis, Neural networks, Nonlinear learning, Functional principal component analysis

1 Introduction

Functional data analysis (FDA) has found extensive application and received growing attention across diverse scientific domains. Functional data, as the core of FDA, are theoretically defined as any random variables that assume values in an infinite-dimensional space, such as time or spatial space (Ferraty and Vieu, 2006, Ramsay and Silverman, 2005), and are usually discretely observed at some regularly or irregularly spaced points over the time span in applications. Due to the complexity and difficulty in interpreting and analyzing infinite-dimensional variables, a common pipeline for FDA is to represent the infinite-dimensional functional data, denoted as $X(t)$, by a finite-dimensional vector of coefficients that extract and summarize the useful information carried by the individual functions (Yao et al., 2021). These coefficients can be of interests themselves or be readily utilized in further analysis (Wang et al., 2016a).

Two predominate approaches for dimension reduction in FDA are basis expansion and functional principal component analysis (FPCA). The first approach basis expansion represents the functional data as $X_i(t) = \sum_{m=1}^{M_B} c_{im}\phi_m(t)$, where $\phi_m(t)$ are known basis functions and c_{im} are corresponding basis coefficients for the i-th subject containing the information from the original functions (Ramsay and Silverman, 2005). This method requires the predetermination of a basis system, for instance, Fourier or B-spline, and the number of basis functions M_B, in order to learn the representation of functional data. The second approach FPCA (Nie and Cao, 2020, Sang et al., 2017) is a fully data-driven approach that compresses the functional data $X_i(t)$ into functional principal component (FPC) scores $\xi_{im} = \int\{X_i(t) - \mu(t)\}\psi_m(t)dt$, where $\mu(t)$ is the mean function of the variable $X(t)$, and $\psi_m(t)$'s are the FPCs which are also the eigenfunctions derived from the spectral decomposition of the variance-covariance function of $X(t)$. By Karhunen-Loève expansion, FPCA can construct the functional data as $X_i(t) = \mu(t) + \sum_{m=1}^{M_P} \xi_{im}\psi_m(t)$, with a predetermined proportion of explained variation, which indirectly defines M_P, the number of FPCs identified. The theoretical details and results on asymptotic distributions have been well derived and fully discussed by Dauxois et al. (1982), Hall and Hosseini-Nasab (2006) and Hall and Hosseini-Nasab (2006).

Representations such as FPC scores have been widely used for establishing functional regression models (Müller and Yao, 2008, Yao et al., 2005b, 2010), clustering (Chiou and Li, 2007, Peng and Müller, 2008) and classification (Müller, 2005, Müller and Stadtmüller, 2005) of functional curves. Both aforementioned methods are fundamentally linear mappings from infinite-dimensional data to the vector of finite scalars, however, learning linear projections of functional data might not be sufficient and informative. Furthermore, FPCA relies on the assumption of a common variance-covariance of all curves, which might be violated when the individual trajectories are labelled with classes.

Numerous extensions to the conventional FPCA have been suggested to adapt the linear representation of functional data for diverse scenarios (Chen and Lei, 2015, Nie et al., 2018, 2022, Peng and Paul, 2009, Sang et al., 2017, Shi et al., 2021, Yao et al., 2005a, Zhong et al., 2022). Nevertheless, limited contributions on nonlinear representation learning of functional data can be found in latest literature. Song and

Li (2021) extended the standard FPCA to a nonlinear additive functional principal component analysis (NAFPCA) for vector-valued functional data to accommodate nonlinear functions of functional data via two additively nested Hilbert spaces. However, similar to the linear FPCA with discrete functional data, this technique requires to first estimate the underlying $X(t)$ using the basis expansion or the reproducing Kernel Hilbert space method in the first-level function space. Chen and Müller (2012) developed nonlinear manifold learning to generate nonlinear representations of functional data by modifying the existing nonlinear dimension reduction methods to satisfy functional data settings. The manifold-based representation is basically designed to be layered on the representation produced by FPCA, while the computational challenges may arise as the sample size increases.

Meanwhile, the advent use of big data and the gradual popularity of deep learning promote the introduction of neural networks to functional data representation learning. Wang and Cao (2024) explored a functional nonlinear learning method, namely FunNoL, which relies on recurrent neural networks (RNNs) to represent multivariate functional data in a lower-dimensional feature space and handle the missing observations and excessive local disturbances in the observed functional data. This method ignores the basic structure of functional data as it regards $X(t)$ as time series data and captures the temporal dependency across time sequences. Moreover, to enable the use of representation for classifying curves,FunNoL is designed to be a semi-supervised model that combines a classification model with a standard RNN, introducing more complexity to network optimization and representation learning. Hsieh et al. (2021) defined a functional autoencoder that generalizes the conventional neural network autoencoders to handle continuous functional data, and developed the functional gradient-based learning algorithm for optimizing the autoencoder to study the nonlinear projection of multidimensional functional data. This approach requires smooth functional inputs and overlooks the common issue where functional data are barely fully observed in practice (Yao et al., 2005a).

The main objective of this study is to propose a solution to the nonlinear representation learning and smoothing of discrete functional data using a novel functional autoencoder (FAE) based on a densely feed-forward neural network, which includes FPCA as a special case under the linear representation setting. As an unsupervised learning technique, autoencoders (AEs) have been frequently used for feature extraction and representation learning in vector-space problems (Bengio et al., 2013, Hinton and Salakhutdinov, 2006, Meiler et al., 2001, Wang et al., 2016b). A traditional AE consists of an encoder and a decoder connected by a bottleneck layer, where the former one is a mapping from a P-dimensional vector-valued input space to a d-dimensional representation space and the latter one maps from the d-dimensional representation space back to a vector-valued output space of P dimensions, where the output layer consists of a reconstruction of the original input. Assuming $d \ll P$, the neurons in the bottleneck layer serve as a lower-dimension representation of the input. This representation is a collection of neuron features extracted from the AE, which can be of interests themselves or used for further research. The relation between AE and principal component analysis (PCA) has been well discussed in several existing studies. Oja (1982, 1992) demonstrated that a neural network employing a linear activation

function essentially learns the principal component representation of the input data. Furthermore, Baldi and Hornik (1989) and Bengio et al. (2013) showed that an autoencoder with one hidden layer and the identity activation is essentially equivalent to PCA. Bourlard and Kamp (1988) and Baldi and Hornik (1989) also explained that the representation captured by such autoencoders is a basis of the subspace spanned by the leading principal components (PCs) instead of necessarily coincident with them. The connection between conventional AE and PCA can be naturally transplanted to that of the designed FAE and FPCA with a relevant discussion provided.

Specifically, in this work, we propose to construct an autoencoder under discrete functional data settings. We design the encoder to incorporate a projection layer computing the inner product of the functional data and functional weights over the observed discrete time spans, and the decoder to equip a recovery layer to map the finite-dimensional vector extracted from the functional data to functional space using a set of preselected basis functions. The developed architecture compresses the discretely observed functional data to a set of representations and then outputs smooth functions. The resulting lower-dimensional vector will be the representation/encoding of the functional data, which serves a similar purpose to the basis coefficients or FPC scores previously mentioned and can be inputted into any further analysis.

The autoencoder we design for functional data have at least the following highlights. First, the proposed FAE addresses the learning of a nonlinear representation from discrete functional data with a flexible nonlinear mapping path captured by neural networks, eliminating the conduct of curve smoothing assuming any particular form in advance. In other words, our method performs a one-step model simultaneously learning the representative features and smoothing the discretely observed trajectories. Second, it allows us to obtain linear and nonlinear projections of functional data, with the former path serving as an alternative approach to FPCA. Third, the proposed method is applicable for both regularly and irregularly spaced data, while the smoothness of the recovered curves is controlled through a roughness penalty added to the objective function in model training. Forth, the architecture of the FAE is flexibly programmable and compatible with existing neural networks libraries/modules. Last but not the least, the robustness and efficiency of our method in representation extraction and curve recovery with small size of data and substantial missing information are supported by the results of various numerical experiments.

The remainder of this article proceeds in the following manner. In Section 2, we provide the methodological details for the proposed FAE, including a description about the network architecture and an explanation on the corresponding training procedure. A brief discussion on the connections between the proposed method and two well-established methods, FPCA and AE, is given in Section 3. In Section 4, we compare the proposed functional autoencoders to the existing methods applicable for functional data representation with a focus on relationship capture and computational efficiency through extensive simulation studies across various scenarios. The designed autoencoders and the other techniques in comparison are further evaluated in Section 5 with a real data application. Finally, we conclude with a discussion and future directions in Section 6. The preprocessed data sets and computing codes of the proposed method on selected applications are available at https://github.com/CedricBeaulac/FAE.

2 Functional autoencoders

2.1 Motivation: autoencoders for continuous functional data

Suppose there are N subjects and for the i-th subject, a functional variable $X_i(t), t \in \mathcal{T}$ is observed in the $L^2(t)$ space, where $\mathcal{T}$ refers to the domain of the functional variable X. To address the limitations of linear representations of functional data $X(t)$, we propose to learn nonlinear mappings from functional data space $L^2(t)$ to K-dimensional vector space $\mathbb{R}^K$ through a neural network autoencoder which contains an encoder compressing the functional input to some scalar-valued neurons, and a decoder reconstructing the functional input back from the encoded representations.

We introduce an autoencoder with L hidden layers (excluding the input and output layers) for continuous functional data $X(t)$, which we suppose, are fully observed over a continuum t and $t \in \mathcal{T}$. Different from conventional autoencoders consuming scalar inputs, in this scenario, functions are served as inputs and fed into the neural network, and the designed autoencoder for continuous functional data is supposed to be trained by minimizing the reconstruction error $L(X(t), \hat{X}(t)) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \int_{\mathcal{T}} (X_i(t) - \hat{X}_i(t))^2 dt$, where N_{train} denotes the number of observations in the training set.

We propose to encode the infinite-dimensional functions to some finite number of numerical neurons by introducing functional weights $w^I(t)$ to bridge the input and the first hidden layer of the encoder. Specifically, the scalar inner product, which connects neurons in the input and the first hidden layer of the conventional AE, is generalized by the inner product of the functional input $X(t)$ and the functional weight $w^I(t)$ in L^2 space. Consequently, the k-th neuron in the first hidden layer $h_k^{(1)}$ is computed as

$$h_k^{(1)} = g\left(\int_{\mathcal{T}} X(t) w_k^I(t) dt\right), \tag{1}$$

where $w_k^I(t)$ is the input functional weight connecting the functional input and the k-th neuron in the first hidden layer, and $g(\cdot)$ is the activation function. To be noted that here we opt to neglect the numerical bias term $b^{(1)}$ for simplicity.

The proposed functional weights together with the inner product of two functions achieve the mapping from L^2 to $\mathbb{R}^{K^{(1)}}$, where $K^{(l)}$ is the number of neurons in the l-th hidden layer and $l \in \{1, 2, ..., L\}$. The resulting numerical neurons are further passed to the continuous hidden layers of the autoencoder, following the same calculation rules as in conventional AEs. Accordingly, the k-th neuron in the l-th hidden layers is given by

$$h_k^{(l)} = g\left(\sum_{j=1}^{K^{(l-1)}} h_j^{(l-1)} w_{jk}^{(l)}\right), \tag{2}$$

with $h_j^{(l-1)}$ being the j-th neuron in the $(l-1)$-th hidden layer connected by the scalar network weight $w_{jk}^{(l)}$.

Similarly, a set of functional weights $\{w_k^O(t)\}_{k=1}^{K^{(L)}}$, instead of scalar weights, are applied at the output layer of the decoder to map the second to last layer from $\mathbb{R}^{K^{(L)}}$

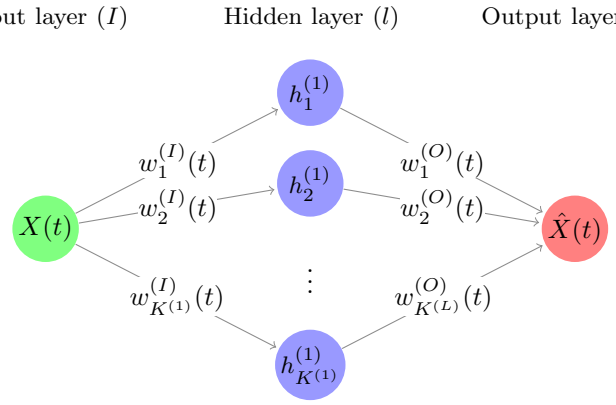


Fig. 1: Functional autoencoder for continuous data with $L = 1$ hidden layer.

back to functional space L^2 and mathematically, the outputted functional neuron is calculated as

$$\hat{X}(t) = \sum_{k=1}^{K^{(L)}} h_k^{(L)} w_k^{(O)}(t), \tag{3}$$

where $w_k^O(t)$ is the output functional weight connecting the k-th neuron in the L-th hidden layer and the output functional neuron. For this output layer to produce the functional output required, the linear activation function must be used.

We name this autoencoder the continuous functional autoencoder (CFAE) and a graphical visualization of the CFAE with $L = 1$ hidden layer(s) for functional data can be seen in Figure 1. In this scenario, $\{h_1^{(1)}, h_2^{(1)}, ..., h_{K^{(1)}}^{(1)}\}$ is regarded as the vector-valued representation of $X(t)$.

2.2 Proposed model: autoencoders for discrete functional data

The CFAE introduced in Section 2.1 serves as an inspiration for the model we proposed to better suit discrete functional data, which reflects how functional data are collected and stored in practical applications. Considering the functional data are discretely observed at J evenly spaced time points $t_1, ..., t_J$ over the time interval $\mathcal{T}$ for all N subjects, and therefore for the i-th subject, we obtain J pairs of observations $\{t_j, X_i(t_j)\}$, $j = 1, 2, ...J$. As a matter of fact, the real functional data are often contaminated with some observational errors, resulting in a collection of noisy discrete observations $\tilde{X}_i(t_j) = X_i(t_j) + \epsilon_i(t_j)$, where $\epsilon_i(t_j)$ is the i.i.d. measurement error. Without knowing the true underlying curves $X(t)$'s, the contaminated observations $\{\tilde{X}(t_j)\}_{j=1}^J$'s are employed as an alternative to $\{X(t_j)\}_{j=1}^J$'s in applications.

We propose to adapt the CFAE to take data $\{X(t_1), X(t_2), ..., X(t_J)\}$, a J-dimensional vector, as the input. Instead of smoothing the discrete data and then applying the previously defined autoencoder, we develop the architecture to satisfy

such discrete functional input. This is a major advantage of our proposed method as the data no longer needs to be preprocessed in any way before being fed to our proposed FAE. To do so, we replace the weight functions $w_k^I(t)$ for the input layer and $w_k^O(t)$ for the output layer with their discrete versions $\{w_k^I(t_j)\}_{j=1}^J$ and $\{w_k^O(t_j)\}_{j=1}^J$ evaluated at the corresponding J time points $t_1, ..., t_J$, respectively. Naturally, the integral $\int_{\mathcal{T}} X(t)w_k^{(I)}(t)dt$ in Eq.(1) is approximated numerically using the rectangular or trapezoidal rule, and accordingly the k-th neuron in the first hidden layer is updated as

$$h_k^{(1)} = g\left(\sum_{j=1}^{J} \omega_j X(t_j) w_k^{(I)}(t_j)\right), \tag{4}$$

where $\{\omega_j\}_{j=1}^J$ are the weights used in the numerical integration algorithm. Likewise, the output layer now consists of J neurons corresponding to the J-dimensional input vector as

$$\hat{X}(t_j) = \sum_{k=1}^{K^{(L)}} h_k^{(L)} w_k^{(O)}(t_j). \tag{5}$$

As illustrated in Figure 2, the mappings $L^2 \to \mathbb{R}^{K^{(1)}}$ and $\mathbb{R}^{K^{(L)}} \to L^2$ in the CFAE are substituted with $\mathbb{R}^J \to \mathbb{R}^{K^{(1)}}$ and $\mathbb{R}^{K^{(L)}} \to \mathbb{R}^J$, respectively, for this discrete setting.

The autoencoder we propose for discrete functional data seemingly behaves the same as a conventional AE, however, our FAE requires a different training process which accounts for the assumption that functional data are the realization of a underlying smooth stochastic process. This way, the proposed FAE considers the serial correlation of the functional data and returns smooth and continuous functions without the need for preemptive smoothing of the input.

We further propose to represent the functional weights used in the input and output layers as $w_k^{(\cdot)}(t) = \sum_{m=1}^{M_k^{(\cdot)}} c_{mk}^{(\cdot)} \phi_{mk}^{(\cdot)}(t)$, where $\{\phi_{mk}^{(\cdot)}(t)\}_{m=1}^{M_k^{(\cdot)}}$'s are some known basis functions from a preselected basis system for the k-th functional weight, $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$ are the corresponding basis coefficients remain to be determined, and $M_k^{(\cdot)}$ is some predefined truncation integer for the k-th weight. The snapshots of the k-th input or output weight function are accordingly marked as

$$w_k^{(\cdot)}(t_j) = \sum_{m=1}^{M_k^{(\cdot)}} c_{mk}^{(\cdot)} \phi_{mk}^{(\cdot)}(t_j), \tag{6}$$

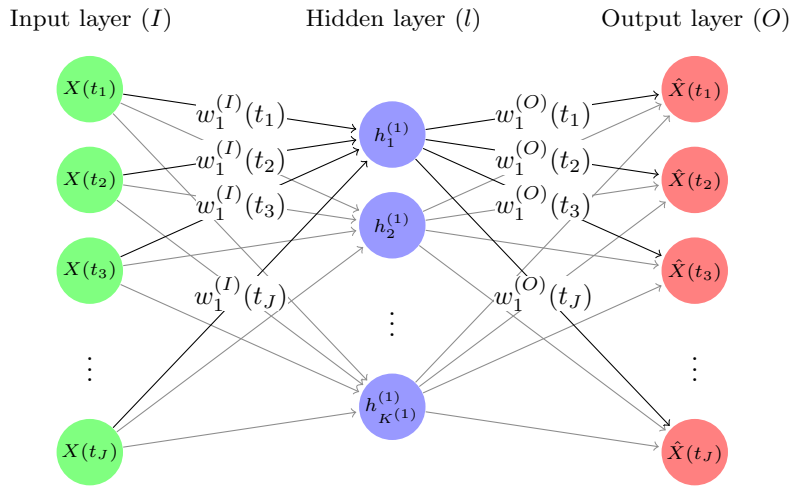


Fig. 2: Functional autoencoder for discrete data with $L = 1$ hidden layer.

and therefore Eq.(4) and Eq.(5), which calculate the k-th neurons in the first hidden layer and the output layer, respectively, can be re-written as

$$\begin{aligned} h_k^{(1)} &= g\left(\sum_{j=1}^{J}\omega_j X(t_j)\sum_{m=1}^{M_k^{(I)}} c_{mk}^{(I)}\phi_{mk}^{(I)}(t_j)\right) \\ &= g\left(\sum_{m=1}^{M_k^{(I)}} c_{mk}^{(I)}\sum_{j=1}^{J}\omega_j X(t_j)\phi_{mk}^{(I)}(t_j)\right), \end{aligned} \tag{7}$$

and

$$\begin{aligned} \hat{X}(t_j) &= \sum_{k=1}^{K^{(L)}} h_k^{(L)}\sum_{m=1}^{M_k^{(O)}} c_{mk}^{(O)}\phi_{mk}^{(O)}(t_j) \\ &= \sum_{k=1}^{K^{(L)}}\sum_{m=1}^{M_k^{(O)}} h_k^{(L)} c_{mk}^{(O)}\phi_{mk}^{(O)}(t_j). \end{aligned} \tag{8}$$

In such a manner, the task of learning the functional weights $w_k^{(\cdot)}(t)$'s using typical machine learning techniques becomes one of learning $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$, the parameters defining $w_k^{(\cdot)}(t_j)$, for $k = 1, ..., K^{(l)}$, $l = 1$ or L. Consequently, we seek to learn the coefficients $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$ through back-propagation.

2.2.1 Encoder with a feature layer

For computational convenience, we let $M_k^{(I)} = M^{(I)}$ and $\phi_{mk}^{(I)}(t) = \phi_m^{(I)}(t)$ for all $k \in \{1, 2, ..., K^{(1)}\}$, indicating that the input weight functions $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$ are expressed with the same basis expansion. In consequence, we can simplify Eq. (7) as

$$\begin{aligned} h_k^{(1)} &= g\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} \sum_{j=1}^{J} \omega_j X(t_j)\phi_m^{(I)}(t_j)\right) \\ &= g\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} f_m\right), \end{aligned} \tag{9}$$

where $f_m = \sum_{j=1}^{J} \omega_j X(t_j)\phi_m^{(I)}(t_j), m = \{1, 2, ..., M^{(I)}\}$, a Riemann sum approximating the inner product of $X(t)$ and $\phi_m^{(I)}(t)$. $\{f_m\}_{m=1}^{M^{(I)}}$ represent the resulting *features* of $X(t)$ projected to the basis function sets and serve as the pivot connecting the input layer and the first hidden layer. Hence, we design our proposed encoder by inserting a *feature layer* of f_m's between the input layer and the first hidden layer, as shown in Figure 3. This deterministic layer translates discretely observed functional data into a scalar structure that can then be processed with existing neural network models and training algorithms.

Specifically, the input layer and the *feature layer* are linked by the snapshots of the basis function set $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}$, while $c_{mk}^{(I)}$ becomes the network weights connecting the *feature layer* and the first hidden layer.

A benefit of this layer architecture is that different observations which might be observed at different time points will all get converted to the same features. Consequently, irregularly observed data are managed through this deterministic layer. It is also possible to recover the continuous functional weights $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$ for visualization purpose after learning the coefficients $\{c_{mk}^{(I)}\}_{m=1}^{M^{(I)}}$.

2.2.2 Decoder with a coefficient layer

Again, for simplicity, we set the basis functions to be the same for representing all output weight functions $\{w_k^{(O)}(t)\}_{k=1}^{K^{(L)}}$ by setting $M_k^{(O)} = M^{(O)}$ and $\phi_{mk}^{(O)}(t) = \phi_m^{(O)}(t)$ for all $k \in \{1, 2, ..., K^{(L)}\}$. Following on Eq.(8), we now have

$$\begin{aligned} \hat{X}(t_j) &= \sum_{k=1}^{K^{(L)}} \sum_{m=1}^{M^{(O)}} h_k^{(L)} c_{mk}^{(O)} \phi_m^{(O)}(t_j) \\ &= \sum_{m=1}^{M^{(O)}} \left(\sum_{k=1}^{K^{(L)}} h_k^{(L)} c_{mk}^{(O)}\right) \phi_m^{(O)}(t_j) \\ &= \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t_j), \end{aligned} \tag{10}$$

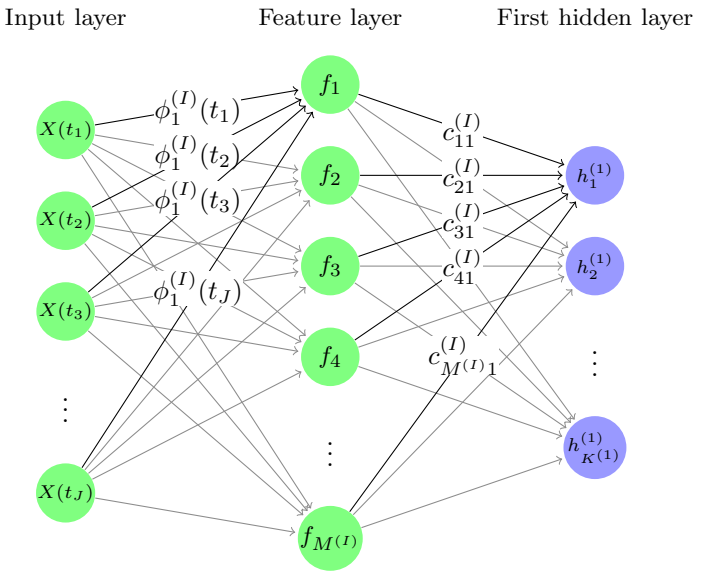


Fig. 3: Encoder with a *feature layer*. Notice that the input and feature layers are devoid of parameters at this point and are entirely deterministic given the data and the choice of basis functions for $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$.

where $b_m = \sum_{k=1}^{K^{(L)}} h_k^{(L)} c_{mk}^{(O)}$. In fact, $\{\phi_m^{(O)}(t)\}_{m=1}^{M^{(O)}}$ can be regarded as the basis functions used in the representation of $\hat{X}(t)$, the reconstructed functional observation. In turn, $\{b_m\}_{m=1}^{M^{(O)}}$ play the role of the corresponding basis coefficients.

Hence, in Figure 4, we visualize b_m's as the neurons of a *coefficient layer* added to the decoder for connecting the last hidden layer and the output layer, while $c_{mk}^{(O)}$ are the network weights between the last hidden layer and the *coefficient layer*. Meanwhile, the *coefficient layer* and the output layer are connected deterministically through snapshots of the basis functions $\{\phi_m^{(O)}(t)\}_{m=1}^{M^{(O)}}$. The proposed decoder is essentially and functionally consistent with NNBR, a neural network designed for scalar input and functional output, developed by Wu et al. (2023), since both approaches decompress the scalar-valued basis coefficients to the functional curves in a linear manner, ensuring the use of back-propagation in model training.

Likewise, an advantage of this layer architecture is its ability to easily handle irregularly spaced data, as explained in Wu et al. (2023). It also provides a smooth reconstruction of the input functional data, which can be evaluated at any point within the domain, effectively smoothing the functional data while simultaneously learning meaningful and practical representations.

The two aforementioned deterministic operations that distinguish our approach will require adjustments to the network training process but will not significantly impact the sensitivity to the width and depth of the architecture or other hyperparameters. Theoretically, increasing the depth and width of the FAE introduces more parameters

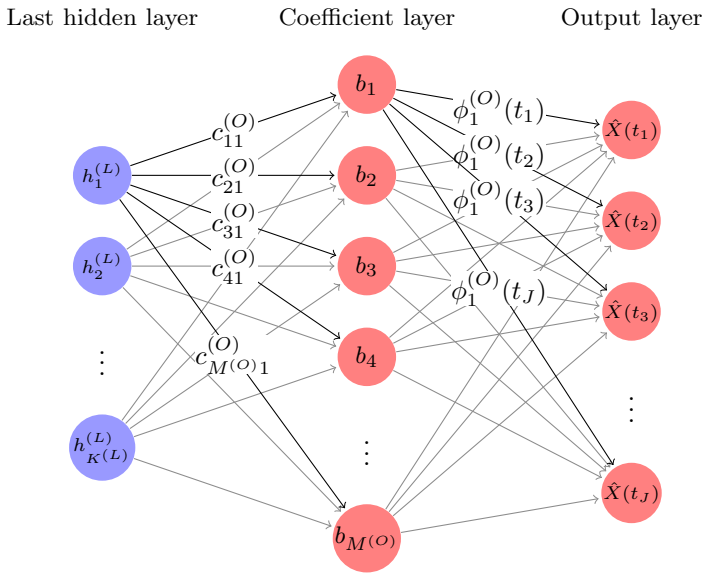


Fig. 4: Decoder with a *coefficient layer*. Similarly, the last two layers are devoid of parameters and are deterministic.

and greater capacity to fit complex functions, leading to more informative and diverse representations for further analysis. However, deeper and wider architectures demand more computational resources and face additional training challenges, such as overfitting. In practice, we have observed that increasing the width of the FAE, as compared to its depth, more effectively enhances its capacity and representation power. As illustrated in Section 4 and Section 5, FAEs with three hidden layers already deliver robust and satisfactory performance in representation learning and data reconstruction with the data sets used. In addition, the prediction and classification results are more significantly influenced by the width of the hidden layers, especially the one used for representation.

2.2.3 Training the proposed FAE

A full architecture (with $L = 1$) of the proposed FAE is displayed in Figure 5. As detailed in Section 2.2.1 and Section 2.2.2, a deterministic *feature layer* of size $M^{(I)}$ is created to follow the input layer without using any unknown parameters or weights for neuron calculation, and each neuron in the *feature layer* produces a scalar value computed as the numerical approximation of the inner product of the input $X(t_j)$ and the preselected basis function $\phi_m^{(I)}(t_j)$ over the observed timestamp. On the other end, a *coefficient layer* of $M^{(O)}$ scalar-valued neurons is handcrafted as the second to last layer, and it connects the output layer through the known basis functions $\phi_m^{(O)}(t)$'s, making the output layer also deterministic. Layers between the *feature layer* and the *coefficient layer* share the same structure as a conventional AE. This specific structure is the essence of the proposed FAE.

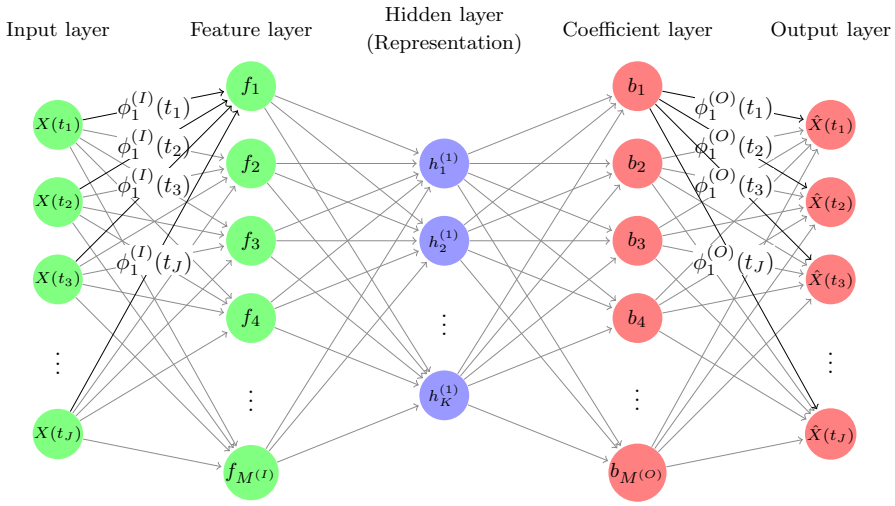


Fig. 5: A graphical representation of the FAE we propose for discrete functional data. The model represented only has a single hidden layer h, that serves the role of latent representation.

Same as traditional AEs, the training process of FAEs comprises of two components, the *forward propagation* and the *backward propagation*, and can be operated using existing neural network libraries or modules, such as `pytorch` (Paszke et al., 2019) and `tensorflow` (Abadi et al., 2015). The *forward propagation* has been previously depicted and is summarized in Algorithm 1. Here we put emphasis on the *backward propagation* that updates the network parameters using gradient-based optimizers.

Let $\theta = \{c_{mk}^{(I)}, c_{mk}^{(O)}, \eta\}$ denote the collection of network parameters, where η stands for all the network weights involved in connecting the hidden layers. The training process targets at finding $\hat{\theta} = \text{argmin}_\theta \, L(X(t_j), \hat{X}(t_j))$, and we employ the standard mean squared error (MSE) between $X(t_j)$ and $\hat{X}(t_j)$ across all the observed time points $t_1, ..., t_J$ and all subjects in the training set as the reconstruction error of the FAE, in specific, $L(X(t_j), \hat{X}(t_j)) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \sum_{j=1}^{J} (X_i(t_j) - \hat{X}_i(t_j))^2$. We design the output layer of FAE to be a linear combination of some preselected basis functions $\{\phi_m^{(O)}\}_{m=1}^{M^{(O)}}$ and the neurons $\{b_m\}_{m=1}^{M^{(O)}}$ outputted by the second to last layer (the *coefficient layer*), and therefore the neuron $\hat{X}(t_j)$ in the output layer of FAE, which is the snapshot of the reconstructed curve $\hat{X}(t)$ at time t_j is the vector product of $\{b_m\}_{m=1}^{M^{(O)}}$ and $\{\phi_m^{(O)}\}_{m=1}^{M^{(O)}}$ evaluated at the specific t_j. The linear relation between the *coefficient layer* and the output layer, together with the differentials of the known basis functions, ensures the feasibility of computing the gradient of $(X_i(t_j) - \hat{X}_i(t_j))^2$ with respect to the coefficients b_m as

$$\frac{\partial L}{\partial b_m} = \frac{\partial L}{\partial \hat{X}(t_j)} \frac{\partial \hat{X}(t_j)}{\partial b_m}. \tag{11}$$

Algorithm 1: FAE Forward Pass

Input: $\boldsymbol{X} = \{X(t_1), X(t_2), ..., X(t_J)\}$

Output: $\hat{\boldsymbol{X}} = \{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

Hyper-parameters: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}$, $\{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, ω_j for all j, a predefined network NN(θ) with L hidden layers, $K^{(l)}$ neurons in the l-th hidden layer, activation functions $g_1, ..., g_L$, E epochs, Optimizer (including learning rate ϱ), etc.

1 **Input Layer** $\rightarrow$ ***Feature Layer***

$\{X(t_j)\}_{j=1}^{J} \rightarrow f_m = \sum_{j=1}^{J} \omega_j X(t_j)\phi_m^{(I)}(t_j), m \in \{1, 2, ..., M^{(I)}\}$

2 ***Feature Layer*** $\rightarrow$ ***Coefficient Layer***

$\{f_m\}_{m=1}^{M^{(I)}} \rightarrow b_m = \sum_{k=1}^{K^L} c_{mk}^{(O)} g_L\left(\cdots g_1\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} f_m\right)\right), m \in \{1, 2, ..., M^{(O)}\}$

Specifically, the k-th neuron in the l-th hidden layer is constructed the same way as that in conventional neural networks as $h_k^{(l)} = g_l(\sum_{j=1}^{K^{(l-1)}} h_j^{(l-1)} w_{jk}^{(l)})$, and $w_{jk}^{(l)}$'s are the scalar network weights

3 ***Coefficient Layer*** $\rightarrow$ **Output Layer**

$\{b_m\}_{m=1}^{M^{(O)}} \rightarrow \hat{X}(t_j) = \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t_j), j \in \{1, ..., J\}$

return $\{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

The gradient with respect to the network weights η in the remaining layers prior to the *coefficient layer* can be subsequently computed in the backward manner as that in a classic neural network until reaching the *feature layer*, while no any further gradient calculation is made from the *feature layer* back to the input layer because they are connected by the predefined input basis functions $\{\phi_m^{(I)}\}_{m=1}^{M^{(I)}}$, instead of some unknown network parameters in need of estimation. Algorithm 2 details the gradient calculation procedure used to update network parameters in the *backward propagation*.

2.3 FAE as a functional data smoother

By design, our proposed FAE outputs a smooth and continuous curve over the entire interval of interest as an estimate of the underlying stochastic process for any input of discretely observed functional data, given by

$$\hat{X}(t) = \sum_{k=1}^{K^{(L)}} \sum_{m=1}^{M^{(O)}} h_k^{(L)} c_{mk}^{(O)} \phi_m^{(O)}(t) = \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t), \tag{12}$$

which is achievable thanks to the continuity of the preselected basis functions $\phi_m^{(O)}(t)$'s. This is a core design choice made so that the FAE we propose acts not only as a representation learner but a smoother itself and could substitute other smoothing processes such as fitting a B-spline model.

Algorithm 2: FAE Backward Pass

Input: $\theta_{\text{current}}, \{X(t_1), X(t_2), ..., X(t_J)\}, \{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

Output: θ_{updated}

Hyper-parameters: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}, \{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, ω_j for all j, a predefined network NN(θ) with L hidden layers, $K^{(l)}$ neurons in the l-th hidden layer, activation functions $g_1, ..., g_L$, E epochs, Optimizer (including learning rate ϱ), etc.

Compute loss function $L(X(t_j), \hat{X}(t_j))$

Set $\theta = \theta_{\text{current}}$

Output Layer $\rightarrow$ ***Coefficient Layer***

$\frac{\partial L(\theta)}{\partial b_m} = \frac{\partial L(\theta)}{\partial \hat{X}(t_j)} \frac{\partial \hat{X}(t_j)}{\partial b_m}$, because $\hat{X}(t_j) = f(b_m)$ and $f'(b_m)$ exists

Coefficient Layer $\rightarrow$ ***Feature Layer***

$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial L(\theta)}{\partial b_m} \frac{\partial b_m}{\partial \theta}$, same gradient calculation as used in traditional neural networks

Feature Layer $\rightarrow$ **Input Layer**

No gradient calculation involved (deterministic operation)

Update network parameters θ^*

return $\theta_{\text{updated}} = \theta^*$

Following the tradition in FDA, we can promote the smoothness of the output curves by adding a roughness penalty to the objective function of the FAE. With a consideration for computational simplicity, among different roughness penalty choices, we propose to apply the difference penalty (Eilers and Marx, 1996) on the elements of the *coefficient layer* as they act as the basis coefficients of the output functional curves. Consequently, including such a penalty term leads to the following objective function:

$$L_{\boldsymbol{pen}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(\sum_{j=1}^{J} \left(X_i(t_j) - \hat{X}_i(t_j) \right)^2 + \lambda \sum_{m=3}^{M^{(O)}} (\Delta^2 b_{im})^2 \right), \tag{13}$$

and $\Delta^2 b_{im} = b_{im} - 2b_{i(m-1)} + b_{i(m-2)}$, where b_{im} is the m-th neuron in the *coefficient layer* for the i-th training subject, and parameter λ controls the smoothness. In implementation, we suggest applying a roughness penalty when $M^{(O)}$ is relatively large ($M^{(O)} \gg J$) and the optimal λ can be selected using cross-validation.

2.4 FAE for irregularly spaced observations

For many existing FDA models, it is quite common to assume that the observed discrete functional data are regularly spaced. A benefit of our designed FAE is that it is free of this assumption and its input layer can actually be of flexible size because of the proposed *feature layer* applied in the early stage of the model.

As detailed in Section 2.2.1, we express the input functional weights $w_k^{(I)}(t_j)$ by a fixed representation of $\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)}\phi_m^{(I)}(t_j)$, and therefore every discrete functional input $X_i(t_{ij}), j = 1, ..., J_i$, where J_i varies with i, are all equivalently projected to the same $M^{(I)}$ basis functions, forming the t_{ij}-free features $f_{im} = \sum_{j=1}^{J_i} \omega_{ij} X_i(t_{ij})\phi_m^{(I)}(t_{ij}), m = \{1, 2, ..., M^{(I)}\}$. These $M^{(I)}$ features then participate in the following forward pass in place of the actual functional inputs $X_i(t_{ij})$ for training the same set of network parameters including the input weight coefficients $c_{mk}^{(I)}$, which are free of i.

The designed *feature layer*, combined with the input functional weight representation, processes irregular inputs by generalizing the problem of estimating irregular snapshots of input weight functions to estimating input weight coefficients that are consistent over all subjects.

3 Connection with existing models

3.1 Relation with FPCA

As previously pointed out by Baldi and Hornik (1989), Bengio et al. (2013), and Bourlard and Kamp (1988), a single-hidden-layer linear autoencoder with its objective function being the squared reconstruction error, i.e., $L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \|X_i - W_{\text{d}} W_{\text{e}} X_i\|^2 = \sum_{i=1}^{N_{\text{train}}} \sum_{p=1}^{P} \{X_{ip} - (W_{\text{d}} W_{\text{e}} X_i)_p\}^2$, where $X_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, W_{d}, W_{e} denote the i-th network input of P dimensions, weight matrix of the decoder and weight matrix of the encoder, respectively, is approximately identical to the conventional PCA, because such an autoencoder is learning the same subspace as the PCA. More precisely, the unique global minimum of L is corresponding to the orthogonal projection of X onto the subspace spanned by the leading eigenvectors (principal components) of the covariance matrix of X. It is worth mentioning that at the global minimum, the uniqueness occurs with the global map $W_{\text{d}} \times W_{\text{e}}$, while the matrices W_{d} and W_{e} may not be unique. This is because for multiple appropriate C we have $W_{\text{d}} \times W_{\text{e}} = (W_{\text{d}} C)(C^{-1} W_{\text{e}})$. In other words, the mapping $W_{\text{d}} \times W_{\text{e}}$ is unique but not the encoder and decoder weight matrices.

When it comes to the functional scenario, a homogeneous relationship exists between FAE and FPCA. For a single-hidden-layer FAE with the linear activation function under continuous functional data setting, the objective function measuring the mean squared reconstruction error turns out to be

$$L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \int_{\mathcal{T}} \left\{ X_i(t) - \sum_{k=1}^{K^{(1)}} \left(\int_{\mathcal{T}} X_i(t) w_k^{(I)}(t) dt \right) w_k^{(O)}(t) \right\}^2 dt. \tag{14}$$

Ramsay and Silverman (2005) concluded that the aforementioned fitting criterion is minimized when the orthonormal-restricted weight functions $w^{(\cdot)}(t)$ are precisely the same set of principal component weight functions of the functional data $X(t)$. Hence, training a one-hidden-layer linear FAE with respect to the squared reconstruction error

criterion and an orthonormal constrain on functional weights is exactly approaching to project the input $X(t)$ onto the subspace generated by the FPCs, the same space learned by FPCA.

For discrete functional data, the objective function becomes

$$L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \frac{1}{J} \sum_{j=1}^{J} \left\{ X_i(t_j) - \sum_{k=1}^{K^{(1)}} \left(\sum_{j=1}^{J} \omega_j X_i(t_j) w_k^{(I)}(t_j) \right) w_k^{(O)}(t_j) \right\}^2 . \tag{15}$$

It is important to notice that this approximation can lead to some difference which should progressively decreases as J increases. Consequently, for relatively large values of J, the FAE optimized by minimizing Eq.(15) will yield functional weights that are approximately the same as those obtained by minimizing Eq.(14). To put it differently, when subjected to the orthonormal constraint on the functional weights, the FAE that minimizes the objective model Eq.(14) is effectively learning the empirical projection of $X(t)$ onto the same space as FPCA does. Importantly, the proposed FAE with discrete configuration generalizes FPCA up to a few approximations, and the functional weights produced by FAE can be identically interpreted as the FPCs in FAE.

3.2 Relation with AE

As pointed out in Section 2.2, the proposed FAE is structurally similar to the classic AEs based on fully connected neural networks. The main difference lies in the first and last layers. In detail, a classic AE consists of network weights (and bias) free of restrictions and the training task aims at optimizing these vectors of network parameters. The proposed FAE also includes such weights to link layers between the *feature layer* and *coefficient layer*, however, the difference lies in the deterministic weights before the *feature layer* and after the *coefficient layer*, which are comprised of snapshots of continuous basis functions. With the goal of optimizing the non-deterministic weights, the main component of the FAE's training process follows the same rule as used in a conventional AE. The FAE can be regarded as an extension of a conventional AE with some deterministic operations added to both ends of the network.

The addition of *feature layer* to the AE architecture enables the FAE to quickly summarize the underlying temporal relationship among observed time span into neurons that actually step into the network, resulting in faster convergence and better generalization during network training compared to a conventional AE. Meanwhile, thanks to the application of the functional output weights, the FAE we developed can recover the discrete functional data to smooth curves over a continuous interval, satisfying the smoothness requirement of functional data, while the classic AE is limited to output discontinuous functions evaluated at some discrete timestamp of observations. Additionally, our method is capable to efficiently handle irregularly spaced functional input along with its underlying correlation in the *feature layer* by adjusting the weights ω used for numerical integration calculation, while AE has to address the issue of having insufficient observations at certain time points by training the model with some null-valued input for the corresponding neurons in the input layer. The designed

structure benefits our method with better performance in less computational cost when manipulating irregularity, which is further highlighted by a series of simulation studies in the following section.

3.3 Computational complexity

One advantage of our FAE is its potential to reduce computational cost in high-dimensional settings. As noted by Hastie et al. (2009), the computational cost of the proposed FAE is approximately $\mathcal{O}(NM^{(I)}n^w_{\text{FAE}}E)$ when back-propagation is employed (Rumelhart et al., 1986), where N is the number of observations, $M^{(I)}$ denotes the number of basis functions, n^w_{FAE} is the number of weights in the FAE, and E represents the number of training epochs. In comparison, the computational complexity of a conventional autoencoder (AE) based on a densely feed-forward neural network is of order $\mathcal{O}(NJn^w_{\text{AE}}E)$ with back-propagation, where J is the total number of distinct observed time points for the functional data, and n^w_{AE} is the number of weights in the AE.

The number of network weights n^w_{AE} (or n^w_{FAE}) is positively correlated with the dimension of network input J (or $M^{(I)}$). Typically, n^w_{AE} (or n^w_{FAE}) is set to be larger than J (or $M^{(I)}$) to ensure sufficient network capacity. For an FAE and an AE sharing the same hidden-layer architecture and training settings, n^w_{FAE} is much smaller than n^w_{AE} when $M^{(I)} < J$. Therefore, the computational cost of the FAE becomes lower compared to that of an AE. In extremely high-dimensional settings (where J is very large), the projection operator in our FAE can reduce the dimension of the actual network input from J to a much smaller value $M^{(I)}$ ($M^{(I)} \ll J$). This reduction leads to a more affordable computational cost for the FAE, as it involves fewer number of weights compared to those required for training an AE ($n^w_{\text{FAE}} \ll n^w_{\text{AE}}$ due to $M^{(I)} \ll J$). Table S1 in Section S1 of the supplementary document shows that the proposed FAE can scale much better with the number of time points J compared to the AE, providing more computationally efficient training performance when $M^{(I)}$ is much smaller than J. This demonstrates that our FAE has excellent scalability, enabling the network to manage large values of J more efficiently.

4 Simulation study

In this section, we aim to compare our proposed FAE with the two existing baseline methods it extends, FPCA and AE respectively, for representation learning and curve smoothing from discretely observed functional data. We concentrate on investigating the effectiveness of our method compared to FPCA in capturing the potential nonlinear relationship, as well as evaluating the smoothing ability and computational efficiency of FAE compared to AE.

4.1 Simulation setup

4.1.1 Data generation

We generate the data by first sampling a d-dimensional representation $\boldsymbol{Z}$ from a Gaussian mixture model. The mean vector and the covariance matrix of each component

are designed so that components are separable. We then apply a function $f(\cdot)$ that maps the representation $\boldsymbol{Z}$, to a set of M-dimensional basis coefficients B_m. Finally, we produce the continuous functional data using a linear combination of M basis functions $\gamma_m(t)$'s and the basis coefficients B_m:

$$X(t) = \sum_{m=1}^{M} B_m \gamma_m(t) = f(\boldsymbol{Z})\boldsymbol{\gamma}. \tag{16}$$

Finally, we evaluate $X(t)$ at some discrete times $\{t_1, t_2, ..., t_J\} \in [0, 1]$ to obtain a discrete version of the functional data.

The basis system used is the B-spline basis system with an order of 4, and the number of bases M varies across experiments. In terms of the mapping function $f(\cdot)$, we employ a neural network NN$(\cdot)$ with various architectures aiming to create different mapping paths from the representation vector $\boldsymbol{Z}$ to the basis coefficients of the functional data. The neural network takes the d-dimensional representation vector as input and outputs the M-dimensional basis coefficients. We apply neural networks with no hidden layers and a linear activation function for linear scenarios, and networks with at least one hidden layer and nonlinear activation functions for nonlinear scenarios.

An optional Gaussian noise can be further added to the discrete functional curve to mimic observational errors. The component of the Gaussian mixture model from which the representation is sampled is used as the label for the functional data in classification experiments.

4.1.2 Implementation of models

FPCA linearly encodes functional curves to FPC scores ξ_{im}'s with corresponding FPCs $\psi_m(t)$'s. We implement FPCA in `python` relying on the **scikit-fda** library (Ramos-Carreño et al., 2022). The discrete functional data are first converted to smooth functions using basis expansion with a customized number of B-spline basis functions, and then the conventional FPCA is performed on the estimated curve with a user-defined number of FPCs. The resulting FPC scores serve as the scalar representation of the functional data and are used for further statistical analyses.

AE based on a densely feed-forward network architecture can learn an encoding from the functional trajectory observed at discrete time points to a lower-dimensional vector of representation without considering any temporal correlations among the discrete observations. We design the input layer of AE to have J neurons with the j-th neuron representing the snapshot of the discrete functional observation at t_j. We adopt different architectures with a bottleneck hidden layer that produces the representation, experiment with both linear and nonlinear activation functions, and initialize the network weights to random values drawn from $\mathcal{N}(0, \sigma)$. We implement the AE using the **PyTorch** library.

Lastly, we implement the proposed **FAE** using **PyTorch**, along with the **scikit-fda** library for applying the basis expansion to functional weights. Analogously, we attempt with different architectures that include a hidden layer for extracting the representation, employ both linear and nonlinear activation functions in model training, and initialize the weights randomly by sampling from a Gaussian distribution $\mathcal{N}(0, \sigma)$.

The linear activation function, also known as the identity activation function, is defined as $f_{\text{Identity}}(x) = x$ and used in constructing AE and FAE for the linear scenario. The optimal activation function for each nonlinear setting is determined in advance from a list of candidates, including sigmoid, softplus, Tanh and ReLU. The sigmoid and softplus activation functions were selected using cross-validation for nonlinear scenarios in the following numerical experiments. The sigmoid activation function has the form of

$$f_{\text{Sigmoid}}(x) = \frac{1}{1+e^{-x}},$$

while the softplus activation function is formulated as

$$f_{\text{Softplus}}(x) = \begin{cases} \frac{1}{\beta}\log(1+e^{\beta x}), & x \leq \frac{\tau}{\beta}, \\ x, & x > \frac{\tau}{\beta}, \end{cases}$$

and in practice, the default values of $\beta = 1$ and $\tau = 20$ are used.

4.2 Results

A series of simulations are performed under various scenarios to investigate the performance of the proposed method in both prediction and classification, comparing it with FPCA and AE individually. The prediction error is measured by the mean squared prediction error (MSE_p) averaged across the number of samples and the number of observed time points in the test set, while the classification accuracy, $\text{P}_{\text{classification}}$, is calculated as the percentage of test observations that can be labelled correctly by a logistic regression based on the representations extracted. For each scenario, we report the mean and standard deviation (SD) of the evaluation metrics across all replications.

4.2.1 FAE vs. FPCA

Scenario 1.1 (Linear & Regular): 6000 discrete functional observations evaluated at 21 equally spaced points over the interval $[0, 1]$ are simulated. A five-dimensional Gaussian mixture model with three components is used to generate the representations and the resulting functional curves are labelled with class 1, 2 and 3. A neural network with no hidden layers and a linear activation function is performed to map the representation to the basis coefficients. We employ 8 B-spline basis functions ($M = 8$) along with the aforementioned basis coefficients to express the underlying functional curves.

We assign 80% of the observations by random to the training set and the remainder to the test set. The FPCA and two types of FAE are successively trained and the model details are summarized in Table S2 in the supplementary document.

Scenario 1.2 (Nonlinear & Regular): We generate 3000 functional observations discretely measured at 51 equally spaced points over the interval $\mathcal{T} = [0, 1]$. We sample a 5-dimensional representation for each curve from a 3-component Gaussian mixture model and label the associated functional curves with class 1, 2 and 3. We map the representations to the basis coefficients using a neural network with one hidden layer

Table 1: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity activation function (FAE(Identity)), functional autoencoder with the softplus activation function (FAE(Softplus)) and functional principal component analysis (FPCA) on 10 random test data sets in Scenario 1.1, with the best results being highlighted in bold.

		FAE (Identity)	**FAE** (Softplus)	**FPCA**
MSE_p	3 Reps	0.0050(0.0001)	**0.0045**(0.0005)	0.0052(0.0001)
	5 Reps	**0.0019(<0.0001)**	0.0022(0.0003)	0.0021(<0.0001)
	10 Reps	**0.0009(<0.0001)**	0.0017(0.0005)	0.0010(<0.0001)
$P_{classification}$	3 Reps	87.24%(0.93%)	**87.72%**(1.62%)	87.68%(0.78%)
	5 Reps	87.94%(0.81%)	86.53%(0.94%)	**89.21%**(0.78%)
	10 Reps	89.16%(0.75%)	**89.61%**(0.99%)	89.22%(0.70%)

Table 2: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity activation function (FAE(Identity)), functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) and functional principal component analysis (FPCA) on 10 random test data sets in Scenario 1.2, with the best results being highlighted in bold.

		FAE (Identity)	**FAE** (Sigmoid)	**FPCA**
MSE_p	3 Reps	0.0070(0.0002)	**0.0038**(0.0002)	0.0070(0.0002)
	5 Reps	0.0035(0.0001)	**0.0026**(0.0004)	0.0036(0.0001)
	10 Reps	**0.0013**(<0.0001)	0.0014(<0.0001)	**0.0013**(<0.0001)
$P_{classification}$	3 Reps	85.05%(1.08%)	**88.68%**(1.46%)	85.17%(1.06%)
	5 Reps	86.62%(1.06%)	**92.42%**(1.02%)	86.65%(1.28%)
	10 Reps	87.55%(1.13%)	**91.20%**(1.06%)	87.53%(1.26%)

with 20 neurons and the sigmoid activation function. Afterwards, individual functional curve is constructed using 10 B-spline basis functions and the basis coefficients described above.

We continue to randomly generate training and test sets that contain 80% and 20% observations, respectively. Again, we put FPCA, linear FAE and nonlinear FAE in comparison with model configuration adjusted and detailed in Table S3 in the supplementary document.

Table 1 and Table 2 summarize the predictive and classification performances of the proposed FAEs and FPCA. In the linear & regular context, we observe that all three approaches in comparison yield similar performance in both prediction and classification for most representation attempts.

In contrast, under the nonlinear scenario, both linear FAE (FAE with the identity activation function) and FPCA generate relatively higher MSE_p and lower $P_{classification}$

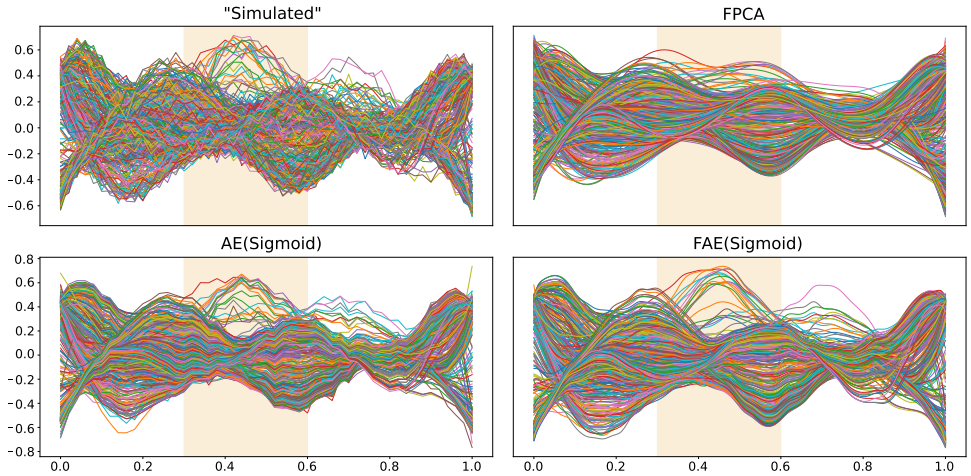


Fig. 6: The simulated curves and the curves recovered by functional principal component analysis (FPCA), classic autoencoder with the sigmoid activation function (AE(Sigmoid)) and functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) using 5 representations for a random test set in Scenario 1.2 and Scenario 2.1.

due to the violation of the linearity assumption. Meanwhile, the FAE with the sigmoid activation function retains superior performance in both prediction and classification in comparison to the linear approaches, with only minimal difference when predicting with 10 representations. This indicates that the nonlinear FAE can capture and comprise the information carried by the discrete data more efficiently and accurately.

With regard to curve smoothing, as displayed in Figure 6, both FPCA and FAE with the sigmoid activation function produce smooth curves based on the inputted discrete observations, while the designed FAE demonstrates additional benefits in curve recovery. Plainly, FAE can not only correctly reconstruct the complete moving trend but also sensitively capture the individual pop-up variations, e.g. the local ∩-shaped mode appearing in the shaded interval.

4.2.2 FAE vs. AE

Scenario 2.1 (Nonlinear & Regular): The simulated data used in the scenario 1.2 in Section 4.2.1 is simultaneously applied for a comparison between FAE and AE. Again, 80% of the random observations are assigned to the training set and the remaining 20% to the test set. Given that this scenario follows a nonlinear setting, we emphasize the nonlinear models by training both the baseline AE and the proposed FAE using the model configurations listed in Table S4 of the supplementary document.

Table 3 presents the means and SDs of MSE_p and $\text{P}_{\text{classification}}$ over 10 replicates trained by AE and FAE with the sigmoid activation function in this nonlinear and regularly-spaced-data scenario. We observe that the two methods achieve competitive performance in representation learning, with the nonlinear AE giving slightly better

Table 3: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) and classic autoencoder with the sigmoid activation function (AE(Sigmoid)) on 10 random test data sets in Scenario 2.1, with the better results being highlighted in bold.

	FAE (Sigmoid)			**AE (Sigmoid)**		
	3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	**0.0038**(0.0002)	**0.0026**(0.0004)	**0.0014**(<0.0001)	0.0046(0.0005)	0.0030(0.0005)	0.0124(0.0069)
$P_{classification}$	88.68%(1.46%)	92.42% (1.02%)	91.02%(1.06%)	**89.35%**(1.39%)	**92.75%** (1.15%)	**92.65%**(1.81%)

results in classifying curves, while the nonlinear FAE excels with smaller predictive errors in reconstructing the functional observations. Figure 6 clearly shows that the proposed FAE can directly and accurately output smooth curves using the given discrete observations for the entire domain, while AE is limited to discretely recover the curve at the time points with available observations, indicating that our FAE is capable of efficiently capturing the representative information and simultaneously smoothing the discretely functional observation.

Scenario 2.2 (Nonlinear & Irregular): In this scenario, we continue to use the data simulated in scenario 1.2 and randomly remove measurements in 25 time points (excluding the start and end time points) for each curve to create irregularly and discretely observed functional data, that is, the resulting functional curve contains 26 irregular observations individually over the domain interval $\mathcal{T}$.

We experiment with two different training set settings: (i) the training set contains 80% observations; (ii) the training set contains 20% data, and focus on a comparison between the nonlinear AE and nonlinear FAE with configurations provided in Table S5 in the supplementary document to examine their performance in handling nonlinearity and irregularity simultaneously. For those time points without observations (randomly removed), we feed the corresponding neurons in the input layer of AE and FAE with zeros and exclude those neurons from the loss computation. When training FAE, we also adjust the weights $\{\omega_j\}_{j=1}^{J_i}$ individually for each discrete curve i for a reasonable numerical integration over all the observed timestamps.

The performances of prediction and classification of nonlinear AE and nonlinear FAE, trained with 80% and 20% irregularly spaced functional data, are illustrated in Table 4 and Table 5, separately, with the performances of both models reported for each thousand epochs. We can see that the proposed FAE shows more advantages in speedily learning the representation and accurately capturing the information for both prediction and classification, especially when the training epochs remain small. On the other hand, the classic AE needs to gradually master the mapping path in respect of reconstruction error, while its resulting representations can outperform those by FAE in classification when the training cost increases. The visual comparisons of how the mean MSE_p and mean $\text{P}_\text{classification}$ of FAE and AE change with the number of training epochs for 80% and 20% training sizes, corresponding to Table 4 and Table 5, are presented in Section S2.2 of the supplementary document. As demonstrated, the computational efficiency of the FAE is robust across different representation numbers, which further confirms that the FAE is able to generalize better and converge faster even with fewer epochs and larger batch size compared to traditional AE that has similar architecture in terms of curve reconstruction and unsupervised representation learning for classification.

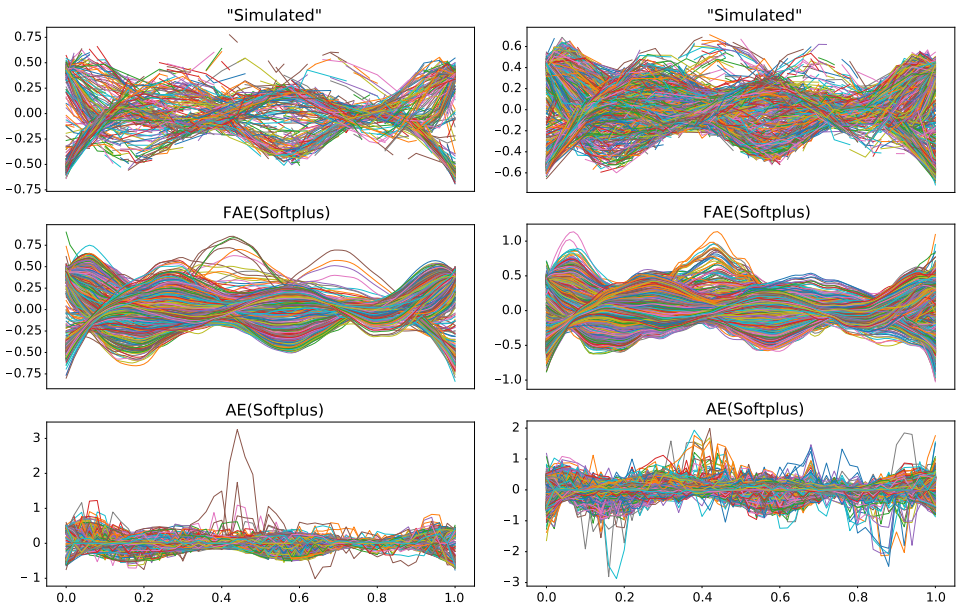
Apart from representation learning, we display the simulated irregularly spaced functional segments, along with the full curves reconstructed by the nonlinear AE and nonlinear FAE in Figure 7 to reveal the smoothing ability of the FAE. When training with 80% observations, it is not surprising to observe that the proposed FAE oversteps the classic AE by generating predominantly smooth curves that effectively capture the entire underlying patterns and primary modes present in the originally observed data. In contrast, trajectories obtained through AE exhibit noticeable oscillations and

Table 4: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the softplus activation function (FAE(Softplus)) and classic autoencoder with the softplus activation function (AE(Softplus)) on 10 random test data sets when training with 80% irregularly observed data in Scenario 2.2, with the better results being highlighted in bold.

		FAE (Softplus)			**AE (Softplus)**		
		3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	epochs=1000	**0.0031**(0.0003)	**0.0023**(0.0002)	**0.0014**(0.0002)	0.0035(0.0002)	0.0029(0.0003)	0.0143(0.0127)
	epochs=2000	**0.0023**(0.0001)	**0.0015**(<0.0001)	**0.0010**(<0.0001)	0.0034(0.0003)	0.0044(0.0059)	0.0103(0.0102)
$P_{classification}$	epochs=1000	86.57%(1.08%)	87.85%(2.03%)	89.22%(1.17%)	**89.85%**(1.32%)	**91.05%**(0.69%)	**90.58%**(1.59%)
	epochs=2000	88.67%(1.22%)	90.12%(1.70%)	**91.75%**(1.10%)	**90.68%**(1.30%)	**91.03%**(1.09%)	90.73%(1.66%)

Table 5: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the softplus activation function (FAE(Softplus)) and classic autoencoder with the softplus activation function (AE(Softplus)) on 10 random test data sets when training with 20% irregularly observed data in Scenario 2.2, with the better results being highlighted in bold.

		FAE (Softplus)			**AE (Softplus)**		
		3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	epochs=1000	**0.0057**(0.0009)	**0.0041**(0.0009)	**0.0039**(0.0026)	0.0386(0.0152)	0.0730(0.0237)	0.4591(0.2692)
	epochs=2000	**0.0046**(0.0011)	**0.0030**(0.0004)	**0.0025**(0.0007)	0.0194(0.0094)	0.0464(0.0266)	0.3579(0.2449)
	epochs=3000	**0.0035**(0.0003)	**0.0027**(0.0003)	**0.0019**(0.0004)	0.0104(0.0027)	0.0230(0.1578)	0.1917(0.0934)
	epochs=4000	**0.0031**(0.0002)	**0.0021**(<0.0001)	**0.0029**(0.0039)	0.0086(0.0012)	0.0093(0.0037)	0.0968(0.0632)
	epochs=5000	**0.0029**(0.0002)	**0.0019**(0.0001)	**0.0015**(0.0004)	0.0094(0.0010)	0.0070(0.0015)	0.0588(0.0434)
$P_{classification}$	epochs=1000	78.32%(1.10%)	**81.59%**(2.12%)	**82.30%**(2.84%)	**78.71%**(2.97%)	80.48%(2.30%)	64.36%(5.26%)
	epochs=2000	81.40%(2.00%)	83.86%(1.17%)	**83.50%**(1.20%)	**85.30%**(0.82%)	**86.16%**(1.39%)	80.63%(6.87%)
	epochs=3000	84.09%(1.06%)	85.75%(1.24%)	85.27%(1.02%)	**86.70%**(1.11%)	**87.50%**(1.57%)	**88.61%**(1.29%)
	epochs=4000	85.05%(0.72%)	86.69%(1.26%)	87.17%(1.04%)	**87.18%**(1.27%)	**88.03%**(1.17%)	**90.05%**(0.65%)
	epochs=5000	85.53%(0.94%)	87.63%(1.26%)	88.27%(1.34%)	**87.50%**(1.25%)	**88.23%**(0.86%)	**89.98%**(0.63%)



(a) 80% observations in training set. (b) 20% observations in training set.

Fig. 7: The simulated irregularly spaced curves and the curves recovered by classic autoencoder with the softplus activation function (AE(Softplus)) and functional autoencoder with the softplus activation function (FAE(Softplus)) using 5 representations for a random test set in Scenario 2.2, when training with 80% observations (left panel) and 20% observations (right panel), respectively.

incoherence with numerous accidents protrudes appearing across the entire domain. In the case of training with only 20% data, as expected, our FAE continues its dominance by showing dramatically leading performance in curve smoothing. This highlights that the FAE, with its specially designed architecture, is able to efficiently learn the representations and simultaneously smooth unequally spaced and noisy functional data, even with limited training information.

5 Real application

To further demonstrate the effectiveness of our method, in this section, we perform the proposed FAE, together with the conventional FPCA and the classic AE on the El Niño data set which is available in `R` package **rainbow** (Shang and Hyndman, 2019). This data set catalogs the monthly sea surface temperatures originally observed in 4 different locations from January 1950 to December 2006. The temperature curves were discretely measured at 12 evenly spaced time points over the entire interval for every year. We treat the measurements from each calendar year as independent observations of the true underlying functional curve (varying with time), resulting in a total of

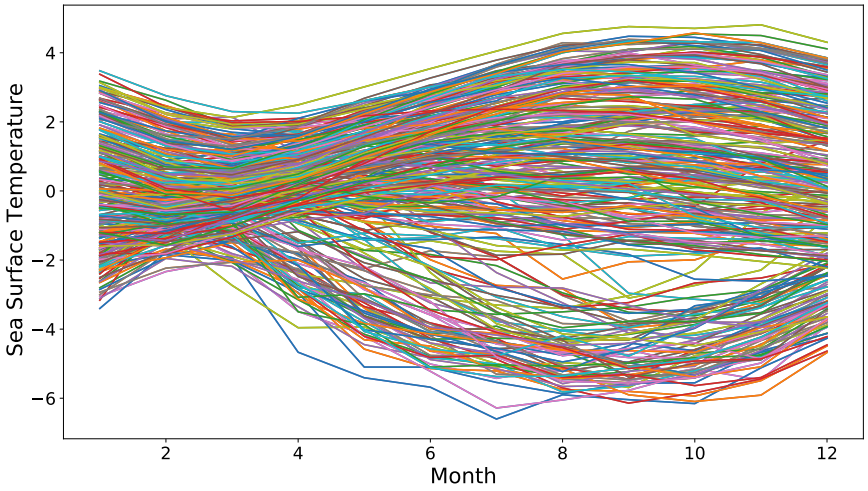


Fig. 8: Centered monthly sea surface temperature measured in the "Niño region" defined by the coordinates 0-10 degree South and 90-80 degree West.

267 observations. The four locations are labeled with numbers from 1 to 4 randomly. To avoid poor random initialization and obtain stable training process for the neural network-based methods, we centre the data by subtracting the sample mean curve across all observations. A visualization of the centered sea surface temperature curves is provided by Figure 8.

We continue to compare the proposed method with two benchmark models, FPCA and AE, on their performances in terms of curve reconstruction and representation extraction. We equip the classic AE and the proposed FAE with different combinations of hyperparameters for a linear mapping path and a potential nonlinear mapping pattern, while FPCA is performed with the focus on measuring the linear relationship. The hyperparameters for all models in comparison are tuned in advance to yield a fair improvement in their performances during the actual training. To reduce the computational cost of the tuning process, for each model, we fix the number of representations to be 5 and then perform a grid search over the hyperparameters of our interests with respect to the loss function by simply building a model for each possible combination of all of the hyperparameter values provided. The optimal model architecture combination of hyperparameters identified by the grid search with 5 representations is further applied to model training with 3 and 8 representations. In the supplementary document, Table S6 provides a summary of the candidate values considered in hyperparameter tuning for all models, while Table S7 details the optimally identified configurations for models in comparison. We proceed with 20 repetitions of random subsampling validation: randomly dividing the data set into a training set and a test set, with 80% and 20% of the total observations assigned to them, respectively. We evaluate the prediction and classification performances of all the mentioned approaches on the 20 test sets using 3, 5, and 8 representations, respectively.

Table 6: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity (FAE(Identity)) and the sigmoid activation function (FAE(Sigmoid)), classic autoencoder with the identity (AE(Identity)) and the sigmoid activation function (AE(Sigmoid)) and functional principal component analysis (FPCA) on 20 random test sets with the El Niño data set.

		FAE (Identity)	**FAE (Sigmoid)**	**AE (Identity)**	**AE (Sigmoid)**	**FPCA**
MSE_p	3 reps	0.0616(0.0051)	**0.0582**(0.0045)	0.0619(0.0051)	0.0715(0.0079)	0.0656(0.0054)
	5 reps	**0.0211**(0.0023)	0.0226(0.0031)	0.0261(0.0052)	0.0329(0.0042)	0.0242(0.0031)
	8 reps	**0.0062**(0.0009)	0.0089(0.0014)	0.0064(0.0008)	0.0071(0.0021)	0.0113(0.0013)
$P_{classification}$	3 reps	76.88%(4.01%)	**77.68%**(5.07%)	76.52%(3.67%)	77.14%(6.05%)	77.59%(4.81%)
	5 reps	85.18%(4.86%)	**86.52%**(4.46%)	84.20%(5.15%)	85.71%(3.48%)	84.38%(5.20%)
	8 reps	85.89%(4.58%)	**87.59%**(4.67%)	85.27%(3.91%)	85.80%(3.89%)	84.81%(4.50%)

Table 6 summaries the performances of all the methods applied for different numbers of representation on curve prediction and classification, using MSE_p and $\text{P}_\text{classification}$ averaged over 20 random test sets. Apparently, the proposed FAEs consistently and comprehensively outperform the FPCA and the AE models in both reconstruction and classification, by reaching the lowest prediction error and the highest classification accuracy for all representation attempts. With regard to the predictive performance, the linear FAE retains to be the top performer, closely followed by the nonlinear FAE. On the other hand, the nonlinear FAE continuously oversteps the other models in the competition of classifying curves, exhibiting its advantages in extracting more informative representations. To further confirm this in the context of statistical significance, we conduct two-sided paired t-tests to compare the MSE_p and $\text{P}_\text{classification}$ of the 20 replicates of the nonlinear FAE to those of the FPCA, and the corresponding p-values are reported in Table S8 in the supplementary document. We observe that the nonlinear FAE remains superior to the FPCA in both evaluation metrics, particularly as the representation size increases. This demonstrates the importance and necessity of the proposed FAE in nonlinear representation learning.

The other highlight of the proposed FAE is its capability of smoothing the originally discrete data. As illustrated in Figure 9, the trajectories recovered by using FAE are smooth over the entire domain due to the fact that they are constructed as the linear combination of the neurons in the *coefficient layer* and the basis functions that can be evaluated at any point within the interval of observation. On the contrary, the classic AE can only achieve point-wise prediction at the timestamp with actual observations, resulting in visible discontinuity in the curve reconstruction. Meanwhile, FPCA can also produce smooth prediction but it usually requires the discrete observation to be firstly smoothed.

In addition, FAE surpasses AE by converging faster to a similarly low prediction error and achieving a higher classification accuracy in both linear and nonlinear scenarios, as displayed in Figure 10 and Figure 11, demonstrating its high efficiency in extracting meaningful features and potential advantage in reducing computational costs. It is also noteworthy that the nonlinear AE has a simpler model configuration compared to the nonlinear FAE, which brings benefits to the training speed of nonlinear AE.

The given results suggest that, for the El Niño data, the true relationship between the functional space and the representation space for the sea temperature curves might be more accurately captured through a nonlinear mapping path, which could yield nonlinear representations that carry more valuable information advantageous for subsequent statistical analysis.

6 Conclusion

In this work, we introduced autoencoders with a novel architecture designed for discrete functional observations, aiming to achieve unsupervised representation learning and direct curve smoothing concurrently. The deterministic *feature layer* added to the encoder reduces the computational effort and enhances the model robustness in learning performance, while the additional *coefficient layer* incorporated into the decoder ensures the usage of back-propagation in model training and allows the decompression from

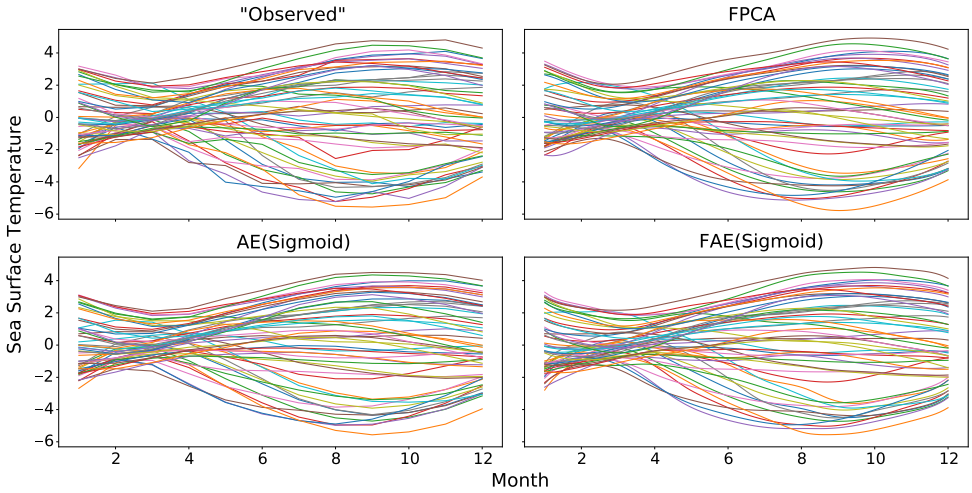


Fig. 9: The observed curves and curves recovered by functional principal component analysis (FPCA), classic autoencoder with the sigmoid activation function (AE(Sigmoid)) and functional autoencoder (FAE(Sigmoid)) with 5 representations for a test set of El Niño data set.

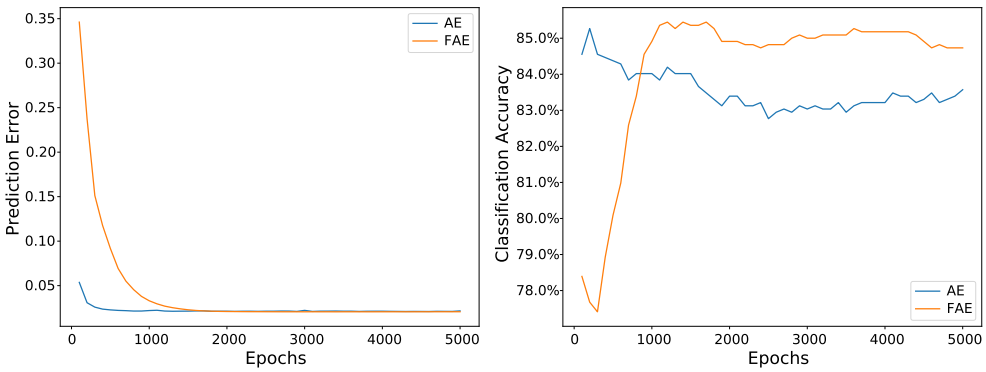


Fig. 10: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the identity activation function using 5 representations on 20 random test sets of the ElNino data set change with the number of epochs.

scalar neurons to functional curve in a linear manner. Additionally, we implemented our proposed FAE in a way to accommodate both regularly and irregularly functional data, with flexible necessity on the size of training data for achieving satisfactory performance. Through several simulation studies and a real application, we demonstrated that our proposed method is superior to the classic linear representation method for functional data, FPCA, in nonlinear scenarios and maintains competitive performance in linear

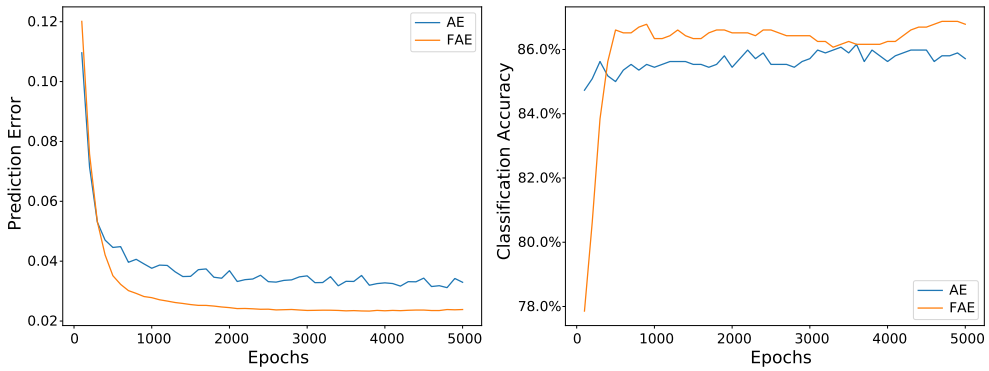


Fig. 11: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the sigmoid activation function using 5 representations on 20 random test sets of the ElNino data set change with the number of epochs.

cases. Moreover, our numerical experiments endorse that our model can lead to an improvement over the classic AE in terms of prediction by generalizing and converging rapidly even with reduced training observations.

Nevertheless, the developed method depends on numerous hyperparameters, including the number of hidden layers, the number of neurons in each hidden layer, the training optimizer, etc., and unfortunately, conducting a grid search on that space can be time-consuming. It is necessary to bring up that the performance of FAE varies from one hyperparameter configuration to another, and the randomness in initializing network parameters will introduce more variance to the results across training replicates. In contrast, FPCA can be effortlessly fitted with only a few hyperparameters necessitating predetermination, but in sacrifice of the ability to accurately capturing the learning maps in nonlinear situations. Another weakness of our approach is its inability to process multidimensional functional data in its current form. Therefore, in a future work we could extend the current network architecture to a more dynamic architecture allowing discrete multivariate functional data. This might be achieved by introducing micro-neural networks (Lin et al., 2014, Yao et al., 2021) to replace the neurons in the *feature layer* and the *coefficient layer*. Furthermore, an analogous architecture of our proposed FAE can be implemented to tackle nonlinear functional regression problems with both a functional predictor and a functional response. More avenues for addressing the nonlinear representation and other complex aspects of functional data can be explored by utilizing various machine learning techniques. For instance, normalizing flows (Kobyzev et al., 2021, Rezende and Mohamed, 2015, Tabak and Turner, 2013, Tabak and Vanden-Eijnden, 2010) can be further developed to adapt functional data. With a well-designed architecture, a functional normalizing flow would excel not only in representation learning, but also in density estimation and generative modeling of functions.

Declarations

The authors gratefully acknowledge the support by the Discovery grants (RGPIN-2023-05155 and RGPIN-2023-04057) from the Natural Sciences and Engineering Research Council of Canada (NSERC) to C. Beaulac and J. Cao, respectively and the Canada Research Chair program.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, 1988.

Dong Chen and Hans-Georg Müller. Nonlinear Manifold Representations for Functional Data. *The Annals of Statistics*, 40(1):1–29, 2012.

Kehui Chen and Jing Lei. Localized functional principal component analysis. *Journal of the American Statistical Association*, 110(511):1266–1275, 2015.

Jeng-Min Chiou and Pai-Ling Li. Functional Clustering and Identifying Substructures of Longitudinal Data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(4):679–699, 2007. doi:10.1111/j.1467-9868.2007.00605.x.

Jacques Dauxois, Alain Pousse, and Yves Romain. Asymptotic theory for the principal component analysis of a vector random function: Some applications to statistical inference. *Journal of Multivariate Analysis*, 12(1):136–154, 1982.

Paul H. C. Eilers and Brian D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2):89–121, 1996. doi:10.1214/ss/1038425655.

Frédéric Ferraty and Philippe Vieu. *Nonparametric Functional Data Analysis: Theory and Practice*. Springer-Verlag, New York, 2006.

Peter Hall and Mohammad Hosseini-Nasab. On properties of functional principal components analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(1):109–126, 2006.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, 2009.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Tsung-Yu Hsieh, Yiwei Sun, Suhang Wang, and Vasant Honavar. Functional autoencoders for functional data representation learning. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 666–674, 2021.

I. Kobyzev, S. D. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 43(11):3964–3979, 2021. ISSN 1939-3539. doi:10.1109/TPAMI.2020.2992934.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv*, 2014. doi:10.48550/arXiv.1312.4400.

Jens Meiler, Michael Müller, Anita Zeidler, and Felix Schmäschke. Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Molecular Modeling Annual*, 7(9):360–369, 2001.

Hans-georg Müller. Functional modelling and classification of longitudinal data. *Scandinavian Journal of Statistics*, 32(2):223–240, 2005.

Hans-Georg Müller and Ulrich Stadtmüller. Generalized Functional Linear Models. *The Annals of Statistics*, 33(2):774–805, 2005. doi:10.1214/009053604000001156.

Hans-Georg Müller and Fang Yao. Functional additive models. *Journal of the American Statistical Association*, 103(484):1534–1544, 2008.

Yunlong Nie and Jiguo Cao. Sparse functional principal component analysis in a new regression framework. *Computational Statistics & Data Analysis*, 152:107016, 2020.

Yunlong Nie, Liangliang Wang, Baisen Liu, and Jiguo Cao. Supervised functional principal component analysis. *Statistics and Computing*, 28:713–723, 2018.

Yunlong Nie, Yuping Yang, Liangliang Wang, and Jiguo Cao. Recovering the underlying trajectory from sparse and irregular longitudinal data. *Canadian Journal of Statistics*, 50:122–141, 2022.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

Erkki Oja. Principal components, minor components, and linear neural networks. *Neural networks*, 5(6):927–935, 1992.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Jie Peng and Hans-Georg Müller. Distance-based clustering of sparsely observed stochastic processes, with applications to online auctions. *The Annals of Applied Statistics*, 2(3):1056–1077, 2008.

Jie Peng and Debashis Paul. A geometric approach to maximum likelihood estimation of the functional principal components from sparse longitudinal data. *Journal of Computational and Graphical Statistics*, 18(4):995–1015, 2009.

Carlos Ramos-Carreño, José Luis Torrecilla, Yujian Hong, and Alberto Suárez. scikit-fda: Computational tools for machine learning with functional data. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 213–218, 2022. doi:10.1109/ICTAI56018.2022.00038.

James O. Ramsay and Bernard W. Silverman. *Functional Data Analysis (Second Edition)*. Springer, New York, 2005.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1530–1538. PMLR, 2015. URL https://proceedings.mlr.press/v37/rezende15.html.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Peijun Sang, Liangliang Wang, and Jiguo Cao. Parametric functional principal component analysis. *Biometrics*, 73(3):802–810, 2017.

Hanlin Shang and Rob Hyndman. *rainbow: Bagplots, Boxplots and Rainbow Plots for Functional Data*, 2019. URL https://CRAN.R-project.org/package=rainbow. R package version 3.6.

Haolun Shi, Jianghu Dong, Liangliang Wang, and Jiguo Cao. Functional principal component analysis for longitudinal data with informative dropout. *Statistics in*

Medicine, 40:712–724, 2021.

Jun Song and Bing Li. Nonlinear and additive principal component analysis for functional data. *Journal of Multivariate Analysis*, 181:104675, 2021. ISSN 0047-259X.

E. G. Tabak and Cristina V. Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. doi:https://doi.org/10.1002/cpa.21423.

Esteban G. Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217 – 233, 2010.

Haixu Wang and Jiguo Cao. Functional nonlinear learning. *Journal of Computational and Graphical Statistics*, 33:181–191, 2024. doi:10.1080/10618600.2023.2233581.

Jane-Ling Wang, Jeng-Min Chiou, and Hans-Georg Müller. Functional data analysis. *Annual Review of Statistics and Its Application*, 3(1):257–295, 2016a.

Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016b.

Sidi Wu, Cédric Beaulac, and Jiguo Cao. Neural networks for scalar input and functional output. *Statistics and Computing*, 33(5):118, 2023.

Fang Yao, Hans-Georg Müller, and Jane-Ling Wang. Functional data analysis for sparse longitudinal data. *Journal of the American statistical association*, 100(470): 577–590, 2005a.

Fang Yao, Hans-Georg Müller, and Jane-Ling Wang. Functional Linear Regression Analysis for Longitudinal Data. *The Annals of Statistics*, 33(6):2873–2903, 2005b.

Fang Yao, Yuejiao Fu, and Thomas C. M. Lee. Functional Mixture Regression. *Biostatistics*, 12(2):341–353, 2010. ISSN 1465-4644.

Junwen Yao, Jonas Mueller, and Jane-Ling Wang. Deep learning for functional data analysis with adaptive basis layers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 11898–11908, 2021.

Rou Zhong, Shishi Liu, Haocheng Li, and Jingxiao Zhang. Functional principal component analysis estimator for non-gaussian data. *Journal of Statistical Computation and Simulation*, 92(13):2788–2801, 2022.