FUNCTIONAL AUTOENCODER

MATH SEMINAR(2025 FALL)

Peida Wu
Institute of Math and Science
ShanghaiTech University
wupd2022@shanghaitech.edu.cn

December 30, 2025

ABSTRACT

High–dimensional data sets in modern applications often take the form of curves or trajectories rather than finite–dimensional vectors. Classical representation learning techniques such as Principal Component Analysis (PCA) and its functional analogue, Functional PCA (FPCA), provide linear low–dimensional summaries but are fundamentally limited when the underlying structure of the data is nonlinear. At the same time, autoencoders and their probabilistic variants, most notably the Variational Autoencoder (VAE), have proved to be powerful tools for nonlinear representation learning in Euclidean spaces.

In this report we bridge these two lines of work. We first review standard autoencoders, several widely–used variants, and the VAE together with the underlying variational inference framework. We then introduce basic concepts from functional data analysis, including basis representations and FPCA, emphasizing their interpretation as linear methods. Building on the functional autoencoder (FAE) proposed in recent work, we present a functional extension of the autoencoder in which both encoder and decoder employ weight functions and L^2 inner products to operate directly on functional inputs. A key ingredient of the model is the use of functional derivatives to derive backpropagation updates for these weight functions. Finally, we discuss how the FAE can be applied to multi–dimensional motion data to learn smooth, low–dimensional latent representations that capture nonlinear temporal patterns.

Keywords Autoencoder, Variational autoencoder, Functional data analysis, Functional autoencoders

1 Introduction

Many modern data sets are high–dimensional and structured, and it is often advantageous to represent them in a lower–dimensional latent space. Autoencoders address this problem by learning an encoder that maps an input x to a latent code z and a decoder that reconstructs $\hat{x}$ from z. When trained to minimize reconstruction error, the latent variables capture the main modes of variation in the data and can be used for visualization, clustering, or as features for downstream tasks. Linear autoencoders are closely related to Principal Component Analysis (PCA), while nonlinear encoders and decoders extend this idea to more complex structures.

In many scientific applications, however, each observation is naturally a *function* of time or space, such as growth curves, spectra, or motion trajectories. Functional Data Analysis (FDA) treats these objects as random functions and provides linear tools such as Functional PCA (FPCA). FPCA assumes that functional observations lie near a low–dimensional *linear* subspace spanned by eigenfunctions of the covariance operator. This linearity is often too restrictive: real functional data typically show nonlinear temporal patterns, local warping and nonlinear dependence across different functional coordinates.

These limitations motivate combining autoencoders with FDA. Rather than discretizing curves and applying a standard autoencoder, a Functional Autoencoder (FAE) replaces scalar weights by weight functions and uses L^2 inner products so that the network operates directly on functions. The encoder maps multi–dimensional functional inputs to a

finite–dimensional latent representation, and the decoder reconstructs the original functions. A key ingredient is the use of functional derivatives to derive backpropagation updates for these weight functions. In this report we review the necessary background on autoencoders and functional data, present the architecture and training objective of the FAE, and derive the corresponding functional gradients.

2 Introduction to Autoencoder

2.1 Motivation

High-dimensional data—such as images, time series, and functional observations—are ubiquitous in modern applications. Direct computation in the ambient space is often inefficient and fragile due to noise, redundancy, limited sample sizes, and the curse of dimensionality. Consequently, it is desirable to learn a compact latent representation z that captures the essential structure of an observation x while discarding irrelevant variability.

An *autoencoder* implements this idea via a pair of maps: an encoder f_ϕ that compresses the input and a decoder g_θ that reconstructs it,

$$z = f_\phi(x), \qquad \hat{x} = g_\theta(z) = g_\theta\big(f_\phi(x)\big).$$

When trained appropriately, the latent variable z serves as a low-dimensional summary that is both computationally efficient and robust for downstream tasks. Practical designs use a bottleneck (undercomplete architecture) or explicit regularization (denoising, sparsity, contractive penalties, variational constraints) to avoid the trivial identity mapping and to encourage representations that generalize.

2.2 Architecture and Loss

An autoencoder consists of three core components:

- **Encoder** $f_\phi : \mathcal{X} \to \mathcal{Z}$: maps input x to a low-dimensional latent code z.
- **Latent space** $\mathcal{Z} \subseteq \mathbb{R}^d$: the bottleneck representation (typically $d \ll \dim(\mathcal{X})$).
- **Decoder** $g_\theta : \mathcal{Z} \to \mathcal{X}$: reconstructs $\hat{x}$ from z.

The empirical objective is to minimize reconstruction loss over a dataset $\{x_i\}_{i=1}^n$. Using a generic per-sample loss $\ell(\cdot,\cdot)$ (e.g. squared error or cross-entropy),

$$\min_{\phi,\theta} \; L_n(\phi,\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(x_i,\, g_\theta(f_\phi(x_i))\big). \tag{1}$$

For mean squared error (MSE) one writes $\ell(x,\hat{x}) = \|x-\hat{x}\|^2$.

Models are trained end-to-end by stochastic optimization (e.g. SGD/Adam) using backpropagation through both encoder and decoder. The combination of a bottleneck and nonlinear model yield compact, robust representations for downstream analysis.

2.3 Autoencoder vs. PCA

At first glance an autoencoder resembles Principal Component Analysis (PCA): both provide a way to compress high-dimensional data into a low-dimensional representation. They are, however, conceptually and practically different in important ways. Below we state the classical equivalence result for the linear case and then summarize the main contrasts.

Theorem 1. *Consider an autoencoder with a single hidden layer (latent dimension k), linear encoder $f(x) = W_e x$ and linear decoder $g(z) = W_d z$, trained with squared-error loss*

$$\min_{W_e,W_d} \; \mathbb{E}\big\|x - W_d W_e x\big\|^2. \tag{2}$$

Then any optimal product $P^\star = W_d^\star W_e^\star$ is a rank-k orthogonal projector onto the subspace spanned by the top-k principal components of the data covariance. In other words, a linear autoencoder recovers the PCA subspace (up to an invertible transform on the latent coordinates).

Remarks.

- The equivalence requires linear encoder/decoder and MSE loss; introducing nonlinearity (activation functions, deep stacks) breaks the closed-form equivalence and yields strictly more expressive representations.
- In practice PCA has a closed-form solution via SVD and is deterministic, while autoencoders (even linear ones in parameter space) are typically trained by gradient methods and suffer from optimization nonconvexity and factorization non-uniqueness.
- Regularized/structured autoencoders (denoising, sparse, contractive, variational) trade reconstruction for robustness, sparsity, or generative modeling — capabilities beyond classical PCA.

Table 1: Comparison between PCA and Autoencoders

	PCA	**Autoencoder**
Model	Linear orthogonal projection (principal subspace)	Parametric neural network(nonlinear)
Solution	Closed-form via SVD / eigendecomposition	Learned by optimization (SGD/Adam); may be nonconvex
Expressiveness	Linear subspaces only	Can be nonlinear and deep (much greater representational power)
Objective	Maximize projected variance	Minimize reconstruction error
Latent output	Principal component coordinates (orthogonal)	Latent code
Typical use cases	Linear dimensionality reduction, visualization, preprocessing	Nonlinear representation learning, denoising, generative modeling, feature learning

2.4 Some common autoencoder variants (brief overview)

Below are concise introductions to several widely used autoencoder variants. Each entry gives the core idea, the typical objective (when helpful) and a short remark on use-cases and trade-offs.

Denoising Autoencoder (DAE)

Idea. Improve robustness by forcing the model to recover clean inputs from corrupted versions. **Objective.** Given a corruption process $\tilde{x} \sim q(\tilde{x} \mid x)$,

$$\min_{\phi,\theta} \ \mathbb{E}_x \, \mathbb{E}_{\tilde{x}|x} \, \ell\big(x, \, g_\theta(f_\phi(\tilde{x}))\big),$$

often with ℓ the squared error. **Remark.** Learns representations that are stable to local perturbations and often improves generalization for noisy data.

Sparse Autoencoder

Idea. Encourage sparse activations in the latent layer so that each code uses few active units (interpretability and feature disentangling).
Objective. Reconstruction loss plus a sparsity penalty on encoder activations $a_j(x)$, e.g.

$$\min_{\phi,\theta} \ \frac{1}{n} \sum_i \ell\big(x_i, g_\theta(f_\phi(x_i))\big) + \lambda \sum_j \mathrm{penalty}\big(\mathbb{E}_i[a_j(x_i)]\big), \tag{3}$$

where $\mathrm{penalty}$ can be an L_1 term or a KL divergence to a small target activation. **Remark.** Useful when one desires localized or disentangled features; choice of penalty controls sparsity strength.

Variational Autoencoder (VAE)

Idea. Treat the encoder as a probabilistic inference network and train a generative model by maximizing a variational lower bound (enables principled sampling and uncertainty quantification).
Objective (ELBO). With approximate posterior $q_\phi(z \mid x)$ and likelihood $p_\theta(x \mid z)$,

$$\max_{\phi,\theta} \ \mathbb{E}_x \Big[\mathbb{E}_{q_\phi(z|x)} \log p_\theta(x \mid z) - \mathrm{KL}\big(q_\phi(z \mid x) \,\|\, p(z)\big) \Big]. \tag{4}$$

Use the reparameterization trick $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$ ($\epsilon \sim \mathcal{N}(0, I)$) for low-variance gradients. **Remark.** Provides a principled generative model with controllable latent priors, but trade-offs exist between reconstruction fidelity and prior regularization.

Adversarial Autoencoder (AAE) / Adversarially Regularized AE

Idea. Use adversarial training (à la GAN) to match the aggregated posterior $q(z) = \mathbb{E}_x q_\phi(z \mid x)$ to a chosen prior $p(z)$, replacing or augmenting explicit KL penalties.
Objective. Minimize reconstruction loss plus an adversarial game: train a discriminator D to distinguish samples from $q(z)$ and $p(z)$, while training the encoder f_ϕ to fool D. Schematic objective

$$\min_{\phi,\theta} \frac{1}{n} \sum_i \ell\big(x_i, g_\theta(f_\phi(x_i))\big) \quad \text{and} \quad \min_\phi \max_D \mathbb{E}_{z\sim p(z)}[\log D(z)] + \mathbb{E}_x[\log(1 - D(f_\phi(x)))]. \tag{5}$$

Remark. Enables flexible prior shaping (e.g. multimodal priors) and often sharper generative samples, but introduces adversarial instability and hyperparameter sensitivity.

2.5 Limitations

Below we list several common limitations of standard autoencoders, each followed by a concise remark or possible mitigation.

1. **Focus on global reconstruction.** Standard losses (e.g. MSE) measure per-pixel error and therefore tend to ignore perceptual or structural quality, often producing overly smooth or blurry reconstructions.
2. **Risk of learning the identity mapping.** A sufficiently large encoder–decoder can simply copy input to output, yielding trivial representations.
3. **Unstructured latent space.** The learned latent codes may be entangled, non-identifiable, or poorly organized for downstream tasks.
4. **Sensitivity to outliers and noise.** Squared-error objectives are strongly influenced by outliers and heavy-tailed noise, which can degrade the learned representation.
5. **Difficulty with very high-dimensional or structured data.** For complex modalities (high-resolution images, long time series) naive dense architectures are inefficient and hard to train.

3 Variational Autoencoder

3.1 Motivation

As mentioned above, although standard autoencoders can learn compressed latent codes, they have several limitations in terms of probabilistic modeling and generation:

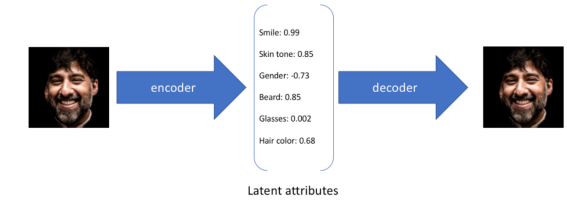
1. Autoencoders lack a clear probabilistic interpretation.
2. The latent space may be irregular and unstructured, which hurts interpretability and makes sampling unreliable.
3. Deterministic autoencoders cannot reliably generate realistic new samples beyond the training set because their latent representations are not constrained to follow a smooth prior.

To address these issues we seek a model that

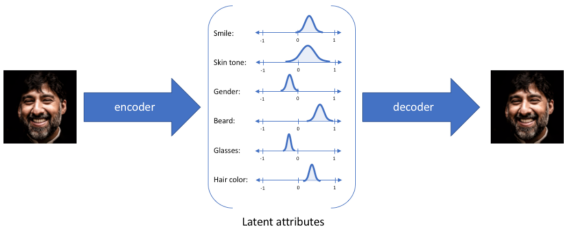
1. Provides an explicit probabilistic model of the data,
2. Induces a smooth, well-structured latent space where sampling is meaningful,
3. Can generate new data samples in a principled way.

The *Variational Autoencoder* (VAE) Kingma and Welling [2013], Doersch [2016] meets these goals by treating the latent variable z as drawn from a prior $p(z)$ (commonly $\mathcal{N}(0, I)$). The encoder is interpreted as an inference network that produces an approximate posterior $q_\phi(z \mid x)$, while the decoder defines a generative likelihood $p_\theta(x \mid z)$. Training maximizes a variational lower bound (ELBO) on the marginal likelihood:

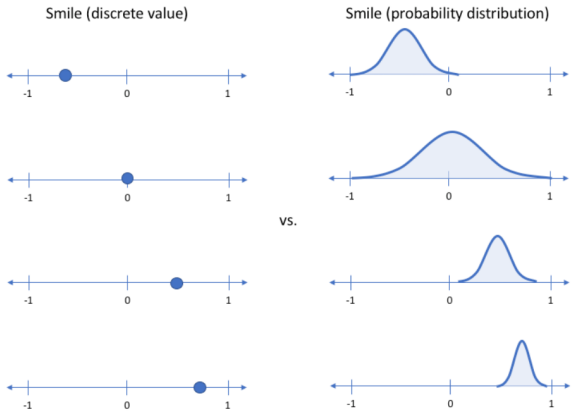
$$\log p_\theta(x) \;\geq\; \mathbb{E}_{z\sim q}\big[\log p_\theta(x \mid z)\big] - \mathrm{KL}\big(q_\phi(z) \,\|\, p(z|x)\big). \tag{6}$$



(a) AE reconstruction



(b) VAE reconstruction



(c) AE latent vs. VAE latent

Figure 1: Comparison of deterministic autoencoders and variational autoencoders.

Practically, the reparameterization trick (e.g. $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon,\ \epsilon \sim \mathcal{N}(0, I)$) provides low-variance gradients for end-to-end training.

Figure 1 illustrates the contrast between autoencoders and VAEs where the latent space of former is a vector while the latter one is in a perspective of probability.

Even though both models can reconstruct inputs, a traditional autoencoder produces a deterministic latent vector, while a VAE learns a distributional (probabilistic) latent representation. This probabilistic latent space is smooth and structured, which enables principled sampling and the generation of meaningful new data.

3.2 Variational Inference

Variational inference (VI) is the core approximate-inference method underlying the VAE. We briefly present the standard derivation and the Evidence Lower Bound (ELBO).

Let $p(z)$ be a prior on latent variables and $p_\theta(x \mid z)$ the likelihood (decoder). Bayes' rule gives the exact posterior

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(x)} = \frac{p_\theta(x \mid z)\, p(z)}{\int p_\theta(x \mid z)\, p(z)\, \mathrm{d}z}, \tag{7}$$

but the marginal likelihood $p_\theta(x)$ is typically intractable. Variational inference approximates the intractable posterior by a parametric family $q_\phi(z)$ (the inference). The approximation quality is measured by the Kullback–Leibler divergence

$$\mathrm{D}_{\mathrm{KL}}\big(q_\phi(z) \,\|\, p_\theta(z \mid x)\big) = \mathbb{E}_{z \sim q_\phi(\cdot \mid x)}\left[\log \frac{q_\phi(z)}{p_\theta(z \mid x)}\right]. \tag{8}$$

Using the identity $\log p_\theta(x) = \log \frac{p_\theta(x,z)}{p_\theta(z|x)}$ and taking expectations under $q_\phi(z)$ yields the standard decomposition

$$\begin{aligned} \log p_\theta(x) &= \mathbb{E}_{z \sim q}\big[\log p_\theta(x, z) - \log q_\phi(z)\big] + \mathrm{D}_{\mathrm{KL}}\big(q_\phi(z) \,\|\, p_\theta(z \mid x)\big) \\ &= \mathcal{L}(\theta, \phi; x) + \mathrm{D}_{\mathrm{KL}}\big(q_\phi(z) \,\|\, p_\theta(z \mid x)\big), \end{aligned} \tag{9}$$

where we define the *Evidence Lower Bound* (ELBO)

$$\mathcal{L}(\theta, \phi; x) := \mathbb{E}_{z \sim q}\big[\log p_\theta(x, z) - \log q_\phi(z)\big]. \tag{10}$$

Because the KL term on the right-hand side of (9) is nonnegative, $\mathcal{L}(\theta, \phi; x)$ is indeed a lower bound on the log marginal likelihood $\log p_\theta(x)$. Maximizing the ELBO with respect to (θ, ϕ) is therefore equivalent to minimizing the KL divergence between $q_\phi(z)$ and the true posterior $p_\theta(z \mid x)$.

It is common and instructive to rewrite the ELBO in the more explicit form

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z\sim q}\big[\log p_\theta(x \mid z)\big] - \mathrm{D}_{\mathrm{KL}}\big(q_\phi(z) \,\|\, p(z)\big), \tag{11}$$

which separates a reconstruction (expected log-likelihood) term and a regularizer that encourages the approximate posterior to stay close to the prior $p(z)$.

3.3 Connect VI to VAE

So how does variational inference connect to the Variational Autoencoder? The whole autoencoder can be viewed as a probabilistic model: the marginal data distribution is $p(x)$, the encoder corresponds to the approximate posterior $q_\phi(z \mid x)$, the prior is specified in the latent space $p(z)$, and the decoder defines the likelihood $p_\theta(x \mid z)$. Maximizing the ELBO (or equivalently minimizing its negative) yields the VAE training objective. Concretely,

$$\begin{aligned}
\text{ELBO} &= \mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x \mid z) + \log p(z) - \log q_\phi(z \mid x)\big] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z \mid x)}{p(z)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] - D_{KL}\big(q_\phi(z \mid x) \,\|\, p(z)\big),
\end{aligned} \tag{12}$$

The first term in (12) is the expected log-likelihood (reconstruction term), which measures how well samples from the approximate posterior reconstruct x under the decoder. The second term is the KL divergence between the approximate posterior and the prior; it acts as a regularizer that encourages the learned posterior to stay close to the chosen prior and thus induces a smooth, structured latent space.

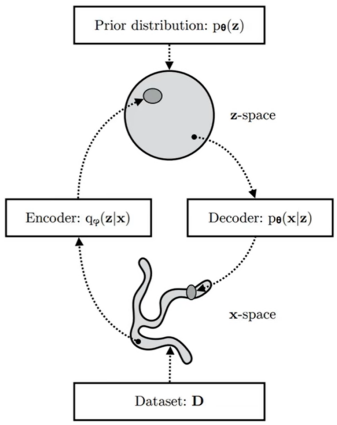


Figure 2: VAE structure

3.4 Reparamteriztion trick

Considering the Gaussian distribution of the model. Suppose the Encoder, latent prior, dimension of latent is:

$$q_\phi(z|x) = N(z; \mu_\phi, \Sigma_\phi), p(z) = N(z, 0, I), k = dim(z) \tag{13}$$

The KL divergence in VAE is:

$$D(N(\mu_\phi, \Sigma_\phi)||N(0, I))) = \frac{1}{2}(\text{tr}(\Sigma_\phi) + \mu_\phi^T \mu_\phi - k - \log\det(\Sigma_\mu)) \tag{14}$$

Then suppose the decoder part is:

$$p_\theta(x|z) = N(x; \mu_\theta(z), \sigma^2, \log p_\theta(x|z) = -c||x - \mu_\theta(z)||^2 + d \tag{15}$$

Thus given the consumption of Gaussian distribution, the reconstruction likelihood seems like MSE to evaluate the reconstruction quality.

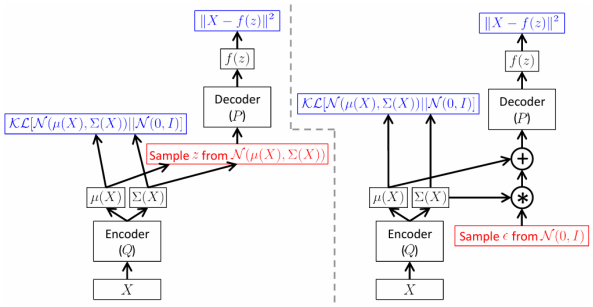


Figure 3: reparameterization

4 Functional Data Analysis & Functional Principal Component Analysis

4.1 Functional Data Analysis

In functional data analysis (FDA), each statistical unit is viewed not as a scalar or a finite–dimensional vector, but as a whole function

$$X_i(t), \qquad i = 1, \ldots, n,$$

defined on a continuous domain $t \in [a, b]$. Thus a data set is a sample of n curves (or trajectories) $\{X_1(t), \ldots, X_n(t)\}$.

In applications we do not observe the entire function continuously. Instead, for each subject i we record noisy measurements at a finite set of time points t_{ij},

$$Y_i(t_{ij}) = X_i(t_{ij}) + \varepsilon_{ij}, \tag{16}$$

where ε_{ij} denotes measurement error. FDA provides tools to reconstruct and analyse the underlying smooth functions $X_i(t)$ from these discrete and noisy observations.

Compared with multivariate data, where a single observation is a vector $x_i \in \mathbb{R}^p$ and the full data set is an $n \times p$ matrix, a functional observation is an entire curve $X_i(t)$ and the sample consists of a set of n curves.

A natural idea is to stack the sampled values into a high–dimensional vector, e.g.

$$x_i = \big(X_i(t_1), \ldots, X_i(t_m)\big)^\top,$$

and then apply standard multivariate methods. However, this approach faces several difficulties:

- **Curse of dimensionality.** The number of sampling points m is often large compared with the sample size n ($m \gg n$). Classical multivariate methods may perform poorly in such high dimensions.
- **Smoothness.** Consecutive evaluations $X_i(t_j)$ and $X_i(t_{j+1})$ are strongly correlated because the underlying function is smooth. Treating these values as unrelated coordinates ignores this structure, whereas FDA makes explicit use of it.
- **Irregular or sparse sampling.** In many studies, the observation times $\{t_{ij}\}$ can differ from one subject to another or be very sparse. In such cases it may not even be possible to form a common vector representation, while FDA methods are designed to handle irregular grids.

To analyze functional data, it is convenient to approximate each function $X_i(t)$ by an expansion in a set of known basis functions $\{\psi_k(t)\}_{k=1}^K$:

$$X_i(t) \approx \sum_{k=1}^{K} c_{ik}\, \psi_k(t). \tag{17}$$

Here $\psi_k(t)$ denotes a pre–chosen basis function (for example, B–spline basis functions or Fourier basis functions), and c_{ik} is the coefficient or "score" of the k-th basis function for the i-th observation.

In this way, each curve is represented by a finite vector of coefficients $(c_{i1}, \ldots, c_{iK})$, which captures the main features of the smooth trajectory while respecting its functional nature. Subsequent statistical analyses (e.g. regression, classification, clustering) can then be carried out using these coefficient vectors.

4.2 Functional Principal Component Analysis

FPCARamsay and Silverman [2005] is the analogue of standard principal component analysis for functional data. Starting from the sample of curves $\{X_1(t), \ldots, X_n(t)\}$, we first define the empirical mean and covariance functions

$$\bar{x}(t) = \frac{1}{n}\sum_{i=1}^{n} X_i(t), \tag{18}$$

$$\hat{\sigma}(s,t) = \frac{1}{n}\sum_{i=1}^{n} \big(X_i(s) - \bar{x}(s)\big)\big(X_i(t) - \bar{x}(t)\big). \tag{19}$$

(If desired, a corresponding correlation function can be obtained by standardising $\hat{\sigma}(s,t)$.

The functional principal components are defined as the eigenfunctions of the covariance operator. They solve the integral equation

$$\int_a^b \hat{\sigma}(s,t)\, w_k(t)\, dt = \lambda_k\, w_k(s), \qquad k = 1, 2, \ldots, \tag{20}$$

where λ_k is the kth eigenvalue and $w_k(t)$ the corresponding eigenfunction (functional principal component, FPC). The eigenfunctions are usually ordered so that $\lambda_1 \geq \lambda_2 \geq \cdots$.

Each centred curve $X_i(t) - \bar{x}(t)$ can then be expanded in this orthonormal system,

$$X_i(t) - \bar{x}(t) \approx \sum_{k=1}^{K} s_{ik}\, w_k(t), \tag{21}$$

where

$$s_{ik} = \int_a^b \big(X_i(t) - \bar{x}(t)\big)\, w_k(t)\, dt \tag{22}$$

is the kth functional principal component score of the ith subject. The first few FPCs typically capture most of the variability in the sample.

In practice we represent the centred curves by a finite basis expansion. Let $\phi(t) = (\phi_1(t), \ldots, \phi_J(t))^\top$ be a vector of known basis functions (e.g. Fourier or B–spline basis). For each i,

$$X_i(t) - \bar{x}(t) \approx \sum_{j=1}^{J} c_{ij}\, \phi_j(t), \tag{23}$$

or, in vector form,

$$x_i(t) = C_i \phi(t), \qquad x(t) = \begin{pmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{pmatrix}, \tag{24}$$

where C is the $n \times J$ coefficient matrix with ith row C_i.

The sample covariance function can be written as

$$\hat{\sigma}(s,t) = \frac{1}{n} x(s)^\top x(t) = \frac{1}{n} \phi(s)^\top C^\top C\, \phi(t). \tag{25}$$

We also represent each eigenfunction in the same basis,

$$w_k(t) = \phi(t)^\top b_k, \tag{26}$$

where b_k is a J-dimensional coefficient vector. Substituting these expressions into the eigen-equation yields

$$\frac{1}{n} C^\top C\, W\, b_k = \lambda_k b_k, \tag{27}$$

subject to the normalisation constraint

$$\int_a^b \{w_k(t)\}^2 dt = b_k^\top W b_k = 1, \tag{28}$$

where

$$W = \int_a^b \phi(t)\,\phi(t)^\top dt \tag{29}$$

is the $J \times J$ Gram matrix of the basis functions. If an orthonormal basis is used, then $W = I_J$ and the problem reduces to the usual matrix eigenproblem

$$\frac{1}{n} C^\top C\, b_k = \lambda_k b_k. \tag{30}$$

Once the eigenvectors b_k have been obtained, the corresponding FPCs are $w_k(t) = \phi(t)^\top b_k$ and the scores are

$$s_{ik} = \int_a^b x_i(t) w_k(t)\, dt = C_i W b_k, \tag{31}$$

(or simply $s_{ik} = C_i b_k$ in the orthonormal basis case). Keeping the first K components gives a low–dimensional representation of the functional data that preserves most of the variation.

5 Functional Antoencoders

5.1 Motivation & Main Idea

Linear methods for representation learning, such as **FPCA**, are widely used and often serve as a natural baseline. However, this assumption can be overly restrictive in practice. Real-world functional data typically exhibit complex patterns such as phase variation, local warping, sharp regime changes, and heterogeneous smoothness across time, all of which induce *nonlinear* structure in the underlying function space. Moreover, different functional coordinates are often coupled through nonlinear dynamics, so that both the marginal behavior of each dimension and the cross-dimensional dependence cannot be adequately captured by linear correlations.

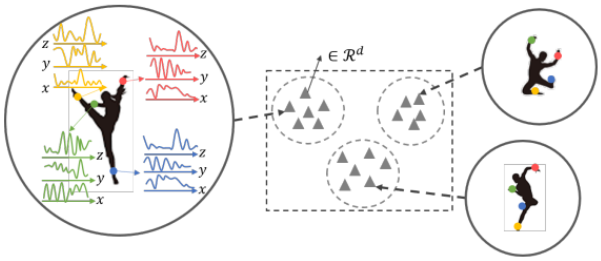


Figure 4: functional data for motion

These observations motivate the use of *Functional Autoencoders* (FAEs)Hsieh et al. [2021]Zhong et al. [2023]Wu et al. [2024]. FAEs extend classical autoencoders to the functional setting by replacing scalar weights with weight functions and by using functional inner products as the basic building blocks. The encoder learns a low-dimensional nonlinear representation of the input trajectories, while the decoder reconstructs the original functions from this latent code. Because the encoder and decoder are composed with nonlinear activation functions, FAEs are able to approximate complicated nonlinear manifolds of functional data and to model nonlinear interactions among multiple functional coordinates, going well beyond what FPCA and other linear methods can express.

5.2 Basic Structure

There are some notations:

1. $\mathbf{x} = [x_1(t), x_2(t), \ldots, x_P(t)] \in \mathcal{H}^P$ where $\mathcal{H}$ denotes the function space, a subspace of L^2.$x_j \in \mathcal{H}^P$ is a function.
2. $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is n samples of functional data.
3. $\mathbf{y} = [\mathbf{y}_1, \ldots, \mathbf{y}_d] \in \mathbb{R}^d$ denotes d dimension of encoding.
4. $\phi : \mathcal{H}^P \to \mathbb{R}^d$ is the encoder where $\mathbb{R}^d$ is a vector space.

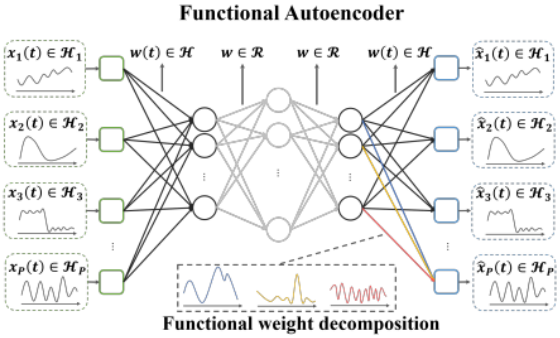


Figure 5: Functional Autoencoder structure

5. $\psi : \mathbb{R}^d \to \mathcal{H}^P$ is the decoder.

Similar to autoencoders, the goal is to minimize MSE. i.e.

$$\phi, \psi = \arg\min_{\phi,\psi} \|\mathbf{x} - (\psi \circ \phi)\mathbf{x}\|^2.$$

Unlike the autoencoder, the first hidden layer and the last hidden layer is not scalar input/output but functional input/output.

We consider multi-dimensional functional observations $\mathbf{x}$ and encode them into a low-dimensional vector $\mathbf{y}$ by replacing the scalar weights and inner products in a standard autoencoder with *functional* weights $w \in \mathcal{H}$ and the L^2 inner product for real-valued functions. The activation of the k-th unit in the first hidden layer, denoted by $O_k^{(1)}$, is given by

$$O_k^{(1)} = \sigma\left(\sum_{j=1}^{P} \langle x_j(t), w_{k,j}^{(1)}(t)\rangle\right) = \sigma\left(\sum_{j=1}^{P} \int_{\tau_j} x_j(t)\, w_{k,j}^{(1)}(t)\, dt\right), \tag{32}$$

where τ_j is the domain of $x_j(t)$ and $\sigma(\cdot)$ is a nonlinear activation function. The framework can work with common choices such as linear, `tanh` and ReLU, and can be naturally extended to deeper architectures ($l > 1$) in order to capture more complex nonlinear encodings of the input functions.

The decoder maps the finite-dimensional code back to the space of functional observations using the functional weights. In particular, the output units $O_j^{(l)}(t) \in \mathcal{H}_j$, $j = 1, \ldots, P$, are computed as

$$\hat{x}_j(t) = O_j^{(l)}(t) = \sum_k O_k^{(l-1)}\, w_{j,k}^{(l-1)}(t). \tag{33}$$

We employ a linear activation at the output layer. After training, the learned functional weights of the FAE provide a decomposition of the input functions and reveal dominant modes of variation present in the data.

Given n functional samples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the FAE is trained by minimizing the reconstruction error between the input and its reconstruction. The objective function is

$$\mathcal{L}(\Omega) = \frac{1}{2n} \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \tag{34}$$

$$= \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{P} \|x_{i,j}(t) - \hat{x}_{i,j}(t)\|_2^2 \tag{35}$$

$$= \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{P} \int_{\tau_j} \big(x_{i,j}(t) - \hat{x}_{i,j}(t)\big)^2 dt, \tag{36}$$

where Ω denotes the collection of all functional weights in the network.

5.3 Discrete Construction

In the continuous-time functional autoencoder (FAE), the first hidden layer is defined as

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

where $X(t)$ is the input function, $w_k^{(I)}(t)$ is a functional weight, and g is an activation function.

In practice, we only observe the function on a discrete grid

$$\{t_1, \ldots, t_J\},$$

with numerical integration weights ω_j. The integral is then approximated by a numerical sum:

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

The input to the network can be written as the vector

$$\big(X(t_1), \ldots, X(t_J)\big) \in \mathbb{R}^J.$$

To make the functional weights learnable, we represent each $w_k^{(I)}$ by a basis expansion,

$$w_k^{(I)}(t) = \sum_{m=1}^{M^{(I)}} c_{mk}^{(I)}\, \phi_m^{(I)}(t),$$

where $\{\phi_m^{(I)}(t)\}$ is a fixed set of basis functions (e.g., B-splines), and $c_{mk}^{(I)}$ are trainable coefficients. On the discrete grid this becomes

$$w_k^{(I)}(t_j) = \sum_{m=1}^{M^{(I)}} c_{mk}^{(I)}\, \phi_m^{(I)}(t_j).$$

Plugging this expansion into the discrete encoder gives

$$h_k^{(1)} = g\Big(\sum_{j=1}^{J} \omega_j X(t_j) \sum_{m=1}^{M^{(I)}} c_{mk}^{(I)}\, \phi_m^{(I)}(t_j)\Big) = g\Big(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} \sum_{j=1}^{J} \omega_j X(t_j)\, \phi_m^{(I)}(t_j)\Big).$$

Define

$$f_m = \sum_{j=1}^{J} \omega_j X(t_j)\, \phi_m^{(I)}(t_j), \quad m = 1, \ldots, M^{(I)},$$

and call $(f_1, \ldots, f_{M^{(I)}})$ the *feature layer*. Then

$$h_k^{(1)} = g\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)}\, f_m\right),$$

which is just a standard fully connected layer mapping the feature layer into the first hidden layer. The mapping from the raw samples $\{X(t_j)\}$ to the feature layer $\{f_m\}$ is fixed and contains no trainable parameters; all learnable parameters in the encoder appear in $c_{mk}^{(I)}$ and the subsequent hidden layers.

Different curves X_i may be observed at different time points $\{t_{ij}\}_{j=1}^{J_i}$, leading to irregular grids. In this case, the feature layer for curve i is defined by

$$f_{im} = \sum_{j=1}^{J_i} \omega_{ij}\, X_i(t_{ij})\, \phi_m^{(I)}(t_{ij}), \quad m = 1, \ldots, M^{(I)}.$$

Thus each curve yields a feature vector

$$f_i = (f_{i1}, \ldots, f_{iM^{(I)}})^\top \in \mathbb{R}^{M^{(I)}},$$

independent of the particular observation times. The rest of the network only sees these fixed-length vectors and does not need to handle irregular sampling explicitly.

On the decoder side, we use another basis expansion for the output weight functions,

$$w_k^{(O)}(t) = \sum_{m=1}^{M^{(O)}} c_{mk}^{(O)}\, \phi_m^{(O)}(t).$$

The discrete reconstruction at time point t_j can be written as

$$\hat{X}(t_j) = \sum_{k=1}^{K^{(L)}} h_k^{(L)}\, w_k^{(O)}(t_j) = \sum_{m=1}^{M^{(O)}} b_m\, \phi_m^{(O)}(t_j),$$

where

$$b_m = \sum_{k=1}^{K^{(L)}} h_k^{(L)}\, c_{mk}^{(O)}.$$

The vector $b = (b_1, \ldots, b_{M^{(O)}})^\top$ defines the *coefficient layer*, which parameterizes the reconstructed function in the output basis. Because the reconstruction is a linear combination of smooth basis functions, the output is a smooth function that can be evaluated at arbitrary time points.

Overall, the discrete FAE:

- works directly with noisy discrete samples,
- naturally handles irregular observation grids via the feature layer,
- produces smooth functional reconstructions via the coefficient layer,
- and yields low-dimensional representations in the hidden layers for downstream tasks.

5.4 Functional Derivatives*

In a functional autoencoder (FAE) the trainable parameters are no longer scalar weights but *weight functions*. Consequently, the usual finite-dimensional gradient used in backpropagation must be replaced by a gradient in function space, i.e. a *functional derivative*. In this subsection we briefly review the basic notion of a functional derivative and then derive the gradients used to train the FAE.

Functionals and their gradients. Let f be a real-valued function and consider the functional

$$J[f] = \int_b^a L\big(x, f(x), f'(x)\big)\, dx, \tag{37}$$

where L is a differentiable function and $f'(x) = df(x)/dx$. The first variation of J at f in the direction of a perturbation η is defined by

$$\delta J[f;\eta] = \frac{d}{d\varepsilon} J[f + \varepsilon\eta]\bigg|_{\varepsilon=0}. \tag{38}$$

If there exists a function $\nabla J[f](x)$ such that

$$\delta J[f;\eta] = \int_b^a \nabla J[f](x)\, \eta(x)\, dx \quad \text{for all admissible } \eta, \tag{39}$$

then $\nabla J[f]$ is called the *gradient* (or functional derivative) of J at f.

A classical result from the calculus of variations states that

$$\nabla J[f] = \frac{\partial L(x, f, f')}{\partial f} - \frac{d}{dx}\left(\frac{\partial L(x, f, f')}{\partial f'}\right). \tag{40}$$

If L does not depend on f', i.e. $L(x, f, f') = g(f(x), x)$, then the second term in (40) vanishes and we simply have

$$\nabla J[f] = \frac{\partial L(x, f)}{\partial f}. \tag{41}$$

As a simple example, for constants a, b,

$$\frac{\partial\, a f(x)^b}{\partial f(x)} = b a f(x)^{b-1}. \tag{42}$$

A basic rule for inner products. A special case that will be used repeatedly in the FAE is the functional

$$I[w] = \int x(t)\, w(t)\, dt, \tag{43}$$

where x is fixed and w is the variable function. Perturbing $w_\varepsilon(t) = w(t) + \varepsilon\eta(t)$ and differentiating with respect to ε yields

$$I[w_\varepsilon] = I[w] + \varepsilon \int x(t)\eta(t)\, dt + o(\varepsilon). \tag{44}$$

Hence the functional derivative of I with respect to w is

$$\frac{\delta I}{\delta w(t)} = x(t), \tag{45}$$

which may be regarded as the continuous analogue of $\partial \sum_i x_i w_i / \partial w_j = x_j$.

Output layer: functional chain rule. In the FAE of Hsieh et al. [2021] the output layer reconstructs each functional coordinate via

$$O_j^{(2)}(t) = \sum_k O_k^{(1)}\, w_{j,k}^{(2)}(t), \tag{46}$$

where $O_k^{(1)}$ is the k-th hidden activation (a scalar) and $w_{j,k}^{(2)}(t)$ is a decoding weight function. For a single sample, the reconstruction loss is

$$\mathcal{L} = \frac{1}{2} \sum_{j=1}^{P} \int_{\tau_j} \big(x_j(t) - O_j^{(2)}(t)\big)^2 dt, \tag{47}$$

so that, using the simplification above (no dependence on derivatives of $O_j^{(2)}$),

$$\frac{\partial \mathcal{L}}{\partial O_j^{(2)}(t)} = -\big(x_j(t) - O_j^{(2)}(t)\big) =: \delta_j^{(2)}(t). \tag{48}$$

Applying the (functional) chain rule to the weight function $w_{j,k}^{(2)}(t)$ gives

$$\frac{\partial \mathcal{L}}{\partial w_{j,k}^{(2)}(t)} = \frac{\partial \mathcal{L}}{\partial O_j^{(2)}(t)} \frac{\partial O_j^{(2)}(t)}{\partial w_{j,k}^{(2)}(t)}. \tag{49}$$

Because $O_j^{(2)}(t)$ depends linearly on $w_{j,k}^{(2)}(t)$,

$$\frac{\partial O_j^{(2)}(t)}{\partial w_{j,k}^{(2)}(t)} = O_k^{(1)}, \tag{50}$$

and thus, for a single sample,

$$\frac{\partial \mathcal{L}}{\partial w_{j,k}^{(2)}(t)} = \delta_j^{(2)}(t)\, O_k^{(1)}. \tag{51}$$

Averaging over n samples yields equation (2.14) in Hsieh et al. [2021].

Hidden layer: functional backpropagation. The hidden activation of unit k is

$$O_k^{(1)} = \sigma(h_k), \qquad h_k = \sum_{j=1}^{P} \int x_j(s)\, w_{k,j}^{(1)}(s)\, ds, \tag{52}$$

where $w_{k,j}^{(1)}(s)$ is an encoding weight function and $\sigma(\cdot)$ is a scalar activation function. The error signal propagated from the output layer back to hidden unit k is defined as

$$\delta_k^{(1)} = \sum_{j=1}^{P} \int \delta_j^{(2)}(t)\, w_{j,k}^{(2)}(t)\, dt, \tag{53}$$

which is an inner product over t between the output error function and the corresponding decoding weight.

Next we differentiate $O_k^{(1)}$ with respect to the functional weight $w_{k,j}^{(1)}(t)$. By the scalar chain rule and the inner-product rule above,

$$\frac{\partial O_k^{(1)}}{\partial w_{k,j}^{(1)}(t)} = \sigma'(h_k)\, x_j(t). \tag{54}$$

Combining these pieces via the functional chain rule,

$$\frac{\partial \mathcal{L}}{\partial w_{k,j}^{(1)}(t)} = \frac{\partial \mathcal{L}}{\partial O_k^{(1)}} \frac{\partial O_k^{(1)}}{\partial w_{k,j}^{(1)}(t)} = \delta_k^{(1)}\, \sigma'(h_k)\, x_j(t) \tag{55}$$

for a single sample. For n samples, the gradient is the average

$$\frac{\partial \mathcal{L}}{\partial w_{k,j}^{(1)}(t)} = \frac{1}{n} \sum_{i=1}^{n} \delta_{i,k}^{(1)}\, \sigma'(h_{i,k})\, x_{i,j}(t), \tag{56}$$

which corresponds to equation (2.15) in Hsieh et al. [2021]. This shows that backpropagation in the FAE has exactly the same structure as in a standard autoencoder, with sums replaced by integrals and scalar weights replaced by weight functions.

5.5 Connections to FPCA and Autoencoders

Consider now a linear FAE with a single hidden layer. For each training function $X_i(t)$, the encoder produces

$$z_{ik} = \int_T X_i(t)\, w_k^{(I)}(t)\, dt,$$

and the decoder reconstructs

$$\hat{X}_i(t) = \sum_{k=1}^{K} z_{ik}\, w_k^{(O)}(t).$$

The reconstruction loss is

$$L = \sum_{i=1}^{N_{\text{train}}} \int_T \big(X_i(t) - \hat{X}_i(t)\big)^2 dt.$$

If we impose orthonormality constraints on the functional weights,

$$\langle w_k^{(\cdot)}, w_\ell^{(\cdot)} \rangle = \delta_{k\ell},$$

then minimizing L is equivalent to finding the same functional principal subspace as in functional principal component analysis (FPCA). In this sense, FPCA is a special case of FAE: a single-layer, linear, orthogonality-constrained functional autoencoder.

In practice, functions are observed at discrete time points $\{t_1, \ldots, t_J\}$ with numerical weights ω_j, so the integral in the encoder is approximated by

$$\int_T X_i(t)\, w_k^{(I)}(t)\, dt \approx \sum_{j=1}^{J} \omega_j\, X_i(t_j)\, w_k^{(I)}(t_j),$$

and the loss becomes

$$L \approx \sum_{i=1}^{N_{\text{train}}} \frac{1}{J} \sum_{j=1}^{J} \big(X_i(t_j) - \hat{X}_i(t_j)\big)^2.$$

As J grows and the grid becomes dense, this discrete objective converges to the continuous one, and the learned weights approximate the FPCA eigenfunctions.

Compared to a classical dense autoencoder (AE), the FAE can be viewed as an AE with functional structure hard-coded at its input and output. A standard AE treats the input and output as generic vectors and makes all weights trainable. The FAE instead:

- uses a deterministic feature layer based on numerical integration and input basis functions,

- uses a deterministic reconstruction from the coefficient layer through smooth output basis functions,
- and places trainable weights only between the feature layer and the coefficient layer.

This design yields several practical advantages over a vanilla AE:

- faster convergence and better generalization, as temporal structure is summarized in the feature layer before entering the trainable network;
- smooth functional outputs, since reconstruction is a linear combination of smooth basis functions;
- natural handling of irregular observation grids, because the feature layer directly incorporates numerical integration on the observed time points, whereas a vanilla AE must treat missing time points explicitly.

6 Experiment

In this section we compare functional PCA (FPCA) and the functional autoencoder (FAE) on a simple univariate functional data example and on several benchmark datasets. Our aim is to illustrate how linear and nonlinear functional representations differ in terms of reconstruction accuracy and clustering performance.

6.1 Methods

We compare FPCA and the FAE using the same latent dimension d.

FPCA. Following Section 4.2, we treat the one-dimensional trajectories as elements of the Hilbert space $H = L^2([0,1])$ and perform FPCA using a B-spline basis on $[0,1]$. The eigenfunctions are estimated from the sample covariance operator, and we retain the first K functional principal components so that at least 90% of the total variance is explained. Reconstruction is obtained by projecting each centred curve onto these components and truncating the corresponding Karhunen–Loève expansion.

Functional autoencoder. The FAE follows the architecture in Section 5. In the present univariate setting ($P = 1$), the encoder maps each functional input $x \in H$ to a d-dimensional latent code by replacing scalar weights with weight functions and inner products with L^2 inner products. The decoder maps the latent code back to H. We use one hidden layer with 16 units and $\tanh$ activation, and a linear activation in the output layer. All functional weights are initialised randomly and trained jointly by minimising the reconstruction loss in equation (36) using Adam. In practice, integrals are evaluated on the observation grid and the L^2 inner products are approximated by Riemann sums.

To ensure a fair comparison, we set the latent dimension of the FAE equal to the number of retained FPCA components, that is $d = K$.

6.2 Reconstruction performance

We first study reconstruction accuracy on simulated univariate functional data.

For each method we measure the reconstruction error of the i-th curve by the integrated squared error

$$\mathrm{MSE}_i = \int_0^1 \left\{x_i(t) - \hat{x}_i(t)\right\}^2 dt,$$

where $\hat{x}_i(t)$ denotes the reconstructed trajectory. Numerically, the integral is approximated by a sum over the observation grid.

6.2.1 Data generation

We consider univariate functional observations on the time interval $\mathcal{T} = [0,1]$. For each subject $i = 1, \ldots, n$ we observe a trajectory

$$x_i(t), \qquad t \in \mathcal{T},$$

on an equidistant grid of m time points. Following Zhong et al. [2023], we generate data under five different scenarios in order to compare the performance of FPCA and the FAE. Specifically,

- Case 1: $X_i(t) = \xi_{i1}\sin(2\pi t) + \xi_{i2}\cos(2\pi t),\ t \in \mathcal{T}$, where ξ_{i1}'s and ξ_{i2}'s are simulated from $\mathcal{N}(0, 3^2)$ and $\mathcal{N}(0, 2^2)$, respectively.

- Case 2: $X_i(t) = \xi_{i2}\sin(\xi_{i1}t),\ t \in \mathcal{T}$, where both ξ_{i1}'s and ξ_{i2}'s are simulated from $\mathcal{N}(0, 2^2)$.
- Case 3: $X_i(t) = \xi_{i2}\cos(\xi_{i1}t),\ t \in \mathcal{T}$, where both ξ_{i1}'s and ξ_{i2}'s are simulated in the same way as in Case 2.
- Case 4: $X_i(t) = \xi_{i1}\sin(2\pi t) + \xi_{i2}\cos(2\pi t) + \xi_{i2}\sin(\xi_{i1}t),\ t \in \mathcal{T}$, where both ξ_{i1}'s and ξ_{i2}'s are simulated in the same way as in Case 2.
- Case 5: $X_i(t) = \xi_{i1}\sin(2\pi t) + \xi_{i2}\cos(2\pi t) + \xi_{i2}\cos(\xi_{i1}t),\ t \in \mathcal{T}$, where both ξ_{i1}'s and ξ_{i2}'s are simulated in the same way as in Case 2.

6.2.2 Reconstruction results

Table 2: Reconstruction loss comparison (MSE)

Case	FAE_Loss	FPCA_Loss	Winner
1	1.190571 ± 0.493171	10.007184 ± 0.450656	FAE
2	0.890170 ± 0.569480	1.436265 ± 0.294106	FAE
3	0.944637 ± 0.435328	1.456197 ± 0.498155	FAE
4	1.578911 ± 0.495404	9.423863 ± 1.051228	FAE
5	2.181033 ± 0.867744	12.442592 ± 1.524700	FAE

For each case we repeat the experiment five times to obtain stable estimates. Table 2 reports the mean and standard deviation of MSE_i over all curves for FPCA and the FAE. For the same latent dimension ($d = K = 8$ for both the latent space and the number of principal components), the FAE consistently achieves a substantially lower reconstruction error than FPCA.

6.3 Clustering in the latent space

We next assess how well the latent representations separate the underlying classes. For FPCA we apply k-means clustering to the first K functional principal component scores. For the FAE we apply k-means to the d-dimensional latent codes produced by the encoder. Clustering performance is evaluated by classification accuracy (ACC) and the adjusted Rand index (ARI). Formally,

$$\mathrm{ACC} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(\hat{y}_i = y_i), \tag{57}$$

where y_i and $\hat{y}_i$ denote the true and predicted labels, respectively. The ARI is defined as

$$\mathrm{ARI} = \frac{\mathrm{RI} - \mathbb{E}[\mathrm{RI}]}{\max(\mathrm{RI}) - \mathbb{E}[\mathrm{RI}]}, \tag{58}$$

where RI denotes the Rand index.

6.3.1 Data selection and clustering results

We consider several datasets from the UCR time series archiveDarke et al. [2022] and treat each time series as a discretized functional observation.

Table 3: UCR datasets: clustering performance comparison

Dataset	N_{samples}	N_{classes}	FAE_ACC	FPCA_ACC	FAE_ARI	FPCA_ARI
BME	180	3	$\mathbf{0.5222 \pm 0.0737}$	0.4556	$\mathbf{0.1547 \pm 0.0326}$	0.1229
DiatomSizeReduction	322	4	0.7416 ± 0.1202	**0.8975**	0.6021 ± 0.1451	**0.8294**
Plane	210	7	$\mathbf{0.9152 \pm 0.0453}$	0.8333	$\mathbf{0.8540 \pm 0.0391}$	0.8102
Fungi	204	18	$\mathbf{0.8118 \pm 0.0255}$	0.7745	$\mathbf{0.7197 \pm 0.0408}$	0.7126

As summarised in Table 2, the FAE often provides higher ACC and ARI than FPCA, indicating a better separation of the classes in the latent space. This suggests that the nonlinear latent representation learned by the FAE is better aligned with the underlying cluster structure in the functional data.

7 Conclusion

We reviewed classical autoencoders, variational autoencoders, and basic tools from functional data analysis, including functional PCA. Building on these ingredients, we described the functional autoencoder (FAE), which replaces scalar weights with weight functions and employs L^2 inner products so that the network operates directly on functional inputs. Using functional derivatives, we showed how backpropagation can be extended to this infinite-dimensional setting.

A simple numerical experiment demonstrated that, for functional data with nonlinear temporal structure, the FAE achieves lower reconstruction error and yields latent representations that are more suitable for clustering than those obtained by FPCA. This highlights the potential of nonlinear functional representation learning for complex functional data.

8 Further Work

An interesting direction for future work is to extend functional autoencoders so that they simultaneously capture both amplitude and phase variability, for example by jointly learning representations of vertical changes in function values and horizontal time warping within a unified latent functional space.

References

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

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

James O Ramsay and Bernard W Silverman. *Functional data analysis*. Springer, 2005.

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. SIAM, 2021.

Rou Zhong, Chunming Zhang, and Jingxiao Zhang. Nonlinear functional principal component analysis using neural networks. *arXiv preprint arXiv:2306.14388*, 2023.

Sidi Wu, Cédric Beaulac, and Jiguo Cao. Functional autoencoder for smoothing and representation learning. *Statistics and Computing*, 34(6), October 2024. ISSN 0960-3174. doi:10.1007/s11222-024-10501-w. URL `https://doi.org/10.1007/s11222-024-10501-w`.

Philip Darke, Paolo Missier, and Jaume Bacardit. Benchmark time series data sets for pytorch – the torchtime package, 2022. URL `https://arxiv.org/abs/2207.12503`.