Functional Autoencoders for Functional Data Representation Learning

- Hsieh, T.-Y., Sun, Y., Wang, S., & Honavar, V. (2021). Functional Autoencoders for Functional Data Representation Learning.
- Wu, S., Beaulac, C., & Cao, J. (2024). Functional Autoencoder for Smoothing and Representation Learning. Statistics and Computing.

Functional Autoencoders for Functional Data Representation Learning - 2021

Goal: Learning non-linear representation of functional data (recorded at regularly or irregularly spaced points) and functional autoencoder.

- Encoder: Squeezes the data into small, simpler representation (Latent space).
- Decoder: Tries to reconstruct the original data from that smaller representation.

Functional Autoencoder: Takes functional curves, summarizes them by nonlinear transformation learned by a neural network.

Approach: Gradient based algorithm.

Applications: Functional data classification, clustering, compression and functional/scalar response regression.

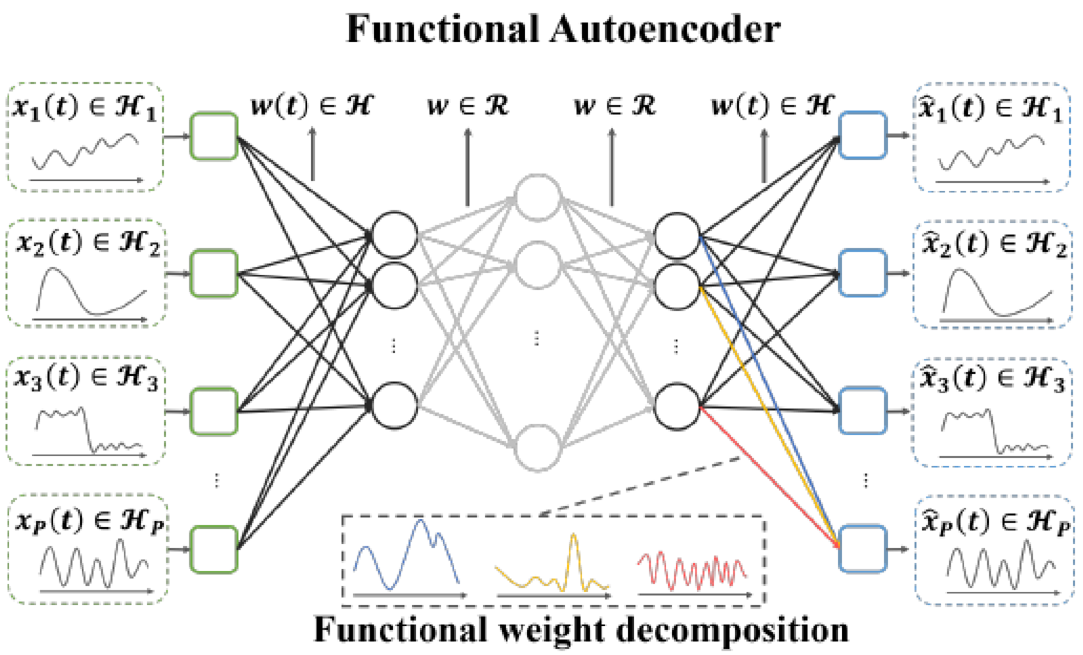
Functional Representation Learning

Problem definition: Given $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ with $\mathbf{x}_i$ as a multidimensional functional data, i.e., $\mathbf{x}_i \in \mathscr{H}^P$, learn a non-linear function f that can encode $\mathbf{x}_i$ as a d-dimensional vector, i.e., $\mathbf{y}_i = f(\mathbf{x}_i)$.

Functional Autoencoder: Functional autoencoder is specified by two mappings, ϕ (encoder) and ψ (decoder), where the encoder is a mapping from a P-dimensional *functional* space to d-dimensional vector space and the decoder maps d-dimensional vector space back to the P-dimensional functional space as follows:

$\phi : \mathscr{H}^P \rightarrow \mathbb{R}^d$ and $\psi : \mathbb{R}^d \rightarrow \mathscr{H}^P$ so as to minimize the reconstruction error:

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


Functional Autoencoder
$x_1(t) \in \mathcal{H}_1$
$x_2(t) \in \mathcal{H}_2$
$x_3(t) \in \mathcal{H}_3$
$x_P(t) \in \mathcal{H}_P$
$w(t) \in \mathcal{H}$
$w \in \mathcal{R}$
$w \in \mathcal{R}$
$w(t) \in \mathcal{H}$
$\hat{x}_1(t) \in \mathcal{H}_1$
$\hat{x}_2(t) \in \mathcal{H}_2$
$\hat{x}_3(t) \in \mathcal{H}_3$
$\hat{x}_P(t) \in \mathcal{H}_P$
Functional weight decomposition

We encode multi-dimensional functional data **x** into **y** by generalizing the weights and scalar inner product in the original autoencoder by functional weights $w \in \mathscr{H}$ and inner product for real functions in L^2 space. The activation of the k-th node in the first hidden layer $O_k^{(1)}$ is given by:

$$O_k^{(1)} = \sigma\left(\sum_{j=1}^{P}\left\langle x_j(t), w_{k,j}^{(1)}(t)\right\rangle\right) = \sigma\left(\sum_{j=1}^{P}\int_{\tau_j} x_j(t) w_{k,j}^{(1)}(t)dt\right) \tag{2.2}$$

The decoder maps the finite dimensional vector encoding of the functional data back to functional space using the functional weights. Specifically, the outputs of the nodes in the output layer $O_j^{(l)}(t)$ in $\mathscr{H}_j$, j = 1, ..., P are calculated by:

$$\hat{x}_j(t) = O_j^{(l)}(t) = \sum_k O_k^{(l-1)} w_{j,k}^{(l-1)}(t). \tag{2.3}$$

Given n samples of functional data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we aim to train the FAE to optimally reconstruct the input. The training objective is given by (Ω is the collection of all functional weights):

$$\begin{aligned}\mathcal{L}(\Omega) &= \frac{1}{2n}\sum_{i=1}^{n}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \\ &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{P}\|x_{i,j}(t) - \hat{x}_{i,j}(t)\|_2^2 \\ &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{P}\int_{\tau_j}\left(x_{i,j}(t) - \hat{x}_{i,j}(t)\right)^2 dt\end{aligned} \tag{2.4}$$

FAE: replaces scalar weights with functional weights and scalar inner products with functional inner products.

A Learning Algorithm for FAE:

To perform gradient back propagation to train an FAE, we need to evaluate the functional gradient of the objective function with respect to the weight function. To train the FAE with one hidden layer, one can derive the following by chain rule:

$$\frac{\partial \mathcal{L}(\Omega)}{\partial w_{j,k}^{(2)}(t)} = \frac{\partial \mathcal{L}(\Omega)}{\partial O_j^{(2)}(t)} \frac{\partial O_j^{(2)}(t)}{\partial w_{j,k}^{(2)}(t)} \tag{2.5}$$

$$\frac{\partial \mathcal{L}(\Omega)}{\partial w_{k,j}^{(1)}(t)} = \sum_{j'=1}^{P} \left\langle \frac{\partial \mathcal{L}(\Omega)}{\partial O_{j'}^{(2)}(t)}, \frac{\partial O_{j'}^{(2)}(t)}{\partial O_k^{(1)}} \right\rangle \cdot \frac{\partial O_k^{(1)}}{\partial w_{k,j}^{(1)}(t)} \tag{2.6}$$

The fundamental problem is to determine a suitable function $f(x) \in \mathcal{H}$ such that:

$$f(x) = arg \min_{f(x)\in\mathcal{H}} J[f]$$

$$J[f] := \int_a^b L\left(x, f(x), f'(x)\right) dx,$$

Gradient of the functional equation above (with constant a, b) is given by :

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

where $f' = df(x)/dx$ and

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

Derivative Calculation:

In light of the above, it is easy to show the first term in Eq. (2.5) is given by:

$$\frac{\partial \mathcal{L}(\Omega)}{\partial O_j^{(2)}(t)} = -\left(x_j(t) - O_j^{(2)}(t)\right) := \delta^{(2)}(t) \tag{2.10}$$

and the second term in Eq.(2.5) is given by:

$$\frac{\partial O_j^{(2)}(t)}{\partial w_{j,k}^{(2)}(t)} = \frac{\partial}{\partial w_{j,k}^{(2)}(t)} \sum_{k'=1}^{d} O_{k'}^{(1)} w_{j,k'}^{(2)}(t) = O_k^{(1)}. \tag{2.11}$$

We then have:

$$\frac{\partial O_j^{(2)}(t)}{\partial O_k^{(1)}} = \frac{\partial \sum_{k'=1}^{d} O_{k'}^{(1)} w_{j,k'}^{(2)}(t)}{\partial O_k^{(1)}} = w_{j,k}^{(2)}(t) \tag{2.12}$$

It then follows that:

$$\begin{aligned}\frac{\partial O_k^{(1)}}{\partial w_{k,j}^{(1)}(t)} &= \frac{\partial \sigma\left(\sum_{j'=1}^{P} \int_{\tau_{j'}} x_{j'}(t) w_{k,j'}^{(1)}(t)dt\right)}{\partial w_{k,j}^{(1)}(t)} \\ &= \sigma' \cdot \frac{\partial \int_{\tau_j} x_j(t) w_{k,j}^{(1)}(t)dt}{\partial w_{k,j}^{(1)}(t)} = \sigma' \cdot x_j(t)\end{aligned} \tag{2.13}$$

where $\sigma' = \partial\sigma(h)/\partial h$. Combing the preceding results, we have:

$$\frac{\partial \mathcal{L}(\Omega)}{\partial w_{j,k}^{(2)}(t)} = \frac{1}{n} \sum_{i=1}^{n} O_{i,k}^{(1)} \delta_{i,j}^{(2)}(t) \tag{2.14}$$

$$\frac{\partial \mathcal{L}(\Omega)}{\partial w_{k,j}^{(1)}(t)} = \sum_{i=1}^{n} \delta_{i,k}^{(1)} \cdot \sigma' \cdot x_{i,j}(t) \tag{2.15}$$

where $\delta_k^{(1)}$ is defined as

(2.16)

$$\delta_k^{(1)} = \sum_{j=1}^{P} \left\langle \delta_j^{(2)}(t), w_{j,k}^{(2)}(t) \right\rangle = \sum_{j=1}^{P} \int_{\tau_j} \delta_j^{(2)}(t) w_{j,k}^{(2)}(t)dt.$$

Functional Adam Optimizer:

Adam optimizer [11] to the functional setting, we maintain exponential moving averages of the first and second order moments of the functional gradients. Let $m(t)$ and $v(t)$ be the first and second order moment function estimates, the exponential moving average steps become:

(2.17)

$$m_{s,j,k}^{(l)}(t) = \beta_1 m_{s-1,j,k}^{(l)}(t) + (1-\beta_1) \frac{\partial \mathcal{L}(\Omega_{s-1})}{\partial w_{s-1,j,k}^{(l)}(t)}$$

(2.18)

$$v_{s,j,k}^{(l)}(t) = \beta_2 v_{s-1,j,k}^{(l)}(t) + (1-\beta_2) \left(\frac{\partial \mathcal{L}(\Omega_{s-1})}{\partial w_{s-1,j,k}^{(l)}(t)}\right)^2$$

where β_1 and β_2 are hyper-parameters. The bias correction steps in functional Adam optimizer can be written as:

$$\hat{m}_s(t) = m_s(t)/\left(1-\beta_1^s\right) \tag{2.19}$$

$$\hat{v}_s(t) = v_s(t)/\left(1-\beta_2^s\right) \tag{2.20}$$

Finally, the functional weights are updated by:

$$w_{s,j,k}^{(l)}(t) = w_{s-1j,k}^{(l)}(t) - \gamma \cdot \frac{\hat{m}_{s,j,k}^{(l)}(t)}{\sqrt{\hat{v}_{s,j,k}^{(l)}(t)} + \epsilon}. \tag{2.21}$$

Training an FAE:

Algorithm 1: Training FAE using Adam optimizer

```
   Input: Multi-variate functional data
          $\mathbf{X} \in \mathcal{H}^{P\times n}$, dimension of embedding $d$
   Output: embedding representation $\mathbf{Y} \in \mathbb{R}^{n\times d}$
1  Initialize functional weights using Eq.(2.22) ;
2  Initialize iteration counter $s = 0$;
3  Initialize all $m_0(t) = 0, v_0(t) = 0$;
4  Feed-forward pass using Eq.(2.2) and Eq.(2.3);
5  while $s \leq$ max iteration and not converged do
6      $s = s + 1$;
7      Compute $\delta^{(2)}(t)$ using Eq.(2.10);
8      Compute $\frac{\partial \mathcal{L}(\Omega)}{\partial w_{s,j,k}^{(2)}(t)}$ using Eq.(2.14);
9      Compute $\delta^{(1)}(t)$ using Eq.(2.16);
10     Compute $\frac{\partial \mathcal{L}(\Omega)}{\partial w_{s,k,j}^{(1)}(t)}$ using Eq.(2.15);
11     Compute all $\hat{m}_{s,k,j}^{(l)}(t)$ using Eq.(2.17,2.19);
12     Compute all $\hat{v}_{s,k,j}^{(l)}(t)$ using Eq.(2.18,2.20);
13     Update all functional weights by Eq.(2.21);
14     Feed-forward pass using Eq.(2.2) and
         Eq.(2.3);
15     Evaluate objective function Eq.(2.4);
16 end
17 Compute encoding $\mathbf{Y}$ by Eq.(2.2)
```

Functional Autoencoder for Smoothing and Representation Learning- 2024

Functional Data Analysis:

- **Functional Data Regression**: Relationship between functional responses and functional or scalar predictor.
- **Functional Data Classification**: Categorizing functional observations into different classes.
- **Functional Data Representation**: Identifying the underlying structure of functional data.

Traditional FDA	Deep Neural Networks
Linear Finite dimensional space Less accurate	It can handle nonlinear structures & complexity Works well with high dimensions Flexible (different types of data) More accurate predictions

Autoencoder Neural Network:

- L hidden layers.
- Reconstruction error: $L(X(t), \hat{X}(t)) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \int_{\mathcal{T}} (X_i(t) - \hat{X}_i(t))^2 dt.$
- The k-th neuron in the first hidden layer: $$h_k^{(1)} = g\left(\int_{\mathcal{T}} X(t) w_k^I(t) dt\right),$$
- The k-th neuron in the l-th hidden layers: $$h_k^{(l)} = g\left(\sum_j^{K^{(l)}} h_j^{(l-1)} w_j^{(l)}\right),$$
- Set of functional weights: $\{w_k^O(t)\}_{k=1}^{K^{(L)}}$
- The outputted functional neuron:

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

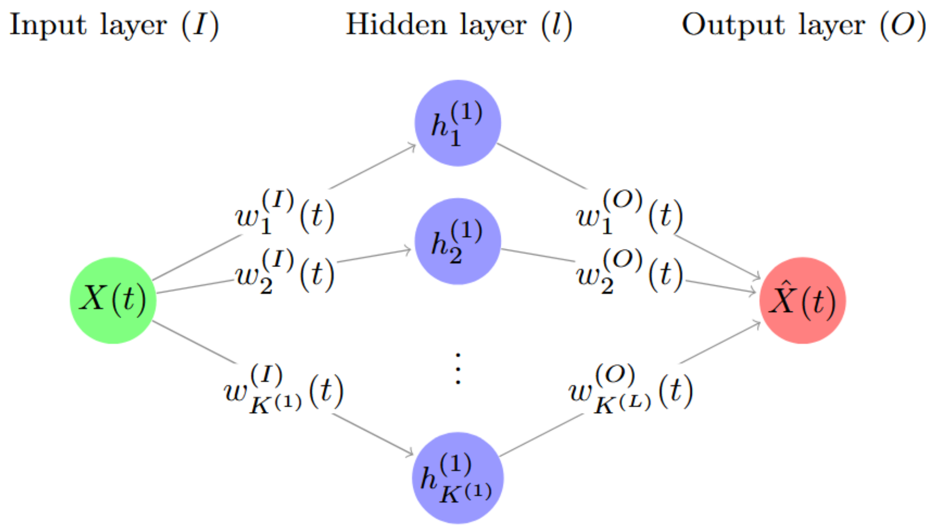


Fig. 1: Functional autoencoder for continuous data with $L = 1$ hidden layer.

Autoencoders for discrete functional data:

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

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

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

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

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

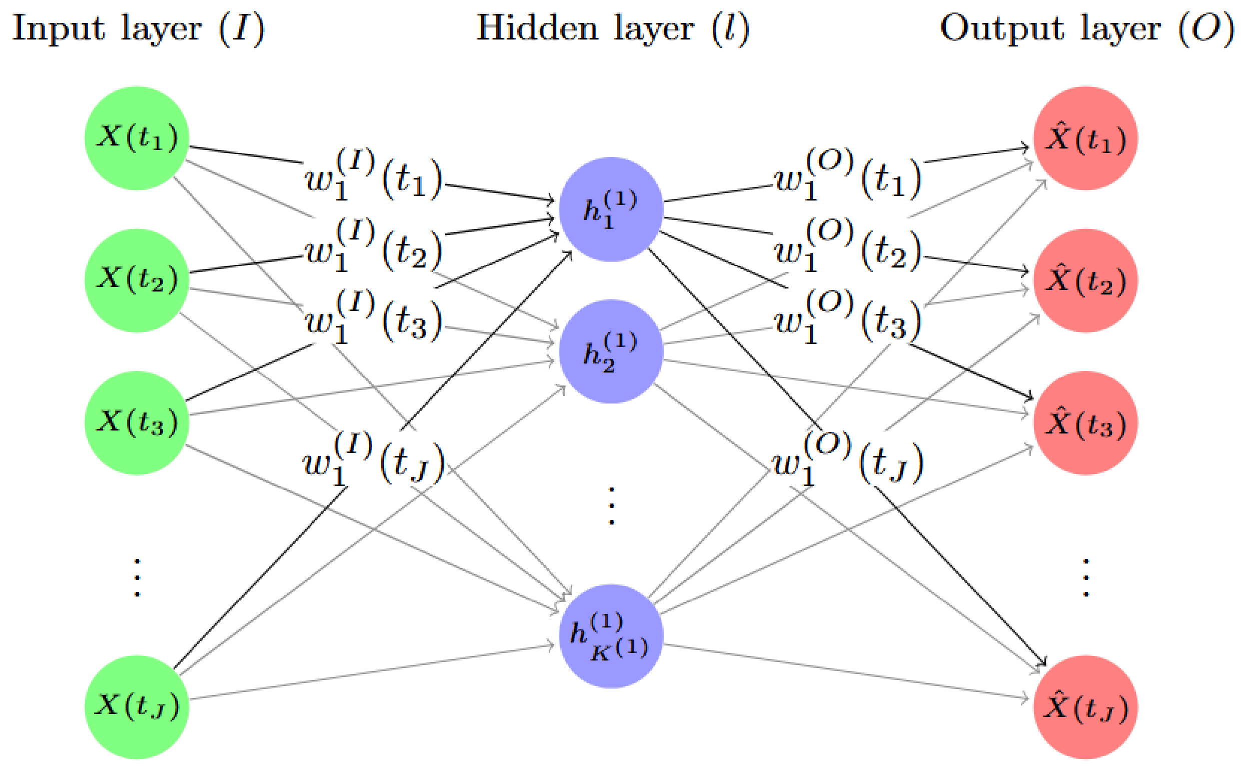


Fig. 2: Functional autoencoder for discrete data with $L = 1$ hidden layer.

Encoder with a feature layer:

A feature layer is the encoder's first step that extracts meaningful features from the raw input before any deeper processing. It denoises and compresses curves by learning task-specific "filters" (features) before the rest of the network, making the encoder effective even when sampling is irregular.

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

Input layer Feature layer First hidden layer

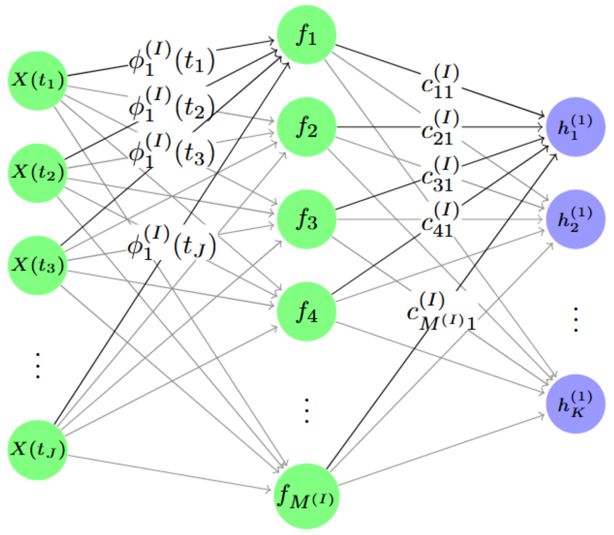


Fig. 3: Encoder with a Feature Layer. Notice that the input and feature layers are devoid of parameters at this point and are entirely deterministic given the data and the choice of basis function for w.

Decoder with a coefficient layer:

The coefficient layer lets the decoder turn small codes into clean, smooth, and interpretable functions, cheaply and reliably.

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

Last hidden layer Coefficient layer Output layer

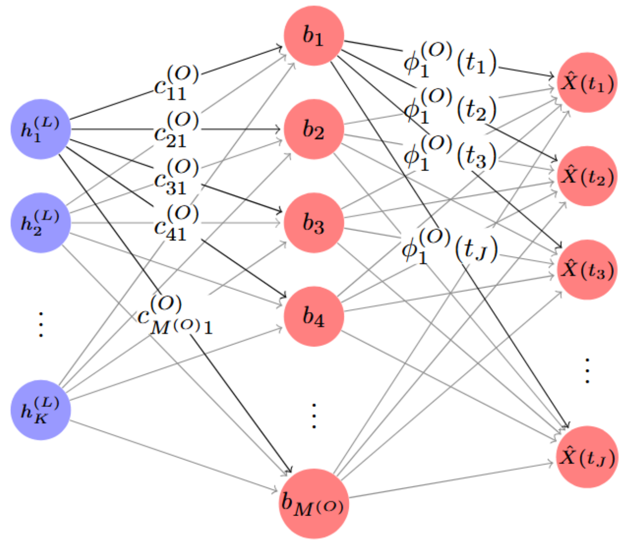


Fig. 4: Decoder with a Coefficient Layer. Similarly, the last two layers are devoid of parameters and are deterministic.

Training the proposed FAE:

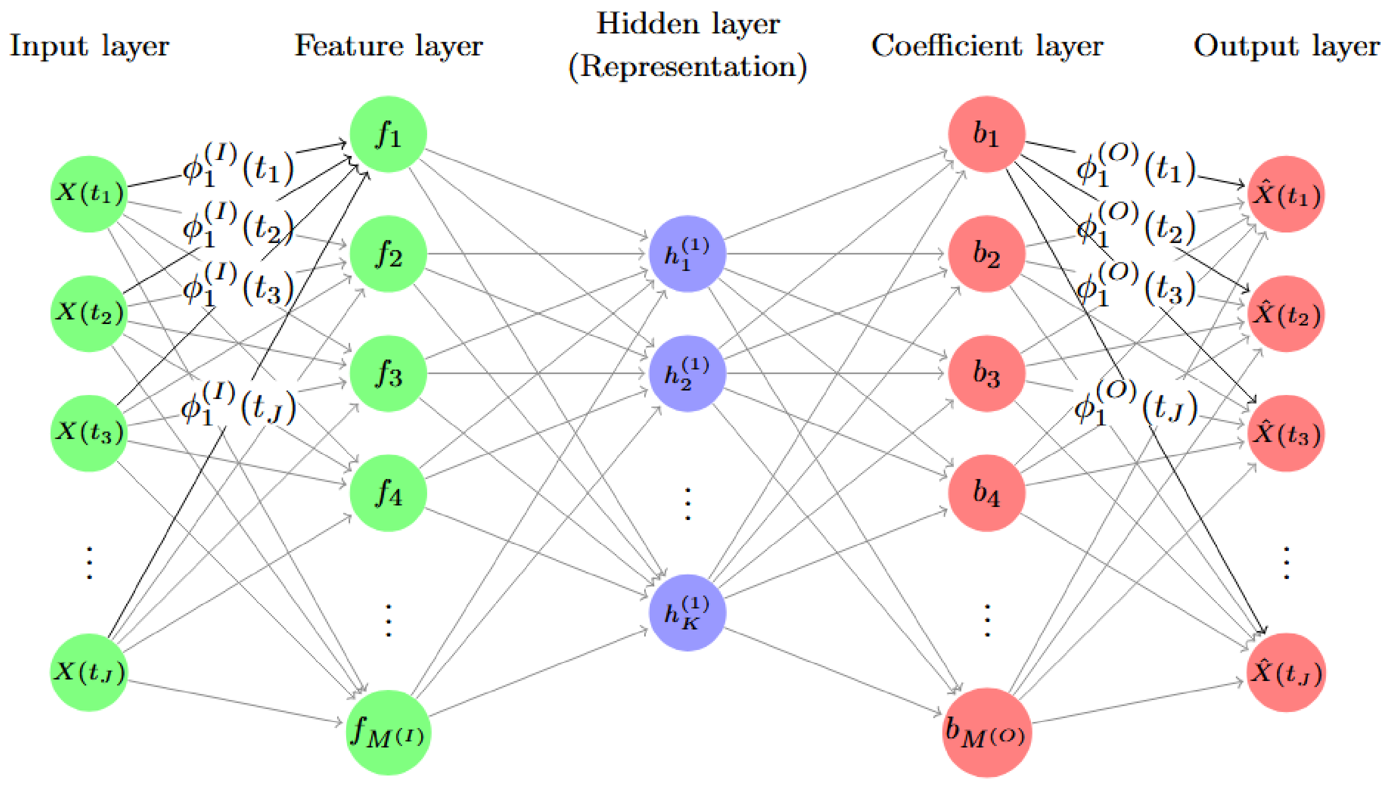


Fig. 5: A graphical representation of the FAE we propose for discrete functional data. The model represented only has a single hidden layer h, that serves the role of latent representation.

Training the proposed FAE:

```
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: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}$, $\{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, $\omega_j$ for all $j$, a pre-defined
  network NN($\theta$) 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 $\varrho$),
  etc.
1 Input Layer → Feature Layer
    $\{X(t_j)\}_{j=1}^{J} \to f_m = \sum_{j=1}^{J} \omega_j X(t_j) \phi_m^{(I)}(t_j), m \in \{1, 2, ..., M^{(I)}\}$
2 Feature Layer → Coefficient Layer
    $\{f_m\}_{m=1}^{M^{(I)}} \to b_m = \sum_{k=1}^{K^L} c_{mk}^{(O)} g_L \left(\cdots g_1 \left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} f_m\right)\right), m \in$
    $\{1, 2, ..., M^{(O)}\}$
    Specifically, the $k$-th neuron in the $l$-th hidden layer is constructed the same
    way as that in conventional neural networks as $h_k^{(l)} = g_l(\sum_{k=1}^{K^{(l)}} h_k^{(l-1)} w^{(l)})$,
    and $w^{(l)}$ are the scalar network weights.
3 Coefficient Layer → Output Layer
    $\{b_m\}_{m=1}^{M^{(O)}} \to \hat{X}(t_j) = \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t_j), j \in \{1, ..., J\}$
  return $\{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$
```

```
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: $\theta_{\text{updated}}$
Hyper-parameters: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}$, $\{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, $\omega_j$ for all $j$, a pre-defined
  network NN($\theta$) 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 $\varrho$),
  etc.
1 Compute loss function $L(X(t_j), \hat{X}(t_j))$
2 Set $\theta = \theta_{\text{current}}$
3 Output Layer → 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
4 Coefficient Layer → Feature Layer
    $\frac{\partial L(\theta)}{\partial \theta}$, same gradient calculation as used in traditional neural networks.
5 Feature Layer → Input Layer
    No gradient calculation involved (deterministic operation)
6 Update NN parameters $\theta^*$
  return $\theta_{\text{updated}} = \theta^*$
```

FAE as a functional data smoother

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

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

Thank You!