Functional Autoencoders for Functional Data Representation Learning

Tsung-Yu Hsieh*‡ Yiwei Sun*‡ Suhang Wang†‡ Vasant Honavar†‡

Abstract

In many real-world applications, e.g., monitoring of individual health, climate, brain activity, environmental exposures, among others, the data of interest change smoothly over a *continuum*, e.g., time, yielding multi-dimensional *functional data*. Solving clustering, classification, and regression problems with functional data calls for effective methods for learning compact representations of functional data. Existing methods for representation learning from functional data, e.g., functional principal component analysis, are generally limited to learning linear mappings from the data space to the representation space. However, in many applications, such linear methods do not suffice. Hence, we study the novel problem of learning non-linear representations of functional data. Specifically, we propose functional autoencoders, which generalize neural network autoencoders so as to learn non-linear representations of functional data. We derive from first principles, a functional gradient based algorithm for training functional autoencoders. We present results of experiments which demonstrate that the functional autoencoders outperform the state-of-the-art baseline methods.

Keywords Representation Learning, Functional Data Analysis, Functional Gradient Method

1 Introduction

In many real-world applications such as monitoring of individual health, climate, brain activity, environmental exposures, among others, the data of interest change smoothly a *continuum*, e.g. space, time, etc [30, 4, 16, 15]. In practice, such data are recorded at regularly or irregularly spaced points along the continuum. An example of such data (See Figure 1) is offered by accelerometer measurements from the person's joints while the person is performing various gestures, e.g., as part of a choreographed dance sequence. The acceleration of different joints can be naturally modelled by curves that denote continuous functions of time, yielding multi-dimensional functional data. Since the movement of different joints are generally not independent, the different curves can exhibit complex dependencies. Because of the increasing availability of functional data in many domains, there is a growing interest in *functional data analysis* (FDA) [22, 12, 35].

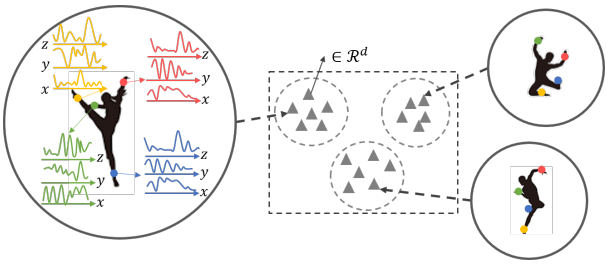


Figure 1: A simple example of functional data: Accelerometer data from multiple joints. Applications such as gesture recognition call for methods for learning compact representations of such data.

Functional data motivate the consideration of the functional counterparts of standard machine learning problems including clustering, classification, and regression. However, because functional data are sampled from a continuum, they are intrinsically infinite dimensional. In contrast, the standard machine learning methods are designed to work with data that are encoded by finite dimensional feature vectors. Hence, there is a need for methods to extract and encode the relevant information from functional data into a finite-dimensional embedding. Functional principal component analysis (FPCA) and its many variants [26, 8, 5, 2, 18] offer examples of this approach to unsupervised learning of low-dimensional representations of functional data. Such methods generalize the well-known principal component analysis (PCA) technique to the functional data setting. However, FPCA methods are generally limited to learning linear representations of functional data.

In many real-world applications, each dimension of the functional data, i.e. each feature function, can be a complex nonlinear curve. Furthermore, the correlations across the dimensions can be nonlinear and complex. Hence, linear representations such as those produced by FPCA and its variants are inadequate for modeling functional data [35]. Hence, Rossi et al. [25, 24] in-

*Department of Computer Science and Engineering,
†College of Information Sciences and Technology,
‡The Pennsylvania State University, USA,
{tuh45, yus162, szw494, vuh14}@psu.edu

troduced functional radial basis function networks for regression problems on functional data. Deep neural networks offer some of the state-of-the-art methods for learning nonlinear representations of many types of data [6, 32]. While there has been some work on the theoretical aspects of the representational power of infinite neural networks and continuous neural networks [14, 1, 20], to the best of our knowledge, existing methods for nonlinear representation learning using neural networks focus primarily on multivariate data. There is little work on effective methods for learning complex non-linear representations from functional data.

Against this background, we study the novel problem of unsupervised learning of compact nonlinear representations of multi-dimensional functional data. Specifically, we introduce functional autoencoders (FAE), a novel generalization of the well-known autoencoder neural networks to the setting of functional data. FAE aim to address two key challenges: (i) How to design a neural network that can learn vector representation from multidimensional functional data; and (ii) How to train such a neural network. The major contributions of this paper are as follows:

- We study a novel problem of learning compact nonlinear representations of multi-dimensional functional data;
- We introduce functional autoencoders that generalize the well known neural network autoencoders to the functional data setting;
- We derive a *functional gradient* based learning algorithm to optimize the parameters of FAE so as to minimize the reconstruction error of the functional data;
- We present the results of experiments on several benchmark data sets that demonstrate the effectiveness of the FAE in learning compact nonlinear representations of functional data.

The resulting FAE and the optimization techniques introduced in this paper can be used to design deep neural networks for functional data classification, scalar/functional response regression, functional data compression, and functional data clustering.

2 Functional Representation Learning

We begin by introducing useful notations and problem definition before proceeding to give the details of FAE.

Notations. Throughout the paper, we denote P-dimensional functional data by $\mathbf{x} = [x_1(t), x_2(t), \ldots, x_P(t)] \in \mathcal{H}^P$, where $\mathcal{H}$ denotes the functional space, a subspace of the square-integrable

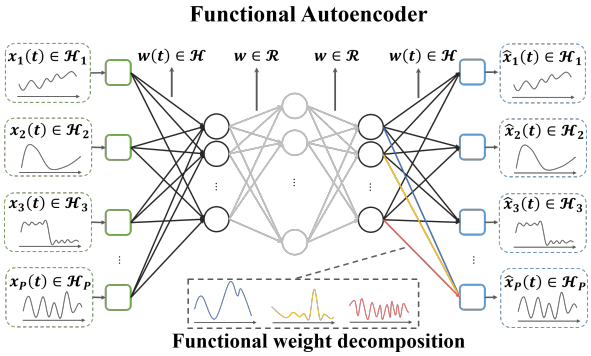


Figure 2: The functional autoencoder takes multivariate functional data as input and outputs a reconstruction of it. The first layer realizes a mapping from the functional space to vector space and the last layer performs a mapping from the vector space into the functional space. Multiple vector space to vector space transformations can be included between the first and the last layers.

functional space L^2. $x_j \in \mathcal{H}_j, j = 1, \ldots, P$ is a function and each x_j can have its own arity. $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is the set of n samples of functional data. We use $\mathbf{y} = [y_1, \ldots, y_d] \in \mathbb{R}^d$ to denote the d-dimensional encoding, and $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ to denote the encodings of the entire set. Subscripts j, k denote nodes in the neural network and i will be used to identify samples. Superscript l indexes the layers of the the neural network. We use s to denote training iteration.

Problem Definition. The unsupervised nonlinear multi-dimensional functional representation learning problem is formally defined as follows:

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

2.1 Functional Autoencoder.

An autoencoder is constructed by learning an encoder ϕ, i.e., a mapping from a P-dimensional vector-valued input space $\mathbb{R}^P$ to a d-dimensional representation space $\mathbb{R}^d$; and a decoder ψ, i.e., a mapping from the representation space $\mathbb{R}^d$ to a P-dimensional vector-valued output space so as to minimize over the training set, a measure of the reconstruction error, e.g., mean squared error, of the output with respect to input. FAE, the functional counterpart of the 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: $\phi : \mathcal{H}^P \to \mathbb{R}^d$ and $\psi : \mathbb{R}^d \to \mathcal{H}^P$ so as to minimize the

reconstruction error:

$$\phi, \psi = arg \min_{\phi,\psi} \|\mathbf{x} - (\psi \circ \phi)\,\mathbf{x}\|^2 . \tag{2.1}$$

The resulting finite dimensional encoding of functional data makes it possible to apply existing machine learning algorithms on downstream tasks e.g., clustering, classification. An illustration of the FAE framework is shown in Figure 2.

We encode multi-dimensional functional data $\mathbf{x}$ into $\mathbf{y}$, by generalizing the weights and scalar inner product in the original autoencoder by functional weights $w \in \mathcal{H}$ 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

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

where τ_j is the domain of $x_j(t)$, and $\sigma(\cdot)$ is the activation function. The framework can accommodate any of the widely used activation functions e.g., linear, *tanh* and *ReLU*. It is straightforward to additional layers $O_k^{(l)}$ $(l > 1)$ to learn more complex nonlinear encodings of the input data.

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 $O_j^{(l)}(t) \in \mathcal{H}_j, j = 1, \ldots, P$ are calculated by

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

At the output nodes, we use linear activation function. Once trained, the functional weights of the FAE decompose functional input and reveal the mode of variations in the data.

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:

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

where Ω is the collection of all functional weights.

2.1.1 FAE generalizes the AutoEncoder. The FAE is a functional generalization of autoencoder to directly handle functional data. FAE replaces the scalar weights of the autoencoder with functional weights and scalar inner product by functional inner product. Thus, the standard autoencoder is a special case of the proposed FAE.

2.1.2 Connection to FPCA. As pointed out in [3], an one-layer autoencoder with a linear activation function essentially behaves like the PCA. A similar relationship holds between the FAE and functional PCA. The FPCA is a special case of FAE when the FAE uses linear activation functions in the hidden layer and the functional weights are constrained to be orthonormal.

2.2 A Learning Algorithm for FAE. Unlike in the case of conventional neural networks where both the objective function is a real-valued function of the real-valued parameters (weights), in the FAE, the objective function is replaced by a *functional*, or function of model parameters that are *real-valued functions*. To perform gradient back-propagation to train an FAE, we need to evaluate the *functional gradient* of the objective functional w.r.t the weight functions, $\frac{\partial \mathcal{L}(\Omega)}{\partial w_{a,b}^{(l)}(t)}$, which measures a change in a functional in response to a change in a function on which the functional depends. We turn to the calculus of variation [21], a field of mathematical analysis, for evaluating functional gradients. Additionally, we extend the Adam optimizer [11], a state-of-the-art method for optimizing the parameters of a standard neural network with vector valued inputs and weights, to the functional setting.

2.2.1 Functional Derivatives. For simplicity, we show the gradient derivation for the FAE with a single hidden layer. However, it is quite straightforward to generalize the derivation to a multi-layer FAE. 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

$$f(x) = arg \min_{f(x)\in\mathcal{H}} J[f] \tag{2.7}$$
$$J[f] := \int_a^b L\left(x, f(x), f'(x)\right) dx,$$

under some appropriate boundary conditions. The minimizers of an objective function defined on a finite-dimensional vector space are characterized as critical points, where the gradient of the objective function vanishes. The calculus of variations provides an analogous construction in the case of the functional space. The gradient of the functional in Eq.(2.7) is given by [21]

$$\nabla J[f] = \frac{\partial L(x,f,f')}{\partial f} - \frac{d}{dx}\left(\frac{\partial L(x,f,f')}{\partial f'}\right), \tag{2.8}$$

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

$$\frac{\partial a f^b(x)}{\partial f(x)} = b \cdot a f^{b-1}(x), \tag{2.9}$$

where a, b are constants.

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

$$\delta_k^{(1)} = \sum_{j=1}^{P} \left\langle \delta_j^{(2)}(t), w_{j,k}^{(2)}(t) \right\rangle = \sum_{j=1}^{P} \int_{\tau_j} \delta_j^{(2)}(t) w_{j,k}^{(2)}(t) dt. \tag{2.16}$$

Note that since $\delta_k^{(1)} \in \mathbb{R}$, the derivatives propagate consistently when we add additional hidden layers that implement additional mappings between vector spaces.

2.2.2 Functional Adam Optimizer To extend the 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:

$$m_{s,j,k}^{(l)}(t) = \beta_1 m_{s-1,j,k}^{(l)}(t) + (1-\beta_1) \frac{\partial \mathcal{L}(\Omega_{s-1})}{\partial w_{s-1,j,k}^{(l)}(t)} \tag{2.17}$$

$$v_{s,j,k}^{(l)}(t) = \beta_2 v_{s-1,j,k}^{(l)}(t) + (1-\beta_2) \left(\frac{\partial \mathcal{L}(\Omega_{s-1})}{\partial w_{s-1,j,k}^{(l)}(t)}\right)^2 \tag{2.18}$$

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:

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

2.2.3 Training an FAE The training of FAE with one hidden layer is summarized in Algorithm 1. The inputs to the algorithm are multi-dimensional functional data and the desired dimensionality d of the encoding to be learned. The functional weights are initialized to random functions as follows:

$$w(t) = \sum_{k=1}^{b} c_k \phi_k(t) \tag{2.22}$$

where c_ks are random coefficients and $\phi_k(t)$s are the set of known basis functions, e.g. b-spline, Fourier among others. Thereafter, the algorithm iterates between feed-forward pass and back-propagation weight update pass until the specified termination criterion is satisfied. Finally, the encoding is obtained by the encoder portion of

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 $\mathbf{Y} \in \mathbb{R}^{n\times d}$

Initialize functional weights using Eq.(2.22) ;
Initialize iteration counter $s = 0$;
Initialize all $m_0(t) = 0, v_0(t) = 0$;
Feed-forward pass using Eq.(2.2) and Eq.(2.3);
while $s \leq$ max iteration and not converged **do**
  $s = s + 1$;
  Compute $\delta^{(2)}(t)$ using Eq.(2.10);
  Compute $\frac{\partial \mathcal{L}(\Omega)}{\partial w^{(2)}_{s,j,k}(t)}$ using Eq.(2.14);
  Compute $\delta^{(1)}(t)$ using Eq.(2.16);
  Compute $\frac{\partial \mathcal{L}(\Omega)}{\partial w^{(1)}_{s,k,j}(t)}$ using Eq.(2.15);
  Compute all $\hat{m}^{(l)}_{s,k,j}(t)$ using Eq.(2.17,2.19);
  Compute all $\hat{v}^{(l)}_{s,k,j}(t)$ using Eq.(2.18,2.20);
  Update all functional weights by Eq.(2.21);
  Feed-forward pass using Eq.(2.2) and Eq.(2.3);
  Evaluate objective function Eq.(2.4);
end
Compute encoding $\mathbf{Y}$ by Eq.(2.2)

FAE. It is straightforward to generalize the algorithm to work with an FAE with multiple layers, e.g., by stacking multiple such autoencoders or inserting additional hidden layers.

3 Experiments

We proceed to describe the experiments we designed to evaluate the effectiveness of the proposed FAE for representation learning from functional data. The experiments aim to answer the following research questions: **1)** Can the FAE be effectively trained using the proposed Functional Adam Optimizer? **2)** How does the encoding learned by FAE compare with those obtained using the state-of-the-art baseline methods?

3.1 Experimental Setup.

3.1.1 Baseline Methods. We compare FAE with the following representative baselines:

- **parAE** exploits multiple standard autoencoder (AE) neural networks [7] in parallel where each AE learns an encoding from one of the functional features, independently from others. The individual encodings are added together to obtain the joint encoding for the multi-dimensional functional data.
- **CONVAE** is based on the convolutional neural network [13]. We adopt one convolution layer with kernel size $P \times t$ following the setup in [10] where t is a hyperparameter. It is followed by a linear transformation layer to obtain the encoding from the functional data. The decoder part of CONVAE consists of two linear layers and a reshape layer for reconstruction.
- **LSTMAE** is based on two unidirectional long short-term memory (LSTM) [27] layers, one for encoding and the other for decoding. The decoder reconstructs the input in reverse order.
- **FPCA** [22] is the functional principal component analysis and the encoding of the functional data is obtained from the principal component scores.

3.1.2 Data Sets. We report the results of our experiments on a synthetic data and several publicly available real-world data sets.

Synthetic Data. To generate a $P-$dimensional synthetic data containing C clusters, we first generate C template functions on each dimension and then the actual samples will be obtained by adding noise to the template functions. Specifically, for a given set of basis functions $\phi_k(t), k = 1, \ldots, b$, a set of P-dimensional random coefficient vectors are drawn from multivariate normal distribution,

$$\boldsymbol{\alpha}_k \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}_{P\times P}\right), \quad k = 1, \ldots, b, \tag{3.23}$$

where $\boldsymbol{\Sigma}_{P\times P}$ is a $P \times P$ covariance matrix and it characterizes the correlation among feature functions. The template function for the $j-$th dimension is then obtained by $\tilde{f}_j = \sum_{k=1}^{b} \boldsymbol{\alpha}_{k,j}\phi_k(t), j = 1, \ldots, P$, and $\boldsymbol{\alpha}_{k,j}$ is the $j-$th element of the vector $\boldsymbol{\alpha}_k$. Samples in the same cluster are produced by modifying the template function to incorporate two sources of variability: The first controls the perturbation of the curve; and the second simulates observation error. For perturbing the curve, we add Gaussian noise to the coefficients of the template function:

$$\boldsymbol{\alpha}'_{k,j} = \boldsymbol{\alpha}_{k,j} + \mathcal{N}\left(0, \beta_1 \cdot |\boldsymbol{\alpha}_{k,j}|\right). \tag{3.24}$$

To simulate observation error, we add Gaussian noise to the function value at each sampling point. Thus, the j-th feature function of a sample is obtained by

$$x_j(t) = \sum_{k=1}^{b} \boldsymbol{\alpha}'_{k,j}\phi_k(t) + \mathcal{N}_t(0, \beta_2), \quad j = 1 \ldots, P, \tag{3.25}$$

Algorithm 2: Synthetic data generation

Input: Number of features P, a set of basis functions $[\phi_1(t), \ldots, \phi_b(t)]$, number of clusters C, samples in each cluster $[n_1, \ldots, n_C]$

Output: Synthetic data $\mathbf{X}$

```
Initialize empty X;
for c = 1, ..., C do
    Generate random covariance matrix Σ_{P×P};
    Generate template vectors α_k by Eq.(3.23);
    for i = 1, ..., n_C do
        for j = 1, ..., P do
            Generate perturbed coefficients α'_{k,j}
              by Eq.(3.24);
            Generate feature function x_j by
              Eq.(3.25);
        end
        x_i ← [x_1(t), ..., x_p(t)];
        X ← [X, x_i]
    end
end
```

where with a slight abuse of notation, we use $\mathcal{N}_t$ to denote that the value is drawn independently at each t, and β_1, β_2 to control the magnitude of the noise. The synthetic data generation process is summarized by Algorithm 2. The algorithm accepts the number of features, number of clusters, number of samples in each cluster etc. as input. For each cluster, the algorithm first defines the template function; Samples from each cluster are drawn after applying curve perturbation to the template coefficients and observation noise.

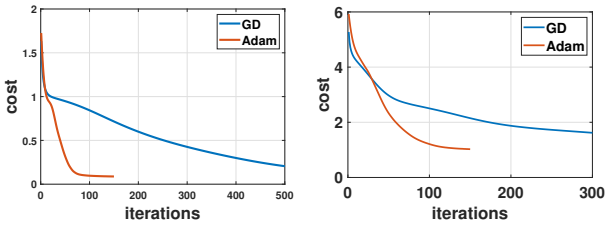
Real-world Data. The real-world data sets are summarized as follows: **AWR** [36] is an Electromagnetic Articulograph (EMA) data set which contains the data measured from the movement of the tongue and lips during speech. **CharTraj** [37] contains the 2-dimensional movement of a pen as well as pen tip force from natural handwriting. **PM2.5** [17] is a data set containing the PM2.5 data for the US Embassy in Beijing. **UWave** [19] contains accelerometer-based data obtained from a set of gestures. The specification of the data sets used in the experiment is provided in Table 1.

3.2 Convergence of the learning algorithm

To answer research question 1, we train an FAE on each data set with the functional variant of the standard gradient descent (GD) and the proposed functional extension of the Adam optimizer and empirically demonstrate their convergence. In this experiment, we set d to 10, and use batch update. Due to space constraints, a subset of the results are shown in Figure 3.

Table 1: Key Statistics of the Data Sets. C denotes the number of clusters in the functional data. T is the length of the functional data.

Data set	P	n	C	T
(1) Synthetic	10	500	5	100
(2) AWR	9	300	25	144
(3) CharTraj	3	1436	20	182
(4) PM2.5	8	1571	6	24
(5) UWave	3	896	8	315



(a) Synthetic (b) PM2.5

Figure 3: Empirical convergence of the optimization algorithms on the FAE.

We observe that the objective functional evaluated on the training data gradually decreases with training iterations for both functional GD and the functional Adam optimizer. We observe that the functional extension of the Adam optimizer is more efficient than the functional variant of standard GD.

The actual run time of the full-batch functional Adam update iteration (based on a MATLAB implementation on a 3.3GHz Intel Core i5 hardware) is shown in Table 3. The total number of training iterations until convergence is usually around a couple hundreds (See Figure 3). We conclude that the functional Adam optimizer is fast enough for practical use.

3.3 Clustering Performance

To answer research question 2, we extract representations from functional data using the aforementioned methods and use them in a clustering task. In each case, the dimensionality d of the representation is set to 20. The resulting 20-dimensional embeddings of the functional data are clustered using the standard K-means clustering algorithm. The clustering performance is evaluated using standard metrics such as clustering accuracy, normalized mutual information (NMI), and purity. We report performance metrics averaged over 20 independent trials in Table 2.

We observe that the two functional approaches FPCA and FAE outperform the other baseline methods parAE, CONVAE, and LSTMAE with in terms of clustering performance. The parAE treats the sampling

Table 2: Performance evaluated by K-means clustering on the representation extracted from functional data using different representation learning methods. Boldface figures are used to denote the best performers when they outperform others significantly (threshold set at 0.05).

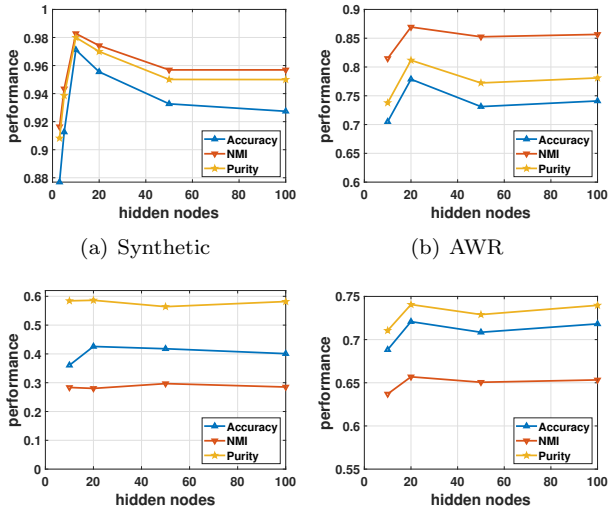
Data set	Metric	parAE	CONVAE	LSTMAE	FPCA	FAE(ours)
Synthetic	Acc	0.6576 ± 0.0503	0.8469 ± 0.1429	0.7117 ± 0.1246	0.8703 ± 0.1472	**0.9555** ± 0.1087
	NMI	0.5705 ± 0.0494	0.9046 ± 0.0885	0.6173 ± 0.1048	0.9225 ± 0.0879	**0.9742** ± 0.0631
	Purity	0.6604 ± 0.0452	0.8900 ± 0.1021	0.7369 ± 0.0869	0.9100 ± 0.1021	**0.9700** ± 0.0733
AWR	Acc	0.5838 ± 0.0398	0.5395 ± 0.0274	0.4798 ± 0.0314	0.7370 ± 0.0437	**0.7787** ± 0.0518
	NMI	0.7022 ± 0.0210	0.6754 ± 0.0161	0.6408 ± 0.0161	**0.8581** ± 0.0223	**0.8694** ± 0.0261
	Purity	0.6155 ± 0.0375	0.5682 ± 0.0249	0.5152 ± 0.0281	0.7812 ± 0.0356	**0.8117** ± 0.0411
CharTraj	Acc	0.4498 ± 0.0240	0.4755 ± 0.0342	0.2724 ± 0.0089	0.4700 ± 0.0399	**0.5302** ± 0.0333
	NMI	0.5195 ± 0.0117	0.5988 ± 0.0221	0.3571 ± 0.0034	0.6043 ± 0.0274	**0.6333** ± 0.0274
	Purity	0.4881 ± 0.0164	0.5216 ± 0.0272	0.2914 ± 0.0086	0.5175 ± 0.0296	**0.5720** ± 0.0251
PM2.5	Acc	0.3014 ± 0.0111	0.2735 ± 0.0141	0.3688 ± 0.0370	0.3877 ± 0.0348	**0.4258** ± 0.0132
	NMI	0.2154 ± 0.0072	0.0595 ± 0.0026	0.2434 ± 0.0197	0.2134 ± 0.0163	**0.2802** ± 0.0319
	Purity	0.5122 ± 0.0059	0.4994 ± 0.0045	0.5539 ± 0.0143	0.5535 ± 0.0107	**0.5862** ± 0.0151
UWave	Acc	0.5950 ± 0.0389	0.5639 ± 0.0419	0.3695 ± 0.0164	0.6870 ± 0.0607	**0.7208** ± 0.0698
	NMI	0.5003 ± 0.0191	0.4983 ± 0.0119	0.3002 ± 0.0155	**0.6563** ± 0.0224	**0.6569** ± 0.0239
	Purity	0.6152 ± 0.0291	0.5863 ± 0.0269	0.4117 ± 0.0143	0.7183 ± 0.0469	**0.7406** ± 0.0510

points as individual features and hence is oblivious to the smoothness of the functions. Furthermore, because it processes each functional feature independently of the others, it fails to model the correlation between features. We note that the performance of LSTMAE suffers from the fairly long length of the sequence (T) combined with limited number of samples available.

Among the two methods that are specifically designed for functional data, FAE outperforms FPCA, with the differences in performance being especially striking in the case of CharTraj, PM2.5, and UWave data. The observed superiority of FAE over FPCA can be explained by FAE's ability to model nonlinearity of each functional variables as well as nonlinear interactions between the functional variables.

3.4 Parameter Analysis Determining the suitable number of hidden nodes and hidden layers in neural network models in practice is challenging and often relies on a process of trial and error. Hence, we empirically examined how the performance of FAE varies as a function of the number of hidden nodes and the number of hidden layers.

Number of Hidden Nodes To test the sensitivity of clustering performance to the number of hidden nodes, we set the number of hidden layers to 1, tune the embedding dimensionality from 10, 20, 50, 100, and report the mean results from 20 independent runs. Because of space constraints, only a subset of results is reported in Figure 4. In the case of Synthetic data



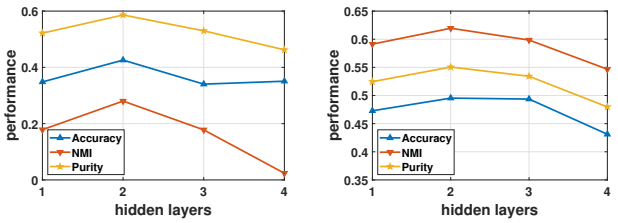
(a) Synthetic (b) AWR

(c) PM2.5 (d) UWave

Figure 4: The effect of embedding dimensionality on clustering performance.

we also report results with the number of hidden nodes set to 3 and 5 (the actual number of basis functions is 12). We note that the clustering performance on the Synthetic data improves when d approaches 10 and decreases when $d > 10$. We conjecture that the the needed modeling capacity is achieved around $d = 10$ and the addition of more hidden nodes may in fact adversely impact the performance due to over-fitting.

Number of Hidden Layers To examine the sensitivity of clustering performance to the number of hidden



(a) PM2.5 (b) CharTraj

Figure 5: The effect of number of hidden layers on clustering performance.

Table 3: Running time per functional Adam update iteration.

Dataset	(1)	(2)	(3)	(4)	(5)
time (s)	0.24	0.16	0.56	0.86	0.53

layers, we set the embedding dimensionality to 10 and experiment with the number of hidden layers ranging from 1 to 4. Due to space constraints, only a subset of the results is shown in Figure 5. The results show that the performance improves as we increase the number of hidden layers from 1 to 2 and starts to decrease as additional hidden layers are added. This is consistent with the expectation that the optimal performance is achieved when the modeling capacity approaches that necessary to capture the variability and nonlinearity present in the data.

4 Summary and Discussion

In many real-world applications, e.g., monitoring of individual health, environmental exposures, among others, the data of interest change smoothly over a *continuum*, e.g., time, yielding multi-dimensional *functional data*. Solving clustering, classification, and regression problems with functional data calls for effective methods for learning compact non-linear representations of functional data. To address this need, we introduced the unsupervised multi-dimensional nonlinear functional representation learning problem. We proposed the functional autoencoder, a generalization of the autoencoder from the vector space to the functional setting. We derived from first principles, a functional gradient based optimization algorithm for training nonlinear functional autoencoders. We extended the Adam optimizer, a state-of-the-art method for training neural networks, to the functional setting. The results of experiments show that the FAE outperforms the state-of-the-art baselines in terms of clustering performance. We note that the concepts and techniques introduced in this paper, including functional weights, functional weight update rules, and functional Adam optimizer can be used for designing functional variants of neural networks for functional data classification, functional data regression, and related tasks.

Some promising directions for future work include extensions to the functional setting of contrastive autoencoders [23], denoising autoencoders [34], etc; regularization schemes for functional weight updates to minimize overfitting; functional variants of attention mechanisms, e.g. [33, 9] to identify important feature functions as well as informative segments in the curves; and functional generalizations of graph neural networks [38, 31, 28, 29].

Acknowledgment

This work was funded in part by the NIH NCATS grant UL1 TR002014 and by NSF grants 2041759, 2020243, 1636795, 1909702, and 1955851, the Edward Frymoyer Endowed Professorship at Pennsylvania State University and the Sudha Murty Distinguished Visiting Chair in Neurocomputing and Data Science funded by the Pratiksha Trust at the Indian Institute of Science (both held by Vasant Honavar).

References

[1] Francis Bach. Breaking the curse of dimensionality with convex neural networks. *JMLR*, 18(1):629–681, 2017.

[2] Daniel Backenroth, Jeff Goldsmith, Michelle D Harran, Juan C Cortes, John W Krakauer, and Tomoko Kitago. Modeling motor learning using heteroscedastic functional principal components analysis. *JASA*, 113(523):1003–1015, 2018.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828, 2013.

[4] Enyan Dai, Yiwei Sun, and Suhang Wang. Ginger cannot cure cancer: Battling fake health news with a comprehensive data repository. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 14, pages 853–862, 2020.

[5] Clara Happ and Sonja Greven. Multivariate functional principal component analysis for data observed on different (dimensional) domains. *JASA*, 113(522):649–659, 2018.

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

[7] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *NeurIPS*, pages 3–10, 1994.

[8] Siegfried Hörmann, Łukasz Kidziński, and Marc Hallin. Dynamic functional principal components. *J. Royal Stat. Soc.: Series B*, 77(2):319–348, 2015.

[9] Tsung-Yu Hsieh, Yiwei Sun, Suhang Wang, and Vasant Honavar. Adaptive structural co-regularization for unsupervised multi-view feature selection. In *2019 IEEE International Conference on Big Knowledge (ICBK)*, pages 87–96. IEEE, 2019.

[10] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv:1404.2188*, 2014.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[12] Piotr Kokoszka and Matthew Reimherr. *Introduction to functional data analysis*. Chapman and Hall/CRC, 2017.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[14] Nicolas Le Roux and Yoshua Bengio. Continuous neural networks. In *Artificial Intelligence and Statistics*, pages 404–411, 2007.

[15] Junjie Liang, Yanting Wu, Dongkuan Xu, and Vasant Honavar. Longitudinal deep kernel gaussian process regression. In *Proceedings of the 35th AAAI Conference on Artficial Intelligence*, 2021.

[16] Junjie Liang, Dongkuan Xu, Yiwei Sun, and Vasant Honavar. Lmlfm: Longitudinal multi-level factorization machine. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4811–4818, 2020.

[17] Xuan Liang, Tao Zou, Bin Guo, Shuo Li, Haozhe Zhang, Shuyi Zhang, Hui Huang, and Song Xi Chen. Assessing beijing's pm2. 5 pollution: severity, weather impact, apec and winter heating. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2182):20150257, 2015.

[18] Zhenhua Lin and Hongtu Zhu. Mfpca: multiscale functional principal component analysis. In *AAAI*, volume 33, pages 4320–4327, 2019.

[19] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[20] Roi Livni, Daniel Carmon, and Amir Globerson. Learning infinite layer networks without the kernel trick. In *ICML*, pages 2198–2207, 2017.

[21] Peter J Olver. Introduction to the calculus of variations. *http://www-users.math.umn.edu/ olver/ln/cv.pdf*, 2019.

[22] James O Ramsay. Functional data analysis. *Encyclopedia of Statistical Sciences*, 4, 2004.

[23] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840. Omnipress, 2011.

[24] Fabrice Rossi and Brieuc Conan-Guez. Theoretical properties of projection based multilayer perceptrons with functional inputs. *Neural Processing Letters*, 23(1):55–70, 2006.

[25] Fabrice Rossi, Nicolas Delannay, Brieuc Conan-Guez, and Michel Verleysen. Representation of functional data in neural networks. *Neurocomputing*, 64:183–210, 2005.

[26] Han Lin Shang. A survey of functional principal component analysis. *AStA*, 98(2):121–142, 2014.

[27] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *ICML*, pages 843–852, 2015.

[28] Yiwei Sun, Suhang Wang, Tsung-Yu Hsieh, Xianfeng Tang, and Vasant Honavar. Megan: a generative adversarial network for multi-view network embedding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3527–3533. AAAI Press, 2019.

[29] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of The Web Conference 2020*, pages 673–683, 2020.

[30] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Charu C Aggarwal, Prasenjit Mitra, and Suhang Wang. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *AAAI*, pages 5956–5963, 2020.

[31] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. Investigating and mitigating degree-related biases in graph convoltuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1435–1444, 2020.

[32] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv:1812.05069*, 2018.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[34] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103, 2008.

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

[36] Jun Wang, Arvind Balasubramanian, Luis Mojica de la Vega, Jordan R Green, Ashok Samal, and Balakrishnan Prabhakaran. Word recognition from continuous articulatory movement time-series data using symbolic representations. 2013.

[37] Ben H Williams, Marc Toussaint, and Amos J Storkey. Extracting motion primitives from natural handwriting data. In *International Conference on Artificial Neural Networks*, pages 634–643. Springer, 2006.

[38] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2019.