Neural network architectures for functional data analysis

Cédric Beaulac

Université du Québec à Montréal

October the 20th 2023

The work presented here is the result of a collaboration with Ph.D candidate Sidi Wu and professor Jiguo Cao from Simon Fraser University and with research intern Valentin Larchevêque from Université Montpellier.

Neural network architectures for functional data analysis

Functional data
Functional Output layer
Functional Input layer
Deep learning models and implementation

The talk

- Definition of functional data.
- Parametric representation of functional data.
- Regression of functional data.
- Functional output layer.
- Functional input layer.
- Proposed models and results.
- Current work (if time allows for it).

Functional data

A brief introduction to functional data

- In functional data analysis (FDA), a replication $x_i(t)$ $t \in T$ is a function.
- The space over which the function is defined T can be time, spatial space, or higher-dimension spaces.
- A data set is a collection of such functions $S = \{x_i(t) | i \in (1, ..., n)\}$ over the same space.
- The data is collected as a collection of points over the space.

A simple functional data set.

- For the presentation, we suppose T is one-dimensional and is some measure of time.
- From now on, let us say the space-time is $[0, T]$.

A sample of points over T

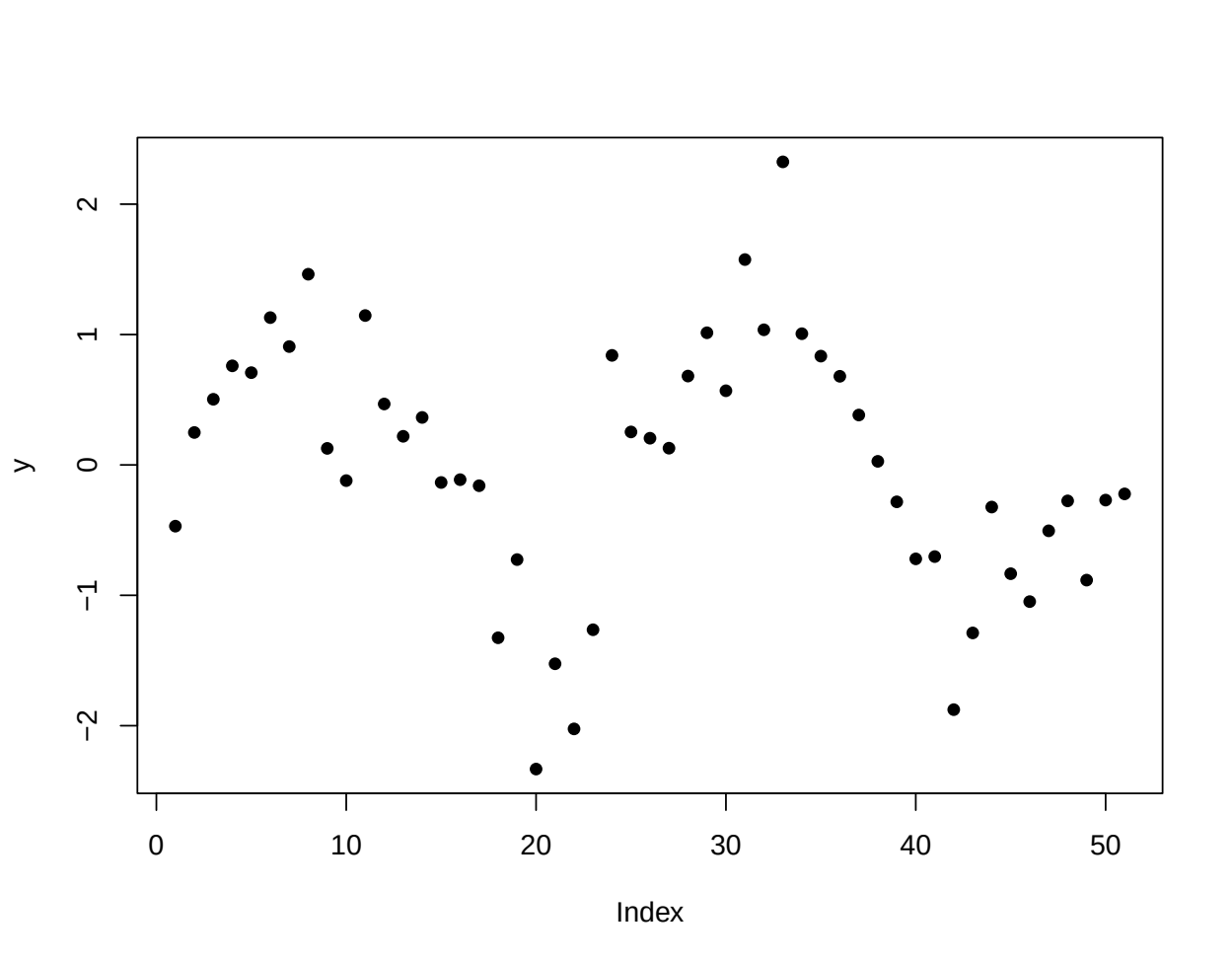

Figure: Sample from a functional data.

A simple functional data set.

This is different from time series:

- We are not trying to forecast the functional data.
- We must have multiple repetitions over T.

Multiple subjects: A sample of points over T

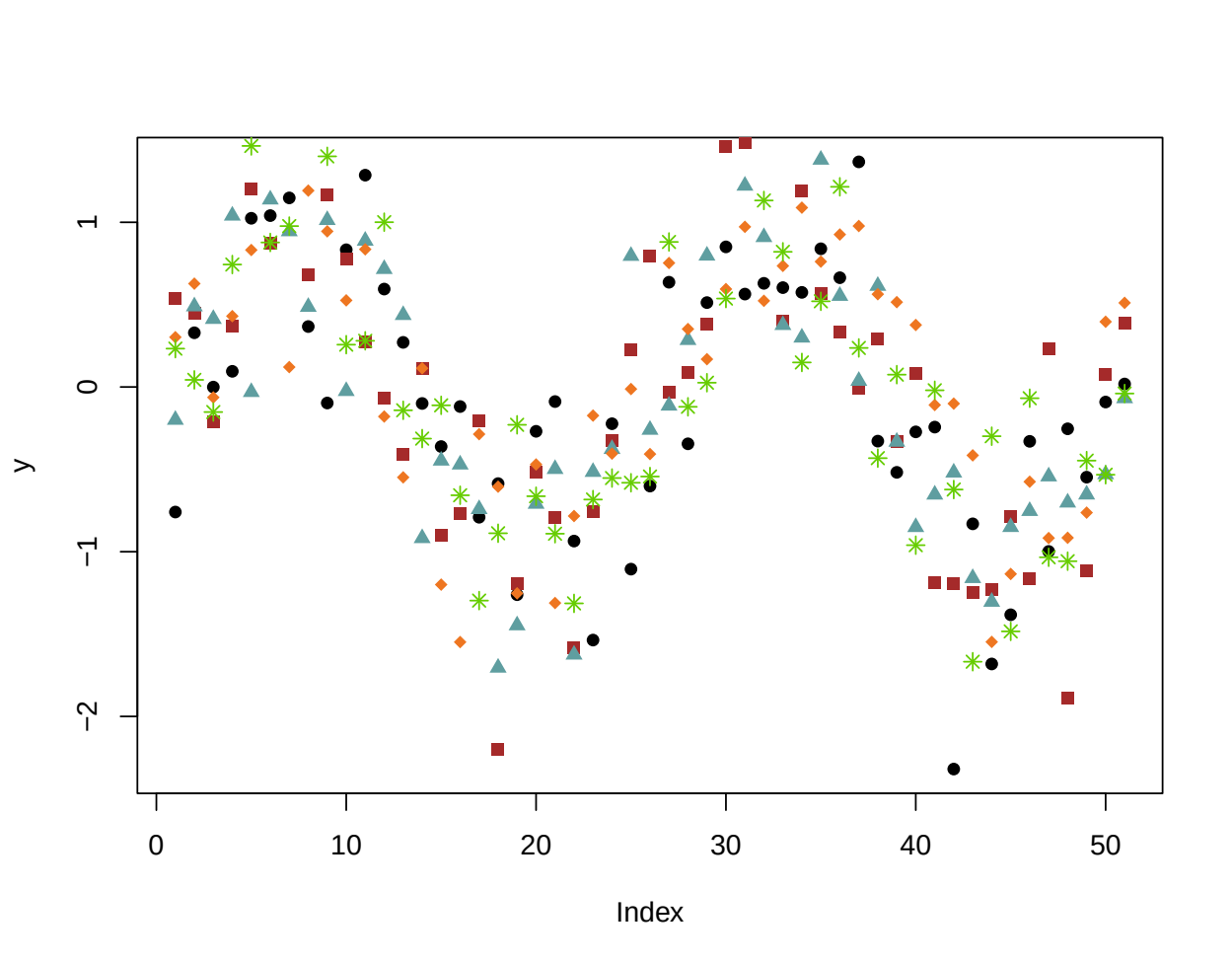

Figure: Sample from multiple functional data.

Exemple: real data set (El Nino)

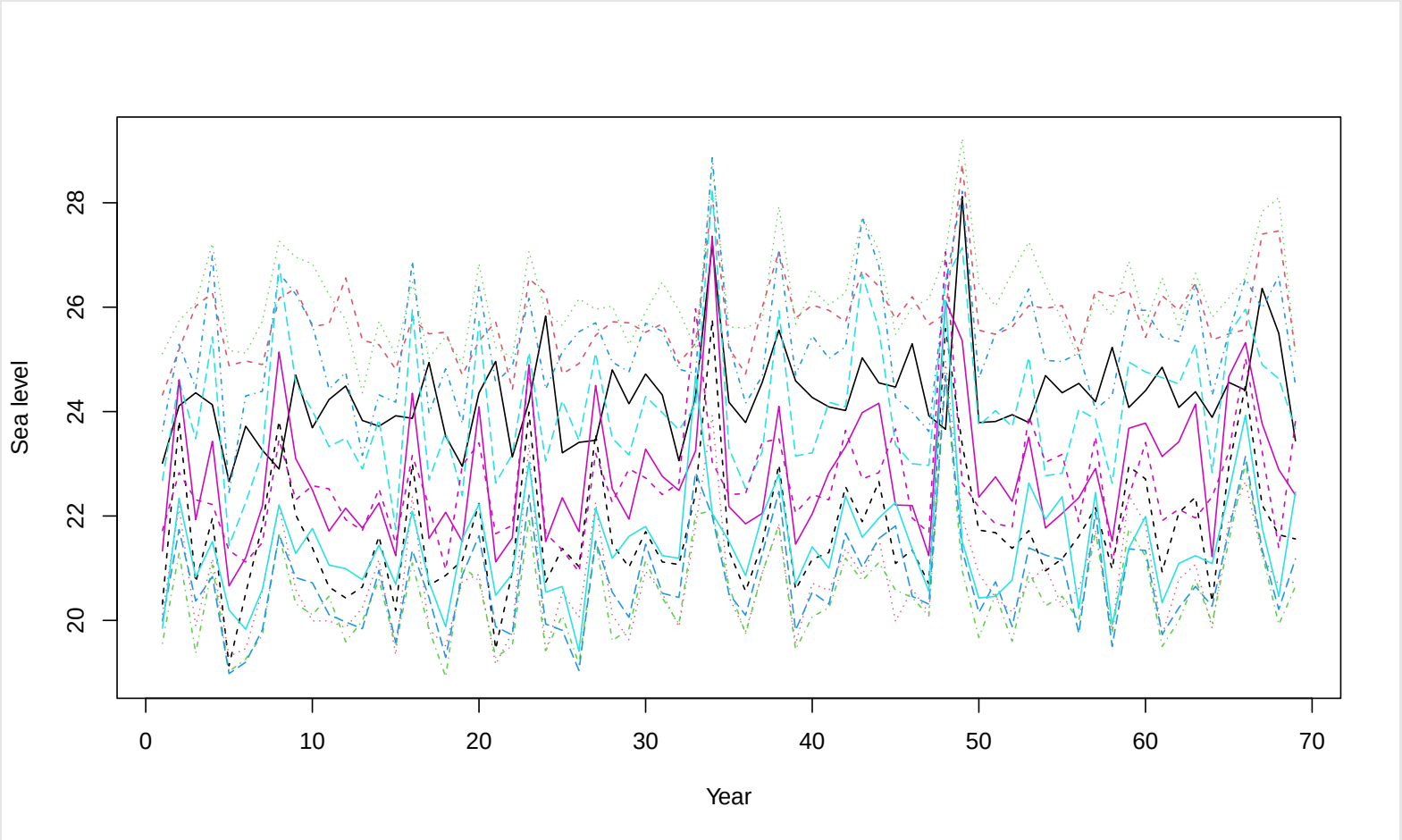

Figure: Yearly sea surface temperature.

Exemples

- Daily stock value of multiple companies.
- Engagement statistics collected with an application.
- Self-captured medical data: Diabetics track their daily sugar level.
- Weight over time.

Functional data representation

- **It is assumed that the functional data is a realization of an underlying smooth stochastic process.**
- It is common to interpolate points and produce a smooth representation for $x_i(t)$
- This smooth representation is later used in the analysis.

Functional data representation

- The standard approach to do so is to use some basis expansion; a set of basis functions (that cover the space $[0, T]$) and a set of basis coefficients.
- Thus, the continuous and smooth curve is estimated as a linear combination of those functions and parameters.
- $x(t) = \sum_{k=1}^{K} c_k B_k(t)$

Functional data representation

- $x(t) = \sum_{k=1}^{K} c_k B_k(t)$
- We obtain a continuous representation of $x(t)$ (can be evaluated for any $t \in [0, T]$).
- We only need to *learn* a discrete number of parameters to produce a smooth and continuous representation of functional data.
- $\sum_j [x(t_j) - \sum_{k=1}^{K} c_k B_k(t_j)]^2$

B-Splines

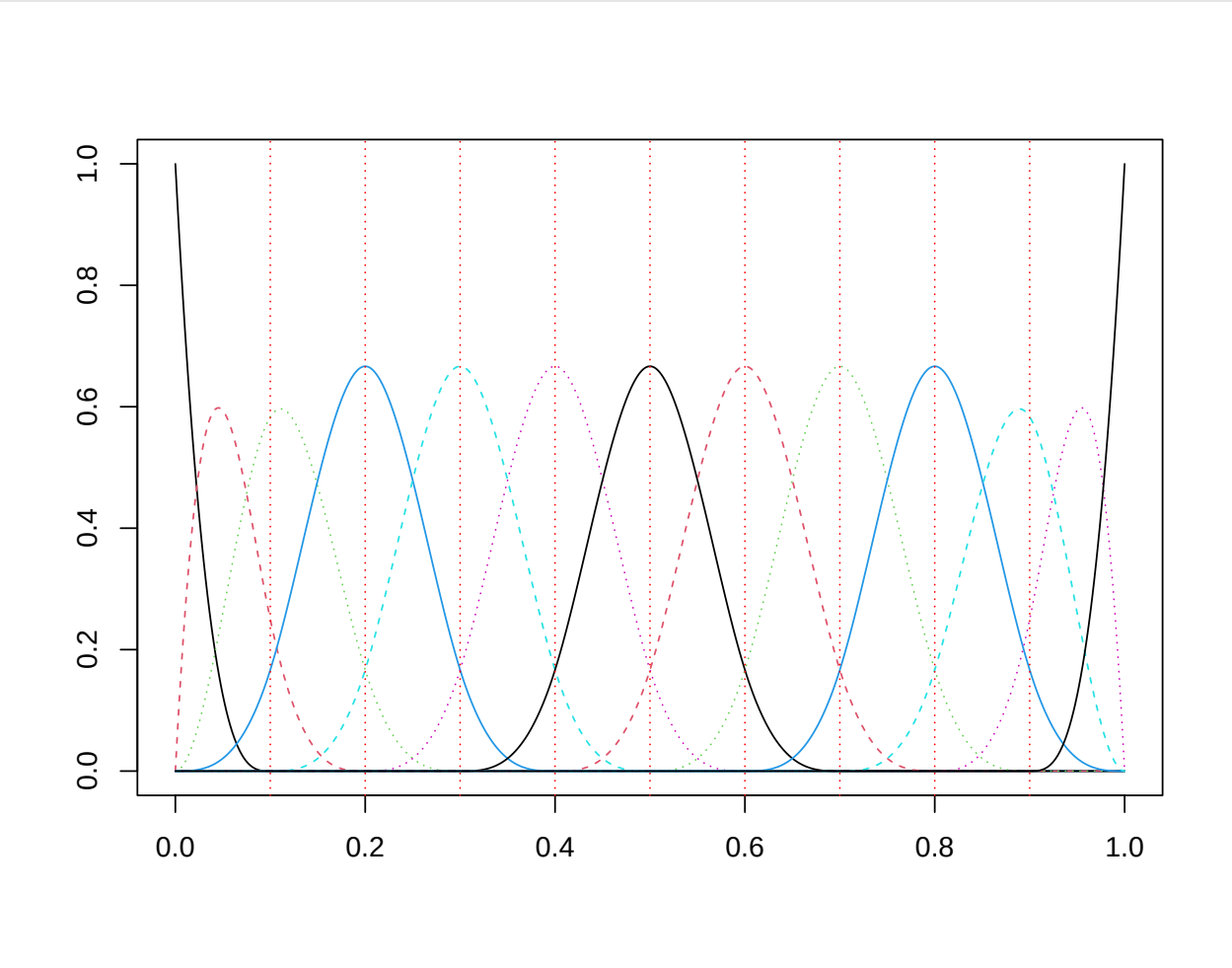

Figure: The thirteen basis functions defining and order four splines with nine interior knots.

A sample of points over T

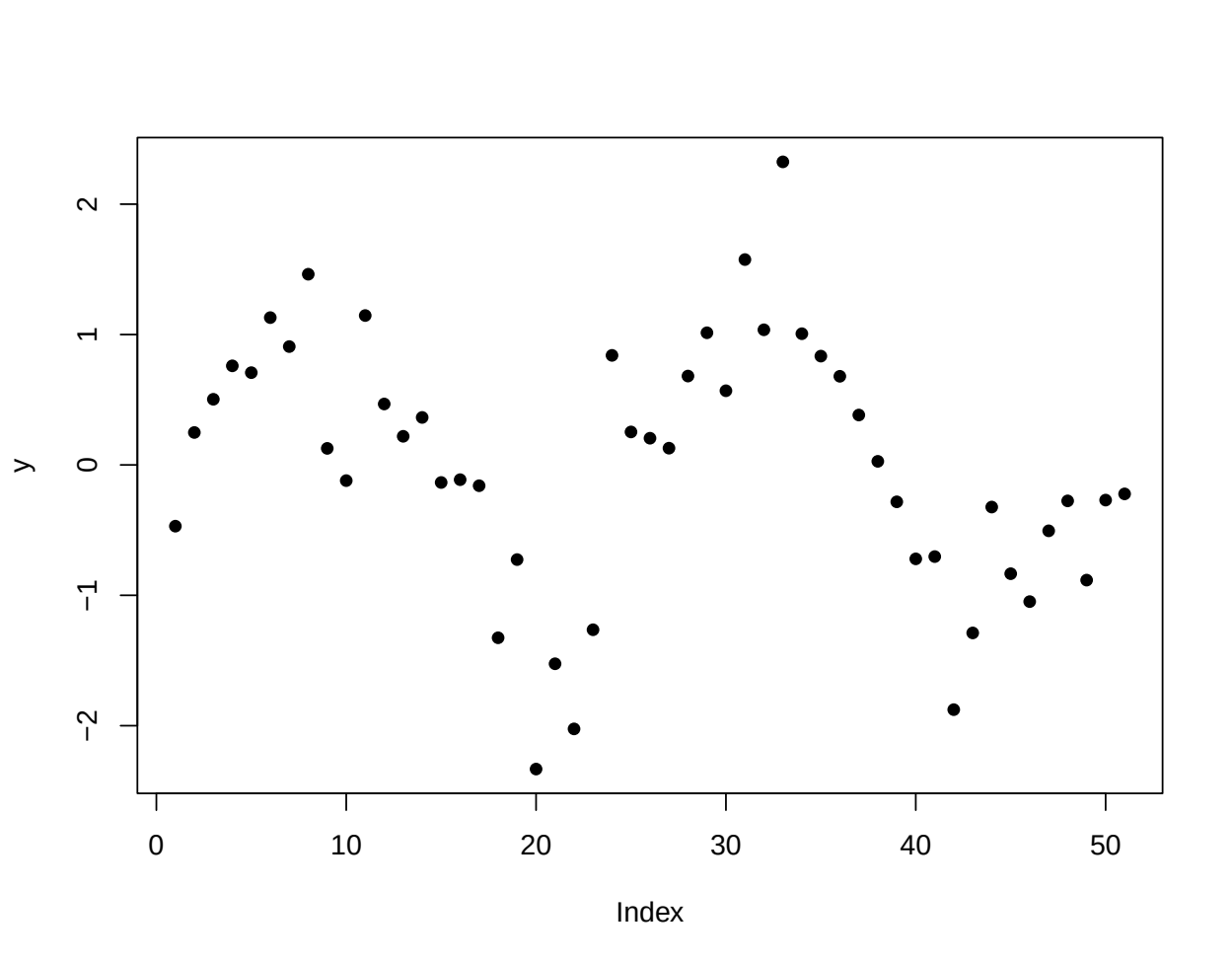

Figure: Sample from a functional data.

Smooth and continuous function over T

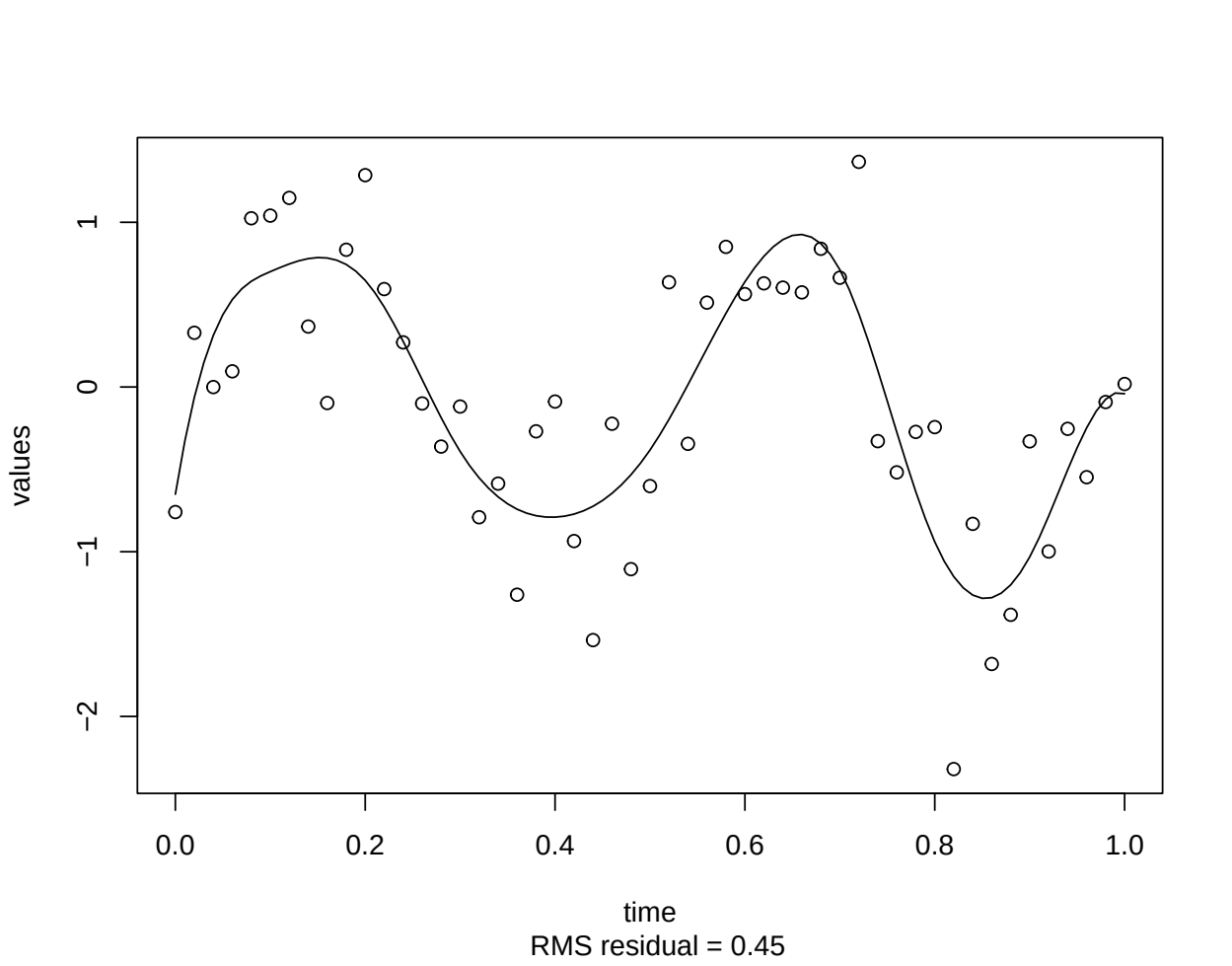

Figure: Smoothing the sample from a functional data.

Smooth and continuous function over T

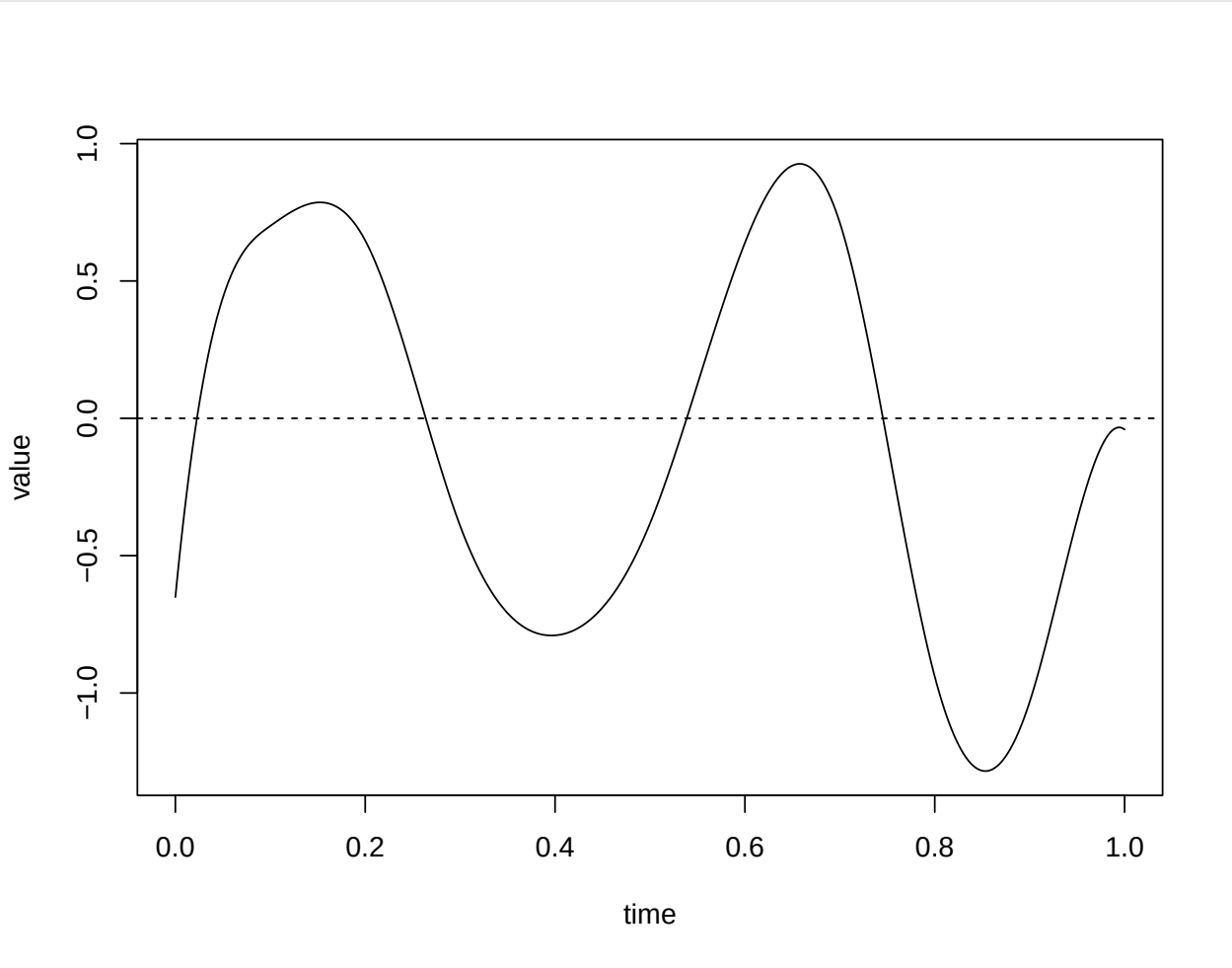

Figure: Keep the curve (smooth and continuous representation)

Functional data representation

- The data is now represented using the B-Spline basis functions (for instance with order 4 and 9 interior knots)
- This observation is later used in the analysis.
- With the following coefficients: [-0.65115973, 0.53578405, 0.70073762, 0.94566931, -0.59899313, -0.89844612, -0.54772926, 0.79263628, 1.21055785, -1.36245376, -1.40150503, 0.05021092, -0.04074864]

Functional data representation

Smoothing has multiple benefits:

- Dimension reduction
- Easy-to-compute derivatives
- Manages irregularly-spaced data
- *Smooth out* measurement errors

Irregularly-space data

What is irregularly-spaced data:

- Irregularly-collected data.
- Different subject/repetitions i are collected at different times $t \in [0, T]$
- The can also be collected a different number of times.

Irregularly-space data

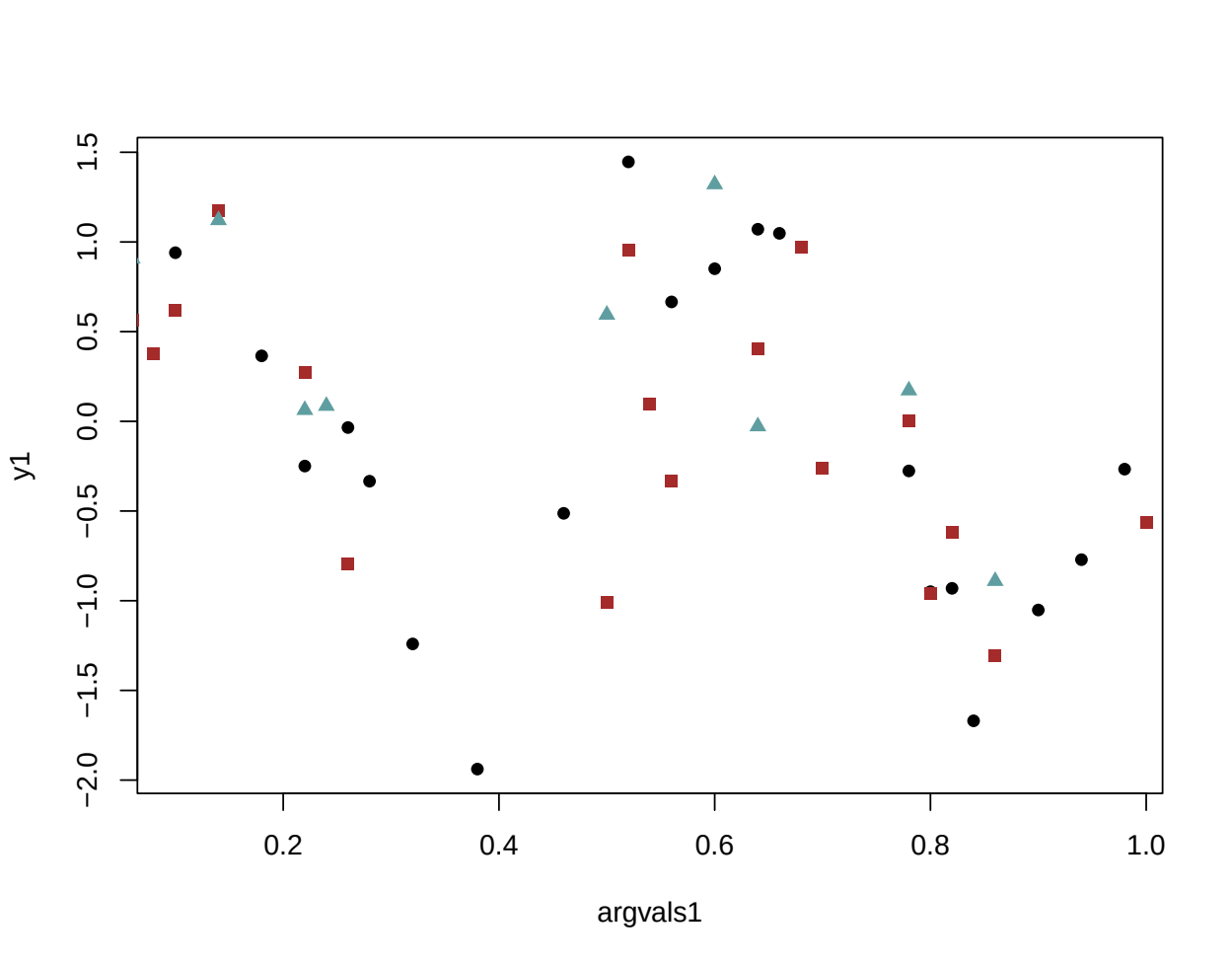

Figure: Irregularly-space functional data.

Problems in functional data analysis

1. Represent the data in a way that aid further analysis.
2. Display the data so as to highlight various characteristics.
3. Study important sources of variation among the data. (Functional principal component analysis)
4. **The regressions of functional data onto scalar variable and vice versa.**

Functional regression

- We cannot simply treat the observed points over $[0, T]$ as scalar and use regular regression.
- Different observations might observe data at different moments.
- Different observations might be observed a different number of times.
- Observations at time points close to another are related.

Traditional Regression cannot be directly applied.

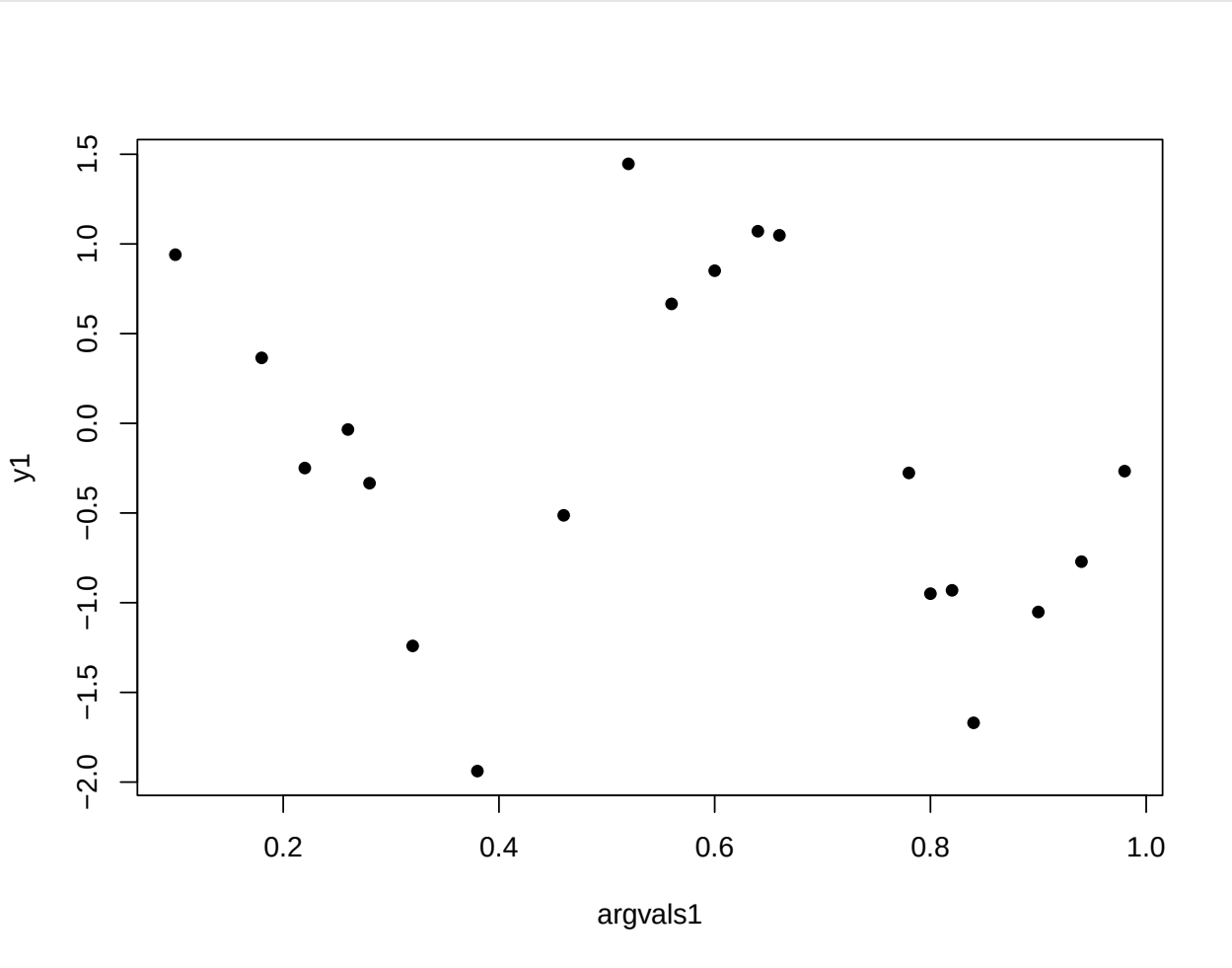

Figure: Sample from a functional data.

Traditional Regression cannot be directly applied.

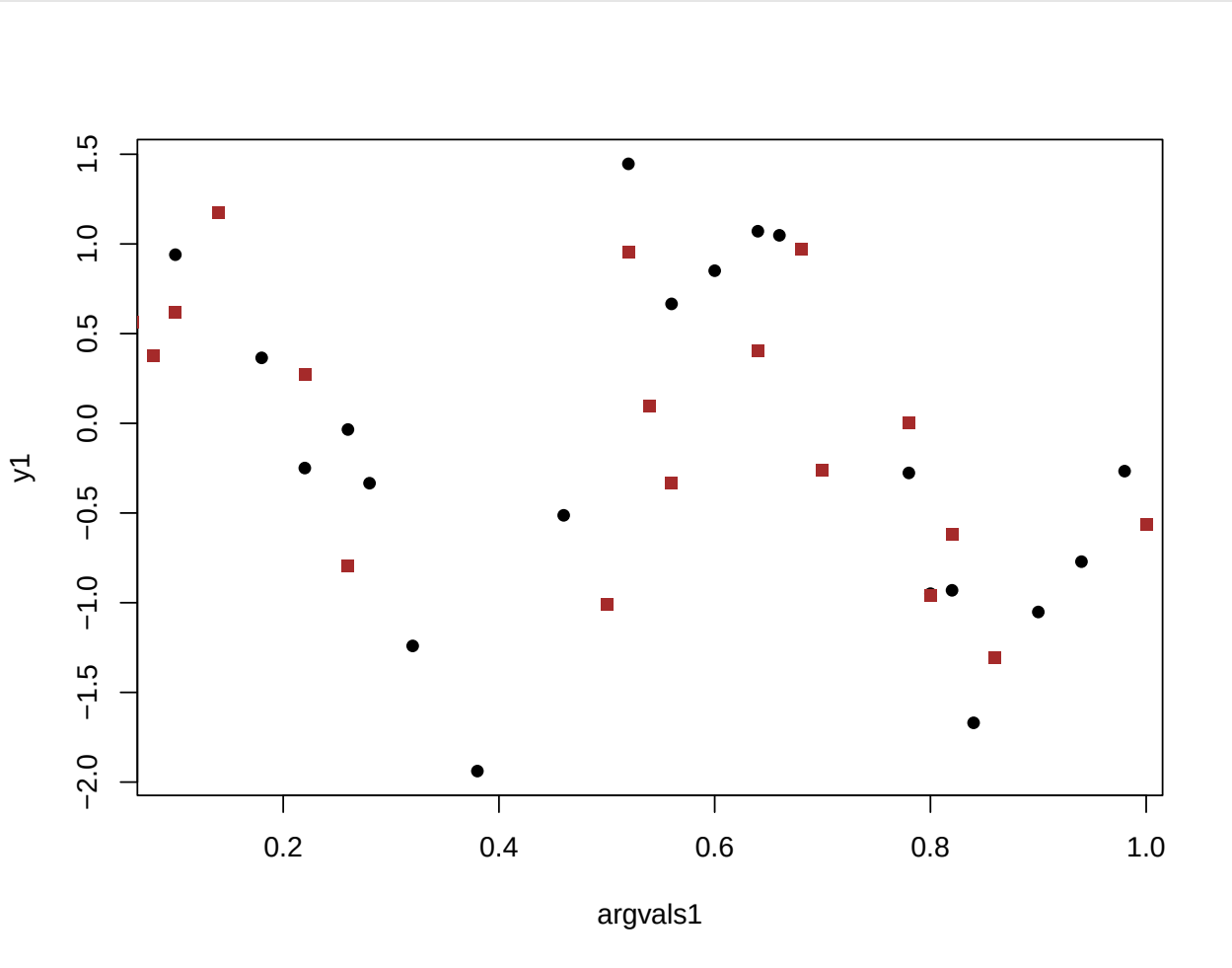

Figure: Sample from a functional data.

Traditional Regression cannot be directly applied.

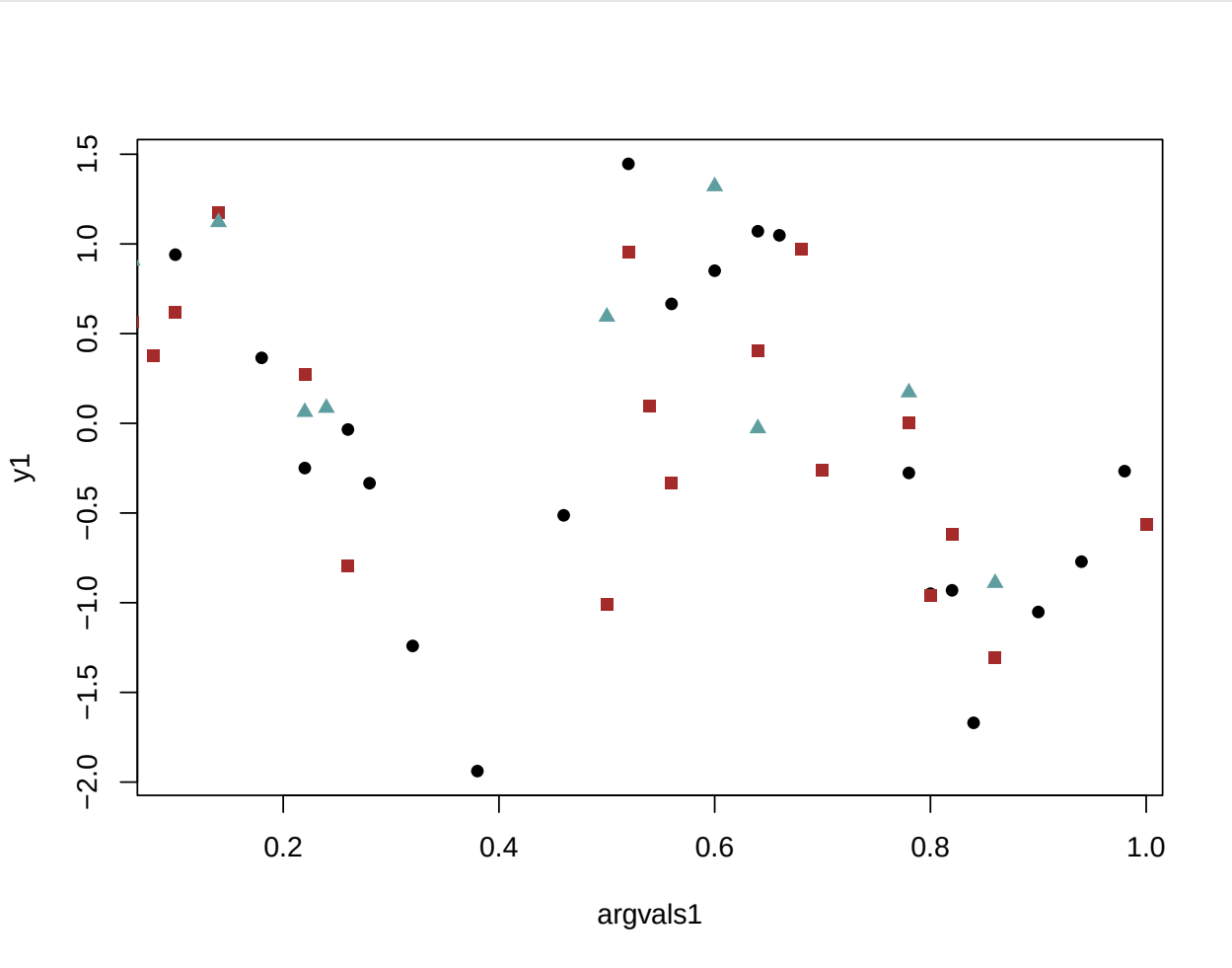

Figure: Sample from a functional data.

Functional regression problems

- Function on scalar (FoS) regression.
- Scalar on functional (SoF) regression.
- Function on function (Fof) regression.

Function on scalar regression (FoS)

- Scalar predictors, functional response.
- Parameters are now functions that must be estimated for $t \in [0, T]$.
- The model takes the form $y(t) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j(t) + \varepsilon(t)$

Function on scalar regression (FoS)

- A common approach is to smooth the response, $y_i(t) = \sum_{k=1}^{K} c_k^i B_k(t)$
- The we regress the coefficients c_k onto the predictors x.
- Thus we learn a matrix of parameters B such that $\mathbf{c} = \mathbf{x}B$ using least square.
- This means, doing two steps of least square sequentially.

Scalar on function regression (SoF)

- Functional predictors, scalar response.
- We take the inner product between the functional predictor and a functional weight.
- The model takes the form: $y = \beta_0 + \int_T x(t)\beta(t)dt + \varepsilon$
- A bit more complicated than FoS regression.

Scalar on function regression (SoF)

- A simple solution is to express the parameter $\beta(t)$ using a basis expansion: $\beta(t) = \sum_k c_k B_K(t)$
- $Y = \beta_0 + \sum_k c_k \int_T x(t) B_K(t) dt + \varepsilon$
- The parameters are now the basis coefficients (c_k) and they can be estimated provided a numerical solution for $\int_T x(t) B_k(t)$ for all k.

Functional regression

- Those are not the only way to proceed.
- Our proposed method are inspired by these.

Neural network architectures for functional data analysis

Integrating deep learning into functional data analysis

- We seek solution to solve all three regression problems described.
- We seek ways that functional data can be integrated, either as input or output, in deep learning models.
- We focus on designing input and output layers that can be connected to already existing deep learning architecture (CNN, LSTM, RNN, etc...)
- Using the functional data as it is collected,
- but considering the smooth and continuous assumption.

Functional Output Layer

- We want to design a functional output layer for neural networks (NN).
- To integrate our model into currently existing deep learning architecture, we have to make sure it can be trained using back-propagation.
- The predictors entering the model can be of any form.
- We propose to first output k basis coefficients, and then through a deterministic layer we reconstruct the functional response.

Functional Output Layer

- The functional data that we have is the following:
- We know $\mathbf{t}^i = [t_1^i, t_2^i, ..., t_m^i]$ the vector of time points when the ith subject has been observed,
- and we have $\mathbf{y}_i = [y_i(t_1), y_i(t_2), ...y_i(t_m)]$.

Functional Output Layer

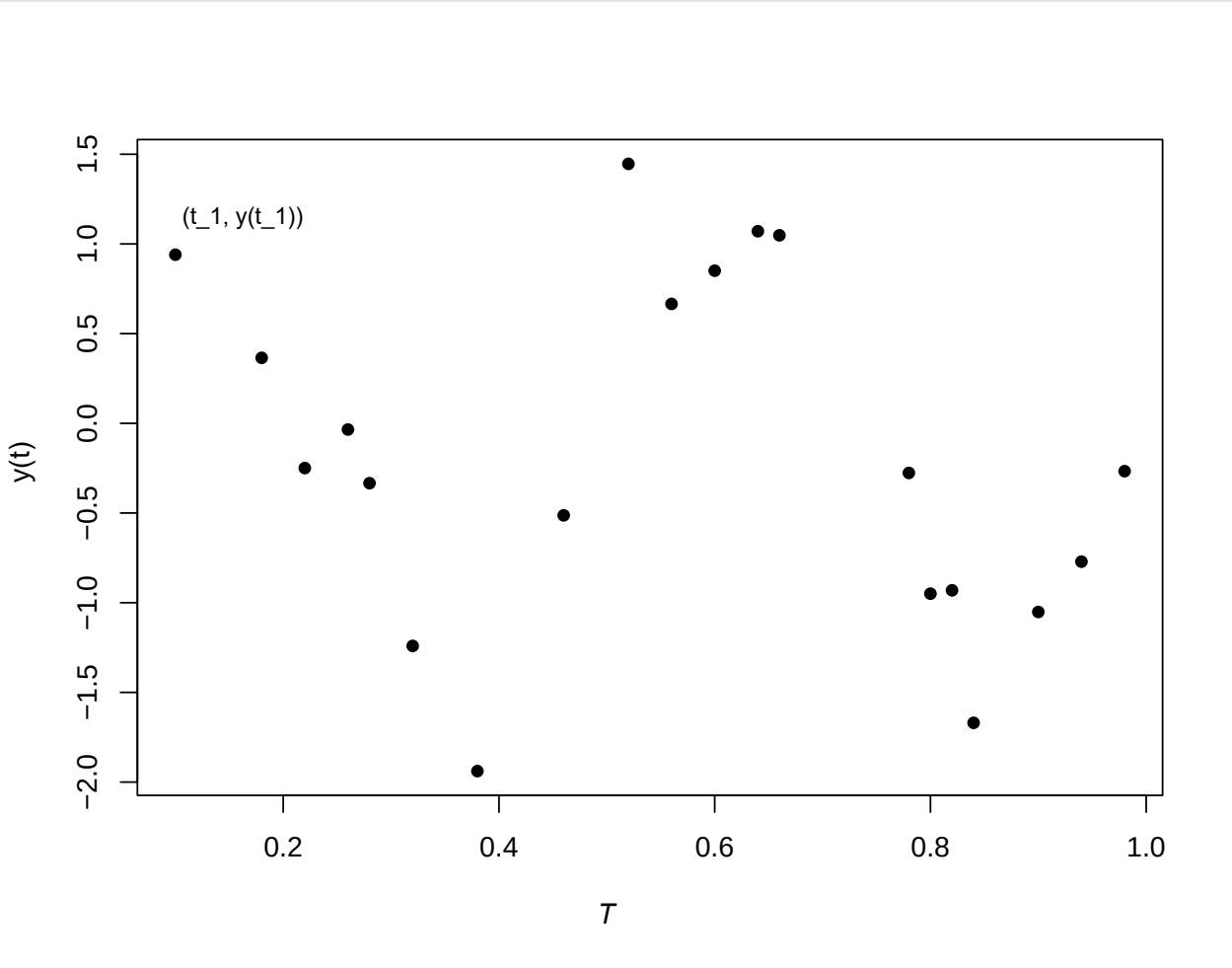

Figure: Sample from a functional data.

Functional Output Layer

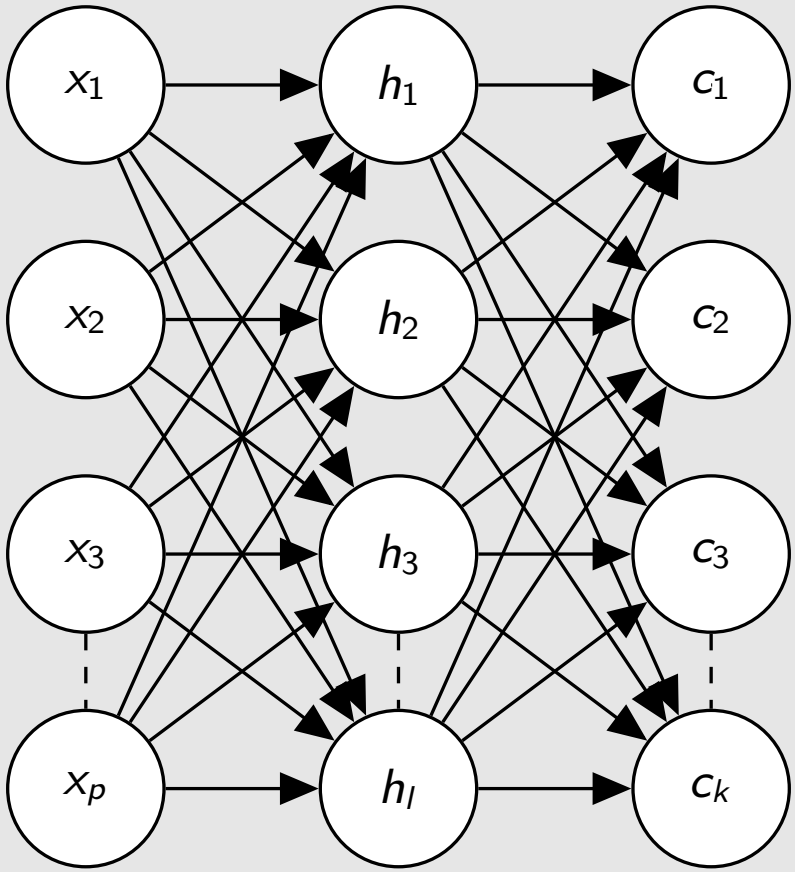

Figure: A 1-hidden layer NN to predict coefficients.

Functional Output Layer

- We want to train the model using the data $\mathbf{y}_i$,
- but we have a NN that outputs $\hat{\mathbf{c}}$: $\text{NN}(\mathbf{x}_i) = \hat{\mathbf{c}}^i$.
- However, we can construct a predicted response $\hat{y}_i(t) = \sum_{k=1}^{K} \hat{c}_k^i B_k(t)$.

Functional Output Layer

- We can evaluate $\hat{y}_i(t)$ at $\mathbf{t}^i$: $\hat{y}_i(\mathbf{t^i}) = [\hat{y}_i(t_1^i), \hat{y}_i(t_2^i), ...\hat{y}_i(t_m^i)]$ $= [\sum_{k=1}^{K} \hat{c}_k^i B_k(t_1^i), \sum_{k=1}^{K} \hat{c}_k^i B_k(t_2^i), ..., \sum_{k=1}^{K} \hat{c}_k^i B_k(t_m^i)]$
- We propose to construct $\mathbf{B}_{K\times m}$ where $B_{k,j} = B_k(t_j)$.
- Thus $\hat{y}_i(\mathbf{t^i}) = \widehat{\mathbf{c}}_{1\times K}^i \times \mathbf{B}_{K\times m}$.
- That way we can train the model using the following objective function : $L_Y = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} [y_i(t_j) - \hat{y}_i(t_j)]^2$

How can you train the model with $\hat{y}$? It is not the output of the NN!

Functional Output Layer

We can use custom operations that are not part of typical NN model, such as the matrix multiplication $\widehat{\mathbf{c}}^i_{1\times K} \times \mathbf{B}_{K\times m}$ because this operation is differentiable.

We only need to use the pytorch or keras *matmult* operator in this case.

Functional Output Layer

- By doing this 2-layers process we smooth the response and regress the coefficients onto x jointly. (a single optimization problem)
- We can train the model using the response variable collected $\mathbf{y}_i$
- $L_Y = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} [y_i(t_j) - \hat{y}_i(t_j)]^2$
- This can be thought of as adding a deterministic layer after the coefficient layer.

Functional Output Layer

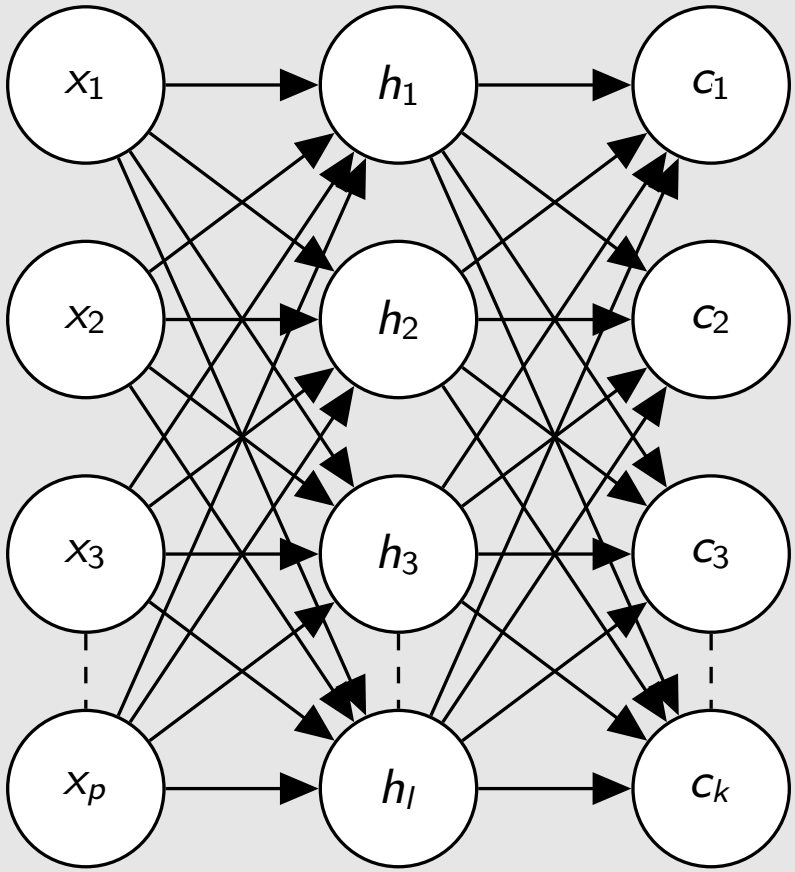

Figure: A 1-hidden layer NN to predict coefficients.

Functional Output Layer

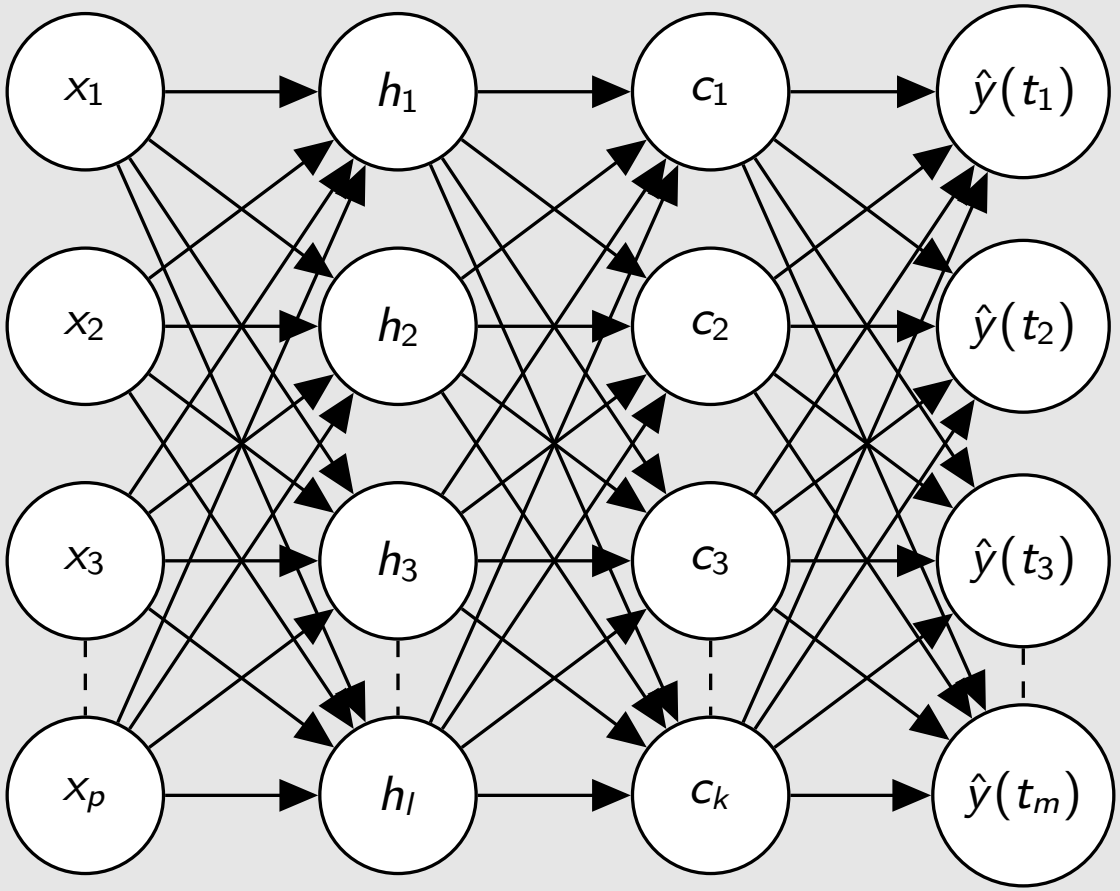

Figure: The model proposed can be perceived as adding a deterministic layer to the model.

Functional Output Layer

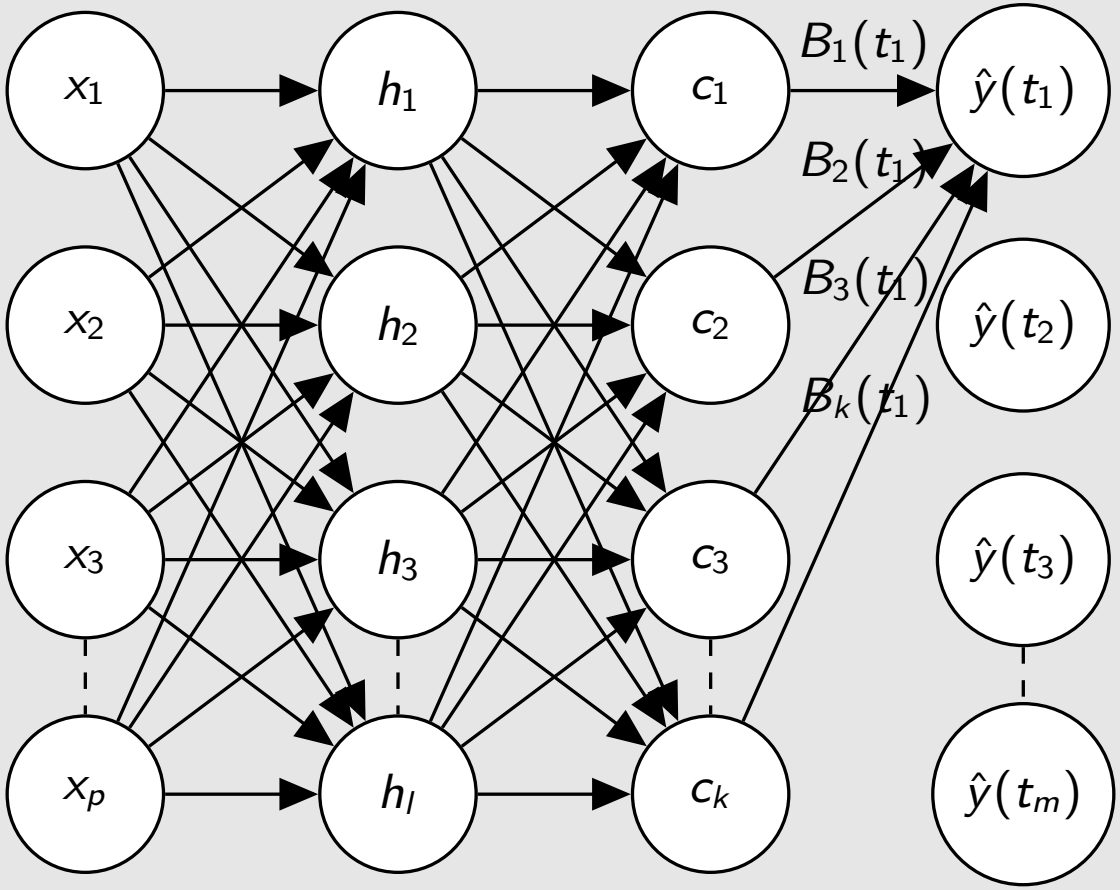

Figure: The model proposed can be perceived as adding a deterministic layer to the model.

Functional Output Layer

- This creates a continuous and smooth prediction ($\hat{y}(t)$ exists for every $t \in [0, T]$/interpolates)
- No need to first smooth the data.
- Learning a basis representation of the functional data is beneficial to further address common FDA concerns:
- Irregularly spaced data and smoothness regularization (coherent with literature).

Functional consideration: irregularly spaced data

- Suppose the time points $\mathbf{t}_i = [t_1^i, t_2^i, ..., t_{m_i}^i]$ observed for subject i are different than time points $\mathbf{t}_j = [t_1^j, t_2^j, ..., t_{m_j}^j]$ observed for subject j.
- Our configuration of NN, that outputs $\mathbf{c}$ instead of $\mathbf{y}$; we simply evaluate the basis functions $B_k(t)$ at different time points for different observations.
- Different subjects can have different numbers of observed time points at different *moments* but they all contribute in the prediction of the same coefficients $\mathbf{c} = NN(\mathbf{x})$

Functional consideration: irregularly spaced data

- We first produce a matrix $\mathbf{B}$ at all time across all data points ($\mathbf{t} = \cup_{i=1}^{n} \mathbf{t}_i$)
- Then we bring to 0 the contribution of unobserved time points in the objective function:

$$L_{\boldsymbol{Y}_{\text{irr}}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{m_i} \left(y_i(t_j) - \hat{y}_i(t_j)\right)^2 \cdot 1\left(y_i(t_j) \text{ is observed}\right), \tag{1}$$

- We can train using this because it is differentiable w.r.t. the coefficients c_k.

Functional consideration: roughness penalty

- To ensure the learned representation is smooth, it is common to regularize the second derivative $\int_{\mathcal{T}} \left(\frac{d^2 \hat{y}(t)}{dt^2} \right)^2 dt$
- In our case, this would mean using objective functions such as:

$$L_{pen}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \Big(\sum_{j=1}^{m} (y_i(t_j) - \hat{y}_i(t_j))^2 + \lambda \int_{\mathcal{T}} \left(\frac{d^2 \hat{y}(t)}{dt^2} \right)^2 dt \Big). \tag{2}$$

(3)

Functional consideration: roughness penalty

- We cannot back-propagate the gradient through the integral
- Because we are able to generate $\hat{Y}_i(t)$, we approximate this integral with as many points as we want:

$$L_{\boldsymbol{pen}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \Big(\sum_{j=1}^{m} (y_i(t_j) - \hat{y}_i(t_j))^2 \tag{4}$$

$$+ \frac{\lambda T}{J-1} \sum_{j=2}^{J} \left(\frac{d^2 \hat{y}_i(t_j)}{dt_j^2} \right)^2, \tag{5}$$

where

Functional consideration: roughness penalty

- What about those 2nd order derivatives ?

$$\hat{y}_i(t) = \sum_{k=1}^{K} c_k B_k(t)$$

$$\Rightarrow \frac{d^2 \hat{y}_i(t)}{dt^2} = \sum_{k=1}^{K} c_k \frac{d^2 B_k(t)}{dt^2}, \tag{6}$$

- There are derivatives of basis functions: known values
- We can train using this regularization because it is differentiable w.r.t. the coefficients c_k.

Functional Input Layer

- To process functional input we need to learn a functional weight/parameter $\beta(t)$.
- $y = \beta_0 + \int_T x(t)\beta(t)dt + \varepsilon$
- Our concept is similar to the functional output layer.

Simple MLP with scalar input

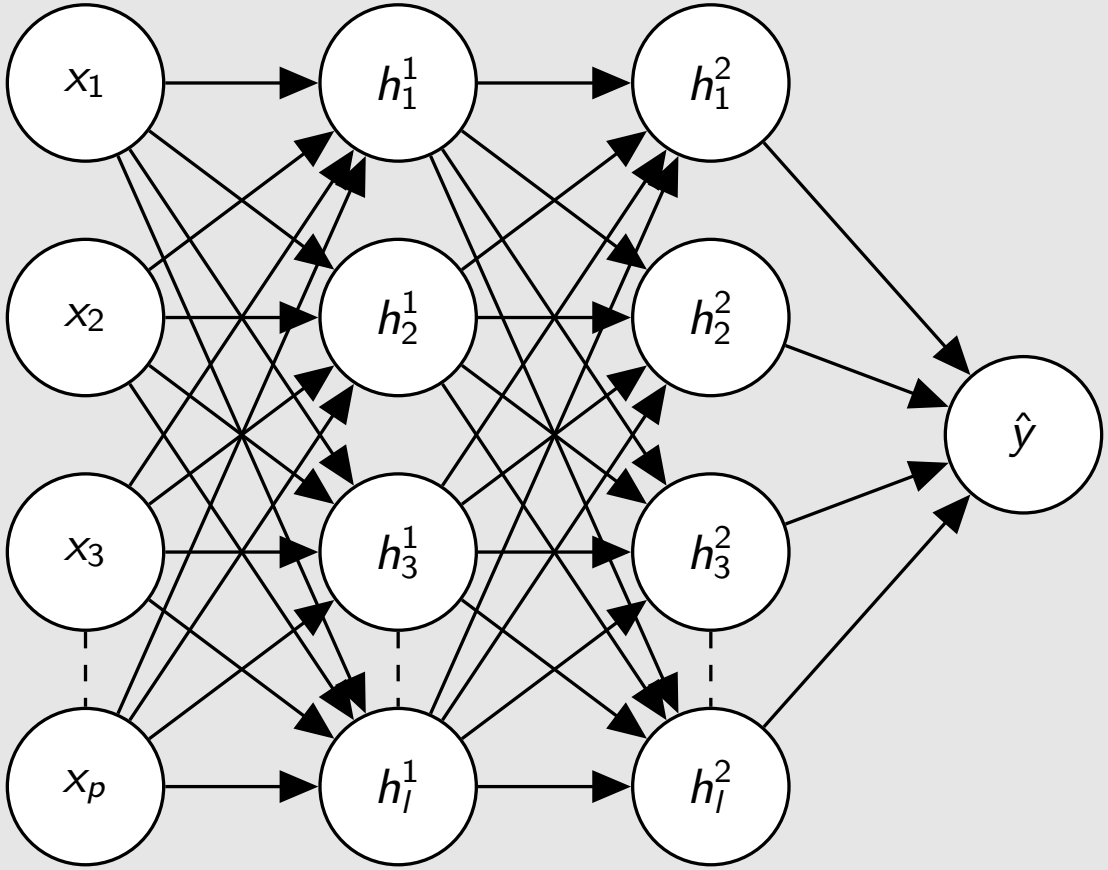

Figure: Simple MLP with scalar input and output.

Functional Input Layer

- $h_l = g(\sum_{j=1}^{p} x_j \beta_{l,j})$ where g is the activation function.
- $\mathbf{h} = g(\beta \mathbf{x})$.
- Consequently we seek to learn a functional equivalent: $h_l = g(\int_T x(t)\beta_l(t))$
- To do so, we propose to use a parametric representation of the functional weight $\beta(t)$ through basis expansion.

Functional Input Layer

- Elements of the first hidden layer are of the form: $h_l = g(\int_T x(t)\beta_l(t))$
- with $\beta_l(t) = \sum_{k=1}^{K} c_{l,k} B_l(t)$
- This means the coefficients $\mathbf{c}$ are the parameters we are trying to learn in this layer through back-propagation (learning the functional weight).
- As long as all the operation involving $\mathbf{c}$ are differentiable, we should be able to train such model.

Functional Input Layer

$$h_l = g(\int_T x(t)\beta_l(t)) \tag{7}$$

$$= g(\int_T x(t)\sum_{k=1}^{K} c_{l,k}B_k(t)) \tag{8}$$

$$= g(\sum_{k=1}^{K} c_{l,k}\int_T x(t)B_k(t)) \tag{9}$$

$$= g(\sum_{k=1}^{K} c_{l,k}f_k) \tag{10}$$

where $f_k = \int_T x(t)B_k(t)$. We can learn c, by constructing a simple fully connected layer mapping $\mathbf{f}$ to $\mathbf{h}$:
$\mathbf{h} = g(\mathbf{Cf})$, where $\mathbf{C}$ is a l by K matrix of coefficients.

Simple MLP with features f as input

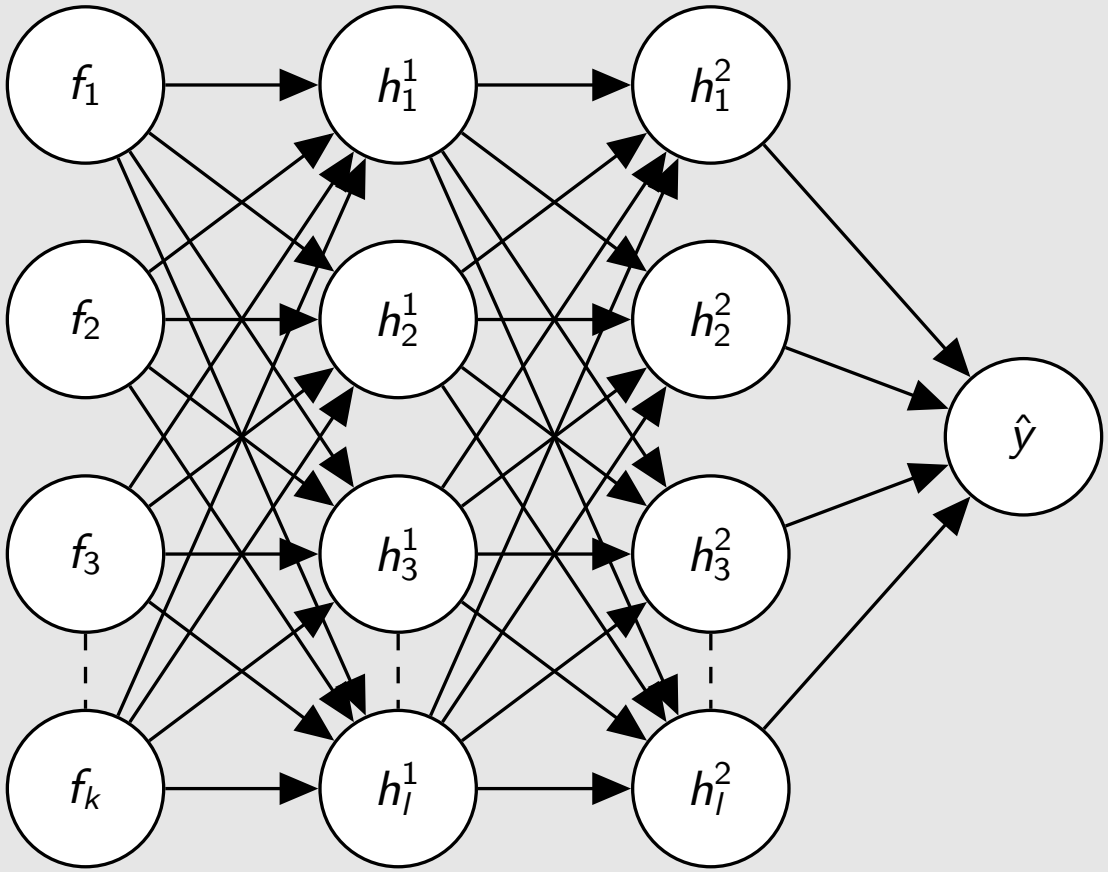

Figure: Simple MLP with scalar input and output.

Simple MLP with features f as input

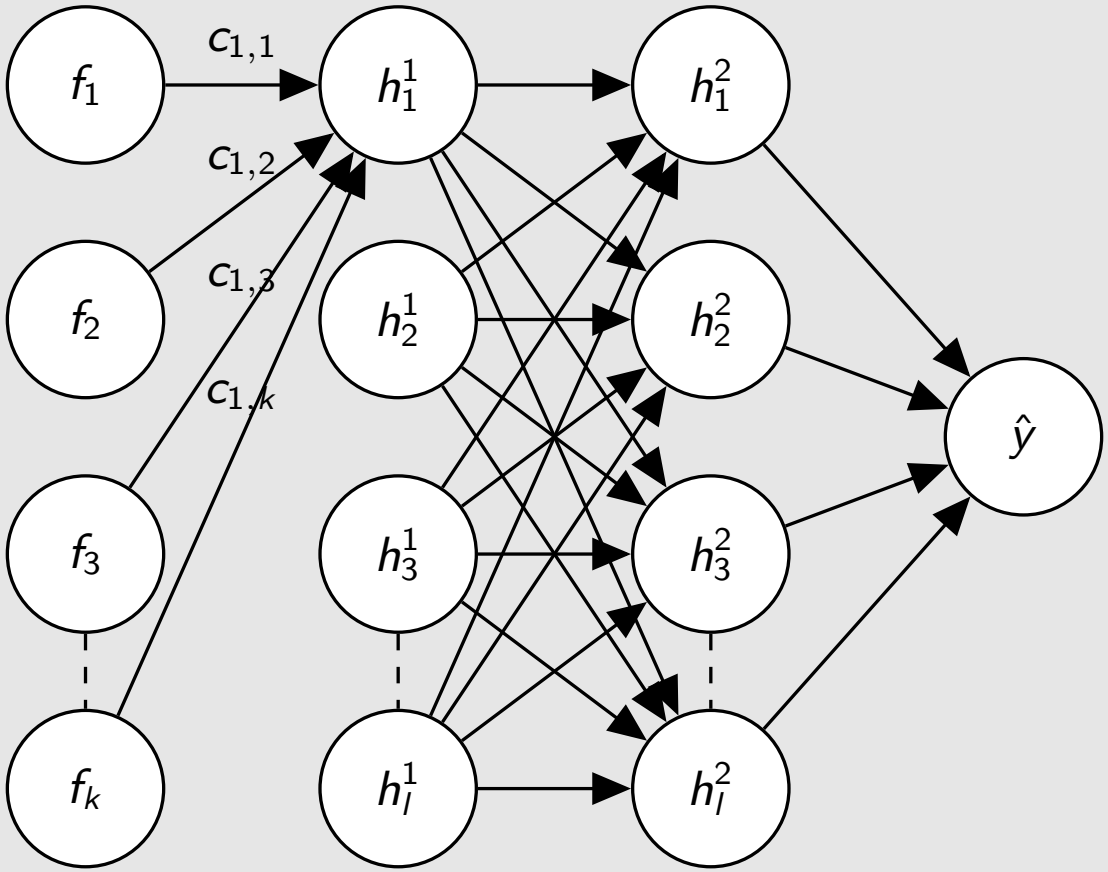

Figure: Simple MLP with scalar input and output.

Functional Input Layer

- $f_k = \int_T x(t)B_k(t)$.
- $x(t)$ is not observed as a function, but as a collection of points over $[0, T]$.
- We propose to directly estimate the integral with a summation.
- We propose $f_k = \sum_{j=1}^{m} w_j x(t_j)B_k(t_j)$ (numerical approximation).
- We can perceive this first step as a deterministic layer since $\mathbf{f}$ is a just a linear combination of $x(t)$

Functional Input Layer

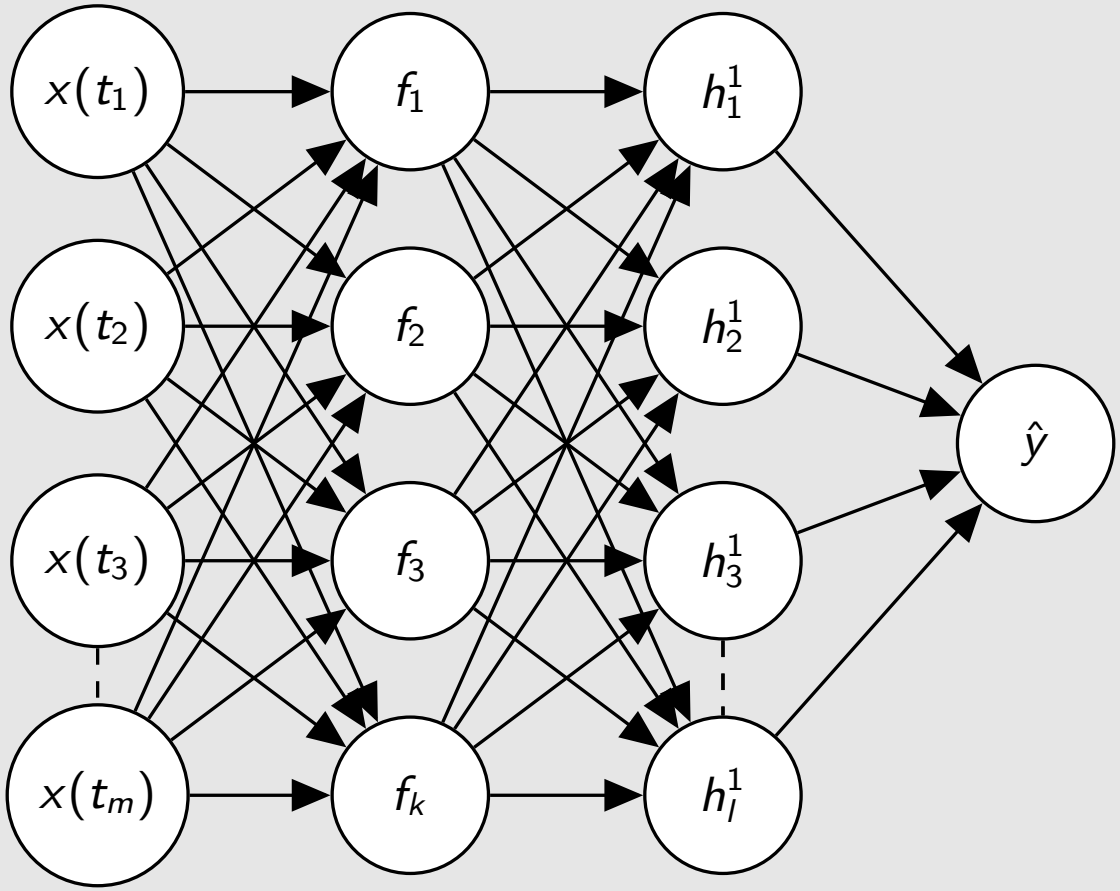

Figure: Simple MLP with scalar input and output.

Functional Input Layer

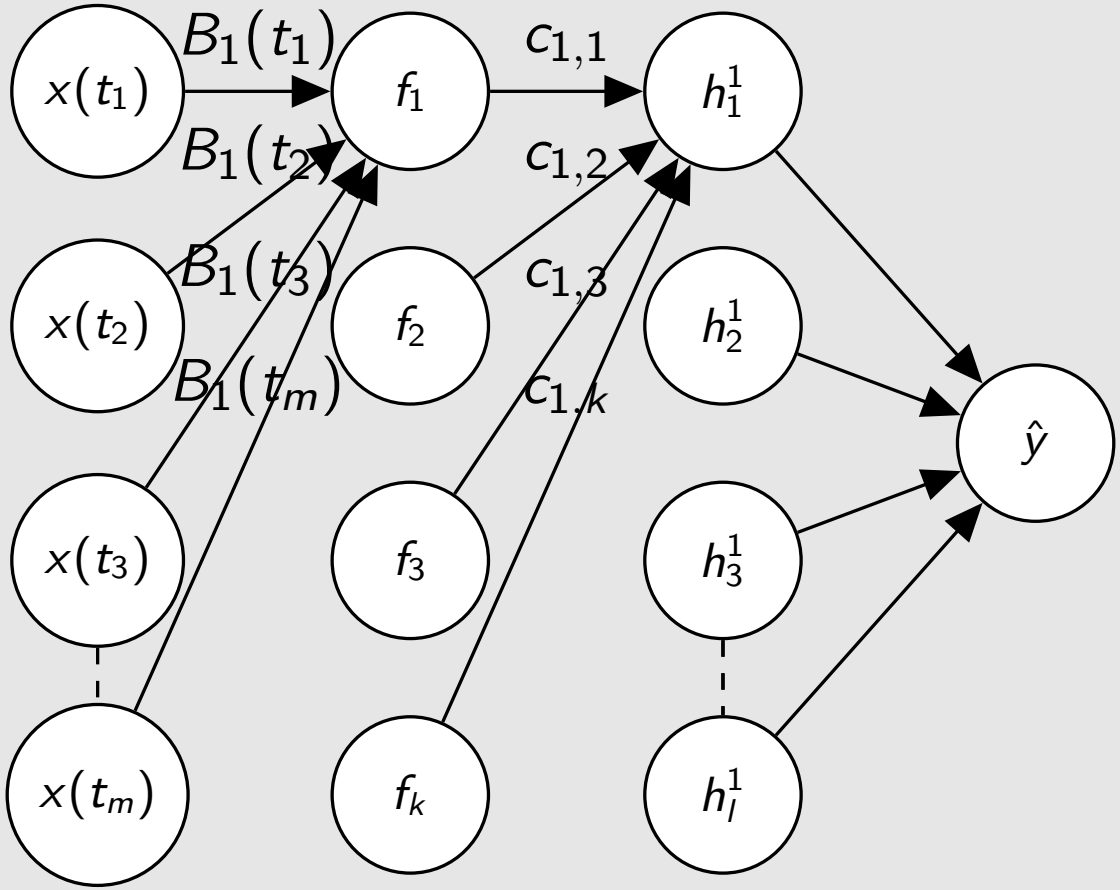

Figure: Simple MLP with scalar input and output.

Functional consideration: irregularly spaced data.

- The use of basis expansion solves the problem again.
- The first parametric layer (the one containing $\mathbf{c}$) is connected to the features $\mathbf{f}$.
- The observations with different time points all contribute to the same parameters.
- The difference between times points is *taken care* by the deterministic layer.

Proposed models and results

Function on scalar regression

- We implemented FoS models for simple regression problems.
- Its strength lies in the ability of neural networks to capture non-linear relations.
- We tested our model on multiple simulated data sets and a real data set.

Wu, S., Beaulac, C. and Cao, J. Neural Networks for Scalar Input and Functional Output, Statistics and Computing, (33), 118, 2023.

Function on scalar regression : results

- To enforce non-linearity between scalar predictors $\mathbf{x}$ and functional response $y(t)$, we made the relation between x and basis coefficients of $y(t)$ non-linear.
- We are able to retrieve the polynomial relation with our proposed model:

Function on scalar regression : results

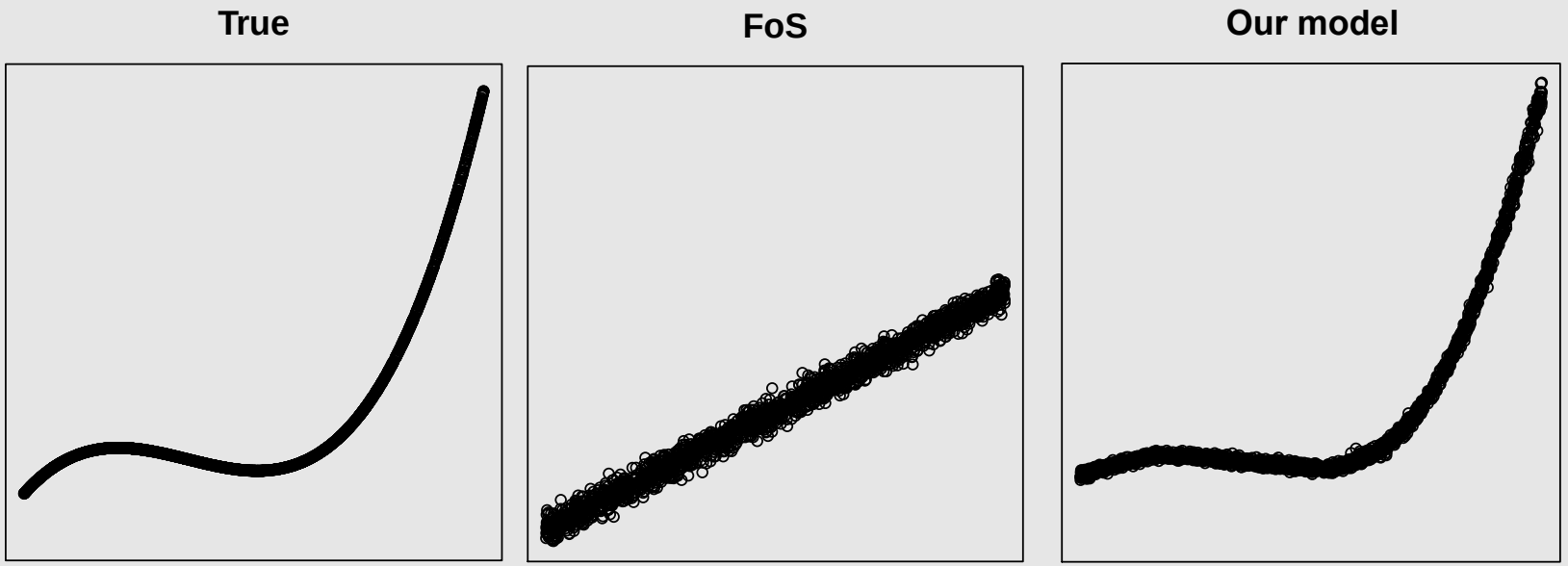

Figure: Basis coefficient as a function predictor X_1

Function on scalar regression : results

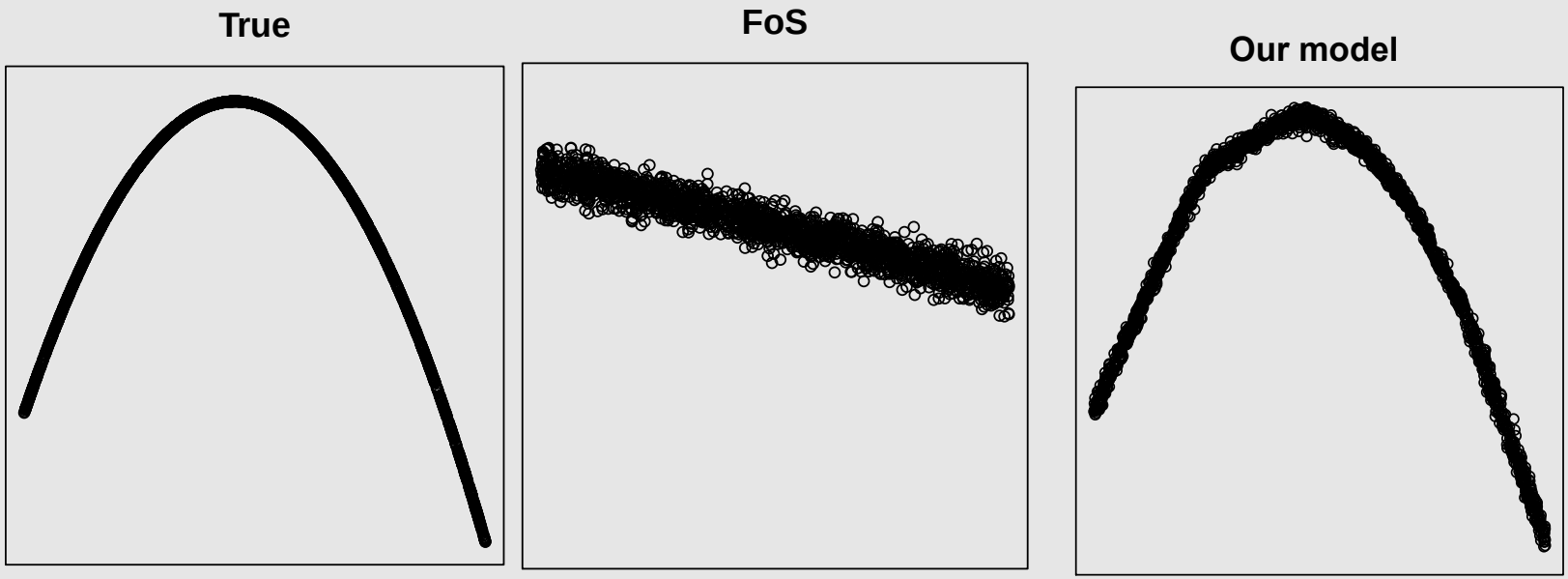

Figure: Basis coefficient as a function predictor X_2

Function on scalar regression : results

Table of MSEs of 20 random test sets. (Design 1)

Methods	FoS (Linear)	Our model (NN)
Mean	49.20	4.95
Std. Dev.	1.64	0.11
p-value	-	$<$2.2e-16

Function on scalar regression : results

Table of MSEs of 20 random test sets. (Design 2)

Methods	FoS (Linear)	Our model (NN)
Mean	4559.20	36.38
Std. Dev.	159.01	18.76
p-value	-	<2.2e-16 6

Function on scalar regression (linear design): results

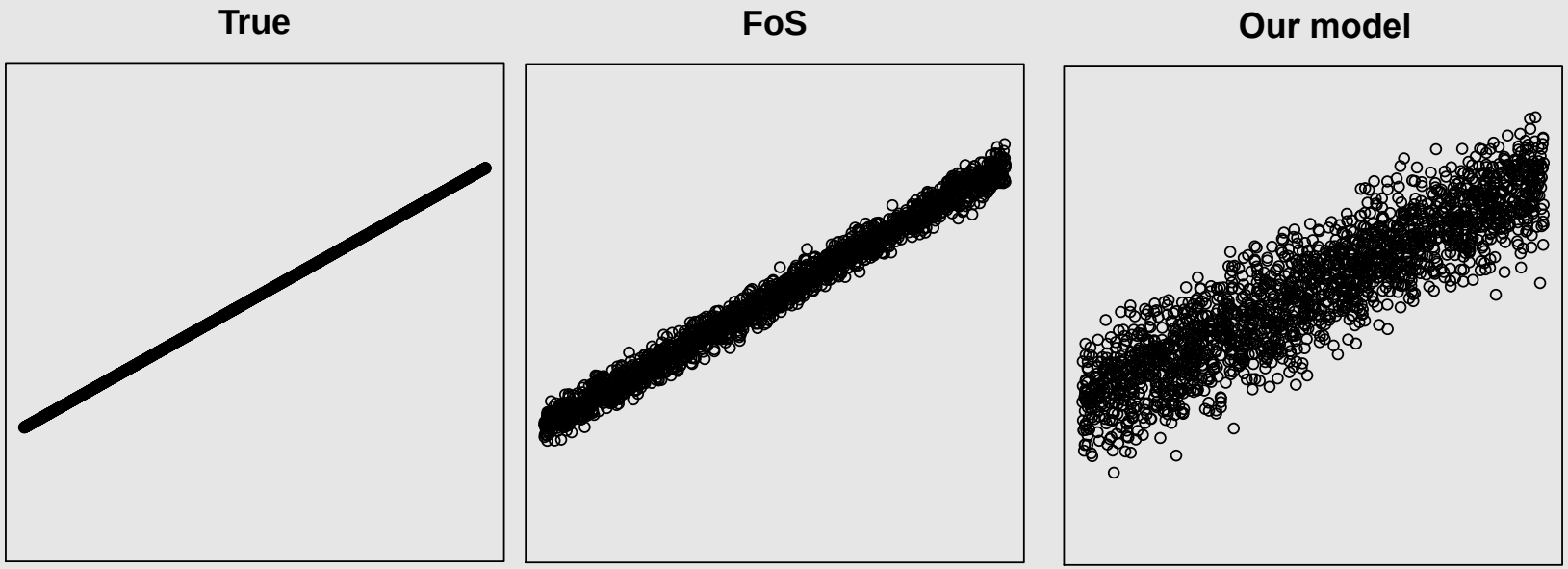

Figure: Basis coefficient as a function predictor X_2

Function on Scalar regression (linear design): results

Table of MSEs of 20 random test sets. (Linear design)

Methods	FoS (Linear)	Our model (NN)
Mean	4.01	4.07
Std. Dev.	0.05	0.05
p-value	-	7.8e-07

Functinal autoencoder : results

- We can use both the functional input layer and functional output layer to create a functional autoencoder.
- Autoencoders are non-linear generalization of PCA. The project the data to a lower dimension representation. We evaluate their performance by reconstruction error and the usefulness of the lower dimension representation.
- The functional autoencoder creates a lower-dimension representation and smooths the data simultaneously.

Wu, S., Beaulac, C. and Cao, J. Functional Autoencoder for Smoothing and Representation Learning, Under Review, 2023.

Functional autoencoder : results

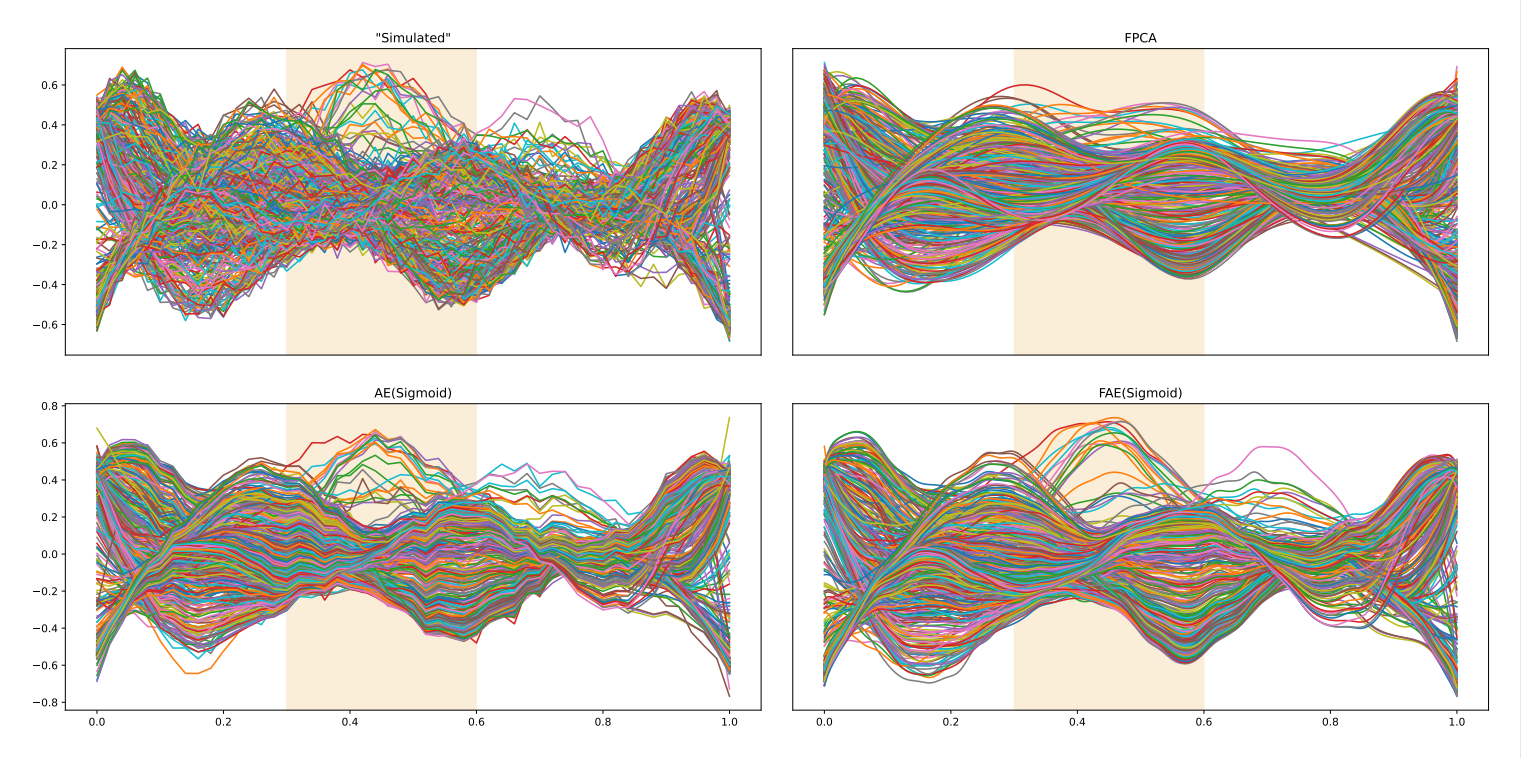

Figure: The simulated curves and the curves recovered by functional principal component analysis (FPCA), classic autoencoder with Sigmoid activation function (AE(Sigmoid)) and functional autoencoder with Sigmoid activation function (FAE(Sigmoid))

Functional autoencoder : results

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

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

Possible extensions

- In both cases, the addition of a single layer can translate functional data to a vector of scalar, such as those found in MLP.
- Thus, functional data can be integrated in any deep learning architecture.
- For instance, we can predict functional data using images with a convolution layers for input and our propose functional output layer.
- Can also be extended to multi-dimensional functional data.

Implementation

- At the moment, we have an R implementation of the output layer.
- We have an implementation of the input and output layer in Python.
- Needs some work to be user-friendly.

Current development

- Designing a new input layer architecture.
- Inspired by convolution neural network, preserve the *shape* of the data.
- We are working on a smooth tunable convolution layer using the dilation parameter as a way to create a collection of models ranging from a simple 1D Convolutional layer to the continuous convolution operator.

Conclusion

- We designed layers to input or output functional data in modern neural network architectures.
- It allows any form of functional data to be now part of larger multi-modality deep learning models.
- It respects the assumptions of functional data.

Conclusion

- It adds a layer of complexity and hyper parameters to adjusts.
- The implementations are not user-friendly.

I would love to answer your questions.

Ramsay, J. O. and Silverman, B. W. . Functional Data Analysis (Second Edition). Springer, 2005.

Morris, J. S. . Functional regression. Annual Review of Statistics and Its Application, 2(1):321–359, 2015.

Wu, S., Beaulac, C. and Cao, J. Neural Networks for Scalar Input and Functional Output, Statistics and Computing, (33), 118, 2023.

Wu, S., Beaulac, C. and Cao, J. Functional Autoencoder for Smoothing and Representation Learning, Under Review, 2023.

Beaulac, C. and Larchevêque, V. Smooth Tunable Convolution: A novel input layer architecture for functional data, in preparation.