Integrating Deep Learning into Functional Data Analysis

Cédric Beaulac

Université du Québec à Montréal

July the 13th 2023

This work has been done in collaboration Sidi Wu and Jiguo Cao at Simon Fraser University and with my current research intern Valentin Larcheveques from l'Université Montpellier.

Integrating Deep Learning into Functional Data Analysis

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

The talk

- What is functional data and what makes it different ?
- How to represent such data in a parametric way ?
- Regression of functional data.
- Adapting neural networks for functional data.
- Proposed models and impact.
- Current work.

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 $[0, T]$.
- Points belong in a shared space with no registration needed.
- This is different from time series; we are not trying to forecast the functional data and we have multiple repetition over T.

A sample of points over T

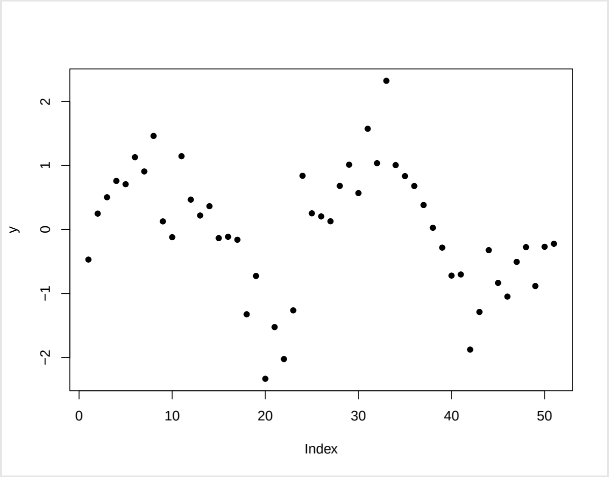


Figure: Sample from a functional data.

Multiple subjects: A sample of points over T

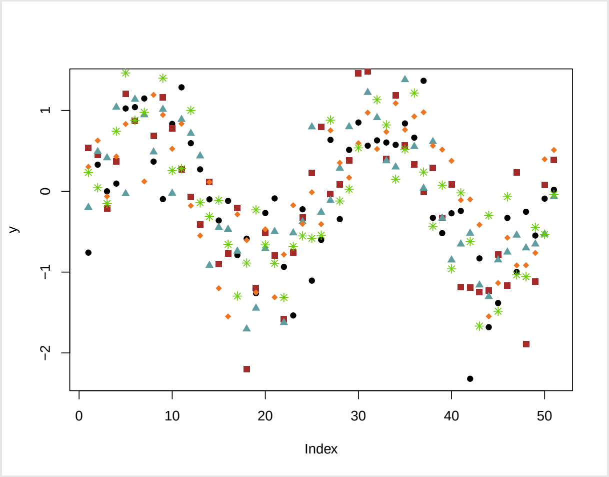


Figure: Sample from multiple functional data.

Exemple: real data set (El Nino)

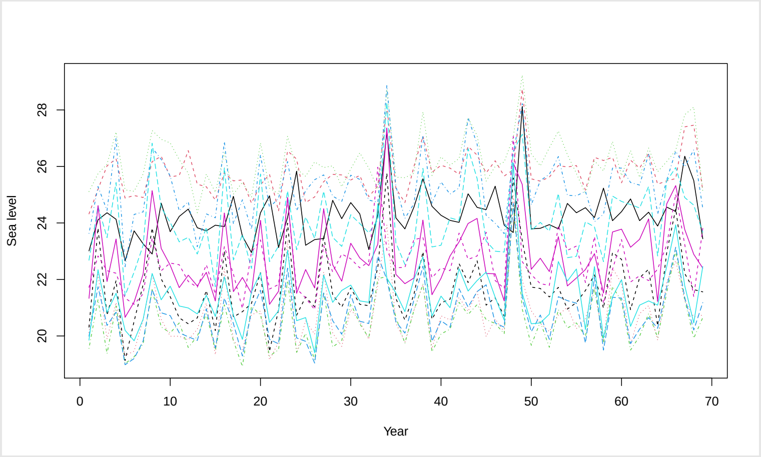


Figure: Yearly sea surface temperature.

Exemples

- Daily stock value of multiple companies.
- Engagement statistics collected with an application (usages of an app).
- Self-captured medical data. (Diabetics track their daily sugar-level)

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 observation 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

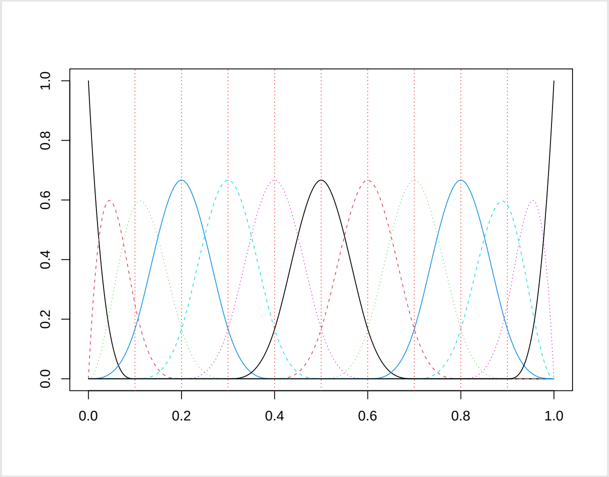


Figure: The thirteen basis functions defining and order four splines with nine interior knots.

A sample of points over T

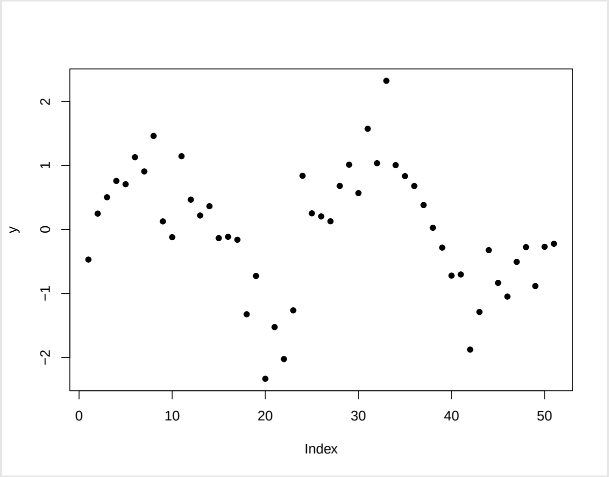


Figure: Sample from a functional data.

Smooth and continuous function over T

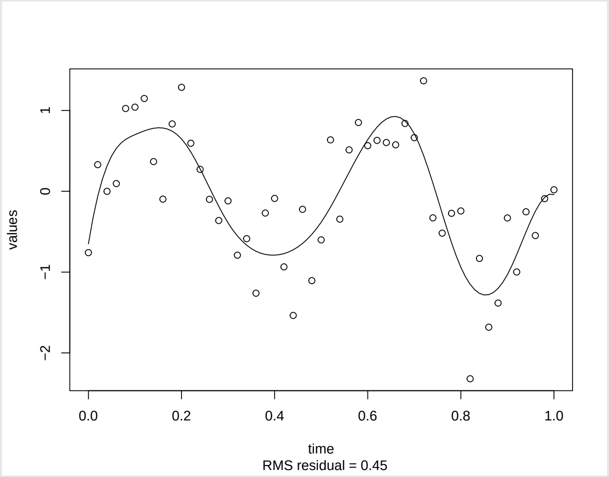


Figure: Smoothing the sample from a functional data.

Smooth and continuous function over T

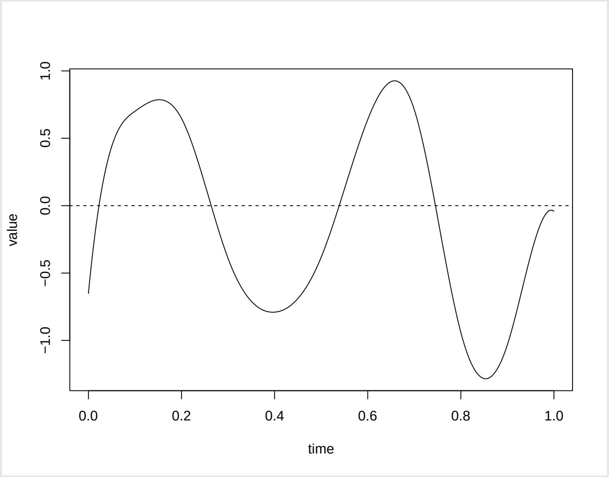


Figure: Keep the curve (smooth and continuous representation)

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]

Problems in functional data analysis

1. Represent the data in a way that aid further analysis.
2. Display the data so as to highlight carious characteristics.
3. Study important sources of variation among the data.
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.

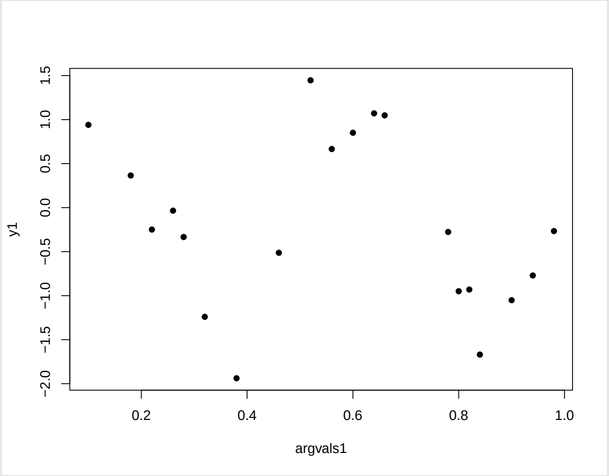


Figure: Sample from a functional data.

Traditional Regression cannot be directly applied.

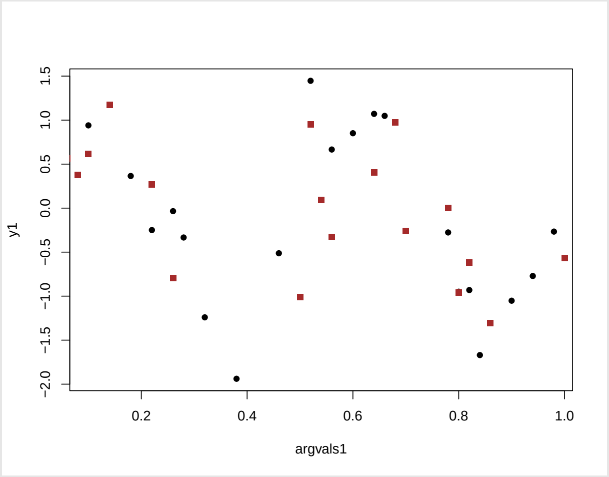


Figure: Sample from a functional data.

Traditional Regression cannot be directly applied.

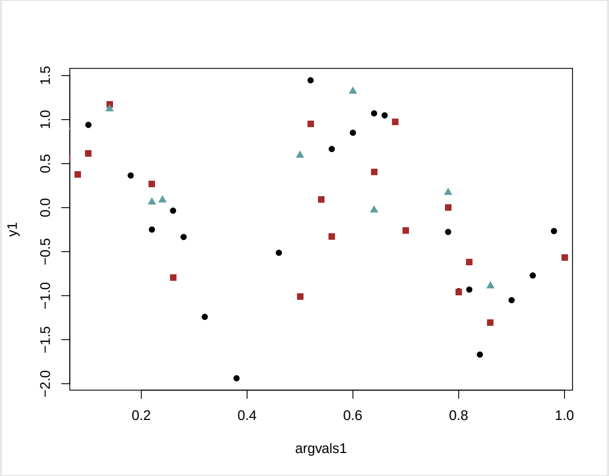


Figure: Sample from a functional data.

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.
- It is common to 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)

- An easier technique is to express the parameters $\beta(t)$ using a basis expansion: $\beta_j(t) = \sum_k c_k B_K(t)$
- $Y = \beta_0 + \sum_k c_k \int_T x(t) B_K(t) dt + \varepsilon$
- 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.

Integrating Deep Learning into Functional Data Analysis

Integrating deep learning into functional data analysis

- We seek solution to solve both regression problems described. (problem 4.)
- 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 regardless.

Functional Output Layer

- First, a functional output layer for neural networks (NN).
- Given $\mathbf{x}$ a p-dimensional vector of predictors.
- We define F a function mapping from $\mathbb{R}^p$ to $\mathbb{R}^K$ so that we can map $\mathbf{x}$ (predictors) to $\mathbf{c}$ (basis coefficient).
- This F is what used to be a linear combination, we propose to replace that with a NN.

Functional Output Layer

- Suppose $\mathbf{t}^i = [t_1^i, t_2^i, ..., t_m^i]$ is the vector of time points when the ith subject has been observed.
- Thus we have $\mathbf{y}_i = [y_i(t_1), y_i(t_2), ...y_i(t_m)]$

Functional Output Layer

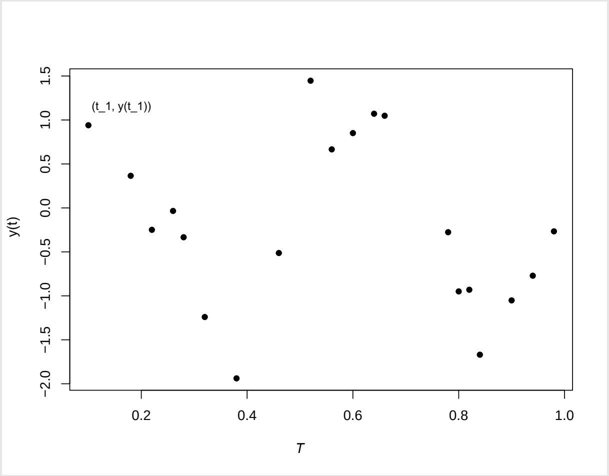


Figure: Sample from a functional data.

Functional Output Layer

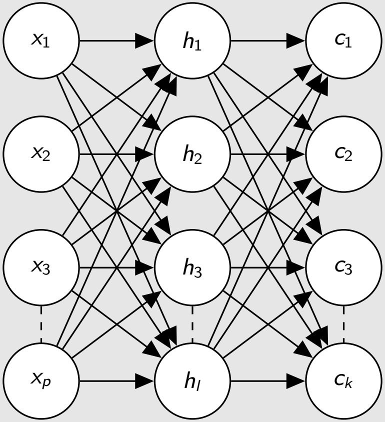


Figure: A 1-hidden layer NN to predict coefficients.

Functional Output Layer

- We have a NN that outputs $\hat{\mathbf{c}}$: $\mathrm{NN}(\mathbf{x}_i) = \hat{\mathbf{c}}^i$.
- We can reconstruct the response $\hat{y}_i(t) = \sum_{k=1}^{K} \hat{c}_k^i B_k(t)$
- We can evaluate $\hat{y}(t)$ at $\mathbf{t}^i$: $\hat{y}_i(\mathbf{t^i}) = [\hat{y}_i(t_1^i), \hat{y}_i(t_2^i), ...\hat{y}_i(t_m^i)]$ $= [\sum_{k=1}^{K} c_k^i B_k(t_1^i), \sum_{k=1}^{K} c_k^i B_k(t_2^i), ..., \sum_{k=1}^{K} c_k^i B_k(t_m^i)]$
- We propose to construct $\mathbf{B}_{K\times m}$ where $B_{k,j} = B_k(t_j)$.
- Thus $\hat{y}_i(\mathbf{t^i}) = \widehat{\mathbf{c}}^i_{1\times K} \times \mathbf{B}_{K\times m}$.

Functional Output Layer

- We smooth the response and regress the coefficients onto x jointly. (a single optimization problem)
- We define the objective function directly using y_i and $\hat{y}_i$ because the matrix multiplication is differentiable.
- For instance: $L_Y = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}[y_i(t_j) - \hat{y}_i(t_j)]^2$

Functional Output Layer

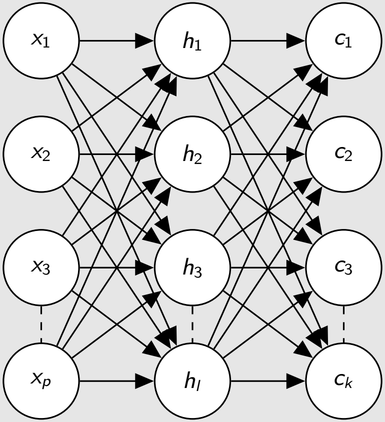


Figure: A 1-hidden layer NN to predict coefficients.

Functional Output Layer

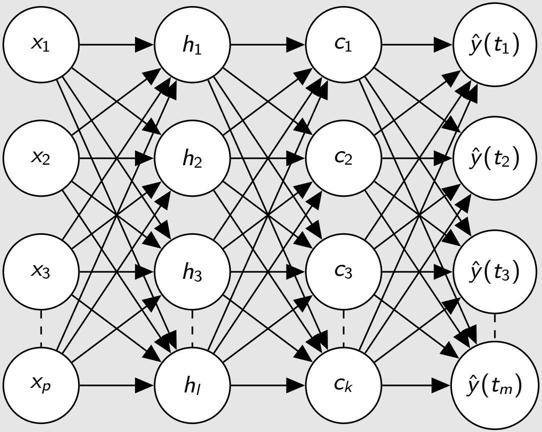


Figure: The model proposed can be perceived as adding a deterministic layer to the model.

Functional Output Layer

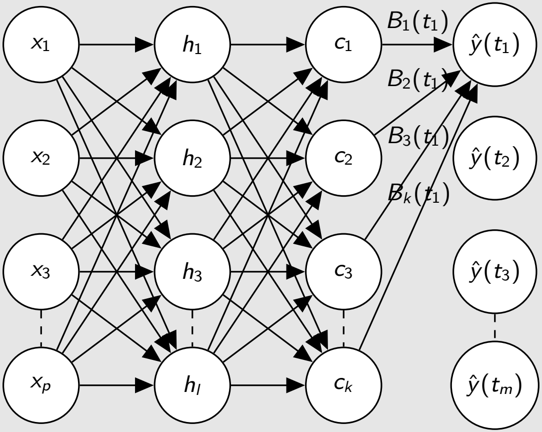


Figure: The model proposed can be perceived as adding a deterministic layer to the model.

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

- What if the time points $\mathbf{t}_i = [t^i_1, t^i_2, ..., t^i_{m_i}]$ observed for subject i are different than time points $\mathbf{t}_j = [t^j_1, t^j_2, ..., t^j_{m_j}]$ 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 **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:

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

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:

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

$$+ \lambda \int_{\mathcal{T}} \left(\frac{d^2\hat{y}(t)}{dt^2}\right)^2 dt \Big). \tag{3}$$

true

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:

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

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

t(5)be

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

Functional Input Layer

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

Simple MLP with scalar input

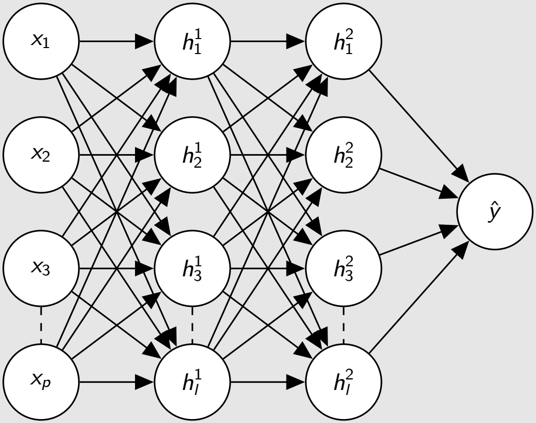


Figure: Simple MLP with scalar input and output.

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 **c** are the parameters we are trying to learn in this layer through back-propagation (learning the functional weight).

Functional Input Layer

$$
\begin{aligned}
h_l &= g\Big(\int_T x(t)\beta_l(t)\Big) && (7)\\
&= g\Big(\int_T x(t)\sum_{k=1}^{K} c_{l,k}B_k(t)\Big) && (8)\\
&= g\Big(\sum_{k=1}^{K} c_{l,k}\int_T x(t)B_k(t)\Big) && (9)\\
&= g\Big(\sum_{k=1}^{K} c_{l,k}f_k\Big) && (10)
\end{aligned}
$$

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

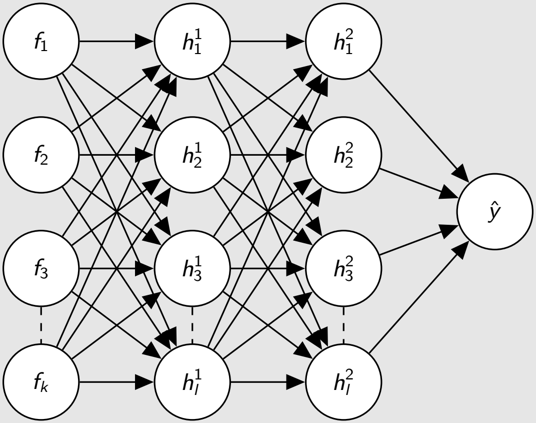


Figure: Simple MLP with scalar input and output.

Simple MLP with features f as input

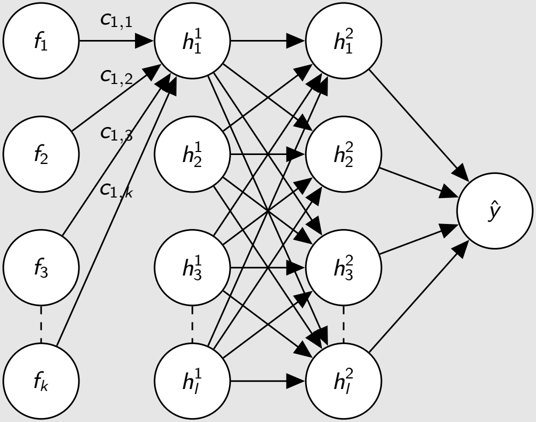


Figure: Simple MLP with scalar input and output.

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 $f_k = \sum_{j=1}^{m} x(t_j)B_k(t_j)$ (numerical approximation).
- We can perceive this first step as a deterministic layer since **f** is a just a linear combination of $x(t)$

Functional Input Layer

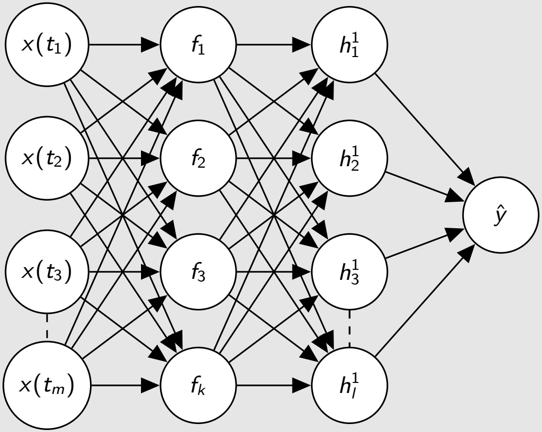


Figure: Simple MLP with scalar input and output.

Functional Input Layer

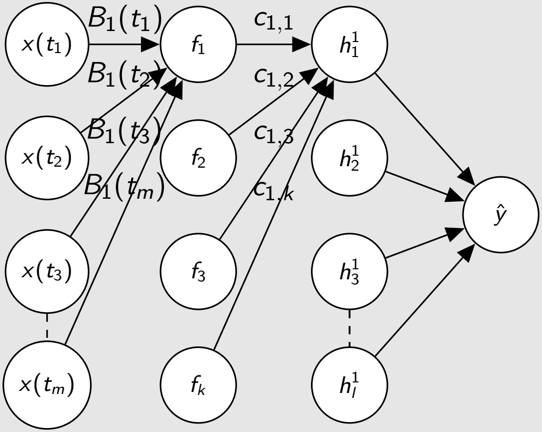


Figure: Simple MLP with scalar input and output.

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.

Models implemented

- We implemented FoS models for simple regression problems.
- We built a FAE that combines both layers proposed above. (we studied its relationship with functional PCA)
- Study important sources of pattern and variation among the data.

Possible models

- 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 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 input and output layer in Python that is not public yet.
- Everything will be available on GitHub.
- 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.

Convolution layer on functional data

Couche de convolution 1D appliquée à une donnée fonctionnelle

Input de la convolution:

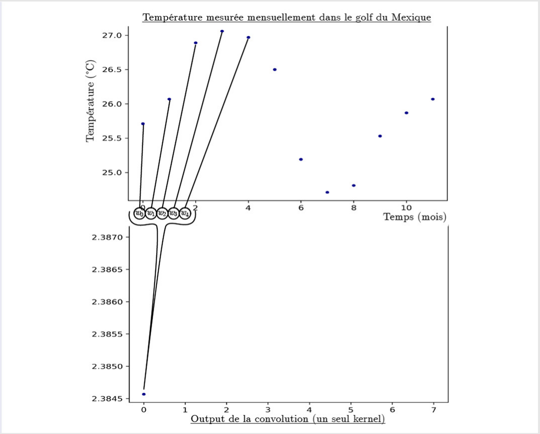
(1 donnée fonctionnelle de taille 12)

Application de la couche de convolution:

- taille du noyau = 5
- stride = 1
- padding = 0
- dilatation = 1

Taille de l'output de la convolution:

$$\left\lfloor \frac{12+2\times0-1\times(5-1)-1}{1}+1 \right\rfloor=8$$



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, Accepted, Statistics and Computing, 2023.

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

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