# Functional Data Analysis With Multi Layer Perceptrons[0]

Fabrice Rossi*[1], Brieuc Conan-Guez† and François Fleuret†

* LISE/CEREMADE, UMR CNRS 7534, Université Paris-IX Dauphine, Place du Maréchal de Lattre de Tassigny, 75016 Paris, France
† INRIA, Domaine de Voluceau, Rocquencourt, B.P. 105
78153 Le Chesnay Cedex, France

*Abstract*—**In this paper, we propose a way to apply Multi Layer Perceptron (MLP) to Functional Data Analysis. We introduce a computation model for functional input data and we show that this model is a well behaving extension of MLP: we show that the proposed model has the universal approximation property. Moreover, parameter estimation for this model is consistent. As a conclusion, we demonstrate functional MLP possibilities on simulated data and show they perform better than numerical MLP for a given number of parameters.**

## I. Introduction

In many practical situations, studied objects can be accurately described by one or more regular functions. For instance, the size of a child at different ages is a continuous function from a real interval to $\mathbb{R}$ (see [1]). Another example is provided by the annual temperature variation in a given region: previous years can be used to predict the next year whole climate thanks to auto-regressive functional models, as in [2].

It might be tempting to consider functional data as classical multivariate data, because they are in general described by a finite set of input/output pairs (for instance a set of ages is associated to a corresponding set of sizes in the previously presented example). Unfortunately, multivariate manipulation does not take into account smoothness or more generally structure of the underlying functions, and numerous practical studies have shown that a direct functional approach gives better results. This is the purpose of Functional Data Analysis (FDA), which is comprehensively presented in [1]. Moreover, in many cases, measurement points differ from one input function to another: multivariate treatment is not possible directly in this case (see section IV for examples).

FDA methods are in general based on linear modeling. Moreover, observed functions are generally replaced by the coordinates of their projection on a well chosen base: it is quite common for instance to use a spline based approximation of input functions. This allows to fall back to a multivariate approach, but without loosing the smoothness assumption which can be taken into account during the projection (and also directly within the method).

In this paper, we propose to use Multi Layer Perceptrons (MLP) for FDA, so as to enable non linear processing of functional inputs. Our model does not use basis function expansion and can therefore work directly on functional inputs.

The rest of this paper is organized as follows: we present first our model. Then we give theoretical results that show the proposed model is an universal approximator and that consistent estimation of optimal parameters is possible. We conclude with simulation results that illustrate the results our model can achieve.

## II. Functional Multi Layer Perceptrons

### A. Neurons with infinite dimensional input

On a theoretical point of view, there is no obvious reason to restrict MLP to finite dimensional inputs. Indeed a classical MLP neuron maps an input $x$ to the real output $T(b + wx)$, where $T$ is an activation function from $\mathbb{R}$ to $\mathbb{R}$, $b$ a real number and $w$ a vector from $\mathbb{R}^n$, exactly as $x$. Whereas $T$ and $b$ cannot be easily extended, $x \mapsto wx$ can: this function is a linear form on the input space $\mathbb{R}^n$, characterized by the vector $w$. If we consider a general vectorial input space $X$, we can use a "weight" $w$ from $X^*$, the dual of $X$, i.e., the set of continuous linear forms on $X$. With this approach, a generalized neuron maps an input $x \in X$ to the real output $T(b + w(x))$. In this formula, $w(x)$ is the generalization of $wx$ to arbitrary vectorial space. This approach is rather natural and was precisely studied on a theoretical point of view in [3].

Using generalized neurons, it's easy to build a MLP that works on an arbitrary vectorial space $X$. As generalized neurons provide a real output, we use them only in the first hidden layer. Subsequent layers use numerical neurons. For instance, an one hidden layer perceptron with one real output computes a function of the following form:

$$H(g) = \sum_{i=1}^{k} a_i T\left(b_i + w_i(g)\right), \tag{1}$$

[0]Published in IJCNN'02 Proceedings.
Available at
`http://apiacoa.org/publications/2002/ijcnn02.pdf`
[1]Up to date contact informations for Fabrice Rossi are avaible at `http://apiacoa.org/`

where the $a_i$ are real numbers, as well as the $b_i$, and $w_i$ are continuous linear forms on $X$.

### B. Functional MLP

Of course, equation 1 cannot be used directly, simply because it's not possible in general to manipulate linear forms on arbitrary vectorial spaces. In this paper, we consider only functional inputs, more precisely, $X$ is a $L^p(\mu)$ space, i.e. a set of measurable functions $f$ from $\mathbb{R}^n$ to $\mathbb{R}$ such that $\int |f|^p d\mu < \infty$, where $\mu$ is a finite positive Borel measure on $\mathbb{R}^n$ (it's the observation measure, see following subsection). In this case, we know that a subset[2] of the dual of $L^p(\mu)$ can be identified to $L^q(\mu)$, where $q$ is the conjugate exponent of $p$. Therefore, we can simplify weights which are no more linear forms but functions. Equation 1 turns into:

$$H(g) = \sum_{i=1}^{k} a_i T\left(b_i + \int w_i g d\mu\right), \qquad (2)$$

where the $a_i$ are real numbers, as well as the $b_i$, and $w_i$ are functions in $L^q(\mu)$.

### C. Practical implementation

We still have two practical problems with equation 2: it is not easy to manipulate functions (i.e., weights) and we have only incomplete data (i.e., input functions are known only thanks to finite sets of input/output pairs).

The first problem can be solved by embedding numerical function approximators into generalized neurons. More precisely, for each weight we can use a parametric regressor, i.e., a function $F$ from $W \times \mathbb{R}^n$ to $\mathbb{R}$, where $W$ is a compact subset of $\mathbb{R}^j$. $F$ is assumed to be easily computable. By adjusting numerical parameters, i.e., by choosing $w \in W$, we can modify the weight function $F(w, .)$. It is possible for instance to use a numerical MLP for each parametric regressor, but it's not mandatory (B-splines as well as trigonometric series can be interesting candidates).

Equation 2 turns now into:

$$H(g, w) = \sum_{i=1}^{k} a_i T\left(b_i + \int F_i(w_i, x) g(x) d\mu(x)\right), \qquad (3)$$

where the $a_i$ are real numbers, as well as the $b_i$, and $w = (w_1, \ldots, w_k)$ are parameter vectors (from a finite dimensional space) for parametric regressors.

The second problem is partly solved thanks to the observation measure $\mu$. Indeed, each input function is in general only known at a finite number of measurement points. We assume that those measurement points are independent identically distributed random variables (defined on a probability space $\mathcal{P}$) and we denote $P_X$ the probability measure induced on $\mathbb{R}^n$ by those random variables. This observation measure weights measurement points and it seems therefore natural to consider that $\mu = P_X$.

If we denote $(X_l)_{l\in\mathbb{N}}$ the sequence of random variables associated to measurement points for function $g$, the exact output for the functional MLP proposed by equation 3 is replaced by the following sequence of random variables:

$$\widehat{H}(g, w)^m = \sum_{i=1}^{k} a_i T\left(b_i + \frac{1}{m}\sum_{l=1}^{m} F_i(w_i, X_l) g(X_l)\right), \qquad (4)$$

where internal sums are approximations of:

$$E(F_i(w_i, X) g(X)) = \int F_i(w_i, x) g(x) dP_X(x), \qquad (5)$$

if we denote $X = X_1$.

### D. Training

There is one practical problem left: how to train a functional MLP? We propose as for numerical MLP to rely on gradient based algorithms. Therefore, we need to be able to compute the derivative of the output of a generalized neuron with respect to its parameters. Under regularity assumptions, derivating the mapping $w \mapsto U(w) = T\left(b + \int F(w, .) g d\mu\right)$ is not a problem. Indeed, if we assume the partial derivative $\frac{\partial F}{\partial w}$ exists $\mu$-almost everywhere, is measurable and is dominated by a positive function of $L^q(\mu)$, then the differential of $U$ is given by:

$$dU(w) = T'\left(b + \int F(w, .) g d\mu\right) \int \frac{\partial F}{\partial w}(w, x) g(x) d\mu(x) \qquad (6)$$

To compute the differential of the modeling error made by a functional MLP, we need to apply an extended back-propagation which is described in [4]. For simple network architecture such as the MLP one, the extended back-propagation simply adapts classical formulae to the case of arbitrary neurons for which the differential of the output with respect to the numerical parameters can be computed: we are exactly in this case.

## III. Theoretical results

### A. Universal approximation

One of the most important properties fulfilled by MLP is the universal approximation property: given a sufficiently regular function and a requested approximation precision, there exists an one hidden layer perceptron that computes a function which approximates the given function to the requested precision (see [5] for one of the first results about universal approximation). In [3], M. Stinchcombe extends universal approximation results to a very broad class of MLP, including MLP on almost arbitrary input spaces. The only practical drawback of those results

[2] In fact, if $p < \infty$ $L^q(\mu)$ can be exactly identified to the dual of $L^p(\mu)$, whereas $L^1(\mu)$ can only be identified to a strict subset of the dual of $L^\infty(\mu)$.

is that they use very general hypotheses on approximation of linear forms on general vectorial spaces which are quite technical. In this section, we propose a simplified corollary of M. Stinchcombe's results which shows that our proposed model is an universal approximator.

**Corollary 1.** *Let $1 \leq p \leq \infty$ be an arbitrary real number and $q$ be the conjugate exponent of $p$. Let $\mu$ be a finite positive Borel measure on $\mathbb{R}^n$. If $p = 1$, $\mu$ is additionally assumed compactly supported. Let $M$ be a subset of $L^q(\mu)$ either that is dense in $L^q(\mu)$ when $p > 1$ or that contains a set which is uniformly dense on compacta in $C(\mathbb{R}^n, \mathbb{R})$ when $p = 1$. Let $T$ be a measurable function from $\mathbb{R}$ to $\mathbb{R}$ that is non polynomial and Riemann integrable on some compact interval (not reduced to one point) of $\mathbb{R}$.*

*Let $K$ be a compact subset $L^p(\mu)$ and $f$ be a continuous function from $K$ to $\mathbb{R}$. Then for each $\epsilon > 0$, there is a functional MLP that computes $g$ from $K$ to $\mathbb{R}$ so that $|f - g|_\infty < \epsilon$. The functional MLP uses $T$ as activation function and elements of $M$ as weight functions.*

*Proof:* See section VI. ■

In practical situation, $M$ is obtained thanks to parametric regressors. In particular, approximation results given by [6], [7], [3] allow to use numerical MLP to realize $M$ for any $p$. The practical consequence of corollary 1 is that given a continuous function from a compact subset of a functional space to $\mathbb{R}$ and a requested approximation precision, there is a functional MLP based on embedded numerical MLP (or on other parametric regressors) that computes an approximation of the proposed function within the requested precision. The quite unexpected thing is that the approximating MLP uses a finite number of parameters despite the infinite dimension of the input space.

## *B. Consistency*

As explained in subsection II-C, input functions are represented in general by a finite set of input/output pairs. Moreover, we have only a limited number of input examples. The problem is that when we estimate optimal parameters for a functional MLP using those limited information, we have no *a priori* reason to believe that the obtained parameters are correct approximation of the real optimal parameters which could be theoretically obtained if we had a complete knowledge.

This *consistency* problem has been assessed in [8] for numerical MLP. Unfortunately, results of [8] cannot be applied to our problem for two reasons: first, they explicitly ask to the MLP input to belong to a finite dimensional vector space; second, we have two approximation problems rather than one (we have limited inputs, but we have also a limited knowledge on each input).

On the mathematical point of view, our problem can be formalized as follows. We want to compute optimal parameters for an approximation task. For a given input function $g$, we know a desired output $y \in \mathbb{R}$ (extension to $\mathbb{R}^l$ is straightforward). We can use for instance a quadratic error, that is we want to minimize $(H(g,w)-y)^2$ for all $g$. Theoretically, we can describe example functions and desired outputs as independent identically distributed realization of a sequence of random elements $(G^j, Y^j)_{j\in\mathbb{N}}$ defined on a probability space $\mathcal{P}$. With this model, "true" optimal parameters for a functional MLP are minimizers of error expectation, i.e.:

$$\lambda(w) = E\left((H(G,w) - Y)^2\right), \tag{7}$$

where we denote $G = G^1$ and $Y = Y^1$. Of course, other error measures can be used, but we always end up with the expectation of the cost for one example. Unfortunately, $\lambda(w)$ cannot be exactly computed and is replaced by the following estimation (which is a random variable):

$$\widehat{\lambda}_n(w) = \frac{1}{n}\sum_{j=1}^{n}(H(G^j, w) - Y^j)^2 \tag{8}$$

Moreover, $H$ cannot be exactly computed but is replaced by a finite sample estimation (which is also a random variable, see section II-C). Therefore, we can only manipulate the following quantity:

$$\widehat{\lambda}_n^m(w) = \frac{1}{n}\sum_{j=1}^{n}(\widehat{H}(G^j, w)^{m^j} - Y^j)^2, \tag{9}$$

where $m = \inf_{1\leq j\leq n} m^j$. Let us call $\widehat{w}_n^m$ a minimizer of $\widehat{\lambda}_n^m(w)$ and $W^*$ the set of minimizers of $\lambda(w)$. Estimated optimal parameters are realization of $\widehat{w}_n^m$, whereas true optimal parameters belong to $W^*$. We have the following theorem:

**Theorem 1.** *Under regularity assumptions on $H$ and the cost function, we have:*

$$\lim_{n\to\infty}\lim_{m\to\infty} d(\widehat{w}_n^m, W^*) = 0 \qquad P_X \ a.s. \tag{10}$$

*Proof:* Proof is again omitted due to size constraints. It can be found in [9]. ■

Regularity assumptions on $H$ are very technical (see [9] for details), but can be summarized as follows: the error for one example, e.g., $d(g,w,y) = (H(g,w) - y)^2$ must be continuous with respect to the parameters and measurable with respect to the observations (input and output). Moreover, $d(.,w,.)$ has to be dominated uniformly on the parameter space by an integrable function. Similar conditions must be satisfied by embedded parametric regressors. Traditional sigmoid based MLP satisfy all the requested properties.

Because of the simplified presentation proposed here, an important fact might remain hidden: the consistency theorem does not request an uniform knowledge for each input function. The number of evaluation points and their position can depend on the input function. The only requirement is that evaluation points are independent identically distributed random variables.

The proposed theorem is a generalization of consistency results to functional MLP. The practical meaning is quite simple: optimal parameters estimated from limited data are consistent approximation of real optimal parameters, in the precise sense that the more data we have, the closer the estimated parameters are to the optimal ones.

## IV. Simulation results

We present in this section simulation results based on artificial data.

### A. Function discrimination

We have tried our model on a simple discrimination problem: we ask to a functional MLP to classify functions into classes. Example of the first class are functions of the form $f_d(x) = \sin(2\pi(x-d))$, whereas function of the second class have the general form $g_d(x) = \sin(4\pi(x-d))$.

For each class we generate example input functions according to the following procedure:
1. $d$ is randomly chosen uniformly in $[0,1]$
2. 25 measurement points are randomly chosen uniformly in $[0,1]$
3. we add a centered Gaussian noise with 0.7 standard deviation to the corresponding outputs

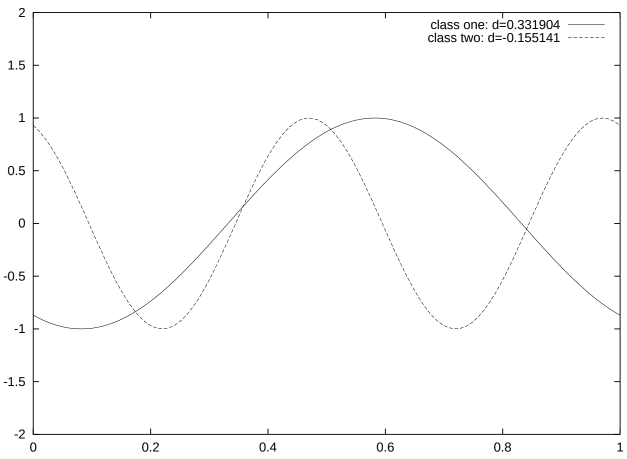

Fig. 1
Examples of smooth curves

Figure 1 give an example of both classes without the noise, whereas figure 2 gives the corresponding actual data.

Training is done thanks to a conjugate gradient algorithm and uses early stopping: we used 100 training examples (50 of each class), 100 validation examples (50 of each class) and 300 test examples (150 for each class). Using 3 hidden functional neurons, we achieved 94.4 % recognition rate. Each functional neuron is built using 4 cubic B-splines centered at regularly spaced points in $[0,1]$ (we have therefore 4 real parameters to which we add one threshold). The functional MLP uses therefore a total of 19 numerical parameters.

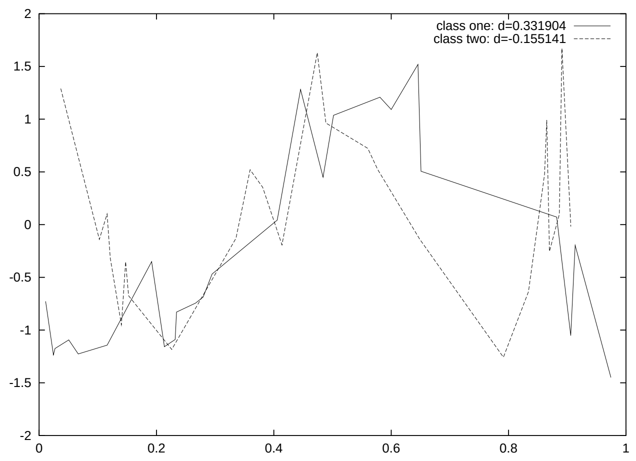

Fig. 2
Examples of actual data for curve classification

We have compared the functional MLP approach to a numerical approach. It is obviously not possible to use directly functional data as an input to a numerical MLP, especially because evaluation points depend on the function. The simplest way to transform functions in a finite number of values is to use averaging. For this experiment, we have divided $[0,1]$ in four sub-intervals and we have transformed a function into four values: the mean of the function on each sub-interval. We have submitted the obtained data to a numerical MLP with 3 hidden neurons (using therefore 19 numerical parameters). The MLP has been trained exactly as the functional ones (using early stopping and with identical training sets up to the recoding). After training, we obtain 94.7% recognition rate, which is slightly better than the functional MLP.

We have conducted other similar experiments, with different functional generator (for instance $f_d(x) = \sin(4\pi(x-d))$ and $g_d(x) = \sin(6\pi(x-d))$). The general behavior is always the same: functional and numerical MLP obtain similar performances. There is no obvious winner. We have presented a simulation in which numerical MLP give slightly better results, but for other input functions, the contrary is true.

Results of this experiment are a little bit disappointing, because the functional MLP does not outperform the numerical one. The main explanation is that the averaging strategy is quite well adapted to the data. With low dimensional input space for the studied functions, it is quite obvious that this kind of averaging strategy will give a correct representation of the function. When we use higher dimensional input spaces, the number of evaluation points needed to achieve a correct representation by a simple averaging technique is too important. As the next experiment will show, functional MLP will not suffer too much from this low volume of data.

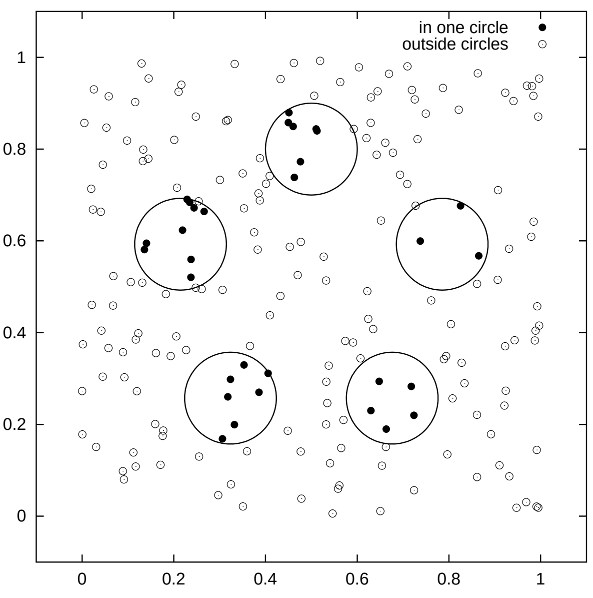

Fig. 3
Examples of circle data with five circles (11111)

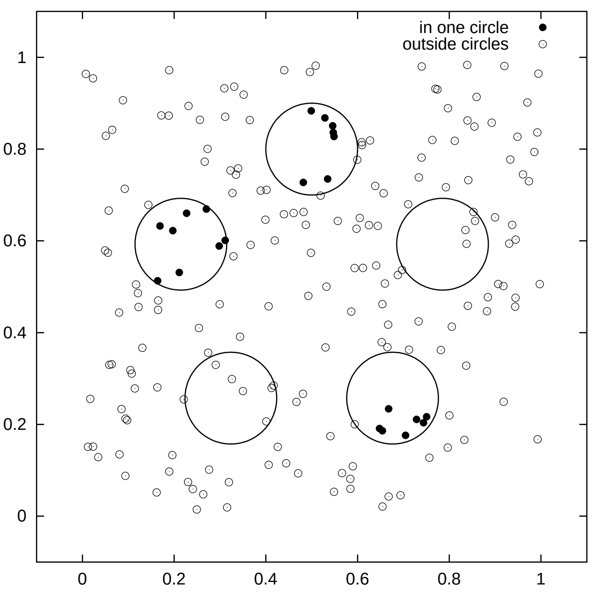

Fig. 4
Examples of circle data with three circles (11010)

## *B. Circle counting*

We have tried functional MLP on another difficult task. We define five small circles in the plane (radius is 0.1). Centers are uniformly spaced on a bigger circle (radius is 0.3). The five circles are represented on figure 3. We generate an input function as follows:

1. We first choose uniformly an integer from $[0, 31]$. The binary representation of this number, $b_0b_1b_2b_3b_4$, is used as the desired output for the currently generated input (we have therefore five output units). Moreover, each digit in the binary representation corresponds to a circle.
2. Then we choose 200 measurement points uniformly in $[0, 1] \times [0, 1]$. The output of the function at a given point is always 0 if the point falls out of the five small circles. If the point falls in small circle number $i$, the output of the function is $b_i$. For instance, figure 3 is an observation of the input function for $11111 = 31$ and figure 4 corresponds to $11010 = 26$.

The goal of the functional MLP is to associate the function to the generating number translated into its binary representation. We have use embedded MLP to represent weight functions. Experiments have been conducted with 100 input functions for the training set as well as 100 input functions for the validation set. Final performances (after early stopping) were calculated on a test set with 300 examples. The following table summarizes experiments. Mean square error and recognition rate have been calculated on the test set. The recognition is obtained by thresholding the output neurons to 1 if the output is greater than 0.5, and to 0 on the contrary.

| hidden functional neurons | hidden numerical neurons | weight number | mean square error | recognition rate |
|---|---|---|---|---|
| 5 | 1 | 60 | 0.071 | 92% |
| 6 | 1 | 71 | 0.059 | 97% |
| 5 | 3 | 100 | 0.016 | 100% |

We have tried to submit those data to a numerical MLP. As in the previous experiment, the simplest choice is to replace the sample data by average performed on a fixed grid (more sophisticated procedures might be used, as explained below). To do this, we have divided the square $[0, 1] \times [0, 1]$ into $r \times r$ regular sub-squares. We use a MLP with $r^2$ inputs. Each input is the average output value of the example function on the considered square. For instance with the functional input data represented in figure 4, we obtain the following reduced input data (for $r = 4$):

| | $x \in [0, \frac{1}{4}[$ | $x \in [\frac{1}{4}, \frac{1}{2}[$ | $x \in [\frac{1}{2}, \frac{3}{4}[$ | $x \in [\frac{3}{4}, 1]$ |
|---|---|---|---|---|
| $y \in [0, \frac{1}{4}[$ | 0 | 0 | 0.37 | 0.14 |
| $y \in [\frac{1}{4}, \frac{1}{2}[$ | 0 | 0 | 0 | 0 |
| $y \in [\frac{1}{2}, \frac{3}{4}[$ | 0.45 | 0.24 | 0.056 | 0 |
| $y \in [\frac{3}{4}, 1]$ | 0 | 0.077 | 0.36 | 0 |

The desired output is encoded as for the functional MLP by 5 output units. We obtain the following results (training sets are identical for both methods):

| $r$ | hidden neurons | weight number | mean square error | recognition rate |
|---|---|---|---|---|
| 3 | 5 | 80 | 0.053 | 94% |
| 4 | 5 | 115 | 0.013 | 99% |

Those experiments clearly show that numerical MLP need more weights than functional MLP to achieve similar results. We think that the power of functional MLP comes from the fact it can adapt its averaging model to the data, whereas it has to be fixed before training to produce good results with numerical MLP. It would have been simple to chose a more adapted averaging strategy for the studied case and therefore to increase numerical MLP performances. But this kind of specialized averaging is possible only with a very good understanding of the data, whereas functional MLP have been applied without any data driven modification. A complete comparison between data driven averaging strategies as a preprocessing stage before numerical MLP processing on the one hand, and functional MLP on the other hand, is currently our main focus, especially when the input dimension is high.

## V. Conclusion

In this paper, we have introduced Functional Multi Layer Perceptrons (FMLP), a simple extension of MLP to functional data. The proposed model is very interesting on a theoretical point view because it shares with its numerical counterpart useful properties.

We have indeed shown that FMLP are universal approximators, that is they can approximate continuous mappings from a compact subset of a functional space to $\mathbb{R}$ with arbitrary precision. For a given function to approximate to a given accuracy, the approximating FMLP uses a finite number of numerical parameters.

Moreover, we have shown that parameter estimation for FMLP is consistent: optimal parameters estimated thanks to a finite number of functions known at a finite number of measurement points converge to the set of true optimal parameters when the size of the data increases.

We have shown on simulated data that FMLP perform in a very satisfactory way. In high dimensional input spaces, even if a numerical MLP approach is possible, it will use more parameters than the corresponding FMLP. More investigation is nevertheless needed to establish the practical possibilities of Functional MLP, especially experiment involving real data. Comparison with sophisticated preprocessing techniques associated to numerical MLP will be very interesting, because we think FMLP are a way to integrate the preprocessing optimization directly in the MLP optimization and therefore to avoid an explicit definition of this preprocessing phase.

## VI. Proof of corollary 1

Dues to space constraints, we give only a sketch of proof. The complete proof can be found in [9].

We denote $A_M$ the set of linear forms on $L^p(\mu)$ of the form $l(f) = \int fgd\mu$, where $g \in M$. We have to consider three cases:
1. $1 < p < \infty$: then, $L^q(\mu)$ can be identified to the dual of $L^p(\mu)$. As $M$ is dense in $L^q(\mu)$, $A_M$ is dense in $(L^p(\mu))^*$ for the weak-$*$ topology. According to [7], hypotheses on $T$ imply hypotheses on the generalized MLP activation function needed by corollary 5.1.3 of [3]. Density of $A_M$ is the other requested property of this corollary which can therefore by applied to conclude that we have universal approximation.
2. $p = \infty$: this case is slightly more technical. Basically, we need to prove that the set of functions defined on $L^\infty(\mu)$ by $l(f) = \alpha + \int fgd\mu$, where $g \in M$, separates points in $K$. That is, given two functions $f_1$ and $f_2$ from $K$, we have to prove that there is a function $l$ such that $l(f_1) \neq l(f_2)$. We do this by approximating the characteristic function of the set $H^+ = \{x \in \mathbb{R}^n \mid f_1(x) > f_2(x)\}$ (is $H^+$ as zero measure, we swap $f_1$ and $f_2$). This function is approximated by a function from $M$ thanks to the density assumption. When the separation property has been established, we apply theorem 5.1 of [3] which implies universal approximation.
3. $p = 1$: we prove that $A_M$ is dense in $(L^p(\mu))^*$ for the weak-$*$ topology and using the identification of this dual to $L^\infty(\mu)$. To do this, we first approximate a function in $L^\infty(\mu)$ by a compactly supported continuous function thanks Lusin theorem (e.g. [10]). Then we approximate this function on the support of $\mu$ by an element of $M$ thanks to the hypothesis. Conclusion is again obtained thanks to corollary 5.1.3 of [3].
It is interesting to note that additional hypothesis on $\mu$ are needed for $p = 1$ because it's not possible in general to approximate functions (even regular) on $\mathbb{R}^n$ for the uniform norm with traditional tools (such as MLP or splines). This kind of non-approximation results is studied in [3].

## References

[1] Jim Ramsay and Bernard Silverman, *Functional Data Analysis*, Springer Series in Statistics. Springer Verlag, June 1997.

[2] Philippe Besse, Hervé Cardot, and David Stephenson, "Autoregressive forecasting of some functional climatic variations," *Scandinavian Journal of Statistics*, vol. 4, pp. 673–688, 2000.

[3] Maxwell B. Stinchcombe, "Neural network approximation of continuous functionals and continuous functions on compactifications," *Neural Networks*, vol. 12, no. 3, pp. 467–477, 1999.

[4] Cédric Gégout, Bernard Girau, and Fabrice Rossi, "Generic Back-Propagation in Arbitrary Feedforward Neural Networks," in *Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, D. W. Pearson, N. C. Steele, and R. F. Albrecht, Eds., Alès, April 1995, pp. 168–171, Springer Verlag,

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989,

[6] Kurt Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[7] Kurt Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, no. 8, pp. 1069–1072, 1993.

[8] Halbert White, "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, vol. 1, no. 4, pp. 425–464, 1989,

[9] Fabrice Rossi, Brieuc Conan-Guez, and François Fleuret, "Functional multi layer perceptrons," Tech. Rep. 0134, CEREMADE & INRIA, `http://www.ceremade.dauphine.fr/`, december 2001, Available at `http://apiacoa.org/publications/2001/Preprint0134.pdf`.

[10] Walter Rudin, *Real and complex Analysis*, Mc Graw Hill, 1974.