SAPIENZA
UNIVERSITÀ DI ROMA

Mackey-Glass Time-Series Prediction: Comparing the Performance of LMS, KLMS and NLMS-FL

Student: Giovanni Murru

Dept. of Informatica e Sistemistica

Professor: Aurelio Uncini

Dept. of Scienza e Tecnica dell'Informazione e della Comunicazione

July 23, 2012

Abstract

Because of its several application fields, time-series prediction is one of the most demanding topic of research. The following paper describes the use of a collaborative functional link based architecture, which is normally used in the field of nonlinear acoustic echo-cancellation, for the alternative well-known problem of Time-Series Prediction. The novel work described in [1] and [2] has been used as reference to develop the time-series predictor.

The performance of the collaborative NLMS-FL approach is then compared with the well known algorithms LMS and KLMS, showing through experimental results its performance superiority in presence of high nonlinearity mappings, such those described by the Mackey-Glass equation.

Contents

Introduction

The information is a constantly increasing flow of numerical data. Every information can be represented as a signal and every signal can be represented as numbers. The problem of predicting the future evolution of these numbers is one of the main topics of discussion in the scientific community. The accurate forecast of this numerical evolution would be of utility for many disciplines.

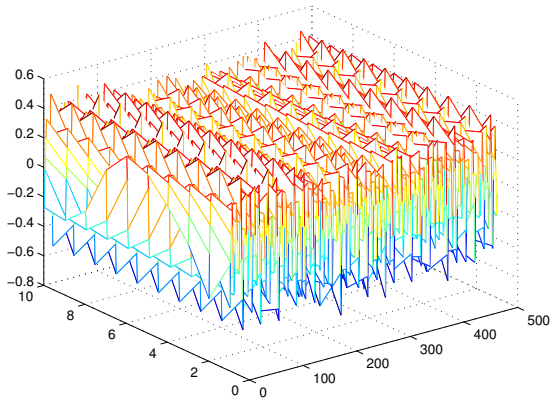


Figure 0.1: The Mackey-Glass time-series embedded for prediction.

Signals in nature is often analog and continuous. However such signals can be elaborated and represented as a time series of numerical information through the sampling technique, which transform analog flows in digital ones. The prediction of signals' evolution as sunspot activity, air temperature or financial stock market is of strong interest.

Prediction consists in taking advantage of the time-series history to forecast the future values in the series. Since nature is non-linear by definition, there have been efforts and attempts to create efficient algorithms able to manage and predict non-linear data mappings. Several algorithms have been developed always taking into account a fair trade-off between performance and cost.

In the following a comparison between the performance of different algorithms on Mackey-Glass time series' prediction is described. This time-series, which well represents a chaotic and periodic dynamics, was initially used to model physiological control systems such as electrolytes, oxygen, glucose in the blood. Mackey-Glass time-series used in the experiments is generated using a non-linear time-delay differential equation, discretized at a sampling period of 6 seconds.

In particular the experiments consist in comparing the performance of 2 linear combiners trained with Least Means Square (LMS) and Kernel LMS, with the behavior of the NLMS-FL collaborative structure described in [2]. This network, which makes use of an adaptive combination between a non-linear functional link network (FL) and a linear filter, is tuned in order to work for the time-series prediction problem.

To validate performance's results a set of Monte Carlo simulations was executed.

1 A Theoretical Review

Prediction is based on past inputs, but since time is constantly moving forward, it's clear that past inputs are infinite. However the predictor input space must be finite, hence the input's history should be truncated to a certain finite number of previous samples. The choice of the time-window size is a critical step in the performance of the prediction, and determines its memory on the past events.

1.1 The Prediction Problem

A Time-Series ($T.S.$) can be formally defined as a sequence of scalars (or vectors) which depends on time.

$$T.S. = \{x(t_0), x(t_1), x(t_2), \dots, x(t_{i-1}), x(t_i), x(t_{i+1}), \dots\} \tag{1.1}$$

The time-series represents the output of some process P that we are interested in.

To predict the future at a certain time, a tapped delay line associated to each $x(t_i)$ contained in our dataset should be created. In practice this is done through a process known as embedding. The output of the embedding operation is a matrix containing for each column the tapped-delay line shifted by one time sample.

$$\begin{pmatrix} x(t_i) & x(t_{i+1}) & \dots & x(t_{i+N-1}) \\ x(t_{i-1}) & x(t_i) & \dots & x(t_{i+N-2}) \\ x(t_{i-2}) & x(t_{i-1}) & \dots & x(t_{i+N-3}) \\ \dots & \dots & \dots & \dots \\ x(t_{i-L-1}) & x(t_{i-L}) & \dots & x(t_{i+N-L}) \end{pmatrix} \tag{1.2}$$

Hence the first column represents the time-history (the past L inputs) of the future event $x(t_{i+1})$. The number of columns is the number of input patterns that the predictor use for the training or the testing process.

Without loss of generality let's consider the first input pattern, which contains L events in the past. This vector is passed to the predictor which computes, using a weight vector, an estimation of the future event $\tilde{x}(t_{i+1})$. The weight vector is then improved and updated using a law, function of the error $e = x(t_{i+1}) - \tilde{x}(t_{i+1})$.

The choice of the proper algorithm for updating the weight vector is a critical step in determining the efficiency of the predictor. Many algorithms exist in literature. In the following subsections two of the three algorithm tested for the prediction problem are briefly described.

In the end on the next section, the third approach, which makes use of the NLMS-FL collaborative adaptive network, is presented.

1.2 Least Mean Square Algorithm

Least Mean Square is a well-known and robust algorithm based on the minimization of the mean square error. It is a stochastic gradient descent method, in fact

Algorithm 1. The least-mean-square algorithm

Initialization

$\mathbf{w}(0) = 0$, choose η

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**
  $e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)$
  $\mathbf{w}(i) = \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i)$
end while

Figure 1.1: The Least Mean Square algorithm [5].

it can be derived by using the instantaneous gradient of the cost function $J(\mathbf{w})$:

$$J(\mathbf{w}) = \sum_{i=1}^{N} (d(i) - \mathbf{w}^T\mathbf{u}(i))^2 \tag{1.3}$$

Applying the method of steepest descent we can formulate the update step of the LMS algorithm as follows:

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i) \tag{1.4}$$
$$e(i) = d(i) - \mathbf{w}(i-1)^T\mathbf{u}(i) \tag{1.5}$$

where η is known as the learning parameter and determines the step-size on correction of the optimal weight. In equation 1.5, $\boldsymbol{u}(i)$ represents the input history, that weighted with $\mathbf{w}(i-1)^T$ gives an estimation $\tilde{d}(i)$ of the future input $d(i)$ whose prediction is of interest.

In point of fact the algorithm represents a way to iteratively find the optimal weight vector $\mathbf{w}^o$ in order to minimize the cost function $J(\mathbf{w})$.

1.3 Kernel Method

The kernel method was introduced in order to deal with the high non-linearity which is often present in natural phenomena on the mapping between input $\mathbf{u}$ and response d. The main idea driving the kernel methods is to map the input data $\mathbf{u}$ into a high dimensional feature space, using a transformation function $\boldsymbol{\varphi}$. Then linear methods can be applied to the transformed data.

Algorithm 2. The kernel least-mean-square algorithm

Initialization

choose step-size parameter η and kernel κ
$\mathbf{a}_1(1) = \eta d(1), \mathcal{C}(1) = \{\mathbf{u}(1)\}, f_1 = \mathbf{a}_1(1)\kappa(\mathbf{u}(1), \cdot)$

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**
  %compute the output
  $f_{i-1}(\mathbf{u}(i)) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa(\mathbf{u}(i), \mathbf{u}(j))$
  %compute the error
  $e(i) = d(i) - f_{i-1}(\mathbf{u}(i))$
  %store the new center
  $\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\}$
  %compute and store the coefficient
  $\mathbf{a}_i(i) = \eta e(i)$
end while

Figure 1.2: The Kernel Least Mean Square algorithm [5].

Kernel Least Mean Square is the simplest among the family of the kernel adaptive filters. As like as its predecessor, goal of KLMS is to learn a continuous input-output mapping given the training data, but using new inputs $\boldsymbol{\varphi}(\mathbf{u}(i))$ with high dimensionality. By applying LMS algorithm to the new training data sequence $\{\langle \boldsymbol{\varphi}(\mathbf{u}(i)), d(i) \rangle\}$, what we obtain is an estimate of the weight vector in the new high dimensional feature space. The estimate $\boldsymbol{\omega}(i)$ can be expressed as a linear combination of all the previous and present inputs, weighted by the prediction errors and scaled by the learning factor η.

$$\boldsymbol{\omega}(i) = \eta \sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(j) \qquad assuming \quad \boldsymbol{\omega}(0) = 0 \tag{1.6}$$

Considering a new input $\boldsymbol{u}'$ and by using the kernel trick equation in 1.7, we can efficiently compute in 1.8 the output of the system in response to a new input $\boldsymbol{u}'$ using kernel evaluations [5]:

$$\boldsymbol{\varphi}(\boldsymbol{u})^T \boldsymbol{\varphi}(\boldsymbol{u}') = k(\boldsymbol{u}, \boldsymbol{u}') \tag{1.7}$$

$$\boldsymbol{\omega}(i)^T \boldsymbol{\varphi}(\mathbf{u}') = \eta \sum_{j=1}^{i} e(j) k(\boldsymbol{u}(j), \boldsymbol{u}') \tag{1.8}$$

where $k(\boldsymbol{u}(j), \boldsymbol{u}')$ represents the kernel equation.

If we denote the estimate of the input-output nonlinear mapping at time i with f_i, we can have the learning rule for the KLMS algorithm expressed as:

$$f_i = f_{i-1} + \eta e(i) k(\boldsymbol{u}(i), .) \tag{1.9}$$
$$e(i) = d(i) - f_{i-1}(\boldsymbol{u}(i)) \tag{1.10}$$

where values of the estimates $f_{i-1}(\boldsymbol{u}(i))$ and f_{i-1} are expressed in equations 1.11 and 1.12.

$$f_{i-1}(\boldsymbol{u}(i)) = \eta \sum_{j=1}^{i-1} e(j) k(\boldsymbol{u}(j), \boldsymbol{u}(i)) \tag{1.11}$$

$$f_{i-1} = \eta \sum_{j=1}^{i-1} e(j) k(\boldsymbol{u}(j), .) \tag{1.12}$$

KLMS is a simple algorithm. As shown in the pseudo-code on figure 1.2, the algorithm needs two fundamental steps for the initialization. One is the well-known choice of the step-size parameter η, while the other is the choice of the kernel, which defines the similarity between data points.

$$k(\boldsymbol{u}, \boldsymbol{u}') = \exp\left(-a \|(\boldsymbol{u} - \boldsymbol{u}')\|^2\right) \tag{1.13}$$

The Gaussian kernel expressed in 1.13 is a typical choice because is numerically stable and usually gives reasonable results. Anyway other kernel functions may be adopted, such as the polynomial, the multiquadrics and so on. As a matter of fact in the experiments we used a Gaussian kernel with parameter $a = 1$.

2 A Collaborative Network for Time-Series Prediction

The massive parallelism and adaptivity which belongs to the human brain can be hardly emulated using conventional serial-in-computation architectures.

The artificial neural networks try to simulate the biological neural system's computational paradigm [4].

2.1 The FLN: an alternative to multi-layer perceptron

Functional Link Networks (FLNs) are a type of artificial neural networks owning a single layer, originally proposed in [3]. They allow to adaptively filter non-linearities using a process known as functional expansion. The process consist in enhancing the input pattern signal projecting it in a higher dimension space. The functional expansion's output is the enhanced pattern $\boldsymbol{z}[n]$ described in [2], which makes use of a purely non-linear transformation of the original input $\boldsymbol{x}[n]$.

$$\boldsymbol{z}[n] = [1,\ z_0[n],\ \ldots,\ z_{L_{in}-1}[n]]^T \tag{2.1}$$

whose elements are defined as: $$\boldsymbol{z}_i[n] = [z_{i,0}[n],\ \ldots, z_{i,2p-1}[n]] \tag{2.2}$$

Moreover the elements of $\boldsymbol{z}_i[n]$ are defined using the following non-linear extension:

$$z_{i,j}[n] = \begin{cases} sin(j\pi x_i[n]) & \text{for} \quad j = 2, 4, \ldots, 2p \\ cos(j\pi x_i[n]) & \text{for} \quad j = 1, 3, \ldots, 2p-1 \end{cases} \tag{2.3}$$

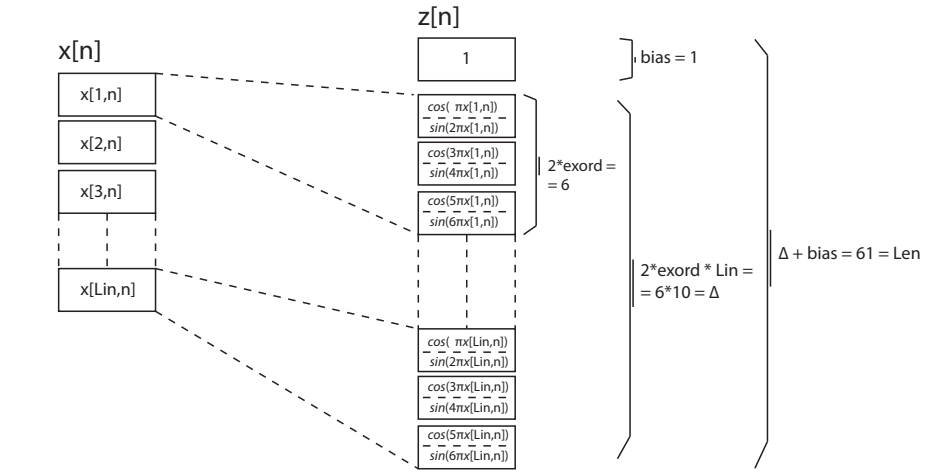


Figure 2.1: Creation of the enhanced pattern $\boldsymbol{z}[n]$.

where p in 2.3 is the chosen expansion order, and $x_i[n]$ is the i-th element of the input pattern at the n-th instant.

The output of this procedure is clearly the enhanced input pattern $\boldsymbol{z}[n]$, whose size is $L_{en} = L_{in}2p + 1$. The $\boldsymbol{z}[n]$ vector is passed in input to an adaptive filter which makes use of the Normalized Least Means Square (NLMS) algorithm to update a weight vector called $\boldsymbol{h}[n]$, which clearly has the same size of $\boldsymbol{z}[n]$.

$$\boldsymbol{h}[n] = [h_0[n],\ h_1[n],\ \ldots,\ h_{L_{en}-1}[n]]^T \tag{2.4}$$

The Normalized version of the Least Mean Square Algorithm is characterized by a normalization factor ζ which scales the overall error signal $e_{FL}[n]$.

$$\zeta = \boldsymbol{z}[n]^T \boldsymbol{z}[n] + \delta_{NL} \tag{2.5}$$

In fact the weight update rule can be described by the following equation:

$$\boldsymbol{h}[n] = \boldsymbol{h}[n-1] + \frac{\mu_{NL}\boldsymbol{z}[n]e_{FL}[n]}{\zeta} \tag{2.6}$$

whose error is defined as: $$e_{FL}[n] = d[n] - y_{FL}[n] - y_L[n] \tag{2.7}$$

The reader should be not confused by the notation $e_{FL}[n]$. In fact the subscript FL means that this error is used for updating the FL nonlinear filter.

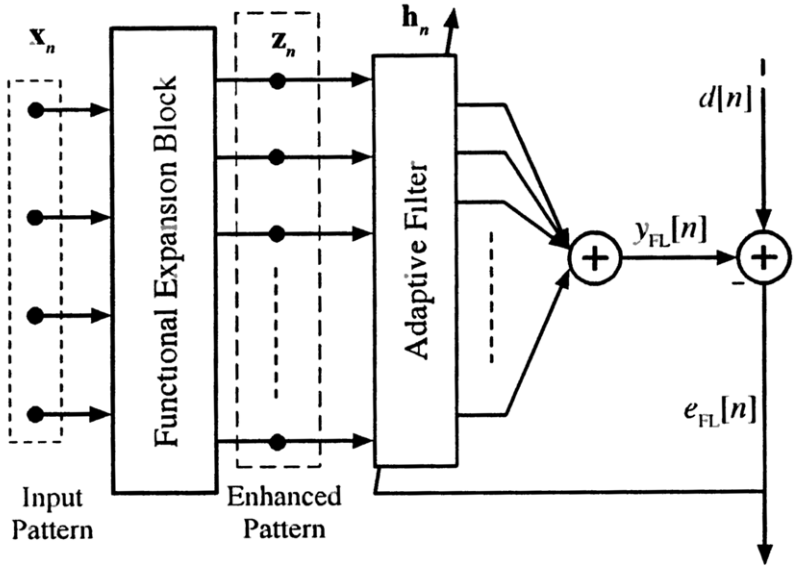


Figure 2.2: The FL nonlinear filter

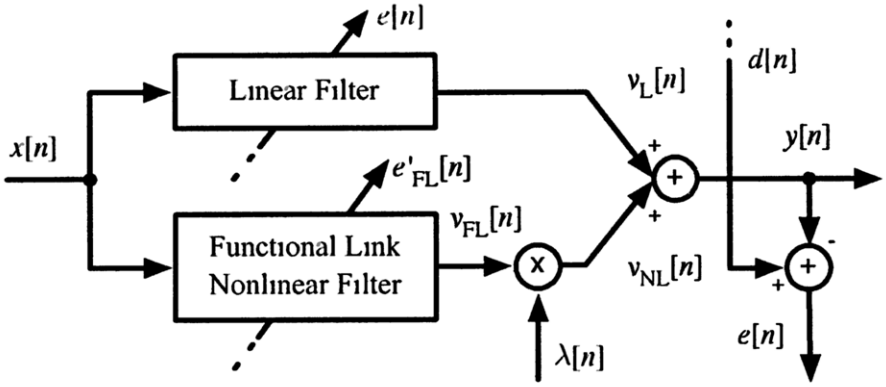


Figure 2.3: The overall collaborative architecture.

However this error takes into account also the output coming from the linear filter $y_L[n]$, as can be seen in equation 2.7. The update step of the linear filter makes use of a different overall error computed using a different law that will be later analyzed.

As shown in figure 2.2 the output $y_{FL}[n]$ of the Functional Link nonlinear filter is obtained by weighting the enhanced input pattern $\boldsymbol{z}[n]$.

$$y_{FL}[n] = \boldsymbol{z}^T[n]\boldsymbol{h}[n] \tag{2.8}$$

2.2 An adaptive robust combination with a NLMS linear filter

Linear Filter shown in figure 2.3 is characterized by a weight vector $\boldsymbol{w}[n]$ whose length is equal to the dimension of the time window L chosen in the tapped-delay line (i.e. the height of the matrix shown in equation 1.2). As a matter of fact L is the size of the input pattern $\boldsymbol{x}[n]$, which is the n-th column of the matrix in 1.2. Notice that each column is shifted in time.

$$\boldsymbol{w}[n] = [w_0[n],\ w_1[n],\ \ldots,\ w_{L-1}[n]] \tag{2.9}$$

The linear filter output is a combination of the input pattern $\boldsymbol{x}[n]$ and the weight vector $\boldsymbol{w}[n]$.

$$y_L[n] = \boldsymbol{x}^T[n]\boldsymbol{w}[n] \tag{2.10}$$

As already said the weight $\boldsymbol{w}[n]$ is updated in order to minimize a different overall square error $e^2_{AZK}[n]$ computed using a new overall output vector $y_{AZK}[n]$. In particular this is done to avoid gradient noise introduced by the nonlinear filter when a pure linear input pattern is passed to the predictor. The idea is to adaptively combine the FL nonlinear filter with an all-zero-kernel (AZK). Using this approach non linear contribution is dropped off when the input pattern is nearly linear.

The new overall signal is defined in function of a new parameter $\lambda[n]$ which can vary in the range $[0, 1]$, so that the FL output can be either kept or removed as required by the filtering scenario [2].

$$y_{AZK}[n] = y_L[n] + \lambda[n] y_{FL}[n] \tag{2.11}$$

The linear filter tries to minimize a new overall error e_{AZK} computed using the output defined in equation 2.11.

$$e_{AZK}[n] = d[n] - y_{AZK}[n] \tag{2.12}$$

The mixing parameter $\lambda[n]$ is defined in function of an adaptive parameter $a[n]$ related to it by the expression:

$$\lambda[n] = \texttt{logsig}(a[n]), \qquad \texttt{logsig}(a[n]) = \frac{1}{1 + \exp(a[n])} \tag{2.13}$$

The adaptive parameter $a[n]$ is updated using a gradient descent rule defined in 2.14. This architecture is robust to any kind of nonlinearity level and makes use of the estimated power of y_{FL} defined in 2.15, where β is a smoothing factor.

$$a[n+1] = a[n] + \frac{\mu_a}{r[n]} y_{FL}[n] e_{AZK}[n] \lambda[n](1 - \lambda[n]) \tag{2.14}$$

$$r[n] = \beta r[n-1] + (1 - \beta) y_{FL}^2[n] \tag{2.15}$$

Finally the weight $\boldsymbol{w}[n]$ is updated using the NLMS algorithm.

$$\gamma = \boldsymbol{x}^T[n]\boldsymbol{x}[n] + \delta_L \tag{2.16}$$

$$\boldsymbol{w}[n] = \boldsymbol{w}[n-1] + \frac{\mu_L \boldsymbol{x}[n] e_{AZK}[n]}{\gamma} \tag{2.17}$$

3 Experimental Results

In this section the performance of three different algorithms in the prediction problem is investigated showing the results of several experiments.

Prediction has been tested on the well-known Mackey-Glass nonlinear time-series using 3 different approaches: LMS, KLMS and the innovative NLMS-FL.

The Mackey-Glass chaotic time series is generated from the following time-delay ordinary differential equation:

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} \tag{3.1}$$

with $b = 0.1$, $a = 0.2$, and $\tau = 17$. The time series is discretized at a sampling time period of 6 seconds.

3.1 Learning Curves for LMS, KLMS and NLMS-FL

First experiment consists in comparing the performance of a linear combiner trained with the three algorithms in different condition of noise. As expected the behavior of KLMS and NLMS-FL converge to a smaller value of MSE because of their ability to handle nonlinear mappings.

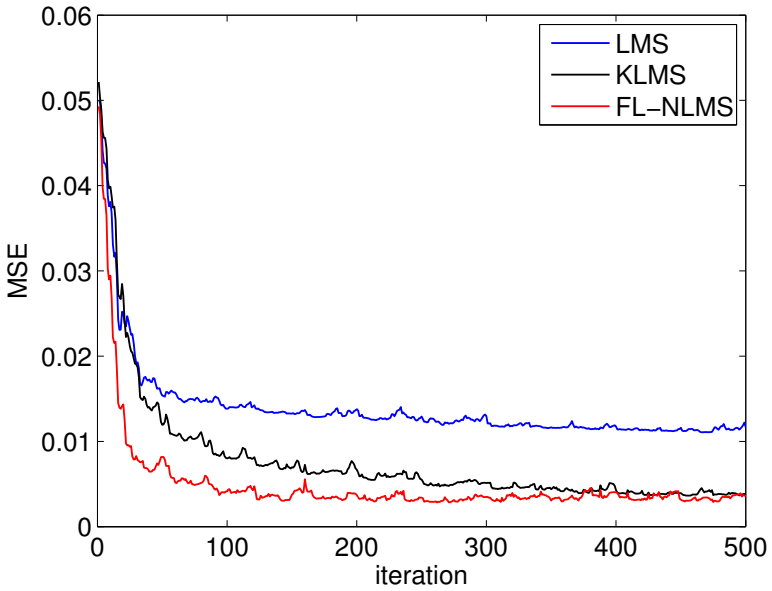
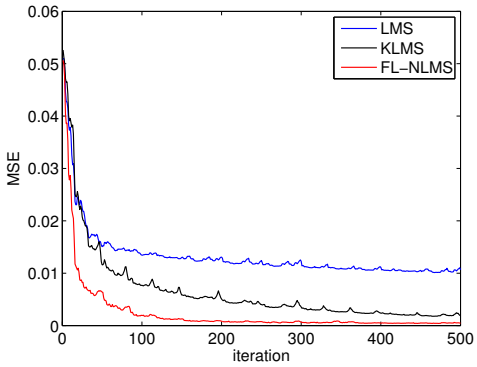
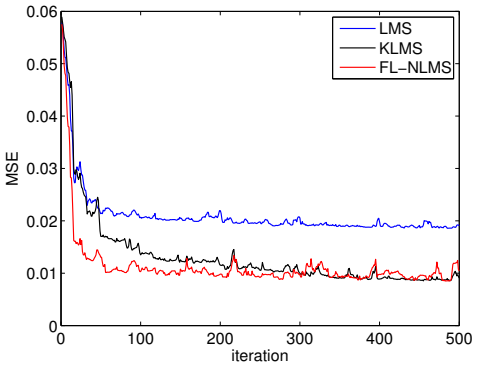


Figure 3.1: MSE for the three algorithms, using $\sigma = 0.04$

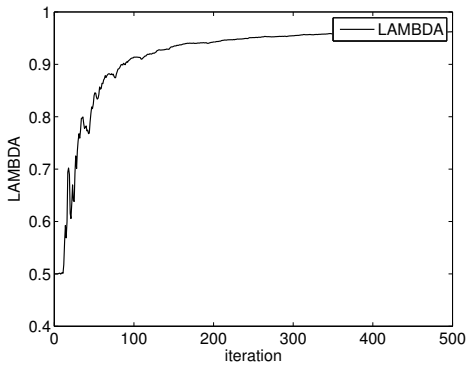


(a) MSE using $\sigma = 0.01$

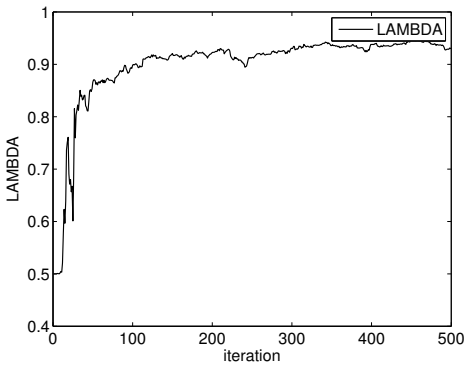


(b) MSE using $\sigma = 0.07$

Figure 3.2: MSE for the three algorithms, using $\sigma = 0.01$ and $\sigma = 0.07$



(a) LAMBDA for $\sigma = 0.01$



(b) LAMBDA for $\sigma = 0.07$

Figure 3.3: $\lambda[n]$ for NLMS-FL algorithm, using $\sigma = 0.01$ and $\sigma = 0.07$.

The time-series are corrupted using additive Gaussian noise with zero mean and a standard deviation equal to σ.

Figure 3.1 and 3.2 represents the evolution of the Mean Square Error on a test set of 100 samples, computed online during a training on 500 samples, using 3 different noise-standard-deviation values.

The time window size L for the samples was chosen equal to 10. In particular for the NLMS-FL method, due to its particular implementation, the dimension of the time-window used in input to nonlinear filter is chosen directly proportional to the linear filter's one, with a scaling factor K.

In the same conditions figures 3.3 and 3.4 show the evolution of the adaptive parameter $\lambda[n]$ used in the NLMS-FL algorithm.

In this first experiment learning parameter for LMS and KLMS is taken equal to

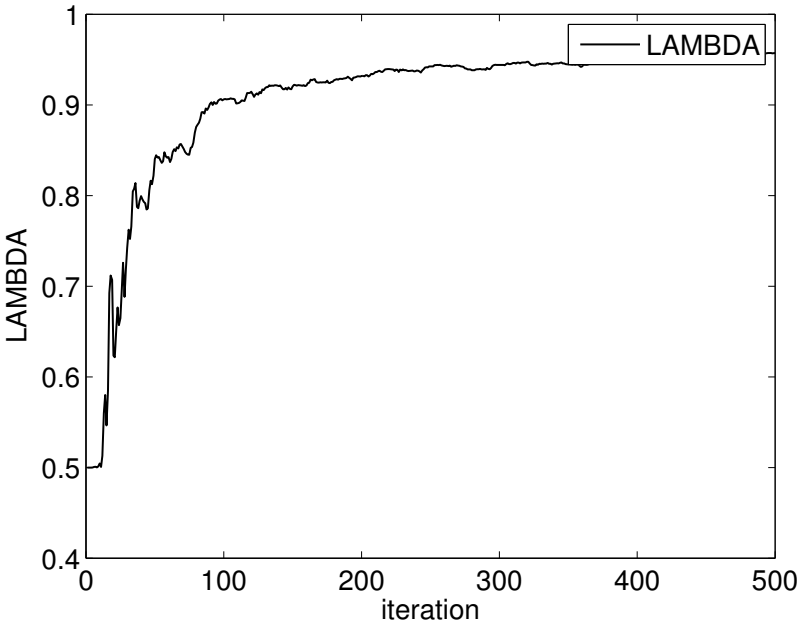
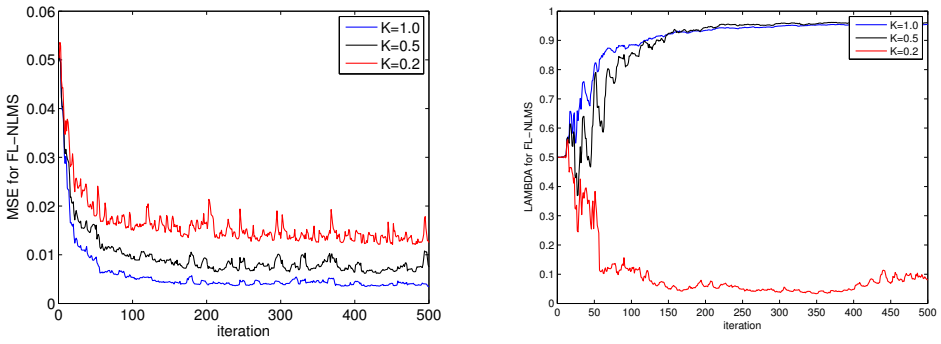


Figure 3.4: $\lambda[n]$ for NLMS-FL algorithm, using $\sigma = 0.04$

0.2, while for the NLMS-FL algorithm the step-sizes are taken $\mu_L = 0.1, \mu_{NL} = 0.3$, the expansion order is `exord` $= 3$, and the ratio between linear and nonlinear input pattern's buffer is $K = 1$.

The best performance of NLMS-FL with these particular parameters is confirmed by a series of tests performed in the same contest of prediction.

In fact, during a **second experiment** whose results are shown in figure 3.5,



(a) MSE for different K

(b) LAMBDA for different K

Figure 3.5: MSE and $\lambda[n]$ for NLMS-FL algorithm, using different sizes of K.

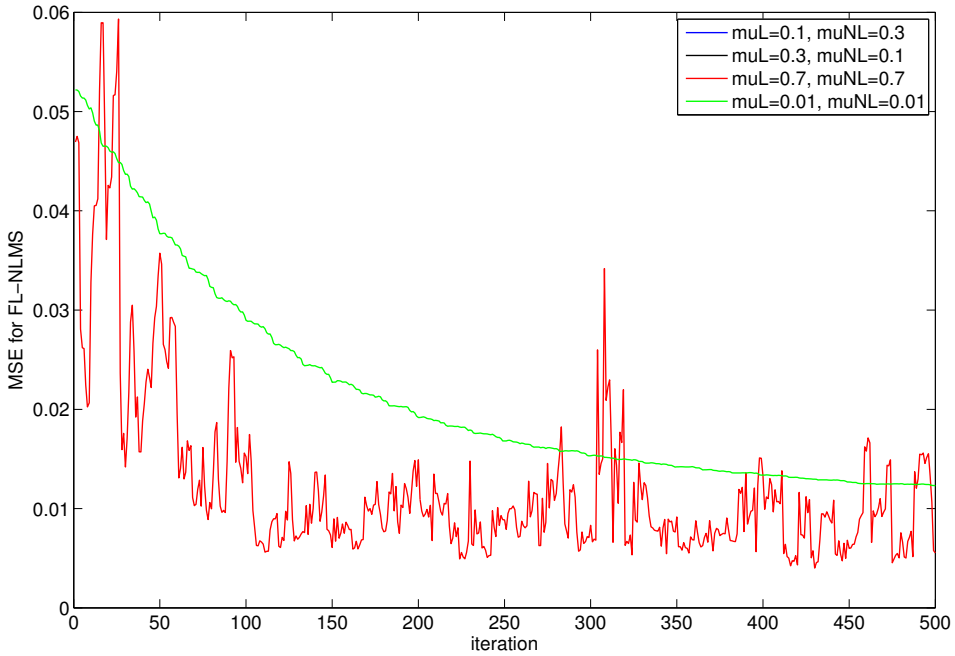


Figure 3.6: MSE for NLMS-FL algorithm, using upper and lower bounds, on linear and nonlinear learning parameters.

the NLMS-FL algorithm is tested with different values of K parameter.

Recalling that the pattern in input to nonlinear filter may be a subset of the one buffered for the linear filter, the algorithm is tested using at first time a size for the nonlinear buffer equal to half of linear one ($K = 0.5$). A second time its performance is measured using a size for the linear buffer five times bigger ($K = 0.2$) than the nonlinear.

In conclusion, as can be observed by MSE's plot in figure 3.5, the choice $K = 1$ was found to be the best. Remark that this choice implies an equal dimension for the time-windows used in both linear and nonlinear filters inside the collaborative network NLMS-FL.

A **third** and a **fourth experiment** aimed to find the best learning parameters and expansion order in the same contest of time series prediction, for the NLMS-FL. As regard the learning parameters associated to linear μ_L and nonlinear μ_{NL} filter, the experiment was divided in two parts. In first time the attempt to find upper and lower bounds for μ_L and μ_{NL} resulted in the values shown in figure 3.6 which clearly don't present a good efficiency. In figure 3.7 the behavior observed with a more coherent choice for the learning parameters is shown, underlining the necessity to choose a greater step-size for the nonlinear filter. Even if not shown in

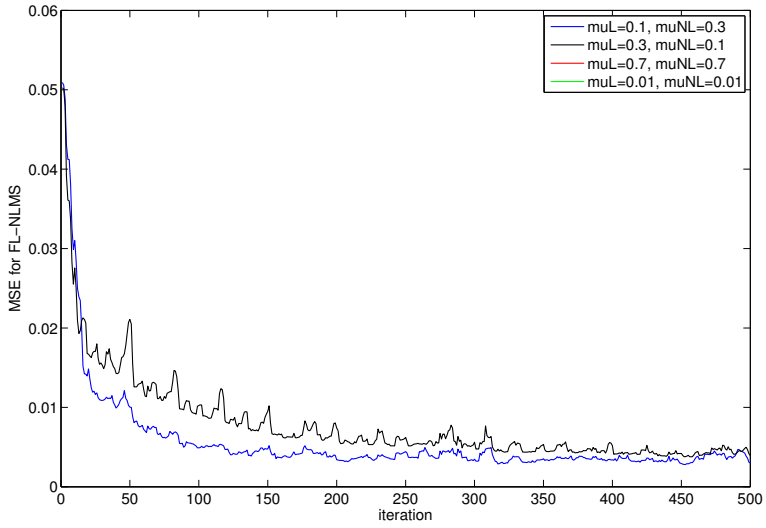
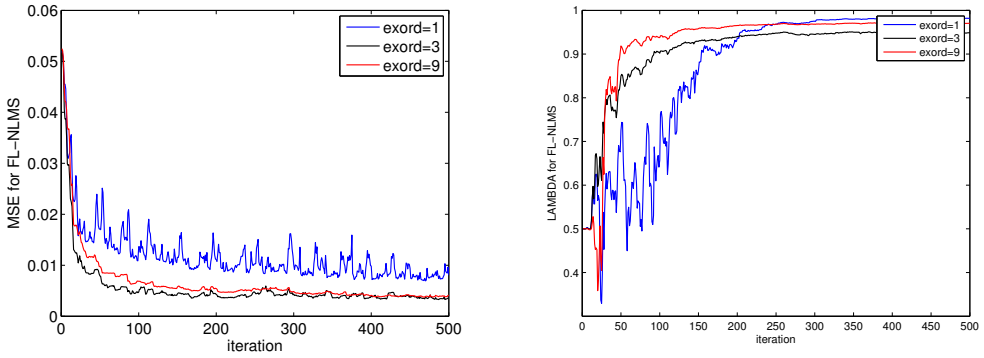


Figure 3.7: MSE for NLMS-FL algorithm, using coherent linear and nonlinear learning parameters.

figure the combinations $\mu_L = \mu_{NL} = 0.1$ and $\mu_L = \mu_{NL} = 0.2$ were tested pointing up the best performance is the one with $\mu_L = 0.1$ and $\mu_{NL} = 0.3$.

The expansion order of the input pattern determines the length of the enhanced pattern passed to the nonlinear filter as schematically represented in figure 2.1. The expansion order should be chosen carefully because it may compromise the performance of the algorithm if taken too high. In fact its size grows linearly with L_{in}, the size of the input pattern: i.e. $L_{en} = 2 \times \texttt{exord} \times L_{in} + 1$.



(a) MSE for different expansion orders (b) LAMBDA for different expansion orders

Figure 3.8: MSE and $\lambda[n]$ for NLMS-FL algorithm, using different expansion order.

As can be observed in the **fourth experiment**'s results shown in figure 3.8, if the expansion order is too small (e.g $exord = 1$) the error starts to oscillate and the system is not very stable. In addition a very big expansion order should be avoided to prevent small trade-off between time of computation and performance of the algorithm. The behavior described in 3.8 by the different choices of expansion order points out the optimal value is $exord = 3$.

For the case of NLMS-FL we recall that some parameters are constant in all the experiments. The regularization factors δ_L and δ_{FL} used in the computation of the normalization factors defined in 2.5 and 2.17, are taken equal to the mean of the squared variance computed on the input data set.

$$\delta = \delta_L = \delta_{FL} = 45\sigma_x^2 \tag{3.2}$$

As regard the power factor r, the adaptation parameter a, the smoothing factor β and the step size μ_a:

$$a[0] = 0, \qquad r[0] = 1, \qquad \beta = 0.7, \qquad \mu_a = 0.5 \tag{3.3}$$

3.2 Monte Carlo Simulations

In the end a **fifth experiment** was accomplished in order to validate the results and to give a more comprehensive comparison among LMS, KLMS and NLMS-FL. In the first test the parameters are taken with the optimal values already seen.

Algorithms	Training MSE	Testing MSE
LMS ($\sigma = 0.01$)	0.010160 $\pm$ 0.000089	0.010659 $\pm$ 0.000228
LMS ($\sigma = 0.04$)	0.012520 $\pm$ 0.000480	0.013121 $\pm$ 0.000959
LMS ($\sigma = 0.07$)	0.017379 $\pm$ 0.000796	0.018440 $\pm$ 0.002044
KLMS ($\sigma = 0.01$)	0.001818 $\pm$ 0.000032	0.001739 $\pm$ 0.000087
KLMS ($\sigma = 0.04$)	0.004160 $\pm$ 0.000339	0.004247 $\pm$ 0.000536
KLMS ($\sigma = 0.07$)	0.009077 $\pm$ 0.000854	0.009444 $\pm$ 0.001421
NLMS-FL ($\sigma = 0.01$)	0.000509 $\pm$ 0.000049	0.000550 $\pm$ 0.000099
NLMS-FL ($\sigma = 0.04$)	0.003655 $\pm$ 0.000513	0.004057 $\pm$ 0.000842
NLMS-FL ($\sigma = 0.07$)	0.010059 $\pm$ 0.001116	0.010637 $\pm$ 0.001967

Table 3.1: Monte Carlo Simulation results varying the noise

One hundred Monte Carlo simulations are run with different realization of the noise. In fact the noise is computed randomly at each of the 100 simulations. Furthermore these one hundred Monte Carlo simulations are performed three times: each time with different values for the noise standard deviation. Then a mean of the MSEs is computed for the Training Dataset and for the Testing Dataset. The results, summarized in table 3.1, confirm NLMS-FL is the best performer, especially in presence of low noise. Results, exhibited using the form `mean`$\pm$`standard deviation`, show best performer is NLMS-FL in condition of low noise ($\sigma = 0.01$), which reveals very low training (0.000509 ± 0.000049) and testing (0.000550 ± 0.000099) MSEs.

In last analysis table 3.2 summarize algorithms' behavior, setting different learning parameters and testing each case on 100 Monte Carlo simulations.

As expected, this last experiment confirms that the best algorithm for Mackey-Glass Time Series' prediction is NLMS-FL having learning parameters assigned as $\mu_L = 0.1$ and $\mu_{NL} = 0.3$. In that particular configuration the computed mean square error, after 100 Monte Carlo simulations, is 0.003647 ± 0.000440 in the training set, and 0.003901 ± 0.000644 in the test set, as is underlined in table 3.2. In order to model an ordinary scenario of noise variation the standard deviation was taken equal to 0.04.

Algorithms	Training MSE			Testing MSE		
LMS ($\mu = 0.1$)	0.013209	$\pm$	0.000426	0.013902	$\pm$	0.000913
LMS ($\mu = 0.2$)	0.012495	$\pm$	0.000435	0.013130	$\pm$	0.000966
LMS ($\mu = 0.6$)	0.013634	$\pm$	0.000936	0.014326	$\pm$	0.001267
LMS ($\mu = 0.9$)	0.016525	$\pm$	0.001752	0.017279	$\pm$	0.001967
KLMS ($\mu = 0.1$)	0.005959	$\pm$	0.000204	0.006194	$\pm$	0.000496
KLMS ($\mu = 0.2$)	0.004141	$\pm$	0.000334	0.004213	$\pm$	0.000453
KLMS ($\mu = 0.6$)	0.004210	$\pm$	0.001276	0.004372	$\pm$	0.001571
KLMS ($\mu = 0.9$)	0.005599	$\pm$	0.002009	0.005498	$\pm$	0.002065
NLMS-FL ($\mu_L = 0.1$, $\mu_{NL} = 0.3$)	0.003647	$\pm$	0.000440	0.003901	$\pm$	0.000644
NLMS-FL ($\mu_L = 0.3$, $\mu_{NL} = 0.1$)	0.004204	$\pm$	0.000443	0.004562	$\pm$	0.000656
NLMS-FL ($\mu_L = 0.7$, $\mu_{NL} = 0.7$)	0.009653	$\pm$	0.003956	0.010526	$\pm$	0.004457
NLMS-FL ($\mu_L = 0.01$, $\mu_{NL} = 0.01$)	0.011550	$\pm$	0.000308	0.012106	$\pm$	0.000847

Table 3.2: Monte Carlo Simulation results varying the learning parameters

List of Figures

List of Tables

References

[1] D.Comminiello, A.Uncini, R. Parisi and M.Scarpiniti, *A Functional Link Based Nonlinear Echo Canceller Exploiting Sparsity.* INFOCOM Dpt., Sapienza, University of Rome.

[2] D.Comminiello, A.Uncini, M.Scarpiniti, L.A.Azpicueta-Ruiz and J.Arenas-Garcia, *Functional Link Based Architectures For Nonlinear Acoustic Echo Cancellation.* INFOCOM Dpt., Sapienza, University of Rome, and STC Dpt., Carlos III de Madrid, University of Madrid

[3] Y.H.Pao, *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Adisson-Wesley, 1989.

[4] A.Uncini, *Neural Networks: Computational and Biological Inspired Intelligent Circuits.* INFOCOM Dpt., Sapienza, University of Rome.

[5] Weifeng Liu, Jose C. Principe, and Simon Haykin *Kernel Adaptive Filtering: A Comprehensive Introduction.* John Wiley & Sons, Inc., Publication

[6] Dave Touretzky and Kornel Laskowski, *Neural Networks for Time Series Prediction.* 15-486/782: Artificial Neural Network, FALL 2006. School of Computer Science, Carnagie Mellon.