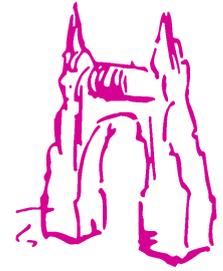




UNIVERSITAT DE
BARCELONA



FACULTAT DE
FARMÀCIA

Electronic tongues multivariate calibration for the analysis of metals

Treball Final de Grau

Ana Rosselló Busquet

March 2017

MAIN FIELD: QUÍMICA ANALÍTICA

SECONDARY FIELDS: MATEMÀTICA I INFORMÀTICA

SANITAT I GESTIÓ AMBIENTAL



This work is licenced under a Creative Commons license

Table of Contents

TABLE OF CONTENTS	2
ABSTRACT	3
RESUM	3
INTRODUCTION	4
ELECTRONIC TONGUES	4
BASELINE	5
DATA COMPRESSION.....	5
ARTIFICIAL NEURAL NETWORKS.....	6
OBJECTIVES	10
INTEGRATION AMONG STUDY AREAS.....	10
MATERIALS AND METHODS	11
ELECTRONIC TONGUE DESIGN	11
VOLTAMMETRIC MEASUREMENTS	11
SOFTWARE	11
BASELINE	12
COMPRESSION	13
ANN	15
RESULTS	18
BASELINE	18
COMPRESSION	18
ANN	19
CONCLUSION	24
FURTHER WORK	24
REFERENCES	25
ANNEX	27
I. REMOVE BASELINE.....	27
a. <i>Tested configurations</i>	27
b. <i>MATLAB Implementation</i>	27
function test_baseline(sensor,E).....	27
function [sensor_bck, baselines,slope_E,slope_B, area]= remove_baseline(sensor,E, ord,s,fc)	28
c. <i>Results</i>	28
II. COMPRESSION	28
a. <i>Tested configurations</i>	28
b. <i>MATLAB Implementation</i>	29
function [wname_over95 fc_over95 lev_over95]= test_compression(sensor)	29
function [sensor_comp, fc]= compression(sensor, wname,lev)	29
c. <i>Results</i>	29
III. ARTIFICIAL NEURAL NETWORK	30
a. <i>Tested configurations</i>	30
b. <i>MATLAB Implementation</i>	30
function [net]= ANN_create_1c(hiddenLayerSize)	32
function ANN_test_comp_fitnet_LM_2_DIF_DIF(sensors, conc, test_sensor,test_conc).....	33
c. <i>Results</i>	34

Abstract

Heavy metals are substances found in the environment that can come from both natural and anthropogenic sources. There is an increasing interest in analytical methods with high selectivity and sensitivity to perform environmental monitoring to detect adulterations or contamination. Electronic tongues are especially interesting as they can respond to a large variety of substances. This report presents the developed MATLAB software used to calculate the concentration of certain substances using data from a voltammetric electronic tongue with four sensors to determine concentration of six metal ions using Artificial Neural Networks. Different ANN configurations have been tested; where the parameters have been changed, such as learning algorithm, transfer function and number of hidden layers/neurons. Furthermore, the input of the ANN has been pre-processed. A MATLAB function has been developed to remove the baseline and another to compress the signal with Discrete Wavelet Transforms. In order to find the best baseline removal and compression configuration different set-ups varying the possible parameters have been tested. The results obtained in this first part of the project are favourable as the baseline has been successfully removed and the data have been compressed from 420 samples to 95 coefficients. The results obtained for ANN prove that more number of hidden layers or neurons/layer does not imply a better performance. Better results are obtained with 2 hidden layers and 5-2 neurons. Even though during the training of the ANN good values are obtained, when the network is feed with other testing data the results are less accurate.

Resum

Els metalls pesant són substàncies que es troben en el medi i poden provenir de fonts naturals o antropogèniques. Els mètodes d'anàlisi amb alta selectivitat i sensibilitat són d'interès per dur a terme la vigilància ambiental, detectar adulteracions o contaminació. Les llengües electròniques són interessants ja que poden respondre a una gran varietat de substàncies. En aquest treball s'ha fet un programa en MATLAB per calcular la concentració de certes substàncies utilitzant dades d'una llengua electrònica formada per quatre sensors per determinar la concentració de sis ions metàl·lics usant xarxes neuronals artificials. S'han provat diferents configuracions; on s'han variat els paràmetres de la ANN, i.e algoritme d'aprenentatge, funció de transferència i el nombre de capes ocultes/neurones. A més, el senyal d'entrada de l'ANN ha estat pre-processat. S'ha desenvolupat una funció MATLAB per eliminar la línia base i un altre per comprimir el senyal amb Wavelet Discret Transform. Per tal de trobar la millor eliminació de la línia base i la configuració de compressió s'han variat els possibles paràmetres. Els resultats obtinguts en aquesta primera part del projecte són favorables; la línia base s'ha eliminat amb èxit i les dades s'han comprimit de 420 mostres a 95 coeficients. Els resultats obtinguts per a ANN demostren que un major nombre de capes o neurones ocultes/capa no implica un millor rendiment. S'obtenen millors resultats amb 2 capes ocultes i entre 5-2 neurones. Encara que s'obtenen valors favorables durant l'entrenament de l'ANN, quan la xarxa és alimentada amb altres dades, els resultats no són tan precisos.

Introduction

Heavy metals are substances found in the environment that can come from both natural and anthropogenic sources (1). These substances are not biodegradable and tend to bioaccumulate and biomagnificate (2). Some small concentrations of heavy metals are essential for ecosystems, however, higher concentrations can be harmful for humans and other species (3,4). Therefore, it is essential to monitor heavy metals to guarantee both human and environmental health.

In the last years different techniques for determination of heavy metals have been used, such as: atomic absorption spectroscopy, inductively coupled plasma atomic emission spectroscopy or inductively coupled plasma mass spectrometry (5). These systems require expensive instruments, trained personal and long time of analysis.

There is an increasing interest in analytical methods with high selectivity and sensitivity to screen products or environmental monitoring to detect adulterations or contamination. Among these, electronic tongues are especially interesting as they can respond to a large variety of substances. They provide analytical information at low cost in real time with a direct and relatively simple measuring setup. On the other hand, they require larger amounts of samples for calibration and generate large amounts of data which need non-trivial processing.

This report presents the developed MATLAB software used to calculate the concentration of certain substances using data from a voltammetric electronic tongue with four sensors to determine the concentration of six different metal ions.

In this section an overview to the different concepts related to electronic tongues and the processing methods used are provided.

Electronic tongues

According to the IUPAC definition, an electronic tongue (ET) is "a multisensor system, which consists of a number of low-selective sensors and uses advanced mathematical procedures for signal processing based on Pattern Recognition and/or multivariate data analysis (artificial neural networks), principal component analysis, etc."(6)

Electronic tongues generate multidimensional information from a sensor array applied to liquid samples. This analytical system also includes a data processing stage to extract coherent information from the complex data. For instance, the data used in this work is a liquid sample with certain substances and electronic tongues are used to calculate their concentration.

Different families of sensors exist: potentiometric, voltammetric and impedimetric. It is even possible to combine responses obtained from sensor of different families. Nevertheless, this work will focus on a voltammetric electronic tongue which is constituted by four screen-printed modified electrodes. Every electrode provides a voltammogram, i.e., a series of currents as a function of the applied potential as illustrated in Figure 1. These different types of electrodes ensure the desired cross-response and have been used but have non-stationary characteristics. Their selectivity and detection limits are different which makes the system more sensitive and accurate.

The data analysis usually consists in two steps: pre-process data and relate readings with outputs. The first step may consist of different procedures such as remove redundant information, enhance signal-to-noise ratio reduce interference effects and/or drifts or non-linearity in the readings from the various sensors. Principal Component Analysis (PCA) is widely used for this pre-processing step. Additionally, a recommended pre-treatment step is to normalize the range of the different input channels to avoid any imbalance between them (7).

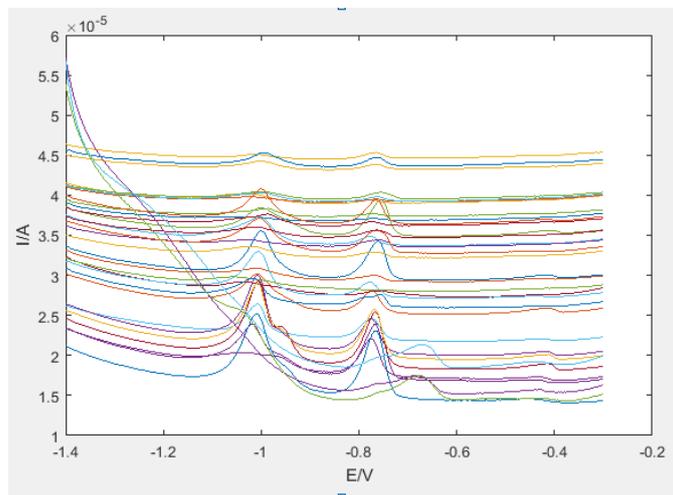


Figure 1: Example of a voltammogram obtained from electronic tongues

On the second step different numeric analysis can be implemented (6,8):

- Qualitative models: Partial Least Squares- Discriminant Analysis (PLS-DA), Linear Discriminant Analysis (LDA) or Support Vector Machine (SVM) and used of tri-linear approaches such as PARAFAC or Tucker models.
- Quantitative models: Multiple Linear Regression (MLR), Principal Components Regression (PCR), Multi-way PLS and Artificial Neural Networks (ANN).

More details on data processing models can be found in (8).

This work will be focused on different compression strategies using wavelets and removing baseline of the signals as a first step. In the second step ANN will be applied to estimate the concentration of each substance.

Baseline

The measurements obtained with the electrodes contain a significant baseline; caused by different reasons such as capacitive currents and impurities. This baseline is significant and not constant through the measurements and therefore is critical to remove it correctly. Removing the baseline or an estimation of it from the spectrum leads to a more interpretable signal allowing to determine peaks and its amplitude more accurately(9). Therefore this substitution of the baseline is done before compressing the signal.

To remove it, the approach presented in (9) is used. According to this method, the baseline is fitted to a low-order polynomial where the polynomial coefficients are estimated by minimising a non-quadratic criteria. The authors have designed four cost functions that avoid the peaks to be too influent on the estimation: symmetric/asymmetric Huber function and symmetric/asymmetric truncated quadratic. In addition to the cost function the hyperparameters have to also be set to perform the removal of the baseline: threshold and polynomial order.

Data Compression

Each voltammogram is made of hundred of measurements leading to a great amount of data (*samples x measurements x sensor*). These can easily be processed by PCA or PLS, however this is not the case for ANN (7). In order to reduce this amount of data compression is used. Additionally, compression also extracts significant features, eliminates irrelevant content, such as noise giving measured signals a clear shape or profile (10,11).

Different data compression strategies like PCA (12), Fourier analysis (13,14), Hadamard transform or Discrete Wavelet Transforms (DWT) (15) can be used. In this project DWT will be used and the different possibilities in this type of compression will be studied. DWT is based on decomposing the signal into different wavelet functions, scaled and translated versions of a base function (mother wavelet). DWT is better at describing non-stationary signals than Fourier transform (16) and the authors in (17) concluded that DWT-ANN option performed better than down sampling and PCA-ANN. This is due to the fact that wavelet transform was developed for the analysis of non-stationary signals and as PCA is a linear projection method that fails to preserve the structure of a non-linear data set.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are broadly used in chemical data-mining, because ANNs are powerful modelling tools for deriving the meaning from complex or imprecise data (18). ANN is a parallel distributed network of connected processing units and are used as powerful non-linear modellers applicable for quantitative applications (7) such as the one studied here. The first studies describing ANN were performed by the authors in (19,20) and the initial idea was based on the human brain functioning. These networks basic unit is called a neuron, which is mathematically represented in Figure 2. Inspired in neural physiology, each neuron is constituted of a summing function which integrates the contributions of different inputs (the experimental values or neurons in a previous layer) and an activation or transfer function ($\phi(v)$) that controls the propagation of the neuron signal to the next layer. Furthermore these inter-neuron connections are called synapses which have associated synaptic weights (w_j).

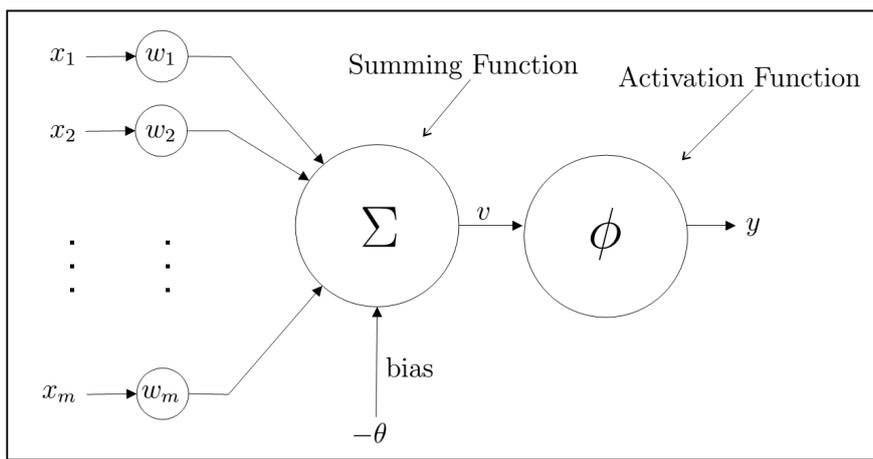


Figure 2 Mathematical Model of a Nonlinear Neuron (21)

The most common transfer functions, depicted in Figure 3, are as follows:

- Threshold or Heaviside Function: used to model all-or-none property. A neuron whose activation function is a function like this is called a perceptron or a McCulloch-Pitts model (19).
- Linear Function: used as an approximation to a non-linear amplifier.
- Sigmoid or hyperbolic tangent Function: This is the most common form of activation function used in artificial neural networks.

However MATLAB toolbox offers other transfer functions that have also been used by researchers and will be discussed later on in this section. In the program developed here the transfer functions tested are: logarithmic sigmoid, symmetric sigmoid, positive saturating linear and linear transfer function.

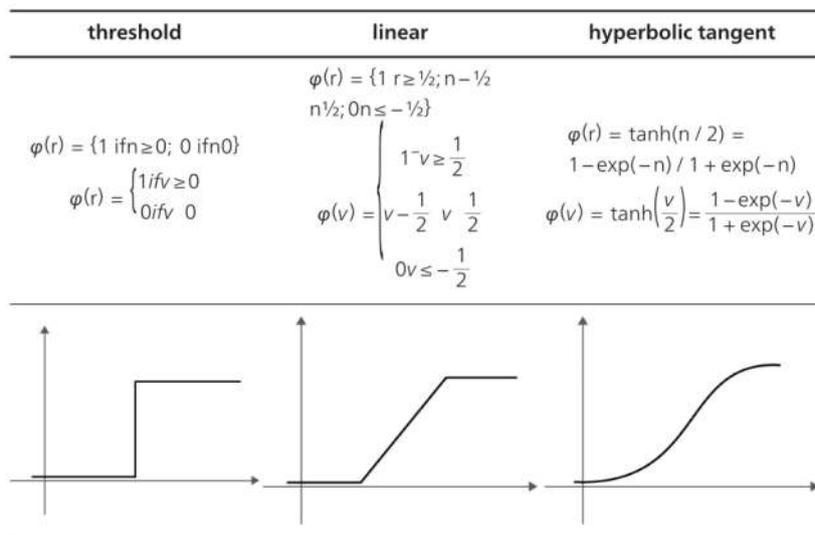


Figure 3: Basic activation functions used in ANN studies (22).

To form an ANN associated neurons are used. Neighbour neurons form a layer and further layers can be concatenated, each taking its input from the previous layer outputs. If the network only consists of input and output neurons it is called a single-layer. If there are layers in between input and output, called hidden layers the network is referred to as a multi-layer network.

Furthermore, depending on the connection between neurons different classes of network architectures can be built (23):

- Feed-Forward networks: are characterized by unidirectional connections between network layers (one direction flow from input to output).
- Recurrent networks: are characterized by their loops as connection among layers occur in both directions. In these cases information is no longer transmitted only in one direction but it is also transmitted backwards. Some recurrent artificial neural networks are Hopfield, Elman, Jordan and bi-directional.

These network architectures are depicted in Figure 4.

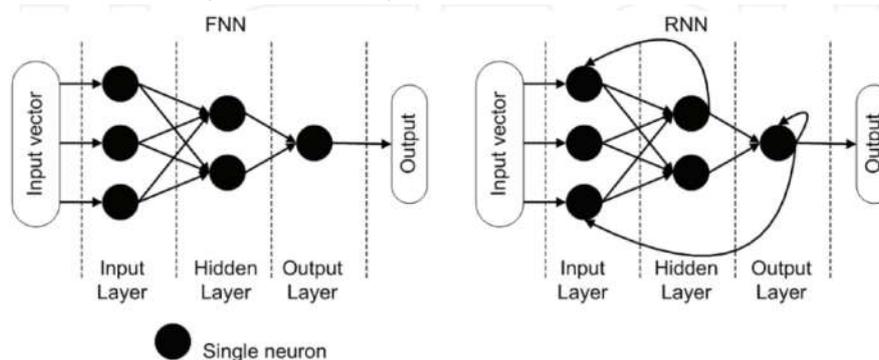


Figure 4: Feed-forward (FNN) and recurrent (RNN) topology of an artificial neural network (24)

Building an ANN topology is half the task before we can use this artificial neural network for solving a given problem. In the next step for setting up the network, the system undergoes a training/learning procedure.

It has to be considered that during neural training overfitting problems may occur: the error on the training set is a very small value but other new data presented to the model show a larger error. There are two methods used to avoid overfitting (7): Bayesian regularization and early stopping (use validation set to check if overfitting is taking place). The Bayesian regularization algorithm, does not make use of an internal validation stage and does not require an extra subset for this purpose (25).

During the training of an ANN several learning strategies can be used. They are classified in three main groups depicted in Figure 6 and described below:

- Supervised Learning: this technique uses the training data to set parameters of an artificial neural network. The training data consists of pairs of input data, in this case the sensors measurements, and the output, the metal ions concentration. This training process is used to find the best parameters for each neuron. Different supervised learning algorithms exist: multilayer perceptron, Support Vector Machines, k-nearest neighbour algorithm, Gaussian mixture model, Gaussian, naive Bayes, decision tree, radial basis function classifiers and backpropagation. These types of network require a validation set to test the performance of learned artificial neural network. Validation data set consist of new data, which has not been previously introduced to artificial neural network (24).
 - Backpropagation network: is a subtype of supervised learning and it reassembles RNN as the output error is used to train the network but there are no back-loops. The network learns by example, after choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. As most algorithms, the training stops when the value of the error function has become sufficiently small.
- Unsupervised Learning: this technique does not have a training data set with known answers. This type of learning is useful in cases such as statistical modelling, compression, filtering, blind source separation and clustering. It uses cost functions which can be any function and it is determined by the task formulation. Self-organizing maps are the ones that the most commonly use unsupervised learning algorithms (24).

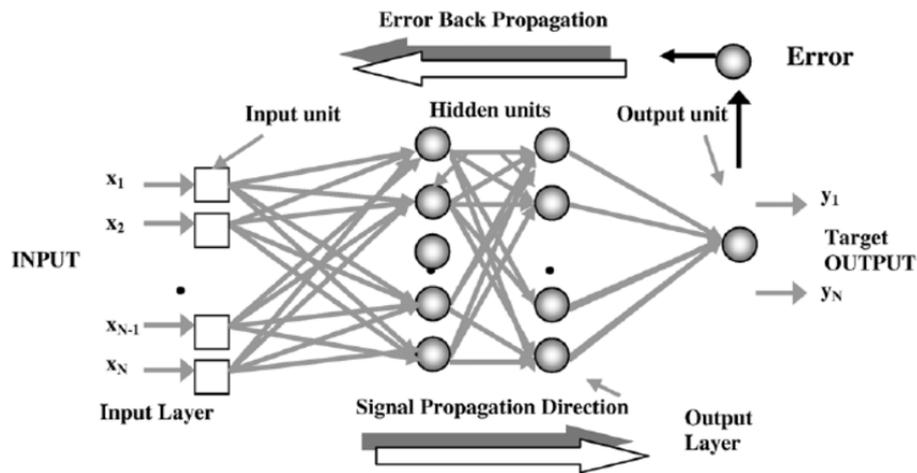


Figure 5 backpropagation neural network (26)

- Reinforcement Learning: this technique uses data which is generated by interactions with the environment to maximize some notion of long-term reward. Using the generated output and several algorithms the ANN finds the policy which produces the maximum return. This type of learning is used for robot control, telecommunications, and games such as chess and other sequential decision making tasks and is also frequently used as a part of artificial neural network's overall learning algorithm (23).

Classification of Learning Algorithms

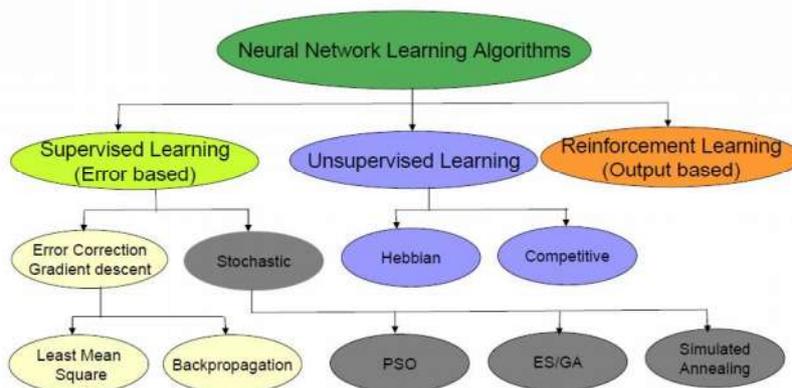


Figure 6 Classification of learning algorithms (27)

Not all of them have been tested in this project; only supervised learning was tested.

The type of ANN has been chosen following related biography. In the list below some of the recent approaches for ANN and voltammetric electronic tongues are summarized:

- In (17) a feed-forward backpropagation network to determine the concentration of three amino acids was designed. It consisted of two hidden layers and one output layer. The input matrix is formed by 72 samples (columns) with 16 coefficients of DWT using the Daubechies mother wavelet (rows) for each one. The first hidden layer had six neurons and tangent-sigmoidal transfer function, the second hidden layer had 24 neurons and logarithmic-sigmoidal transfer function, and the output layer had a single neuron and linear transfer function. The used training algorithm was the Bayesian regularization algorithm, which avoids overfitting without the need to monitor the fitness degree of an internal validation subset (28). The goal for convergence training they use was a sum of squared error (SSE) of 0.001 in 200 or less training epochs. SSE was calculated as follows:

$$SSE = \sum_{j=1}^N (c_{expected_j} - c_{calculated_j})^2$$

where c is the concentration value and N the number of training samples.

To select among the different ANN configuration they use a percentile relative absolute error (PRAE) to evaluate the different networks' performance:

$$PRAE = \frac{1}{M} \sum_{i=1}^M \frac{|c_{expected_i} - c_{calculated_i}|}{c_{expected_i}} \times 100$$

where c is the concentration value obtained experimentally and calculated with ANN and M the number of testing samples.

- In (15) different feed-forward backpropagation networks are tested, but all have an input layer of 16 neurons corresponding to the 16 coefficients obtained from DWT. They tested two set ups where one network will provide three outputs or three networks with only one output to calculate the concentration of the substances studied. They tested networks with one or two hidden layers with variable number of neurons, transfer functions (logsig: log sigmoid, tansig: hyperbolic tangent sigmoid; and purelin: linear transfer function) and different supervised learning (conjugate gradient, descent gradient, descent gradient with adaptive learning rate, descent gradient with momentum, descent gradient with momentum and adaptive learning rate, Levenberg–Marquardt optimization algorithm and the best results with Bayesian regularization algorithm).
- In (29) 16 approximation coefficients are used as input and to output setups: one network with three outputs and two hidden layers with 10 and 5 neurons; and three networks with one output and two hidden layers with 6 and 24 neurons. The network is a feed-forward topology and Bayesian regularization learning algorithm. The error goal for training used was a SSE of 0.01, to be reached in 300 epochs of training or less.
- In (30) they use a training algorithm (Bayesian regularization) and as input the coefficients of the DWT (Daubechies) and FFT parameters. For the ANN, different parameters were tested and are summarized in Figure 7 and the maximum of training epochs were set at 200.

To select the best model the predicted concentrations for the three amino acids were compared with the expected ones. The best regression parameters (slope (m) and correlation value (r) close to 1 and intercept (b) close to 0) and the lowest root-mean-square error (RMSE) and normalized root-mean-square error (NRMSE).

The best results were achieved without any autoscaling of the data, by FFT compression and using 160 input neurons, 1 single hidden layer formed by 7 neurons, the purelin hidden layer transfer function and satlins (symmetric saturating linear transfer function) output transfer function and a residual error of 10^{-10} M.

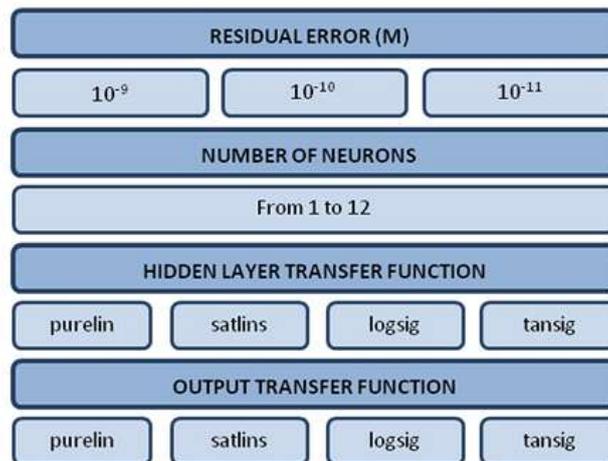


Figure 7: ANN parameters tested by authors in (30)

- In (18) a feed-forward and backpropagation architecture and Bayesian Regularization learning algorithm was used. The input layer consisted of 23 neurons with a linear transfer function and the output layer consisted of three neurons. The number of neurons in the hidden layer was changed from 1 to 10 and combinations of tansig, logsig, purelin, satlin (saturating linear transfer function) and satlins were tested. RMSE was used for an overall comparison. The optimum configuration had 5 neurons and logsig transfer function in the hidden layer and 3 neurons and purelin transfer function in the output layer.

Objectives

In this project electronic tongues used to measure heavy metals have been presented. The data obtained from this sensor array has to be processed to obtain the estimated concentrations of heavy metals. In this project different baseline removal, compression and ANN are tested. The objectives of these project are to study ANN and how they can be developed using MATLAB to improve the results previously obtained in the data treatment of the voltammetric tongue results in (5) with a simpler, linear method like PLS.. There are two ways to do this implementation using the neural network fitting tool GUI (Graphical User Interface) or by coding the network using the MATLAB toolbox commands. The baseline removal and compression algorithms have also been developed in MATLAB.

To summarize the main objectives are:

- study electronic tongues and how they are used to measure heavy metals
- study and test baseline removal algorithms
- study and test compression algorithms: focus on DWT
- study and test ANN
- find best set up for the input data used
- develop re-useable MATLAB functions for further use in the Department of Chemical Engineering and Analytical Chemistry

Integration among study areas

This project involves the following study areas:

- 17. Matemàtica i informàtica: developing a program in MATLAB for baseline removal, compression and ANN involves applying programming skills in MATLAB language and also math. To implement such programs MATLAB works with matrices and the performance is measured using statistical methods.
- 21. Química analítica: The input data consists on data obtained from electronic tongues to analyze the concentration of heavy metals. These sensors have been developed in the Department of Chemical Engineering and Analytical Chemistry
- 26. Sanitat i gestió ambiental: Electronic tongues have the drawback that the data obtained with them require processing to obtain the concentrations of heavy metals (not a linear/trivial relation between measurements and concentrations). However, if a more accurate and reliable processing algorithm/implementation is developed, these sensors could be used to measure heavy metals concentration in rivers, lakes and sewers among others. Being able to have a fast and cheap method to take these measurements will help in having a better knowledge and control in environmental management.

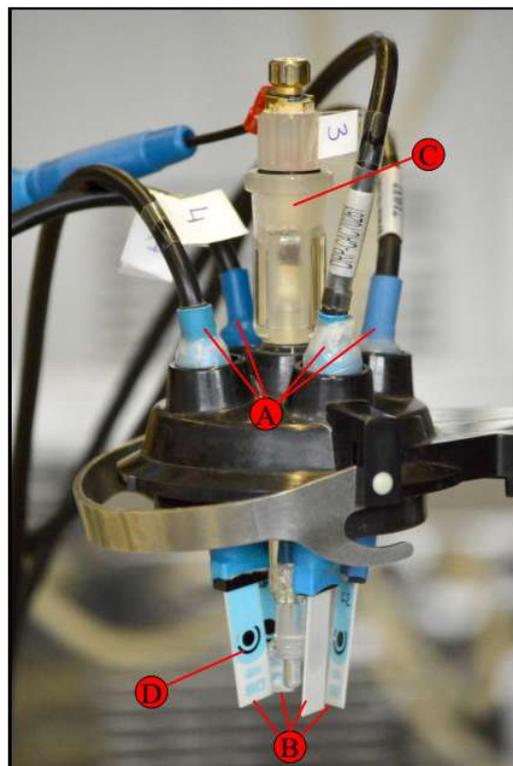
Materials and Methods

Electronic tongue design

The data set for this project was obtained with a voltammetric electronic tongue constituted by four screen-printed modified electrodes developed by the authors in (5):

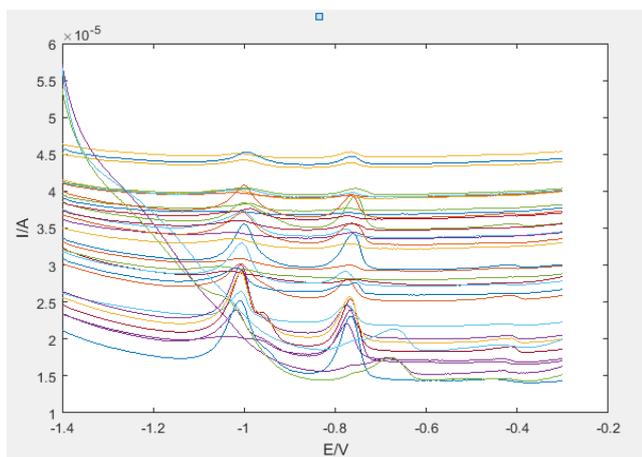
- Carbon nanofiber modified electrode (SPCE-CNF)
- Ex-situ antimony film electrode prepared from carbon nanofiber modified electrode (ex-situ-SbSPCE-CNF)
- Carbon nanofiber chemically modified with Cys electrode (Cys-SPCE-CNF)
- Carbon nanofiber chemically modified with GSH electrode (GSH-SPCE-CNF)

Figure 8: Experimental setup for simultaneous stripping voltammetric measurements with four different screen-printed electrodes. A) DRP-CAC connectors. B) Screenprinted electrodes. C) Ag/AgCl reference electrode. D) Carbon auxiliary electrode (5).



Voltammetric measurements

The data set used in this project is the result of the analysis of solutions containing different heavy metal concentrations with the above described voltammetric tongue. Using the electrodes presented in the previous section, stripping voltammetric measurements were performed by applying a deposition potential (E_d) of $-1,4$ V and an anodic (oxidation) scan from $-1,4$ V to $-0,4$ V in differential pulse mode. Details on chemicals, solutions, instrumentation and how these measurements were performed can also be found in (5).



The measurements correspond to a total of 80 metal mixtures samples which are used for the simultaneous determination of Zn(II), Cd(II), Pb(II), Tl(I), In(III) and Bi(III) ions inside the concentration range 0 to 200 $\mu\text{g/L}$ and are represented in Figure 9.

These voltammograms have been modified to eliminate electrodes anomalies and to reduce the effects of the baseline and only the measurements 81 to 500 are used.

Figure 9: The four subfigures: I1, I2, I3, I4 represent the voltammograms of each sensor for all the samples analyzed

Software

MATLAB version 9.0.0.34360 R2016a was used to implement DWT and ANN modelling which are then used to process the data collected from the electronic tongue. This program has been chosen as it provides Wavelet Toolbox (Version 4.16) and Neural Network Toolbox (Version 9.0 (R2016a)). In addition, the baseline removal was done employing the background estimation by minimizing a non-quadratic cost function (9).

Figure 10 is an schematic depicting the main concepts of the software developed. The measurements are first fed to baseline removal algorithm and then compressed using wavelet transform. These values are then used as an input for the ANN.

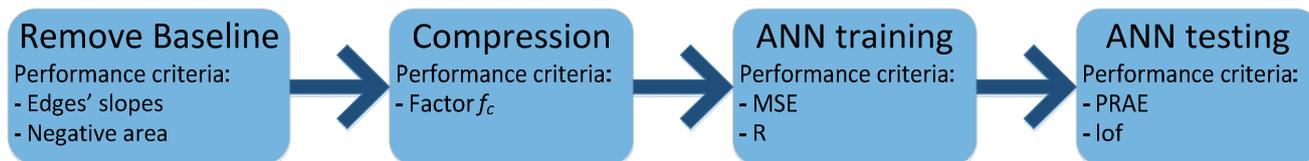


Figure 10: Schematic of software developed concepts.

Baseline

To remove the baseline from the signals obtained from the electronic tongues, the approach presented in (9) is used. It fits the baseline to a low-order polynomial where the polynomial coefficients are estimated by minimising a non-quadratic criteria. The authors have designed four cost functions that avoid the peaks to be too influent on the estimation: symmetric/asymmetric Huber function and symmetric/asymmetric truncated quadratic. In addition to the cost function the hyperparameters have to also be set to perform the removal of the baseline: threshold and polynomial order.

The criteria used to choose the best type of low-order polynomial to remove the baseline are:

1. Avoid raising the ends of the function
2. Avoid negative values

These are two criteria that are usually evaluated by the person analysing the signals. However in order to ease this task three parameters have been calculated: slope beginning to first minimum, slope from last minim to end of signal and negative area. The slope values can help evaluate how the signal rises; more slope more the edges are rising and therefore worst baseline removal. The negative area refers to the total area between x axis < 0 and the signal. All this parameters are represented in the Figure 11.

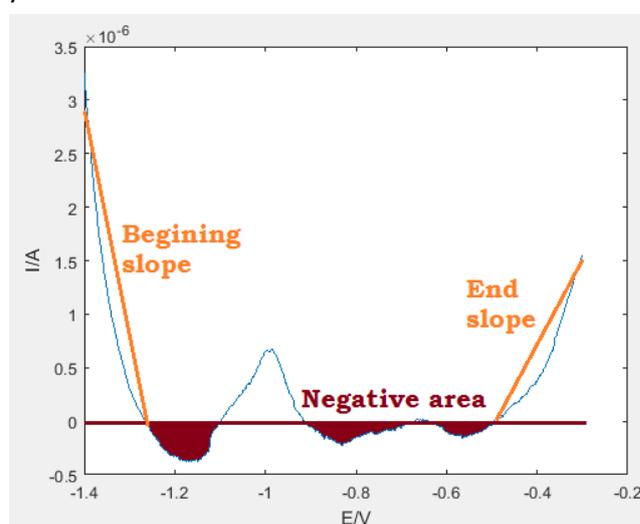


Figure 11: Representation of performance parameters for baseline removal.

To find the most suitable baseline removal function different estimations, varying all the parameters have been performed and the slopes and negative area have been calculated: functions: symmetric Huber function, asymmetric Huber function, symmetric truncated quadratic, asymmetric truncated quadratic; polynomial order from 1-10; thresholds: 0.001-0.1 (steps of 0.002). The program then chooses the best 10 estimations: it first selects the estimations which have slope lower than threshold provided by the user and then from those it selects the 10 with less negative area.

Figure 12 Compares an example of the sensor's raw data and data without baseline.

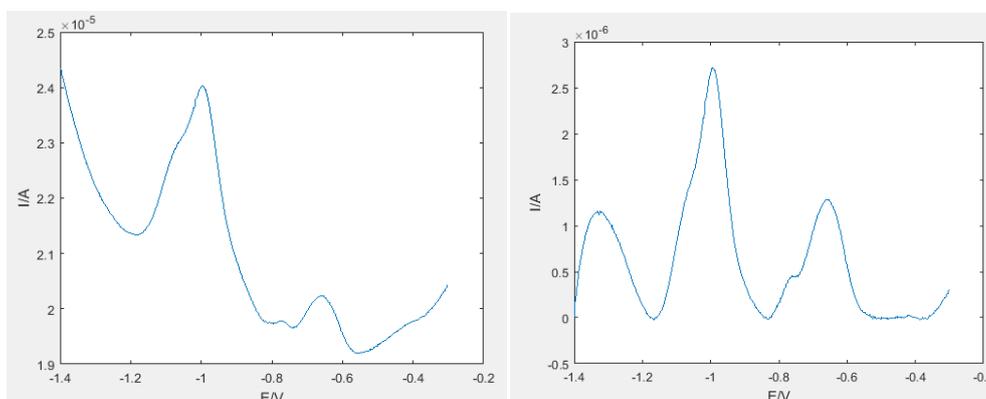


Figure 12: a) Raw voltammogram data from one sample of sensor SPCE-CNF; b) Voltammogram data from sensor SPCE-CNF after baseline removal.

The MATLAB implementation can be found in the Annex I.b. and a schematic and brief description of the different functions is shown in Figure 13. Different set-ups were tested for each set of voltammograms (IX, corresponding to 4 sets of data (4 sensors) with 80 samples): polynomial order between 1 to 10, threshold between 0,01 to 0,1 and cost function: sh, ah, stq, atq. As output of this software, the best 10 baseline removal set-ups for each cost-functcion and sensor are obtained. The user can then chose the most suitable one.

The test function is just used to set the input options, to remove the baseline another function is used. This function requires an specific set-up (polynomial order, threshold and cost function) and returns the values for that set up of slopes and negative area, whereas the test function returns a vector for those parameters (each position corresponding to one set-up). It has to be considered that each sensor has 80 samples and that the baseline removal is the same for all. Therefore the slope beginning (slope_B), sloped end (slope_E) and negative area are presented as the mean of those samples.

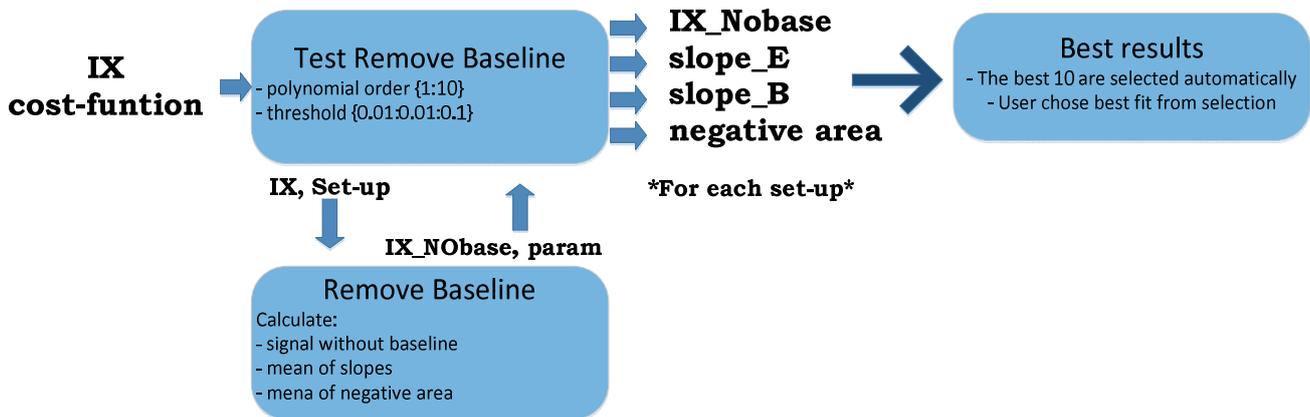


Figure 13: MATLAB functions developed for baseline removal

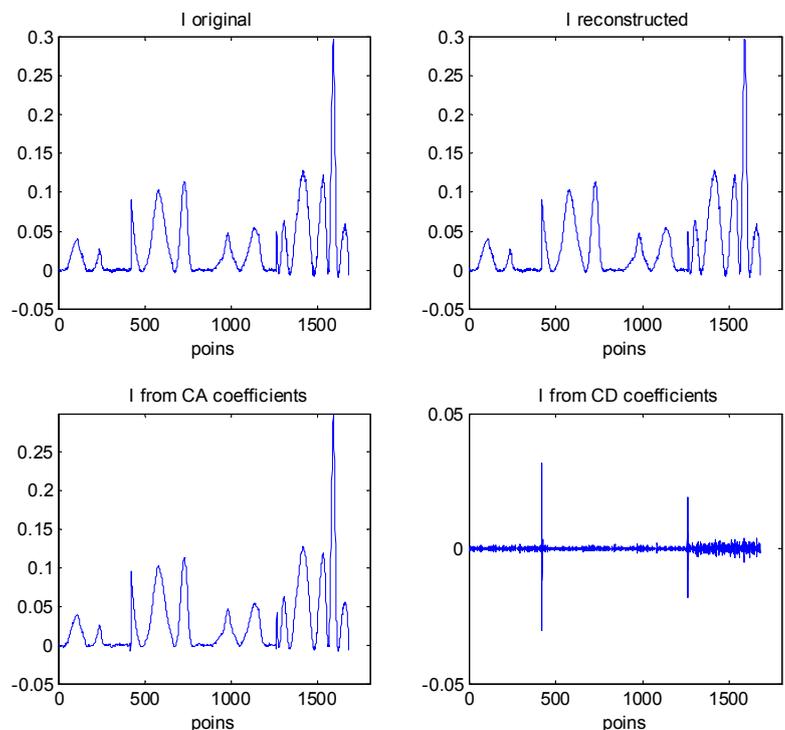
Compression

The compression used is DWT, based on representing a function $f(t)$ with wavelet functions, generated by translating and scaling this mother wavelet. The MATLAB function `dwt` can be used, which performs a single-level one-dimensional wavelet decomposition (31). This function returns two types of coefficients of half the length of the original signal:

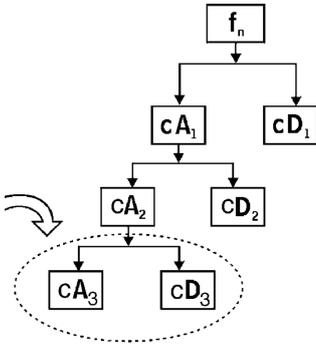
- CA or approximation coefficients: contain information of the lower frequencies of the voltammogram, associated to the signal.
- CD or detail coefficients: contain information of the higher frequencies of the voltammogram, associated to the noise

The original signal can be reconstructed using both sets of coefficients or only one of them as it is illustrated in Figure 14 with one example.

Figure 14: Graphics of an example signal comparing the original signal with the reconstructed signal using both coefficients, only CA and only CD.



It can be seen with naked eye that the reconstructed signal with CA is similar to the original one whereas the reconstructed signal with CD reassembles noise. The reconstructed signal has the same values as the original but the CA is half the values and could be fed to the ANN. Therefore DWT is a method that eliminates noise and compresses the signal.



This function can be concatenated as illustrated in Figure 15. First, the wavelet transform is applied to the original signal and then applied to the obtained CA coefficients. This is done as many times as necessary, called decomposition level. This is known as Mallat's pyramidal algorithm and it is used to furthermore eliminate noise and/or to compress the signal. However, it has to be considered that with every level of DWT information is also lost.

Figure 15: Mallat's pyramidal algorithm used to implement the DWT. Approximation and detail vectors are indicated by cA and cD respectively; subindices denote decomposition level. (15)

In order to perform DWT with more than one level of decomposition *wavedec* function (32) is used (Wavelet Toolbox™). This function inputs are: the original signal, the mother wavelet, and the level of decomposition. Part of the effort to obtain a acceptable compression consists on determining the mother wavelet function and the maximum decomposition level that best represents the original signal without significant loss of information (29). As not all the wavelet available in the MATLAB toolbox can be tested only some of them will be studied, following the conclusion of other researchers¹:

- In (15) the mother wavelets evaluated were Coiflets, Daubechies, biorthogonal and symlets. Voltammograms were compressed using 165 combinations obtained with 33 different mother wavelets and five different decomposition levels. They finally use a fourth order Daubechies' wavelet and a decomposition level 4, which represents compromise between number of approximation coefficients obtained at each decomposition level and the degree of similarity between the original voltammogram and the one recovered with these coefficients.
- In (29) several mother wavelets (Daubechies, Coiflets, symlets and biorthogonal) and four successive decomposition levels were tested. The combination that fulfilled our purpose was obtained with the Daubechies wavelet of eighth order and decomposition level 3.
- In (33) different kinds of wavelet basis were tested: Haar, Daubechies 'n' (n = 3, 4, 6, 8, 10, 12, 14, 16, 18 and 20), Symlet 'm' (m = 2–8) and Coiflets 'g' (g = 1–5). Five decomposition levels were applied. Symlet 3 and Symlet 4 wavelets with decomposition level 4 were selected (typical wavelets to represent symmetric signals).

In order to selected the best mother wavelet and level of decomposition among the tested wavelets the same criteria as (15) is used: degree of similarity between the original voltammogram and the one reconstructed from approximation coefficients after compression. The authors propose a comparison factor named f_c which considers the area under both signals when superimposed. The f_c can range from 0 to 1, depending on similarity; 1 indicating identical signals. Considering discrete signals of length M, the f_c factor can be calculated as:

$$f_c = \frac{A \cap B}{A \cup B} = \frac{\sum_{i=1}^M \min(a_i, b_i)}{\sum_{i=1}^M \max(a_i, b_i)}$$

where a_i is the i th data point of the original voltammogram, b_i the i th data point of the reconstructed signal from approximation coefficients.

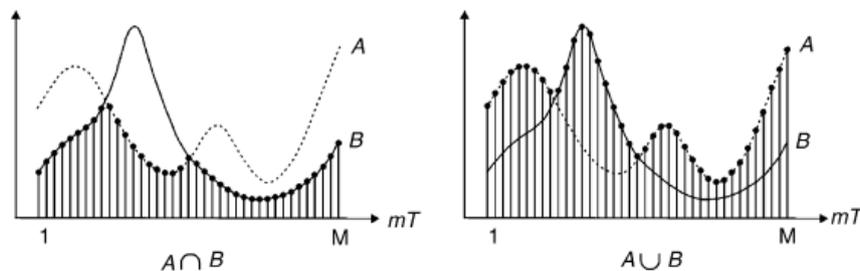


Figure 16: Scheme of surface elements involved in the calculation of f_c (15)

To reconstruct the signal the function *waverec* (34) is used. It has been considered that the comparison factor should be $f_c > 0,95$.

¹ The differences between these types of wavelets are well explained in (28).

Table 1: Compression parameters tested

Wavelet family	Order	Decomposition level	MATLAB wavelet names
Daubechies	1, 2, 3, 4, 6, 8, 10, 12, 15, 20, 25, 30, 35, 40, 45	1 to 6 for all	'db1','db2',... , 'db12', 'db15', 'db20', 'db25', 'db30', 'db35', 'db40', 'db45'
Coiflets	1, 2, 3, 4, 5	1 to 6 for all	'coif1', ... , 'coif5'
Symlets	2, 3, 4, 6, 8, 10, 12, 15, 20, 25, 30, 35, 40, 45	1 to 6 for all	'sym2', 'sym3',... , 'sym12', 'sym15', 'sym20', 'sym25'
Biorthogonal	1.1, 1.3, 2.2, 2.6, 3.1, 3.7, 4.4, 5.5, 6.8	1 to 6 for all	'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.7', 'bior4.4', 'bior5.5', 'bior6.8'
Reverse Biorthogonal	1.1, 1.3, 2.2, 2.6, 3.1, 3.7, 4.4, 5.5, 6.8	1 to 6 for all	'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.7', 'rbio4.4', 'rbio5.5', 'rbio6.8'

The MATLAB implementation can be found in the Annex II.b and a schematic and brief description of the different functions is shown in Figure 17. For each sensor, the voltammogram of all samples are feed to a function that prepares the data for compression: first the baseline is remove with the best suitable set-up found, only the measurements between samples 81:500 are kept as to avoid undesired interferences for the ANN, and finally the signal is normalized. The output signal then undertakes a testing process where the signal is compressed using different parameters: mother wavelet and decomposition level. For each set-up the results with $f_c > 0,95$ are presented to the user whom can chose the best suitable one; balance compression level and signal degradation.

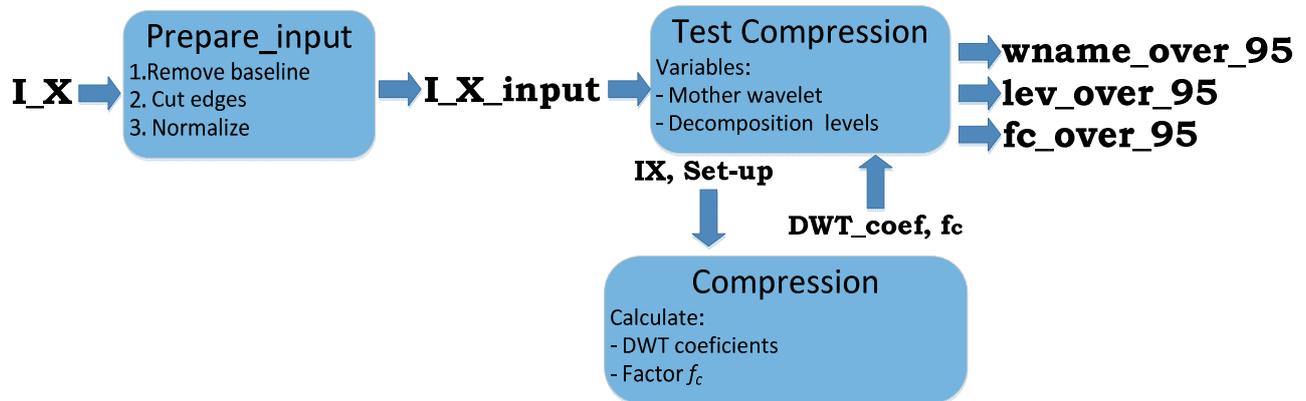


Figure 17: MATLAB functions developed for DWT compression

ANN

The Neural Network Toolbox of MATLAB provides the tools to build ANN and the possibility of setting its parameters. For the case here studied different ANN have been tested and its performance has been compared using the neural network fitting tool GUI or by coding the network using the MATLAB toolbox commands. The neural network fitting tool GUI provides limited options such as: only one input neuron and one hidden layer allowed, no option to choose transfer function in hidden layers, and test one set-up at a time among other limitations. However the first ANN were implemented using the GUI. The input were the 80 samples and 380 DWT coefficients (95 from each sensor) and for all the simulations 64 samples were used for training 8 for testing and 8 for validation. Different number of neurons for the hidden layer were chosen: 1-10 and the training algorithms were Levenberg-Marquardt or Bayesian Regularization.

Therefore an implementation was developed using commands and MATLAB functions (which can be found in Annex III.b). The ANN implemented using MATLAB commands provides more options that are described below. The number of inputs in the ANN is fixed to the number of sensors and the data provided. In this case 4 sensors were used and the data were compressed to DWT coefficients. Other authors (7,14,15,35), instead of

using the each sensor as an input, use each DWT coefficient as an input as depicted in Figure 18. This set-up has not been tested, nonetheless it should be considered for further testing.

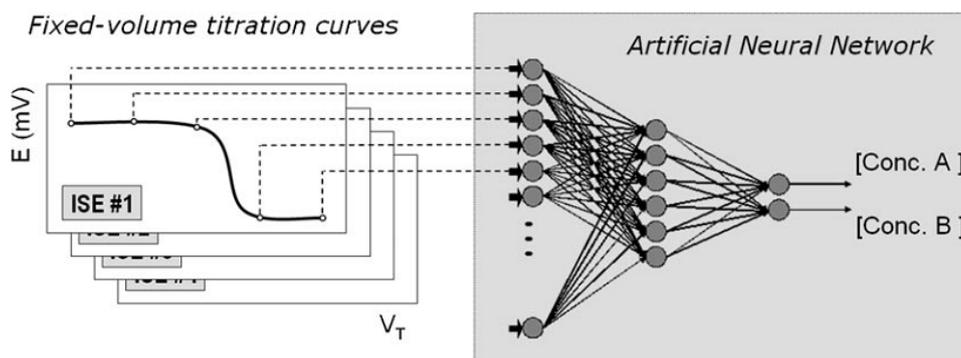


Figure 18: Conceptual scheme of the titration procedure with the use of the electronic tongue (35)

The outputs can be one or more as an ANN can be modelled for each substance or an ANN can be used for all substances at once. In this work only networks with one output have been developed as they have shown to give better results (15).

There is no guide for ANN topology as to infer which number of neurons and hidden layers will yield better ANN models. According to (7), almost all recorded chemistry situations employ a single layer. However as it has been described previously various authors use multi-layer ANN so both scenarios are also studied. In order to deduce the best number of neurons the recommendation is trial and error, starting with a large number and decreasing until the model deteriorates due to manifestation of a too simple network. The ANN used consists of a feed-forward network where different number of layers/sensors, transfer functions and learning algorithms have been tested. The number of hidden layers vary from 1 to 3 and different number of neurons for each layer have been tested. The list below provides more details about the other parameters tested:

- To build the network two different functions have been used to compare their results: *feedforwardnet* (36) and *fitnet* (37).
- Two backpropagation training functions have been tested:
 - Levenberg-Marquardt backpropagation: is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization. It is often the fastest backpropagation algorithm; however it does require more memory than other algorithms (38).
 - Bayesian regularization backpropagation: is a network training function that updates the weight and bias values according to Levenberg-Marquardt optimization. It minimizes a combination of squared errors and weights, and then determines the correct combination that reduces the performance error (39).
- Different transfer functions have been tested: *logsig*, *satlins*, *tansig* and *purelin*. These transfer functions have been combined in the different hidden layers and also in the output layer.

Another factor to consider is the kind of performance error used during the training process, SSE was used as the authors in (28,29):

$$SSE = \sum_{j=1}^N (c_{expected} - c_{calculated})^2$$

To compare the different ANN simulated two parameters have been compared: lack of fit (lof) and percentile relative absolute error (PRAE). These allow to evaluate the different networks' performance:

$$PRAE = \frac{1}{M} \sum_{i=1}^M \frac{|c_{expected_i} - c_{calculated_i}|}{c_{expected_i}} \times 100$$

$$lof = \sqrt{\frac{\sum_{i=1}^M (c_{expected_i} - c_{calculated_i})^2}{\sum_{i=1}^M c_{expected_i}^2}} \times 100$$

where c is the concentration value obtained experimentally and calculated with ANN and M the number of testing samples.

Figure 19 summarizes the different configurations for the ANN implemented. For each simulation a type of neural network function to create the network was chosen: *fitnet* or *feedforwardnet* and also the learning algorithm was selected: Levenberg-Marquardt or Bayesian Regularization. Then the number of hidden layers was chosen and ANN configuration .m file was implemented. In order to keep track of the different set-ups simulated these type of files where name as follows: 'ANN_create_XX_YY_ABCD.m' where XX corresponds to the neural network function used (fit: *fitnet* or feed: *feedforwardnet*); YY indicates the learning algorithm (LM or Bay) and ABCD are the transfer function used for each layer (L: logsig, S: satlins, T: tansig and P: purelin). For example: ANN_create_fit_LM_LS_P: indicates that the file creates a ANN using fitnet with training algorithm Levenberg-Marquardt with two hidden layers: first layer uses logsig transfer function and the second layer uses satlins transfer function and the output layer uses a purelin transfer function. This configuration files are then called from another .m file which trains and tests the ANN as shown in Figure 20. These type of files are called 'ANN_test_XX_YY_ZZ.m' where XX corresponds to the neural network function used (fit: *fitnet* or feed: *feedforwardnet*); YY indicates the learning algorithm (LM or Bay) and Z indicates de number of hidden layers 1-3. For networks with 1 or 2 hidden layers the transfer function can be the same for all the layers (including output) or different, generating more possibilities. Hence, each test file will call more than one ANN_create.m file, i.e. 'ANN_test_fit_BAY_2.m' will call all create ANN .m files corresponding to two hidden layers: 'ANN_create_fit_BAY_AB_D.m'. On the other hand, for 3 hidden layers the same transfer function has been used for all layers, i.e. all neurons in the ANN use the same type of function. Over 1000 different ANN configurations were tested for each concentration.

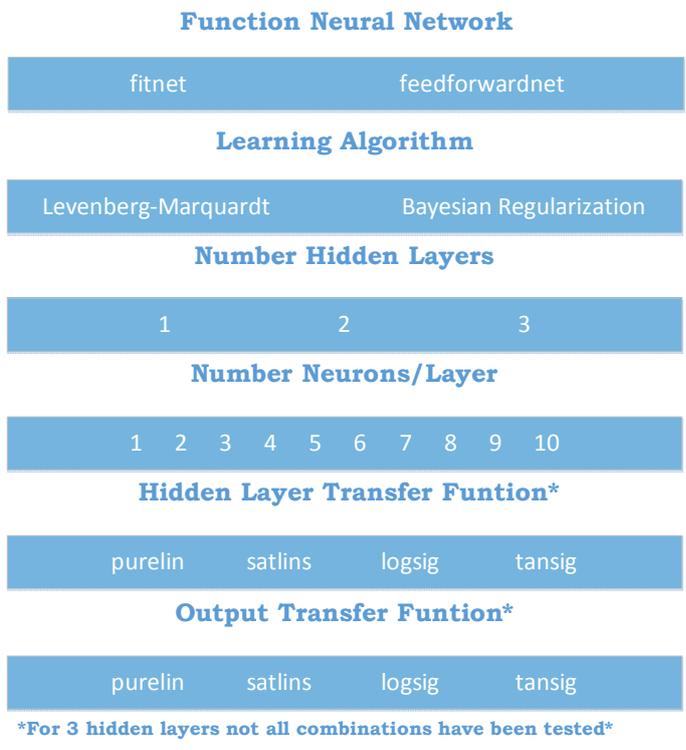
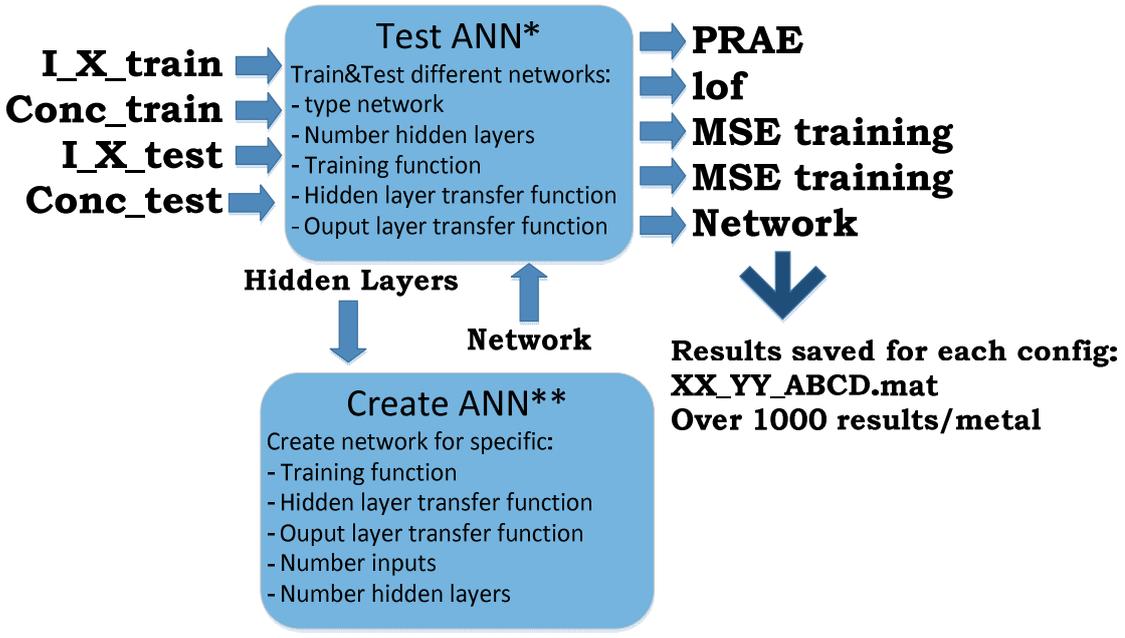


Figure 19: ANN parameters used

The MATLAB implementation can be found in the Annex III.b and a schematic and brief description of the different functions is shown in Figure 20.



***ANN_test_XX_YY_ZZ.m: 29 configuration files (tests ≠ #neurons: 1-5/10)**
****ANN_create_XX_YY_ABCD.m: 163 configuration files**

Figure 20: MATLAB functions developed for ANN

Results

Baseline

The removal has been done separately for each type of sensor, therefore 4 estimations have been obtained. In all cases the best estimations were found using asymmetric truncated quadratic and the final results are summarized in Table 2.

Table 2: Best baseline removal parameters

Sensor	Function	Polynomial order	Threshold
SPCE-CNF	Asymmetric truncated quadratic	4	0,030
ex-situ-SbSPCE-CNF	Asymmetric truncated quadratic	4	0,100
Cys-SPCE-CNF	Asymmetric truncated quadratic	4	0.050
GSH-SPCE-CNF	Asymmetric truncated quadratic	4	0.040

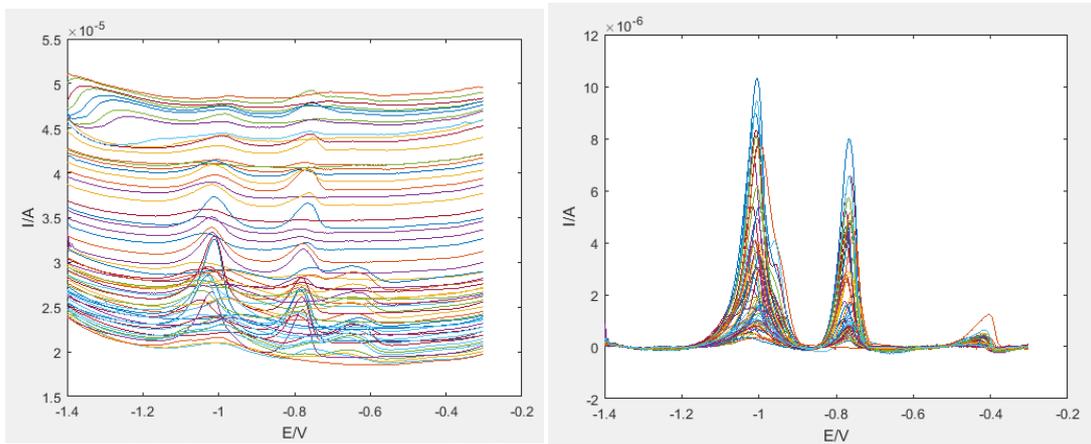


Figure 21: Sensor Cys-SPCE-CNF for all samples a) Raw data, b) Data after removing baseline.

Additionally, to eliminate electrodes anomalies and to reduce the effects of the baseline and only the measurements 81 to 500 are used.

Compression

The best f_c factor is obtained with level 1 compression as it reduces de data input from 420 measurements to 234 DWT coefficients. However, higher compression levels are used to further reduce the number of DWT input coefficients. As stated before only compressions that provide a factor above 0,95 are considered. From all the different set-ups tested over a 100 fulfil this condition and some of the results can be found in Annex III.c. Some of them have a similar performance (similar f_c value), but the compression chosen was Daubechies order 25 with compression level 3 reducing the data into 95 DWT coefficients and obtaining a $f_c > 0,995$ for all 4 electronic tongues.

Sensor	f_c
SPCE-CNF	0,9966
ex-situ-SbSPCE-CNF	0,9989
Cys-SPCE-CNF	0,9976
GSH-SPCE-CNF	0,9962

Table 3: Compression results for Daubechies order25 with compression level 3

ANN

After testing the ANN using neural network fitting tool GUI the developed .m files explained in the previous section were evaluated. Some of the ANN implemented have good performance with the training data, however this performance is drastically reduced with test and validation data. For each heavy metal, an ANN has been implemented; and therefore different configurations can be used for each one. However, it has been hard to determine which were the best configuration as none of them performed as good as expected and do not provide enough accurate results.

The first ANN implemented was done using the neural network fitting tool GUI. As it provides limited options and it has to be done all through the GUI (time consuming) only a few ANN have been tested to analyze one of the heavy metals.

After these preliminary tests the MATLAB software explained in previous chapter was tested. The results are not as stable as foreseen as for the same configuration re-training can give very different results. Figure 22, shows the results of the same network after different train tests, the best and worst results are plotted.

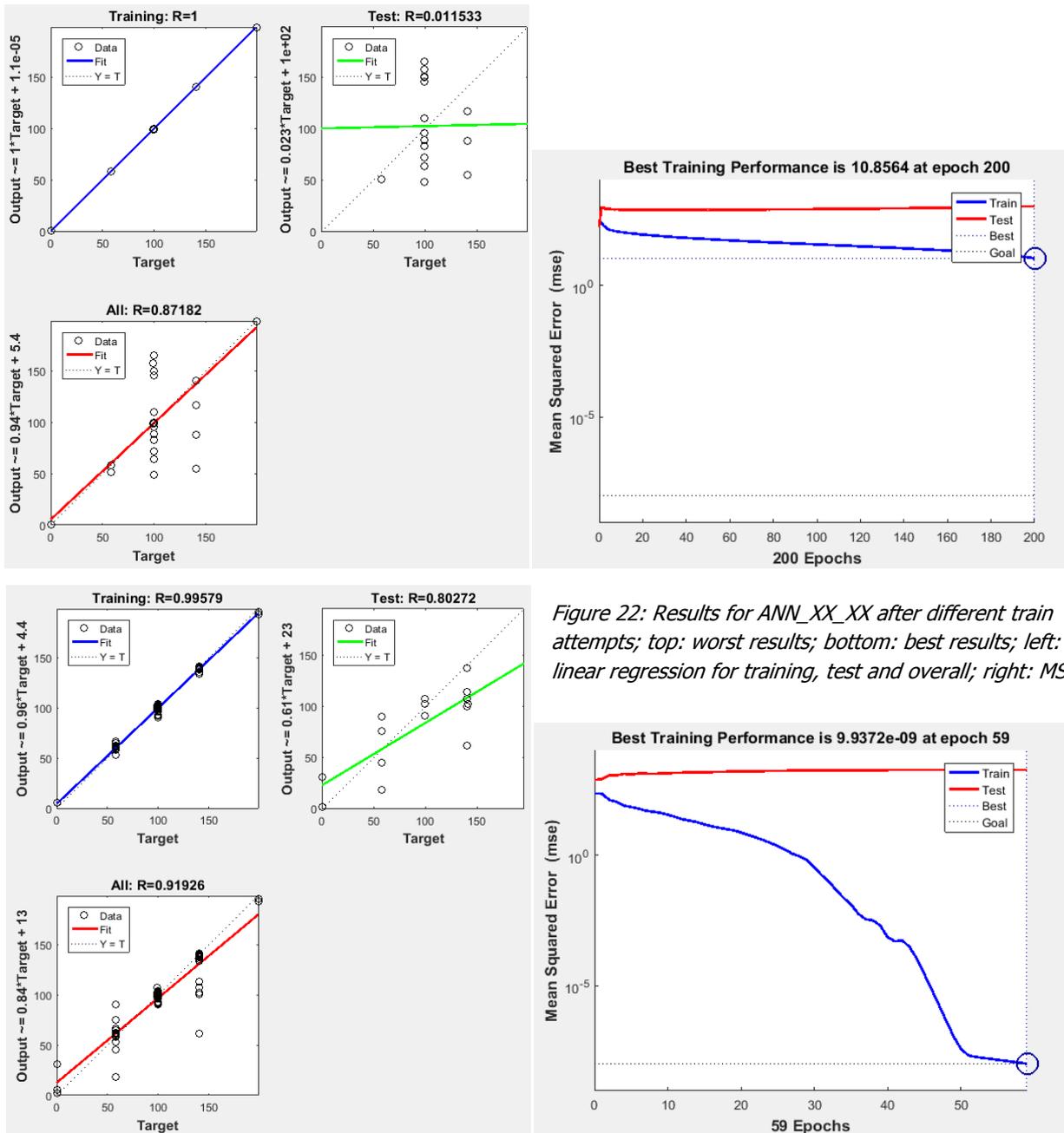
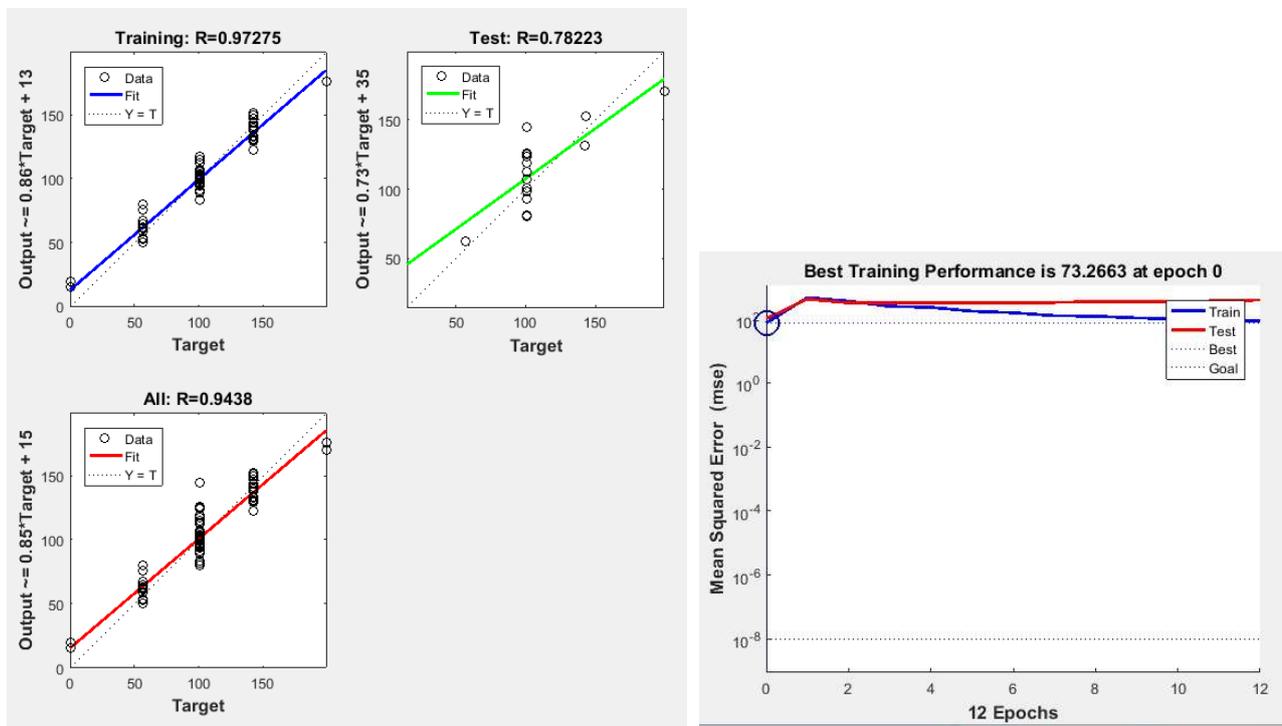
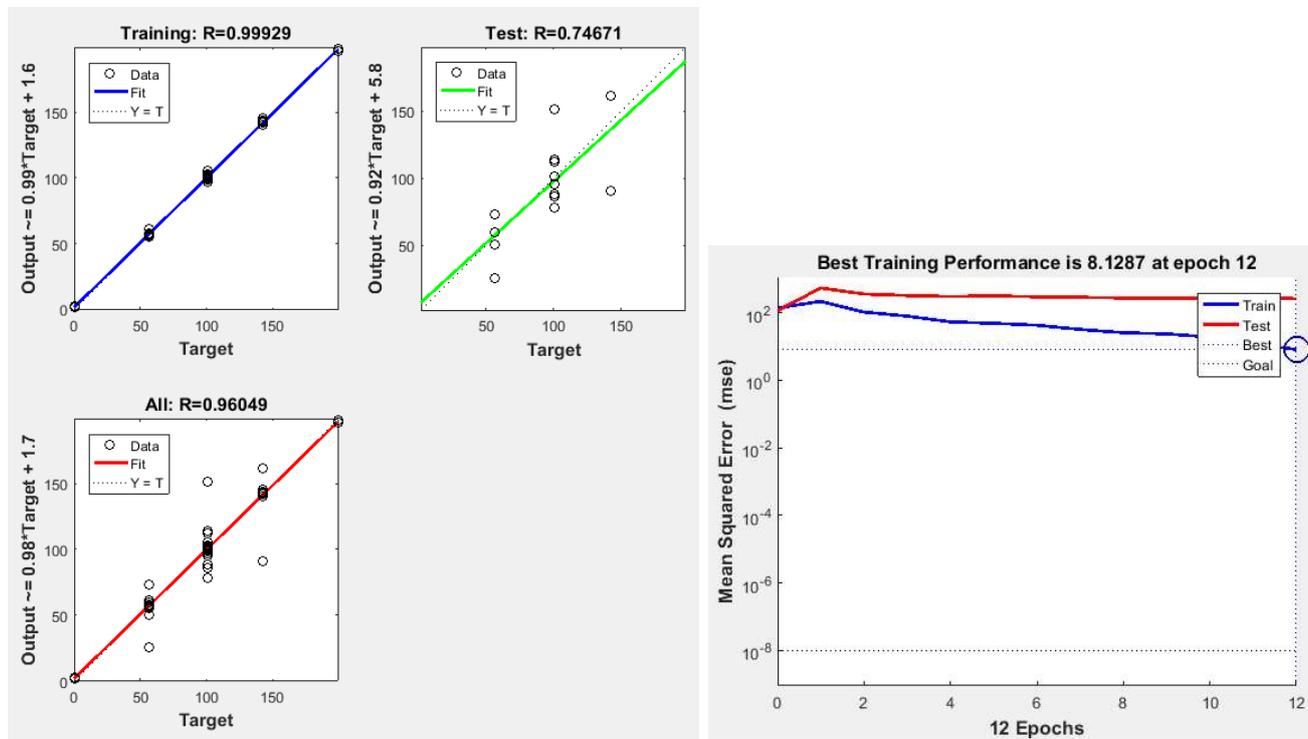


Figure 22: Results for ANN_XX_XX after different train attempts; top: worst results; bottom: best results; left: linear regression for training, test and overall; right: MSE

After analyzing the output of over a 1000 the simulations the best results seem to be obtained with 2 hidden layers with 5-1 neurons/layer; but always more than 5 neurons in the whole network. Due to the variance of ANN explained earlier; i.e. different results after every training, the best results seem to be obtained with LT_S ANN with 5-4 neurons in each hidden layer. Nevertheless, further testing should be done: train the ANN and analyze if any configuration has better results. Figure 23 plots the best results obtained with one hidden layer and with three hidden layers and in Figure 24 the best results with LT_S ANN are presented.



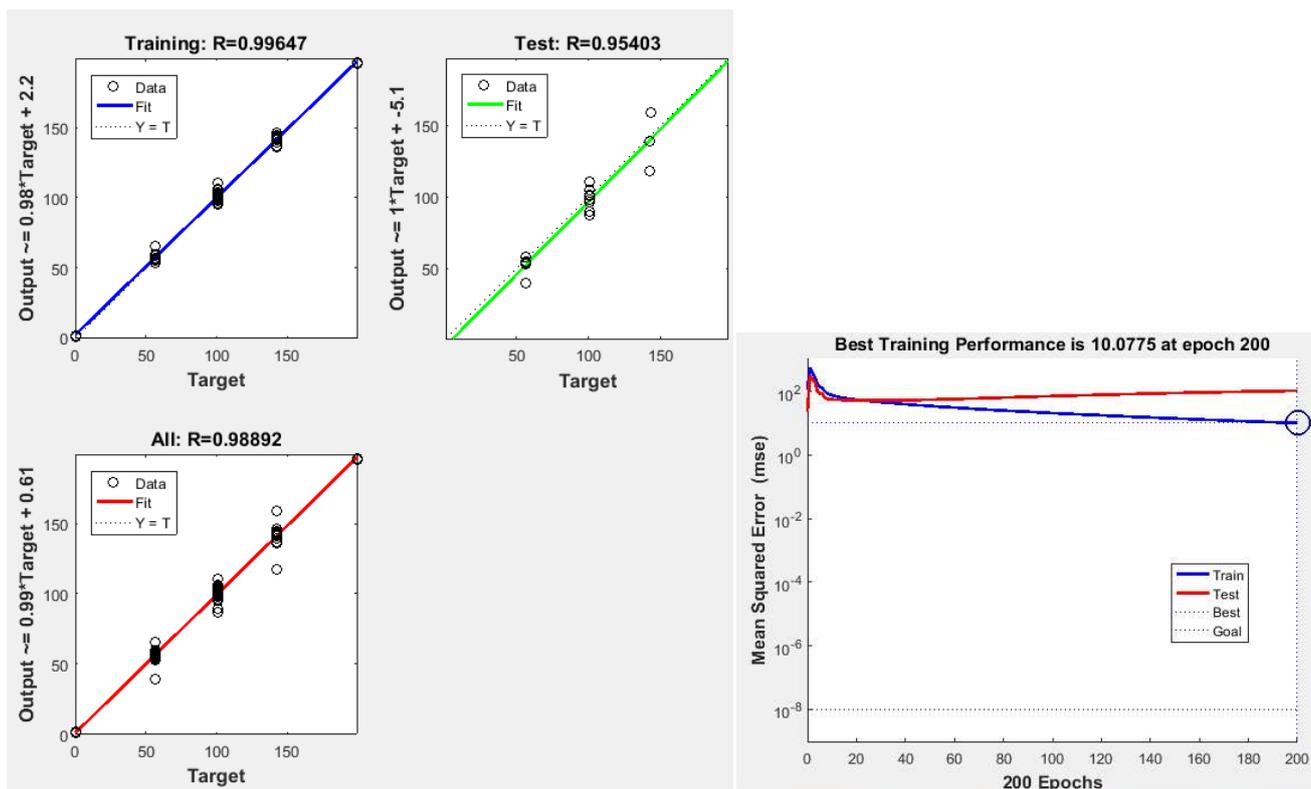
a) Performance results for ANN_feed_BAY_L_P with 5 neurons



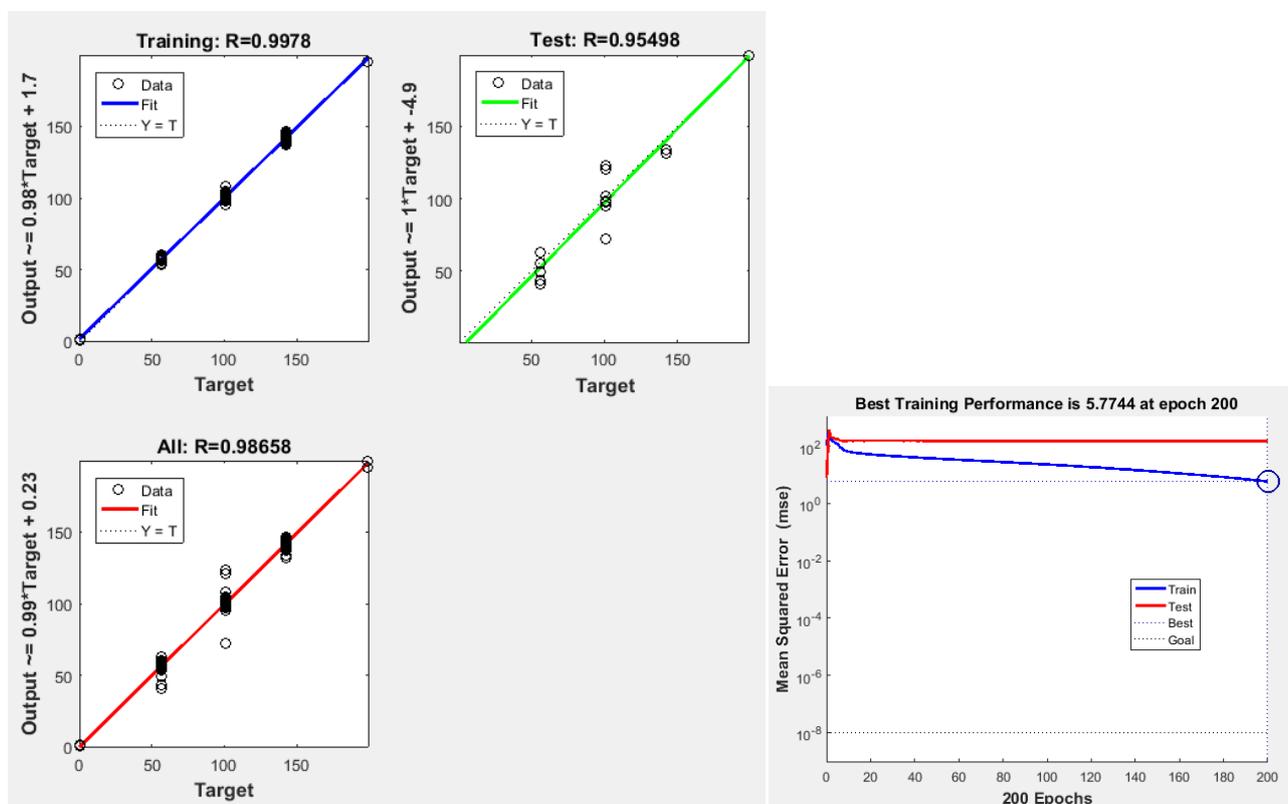
b) Performance results for ANN_feed_BAY_LLLP with 3,5 and 3 neurons per layer

Figure 23: Best performance results for one and three hidden layers; left: linear regression for training, test and overall; right: MSE for training and testing

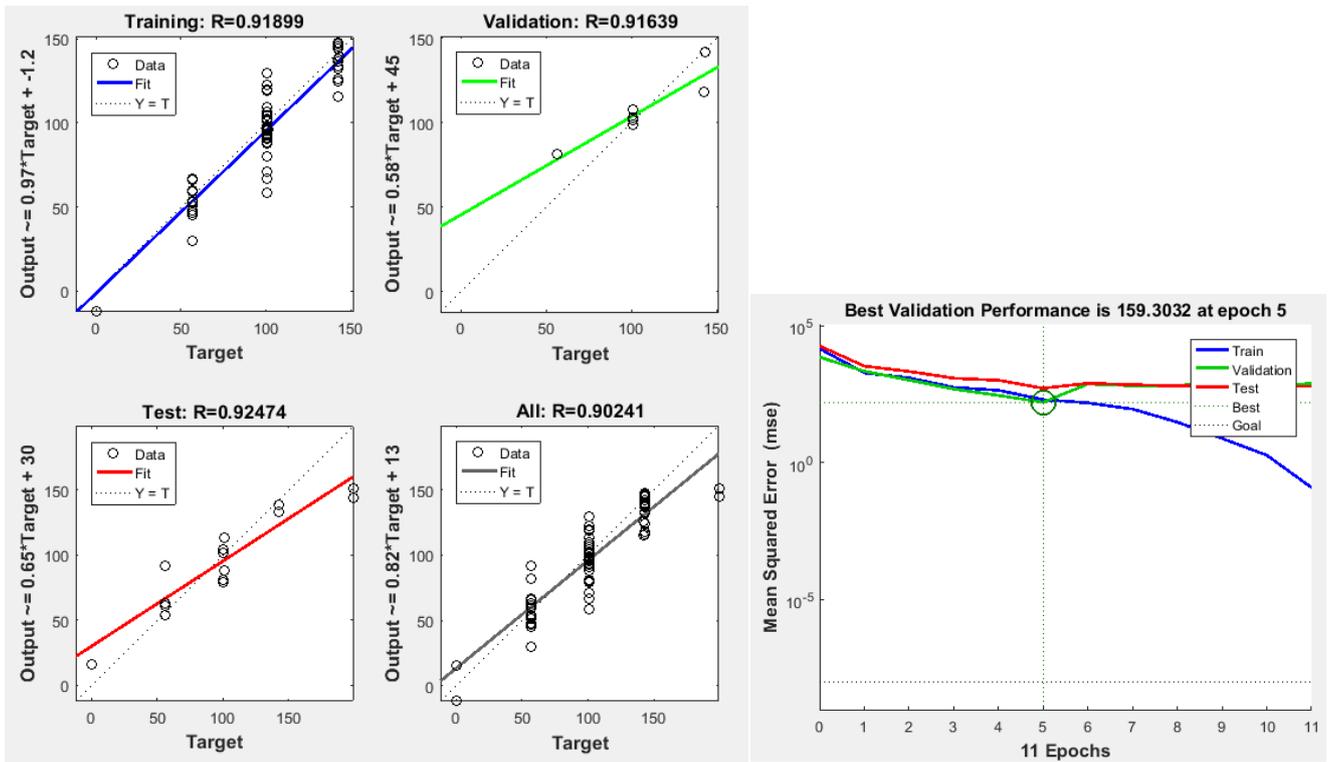
In order to study which MATLAB function (*fitnet* or *feedforwardnet*) and learning algorithm (LM or BAY) are better four different configuration have been tested. The results comparing them are presented in Figure 24. These results are the best obtained among 6 train attempts.



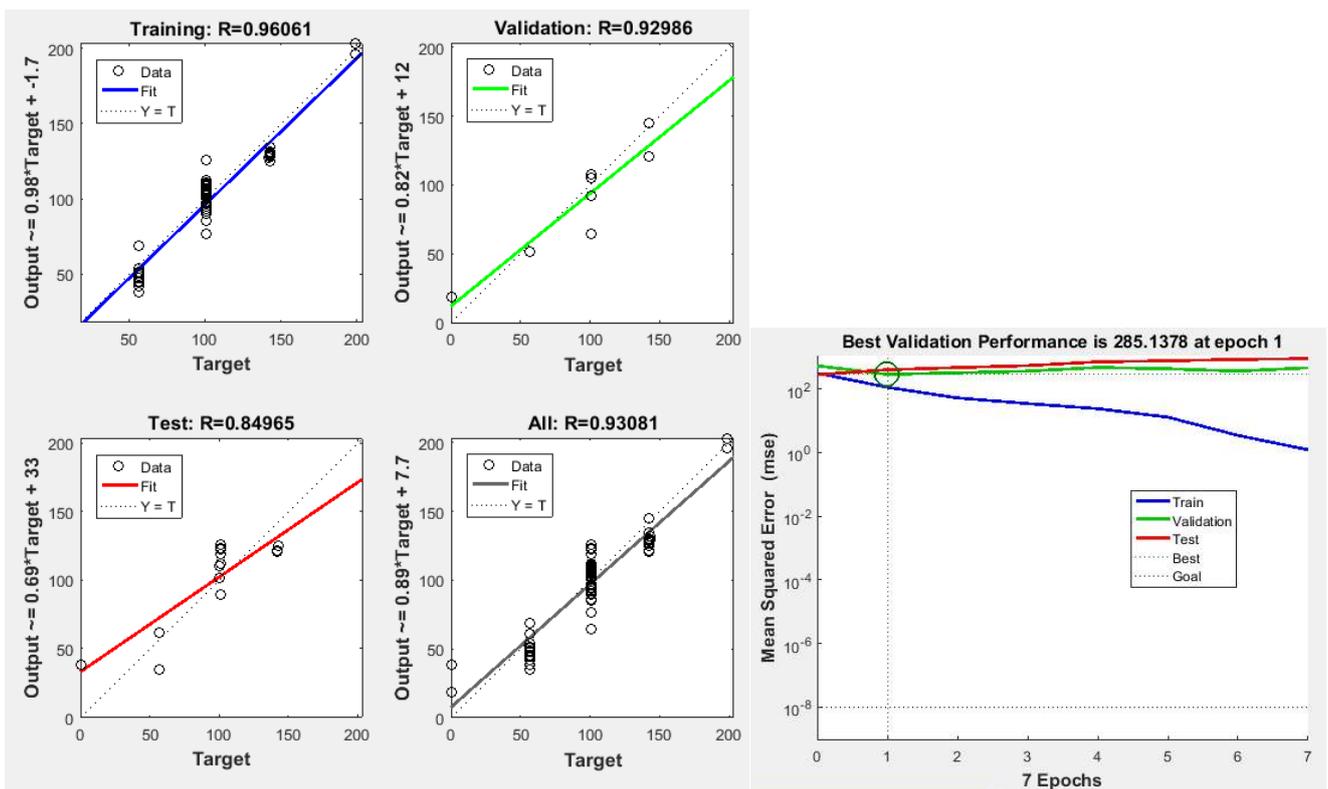
a) Performance results for ANN_feed_BAY_LT_P



b) Performance results for ANN_fit_BAY_LT_P



c) Performance results for ANN_feed_LM_LT_P



d) Performance results for ANN_fit_LM_LT_P

Figure 24: Best results for ANN_LT_P with 5 neurons in each hidden layers after 6 train attempts; left: linear regression for training, test and overall; right: MSE for training and testing

From the performance results depicted in Figure 24 it can be seen that the best results are obtained with ANN_fit_BAY_LT_P: better linear regression (higher values for R) and less MSE for training and test (lower MSE).

To conclude, Figure 25 shows the concentration one heavy metal obtained experimentally compared with the concentration estimated with the ANN. As it can be seen these estimations are not good estimations as they

diverse from the real concentration. The best results obtained seemed promising during training (summarized in Table 4, however the results with testing data were not accurate as it can be seen in Figure 25.

Table 4: Performance result for ANN_feed_BAY_LT_P

MSE training data	$9,4 \cdot 10^{-8}$
MSE test data (during training)	180
R linear regression training	0,9987
R linear regression testing	0,9541
R test data (After training)	-0,5907

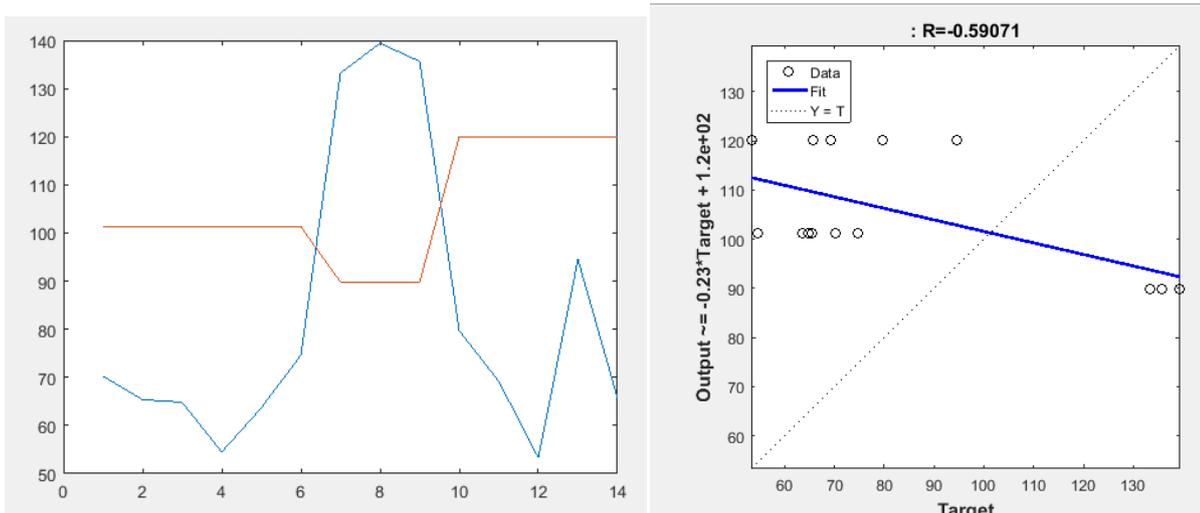


Figure 25: Comparison of the test data concentration with the results obtained from ANN_feed_BAY_LT_P; left: Concentrations for each sample: experimental vs ANN results; b) linear regression

Conclusion

This work evaluates the possibility of processing with ANN the data obtained by a voltammetric electronic tongue for the simultaneous quantitative determination of six heavy metals: Zn(II), Cd(II), Pb(II), Tl(I), In(III) and Bi(III) ions inside the concentration range 0 to 200 $\mu\text{g/L}$. In order to do this the raw data has been pre-processed before using ANN. The pre-process consisted in baseline removal and DWT compression. Different baseline removing configurations and different mother wavelets for compression have been tested. The results obtained in this first part of the project are favourable as the baseline has been successfully removed and the data have been compressed from 420 samples to 95 for each sensor. The pre-processing of the voltammograms by DWT has permitted the reduction of the amount of information needed to represent its content in a factor of almost 5, which means a huge reduction considering the high difficulty of the case studied: overlapped combination of six compounds plus noise and the oxidation of containing media.

The presented work used the pre-processed data as input for ANN. Different configurations for ANN have been tested; i.e. different number of hidden layers and neurons, transfer function and training algorithm have been tested. The results do not perform as expected and heavy metal concentrations for the test data could not be approximated accurately. However it can be seen from the results obtained that more number of hidden layers or neurons/layer do not imply a better performance. Better results are obtained with 2 hidden layers and between 5--2 hidden neurons, but less more than 4 neurons in total. From the results obtained it seems that the best configuration is ANN_feed_BAY_LT_P; however, further testing with the prepared configurations should be performed.

Even though during the training of the ANN good R values are obtained for training and testing sets. However when the network is feed with other testing data the results are not accurate (Figure 24).

A possible reason the results are not as favourable as in other articles such as (15,29,35), is that the concentration used for the six heavy metals are not incremental; i.e. they are not equally distributed in the studied range 0 to 200 $\mu\text{g/L}$ as it can be seen in Figure 24. This fact can influence the ANN training process and therefore the results are not as reliable as for other authors. It should be considered to use other input data where the concentrations where more distributed over a range and avoid repeating concentration values. This could also be done by . In Figure 25 the relation of concentrating among three of the heavy metals is represented.

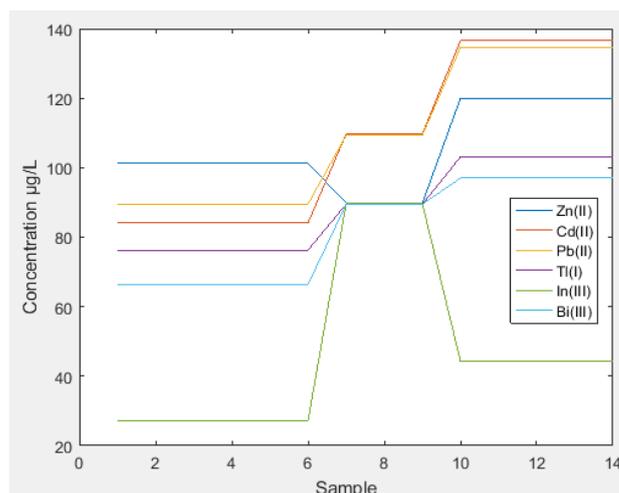


Figure 25: Heavy metal concentrations corresponding to the electronic tongues data used.

Further work

As just stated the different ANNs developed could be tested with the same data but providing a change in axis. Another test that could be done is change the pre-processing parameters; i.e. use more or less DWT; and test the ANN again and check if the performance improves or deteriorate. The program developed could be also used with other input data. However, the network parameters should be adjusted to the new data; i.e. number of sensors and input DWT and number of substances to be tested. As it can be seen in the annexes the program files are commented to help further researchers reuse this program in their studies/projects. Additionally, a GUI for software developed could be implemented. From this GUI the user could chose among more options than the ones provided by GUI ; however it would also have some limitations. This GUI could also include the pre-processing calculations; where the user could test which baseline removal and/or compression configuration is best suited for a specific set-up before using the ANN.

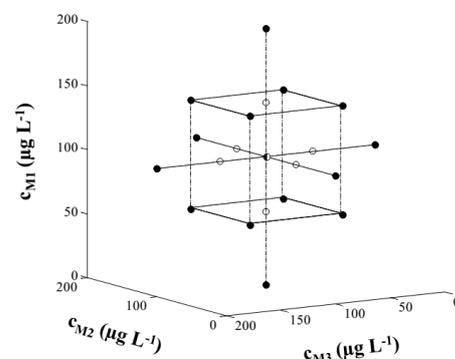


Figure 26: Experimental design for 3 of the 6 heavy metals studied. Small circles indicate the concentration of the heavy

References

1. Foster IDL, Charlesworth SM. Heavy metals in the hydrological cycle: trends and explanation. *Hydrol Process*. Wiley Online Library; 1996;10(2):227–61.
2. Jorgensen SE, Fath B. *Encyclopedia of ecology*. Newnes; 2014.
3. Baby J, Raj JS, Biby ET, Sankarganesh P, Jeevitha M V, Ajisha SU, et al. Toxic effect of heavy metals on aquatic environment. *Int J Biol Chem Sci*. Ivyspring International Publisher; 2010;4(4).
4. Castro-González MI, Méndez-Armenta M. Heavy metals: Implications associated to fish consumption. *Environ Toxicol Pharmacol*. Elsevier; 2008;26(3):263–71.
5. Pérez-Ràfols C, Serrano N, Díaz-Cruz JM, Ariño C, Esteban M. A screen-printed voltammetric electronic tongue for the analysis of complex mixtures of metal ions. Submitted 2017.
6. Vlasov Y, Legin A, Rudnitskaya A, Di Natale C, D'Amico A. Nonspecific sensor arrays ('electronic tongue') for chemical analysis of liquids (IUPAC Technical Report). *Pure Appl Chem*. 2005;77(11):1965–83.
7. del Valle M. Sensor arrays and electronic tongue systems. *Int J Electrochem*. Hindawi Publishing Corporation; 2012;2012; 11 pages.
8. Cetó X, Céspedes F, del Valle M. Comparison of methods for the processing of voltammetric electronic tongues data. *Microchim Acta*. Springer; 2013;180(5–6):319–30.
9. Mazet V, Carteret C, Brie D, Idier J, Humbert B. Background removal from spectra by designing and minimising a non-quadratic cost function. *Chemom Intell Lab Syst*. Elsevier; 2005;76(2):121–33.
10. Simons J, Bos M, van der Linden WE. Data processing for amperometric signals. *Analyst*. Royal Society of Chemistry; 1995;120(4):1009–12.
11. Despagne F, Luc Massart D. Neural networks in multivariate calibration. *Analyst* [Internet]. The Royal Society of Chemistry; 1998;123(11):157R–178R. Available from: <http://dx.doi.org/10.1039/A805562I>
12. de Carvalho RM, Mello C, Kubota LT. Simultaneous determination of phenol isomers in binary mixtures by differential pulse voltammetry using carbon fibre electrode and neural network with pruning as a multivariate calibration tool. *Anal Chim Acta*. Elsevier; 2000;420(1):109–21.
13. Gemperline PJ. Rugged spectroscopic calibration for process control. *Chemom Intell Lab Syst*. Elsevier; 1997;39(1):29–40.
14. Serrano N, Prieto-Simón B, Cetó X, del Valle M. Array of peptide-modified electrodes for the simultaneous determination of Pb (II), Cd (II) and Zn (II). *Talanta*. Elsevier; 2014;125:159–66.
15. Moreno-Barón L, Cartas R, Merkoçi A, Alegret S, Del Valle M, Leija L, et al. Application of the wavelet transform coupled with artificial neural networks for quantification purposes in a voltammetric electronic tongue. *Sensors Actuators B Chem*. Elsevier; 2006;113(1):487–99.
16. Leung AK, Chau F, Gao J. A review on applications of wavelet transform techniques in chemical analysis: 1989–1997. *Chemom Intell Lab Syst*. Elsevier; 1998;43(1):165–84.
17. Moreno-Barón L, Cartas R, Merkoçi A, Alegret S, Gutiérrez JM, Leija L, et al. Data Compression for a Voltammetric Electronic Tongue Modelled with Artificial Neural Networks. *Anal Lett* [Internet]. Taylor & Francis; 2005 Oct 1;38(13):2189–206. Available from: <http://dx.doi.org/10.1080/00032710500259342>
18. Cetó X, Céspedes F, Pividori MI, Gutiérrez JM, Del Valle M. Resolution of phenolic antioxidant mixtures employing a voltammetric bio-electronic tongue. *Analyst*. Royal Society of Chemistry; 2012;137(2):349–56.
19. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*. Springer; 1943;5(4):115–33.

20. Pitts W, McCulloch WS. How we know universals the perception of auditory and visual forms. *Bull Math Biophys.* Springer; 1947;9(3):127–47.
21. Veitch D. Wavelet Neural Networks and their application in the study of dynamical systems. Dep Math Univ York UK. 2005;
22. Maltarollo VG, da Silva ABF, Honório KM. Applications of artificial neural networks in chemical problems. INTECH Open Access Publisher; 2013.
23. Suzuki K. Artificial neural networks: methodological advances and biomedical applications. InTech; 2011.
24. Krenker A, Kos A, Bešter J. Introduction to the artificial neural networks. INTECH Open Access Publisher; 2011.
25. Gutiérrez JM, Gutés A, Céspedes F, del Valle M, Muñoz R. Wavelet neural networks to resolve the overlapping signal in the voltammetric determination of phenolic compounds. *Talanta.* Elsevier; 2008;76(2):373–81.
26. Kim H, Soibelman L, Grobler F. Factor selection for delay analysis using knowledge discovery in databases. *Autom Constr.* Elsevier; 2008;17(5):550–60.
27. Bevrani H, Syafaruddin, Hiyama T. Artificial Neural Network (ANN) [Internet]. Electric Power System Laboratory. 2010 [cited 2017 Feb 4]. Available from: <http://www.cs.kumamoto-u.ac.jp/epslab/ICinPS/Lecture-2.pdf>
28. Beale MH, Hagan MT, Demuth HB. *Neural Network Toolbox™ User's Guide.* MathWorks Inc. Citeseer; 1992;
29. Gutés A, Céspedes F, Cartas R, Alegret S, del Valle M, Gutierrez JM, et al. Multivariate calibration model from overlapping voltammetric signals employing wavelet neural networks. *Chemom Intell Lab Syst.* Elsevier; 2006;83(2):169–79.
30. Faura G, González-Calabuig A, del Valle M. Analysis of Amino Acid Mixtures by Voltammetric Electronic Tongues and Artificial Neural Networks. *Electroanalysis.* Wiley Online Library; 2016;28(8):1894–900.
31. dwt [Internet]. MathWorks Documentation. [cited 2017 Feb 4]. Available from: <https://es.mathworks.com/help/wavelet/ref/dwt.html>
32. wavedec [Internet]. MathWorks Documentation. [cited 2017 Feb 5]. Available from: https://es.mathworks.com/help/wavelet/ref/wavedec.html?searchHighlight=wavedec&s_tid=doc_srchtile
33. Palacios-Santander JM, Jimenez-Jimenez A, Cubillana-Aguilera LM, Naranjo-Rodriguez I, Hidalgo-Hidalgo-de-Cisneros JL. Use of artificial neural networks, aided by methods to reduce dimensions, to resolve overlapped electrochemical signals. A comparative study including other statistical methods. *Microchim Acta.* Springer; 2003;142(1–2):27–36.
34. waverec [Internet]. MathWorks Documentation. [cited 2107 Feb 5]. Available from: https://es.mathworks.com/help/wavelet/ref/waverec.html?searchHighlight=waverec&s_tid=doc_srchtile
35. del Valle M. Electronic tongues employing electrochemical sensors. *Electroanalysis.* Wiley Online Library; 2010;22(14):1539–55.
36. feedforwardnet [Internet]. MathWorks Documentation. [cited 2017 Feb 12]. Available from: <https://es.mathworks.com/help/nnet/ref/feedforwardnet.html>
37. fitnet [Internet]. MathWorks Documentation. [cited 2017 Feb 12]. Available from: <https://es.mathworks.com/help/nnet/ref/fitnet.html>
38. trainlm - Levenberg-Marquardt backpropagation [Internet]. MathWorks Documentation. [cited 2017 Feb 12]. Available from: <https://es.mathworks.com/help/nnet/ref/trainlm.html>
39. trainbr - Bayesian regularization backpropagation [Internet]. MathWorks Documentation. [cited 2017 Feb 12]. Available from: <https://es.mathworks.com/help/nnet/ref/trainbr.html>

Annex

I. Remove Baseline

a. Tested configurations

Parameter	Tested Options
cost-function	Symmetric Huber function Asymmetric Huber function Symmetric truncated quadratic Asymmetric truncated quadratic
polynomial estimation order for baseline	1-10
threshold for baseline	0.01-0.1 steps of 0.01

b. MATLAB Implementation

Only one of the cost-function is included in this annex (due to space limitations). For each cost function the same code is applied.

```
function test_baseline(sensor,E)
```

```
% This implementation is repeated for each cost function fct= {'sh', 'ah', 'stq', 'atq'}
fct='sh';
count=1;
for(ord=1:10)
    for s=0.01:0.01:0.1
        [sensor_bck_sh(:,:,count), baselines_1(:,:,count),slope_E_sh(count),slope_B_sh(count), area_sh(count)]=
remove_baseline(sensor,E, ord,s,fct);
        set_sh(count,:)= [ord s];
        count=count+1;
    end
end
%best 10
slope_E_sh_aux=slope_E_sh;
for count=1:20
    [mins(count) pos_slope_E_sh(count)]=min(slope_E_sh_aux);
    slope_E_sh_aux(pos_slope_E_sh(count))=99999;
end
slope_B_sh_aux=slope_B_sh;
for count=1:20
    [mins_slope_B(count) pos_slope_B_sh(count)]=min(slope_B_sh_aux);
    slope_B_sh_aux(pos_slope_B_sh(count))=99999;
end
area_sh_aux=area_sh;
for count=1:20
    [mins_area_sh(count) pos_area_sh(count)]=min(area_sh_aux);
    area_sh_aux(pos_area_sh(count))=99999;
end
%worst
worst_slope_E_sh=max(slope_E_sh);
worst_slope_B_sh=max(slope_B_sh);
worst_area_sh=max(area_sh);
```

```
function [sensor_bck,baselines,slope_E,slope_B,area]=remove_baseline(sensor,E,ord,s,fcn)
```

```
[samples,recordings]=size(sensor);
area_aux=zeros(1,samples);
for signal = 1:samples
    [z,a,it]=backcor(E,sensor(signal,:),ord,s,fcn);
    baselines(signal,:)=z;
    sensor_bck(signal,:)=sensor(signal,:)-baselines(signal,:);
    %calculate slope from beginning to min
    slope_B_aux(signal)=(sensor_bck(signal,1)-sensor_bck(signal,40))/40;
    %calculate slope from min to end
    slope_E_aux(signal)=(sensor_bck(signal,recordings)-sensor_bck(signal,520))/35;
    %negative area
    for i=1:555
        if sensor_bck(signal,i)<0
            area_aux(signal)=area_aux(signal)+abs(sensor_bck(signal,i));
        end
    end
end
slope_E=mean(slope_E_aux);
slope_B=mean(slope_B_aux);
area=mean(area_aux);
```



c. Results

The results, and also the MATLAB functions can be found in:

<https://www.dropbox.com/sh/grx8ddo7osv3q03/AACT9xszzQZumGa3D1EpK4z7a?dl=0>

The results_baseline remove.mat file contains the outputs of test_baseline.m function.

For each cost function the following vectors are created as out from the test function:

- set_XX: The first column contains the polynomial order and the seconds column the threshold of each simulation (rows).
- Sensor_bck_XX: Contains the signal without baseline for each simulation.
- slope_E_XX, slope_B_XX, area_XX: contains the results for each simulation of the slope at the beginning of the signal, the slope at the end of the signal and the negative are respectively.
- pos_slope_E, pos_slope_B, pos_area_XX: contains the position in the above vectors of the 20 best results of the slope at the beginning of the signal, the slope at the end of the signal and the negative are respectively.
- worst_slope_E_XX, worst_slope_B_XX, worst_area_XX: contains the worst result.

II. Compression

a. Tested configurations

Parameter	Tested Options
wname	
Daubechies	'db1','db2','db3','db4','db5','db6','db7','db8','db9','db10','db11','db12','db15','db20','db25','db30','db35','db40','db45'
Coiflets	'coif1','coif2','coif3','coif4','coif5'
Symlets	'sym2','sym3','sym4','sym5','sym6','sym7','sym8','sym9','sym10','sym11','sym12','sym15','sym20','sym25'
Biorthogonal	'bior1.1','bior1.3','bior1.5','bior2.2','bior2.4','bior2.6','bior2.8','bior3.1','bior3.3','bior3.7','bior4.4','bior5.5','bior6.8'
Reverse Biorthogonal	'rbio1.1','rbio1.3','rbio1.5','rbio2.2','rbio2.4','rbio2.6','rbio2.8','rbio3.1','rbio3.3','rbio3.7','rbio4.4','rbio5.5','rbio6.8'
Decomposition level	1-6

b. MATLAB Implementation

`function [wname_over95 fc_over95 lev_over95]= test_compression(sensor)`

```
wname_best=";  
fc_best=0;  
lev_best=0;  
fc_over95=zeros(1,500);  
wname_over95=cell(1,500);  
wname_over95(:) = {'as'};  
lev_over95=zeros(1,500);  
i=1;  
% this is done for each wname  
wname = 'db1';  
for (lev=1:6)  
wname;  
lev;  
[sensor_comp, fc]= compression(sensor, wname,lev);  
if (fc_best<fc)  
fc_best=fc;  
wname_best=wname;  
lev_best=lev;  
end  
if (0.95<fc)  
fc_over95(i)=fc;  
wname_over95(i)={wname};  
lev_over95(i)=lev;  
i=i+1;  
end
```

`function [sensor_comp, fc]= compression(sensor, wname,lev)`

```
[samples recordings]= size(sensor);  
fc_bot=0;  
fc_top=0;  
for signal = 1:samples  
[c,l] = wavedec(sensor(signal,:),lev,wname);  
sensor_comp(signal,:) = c(1:l(1));  
c(l(1)+1:end)=0;  
reconstruction= waverec(c,l,wname);  
for i=1:recordings  
fc_top_now=min(sensor(signal,i), reconstruction(i));  
fc_bot_now=max(sensor(signal,i), reconstruction(i));  
fc_top=fc_top_now+fc_top;  
fc_bot=fc_bot_now+fc_bot;  
end  
fc_all(signal)=fc_top/fc_bot;  
end  
fc=mean(fc_all); %% calculate error
```

c. Results

The results, and also the MATLAB functions can be found in:

https://www.dropbox.com/sh/1e8uh1gwhlv9jo/AABln_CKzYfWHAHo_GdjQdZha?dl=0

For each sensor the following vectors are created as out from the test function: `fc_over95_X`, `wname_over95_X`, `lev_over95_X`. Each column contains the results and



configuration (wname and lev) that fulfil $0.95 < fc$: fc value, the mother wavelet and the level of decomposition respectively.

III. Artificial Neural Network

a. Tested configurations

Parameter	Description	Set value
Create network	Used to create different types of networks	Feedforwardnet fitnet
hiddenLayerSize	defines number of hidden layers (number of columns) and neurons (value in each column)	1 hidden layer: [1-10]; 2 hidden layer: [1-5; 1-5]; 3 hidden layer: [1-5; 1-5; 1-5];
net.numInputs	number of sensors	4
net.inputs{i}.size	number of DWT coefficients	95
net.inputs{i}.processFcns	Normalize all inputs	'mapminmax'
trainFcn	Training function	trainbr trainlm
net.layers{i}.transferFcn	transfer functions is used to calculate the ith layer's output, given the layer's net input, during simulation and training.	Logsig Satlins Purelin tansig
net.inputConnect	connections between inputs and hidden layers	1 hidden layer: [1 1 1 1; 0 0 0 0]; 2 hidden layer: [1 1 1 1; 0 0 0 0; 0 0 0 0]; 3 hidden layer: [1 1 1 1; 0 0 0 0; 0 0 0 0; 0 0 0 0];
net.divideFcn	Divide data randomly into tran/validation/test	'dividerand'
net.divideMode	Divide up every sample	'sample'
net.divideParam.trainRatio	% of saamples for training	LM: 70; BAY: 80
net.divideParam.valRatio	% of saamples for validation	LM: 10; BAY: 0
net.divideParam.testRatio	% of saamples for testing	for LM: 20; BAY: 20
net.performFcn	Performance Function Mean squared error	'mse'
net.trainParam.epochs	Maximum number of epochs to train/iterations	200
net.trainParam.goal	Performance goal	0.00000001

The specific configurations tested can be seen in the list of functions in the next section as each ANN_create_createNet_trainFcn _ transferFcn.

b. MATLAB Implementation

The ANN_test functions variate the number of neurons per layer and call ANN_create to create the network with a specific configuration.

1 hidden layer (1-10 neurons)

- ANN_test_fit_LM_1:
 - ANN_create_fit_LM_L_P
 - ANN_create_fit_LM_S_P
 - ANN_create_fit_LM_T_P
 - ANN_create_fit_LM_P_P
 - ANN_create_fit_LM_L_S
 - ANN_create_fit_LM_S_S
 - ANN_create_fit_LM_T_S
 - ANN_create_fit_LM_P_S
- ANN_test_feed_LM_1
 - ANN_create_feed_LM_L_P
 - ANN_create_feed_LM_S_P
 - ANN_create_feed_LM_T_P
 - ANN_create_feed_LM_P_P
 - ANN_create_feed_LM_L_S
 - ANN_create_feed_LM_S_S
 - ANN_create_feed_LM_T_S
 - ANN_create_feed_LM_P_S
- ANN_test_fit_BAY_1
 - ANN_create_fit_BAY_L_P
 - ANN_create_fit_BAY_S_P
 - ANN_create_fit_BAY_T_P
 - ANN_create_fit_BAY_P_P
 - ANN_create_fit_BAY_L_S
 - ANN_create_fit_BAY_S_S
 - ANN_create_fit_BAY_T_S
 - ANN_create_fit_BAY_P_S
- ANN_test_feed_BAY_1
 - ANN_create_feed_BAY_L_P
 - ANN_create_feed_BAY_S_P
 - ANN_create_feed_BAY_T_P
 - ANN_create_feed_BAY_P_P
 - ANN_create_feed_BAY_L_S
 - ANN_create_feed_BAY_S_S
 - ANN_create_feed_BAY_T_S
 - ANN_create_feed_BAY_P_S

2 hidden layers (1-5 neurons per layer)

- ANN_test_fit_LM_2_EQU:
 - ANN_create_fit_LM_LL_L
 - ANN_create_fit_LM_SS_S
 - ANN_create_fit_LM_TT_T
 - ANN_create_fit_LM_PP_P
 - ANN_create_fit_LM_PP_S
 - ANN_create_fit_LM_LL_S
 - ANN_create_fit_LM_TT_S
 - ANN_create_fit_LM_SS_P
 - ANN_create_fit_LM_LL_P
 - ANN_create_fit_LM_TT_P
- ANN_test_fit_LM_2_DIF_DIF:
 - ANN_create_fit_LM_TS_P
 - ANN_create_fit_LM_ST_P
 - ANN_create_fit_LM_TL_P
 - ANN_create_fit_LM_LT_P

- ANN_create_fit_LM_LS_P
 - ANN_create_fit_LM_SL_P
 - ANN_create_fit_LM_TS_S
 - ANN_create_fit_LM_ST_S
 - ANN_create_fit_LM_TL_S
 - ANN_create_fit_LM_LT_S
 - ANN_create_fit_LM_LS_S
 - ANN_create_fit_LM_SL_S
- ANN_test_feed_LM_2_EQU
 - ANN_create_feed_LM_LL_L
 - ANN_create_feed_LM_SS_S
 - ANN_create_feed_LM_TT_T
 - ANN_create_feed_LM_PP_P
 - ANN_create_feed_LM_PP_S
 - ANN_create_feed_LM_LL_S
 - ANN_create_feed_LM_TT_S
 - ANN_create_feed_LM_SS_P
 - ANN_create_feed_LM_LL_P
 - ANN_create_feed_LM_TT_P
- ANN_test_feed_LM_2_DIF_DIF
 - ANN_create_feed_LM_TS_P
 - ANN_create_feed_LM_ST_P
 - ANN_create_feed_LM_TL_P
 - ANN_create_feed_LM_LT_P
 - ANN_create_feed_LM_LS_P
 - ANN_create_feed_LM_SL_P
 - ANN_create_feed_LM_TS_S
 - ANN_create_feed_LM_ST_S
 - ANN_create_feed_LM_TL_S
 - ANN_create_feed_LM_LS_S
 - ANN_create_feed_LM_SL_S
- ANN_test_fit_BAY_2_EQU
 - ANN_create_fit_BAY_LL_L
 - ANN_create_fit_BAY_SS_S
 - ANN_create_fit_BAY_TT_T
 - ANN_create_fit_BAY_PP_P
 - ANN_create_fit_BAY_PP_S
 - ANN_create_fit_BAY_LL_S
 - ANN_create_fit_BAY_TT_S
 - ANN_create_fit_BAY_SS_P
 - ANN_create_fit_BAY_LL_P
 - ANN_create_fit_BAY_TT_P
- ANN_test_fit_BAY_2_DIF_DIF
 - ANN_create_fit_BAY_TS_P
 - ANN_create_fit_BAY_ST_P
 - ANN_create_fit_BAY_TL_P
 - ANN_create_fit_BAY_LT_P
 - ANN_create_fit_BAY_LS_P
 - ANN_create_fit_BAY_SL_P
 - ANN_create_fit_BAY_TS_S
 - ANN_create_fit_BAY_ST_S
 - ANN_create_fit_BAY_TL_S
 - ANN_create_fit_BAY_LT_S

- ANN_create_fit_BAY_LS_S
- ANN_create_fit_BAY_SL_S
- ANN_test_feed_BAY_2_EQU
 - ANN_create_feed_BAY_LL_L
 - ANN_create_feed_BAY_SS_S
 - ANN_create_feed_BAY_TT_T
 - ANN_create_feed_BAY_PP_P
 - ANN_create_feed_BAY_PP_S
 - ANN_create_feed_BAY_LL_S
 - ANN_create_feed_BAY_TT_S
 - ANN_create_feed_BAY_SS_P
 - ANN_create_feed_BAY_LL_P
 - ANN_create_feed_BAY_TT_P
- ANN_test_feed_BAY_2_DIF_DIF
 - ANN_create_feed_BAY_TS_P
 - ANN_create_feed_BAY_ST_P
 - ANN_create_feed_BAY_TL_P
 - ANN_create_feed_BAY_LT_P
 - ANN_create_feed_BAY_LS_P
 - ANN_create_feed_BAY_SL_P
 - ANN_create_feed_BAY_TS_S
 - ANN_create_feed_BAY_ST_S
 - ANN_create_feed_BAY_TL_S
 - ANN_create_feed_BAY_LT_S
 - ANN_create_feed_BAY_LS_S
 - ANN_create_feed_BAY_SL_S
- ANN_create_fit_LM_TTTP
- ANN_create_fit_LM_LLLP
- ANN_test_feed_LM_3
 - ANN_create_feed_LM_PPPP
 - ANN_create_feed_LM_SSSS
 - ANN_create_feed_LM_TTTT
 - ANN_create_feed_LM_LLLL
 - ANN_create_feed_LM_PPPS
 - ANN_create_feed_LM_TTTS
 - ANN_create_feed_LM_LLLS
 - ANN_create_feed_LM_SSSP
 - ANN_create_feed_LM_TTTP
 - ANN_create_feed_LM_LLLP
- ANN_test_fit_BAY_3
 - ANN_create_fit_BAY_PPPP
 - ANN_create_fit_BAY_SSSS
 - ANN_create_fit_BAY_TTTT
 - ANN_create_fit_BAY_LLLL
 - ANN_create_fit_BAY_PPPS
 - ANN_create_fit_BAY_TTTS
 - ANN_create_fit_BAY_LLLS
 - ANN_create_fit_BAY_SSSP
 - ANN_create_fit_BAY_TTTP
 - ANN_create_fit_BAY_LLLP
- ANN_test_feed_BAY_3
 - ANN_create_feed_BAY_PPPP
 - ANN_create_feed_BAY_SSSS
 - ANN_create_feed_BAY_TTTT
 - ANN_create_feed_BAY_LLLL
 - ANN_create_feed_BAY_PPPS
 - ANN_create_feed_BAY_TTTS
 - ANN_create_feed_BAY_LLLS
 - ANN_create_feed_BAY_SSSP
 - ANN_create_feed_BAY_TTTP
 - ANN_create_feed_BAY_LLLP

3 hidden layers (1-5 neurons per layer)

- ANN_test_fit_LM_3:
 - ANN_create_fit_LM_PPPP
 - ANN_create_fit_LM_SSSS
 - ANN_create_fit_LM_TTTT
 - ANN_create_fit_LM_LLLL
 - ANN_create_fit_LM_PPPS
 - ANN_create_fit_LM_TTTS
 - ANN_create_fit_LM_LLLS
 - ANN_create_fit_LM_SSSP

Only one creation and test files have been included as an example. It is indicated where the paramaters vary from file to file.

function [net] = ANN_create_1c(hiddenLayerSize)

%Acoording to learning algorithm: trainbr-Bayesian Regulation or trainlm - Levenberg-Marquardt

trainFcn = 'trainlm';

net = fitnet(hiddenLayerSize,trainFcn); %%net=feedforwardnet(hiddenLayerSize,'trainbfg');

net.numInputs=4; % number of sensors

for i=1:4

net.inputs{i}.processFcns = {'mapminmax'};

net.inputs{i}.size= 95; % number of coefficients per sensor

end

```

% % Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.divideParam.trainRatio = 70/100; % for LM: 70; BAY: 80
net.divideParam.valRatio = 10/100; % for LM: 10; BAY: 0
net.divideParam.testRatio = 20/100; % for LM: 20; BAY: 20

%According to number of hidden layers 1 hidden layer [1 1 1 1; 0 0 0 0]; 2 hidden layer: [1 1 1 1; 0 0 0 0; 0
0 0 0]; 3 hidden layer: [1 1 1 1; 0 0 0 0; 0 0 0 0; 0 0 0 0];
net.inputConnect=[1 1 1 1; 0 0 0 0; 0 0 0 0];
%% transfer functions for hidden layers; changes for each file
num_Layer=net.numLayers; % number layers
net.layers{1}.transferFcn='tansig';
net.layers{num_Layer}.transferFcn='purelin'; %% Output layer
% Choose a Performance Function
net.performFcn = 'mse'; % Mean squared error
net.trainParam.epochs = 200; % %200 Maximum number of epochs to train
net.trainParam.goal= 0.00000001; % % Performance goal
net.trainParam.time=60*60*5; % %Maximum time to train in second

```

function ANN_test_comp_fitnet_LM_2_DIF_DIF(sensors, conc, test_sensor, test_conc)

```

PRAE_1c=zeros(200,1);
lof_1c=zeros(200,1);
test_results_1c=zeros(200,14);
hiddenLayerSize=[6 6 6];
hidden_1c=zeros(200,2);
count=1;
for i1=1:5
    for i2=1:5
        %set hidden layers neurons
        hiddenSize(1)=hiddenLayerSize(1)-i1;
        hidden_1c(count,:)=hiddenSize; %save config
        %create network -->each repetition change what file calls
        [net]= ANN_create_fit_LM_TS_P(hiddenSize);
        [net,tr] = train(net,sensors,conc);
        trs_1c(count)=tr; %save results
        test_results = net(test_sensor); %save test concentrations (estimated conc_p)
        aux=cell2mat(test_results);
        test_results_1c(count,:)=aux;
        PRAE_aux2=0; lof_aux1=0; lof_aux2=0;
        for m=1:size(test_conc)
            PRAE_aux=abs(test_conc(m)-aux(m))/test_conc(m);
            PRAE_aux2=PRAE_aux2+PRAE_aux;
            lof_aux1=(test_conc(m)-aux(m))^2+lof_aux1;
            lof_aux2=(test_conc(m)^2)+lof_aux2;
        end
        [aux2, aux3]=size(test_conc);
        PRAE_1c(count)=PRAE_aux2/aux3*100;
        lof_1c(count)=sqrt(lof_aux1/lof_aux2)*100;
        count=count+1;
    end
end
end

```

%change where to save each repetition

```
save('fitnet_LM_TS_P.mat','PRAE_1c','lof_1c','hidden_1c','trs_1c','test_results_1c','sensors','conc','test_sensor','test_conc');
```



c. Results

The results, and also the MATLAB functions can be found in:

<https://www.dropbox.com/sh/f6owl6ww0tj3sxs/AABvXhPvrcQWf3znfBtoVYdya?dl=0>

For each ANN configuration the following vectors are created as out from the different test functions:

- Hidden: Each column represents a hidden layer and the value it contains de number of neurons.
- lof: lof value for each configuration
- PRAE: PRAE value for each configuration
- trs: each row contains details of a specific network. Particularly interesting the MSE values for training, validation and testing.
- test_results: concentrations obtained after training the network and running it with test data.