UNIVERSITÀ DI PISA

Computer Engineering

Intelligent Systems

ECG Fitting, Forecasting and Activity Classification

Group Project Report

TEAM MEMBERS:
Biagio Cornacchia
Gianluca Gemini
Matteo Abaterusso

Academic Year: 2021/2022

Contents

1 Introduction

The aim of the project is to apply **machine learning**, **deep learning** and **fuzzy system** technologies in order to classify **human activities** (such as *walking*, *sitting* and *running*), fit **ECG feature values** (such as *mean* and *standard deviation*) and forecast **ECG values**. To achieve these goals, the following technologies have been used:

- **Multi-layer Perceptron**, **Radial Basis Function** and **Convolutional Neural Network** for ECG feature values fitting
- **Multi-layer Perceptron**, **Fuzzy Inference System** and **Adaptive Network-based Fuzzy Inference System** for activity classification
- **Recurrent Neural Network** for ECG forecasting

1.1 Dataset Structure

The dataset refers to **22 subjects** who have been monitored during **three different activities**: *sit*, *walk* and *run*. For each subject **11 signals** have been recorded:

- **pleth_1**: red wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)
- **pleth_2**: infrared wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)
- **pleth_3**: green wavelength PPG from the distal phalanx (first segment) of the left index finger palmar side (sampling rate 500 Hz)
- **pleth_4**: red wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)
- **pleth_5**: infrared wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)
- **pleth_6**: green wavelength PPG from the proximal phalanx (base segment) of the left index finger palmar side (sampling rate 500 Hz)
- **lc_1**: load cell proximal phalanx (first segment) PPG sensor attachment pressure (sampling rate 80Hz)
- **lc_2**: load cell (base segment) PPG sensor attachment pressure (sampling rate 80Hz)
- **temp_1**: distal phalanx (first segment) PPG sensor temperature (°C, sampling rate 10Hz)
- **temp_2**: proximal phalanx (base segment) PPG sensor temperature in (°C, sampling rate 10 Hz)
- **temp_3**: ambient temperature (°C, sampling rate 500 Hz)
- **ecg**: 3-channel ECG sampled at 500 Hz

ECG and signals are associated with a **timestamp** that represents the sampling time. The dataset is organized into 66 csv files, each one containing signals/ECG for a subject for a specific activity.

2 Fitting and Classification using Neural Networks

In this phase, the aim is to exploit different types of neural networks to estimate the mean and the standard deviation of the ECG. This is done starting from a set of features extracted from the dataset introduced in Section 1.1.

2.1 Data Preprocessing

The first step is to extract the features from the dataset. For each signal, the following features have been computed:

- Mean
- Median
- Variance
- Standard Deviation
- Minimum
- Maximum
- Kurtosis
- Skewness
- Inter-quartile Range Gsr
- Energy
- Mean Frequency
- Median Frequency
- Occupied Bandwidth

The first requirement is to find neural networks that have to predict the mean and standard deviation of the ECG (Section 2.2). So, it is necessary to compute these values starting from the time-series in the initial dataset. A single mean/standard deviation value can be computed over an entire file of the dataset or over a subset of it, named **window**. In the second case, also the features of the signals have to be computed over the window. The chosen approach is the window one, specifically, **overlapped window** have been chosen (every window is composed by *50.000* samples, with an overlapping factor of *30%* of the window size). In addition, for each window the corresponding activity is extracted, which is used for the activity classification requirement (Section 2.3).

Since the number of samples to perform a training is not enough, **data augmentation** is required. To generate new samples, an **auto-encoder** is used. In particular, the initial features matrix is shuffled and then divided into K folds, where *K-1* are used to train the auto-encoder and the last one is used to generate the new samples. This is repeated for K times, varying the fold used for data generation.

Finally, the 143 features (13 features per 11 signals) are reduced to the **10 most relevant**. This is done by removing the features correlated by a factor greater than 0.9 and, after that, applying the **sequentialfs**. Specifically, the sequentialfs is performed two times in

order to generate the best features matrix for the mean and the standard deviation ECG target vector.

2.2 ECG Fitting

For this requirement, the features matrix is used to predict the **mean/standard deviation ECG** value. This is done using an **Multi-layer Perceptron** and a **Radial Basis Function**. As said, the features matrix is different for both targets and each of them contains the 10 most relevant features.

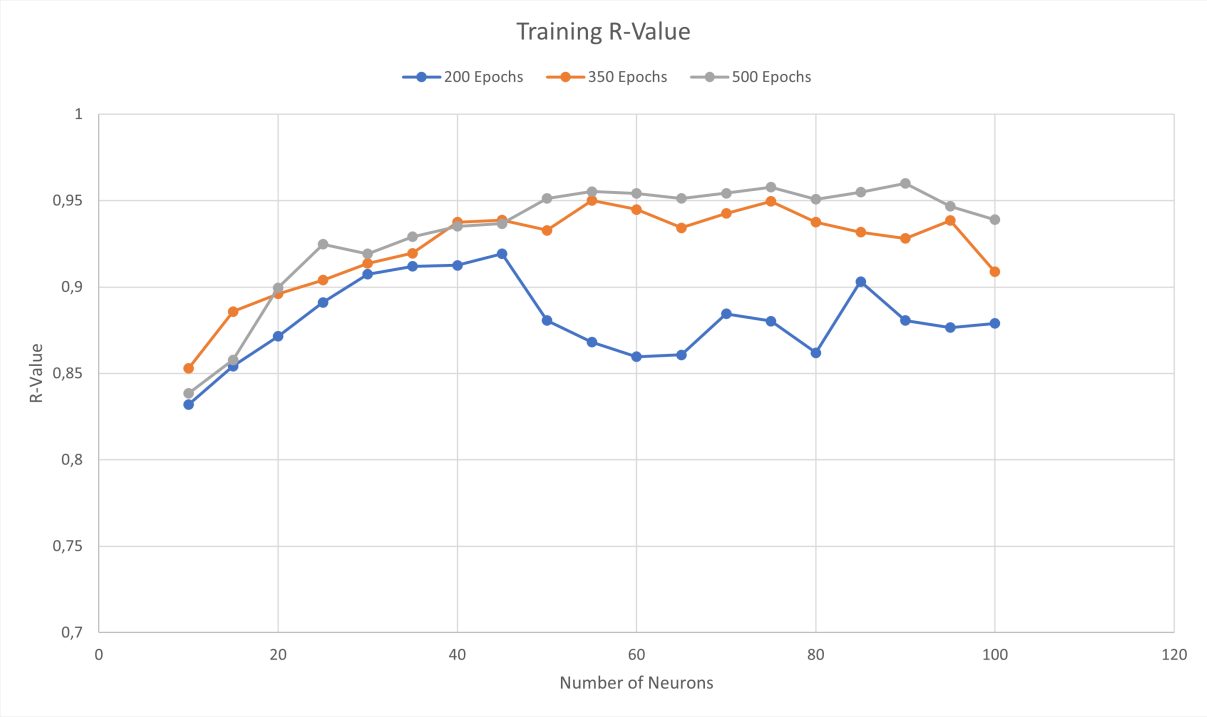
2.2.1 Multi-layer Perceptron

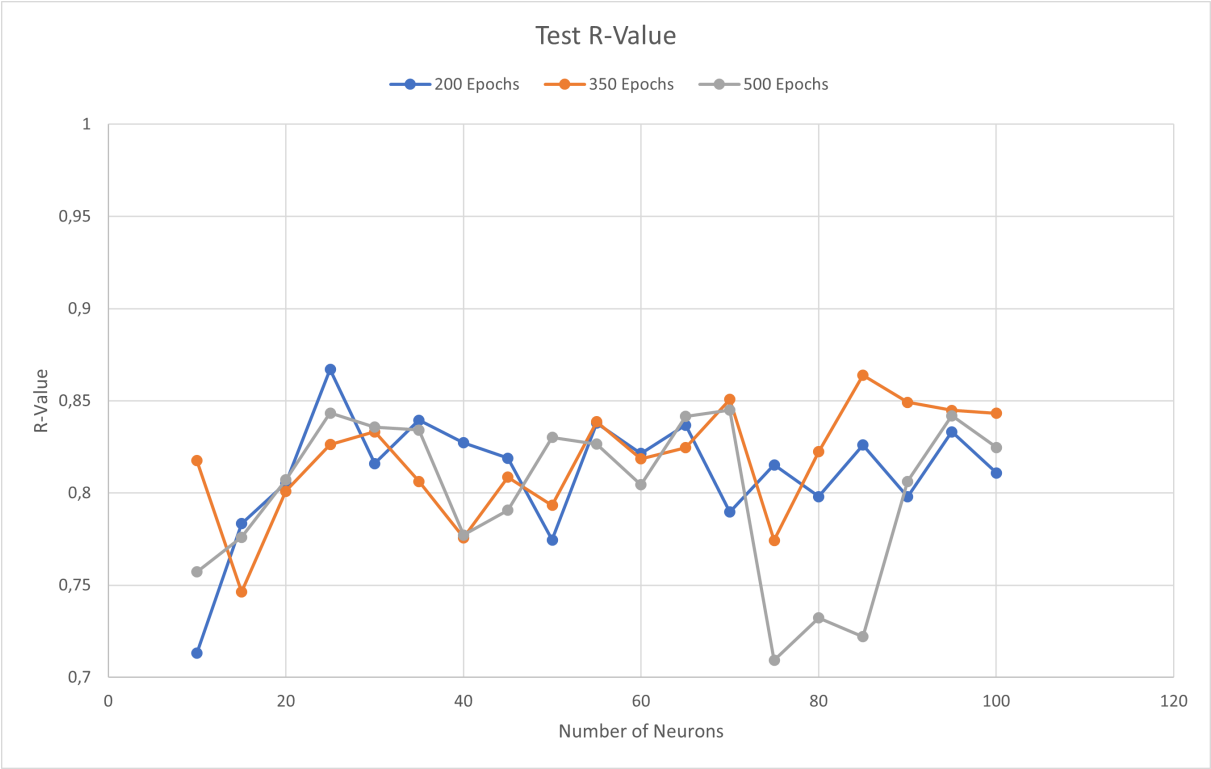
This is a **fitting problem** that can be solved using the *fitnet* class in MatLab. For this type of network, the main parameters that have been tuned are the **number of neurons** and the **maximum number of epochs**. So, a finder script has been implemented. It tests the different combinations of the following parameters range:

- Number of neurons: [10 100] with a step of 5
- Maximum number of epochs: [200 350 500]

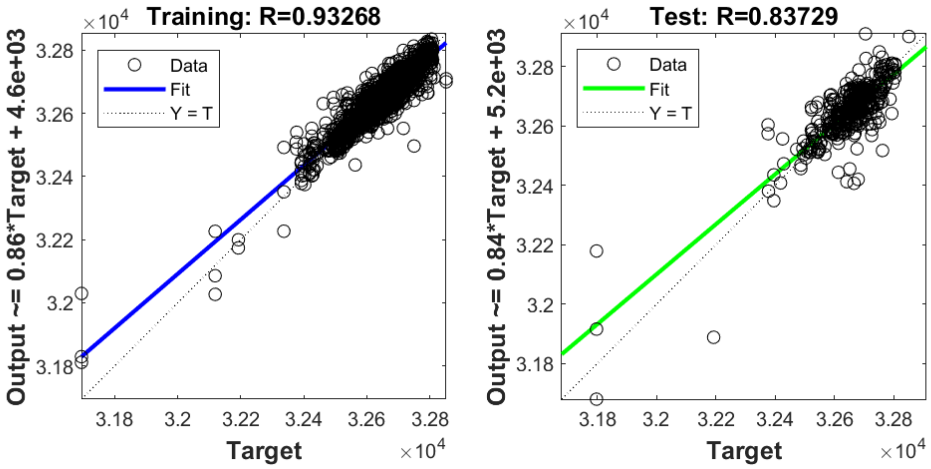
These range extremes have been chosen taking into account that for values greater than *100* neurons and *500* epochs, the network has a tendency to overfit. The chosen **training function** is *trainbr* as it is the one with better efficiency. Moreover, it doesn't use the validation set, so the chosen data partitioning consists of 85% training set and 15% test set.

The results for the mean ECG obtained by the finder are:

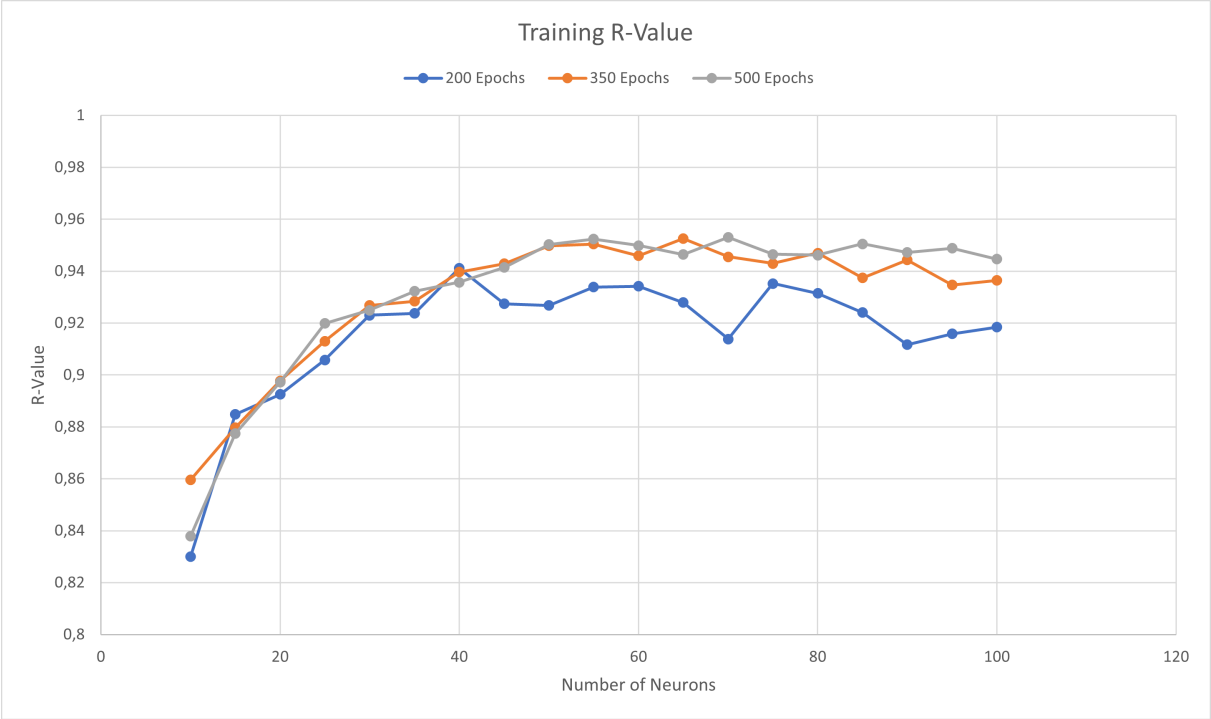


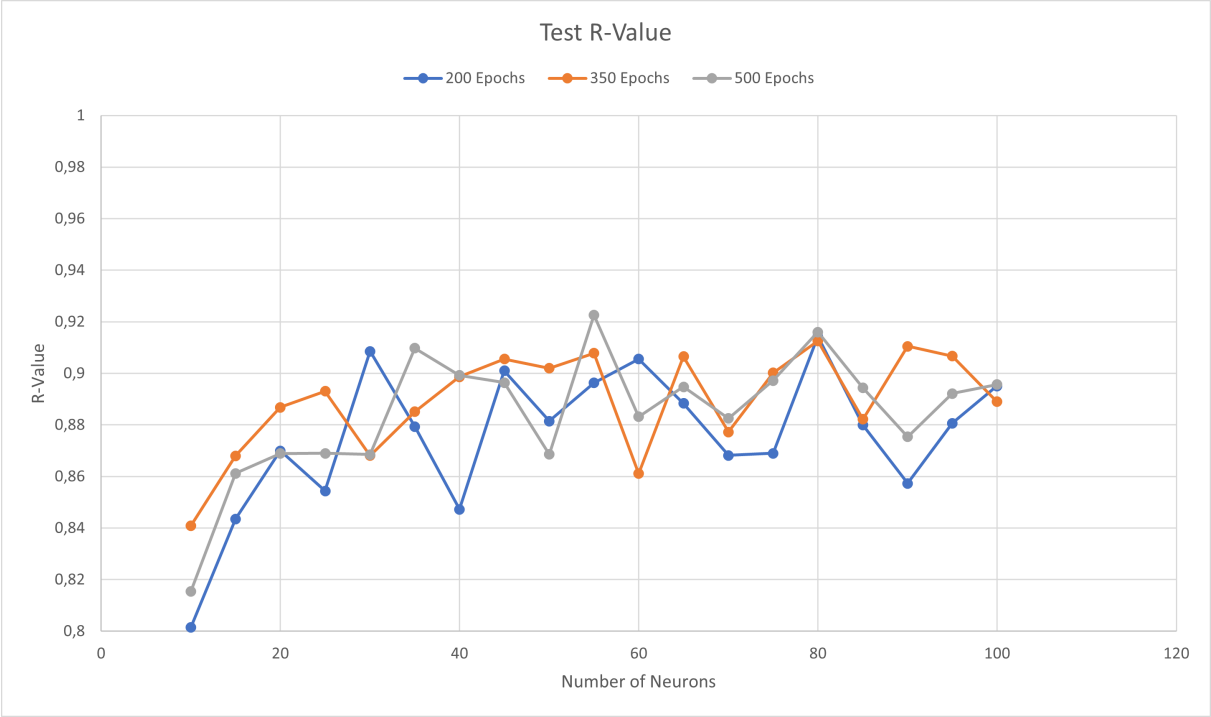


The best regression value is achieved using a number of neurons equal to *85* and *350* epochs. The network trained using these parameters allows to obtain a **mean squared error** of *3969.854* and the following **regression** values:

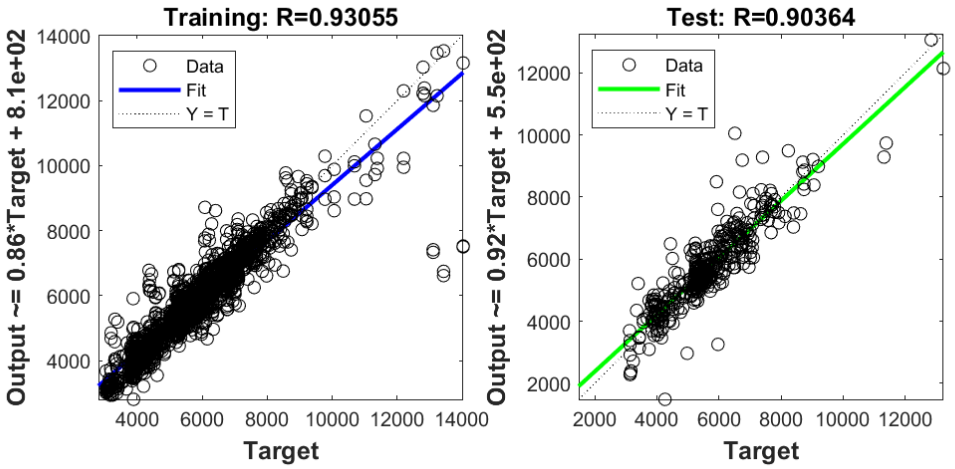


Instead, the results for the standard deviation ECG obtained by the finder are:





In this case, the best regression value is achieved using a number of neurons equal to *95* and *350* epochs. The network trained using these parameters allows to obtain a **mean squared error** of *373661.4* and the following **regression** values:



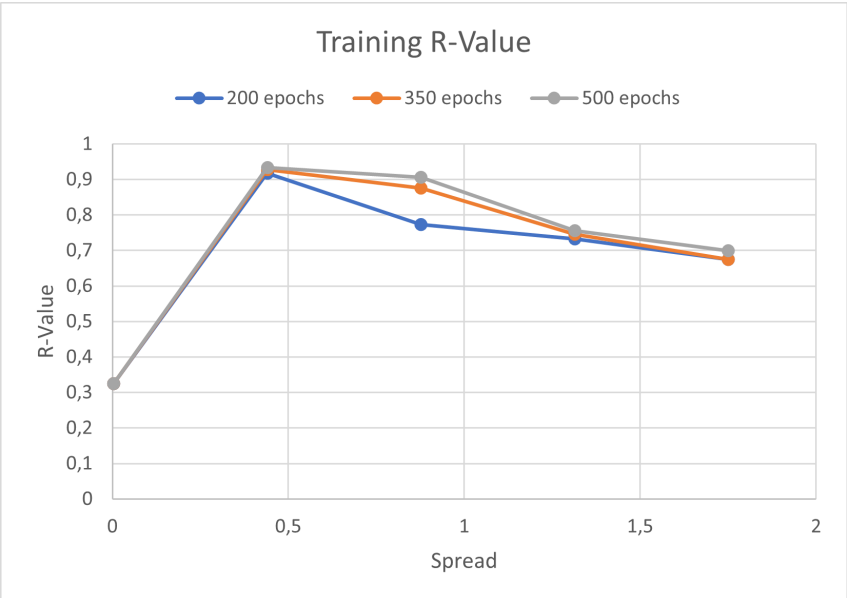
2.2.2 Radial Basis Function

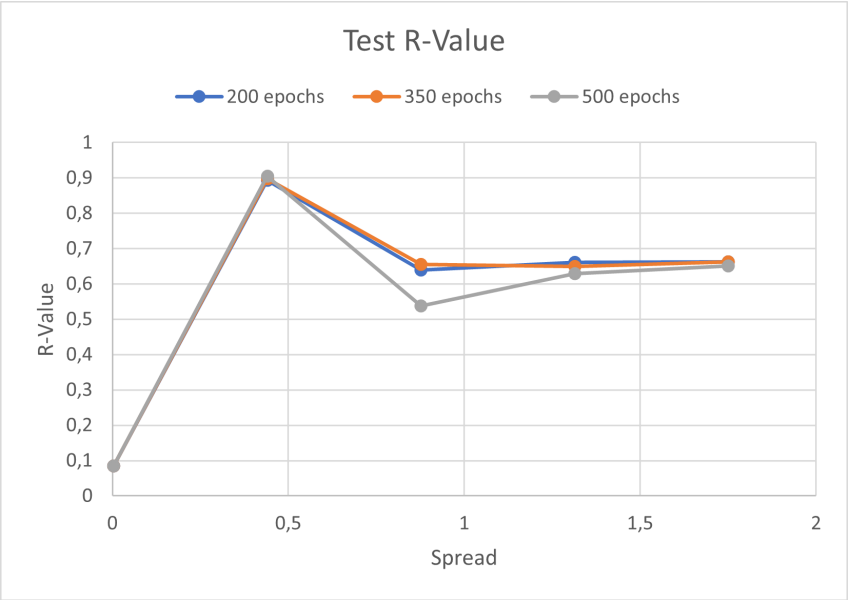
A variant of the *fitnet* to solve a fitting problem is the *newrb* that is based on Radial Basis Function. This class of networks automatically discover the best number of neurons, but it requires the **spread** value as input. Usually, the spread value is chosen considering the Euclidean distance d between the features matrix points. So, even in this case a finder script has been implemented, that tests the different combinations of the following parameters range:

- Spread value: [*min(d) max(d)*] with a step equal to 20% of the distance between the two extremes
- Number of epochs: [200 350 500]

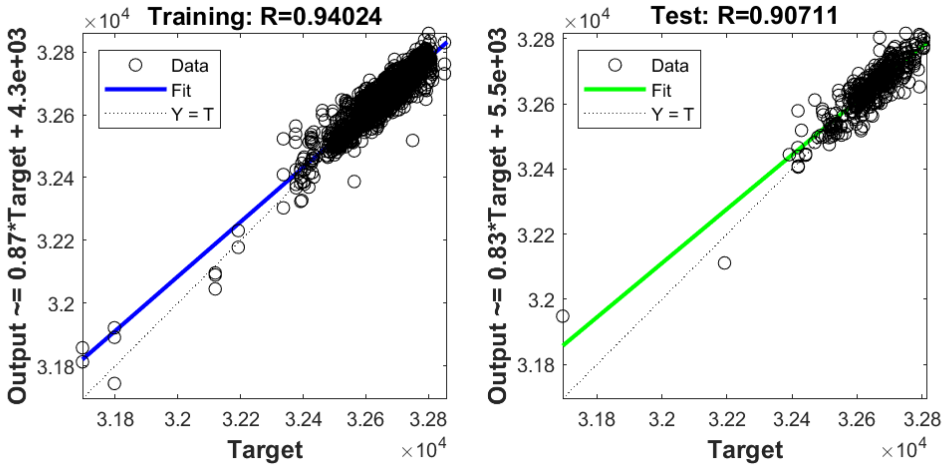
As for the Multi-layer Perceptron, the best efficiency is given by the *trainbr* **training function**, with a data partitioning of 85% training set and 15% test set.

The results for the mean ECG obtained by the finder are:

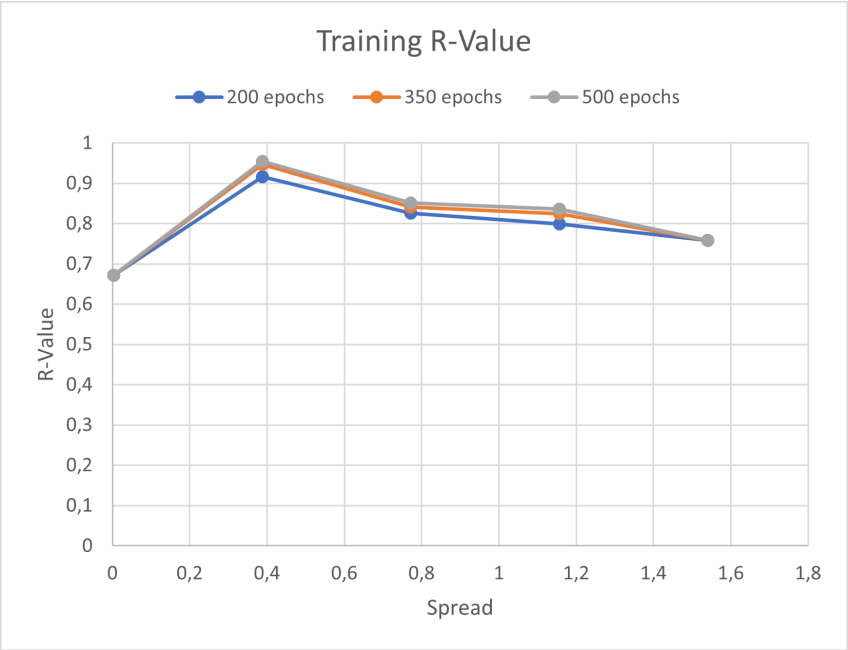


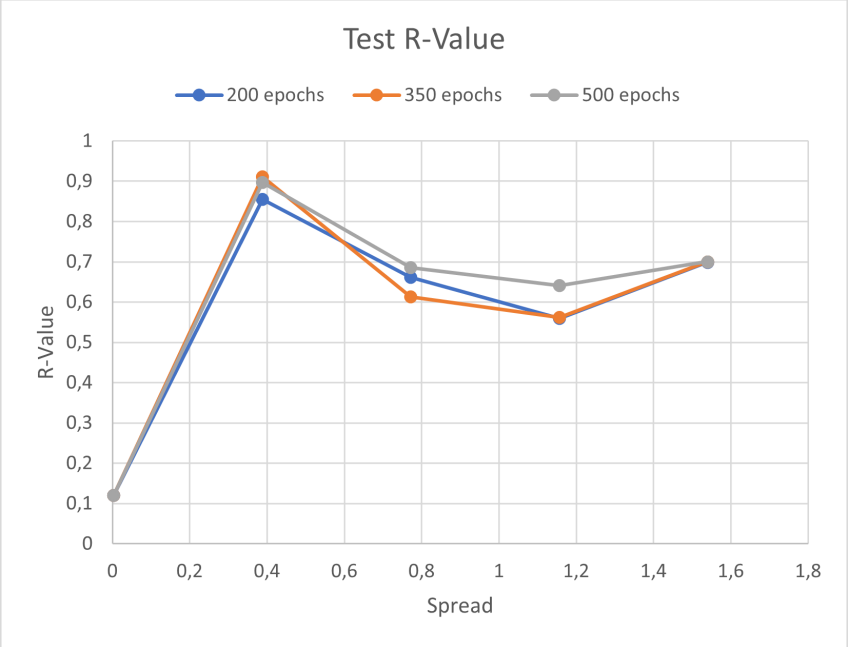


The best regression value is achieved using a spread value equal to *0.44* and *500* epochs. The network trained using these parameters allows to obtain a **mean squared error** of *1567.326* and the following **regression** values:

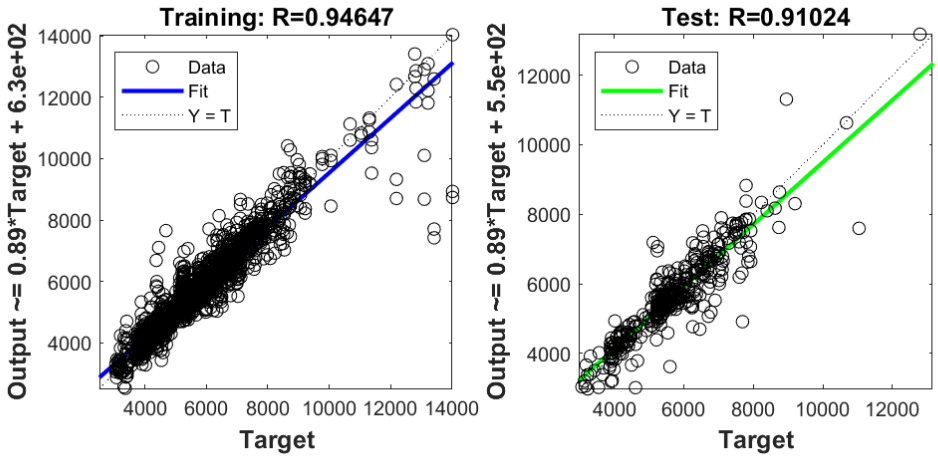


Regarding the standard deviation ECG, the results obtained by the finder are:





In this case, the best regression value is obtained using a spread value equal to *0.388* and *350* epochs. These parameters produces a **mean squared error** of *281541.7* and the following **regression** values:



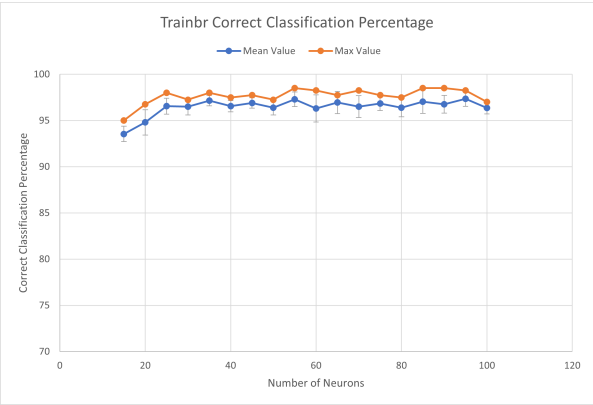
2.3 Activity Classification

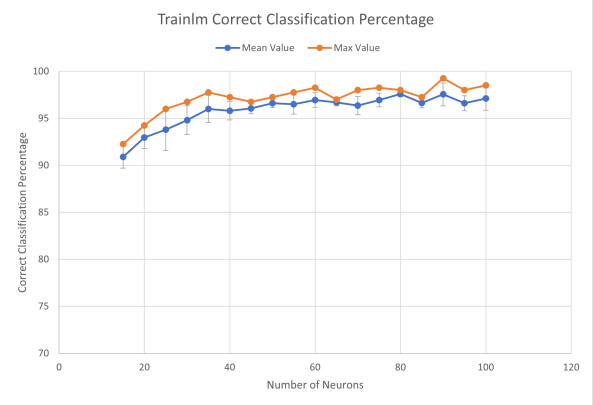
For this requirement, a **Multi-layer Perceptron** has to be used. The required network has to be capable of classify an activity given as input a features matrix that is the union of the two features matrices used for the mean and standard deviation ECG fittings.

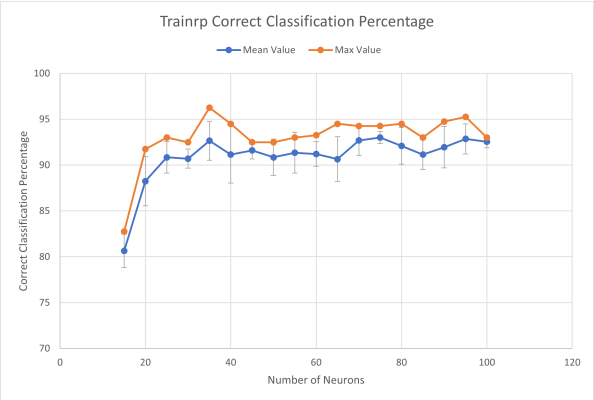
This is a **classification problem** that can be solved using the *patternet* class in MatLab. For this type of network, the main parameters that can be tuned are the **number of neurons** and the **training function**. So, even in this case, a finder script has been implemented. It tests the different combinations of the following parameters range:

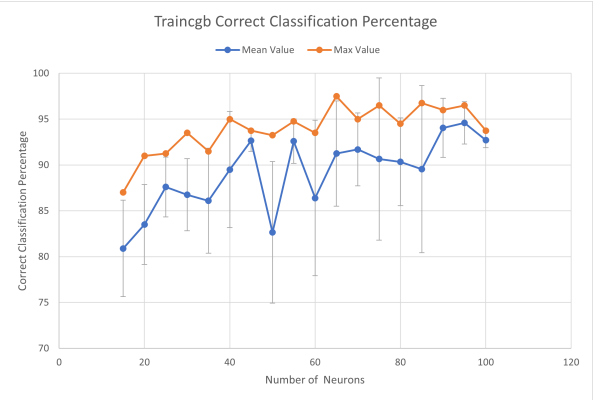
- Number of neurons: [10 100] with a step of 5
- Training function: [*trainlm*, *trainbr*, *trainbfg*, *trainrp*, *trainscg*, *traincgb*, *traincgf*, *traincgp*, *trainoss*, *traingdx*, *traingdm*, *traingd*]

For every combination, 5 different networks have been trained. The best 4 training functions are:

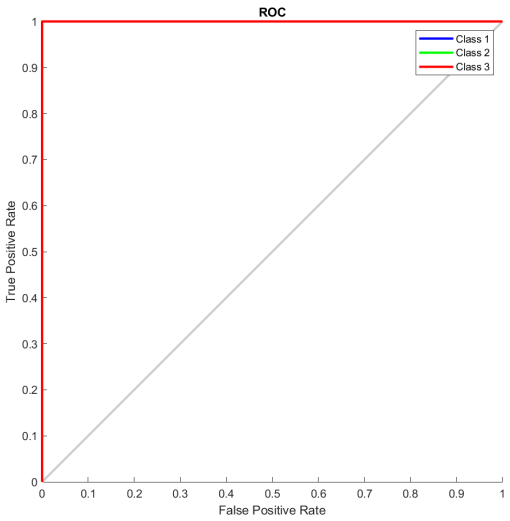


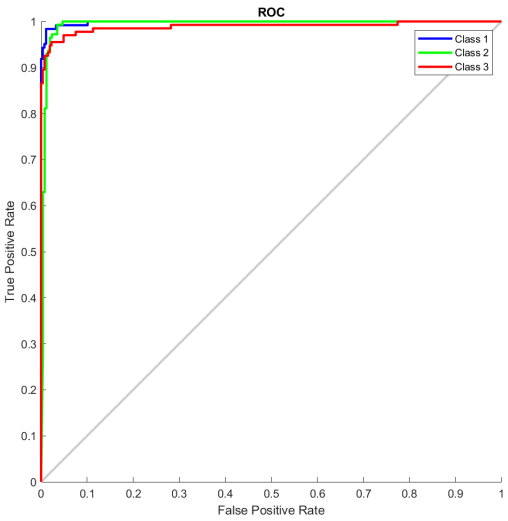




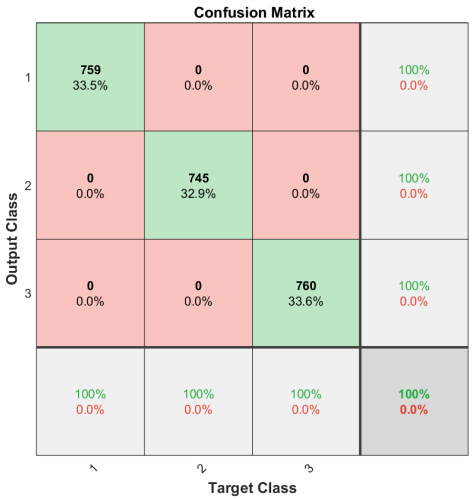


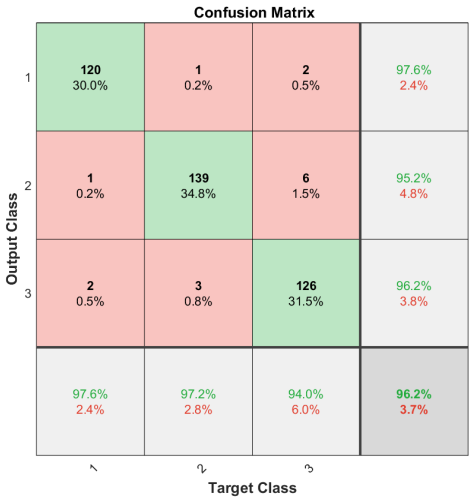
So, the selected parameters are *35* neurons and *trainbr* function. Using these parameters, the following **ROC curves** have been achieved:





The same results can be seen in the **confusion matrix**:





3 Classification using Fuzzy Systems

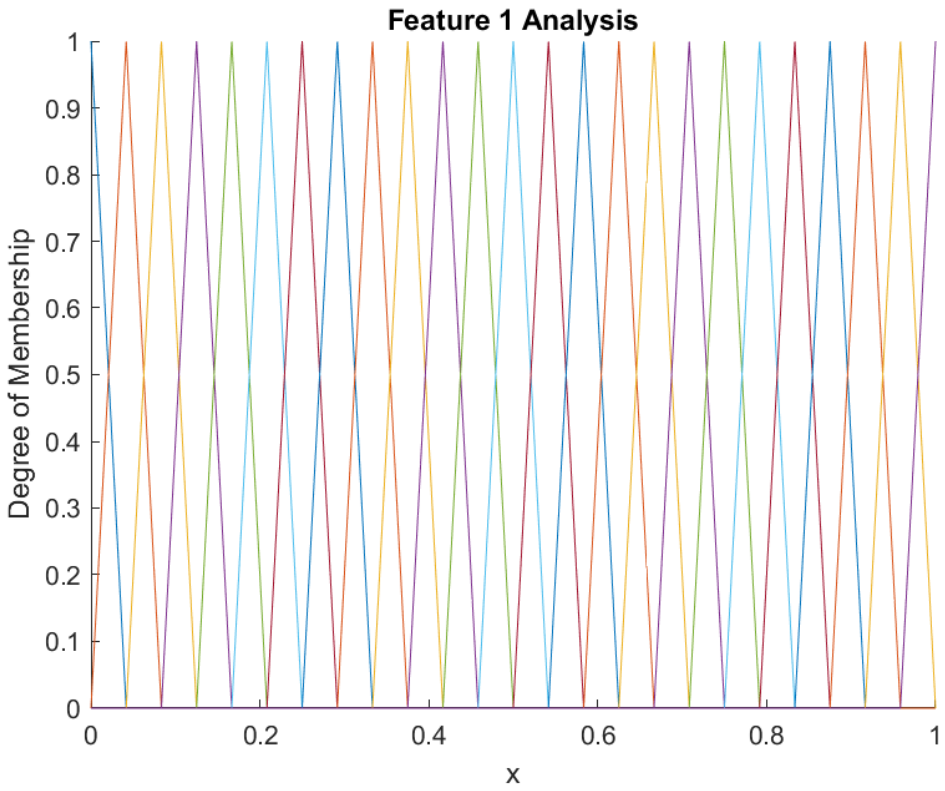
The aim of this part is to exploit the **Fuzzy Systems** to implement a person's activity classifier. This is done using the **5 most relevant features** taken from the set of features used to train the classifier developed in Section 2.3.

3.1 Fuzzy Inference System

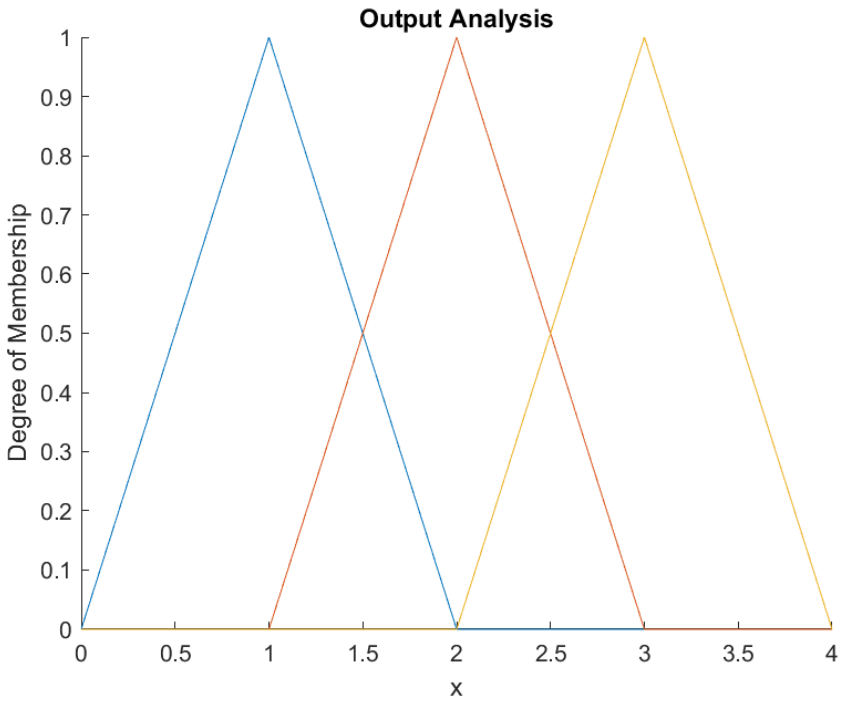
The first type of system is the **Fuzzy Inference System**, implemented by the *mamfis* MatLab class. This kind of system requires rules to be generated manually by an expert. In order to get the better result, the **Wang-Mendel** method has been used.

3.1.1 Fuzzy Rules Generation

The Wang-Mendel generates **automatically** the fuzzy rules, starting from the features matrix and the targets. Firstly, it has been chosen to assign the same number of membership functions to each feature. Specifically, the membership functions used are **triangular** ones as shown below:



Also for the output, **triangular** membership functions have been used, specifically, one for each class (*sit*, *walk* and *run*):



In the Wang-Mendel method, for each pair features-target, every feature X_i is associated to the membership function B_i with the **highest membership degree**. The same is done for the target which is associated to the best membership function C_i. In this way, the resulting rule is:

$$\text{if } X_1 \text{ is } B_1 \text{ and } X_2 \text{ is } B_2 \text{ and ... then } \textit{output} \text{ is } C_i$$

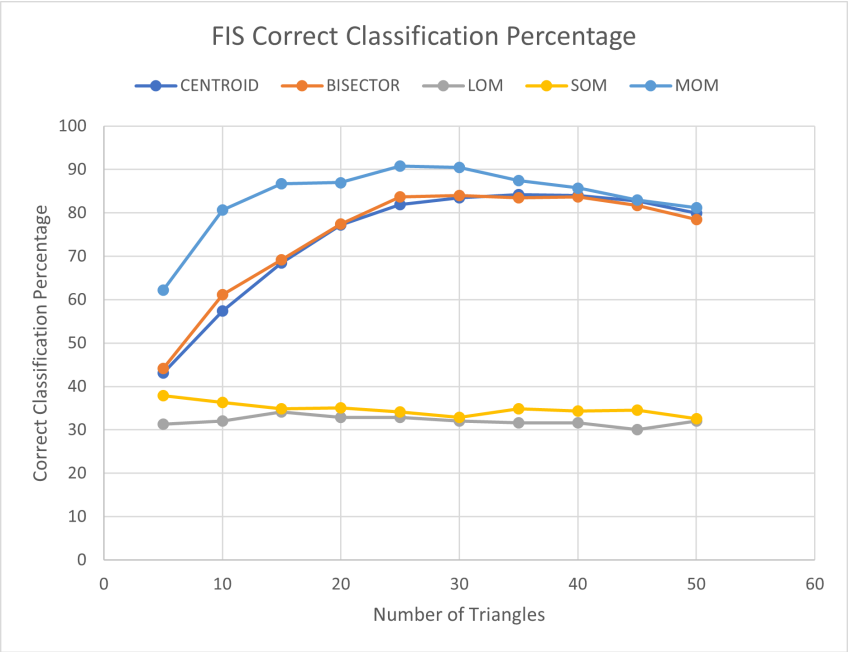
Finally, a **weight** is associated to each rule. It is computed as the product of the all membership degrees of the features used for the generation of the rule.

3.1.2 Search for the Best System

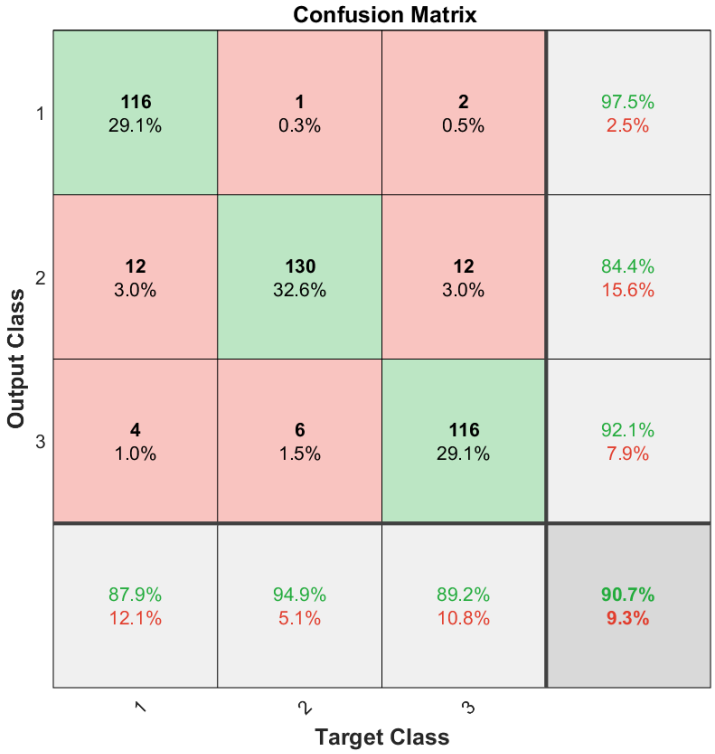
In order to find the best Fuzzy Inference System, the parameters that have been tuned are the **number of triangles** that compose the inputs membership functions and the **defuzzification method**. So, a finder script has been implemented to test all the possible combinations between these ranges:

- Number of triangles: [5 50] with a step of 5
- Defuzzification method: [*centroid*, *bisector*, *lom*, *som*, *mom*]

The results obtained are the following:



So, the best results are given by a number of triangles equal to *25* and the *mom* defuzzification method. With these parameters, the classification performs as follows:



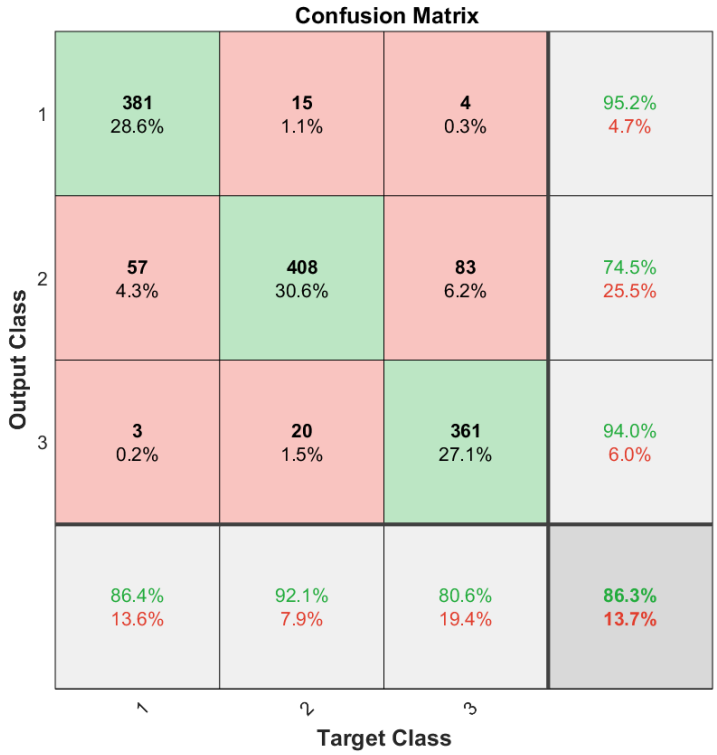
3.2 Adaptive Network-based Fuzzy Inference System

The second type of system is the **Adaptive Network-based Fuzzy Inference System** that is implemented by the *anfis* MatLab class. In this case, the system performs a sort of training to tune the main parameters. Instead, the options used in the *anfis* creation are the following:

- Number of membership functions equal to *3*
- Input membership functions type equal to the *generalized bell-shaped*
- Output membership function type equal to *linear*
- Maximum number of epochs equal to *25*

The number of membership functions is *3* because if it's greater the training fails. Regarding to number of epochs, going beyond *25* does not improve the percentage of correctness.

The resulting ANFIS performs the classification with the following results:



4 Fitting using Convolutional Neural Network

The aim of this part is to improve the performance of the Multi-layer Perceptron discussed in Section 2.2.1, but using a **Convolutional Neural Network**. In particular, it is necessary to try to improve the worst metric, in this case the mean.

4.1 Data Preprocessing

Convolutional Neural Networks are deep neural networks, that can take **raw signals** as input and not features. So, a new dataset generation is required.

The new dataset is an heterogeneous array, that corresponds to the *cells array* Matlab class. Each cell contains a **window**, which is a subset of a file in the original dataset. **No overlapping** is applied between windows. Concerning the vector of outputs, each element corresponds to the **mean** of a specific window. The chosen window size is *5000*, because lower values led to worse performance, while higher values caused out-of-memory errors.

4.2 Network Architecture

Convolutional Neural Networks can come in different forms, depending on the arrangement of the various **layers** (i.e. *convolutional*, *batch normalization*, *pooling*, etc.) that is chosen. Each layer also has different parameters such as *size*, *padding*, *stride*, etc., which determine its output and behavior.

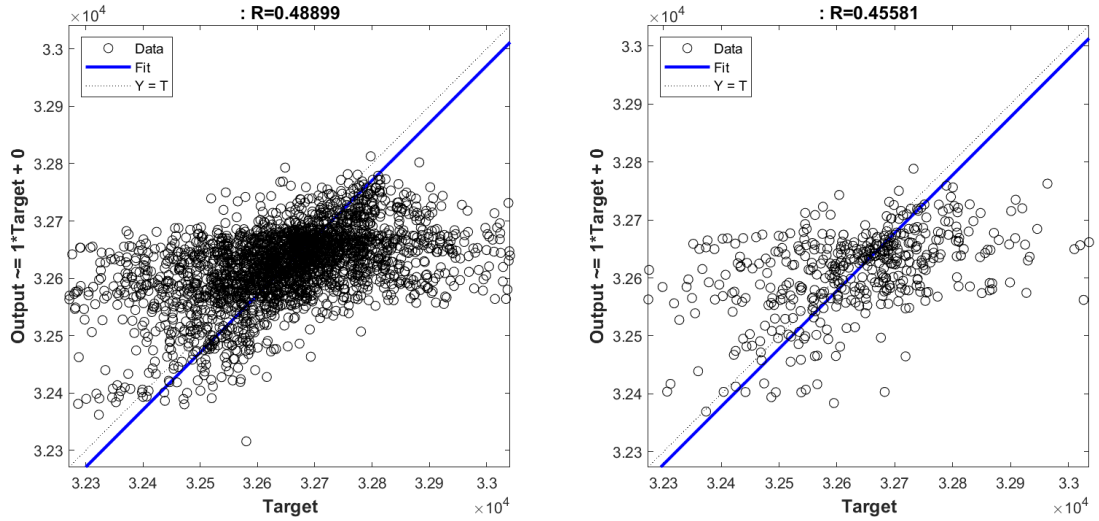
The best architecture found to solve the problem uses the following layers:

```
sequenceInputLayer(12, "Normalization", "rescale-zero-one")

convolution1dLayer(11, 96, 'Stride', 4, 'Padding', 'same')
batchNormalizationLayer()
maxPooling1dLayer(2, 'Stride', 2, 'Padding', 'same')
reluLayer()

convolution1dLayer(5, 256, 'Stride', 4, 'Padding', 'same')
batchNormalizationLayer()
maxPooling1dLayer(2, 'Stride', 2, 'Padding', 'same')
reluLayer()

convolution1dLayer(3, 348, 'Stride', 4, 'Padding', 'same')
batchNormalizationLayer()
reluLayer()

convolution1dLayer(3, 348, 'Stride', 4, 'Padding', 'same')
batchNormalizationLayer()
reluLayer()

convolution1dLayer(3, 256, 'Stride', 4, 'Padding', 'same')
batchNormalizationLayer()
maxPooling1dLayer(2, 'Stride',2, 'Padding', 'same')
```

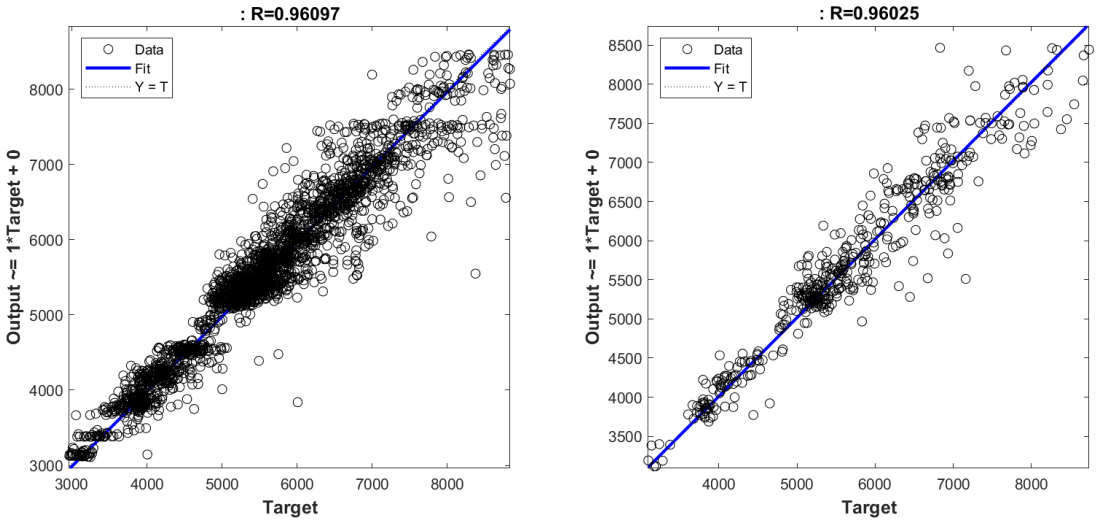
```
reluLayer()

globalAveragePooling1dLayer()
fullyConnectedLayer(256)
fullyConnectedLayer(1)

regressionLayer()
```

The network is trained using *80* epochs, *64* as mini-batch size and *adam* as solver.

Network structure is inspired by *CaffeNet*. The number of layers has been chosen to achieve a good trade-off between network **complexity** and **root mean squared error**. In addition, a slight improvement is achieved by using *rescale-zero-one* as the normalization method. The results obtained are the following (on the left the training regression, on the right the test regression):



Even the best network found does not yield acceptable results. For testing, the network has been tried using the **standard deviation** as a target, and the results are:



5 Forecasting using Recurrent Neural Networks

The aim of this part is to use **Recurrent Neural Networks** to forecast one or more values of a person's ECG.

5.1 Single-step ECG Forecasting

This network has to predict the **next ECG value**. To do that, a new training dataset is required. Specifically, each row is constructed by arranging the **11 signals** sampled at a given instant and the corresponding **ECG**, retrieved from the original dataset. Whereas, in the vector of targets, each row from the new dataset is associated with the ECG of the **next instant**. As described in Section 4.1, even in this case a *cells array* is created, where each cell contains samples related to a **window**.

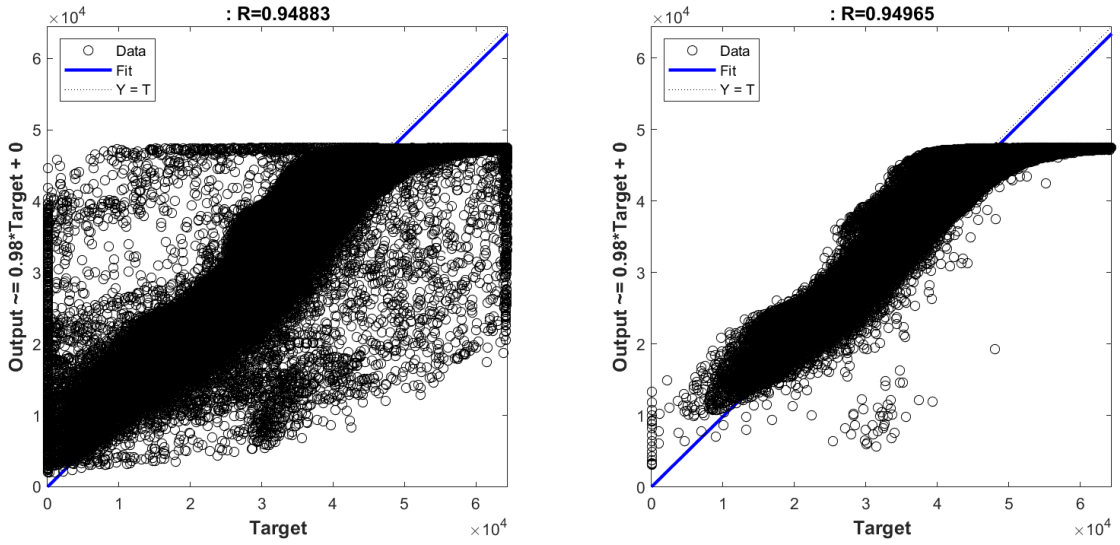
5.1.1 Network Architecture

As for the Convolutional Neural Networks, even the Recurrent Neural Networks are defined in a **layered** fashion. Usually, they consist of a *Long-Short Term Memory layer* and one or more *fully connected layers*. For each of them is possible to define the *size*.

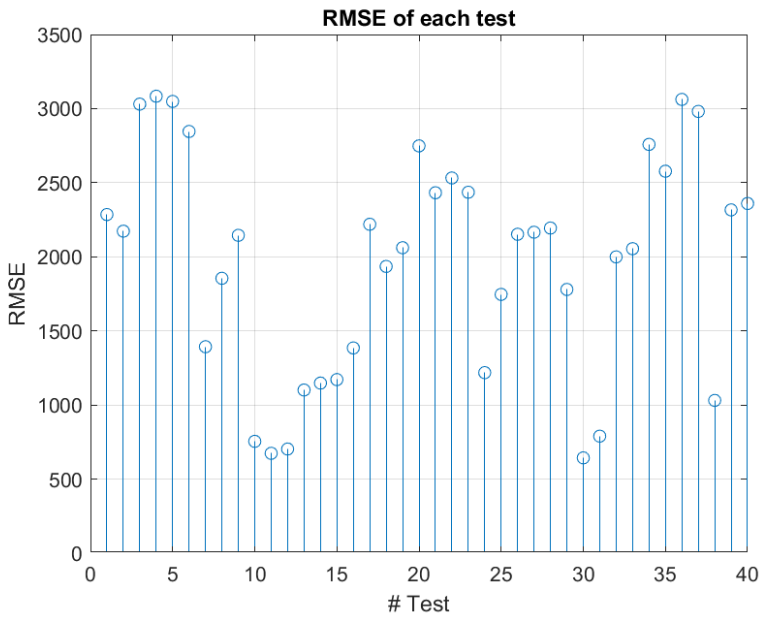
The best results are given by the following network:

```
sequenceInputLayer(12)
lstmLayer(200)
fullyConnectedLayer(256)
dropoutLayer(0.4)
fullyConnectedLayer(1)
regressionLayer
```

The network has been trained using *30* epochs, *4* as mini-batch size and *adam* as solver. The results obtained are the following (on the left the training regression, on the right the test regression):



In addition to the **r-value**, the **root mean squared error** has been calculated for each cell in the test set. The results are the following:



5.2 Multi-step ECG Forecasting

This network has to predict a set of **consecutive future values** of a person's **ECG**. In this case, the best results have been obtained using only the ECG value as input. So, a row in the dataset that contains the ECG sampled at the instant *k*, is associated with the ECG sampled at instant *k+1* in the target vector. Even in this case, the dataset is organized into a *cells array*, where each cell contains ECG related to a **window**.

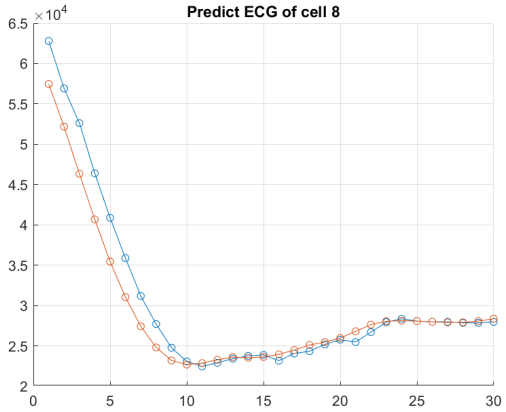
To get the wanted behaviour, the network works in a **closed loop** fashion. In particular, its state is initialized using a set of consecutive ECG values sampled from the person. Then, the ECG predicted is used as new input to forecast the next one. In this way is possible to predict more than one value.

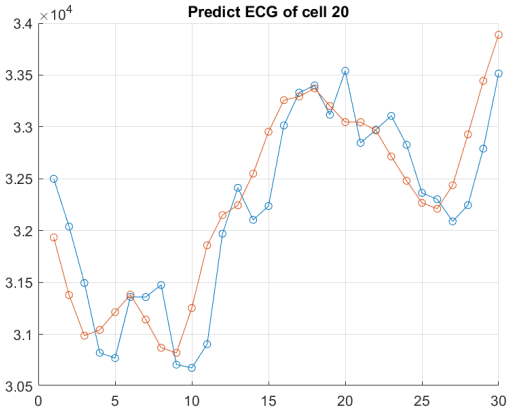
5.2.1 Network Architecture

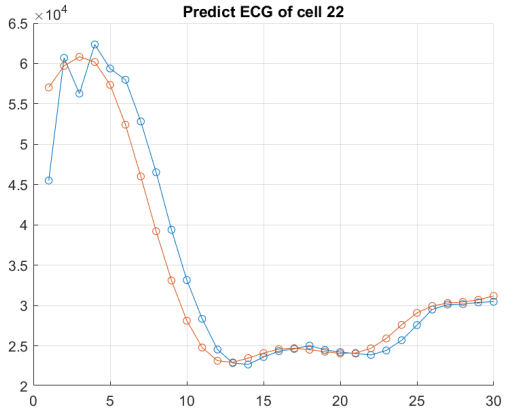
The best results are given by the following network:

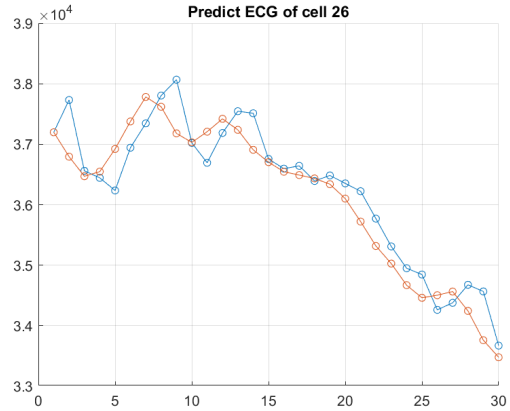
```
sequenceInputLayer(1)
lstmLayer(100)
fullyConnectedLayer(200)
fullyConnectedLayer(1)
regressionLayer
```

The network has been trained using *30* epochs, *32* as mini-batch size and *rmsprop* as solver. In order to verify the correctness of the network, the first step has been to test his ability to guess the *k+1 th* ECG value given a sequence of *k* ECG, using an **open loop** architecture. The following are some predicted cells:

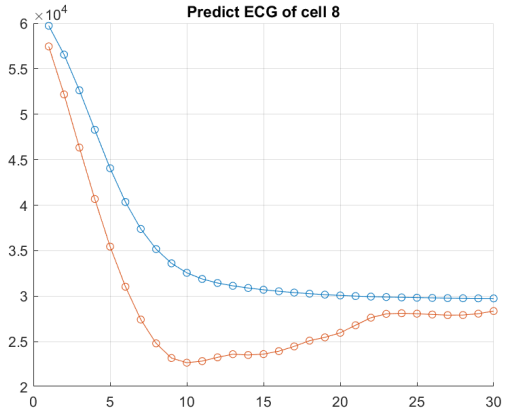


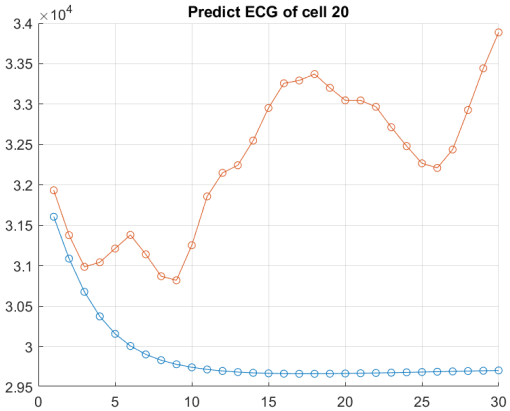


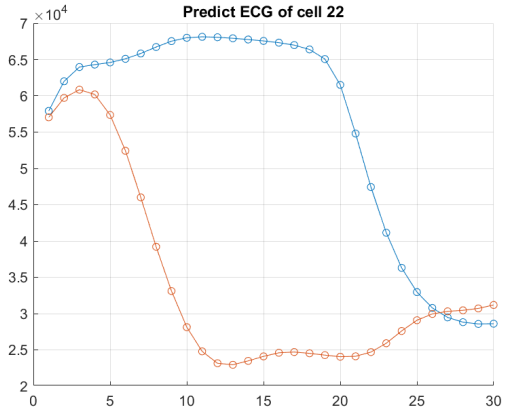


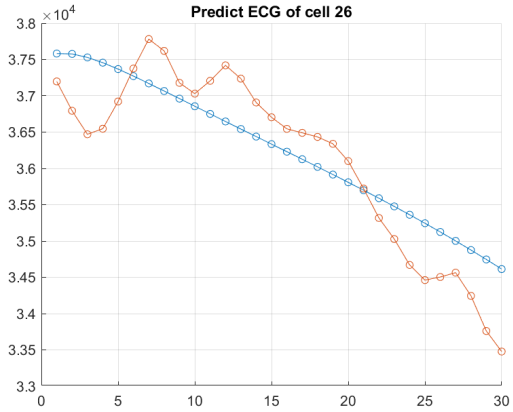


The blue plot is the **true ECG trend** and the orange is the **predicted one**. The network is visibly able to predict the *k+1 th* ECG value in open loop. So, the network has been tested to predict *30* consecutive values in closed loop. The predictions of the same cells previously analyzed are presented below:









As before, the blue plot is the **true ECG trend** and the orange is the **predicted one**. If the trend of the actual ECG does not show much variation as for cell *8*, the predicted ECG approximates the true one fairly well. On the other hand, if the real ECG has many variations in some cases the predicted ECG follows the trend but not the peaks (as for cell *26*), in other cases it diverges and then recovers later (as for cell *22*), or in other cases the network completely fails the predictions after the first three (as for cell *20*). However, the overall **mean root mean squared error** computed over all cells is *3306.93*, that is a good result.